Runtime Variation in Big Data Analytics

YIWEN ZHU, Microsoft, USA
RATHIJIT SEN, Microsoft, USA
ROBERT HORTON, Microsoft, USA
JOHN MARK AGOSTA, Microsoft, USA

The dynamic nature of resource allocation and runtime conditions on Cloud can result in high variability in a job's runtime across multiple iterations, leading to a poor experience. Identifying the sources of such variation and being able to predict and adjust for them is crucial to cloud service providers to design reliable data processing pipelines, provision and allocate resources, adjust pricing services, meet SLOs and debug performance hazards.

In this paper, we analyze the runtime variation of millions of production SCOPE jobs on Cosmos, an exabyte-scale internal analytics platform at Microsoft. We propose an innovative 2-step approach to predict job runtime distribution by characterizing typical distribution shapes combined with a classification model with an average accuracy of >96%, out-performing traditional regression models and better capturing long tails. We examine factors such as job plan characteristics and inputs, resource allocation, physical cluster heterogeneity and utilization, and scheduling policies.

To the best of our knowledge, this is the first study on predicting categories of runtime distributions for enterprise analytics workloads at scale. Furthermore, we examine how our methods can be used to analyze what-if scenarios, focusing on the impact of resource allocation, scheduling, and physical cluster provisioning decisions on a job's runtime consistency and predictability.

CCS Concepts: • **Computer systems organization → Cloud computing**; • **Computing methodologies → Causal reasoning and diagnostics**; • **Information systems** → *Data analytics.*

Additional Key Words and Phrases: big data, variation, predictions, interpretability, clustering

ACM Reference Format:
Yiwen Zhu, Rathijit Sen, Robert Horton, and John Mark Agosta. 2023. Runtime Variation in Big Data Analytics. *Proc. ACM Manag. Data* 1, 1, Article 67 (May 2023), 20 pages. https://doi.org/10.1145/3588921

1 INTRODUCTION

Big Data platforms have become ubiquitous over the last decade, enabling scalable data processing with high efficiency, security, and usability [3, 11, 17, 52, 55, 68, 69, 74]. However, the dynamic nature of resource provisioning, scheduling, and co-location with other jobs can cause occasional job slowdowns. Additionally, intrinsic properties of the job such as parameter values and input data sizes can change across repeated runs leading to variations in runtime. Figure 1 shows a set of recurring jobs in Cosmos [49], a Big Data analytics platform at Microsoft, submitted with different frequencies. We can see that some jobs have more stable runtime while some have occasional slow downs with non-regular patterns. But, it is not apparent *why such variations are happening*, *how*

Authors' addresses: Yiwen Zhu, Microsoft, USA, yiwzh@microsoft.com; Rathijit Sen, Microsoft, USA, rathijit.sen@microsoft.com; Robert Horton, Microsoft, USA, rhorton@microsoft.com; John Mark Agosta, Microsoft, USA, john-mark.agosta@microsoft.com.

2836-6573/2023/5-ART67 $15.00
https://doi.org/10.1145/3588921

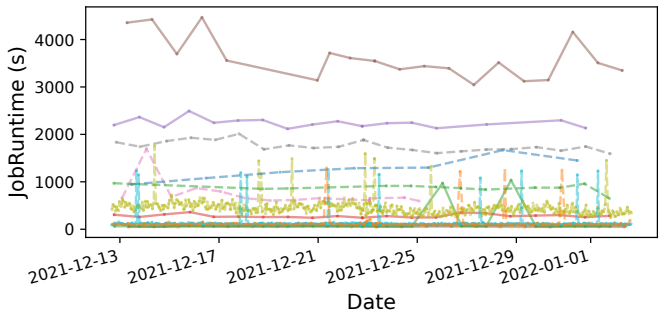


Fig. 1. Recurring jobs with runtime variation.

they can be mitigated, or *how likely it is for the next job run to be an outlier* compared to historic runs.

In production systems, jobs are often scheduled or pipelined with strong data dependencies (jobs using other jobs' output data as inputs) [13]. Stability and predictability of job runtimes are important factors that affect the fundamental design and architecture of data processing pipelines. Unfortunately, they are often neglected by operators due to the difficulties of assessment even though job slowdowns are inevitable [65]. Even with massive amounts of telemetry data, cloud providers still default to a manual triage process due to the difficulty of capturing the compounding factors that impact job runtime and its stability, which is not scalable and error-prone.

Although prior works [20, 56, 85] have empirically characterized runtime variation, they do not propose methods to predict the variation nor the likelihood of a new run being an outlier compared to the average or median runtimes. Other works such as Griffon [65] used machine learning models to predict the minor slowdown in runtimes for a limited number of job templates. They are unable to predict significant slowdowns that appear as outliers. As ML models are notoriously bad at handling outliers especially with a low existence, prior time-series based approaches [43, 67] are not applicable. In this paper, we aim to address this gap for production data analytics systems by developing a novel and systematic approach for modeling, predicting, and explaining the job runtime variation, allowing for finer-grained differentiation in characteristics.

For our study, we comprehensively examine the runtime variation for millions of production SCOPE [11] jobs on Cosmos [49], an exabyte-scale analytics platform at Microsoft that supports a broad spectrum of Microsoft products [49]. Our key contribution is a framework for systematically analyzing, predicting and explaining runtime variation that includes:

(1) *Descriptive* analysis: by examining historic data including intrinsic job properties, resource allocation, and physical cluster conditions, we provide a better understanding of the factors affecting runtime variation for each individual job. In particular:
   (a) We show that popular scalar metrics, such as Coefficient of Variation (COV) considered by prior work [56], are not sufficient to characterize variation with the existence of outliers. Instead, we propose a novel scheme of characterizing variation using properties of the distribution of normalized runtime of the jobs that provides fine-grained information such as the probability of outliers, quantiles, and shapes of the distribution. [Section 4]
   (b) We make novel use of Shapley values [66] for explaining predictions for variation and quantitatively analyze the contributions of different features. [Section 6]
(2) *Predictive* analysis: we develop an innovative approach based on likelihood to identify distinctively-diverse runtime distributions, and predict the distribution with >96% accuracy, out-performing previous methods [65]. [Section 5]

(3) *Prescriptive* analysis: based on the predictor, we quantitatively analyze what-if scenarios and identify potential opportunities to reduce variation by limiting spare tokens, scheduling on newer generations of machines, and better load balancing. [Section 7]

The rest of this paper is organized as follows. Section 2 discusses challenges in estimating and predicting runtime variation and our goals and approach in this work. Section 3 gives a brief overview of SCOPE jobs in Cosmos, potential sources of variation, and the datasets that we study. Sections 4–7 present the descriptive, predictive and prescriptive analyses as outlined above. Section 8 discusses related work in this space and Section 9 concludes the paper.

2 GOALS, CHALLENGES, AND APPROACH

Reasoning about performance changes is often done manually by experienced engineers with strong assumptions that can potentially lead to biased results. More recently, the availability of massive telemetry data in the cloud, that includes both information about job characteristics as well as status of the physical clusters, and the advent of data analytic methods, raise expectations that this process can be improved with more systematic and rigorous approaches.

Our goal is to evaluate and predict runtime variation at the *individual* job level with a customized and use-case specific measurement that is more insightful for customers for both monitoring and planning purposes. And this is a highly desired metric from the customer's point of view, as validated by several conversations with the program managers (PMs). We also want to provide rich information regarding variation, such as the probability that a job runtime may exceed an extreme value, or various quantitative properties of the runtime distributions, e.g., quantiles, outliers, to the user, which was difficult to capture using traditional ML methods.

Moreover, performance modeling of computational jobs in distributed systems is difficult, especially when focusing on reliability, due to the following challenges.

- **Complex environmental factors (C1)**: Resource sharing in cloud computing platforms adds complexity to the modeling of the job runtime due to noisy neighbors and other environmental changing factors. It is untractable for manual approaches to relay the dynamic condition of each computation node and unravel the potential issues that result in performance degradation.
- **Existence of rare events (C2)**: For rare events (such as occasional service disruption) that result in outliers and longer tails of the runtime distributions, it is difficult to collect sufficient observations of outliers for a recurring job in order to accurately estimate their distributions. It is therefore crucial to be able to leverage the learning from job instances in other job groups that have sufficient observation samples.
- **Lack of proper metrics (C3)**: How to measure variation remains a challenge in the case of the characteristic long-tailed distributions of runtime, for which conventional variance-based measures do not capture the extreme values of interest. Metrics such as COV that are commonly used to evaluate the runtime variation are not sufficient to capture detailed characteristics of various runtime distributions.
- **Lack of labeled data (C4)**: While the majority of machine learning approaches for predictive analysis require labeled data, there is no label recorded for the causes of runtime distributions or job slowdowns. Manually evaluating runtime reliability to determine the distribution *category* each job belongs to is also infeasible for new jobs with a small number of occurrences.

Our 2-step approach in this work is to characterize and then predict variation based on the distribution of normalized runtimes of recurring jobs (see Figure 2). Leveraging information both at the job level and the machine level, we:

- **Characterize runtime distributions**: Our *Clustering Analysis* uses a novel scheme of featurization to cluster [5] historic jobs with distinctively-diverse runtime distributions. We associate

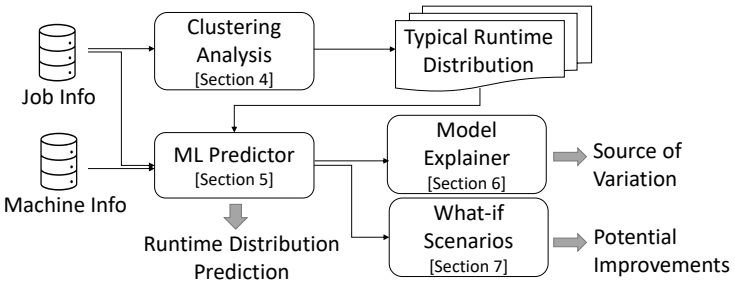


Fig. 2. Framework.

the job with one distribution it belongs to using an innovative and adaptive *posterior likelihood* method. For each type (single-mode, multi-mode) of distribution, we define key metrics to depict the distribution, and quantify the variation in numeric terms that can be easily understood by users [challenge **C3**, Section 4].

- **Develop *ML predictor***: We leverage Machine Learning (ML) classification techniques to predict which distribution of runtimes the job most likely belongs to, taking into account job properties, resource allocation, and environmental conditions such as system load that potentially leads to noisy neighbors [**C1**]. To overcome the challenges **C2** and **C4**, we develop the model using the observations of distributions over a long time interval and for jobs with more recurrences (Dataset D1, Table 1), while the model can be applied to any new jobs. [Section 5]
- **Explain predictions**: We use *Model Explainer* based on feature contribution algorithms to better understand the various factors associated with runtime variation [challenge **C1**, Section 6].
- **Analyze *what-if scenarios***: Based on the prediction model, we propose hypothetical scenarios and evaluate the potential improvement of runtime performance quantitatively [Section 7].

While point prediction for the job runtime is an important and challenging problem on its own [19, 28, 29, 46–48, 51, 82], we want to predict the potential variation in runtimes for recurring jobs, rather than the absolute runtimes. Thus, direct prediction of job runtimes is a *non-goal* for this work.

3 PLATFORM AND DATASETS

Cosmos [49] is an exabyte-scale big data platform developed at Microsoft since 2002, with more than 300k machines across multiple data centers worldwide [83]. Using a YARN-based [73] resource manager, the system processes >600k jobs per day from tens of thousands of Microsoft internal users. It is a big internal shared-cluster where efficiency is paramount. Over the past decades, a multitude of research projects and engineering efforts have improved its efficiency, security, scalability and reliability [6–9, 11, 14, 16, 21, 32–34, 50, 52, 64, 76, 80–83, 83].

Cosmos jobs can be authored using a SQL-like dialect, named SCOPE [6] with heavy use of C# and user-defined functions (UDFs). Upon submission, a job is compiled to an optimized execution plan as a DAG of operators, and distributed across different machines. Each job consists of multiple *Vertices*, i.e., an individual process that will be executed on a container assigned to one physical machine.

3.1 Job Groups

Our work focuses on understanding and predicting variation in runtimes over repeated runs of jobs that we assemble into *job groups*. Variation is meaningful only when jobs recur (i.e., sample size > 1). Prior studies [14, 33, 82] have shown that 40–60% of jobs on Cosmos are recurring jobs.

Fig. 3. Token usage for an example job during its run.

Others [20] have also reported a significant fraction of jobs as recurring on their systems. We identify recurrences by matching on a key that combines the following.

- The normalized job name, which has specific information like submission time and input dataset removed [32, 82].
- The job signature [32], which is a hash value computed recursively over the DAG of operators in the compiled plan. The signature does not include job input parameters.

We thus have *job groups* with *job instances* belonging to each group, corresponding to recurrences of the job. Job instances have the same key value within each job group.

3.2 Sources of Variation

Within each job group, runtimes of job instances can vary due to any of the following reasons:

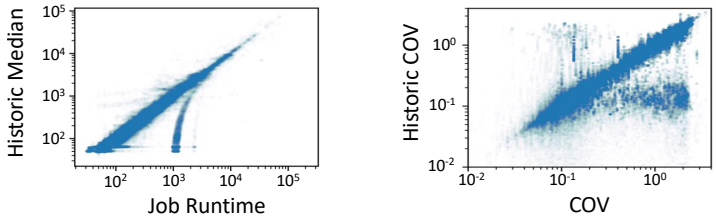
Intrinsic characteristics. The key used for grouping jobs includes information on the execution plan (e.g., type of operators, estimated cardinality, dependency between operators) while not including the job input parameters (e.g., parameters for filter predicates) or input datasets. Different instances can have different values for these parameters, datasets, and their sizes. This can lead to different runtimes within the group if the parameter changes are not accompanied by a change in the compiled plan. In our datasets, we have observed that input data sizes can vary by up to a factor of 50 within the same job group.

Resource allocation. In Cosmos, the unit of resource allocation is a *token* [49], which is analogous to the notion of a container. The number of tokens guaranteed for a job can be specified by users at the time of job submission or it may be recommended by the system [63]. To further improve the utilization of the existing infrastructure, unused resources are repurposed as preemptive spare tokens [7] that can be leveraged by any jobs freely[1]. The availability of these spare tokens is difficult to predict, and can have meaningful effects on runtimes. Figure 3 shows the skyline of token usage for a Cosmos job that was allocated with 66 tokens (dashed line). Including spare tokens, the job consumed up to 198 tokens in total throughout its processing time.

The maximum number of tokens used by a job depends on how much parallelism it can exploit subject to the number of tokens allocated. While observing the execution of various workloads on Cosmos, we have seen maximum token counts vary by a factor of 10 within the same job group. There is also variation in the characteristics of allocated resources. Tokens map to computational resources on compute nodes with different Stock Keeping Units (SKUs). Having evolved for over a decade, the Cosmos cluster consists of 10–20 different SKUs with different processing speeds [83]. In our datasets, we have observed different job instances within the same job group run on one to nine different SKUs simultaneously.

Physical cluster environment. Finally, physical cluster environment also leads to variations in runtimes. This includes both the availability of spare tokens (discussed above) and the load on

[1]The usage of spare tokens is capped by the allocation as specified by users.



(a) Median vs instance runtimes (b) Historic COV vs COV

Fig. 4. Correlations between historic median and job runtimes (a), and historic COV and COV of all observations (b).

the individual machines. Higher utilization (load) is likely to cause more contention for shared resources, and a larger range of loads may increase runtime variation.

3.3 Datasets

To develop insights into the sources of variation as discussed above, we collected data by: i) extracting information about intrinsic characteristics such as operator counts in the plan, input data sizes, and cardinalities, costs, etc., estimated by the SCOPE optimizer using the Peregrine framework [32]; ii) obtaining token usage information from the job execution logs, and SKU and machine load information using the KEA framework [83]; and iii) joining all this information together by matching on the job ID, name of the machine that executes each vertex, and the corresponding vertex start/end time.

Table 1. Datasets used for this study.

Dataset	Interval	Job Groups	Job Instances	Support
D1	6 months	>9K	>3M	20
D2	15 days	>11K	>700K	3
D3	5 days	>11K	>200K	3

Table 1 summarizes the datasets that we use for our study. The datasets consist of a subset of jobs run over the corresponding interval, and are included if the number of instances per group (support) exceeds a minimum threshold. With a support of minimum 3 occurrences, 53% of jobs are included. In this research, we only focus on batch jobs as opposed to streaming jobs or interactive jobs that Cosmos also supports. We use dataset D1 to identify and group distributions of runtimes (see Section 4.2) for jobs with a large number of occurrences (>20). We used D2 to train a predictor for runtime variation and D3 to test its accuracy (Section 5).

4 CHARACTERIZING RUNTIME VARIATION

We now discuss how we characterize and quantify runtime variation for recurring jobs. This will form the basis for our prediction strategy that we discuss in Section 5.

4.1 Scalar Metrics

As outlined previously (Section 1), a job's average runtime does not give much insight into variations across repeated runs, or how long the next run of the job will take. Thus, it is not very useful for characterizing, predicting, or explaining the variations.

Next, we investigate how well a job's median runtime correlates with runtimes over the different repetitions of the job. Figure 4a shows how runtimes for individual repetitions of the job compare with its historic median using dataset D2 in log-scale.

We observe two distinct patterns in Figure 4a—a set of points are clustered along the diagonal, indicating a good correlation of individual job instances' runtimes to the median, and another set of points are clustered separately in a pattern resembling a stalagmite[2]. The runs corresponding to the points in the stalagmite are much slower than the median runtime and contribute to the (long) tail of the runtime distributions. Such runs are rare (comprising less than 5% of all runs), with the probability reducing with larger median values. But we have found it to be very difficult to predict upfront if the time for a new run will end up on the diagonal or on the stalagmite. The existence of these two patterns, as well as the difficulty of predicting to which pattern a new run will belong, even if the median runtime is stable and known, makes the median a poor choice to use for predicting runtimes or for characterizing variations. A similar trend can be found for the average and 95th percentile of historic runtime.

The Coefficient of Variation (COV) is another commonly-used metric to characterize variation. It is defined as the (unitless) ratio of standard deviation to the average. COV is straightforward to compute and interpret, and prior work [56] has used COV to characterize variation in job runtimes. But COV has several limitations:

- **Bias**: The runtimes of SCOPE jobs that we study range from seconds to days, with significant differences in the average values. This may cause COV to be biased and one could always observe very large COV for short-running jobs.
- **Instability**: The average runtime can increase due to the existence of outliers (in such large distributed systems, some jobs inevitably run slow occasionally). Thus, COV can be unstable with addition of more jobs in the dataset. Unlike the average, COV does not converge with a large sample size thus does not have a consistent estimator [22].
- **Coarse-grained**: COV does not capture many characteristics of a distribution, such as its shape (such as unimodal, bimodal, and the existence of outliers). Hence it cannot readily explain variation in a fine-grained manner.

Figure 4b shows how well the COV computed from historic runs (y-axis) for each job instance based on dataset D2 compares with the COV of times from all runs (x-axis) based on the observation in D3. Similar to the discussion for medians above, we see multiple groups of points, with the same historic COV appearing for different COV values from new runs and it is difficult to predict for a new run to which group it will belong. Additionally, the COV metric suffers from the limitations mentioned above.

Overall, we found that scalar metrics such as average, median, quantiles, and COV by themselves are not sufficient for understanding or predicting runtime variation.

4.2 Distributions

We represent runtime variation for each recurring job group by its *runtime distribution*. Although there is a large variety in the runtimes of SCOPE jobs, we found that runtimes of many different jobs have similar probability distributions. We refer to these as *shapes*. Knowing a job's distribution is sufficient to determine any characteristic of its variation, including the risk that its runtime will exceed a specified threshold.

To compute the shapes, we first normalize the job runtimes, then compute their empirical Probability Mass Functions (PMFs, i.e., histogram). Jobs are *clustered* based on the similarity of their runtime distributions, and for any new job, we can predict the cluster it belongs to. We identify the job's PMF as that of the cluster it belongs to. This methodology allows us to generalize our analysis across different jobs and work instead with a small number of clusters that can be easily understood by the users.

[2]A stalagmite is a rocky formation that arises from the floor of a cave and may reach the ceiling [1].

We used the following two normalization strategies to transform job runtimes, using medians computed on "historic" data from Dataset D1 as in Table 1:

Definition 4.1. The **Ratio-normalization** *is defined by the ratio of job runtime to its historic median, i.e., job runtime / median runtime. And* **Delta-normalization** *is defined by the difference, job runtime - median runtime.*

The Ratio-normalization distribution measures relative change in runtimes, while the Delta-normalization distribution measures the absolute deviation from median, measured in seconds. Note that runtime with various ranges can be normalized more effectively using ratio-normalization. E.g., absolute variations for long-running jobs are typically higher (1h±10min), whereas those for short-running jobs can be a lot lower. In this regard, ratio-normalization improves consistency and lumps together comparable distributions with different runtime ranges. On the other hand, for very short or very long jobs, it might be less insightful to measure variances in percentage. For short-running jobs, the percentage can be very large (e.g., 5s±300%). For long jobs, the percentage can be very small, leading to a very "thin" distribution measured by the ratio-normalization. Therefore, in this work, we leverage the delta-normalization combined with ratio-normalization to capture the variation in absolute terms.

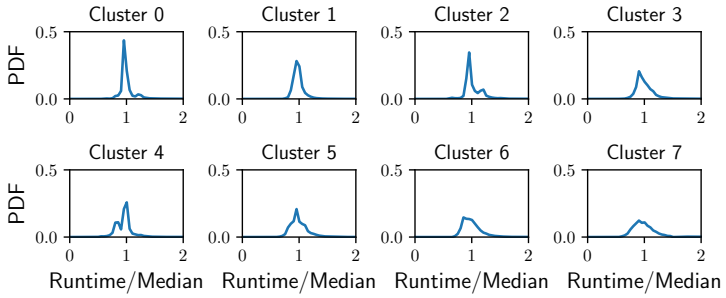
Our clustering analysis to recover the "typical" distribution shape across jobs uses dataset D1 where only jobs with >20 occurrences are included for more accurate estimation for their runtime distribution. For each job group, we derive its histogram for the distribution of normalized runtimes and then use an unsupervised machine learning algorithm to cluster them. Note that the inputs to the clustering analysis are the PMF probabilities of each bin of the histogram as opposed to the job features (e.g., input size, etc.).

Our principal design choices for the runtime distribution clustering method are as follows:

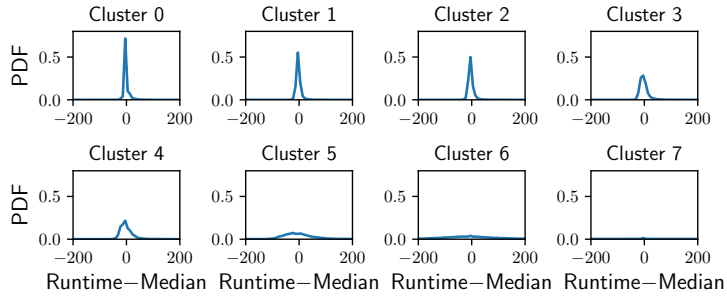
- **Bin size and Range**: The range should cover the majority of values with relatively fine granularity but not too small to capture fluctuation due to noise. We merge the outliers into one bin (based on being $\leq$ or $\geq$ some thresholds)[3]. We evaluated 50, 100, 200 and 500 bins, and chose 200 bins that has relatively smooth PMF curves, and different shapes of distributions are observable.
- **Clustering algorithm**: Hierarchy clustering based on dendrogram [44] and Agglomerative clustering [54] take different distance metrics, linkage methods, and user-specified number of clusters. However, they result in imbalanced clusters (some with >90% of the data in one cluster). K-means clustering [62] resulted in more balanced clusters, so is chosen for the following analysis.
- **Number of clusters**: It is determined based on: (i) numeric analysis of *inertia*, defined by the sum of squared distances between each sample and its cluster centroid (we pick an elbow point where adding more clusters does not significantly decrease the inertia), and (ii) by visually examining the clustering results to check if the clusters are sufficiently different from each other and have unique characteristics.
- **Smoothing histograms**: The standard clustering algorithms are based on using PMF probabilities as input vectors assuming each dimension is independent. In reality, adjacent density values of bins (e.g., the probability of a runtime being in the 4^{th} or 5^{th} bin) are correlated. However, with any distance measurement (e.g., dot product), correlation between adjacent bins is not considered. Therefore, we introduce a *smoothing step* after deriving the PMFs to jointly consider any adjacent bins' values such that the two smoothed vectors mentioned above will have a higher affinity.

[3]For Delta-normalization, we use [-900, 900] (where 1% of jobs are 1066s slower than median, we round down to 900s, i.e., 15 min), and for Ratio-normalization, we use [0, 10] (where 1% of jobs are 10.6x slower than median, we round down to 10x). Jobs >900s or 10x slower than median are defined as outliers.

Figure 5 shows the distributions for the 8 clusters using Ratio-normalization and Delta-normalization policies. We see that some distributions have two modes (e.g., Cluster 0, 2, 4 using Ratio-normalization) and with different variances. Table 2 summarizes important statistics for each cluster. For example, Cluster 0 with Ratio-normalization has a outlier probability of 1.63% (defined by ≥ 10x slower than median for Ratio-normalization); the difference between 25 and 75th percentile is 0.06; the 95th percentile of this distribution is 1.41, and the standard deviation is 2.46. The outlier probability decreases to 0.06% for Cluster 7 with Ratio-normalization. Clusters are ranked according to the difference between the 25th and 75th percentiles.



(a) Ratio-normalization



(b) Delta-normalization

Fig. 5. Typical distributions of normalized runtime.

Table 2. Statistics for the clusters of runtime distributions.

	Ratio					Delta			
cid	outlier (%)	25−75th	95th	std	cid	outlier (%)	25 − 75th (s)	95th (s)	std (s)
0	1.63	0.06	1.41	**2.46**	0	1.93	4	28	155
1	0.42	0.11	1.2	0.93	1	0.49	11	19	140
2	**1.66**	0.16	1.37	2.18	2	0.53	11	23	148
3	0.25	0.17	1.29	1.45	3	0.55	16	33	140
4	1.46	0.17	1.35	1.94	4	0.98	31	63	153
5	0.25	0.19	1.34	0.82	5	0.73	69	128	179
6	0.26	0.20	1.37	0.97	6	2.43	199	408	296
7	0.06	**0.29**	**1.46**	0.55	7	**24.23**	**936**	**1359**	**2548**

5 PREDICTING RUNTIME VARIATION

We now develop a prediction model based on explainable ML to predict the *shape* of runtime distribution (as in Figure 5) of jobs. We use dataset D2 as training set and D3 as testing set.

5.1 Feature Selection

We consider three classes of predictive features that are available at the job compile time: those derived from the job execution plan ("intrinsic"), those representing statistics of the job's past resource use, and features describing the load in the physical cluster where the job will run. We describe the classes below.

Intrinsic characteristics. We leverage information on the job execution plan obtained from the query optimizer [32] at compile time as input, which can be indicative of the query type, data schema and its potential computation complexity. It includes the number of operators in each type (e.g., Extract, Filter), estimated cardinality, etc.. For a newly submitted job, its detailed input data size is unknown, and the estimated cardinality can be quite off [82]. Therefore, using historic job instances of the same job group, we extract statistics for the total data read, temp data read, as well as the statistics related to the execution plan as additional input features that can be informative for the size of the job.

We also derive the fraction of vertices running on each SKU as the input features, which indicates the resource consumption by each SKU. A previous study [83] shows that, in Cosmos, some newer SKUs might process data faster than the others; therefore, we believe that the fractions of vertices executed on different SKUs would impact the runtime distribution.

Resource allocation. The token allocation is a good indicator for the resources being utilized by a particular job thus impacting the runtime. However, [63] detects that users are often over-allocating (e.g., user selects to allocate 1000 tokens, but the peak actual usage is only 600). In this work, we integrated historic token utilization with token allocation as the input. For historic job instances of the same job group, we extract the resource utilization (min, max, and average token usage based on the skyline as in Figure 3) and use the historic statistics as features (historic average and standard deviation). We also created a new variable for spare tokens (historic average). The model learned to place less importance on token allocation as a feature compared to actual utilization, and we corroborate this from the Shapley scores in Section 5.2.

Physical cluster environment. The job runtime can be affected by the utilization of the machines that execute its vertices—a higher utilization level indicates a hotter machine that is likely to have more severe issues related to noisy neighbors and resource contention. Therefore, we extract the CPU utilization level of the corresponding machines in each SKU at the job submission time as the input. Compared to existing methods, such as [65] and [85], incorporating new-real-time machine status information improves the model accuracy (see Section 5).

Work is ongoing to record the per container usage for both CPU and memory that includes more targeted information on the particular job and captures more accurately the resource consumption compared to the machine-level counters [25]. Once available, features can easily be replaced or added to our models. We expect them to be strong indicators for the job runtime as they capture more job-level charateristics. They might also reveal if a job that is CPU intensive or memory intensive is more likely to have large runtime variances.

5.2 Cluster membership prediction

For a new job to be submitted, we want to *predict* its runtime distribution shape based on information that is available at compile time. This naturally maps to a classification problem, where the prediction

target is to map each job to a particular distribution shape (e.g., one among the 8 different shapes as in Figure 5).

Cluster membership based on posterior likelihood. For each job instance, to determine which distribution shape it has, we leverage the set of similar job instances in the analyzed period with the same job group (i.e., same job name and execution plan) to derive the group's empirical Probability Mass Function (PMF), i.e., the histogram of the runtime distribution. Based on even a small number of runtime observations, we derive the posterior likelihood of these observations to be drawn from any one of the pre-defined distribution shapes as in Figure 5.

Based on Bayes' Theorem [5], the *posterior log-likelihood* that a job group with N runtime observations, $x_{n=1\cdots N}$, belongs to a cluster $z_{i=1\cdots K}$ can be derived based on the PMF of these N observations, $\phi_{h=1\cdots H}$, and the PMFs of the $K = 8$ pre-defined clusters, $\theta^{i=1\cdots K}_{h=1\cdots H}$, which is adaptive to larger sample size:

$$p(x_1, x_2 \cdots x_N|z_i) = \prod_{n=1\cdots N} F(x_n|\theta^i_{h=1\cdots H}) \tag{1}$$

$$= \prod_{n=1\cdots N} \theta^i_{h(x_n)} \tag{2}$$

$$\log p(x_1, x_2 \cdots x_N|z_i) = \sum_{n=1\cdots N} \log(\theta^i_{h(x_n)}) \tag{3}$$

$$p(z_i|x_1, x_2 \cdots x_N) = \frac{p(x_1, x_2 \cdots x_N|z_i)p(z_i)}{\sum_{i=1\cdots K} p(x_1, x_2 \cdots x_N|z_i)p(z_i)} \tag{4}$$

$$= \frac{\prod_{n=1\cdots N} \theta^i_{h(x_n)}}{\sum_{i=1\cdots K} \prod_{n=1\cdots N} \theta^i_{h(x_n)}} \tag{5}$$

$$\sim \prod_{n=1\cdots N} \theta^i_{h(x_n)} \tag{6}$$

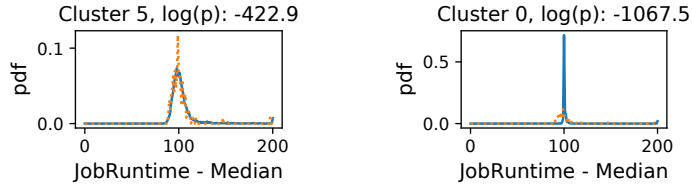
$$\log p(z_i|x_1, x_2 \cdots x_N) = \sum_{n=1\cdots N} \log\left(\theta^i_{h(x_n)}\right) - \text{constant} \tag{7}$$

$$= \sum_{h=1\cdots H} n_h \log\left(\theta^i_h\right) - \text{constant} \tag{8}$$

$$\sim \sum_{h=1\cdots H} \phi_h \log\left(\theta^i_h\right) \tag{9}$$

Where,

- H: number of discrete bins when we derive the PMF for each distribution, a constant across all distributions.
- $\theta^{i=1\cdots K}_{h=1\cdots H}$: parameter of normalized runtime distribution for cluster i, specifically, the PMF value for bin h.
- $\phi_{h=1\cdots H}$: parameter of distribution based on observations for a particular job group (i.e., $x_{n=1,2\cdots N}$), specifically, the probability for bin h of the PMF.
- $h(x_n)$: the bin index that observation x_n belongs to.
- n_h: number of observations of runtime (i.e., $x_{i=1\cdots N}$) for the job group that belongs to bin h.
- $x_{n=1\cdots N}$: runtime observation n, where $x_{n=1\cdots N}|z_{i=1\cdots K} \sim F(\theta^{i=1\cdots K}_{h=1\cdots H})$.
- $p(z_i)$: prior on the probability of each cluster, assuming to be a constant across all clusters (non-informative prior [5]).



(a) Most likely cluster (b) Least likely cluster

Fig. 6. Examples of likelihood values (higher values indicate more probable).

It is interesting to point out that the log-posterior-likelihood is proportional to the dot product of the PMF of observations for the particular job group, i.e. ϕ_h, and the one of the pre-defined 8 clusters (after taking the log), i.e., θ_h^i (see Equation 9).

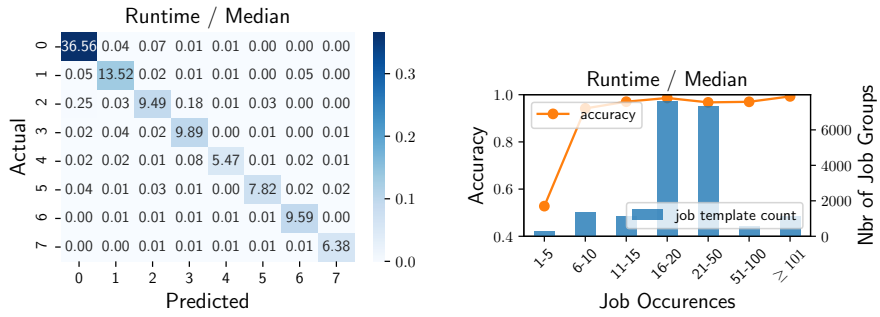
Figure 6 shows an example of normalized runtime distribution (by Delta-normalization) for a job with 10 occurrences compared with 2 clusters. The dashed line is the PMF for observations for this job group, i.e. ϕ_h, and the solid line is for the predefined clusters, θ_h^i. We see that Cluster 5, with the highest log-likelihood of -422.9, has the most similar shape (see Figure 6a), while Cluster 0 has the least similar shape (see Figure 6b). Each job instance together with its job group is then associated with a cluster label with the highest likelihood as the prediction target (label). This cluster association algorithm will always place a job in the most likely cluster. We observed that jobs with fewer observations may not be similar to any existing cluster's runtime distribution. Therefore, we focus only on job groups with sufficient samples. We employ the inertia curve to tune the number of clusters as proposed in Section 4.2 to avoid overfitting.

Classification model. Based on the inputs, we conduct (1) passive-aggressive feature selection [78] based on feature importance to avoid the use of correlated features, (2) parameter sweeping to select the best hyper-parameters for the classification algorithm, such as the number of trees for tree-based algorithms, and (3) fitting using RandomForestClassifier [61], LightGBMClassifier [41] and EnsembledClassifier [57] by combining a set of popular classification algorithms, such as RandomForestClassifier, LightGBMClassifier, GradientBoostingClassifier [59], GaussianNB [58], and XGBClassifier [60], using soft voting. Note that RandomForestClassifier and LightGBMClassifier are well-known to have high accuracy for ML tasks using tabular data, especially for out-of-sample tests. In this work, among the classifiers, LightGBMClassifier has the highest accuracy, thus we report its result for the rest of the paper. By analyzing the prediction results, we noticed:

By examining the Gini importance [38] of the input features, we found that features related to the computation complexity and input data sizes (such as count of vertices, and data read) are significant and the features related to the historic runtime observations are also significant. The token utilization (such as the max), and compile time information (such as cardinality estimates) are also important. The CPU utilization of machines also impacts the prediction, which coincides with our belief that the physical cluster environment will affect the runtime variation of jobs. We also noticed that many of the operators turned out to be less important. The total vertex count is less important than the data size (total data read or cardinality-related metrics). It is possibly due to the huge variation in data processed by each vertex. In Section 6, we dive into more details on the contribution of some features.

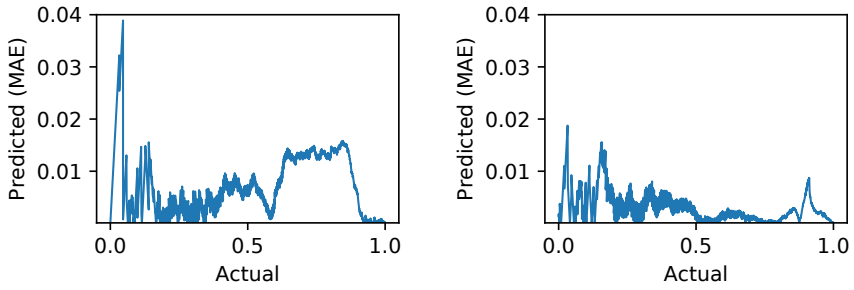
> **Insight:** The feature importance learned from the model is mostly consistent with our expectation.

Figure 7a shows the confusion matrix on test data comparing the predicted label (the x-axis) and the actual label (the y-axis) where each cell shows the portion of jobs of each category. Predictions



(a) Confusion matrix (b) Accuracy by occurrences

Fig. 7. Prediction accuracy for Ratio-normalization.



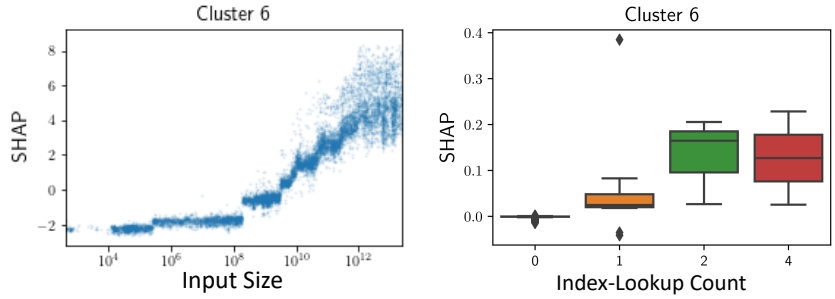
(a) Regression (b) Proposed approach

Fig. 8. Prediction accuracy for Delta-normalization compared with traditional regression model.

using both Ratio and Delta-normalization achieve overall accuracy of >96%. Figure 7b (orange line) shows the accuracy for jobs with different numbers of historic occurrences. We can see that for jobs with more historic occurrences, the prediction accuracy is higher, which indicates that the model prediction can be further refined by adding more observations from the same job group. The blue bar shows the count of job groups based on the number of job occurrences (1-5, 6-10, etc.). We can see that most of the jobs have 16-50 historic observations over the analyzed period. Similar trend can be seen for Delta-normalization.

Insight: Model predictions using both Ratio and Delta-normalization achieve high accuracy.

We extended the traditional random forest regression model as proposed in [65] by adding more query optimizer and near-real-time machine status information as features to predict the job runtime as the label. Figure 8 compares the predicted distribution for all job runtimes based on the proposed method (using classification model to predict the distribution shape) and the regression model against the actual job runtime distribution using Quantile-Quantile Plot [24], plotting the mean absolute error (MAE) in the y-axis. If two distributions are identical, the plotted quantile should align and the MAE=0. We can see that the proposed classification model (Figure 8b) has better accuracy compared with the traditional regression model (Figure 8a) especially for higher percentiles as it captures better the existence of outliers in the distributions of clusters of jobs (see Figure 5). The Kolmogorov–Smirnov distance [35] is also reduced by 9.2%, indicating better accuracy.



(a) By Input Size (b) By operator counts

Fig. 9. SHAP value distribution.

Insight: The proposed method outperforms existing model in predicting outliers.

6 EXPLAINING RUNTIME VARIATION

In this section, we conduct *descriptive analysis* to better understand the job characteristics that lead to different runtime distributions. Starting with the classification models from the previous section, we use machine learning explanation tools to better understand the sources of runtime variation.

6.1 Shapley Value

Shapley values [39, 66] that explain the contribution of each "player" in a game-theoretic setting have been adopted for explaining the contribution of features in ML models. In our context, based on the predictors developed in Section 5, they explain the quantitative contribution of each feature by randomly permuting other feature values and evaluating the marginal changes of the predictions [42].

Figure 9a illustrates the distribution of Shapley values with respect to the total input data read, where each dot corresponds to one job instance. We can see that jobs with large input size are more likely to be in Cluster 6 (as their feature values lead to higher Shapley values and a thus higher likelihood of being in Cluster 6) using Delta-normalization. Note that Cluster 6 has a relatively high variance and high probability of outliers. Similar trends can be found for jobs with fewer tokens.

Insight: Jobs with larger inputs and using fewer tokens are more likely to have a large variation. A larger number of tokens can potentially evacuate other jobs from the same machine, which potentially reduces interference and the impact of noisy neighbors.

Similarly, job characteristics such as operator counts significantly impact the prediction, indicating that the existence of certain operators more likely results in different runtime distributions (see Figure 9b for Shapley values for Cluster 6 with Delta-normalization).

Insight: Certain operator counts, such as Index-Lookup, Window, and Range, increase the variation.

Using Ratio-normalization, Cluster 0 has smaller variance and smaller probability of outliers than Cluster 2, while both have two modes. Focusing on a set of high-importance features, we compare the Shapley values for predicting these two clusters and found that, with lower CPU utilization, standard deviation and low usage of spare tokens, jobs are more likely to be in Cluster 0 (with more reliable performance) compared to Cluster 2. In general, we expect that machines with high utilization levels or standard deviations will have less reliable performance, which coincides

with our observations here. The usage of spare tokens (whose availability is less predictable) can also lead to less stable runtimes.

> **Insight:** Lower CPU utilization (load), lower standard deviation, and less use of spare tokens can improve runtime reliability.

For Ratio-normalization, increasing the vertex count on Gen5 and Gen6 (newer generations) tends to shift the prediction to Clusters 0 and 1, indicating that running vertices on those machine SKUs can potentially help with the runtime variation. Compared with Gen3 and Gen4 machines, those are in general faster and with large resource capacity [83].

> **Insight:** The model identified certain SKUs where larger vertex count on those machines increases the likelihood of Clusters 0 and 1, which have smaller variance.

7 CONTROLLING RUNTIME VARIATION

Using the predictive model (Section 5) and drawing from the insights from the Shapley values (Section 6), we now identify several what-if scenarios for scheduling and resource allocation and evaluate their performance. Based on the changes of jobs' runtime, one can quantitatively evaluate the detailed monetary impact.

7.1 Scenario1: Spare Token allocation

Availability of spare tokens depends on physical cluster conditions that are affected by the execution of other jobs and hence is a source of variation. Here we use our models to estimate the impact on runtime variation if spare tokens are not allocated.

With the predictor, we disable spare tokens for all jobs in the test set (dataset D3 as in Table 1). With Ratio-normalization, 15% of jobs that were predicted in Cluster 2 are now in Cluster 1, where the outlier probabilities gap between 25th and 75th percentiles, and the 95th percentile of the normalized runtime are reduced (also see Table 2). The second significant change is from Cluster 3 to 5 where the gap between 25 and 75th percentile increased slightly from 0.17 to 0.19, while the standard deviation decreased dramatically (from 1.45 to 0.82). Similar changes can be seen for Delta-normalization. Based on the predictor, for the set of jobs desiring low variances, we propose disabling their usage of spare tokens to maximize performance improvement. In the production system, there has been work ongoing to reduce the maximum number for spare tokens as a multiplier of the number of allocated tokens. We observed that the jobs with fewer spare tokens runs slower but with less variance, which agrees with our model predictions.

7.2 Scenario2: Scheduling on later generation of machines

A job's vertices can be executed by multiple machines in a distributed manner and as discussed in Section 3.2, different job instances within the same group can be allocated to many different SKUs. Here we estimate the impact on runtime variation if we execute more vertices on later generations of machines.

By shifting all the vertices (both fractions and count) from Gen3.5 to Gen5.2, the most likely change for 20.95% of jobs is from Cluster 2 to 0, with a significant drop in the gap between 25 and 75th percentile for Ratio-normalization. And for Delta-normalization, the most likely prediction change is from Cluster 1 to 0 where the gap between 25 and 75th percentile dropped from 11s to 4s.

Hence, it's better to run more vertices on later generation SKUs. However, our model doesn't capture the compounding of changes due to workload re-balancing, such as the changes of CPU utilization levels. Models that can predict the utilization levels given different workload distributions can be easily integrated, such as in KEA [83], to quantitatively capture this dynamic impact.

7.3 Scenario3: Improving load balance

As discussed in Section 3.2, physical cluster conditions such as load differences across machines are a source of runtime variation. Here we estimate this impact of more uniformly distributed loads.

If the standard deviation of CPU utilization could be reduced to 0 (i.e., equal load on all machines and at all times), with Ratio-normalization, the most likely change is from Cluster 2 to 0 (29.78% of jobs), with outlier probability reduced from 1.66% to 1.63% and the variation measured by the difference between the 25^{th} and 75^{th} percentiles reduced from 0.16 to 0.06. Similar improvement can be seen for Delta-normalization. Thus there is significant monetary value that can be realized by a better scheduling system that further motivates other research projects for Cosmos.

8 RELATED WORK

A number of works have studied the problem of predicting job runtimes under different resource allocations and platform parameter settings [19, 26, 28, 28, 29, 46–48, 51]. Morpheus [33] examines resource usages of recurring jobs and finds the best-fitting pattern that it uses for better resource allocation and scheduling, but does not predict the outlier probability for a given job. Runtime estimation has also been used to infer job scheduling to support policies, such as Shortest Processing time First (SPF) [23, 36, 70, 84]. Zrigui et al. [85] clustered job instances into small and large based on runtime and used a classification algorithm to inform their job scheduler with high accuracy. In our work, we derived a larger number of clusters for delta- and ratio-normalized runtime distributions with richer information (percentiles, variances, etc.), providing a comprehensive oversight of customer experiences.

Schad et al. [56] studied performance variation on Amazon EC2 [2] by running benchmarks for CPU and Memory performance, Disk I/O, and network bandwidth. Feitelson et al. [20] examined a parallel scientific workload on a 128-node cluster at NASA Ames and presented the changes of job submission rate, system utilization, and the distributions of job characteristics such as job type, runtime, and degree of parallelism. However, they do not model or predict the job runtime variation.

Prior works have proposed automated methods, including ML techniques, for analyzing system failures, slowdowns, and potential anomalies [4, 12, 18, 65]. Causal inference and dependency/graph learning may also be used [27, 37, 40, 45, 53, 77, 79] for these applications. However, in our work, where there is a large number of feature dimensions and complex correlation between groups of features, such methods require manual input to tune the dependency structure such as adding or deleting a detected dependency link and might still be biased. In our work, we do not manually craft dependencies among causes of variation but use Shapley scores for inference.

Huang et al. [30, 31] investigated the causes for query latency variation in transactional databases. They used variance (and covariance) as the primary metric for quantifying variation and built a variance tree corresponding to the call graph for gaining insights into the contributors of variance. In contrast, we use runtime *distributions* for predicting variation, instead of scalar metrics due to the insufficiency of the latter in fully characterizing or explaining variation. Additionally, we use feature importance and Shapley scores to identify the main contributors to the prediction for the distribution to which a new run of a job may belong.

Prior work has also explored reducing performance variation with concurrent queries. Crescando [71] presents a relational table implementation that prioritizes predictable query performance over optimal performance through design choices such as having a scan-only architecture without indexes and new collaborative scan and update-join algorithms. CJOIN [10] introduces a new join operator and shares computation and resources among concurrent queries to improve both throughput and performance stability. Augmenting our prescriptive analysis with capabilities for

evaluating the impact of computation sharing and other optimizations for concurrent queries is an interesting direction for future work.

For tuning job performance, Black-box optimization such as Bayesian optimization [5, 15], gradient descent [72, 75], etc. requires multiple runs of experiments. In this work, we provide a one-shot method that can directly determine the best course of action.

9 CONCLUSION

In this work, we did an extensive analysis of the runtime variation of recurring production jobs on Microsoft Cosmos by systematically characterizing, modeling, predicting, and explaining job runtime variations. Our original 2-step approach computes a posterior likelihood for each job to associate it with a predefined probability distribution, whose shape differs according to (1) intrinsic job characteristics, (2) resource allocation and (3) cluster condition when the job is submitted. We infer the distribution of job runtime with >96% accuracy, out-performing the traditional regression models and capturing better the long tail of the distribution. Using an interpretable machine learning algorithm, we examined the sources of variation such as usage of spare tokens, skewed load on computing nodes, fractions of vertices executed on different SKUs. We quantified the improvement by adjusting these control variables. Our techniques can be used along with models that capture the effects on system utilization with workload re-balancing to dynamically optimize the performance of individual jobs.

REFERENCES

[1] 2022. *Stalagmite*. Retrieved 2022 from https://en.wikipedia.org/wiki/Stalagmite

[2] Amazon. 2022. *Amazon EC2*. Retrieved Feb 15, 2022 from https://aws.amazon.com/aws/ec2

[3] Amazon.com, Inc. 2020. *Amazon Athena*. Retrieved July 4, 2020 from https://aws.amazon.com/athena/

[4] Kanishka Bhaduri, Kamalika Das, and Bryan L Matthews. 2011. Detecting abnormal machine characteristics in cloud infrastructures. In *2011 IEEE 11th International Conference on Data Mining Workshops*. IEEE, 137–144.

[5] Christopher M Bishop and Nasser M Nasrabadi. 2006. *Pattern recognition and machine learning*. Vol. 4. Springer.

[6] Eric Boutin, Paul Brett, Xiaoyu Chen, Jaliya Ekanayake, Tao Guan, Anna Korsun, Zhicheng Yin, Nan Zhang, and Jingren Zhou. 2015. Jetscope: Reliable and interactive analytics at cloud scale. *Proceedings of the VLDB Endowment* 8, 12 (2015), 1680–1691.

[7] Eric Boutin, Jaliya Ekanayake, Wei Lin, Bing Shi, Jingren Zhou, Zhengping Qian, Ming Wu, and Lidong Zhou. 2014. Apollo: Scalable and coordinated scheduling for cloud-scale computing. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*. 285–300.

[8] Nicolas Bruno, Sameer Agarwal, Srikanth Kandula, Bing Shi, Ming-Chuan Wu, and Jingren Zhou. 2012. Recurring job optimization in scope. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*. 805–806.

[9] Nicolas Bruno, Sapna Jain, and Jingren Zhou. 2013. Continuous cloud-scale query optimization and processing. *Proceedings of the VLDB Endowment* 6, 11 (2013), 961–972.

[10] George Candea, Neoklis Polyzotis, and Radek Vingralek. 2011. Predictable performance and high query concurrency for data analytics. *The VLDB Journal* 20, 2 (2011), 227–248.

[11] Ronnie Chaiken, Bob Jenkins, Per-Åke Larson, Bill Ramsey, Darren Shakib, Simon Weaver, and Jingren Zhou. 2008. SCOPE: easy and efficient parallel processing of massive data sets. *Proceedings of the VLDB Endowment* 1, 2 (2008), 1265–1276.

[12] Roshan Chitrakar and Chuanhe Huang. 2012. Anomaly based intrusion detection using hybrid learning approach of combining k-medoids clustering and naive bayes classification. In *2012 8th International Conference on Wireless Communications, Networking and Mobile Computing*. IEEE, 1–5.

[13] Andrew Chung, Subru Krishnan, Konstantinos Karanasos, Carlo Curino, and Gregory R Ganger. 2020. Unearthing inter-job dependencies for better cluster scheduling. In *14th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 20)*. 1205–1223.

[14] Andrew Chung, Subru Krishnan, Konstantinos Karanasos, Carlo Curino, and Gregory R. Ganger. 2020. Unearthing inter-job dependencies for better cluster scheduling. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. 1205–1223.

[15] Carlo Curino, Neha Godwal, Brian Kroth, Sergiy Kuryata, Greg Lapinski, Siqi Liu, Slava Oks, Olga Poppe, Adam Smiechowski, Ed Thayer, et al. 2020. MLOS: An infrastructure for automated software performance engineering. In *Proceedings of the Fourth International Workshop on Data Management for End-to-End Machine Learning*. 1–5.

[16] Carlo Curino, Subru Krishnan, Konstantinos Karanasos, Sriram Rao, Giovanni M Fumarola, Botong Huang, Kishore Chaliparambil, Arun Suresh, Young Chen, Solom Heddaya, et al. 2019. Hydra: a federated resource manager for data-center scale analytics. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*. 177–192.

[17] Francesco Diaz and Roberto Freato. 2018. Azure Data Lake Store and Azure Data Lake Analytics. In *Cloud Data Design, Orchestration, and Management Using Microsoft Azure*. Springer, 327–392.

[18] Songyun Duan, Shivnath Babu, and Kamesh Munagala. 2009. Fa: A system for automating failure diagnosis. In *2009 IEEE 25th International Conference on Data Engineering*. IEEE, 1012–1023.

[19] Zhiwei Fan, Rathijit Sen, Paraschos Koutris, and Aws Albarghouthi. 2020. Automated Tuning of Query Degree of Parallelism via Machine Learning. In *Proceedings of the Third International Workshop on Exploiting Artificial Intelligence Techniques for Data Management*. Article 2, 4 pages.

[20] Dror G Feitelson and Bill Nitzberg. 1995. Job characteristics of a production parallel scientific workload on the NASA Ames iPSC/860. In *workshop on job scheduling strategies for parallel processing*. Springer, 337–360.

[21] Andrew D Ferguson, Peter Bodik, Srikanth Kandula, Eric Boutin, and Rodrigo Fonseca. 2012. Jockey: guaranteed job latency in data parallel clusters. In *Proceedings of the 7th ACM european conference on Computer Systems*. 99–112.

[22] Hans Fischer. 2011. *A history of the central limit theorem: From classical to modern probability theory*. Springer.

[23] Eric Gaussier, David Glesser, Valentin Reis, and Denis Trystram. 2015. Improving backfilling by using machine learning to predict running times. In *SC'15: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 1–10.

[24] Ramanathan Gnanadesikan and Martin B Wilk. 1968. Probability plotting methods for the analysis of data. *Biometrika* 55, 1 (1968), 1–17.

[25] Gregg, Brendan. 2022. *CPU Utilization is Wrong*. Retrieved Oct 4, 2022 from https://www.brendangregg.com/blog/2017-05-09/cpu-utilization-is-wrong.html

[26] Jian Guo, Akihiro Nomura, Ryan Barton, Haoyu Zhang, and Satoshi Matsuoka. 2018. Machine learning predictions for underestimation of job runtime on HPC system. In *Asian Conference on Supercomputing Frontiers*. Springer, Cham, 179–198.

[27] David Heckerman and John S Breese. 1996. Causal independence for probability assessment and inference using Bayesian networks. *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans* 26, 6 (1996), 826–831.

[28] Herodotos Herodotou, Fei Dong, and Shivnath Babu. 2011. No One (Cluster) Size Fits All: Automatic Cluster Sizing for Data-Intensive Analytics *(SOCC '11)*. Association for Computing Machinery, New York, NY, USA, Article 18, 14 pages. https://doi.org/10.1145/2038916.2038934

[29] Zhiyao Hu, Dongsheng Li, Dongxiang Zhang, and Yixin Chen. 2020. ReLoca: Optimize Resource Allocation for Data-parallel Jobs using Deep Learning. In *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications*. 1163–1171.

[30] Jiamin Huang, Barzan Mozafari, Grant Schoenebeck, and Thomas Wenisch. 2016. Identifying the major sources of variance in transaction latencies: Towards more predictable databases. *arXiv preprint arXiv:1602.01871* (2016).

[31] Jiamin Huang, Barzan Mozafari, Grant Schoenebeck, and Thomas F Wenisch. 2017. A top-down approach to achieving performance predictability in database systems. In *Proceedings of the 2017 ACM International Conference on Management of Data*. 745–758.

[32] Alekh Jindal, Hiren Patel, Abhishek Roy, Shi Qiao, Zhicheng Yin, Rathijit Sen, and Subru Krishnan. 2019. Peregrine: Workload Optimization for Cloud Query Engines. In *Proceedings of the ACM Symposium on Cloud Computing*. 416–427.

[33] Sangeetha Abdu Jyothi, Carlo Curino, Ishai Menache, Shravan Matthur Narayanamurthy, Alexey Tumanov, Jonathan Yaniv, Ruslan Mavlyutov, Íñigo Goiri, Subru Krishnan, Janardhan Kulkarni, et al. 2016. Morpheus: Towards automated slos for enterprise clusters. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. 117–134.

[34] Konstantinos Karanasos, Sriram Rao, Carlo Curino, Chris Douglas, Kishore Chaliparambil, Giovanni Matteo Fumarola, Solom Heddaya, Raghu Ramakrishnan, and Sarvesh Sakalanaga. 2015. Mercury: Hybrid centralized and distributed scheduling in large shared clusters. In *2015 USENIX Annual Technical Conference (USENIX ATC 15)*. 485–497.

[35] Marvin Karson. 1968. Handbook of Methods of Applied Statistics. Volume I: Techniques of Computation Descriptive Methods, and Statistical Inference. Volume II: Planning of Surveys and Experiments. IM Chakravarti, RG Laha, and J. Roy, New York, John Wiley; 1967, $9.00.

[36] Michael Kuchnik, Jun Woo Park, Chuck Cranor, Elisabeth Moore, Nathan DeBardeleben, and George Amvrosiadis. 2019. This is why ML-driven cluster scheduling remains widely impractical. *Tech. rep.* (2019).

[37] Po-Ling Loh and Peter Bühlmann. 2014. High-dimensional learning of linear causal networks via inverse covariance estimation. *The Journal of Machine Learning Research* 15, 1 (2014), 3065–3105.
[38] Gilles Louppe, Louis Wehenkel, Antonio Sutera, and Pierre Geurts. 2013. Understanding variable importances in forests of randomized trees. *Advances in neural information processing systems* 26 (2013).
[39] Scott M Lundberg and Su-In Lee. 2017. A unified approach to interpreting model predictions. *Advances in neural information processing systems* 30 (2017).
[40] Panagiotis Mandros, Mario Boley, and Jilles Vreeken. 2017. Discovering reliable approximate functional dependencies. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 355–363.
[41] Microsoft Corporation. 2022. *LGBMClassifier*. Retrieved Feb 4, 2022 from https://lightgbm.readthedocs.io/en/latest/pythonapi/lightgbm.LGBMClassifier.html
[42] Christoph Molnar. 2020. *Interpretable machine learning*. Lulu. com.
[43] Mina Naghshnejad and Mukesh Singhal. 2018. Adaptive online runtime prediction to improve HPC applications latency in cloud. In *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*. IEEE, 762–769.
[44] Frank Nielsen. 2016. *Introduction to HPC with MPI for Data Science*. Springer.
[45] Harsha Nori, Samuel Jenkins, Paul Koch, and Rich Caruana. 2019. Interpretml: A unified framework for machine learning interpretability. *arXiv preprint arXiv:1909.09223* (2019).
[46] Ilia Pietri, Gideon Juve, Ewa Deelman, and Rizos Sakellariou. 2014. A performance model to estimate execution time of scientific workflows on the cloud. In *2014 9th Workshop on Workflows in Support of Large-Scale Science*. IEEE, 11–19.
[47] Anish Pimpley, Shuo Li, Rathijit Sen, Soundararajan Srinivasan, and Alekh Jindal. 2022. Towards Optimal Resource Allocation for Big Data Analytics. In *25th International Conference on Extending Database Technology (EDBT)*. 338–350.
[48] Anish Pimpley, Shuo Li, Anubha Srivastava, Vishal Rohra, Yi Zhu, Soundararajan Srinivasan, Alekh Jindal, Hiren Patel, Shi Qiao, and Rathijit Sen. 2021. Optimal Resource Allocation for Serverless Queries. *arXiv preprint arXiv:2107.08594* (2021).
[49] Conor Power, Hiren Patel, Alekh Jindal, Jyoti Leeka, Bob Jenkins, Michael Rys, Ed Triou, Dexin Zhu, Lucky Katahanas, Chakrapani Bhat Talapady, et al. 2021. The Cosmos Big Data platform at Microsoft: over a decade of progress and a decade to look forward. *Proceedings of the VLDB Endowment* 14, 12 (2021), 3148–3161.
[50] Shi Qiao, Adrian Nicoara, Jin Sun, Marc Friedman, Hiren Patel, and Jaliya Ekanayake. 2019. Hyper dimension shuffle: Efficient data repartition at petabyte scale in scope. *Proceedings of the VLDB Endowment* 12, 10 (2019), 1113–1125.
[51] Kaushik Rajan, Dharmesh Kakadia, Carlo Curino, and Subru Krishnan. 2016. PerfOrator: Eloquent Performance Models for Resource Optimization. In *Proceedings of the Seventh ACM Symposium on Cloud Computing*. 415–427.
[52] Raghu Ramakrishnan, Baskar Sridharan, John R Douceur, Pavan Kasturi, Balaji Krishnamachari-Sampath, Karthick Krishnamoorthy, Peng Li, Mitica Manu, Spiro Michaylov, Rogério Ramos, et al. 2017. Azure data lake store: a hyperscale distributed file service for big data analytics. In *Proceedings of the 2017 ACM International Conference on Management of Data*. 51–63.
[53] Garvesh Raskutti and Caroline Uhler. 2018. Learning directed acyclic graph models based on sparsest permutations. *Stat* 7, 1 (2018), e183.
[54] Lior Rokach and Oded Maimon. 2005. Clustering methods. In *Data mining and knowledge discovery handbook*. Springer, 321–352.
[55] Aurobindo Sarkar and Amit Shah. 2018. *Learning AWS: Design, build, and deploy responsive applications using AWS Cloud components*. Packt Publishing Ltd.
[56] Jörg Schad, Jens Dittrich, and Jorge-Arnulfo Quiané-Ruiz. 2010. Runtime Measurements in the Cloud: Observing, Analyzing, and Reducing Variance. *PVLDB* 3, 1–2 (2010), 460–471. https://doi.org/10.14778/1920841.1920902
[57] Scikit-Learn. 2022. *EnsembledClassifier*. Retrieved Feb 4, 2022 from https://scikit-learn.org/stable/modules/ensemble.html
[58] Scikit-Learn. 2022. *GaussianNB*. Retrieved Feb 4, 2022 from https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html
[59] Scikit-Learn. 2022. *GradientBoostingClassifier*. Retrieved Feb 4, 2022 from https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html
[60] Scikit-Learn. 2022. *GradientBoostingClassifier*. Retrieved Feb 4, 2022 from https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html
[61] Scikit-Learn. 2022. *RandomForestClassifier*. Retrieved Feb 4, 2022 from https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
[62] David Sculley. 2010. Web-scale k-means clustering. In *Proceedings of the 19th international conference on World wide web*. 1177–1178.
[63] Rathijit Sen, Alekh Jindal, Hiren Patel, and Shi Qiao. 2020. AutoToken: Predicting Peak Parallelism for Big Data Analytics at Microsoft. *PVLDB* 13, 12 (2020), 3326–3339.

[64] Rathijit Sen, Abhishek Roy, Alekh Jindal, Rui Fang, Jeff Zheng, Xiaolei Liu, and Ruiping Li. 2021. AutoExecutor: predictive parallelism for spark SQL queries. *Proceedings of the VLDB Endowment* 14, 12 (2021), 2855–2858.
[65] Liqun Shao, Yiwen Zhu, Siqi Liu, Abhiram Eswaran, Kristin Lieber, Janhavi Mahajan, Minsoo Thigpen, Sudhir Darbha, Subru Krishnan, Soundar Srinivasan, et al. 2019. Griffon: Reasoning about Job Anomalies with Unlabeled Data in Cloud-based Platforms. In *Proceedings of the ACM Symposium on Cloud Computing*. 441–452.
[66] S Shapley Ll. 1953. A value for n-person games. *Contributions to the Theory of Games II, Annals of Mathematical Studies* 28 (1953).
[67] Ozan Sonmez, Nezih Yigitbasi, Alexandru Iosup, and Dick Epema. 2009. Trace-based evaluation of job runtime and queue wait time predictions in grids. In *Proceedings of the 18th ACM international symposium on High performance distributed computing*. 111–120.
[68] Ashish Thusoo, Joydeep Sen Sarma, Namit Jain, Zheng Shao, Prasad Chakka, Suresh Anthony, Hao Liu, Pete Wyckoff, and Raghotham Murthy. 2009. Hive: a warehousing solution over a map-reduce framework. *Proceedings of the VLDB Endowment* 2, 2 (2009), 1626–1629.
[69] Jordan Tigani and Siddartha Naidu. 2014. *Google BigQuery Analytics*. John Wiley & Sons.
[70] Dan Tsafrir, Yoav Etsion, and Dror G Feitelson. 2007. Backfilling using system-generated predictions rather than user runtime estimates. *IEEE Transactions on Parallel and Distributed Systems* 18, 6 (2007), 789–803.
[71] Philipp Unterbrunner, Georgios Giannikis, Gustavo Alonso, Dietmar Fauser, and Donald Kossmann. 2009. Predictable performance for unpredictable workloads. *Proceedings of the VLDB Endowment* 2, 1 (2009), 706–717.
[72] Dana Van Aken, Andrew Pavlo, Geoffrey J Gordon, and Bohan Zhang. 2017. Automatic database management system tuning through large-scale machine learning. In *Proceedings of the 2017 ACM international conference on management of data*. 1009–1024.
[73] Vinod Kumar Vavilapalli, Arun C Murthy, Chris Douglas, Sharad Agarwal, Mahadev Konar, Robert Evans, Thomas Graves, Jason Lowe, Hitesh Shah, Siddharth Seth, et al. 2013. Apache hadoop yarn: Yet another resource negotiator. In *Proceedings of the 4th annual Symposium on Cloud Computing*. ACM, 5.
[74] Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, Ion Stoica, et al. 2010. Spark: Cluster computing with working sets. *HotCloud* 10, 10-10 (2010), 95.
[75] Ji Zhang, Yu Liu, Ke Zhou, Guoliang Li, Zhili Xiao, Bin Cheng, Jiashu Xing, Yangtao Wang, Tianheng Cheng, Li Liu, et al. 2019. An end-to-end automatic cloud database tuning system using deep reinforcement learning. In *Proceedings of the 2019 International Conference on Management of Data*. 415–432.
[76] Jiaxing Zhang, Hucheng Zhou, Rishan Chen, Xuepeng Fan, Zhenyu Guo, Haoxiang Lin, Jack Y Li, Wei Lin, Jingren Zhou, and Lidong Zhou. 2012. Optimizing data shuffling in data-parallel computation by understanding user-defined functions. In *Presented as part of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*. 295–308.
[77] Yunjia Zhang, Zhihan Guo, and Theodoros Rekatsinas. 2020. A statistical perspective on discovering functional dependencies in noisy data. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 861–876.
[78] Hai-Tao Zheng and Haiyang Zhang. 2015. Online feature selection based on passive-aggressive algorithm with retaining features. In *Asia-Pacific Web Conference*. Springer, 707–719.
[79] Pengfei Zheng and Benjamin C Lee. 2018. Hound: Causal learning for datacenter-scale straggler diagnosis. *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 2, 1 (2018), 1–36.
[80] Jingren Zhou, Nicolas Bruno, and Wei Lin. 2012. Advanced partitioning techniques for massively distributed computation. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*. 13–24.
[81] Jingren Zhou, Per-Ake Larson, and Ronnie Chaiken. 2010. Incorporating partitioning and parallel plans into the SCOPE optimizer. In *2010 IEEE 26th International Conference on Data Engineering (ICDE 2010)*. IEEE, 1060–1071.
[82] Yiwen Zhu, Matteo Interlandi, Abhishek Roy, Krishnadhan Das, Hiren Patel, Malay Bag, Hitesh Sharma, and Alekh Jindal. 2021. Phoebe: A Learning-Based Checkpoint Optimizer. *Proc. VLDB Endow.* 14, 11 (jul 2021), 2505–2518. https://doi.org/10.14778/3476249.3476298
[83] Yiwen Zhu, Subru Krishnan, Konstantinos Karanasos, Isha Tarte, Conor Power, Abhishek Modi, Manoj Kumar, Deli Zhang, Kartheek Muthyala, Nick Jurgens, et al. 2021. KEA: Tuning an Exabyte-Scale Data Infrastructure. In *Proceedings of the 2021 International Conference on Management of Data*. 2667–2680.
[84] Dmitry Zotkin and Peter J Keleher. 1999. Job-length estimation and performance in backfilling schedulers. In *Proceedings. The Eighth International Symposium on High Performance Distributed Computing (Cat. No. 99TH8469)*. IEEE, 236–243.
[85] Salah Zrigui, Raphael Y de Camargo, Arnaud Legrand, and Denis Trystram. 2022. Improving the performance of batch schedulers using online job runtime classification. *J. Parallel and Distrib. Comput.* 164 (2022), 83–95.

Received July 2022; revised October 2022; accepted November 2022