# CENSUS: Counting Interleaved Workloads on Shared Storage

Si Chen
*Emory University*
Atlanta, USA
si.chen2@emory.edu

Jianqiao Liu
*Emory University*
Atlanta, USA
jianqiao.liu@emory.edu

Avani Wildani
*Emory University*
Atlanta, USA
avani@mathcs.emory.edu

***Abstract*—Understanding the different workload-dependent factors that impact the latency or reliability of a storage system is essential for SLA satisfaction and fair resource provisioning. However, due to the volatility of system behavior under multiple workloads, determining even the *number* of concurrent types of workload functions, a necessary precursor to workload separation, is an unsolved problem in the general case.**

**We introduce CENSUS, a novel classification framework that combines time-series analysis with gradient boosting to identify the number of *functional workloads* in a shared storage system by projecting workload traces into a high-dimensional feature representation space. We show that CENSUS can distinguish the number of interleaved workloads in a real-world trace segment with up to 95% accuracy, leading to a decrement of the mean square error to as little as 5% compared to the fairest guess according to the daily average.**

***Index Terms*—workload analysis, machine learning, interleaved workloads, workload feature extraction.**

## I. INTRODUCTION

A storage system which provides fine grained monitoring and optimization strategies would help data-driven enterprises to gain greater value from their mass data. However, storage system tuning is a difficult, delicate balance of reliability, availability, security, and performance concerns spread over all of the different types of usage patterns that the system may encounter. As storage systems increase in size, in order to reduce hardware and energy consumption, multi-tenancy, or multiple workloads sharing the same space has become commonplace. As a result of normal OS processes such as batching or CPU scheduling, I/Os from different workloads tend to appear interleaved when viewed in a trace. In a recent study, 60% of traces recorded were strided and composed of interleaved workloads [37], which makes performance management and optimization more complicated.

This work is funded in part by NSF-CRII #CNS1755958

The general definition of a "workload" often corresponds to a user layer application, where each workload may have several phases (e.g. first heavy sequential reads, then some random reads, then sequential writes). When discussing system provisioning, we care more about the observable metrics, such as cost or latency, that a workload may contribute to than the source application. To this end, we propose *functional workload*, or *fworkloads*, as a functionally distinct usage of a storage system. For this work, we define *functionally distinct* as distinctly impact system metrics. Once we identify and distinguish the fworkloads, we posit future system configuration would be more carefully provisioned and equitable. For example, a fworkload characterized by sequential write accesses would be a good candidate offloading onto a LFS (log-structured file system) partition but not to an SSD (Solid State Drives). Thus, our performance optimization is not per application, but instead per function.

A storage provider typically knows the different tenants that are present on their system. Our goal is not to re-derive this number. Instead, we want to know what different individual *functional use cases* co-exist at any given time, which is essential for automated storage performance management. For example, imagine that developers, software testers, and managers are working in VMs that rely on the shared underlying storage. While there may be dozens of tenants in the system, there may be only three types of storage uses, or fworkloads, disparate enough to justify separate provisioning. Conversely, a single client may have multiple, fworkloads that would benefit from specialized storage tuning [12].

Unlike other indicators, we do not have fworkload labels in a trace. We use the process ID (PID) associated with an individual I/O as a non-ideal proxy for the ground truth fworkload labels in lieu of a better validation parameter. Similar choices could be the user

ID (UID) or process name. Although PID is also not a perfect proxy for fworkload, it is a sufficient stand-in for our current work, because our traces are user level, so PIDs tend to have limited functional variation.

Although systems with multiple fworkloads are increasingly commonplace, characterization of multiple fworkloads within a single system trace remains difficult because interleaved fworkloads are highly dynamic. Previous attempts to separate workloads have been stymied by the fact that every technique requires the *number* of workloads as an input [10]. Counting the number of fworkloads in a system requires a deep understanding of how the system I/O is cross-correlated to itself and how it changes over time. Common workload characterization features in the literature typically are aggregate values such as "read count" or simple heuristics, such as "sequentiality," that fail to capture the level of complexity we need to answer this superficially trivial question [4], [7]. Previous efforts in this space rely on ground truth values provided by system operators, but often customers do not know how their storage systems are being used [36]. As the hybrid multi-cloud becomes common for storage deployments, such decision-making solutions shift complexity and responsibility to customers who perform low-level storage operation and maintenance. Thus, identifying the number of independent fworkloads contributing to a sample trace is itself a difficult, important, and unsolved problem.

We propose CENSUS, a novel classification framework to predict the number of interleaved fworkloads in I/O traces. CENSUS uses novel time-series methods to select fworkload features customized for a tree-based classifier to predict the number of interleaved fworkloads. By projecting the trace into a high-dimensional feature representation space, we extract features such as *address complexity* and *address change quantiles* that are not typically collected directly by administrators, and interpret features with system readable meanings. This extension of fworkload features is crucially, highly dissimilar to prior workload analysis such as *IOPS* or *read/write ratio*, opening the field to insights derivable from formerly overlooked metrics.

In our experiments, we evaluate the performance based on existing public I/O traces and a newly collected trace of a deep learning workload from a research server. We intend to publicly release this trace. We demonstrate that, over sliding windows of a storage trace, we can identify the number of interleaved fworkloads with as little as 5% error and up to 95% accuracy over three real datasets. If we extend our classifier to predicting a close approximation of our number of fworkloads, CENSUS maintains an average 90% accuracy. We further demonstrate that CENSUS significantly decreases the mean squared error of the current best technique to separate interleaved fworkloads.

Our primary contributions include:

1) We introduce CENSUS, a gradient and feature based learning framework that returns the number of fworkloads represented in an I/O trace, demonstrating that functionally identifying the number of fworkloads is tractable for real fworkloads. CENSUS is publicly available on GitHub.
2) We demonstrate that CENSUS improves fworkload separation in a test case.
3) We identify over 700 storage trace features that broaden the options available to the storage community for workload characterization.

## II. BACKGROUND AND RELATED WORK

Workloads are considered interleaved if the accesses of multiple workloads, or functionally distinguishable I/O patterns, are commingled. Several studies have been performed to identify or disentangle interleaved workloads. Oh *et al.* [33] shows that under the certain system, a supervised classifier is able to predict the presence of the original workloads from a mixture of at most three workloads. Chen *et al.* [10] demonstrates an independent component analysis based method to separate interleaved workloads. Both Oh and Chen, as well as every other interleaved workload separation project [25], [28] we are aware of, need the number of workloads expected in the trace as input.

The few separation methods that do not require the number of workloads as input require hyperparameters such as similarity and density thresholds that are even more fragile [16]. Though projects such as Crystal [17] discuss how to implement provisioning solutions in the presence of multiple workloads, to the best of our knowledge no one has successfully separated interleaved workloads without receiving the number of workloads out of band.

### A. Workload Feature Extraction

Recently, some studies have started to learn memory access patterns directly [33], [44], [29], [6], [37] to characterize or identify workloads. However, fworkloads are difficult to rigorously define with the features typically used for characterization [12]. Firstly, many of these workloads' attributes (*e.g.*, I/O rate, queue length, request disk arrival rate, or inter-arrival time) are file system or environment dependent [4]. Also, while there

is a wealth of work analyzing both block-level [3], [23], [22] and file system [13], [1] traces to optimize storage system design, most studies focus on understanding the behavior and characteristics of specific storage systems. Signature based methods are often more generalizable [29], [4], but they cannot handle highly dynamic interleaved workloads, which workloads on mixed-use systems are more likely to be [35], [7]. BPP [49] reduces the time and space overhead by classifying block access patterns into simple and compound ones based on the mining costs of different patterns and differentiates the mining policies for I/O prediction and data caching accordingly. However, BPP only concerns traditional sequential and strided patterns, which is insufficient when the workloads are interleaved.

Since there is no good enough fworkload characterization approach for shared storage, we need a method that returns a large breadth of features to capture the basic patterns from the interleaved trace. We chose to build our own extractor because existing characterizations return at best a few dozen features [42], which we found to be ineffective for the granularity of our work. Time-series methods analyze a series of data points indexed in time order to extract meaningful statistics and characteristics [18]. Several forecasting models for storage system workload analysis have been built based on time-series methods, though these projects do not attempt to expand the feature set for workload characterization [2], [38], [19]. Traditionally, Auto-regressive Integrated Moving Average (ARIMA) models are used for analyzing or predicting stationary time-series data, which is seasonal and has the same variance. Shape-based approaches identify similar pairs of time-series in terms of their values through time [40]. For example, Dynamic Time Warping (DTW), which accommodates local temporal shifts in the data, is specialized in considering an ensemble of dedicated time-series types [31]. We are not using time-series analysis to predict the next access address, which is impossible to be accurate given the sparse address space. Instead, we cast finding the number of fworkloads as a classification problem, using time-series analysis as a feature extraction tool to explore the internal structure of the I/O trace.

### *B. Learning Models*

Time-series information extraction is similar to text mining, and various deep learning models, particularly RNNs (*e.g.*, LSTM). have shown promise for text mining tasks [45]. However, due to the lack of publicly available large-scale workload data, it is hard to build a reasonable deep learning model for predicting the number of interleaved fworkloads. Also, training a deep learning model is time consuming and most importantly, the result is typically uninterpretable [9]. Interpretability is critical because administrators have a wealth of domain knowledge, which could both validate and refine fworkload counts that they understand. We address these concerns by using tree based methods [11], [24], [20] for CENSUS, which we discuss in detail in Section III-B1.

We also considered directly segmenting and clustering our traces. However, all clustering methods that do not require the number of clusters as input required hyperparameters that we could not rigorously tune. For example, while DBscan could automatically get the cluster number, it still needs the default similarity distance and density threshold to be defined [16].

## III. CENSUS DESIGN

CENSUS includes two main components: time-series based feature extraction to select fworkload attributes and a gradient boosting classification model that predicts the final fworkload number (Figure 1). The inputs to CENSUS are I/O trace segments of equal length. During training, we use the process ID (PID) associated with an individual I/O as a proxy for the fworkload type. While PID is an imperfect proxy for fworkload, over the window sizes we are considering, the PID landscape is demonstrated to be relatively static as well as functionally unique. For example, in"Home", the largest trace in the FIU dataset (Section V), we observed that the average number of PIDs for an application in the Home trace over 20 days is only 1.27, with an average standard deviation of 0.33. In the traces we study, 83% of applications have a static PID across the entire observation period.

We also collect traces on single-application servers; here, the server index is used as ground truth instead of PID. For both classes of data, we consider the 29 most prevalent indices, with a 30th index reserved for "Other." Though CENSUS is a general framework, we focus on block I/O traces because of their availability and ease of collection. Block I/O traces record the I/O events that happen on block devices, such as arrival time, mode (read and write), size, and logical block address (LBA). Since block I/O traces do not include I/O content, it has low overhead compared to other I/O logging, along with smaller data size and privacy concerns.

### *A. Feature Extraction*

Since an I/O trace may contain thousands of accesses per minute, we use time-series analysis as a data pre-processing step that maps a segment of the I/O trace to a set of lower-dimensional representations

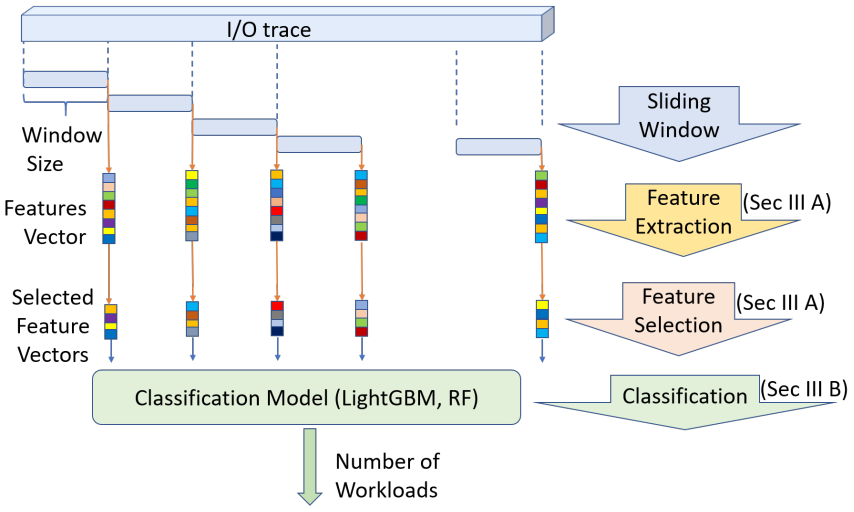



Fig. 1. CENSUS architecture. Features are extracted from the address value and time interval of I/O trace segments within a window. Labeled training data is then fed to a classification model, which predicts the number of fworkloads for unseen I/O traces.

called "features." Since we expect coarse-grained self-similarity in our traces, we do feature selection offline to capture the full spectrum of fworkload behavior. After experimenting with a variety of segment lengths, we settled on 10 minutes trace segments to balance between trace lengths that are too short for multiple fworkloads or too long to identify the transient burst. To take advantage of time-series analysis tools, we must either sub-sample the trace with uniform time intervals, which works poorly for fworkloads with varying intensity levels [26], or interpolate fake values into the I/O trace, which will affect the data authenticity. To avoid these issues, we add a time dimension to the data, using delta time $\Delta_N = Time_{N+1} - Time_N$ instead of raw time accesses. For example, if the original trace timestamps are $12345, 12380, 12467, 12493, \ldots$, the delta time trace would be $35, 87, 26, \ldots$.

Time-series techniques yield some familiar workload characterization features such as max, min, mean, median, variance, or standard variance. However, they also yield several features that are less readily intuitive such as entropy, and coefficient of some mathematical transform such as fast Fourier transform (FFT), or the *time reversal asymmetry statistic*, which is the expectation of the difference between some multiplication of two values with a certain time lag $t$. Features such as these lack a clear corresponding system behavior, and so they have not to this point been included in the post-hoc analyses of many workload characterization efforts, however, they may include some overlooked patterns of the trace.

Individually calculating these features is a burden, and it is hard to pre-determine which features are important. CENSUS extends the open source `tsfresh` library [14] to rapidly extract a large number of high-information features automatically.

After segmentation, we calculate 1576 features per segment trace based on the default feature extraction functions and parameter setting. To limit the number of irrelevant features, we judge feature *criticality*, the impact a feature has on the CENSUS classifier. Critical features in CENSUS are found by calculating

$$\arg\max_{feature} \sum_{Tree} Gain(Tree, feature)$$

, which returns the features with highest information gain per branch point in the tree.

### B. Classification

Once we select features, we need to build a classification model to determine the number of interleaved fworkloads in our sample. Our model considerations include applicability to a high-dimensional feature space, lightweight and interpretable model architecture, efficiency, and high, stable classification performance. To satisfy these constraints, we focus on the LightGBM [24] and Random Forest (RF) [20] algorithms. Both of these algorithms have self-contained feature selection, high training speed, and good prediction performance. Moreover, they both are tree based, allowing for non-expert interpretation as well as the potential for inclusion of out-of-band domain knowledge in the classifier. Additionally,

the optimal subset of correlated features can be found from initial features by checking the feature importance and tuning the parameters, including a maximum ratio of features for tree building.

*1) LightGBM:* LightGBM [24] is a tree based gradient boosting learning algorithm. While boosting converts weak learners (decision trees) into strong learners, we chose to use gradient boosting because it strengthens the learner by adjusting the training target to the residual error on each iteration. We use gradient boosting to produce a model from an ensemble of several weak decision trees using the log loss function and fit the gradient of the loss function. During the boosting portion of the algorithm, the loss function is optimized and a new weak tree is added to the ensemble. $\mathrm{F_m(X)} = \alpha_0 \mathrm{f_0(X)} + \alpha_1 \mathrm{f_1(X)} + \ldots + \alpha_\mathrm{m} \mathrm{f_m(x)}$. LightGBM grows the tree leaf-wise (vertically), while other tree based algorithms such as XgBoost [11] grow trees level-wise (horizontally). This vertical growth enables LightGBM to be parallelizable and converge much faster on real data [24]. The multi-class classification is based on log loss for $n$, which uses the softmax function $S\left(y_i\right) = \frac{e^{y_i}}{\sum_j e^{y_j}}$ to normalize the raw class probabilities. LightGBM uses a histogram method to bundle features in bins for the best tree split points, which further optimises memory usage and training speed by increasing cache hit likelihood.

LightGBM is underused because it has several dozen parameters, *e.g.*, max tree depth, learning rate, min data in leaf, max number of features, number of leaves, and regularization terms. We address this parameter tuning issue by using 5-fold RandomizedSearchCV[34], which efficiently searches the optimal parameters in a given range of possible parameter values and the number of training iterations. The parameters we chose are relatively robust to perturbation and can be found in Table I. While LightGBM shows good performance on large-scale datasets, it is traditionally sensitive to overfitting for small datas [30]. Fine-tuning model parameters is known to make LightGBM generalizable for smaller datasets, and our results show that overfitting is not an issue for LightGBM in CENSUS.

*2) Random Forests:* To address potential overfitting issues for small datasets trained with LightGBM, we also built a random forest [20] (RF) model to compare against. RF is an ensemble learning method constructing a multitude of decision trees. The predictions for unseen samples $x'$ can be made by averaging the predictions from all the individual decision trees on $x'$, while each tree has a random subset of features: $\hat{f} = \frac{1}{B} \sum_{b=1}^{B} f_b\left(x'\right)$. Using a random subspace method and random feature selection, RF does not increase generalization error when more trees are added. Section IV discusses the relative strengths of these 2 models in CENSUS.

*3) Training Methodology:* To train a generalized model that is robust to random time window size and arbitrary domain traces, we need to collect a large representative pool of training datasets. Currently, to counter potential class imbalance due to limitations in training set breadth, together with a huge distribution difference between the training set and unforeseen test set, we present 2 different training methods: **Generalized**, and **Identical Distribution (ID)**.

The generalized training space includes data from multiple domains segmented with various time windows to simulate different access intensities. With the high diversity and sufficient amount of training data, this model has high generalization ability and adaptability, which can capture fast changing fworkload characteristics.

With the **Identical Distribution (ID)** training method, for a dataset that contains $N$ independent data files, each file is tested using leave-one-out cross validation: *i.e.*, each file is tested using a model trained by the remaining $N-1$ files before the average performance is calculated. This method better tests model effectiveness when training data is scarce, resulting in a stricter, more domain-specific model since the training set and test set have a similar distribution.

## IV. EXPERIMENTAL RESULTS

We tested CENSUS on three real, publicly available block I/O datasets: FIU [26], MSR [32], and EMORYML, a dataset that we collected of a month of machine learning applications running on a research server. We were particularly interested in testing CENSUS on an ML workload because a large number of machine learning training jobs may be co-located on cloud servers and have unique characteristics such as cyclic memory usage [43]. In the absence of ML block I/O workloads, we collected EMORYML from our research server, which has 16 2.2GHz Intel CPUs and 65.9GB of memory and runs arch-Linux 5.1.5, while running machine learning training workloads. EMORYML is comprised of 30 days of block I/O traces collected by `blktrace` [5]. Tasks running on EMORYML include our daily research work of machine learning modeling and development, including deep learning model training and data preprocessing. Each I/O access is represented by a tuple of `<Time, PID, LBA, size in 512 Bytes blocks, Write or Read>`. These traces will be made publicly available on publication.

FIU [26] represents nearly three weeks of block I/O traces. The FIU dataset contains 8 sub-traces, which are Webmail, Online, Webresearch, Webusers, Home4, Home3, Home2, and Home1. We combine the 4 Home traces into a single "Home" to better compare to other data. Finally, MSR represents 1 week of block I/O traces from 36 different volumes on 13 enterprise servers. While the FIU and EMORYML traces are annotated with PID and process name, MSR lacks PIDs, but contains a volume ID that we use as a high-level indicator of "workload type" when combining traces for training. To combine the MSR traces, we reordered data across all MSR volumes by time and each record is labeled with the original volume ID. We randomly chose 23 of 36 volumes to limit training time.

### A. *Extracted Features*

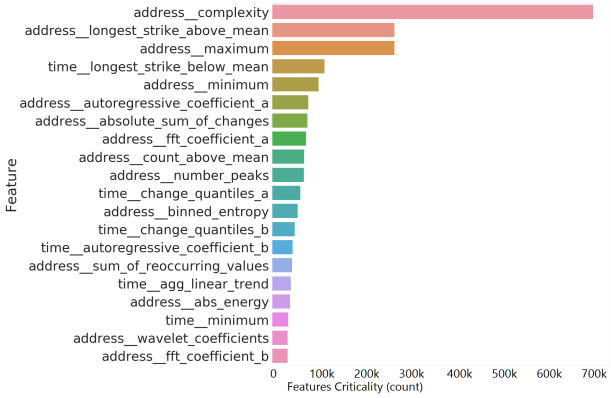



Fig. 2. LightGBM feature selection example. The LightGBM API measures the feature criticality by counting the total loss fuction reduction (Section III-A) when using each feature for splitting.

Figure 2 shows 20 features with highest overall criticality in our LightGBM classifier. Critical features were similar for RF. The prefix appended to the feature name ("address" or "time") indicates whether the feature is spatial or temporal.

*1) Feature Universality:* Feature criticality was Zipfian across the 1576 original features we extracted, so our first step was to drop all features that had little to no information gain. This left us with 700 features. To better understand the global feature criticality and, from there, baseline I/O activities in our fworkloads, we then map feature criticality across fworkloads and training models for the most critical features (Figure 3). We observe that, although the order of top features may differ for each dataset, the main important features come from several consistent categories. These features are either calculated

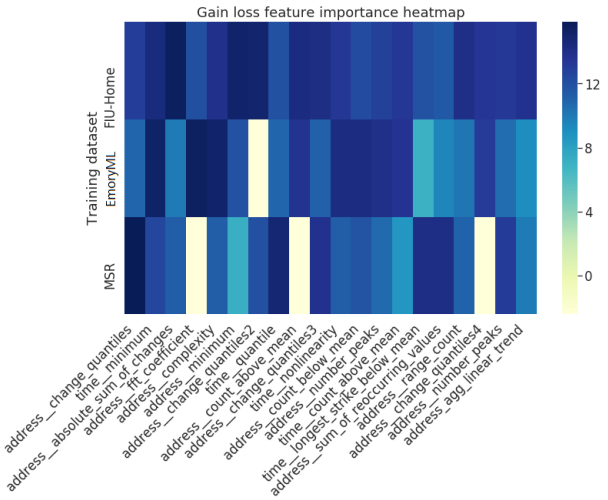



Fig. 3. Feature criticality heat map. For LightGBM classification models with FIU, MSR, and EMORYML training data, this figure shows the top 20 most important features based on the normalized total average training loss reduction.

by the same function while using different parameters or are multiple values (*e.g.*, different coefficients for FFT transform) generated by the same function. Therefore, when it is not possible to give an explicit order of top features, we believe these features contain important fworkload characteristics.

*2) Interpretation:* Time-series features are less readily interpretable than classical observed fworkload features such as IOPS or read/write ratios. The most critical features in EMORYML and FIU are similar, while MSR features are distinct, likely due to different index identification. We describe and contextualize the eight most critical features below, and further feature descriptions can be found in Appendix VII-B. As mentioned in Section III-A, these features are chosen based on their criticality.

- *address_complexity* measures the complexity of the address series by calculating the root of the sum over the squared distance between the consecutive data points.

$$\sqrt{\sum_{i=0}^{n-2} (x_i - x_{i+1})^2}$$

A high feature value indicates that more random accesses and less sequential accesses are in the trace, which implies more concurrent fworkloads during that time window. We tested this hypothesis on our data and saw a clear correlation between

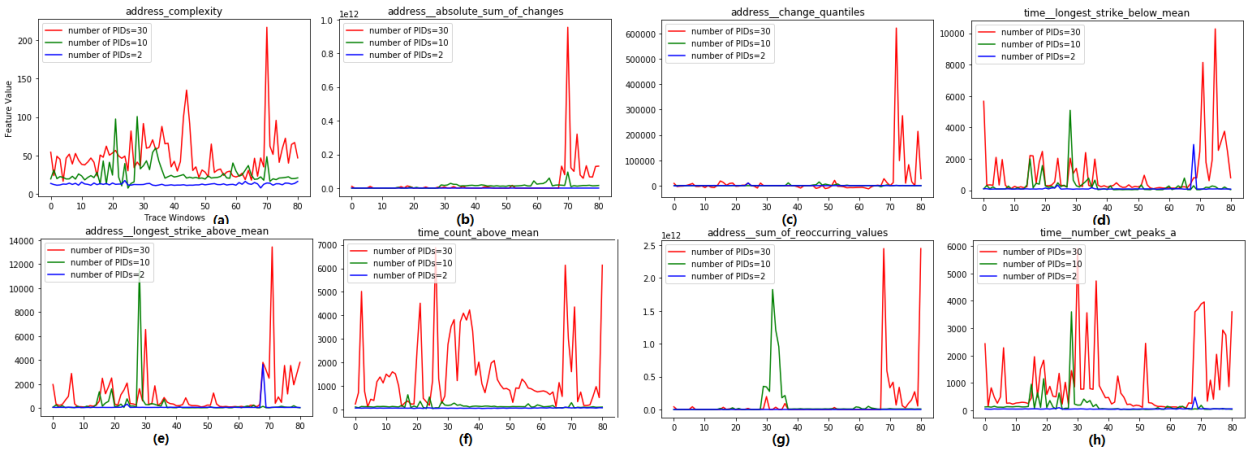



Fig. 4. Top features for classification. Features are compared between the different numbers (2, 10, 30) of fworkloads from FIU for 80 time windows ($x$ axis). Feature value is plotted on the $y$ axis. The strong correlations between feature value and high numbers of PIDs support the validity of our criticality measures for estimating the number of fworkloads in a sample.

the number of processes and address_complexity (Fig. 4a).

- *address_absolute_sum_of_changes* calculates the sum over the absolute value of the changes between consecutive data points in the address series in a certain time window:

$$\sum_{i=0,\ldots,n-2} |x_{i+1} - x_i|$$

. Similar to address_complexity, a high feature value indicates that more random accesses and less sequential accesses are in the trace, which implies more concurrent fworkloads during that time window. which indicates the (Fig. 4b).
- *address_change_quantiles* returns the average absolute consecutive changes of the address series identified between given quantiles. The defined quantiles set a threshed to constrain the random address change and remove the outlier, which works as a hedge for noisy random addresses. A high feature value means diverse consecutive changes exist in address series. Complex activities always suggest more concurrent fworkloads in that time window (Fig. 4c).
- *time_longest_strike_below_mean* returns the length of the longest sub-sequence that is smaller than the mean of time-series deltas. A high feature value corresponds to longer subtraces with high access density, which suggests, again, more concurrent fworkloads in that time window (Fig. 4d).
- *address_longest_strike_above_mean* returns the length of the longest sub-sequence of addresses that is larger than the mean address. A high feature value may indicate long sequential I/O subtraces are long or a group of related fworkloads with close access addresses. A trace with more concurrent fworkloads is more likely to generate higher feature value. (Fig. 4e).
- *time_count_above_mean* is the same metric applied to time delta sequentiality instead of address sequentiality. A high feature value means that more consecutive accesses do not request closely in time, which may suggest that they come from concurrent fworkloads (Fig. 4f).
- *address_sum_of_reoccurring_values* calculates the sum of values that recur in an address series. A high feature value means that more repeated addresses are present in the address series. Since one fworkload may repeatedly access some addresses, a high feature value may suggest more concurrent fworkloads in that window (Fig. 4g).
- *time_number_cwt_peaks_a* is a measure of the signal-to-noise (SN) ratios in a time-series. First, this feature smooths the delta-time series by convolving it with a Ricker wavelet [41], which is the second derivative of a Gaussian, and then it returning the number of peaks with SN ratios above a threshold. Many such peaks mean that many consecutively significant changes exist in the time-series. This instability indicates that there may be interfering operations in a storage system, which suggests that denser accesses and more concurrent fworkloads in that time window (Fig. 4h).

### B. Model Evaluation

After feature extraction and filtering, we implement the classification models using the model designs described in section III-B.

Since the FIU dataset has multiple domain-specific sub-traces, it is well suited for the generalized model that tests by window. EMORYML and MSR datasets are considerably smaller, so CENSUS under the ID model is a better fit since it tests at a file-level granularity.

To avoid comparing against a straw man, we calculate a baseline classification to compare against by randomly generating labels based on the known distribution of labels in the training set, which is by far the fairest guess of fworkload numbers. By comparing our accuracy values against this baseline, we show that CENSUS does more than reconstructing a distribution; it provides new insight into workload composition and the features that are most linked to workload differentiation.

For performance evaluation, we use true accuracy, approximate accuracy ($x$-accuracy), and mean absolute percentage error (MAPE) as metrics. The $x$-accuracy is computed by treating instances with prediction error within $x$ of the actual value for $x \in \mathbb{Z}^+$ as correct. MAPE is defined as

$$\mathrm{M} = 100 \times \frac{1}{n} \sum_{t=1}^{n} \left| \frac{A_t - F_t}{A_t} \right|$$

, where $A_t$ is the actual value and $F_t$ is the forecast value. MAPE measures the size of the prediction error in the classification model, allowing us to identify instances where CENSUS returns the wrong but "close" number of fworkloads. While $x$-accuracy gives us a direct sense of the relaxing effect of accuracy, MAPE shows the acceptable error range. We could evaluate the result better by combining these two metrics. Using MAPE in addition to $x$-accuracy allows us to scale our model as the number of fworkloads increases, capturing the intuition that predicting 4 fworkloads when there are 2 is much worse than predicting 12 fworkloads when there ought to be 10. For example, given a series of ground truth labels like *10, 20, 25, 30*,. . ., a predicted label series *11, 19, 26, 28,*. . ., has an impressively low MAPE of 6.416%, although the accuracy is 0.

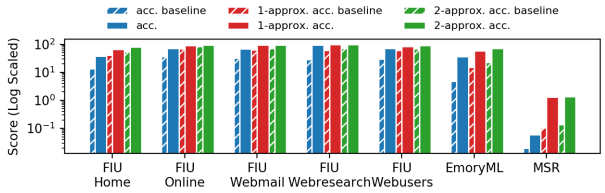
*1) Generalized Model:* To guarantee the framework has high generalization ability, we trained this model to capture fast changing fworkload characteristics, as described in section III-B3. In the training process, we used all available data from multiple domains of FIU, EMORYML, and MSR, which are interleaved in their raw form. We then segment the data over various time windows (10, 30, 60, 120, 240, and 360 minutes) to increase the training space and improve the adaptability.

The log-scaled accuracy and MAPE scores of LightGBM classifier are shown in Figure 6(a) and Figure 6(b).

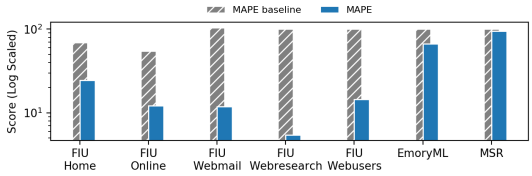
Since we synthetically combine MSR data before counting how many fworkloads are interleaved, we were worried that including it in our aggregated training set would reduce the applicability of the generalized model to real-world problems, particularly given the significant difference in the distribution for its underlying fworkloads compared to the other datasets (Figure 7). We tested this theory by removing MSR from the generalized model, and we see a small but appreciable improvement both in MAPE and accuracy (Figure 5). Especially for EMORYML and FIU Home, the average accuracy of them increase by 20% and 8% respectively. We see that the generalized Model presents stable, satisfactory performance while testing on various traces.

Even with MSR included, CENSUS with LightGBM (Figures 6(a) and 6(b)) achieves 23% higher accuracy and 63% lower MAPE than baseline on average. We see a similar result when using RF classifiers (Figures 6(c) and 6(d)): CENSUS with RF achieves 26% higher accuracy and 61% lower MAPE than baseline on average. This performance shows that the generalized model has high stability and generalizability across classifiers and datasets.

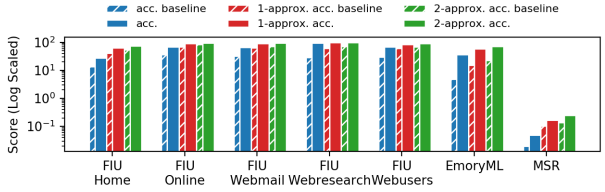
*2) ID Model:* Since the MSR and EMORYML datasets have fewer records than FIU, we trained three models respectively to show the impact of training data distribution. These three models are trained on different datasets: MSR, EMORYML and FIU-Home2 sub-trace, which has a relatively similar class distribution with EMORYML as shown in Figure 7. For model evaluation, 20 days data of Home2 in FIU are trained, and the average of RF and LightGBM classification performances is shown in Figure 8(a) and Figure 8(b). We can see that the MSR shows a quite different distribution with either EMORYML or FIU. This may answer the question of why CENSUS shows such bad result in the generalized model, but performs better when the training data fit its distribution. The selected data volumes are tested with both RF and LightGBM classifiers, and the detailed average results are shown in Figure 8(a) and Figure 8(b). The ID Model achieves 25% higher accuracy and 52% lower MAPE than baseline on average. This performance shows that the ID Model possesses high precision while fitting data from various domains.



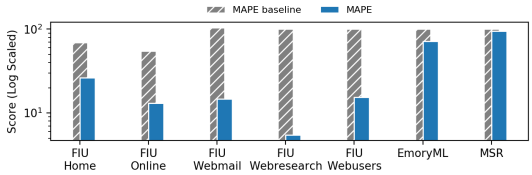

(a) LightGBM based Generalized-MSR Accuracy vs. Baseline




(b) LightGBM based Generalized-MSR MAPE vs. Baseline




(c) RF based Generalized-MSR Accuracy vs. Baseline




(d) RF based Generalized-MSR MAPE vs. Baseline

Fig. 5. The LightGBM based generalized-MSR model achieves 27% higher accuracy and 57% lower MAPE than the fair guess baseline on average. The RF based generalized-MSR model achieves 26% higher accuracy and 55% lower MAPE than the fair guess baseline on average. The average raw accuracy score of all experiments is 53%, and the highest is 97% (Web Research). The average $x$-accuracy score is 73%. The MAPE is 56% lower than the fair guess baseline on average. The average MAPE score of all experiments is 34%, and the lowest is 5.4% (Web Research).

### C. Application: Separating fworkloads

To illustrate how important knowing the number of fworkloads is for fworkload separation, we demonstrate the effect of having a good estimate for ground truth on the current state-of-the-art fworkload separation approach [10]. Blind source separation (BSS) techniques like Joint Approximation Diagonalization of Eigenmatrices (JADE) [8] isolate individual non-Gaussian signals within a shared data pipeline by selecting a set of candidate signals and then minimizing the mutual information across said signals. This is accomplished by measuring signal kurtosis and using non-Gaussianity as a proxy for independence [21].

Blind Source Separation (BSS) algorithms are designed to calculate a linear mixing matrix to separate sources. Previous work has shown that I/O contention resembles a linear mixture [10]. The rank of the mixing matrix is a necessary input to the system, and where the predicted number of fworkloads comes in.

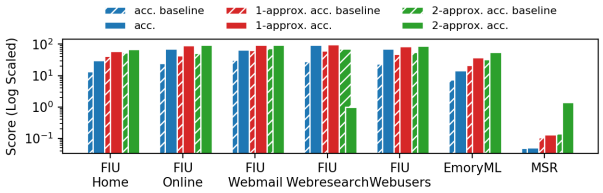
Establishing this rank using allows us to separate the signals using the JADE BSS algorithm.

We measure the recovery accuracy using the mean square error (MSE) between the recovered signals and the true source signals, defined as $\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \tilde{y}_i)^2$, where $y_i$ is the true source signal and $\tilde{y}_i$ is the recovered signal. In Figure 9, we can see that the estimate for the number of fworkloads provided by CENSUS decreases the average MSE compared to the fair guess from 0.5869 to 0.5137. For some extreme cases, the MSE even decrease to 5% compared to the fair guess. This demonstrates the value of predicting the number of fworkloads when separating interleaved fworkloads.

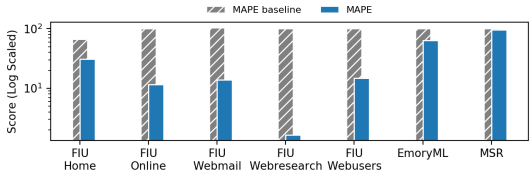
## V. ANALYSIS AND DISCUSSION

While predicting an exact number of fworkloads is a hard task, CENSUS maintains an average of 90% 2-approximate accuracy and exceeds 95% actual accuracy on easily separable data such as web traces. The Generalized model 5(a) has much higher accuracy for Web than Home traces. We attribute this performance difference to the data distribution. According to the instance distribution Figure 7, sub-traces in FIU "Home" directory have higher variance. In web traces (*e.g.*, online trace), however, we can see the variances are relatively lower, which results in better performance.

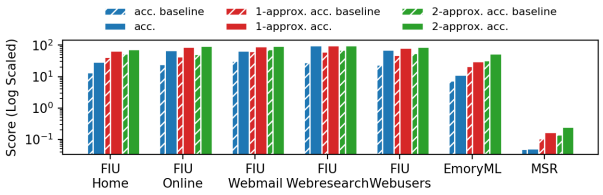
We observe that distribution of the number of fworkloads for EMORYML is more randomly distributed, resembles Home2, which indicates more I/O bursts and I/O intensity changes existing compared to the rest subtraces of FIU. These are the factors that lead to the relatively lower classification performance of EMORYML and Home2. Among Home2 and EMORYML, CENSUS performs better for Home2, since the EMORYML data



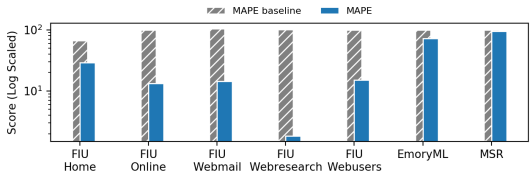

(a) LightGBM based Generalized Accuracy vs. Baseline




(b) LightGBM based Generalized MAPE vs. Baseline




(c) RF based Generalized Accuracy vs. Baseline




(d) RF based Generalized MAPE vs. Baseline

Fig. 6. The LightGBM based Generalized model achieves 23% higher accuracy and 63% lower MAPE than the fair guess baseline on average. The RF based Generalized model achieves 26% higher accuracy and 61% lower MAPE than the fair guess baseline on average. The average raw accuracy score of all experiments is 49%, and the highest is 96% (Web Research). The average $x$-accuracy score is 65%. The MAPE is 62% lower than the fair guess baseline on average. The average MAPE score of all experiments is 34%, and the lowest is 5.6% (Web Research).

distribution is uniformed on a wider range of value while Home2 is more centered around two main value. Therefore, it is harder for CENSUS to predict accurately on EMORYML data than Home2.

MSR is synthetically combined, and the apparently different distribution of it with other traces shown in Figure 7 contributes to the worse classification performance of the Generalized model compared to other traces. However, while testing MSR by ID model, it achieves 10% higher raw accuracy and 20% lower MAPE over the fair guess baseline performance. Compared to the baseline, our models show up to 27% accuracy increase and 57% MAPE decrease, which indicates that our models are robust against data of different distribution patterns and I/O intensity.

In contrast, the Web Research baseline MAPE is almost 100% because the baseline prediction errors are almost equal to the ground truth. Since the baseline results are generated based on training labels distribution, the high baseline MAPE illustrates that the training distribution is quite different from the test data, which indicates our model performs much better.

### A. Workload Characteristics

To our surprise, the fworkload features related to time delta, which corresponds to interval time in the trace, do not significantly impact classification accuracy. As we can see in figure 3, only 30% of top features are related to time delta. The final result only improves about 1% when we add the time delta features. We hypothesize this is because the address may carry more effective information than time, and throughput change is not critical for determining the number of fworkloads. However, we still keep the time delta features in the model to achieve better accuracy.

### B. Classification

LightGBM and RF classifiers have similar classification accuracy regardless of the training method used (Figure 5 and Figure 6). This indicates that the concern that LightGBM would easily overfit to small amounts of data is unfounded. The authors recommend that, even though both classify similarly, CENSUS should be run with a LightGBM classifier as they are simpler to build [24] and run faster than RF classifiers because of their leaf-wise growth strategy and feature histogram optimization [24]. Figure 10 shows the comparison of training time between RF and LightGBM based Generalized models. While running unoptimized code on a local 4-node cluster, the average training time of LightGBM classifiers for 42,000 samples with 1578 features was 1718 seconds, which is almost half of the average time of 3336 seconds of our RF classifiers.

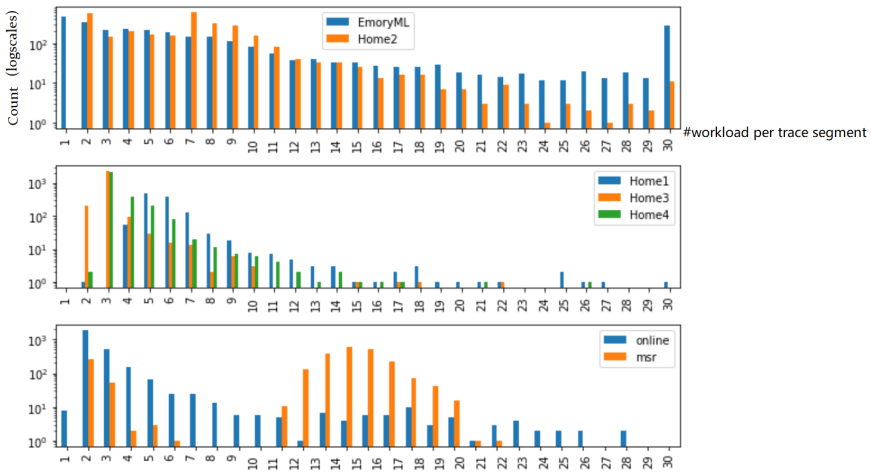
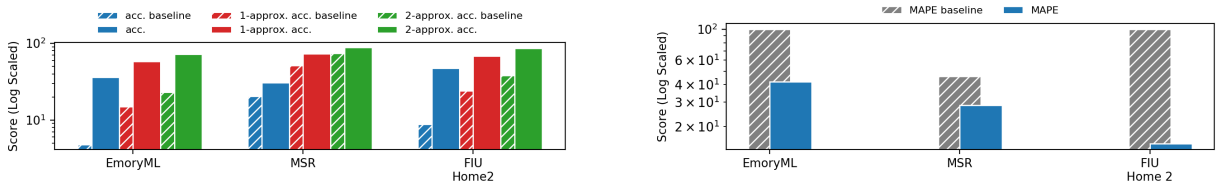



Fig. 7. #Workload distribution per trace segment (x axis) counts (y axis) comparison between EMORYML, sub-traces in FIU and MSR. #Workload per trace segment of Home3 and Home4 sub-traces are centered around 3 and 4. Compared to Home3, EMORYML data has a more even #workload distribution per trace segment. MSR shows a different peak from the other traces. All #workload are sorted by prevalence and numbered; #workload do not correlate between traces.




(a) ID classification accuracy

(b) ID classification MAPE

Fig. 8. The ID model achieves 25% higher accuracy and 52% lower MAPE than the fair guess baseline on average. The average raw accuracy score of all experiments is 37%, and the highest is 47% (FIU Home2). The average $x$-accuracy score is 74%. The MAPE is 52% lower than fair guess baseline on average. The average MAPE score of all experiments is 29%, and the lowest is 16% (FIU Home2).

While our prototype for CENSUS is designed to be run offline, we are working to lower runtimes through optimizing code and shifting development from Python to a lower level language. Moreover, LightGBM can be accelerated by modern processing hardware optimized for ML workloads [48]. Both LightGBM and RF are much faster than deep learning models, which took both our team and others solving similar problems multiple hours to train [46].

### C. Future Work

While CENSUS demonstrates high accuracy of prediction of fworkload numbers within each domain, we expect that large-scale sets of richer training data, such as co-located VMs, would allow us to train an even more robust classifier. This sort of dataset would allow us to do a broad domain-specific analysis and potentially learn an underlying representation of "groups" of fworkloads with a high likelihood of co-occurrence. We encourage industry colleagues to reproduce our work on internal data and share the distributions of fworkloads in production systems with more complex interleaving patterns. We expect that the numbers will be smaller than anticipated and could renew the conversation about custom tuning storage for major fworkload classes. CENSUS, through the parallelism and lightweight memory footprint of LightGBM [30], should have minimal impact on production systems.

The penultimate future goal of this work is specialized tuning for functionally distinct yet stable storage system access patterns in a shared system. Research has shown that fully isolating fworkloads leads to inefficient resource usage while impacting Quality-of-Service [27], [47], [2], [39], but managing multi-workloads in a mod-

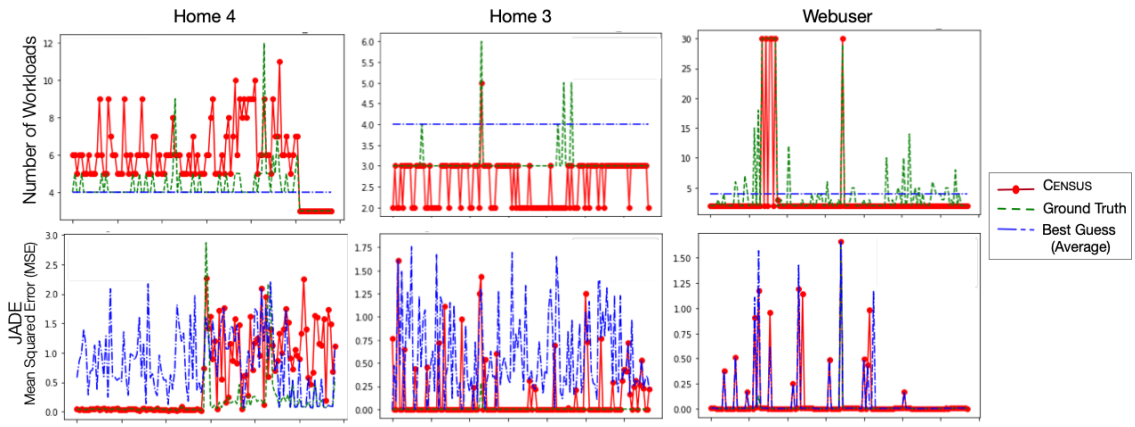



Fig. 9. Each window represents one day of the respective FIU trace, with time on the $x$ axis. The top row shows the number of fworkloads predicted by CENSUS plotted against ground truth and daily average value of ground truth. This daily average, which is dependent on the ground truth, is used to compare CENSUS against instead of a random fworkload number to address the claim that clients may have some idea of their fworkload mix, but they lack granularity. The bottom row shows that the MSE of the JADE method of fworkload separation is no worse, and often better, using the number of fworkloads returned by CENSUS.

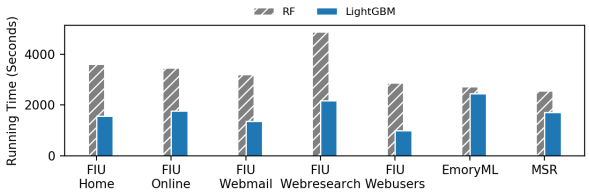



Fig. 10. RF based Generalized Model running time vs. LightGBM

ern shared storage cache is famously difficult [15]. Multi-workload cache management would be much simpler if the fworkloads were both few and well characterized. We are also now recurrently training the model when we detect the fworkloads have changed. Fworkload types that are too new to have standard practices for provisioning could benefit from our identification of fworkloads.

## VI. CONCLUSIONS

In this paper, we introduced CENSUS, a time-series based feature extraction and trace analysis tool to calculate the number of functionally distinct fworkloads in a multi-workload setting and separate these fworkloads, with 90% average 2-approximate accuracy for detection, and improve fworkload separation accuracy by deceasing the mean square error to as low as 5% compared to the fair guess according to the daily average.

With a powerful estimator for the number of extant interleaved fworkloads in hand, we have set the stage for building a classifier that can, given interleaved fworkloads, automatically distinguish the fworkloads and determine the fworkload type. Once we understand the nature of the fworkload by function, we can tune the cache and storage system by fworkload usage, improving resource provisioning and system performance, which in turn can improve storage economics.

We also derived critical extracted features, which are, crucially, highly dissimilar to features presented by prior fworkload analyses, opening the field to insights derivable from formerly overlooked metrics. Finally, we hope this study motivates further rigorous time-series analyses of systems fworkloads.

## AVAILABILITY

All CENSUS code is licensed under Creative Commons and publicly available at https://github.com/meditates/CENSUS. The EMORYML dataset will be submitted to the public SNIA IOTTA trace repository upon publication.

## ACKNOWLEDGEMENTS

We are grateful to our shepherd George Amvrosiadis and the anonymous reviewers for their constructive comments. We would like to thank members of the Emory SimBioSys Lab, particularly Nosayba El-Sayed, for valuable feedback, as well as Laney Graduate School and the National Science Foundation for funding this work.

## REFERENCES


[1] C. L. Abad, N. Roberts, Y. Lu, and R. H. Campbell. A storage-centric analysis of mapreduce workloads: File popularity, temporal locality and arrival patterns. In *2012 IEEE International Symposium on Workload Characterization (IISWC)*, pages 100–109. IEEE, 2012.

[2] A. Adegboyega. Time-series models for cloud workload prediction: A comparison. In *2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, pages 298–307. IEEE, 2017.

[3] I. Ahmad. Easy and efficient disk i/o workload characterization in vmware esx server. In *2007 IEEE 10th International Symposium on Workload Characterization*, pages 149–158. IEEE, 2007.

[4] J. Basak, K. Wadhwani, and K. Voruganti. Storage workload identification. *ACM Transactions on Storage (TOS)*, 12(3):14, 2016.

[5] A. D. Brunelle. Blktrace user guide.

[6] A. Busch, Q. Noorshams, S. Kounev, A. Koziolek, R. Reussner, and E. Amrehn. Automated workload characterization for i/o performance analysis in virtualized environments. In *Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering*, pages 265–276, 2015.

[7] M. C. Calzarossa, L. Massari, and D. Tessera. Workload characterization: A survey revisited. *ACM Computing Surveys (CSUR)*, 48(3):48, 2016.

[8] J.-F. Cardoso. High-order contrasts for independent component analysis. *Neural computation*, 11(1):157–192, 1999.

[9] S. Chakraborty, R. Tomsett, R. Raghavendra, D. Harborne, M. Alzantot, F. Cerutti, M. Srivastava, A. Preece, S. Julier, R. M. Rao, et al. Interpretability of deep learning models: a survey of results. In *2017 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computed, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI)*, pages 1–6. IEEE, 2017.

[10] S. Chen and A. Wildani. Chasing the signal: Statistically separating multi-tenant I/O workloads. *NeurIPS2018*, 2018.

[11] T. Chen and C. Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794. ACM, 2016.

[12] M. Chesire, A. Wolman, G. M. Voelker, and H. M. Levy. Measurement and analysis of a streaming media workload. In *USITS*, volume 1, pages 1–1, 2001.

[13] J. Choi, S. H. Noh, S. L. Min, and Y. Cho. Towards application/file-level characterization of block references: a case for fine-grained buffer management. In *ACM SIGMETRICS Performance Evaluation Review*, volume 28, pages 286–295. ACM, 2000.

[14] M. Christ, A. W. Kempa-Liehr, and M. Feindt. Distributed and parallel time series feature extraction for industrial big data applications. *CoRR*, abs/1610.07717, 2016.

[15] A. Cidon, D. Rushton, S. M. Rumble, and R. Stutsman. Memshare: a dynamic multi-tenant key-value cache. In *2017 {USENIX} Annual Technical Conference ({USENIX}{ATC} 17)*, pages 321–334, 2017.

[16] M. Ester, H.-P. Kriegel, J. Sander, X. Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Kdd*, volume 96, pages 226–231, 1996.

[17] R. Gracia-Tinedo, J. Sampé, E. Zamora, M. Sánchez-Artigas, P. García-López, Y. Moatti, and E. Rom. Crystal: software-defined storage for multi-tenant object stores. In *FAST17*, pages 243–256, 2017.

[18] J. D. Hamilton. *Time series analysis*, volume 2. Princeton university press, Princeton, NJ, 1994.

[19] N. R. Herbst. *Workload classification and forecasting*. PhD thesis, IBM, 2012.

[20] T. K. Ho. Random decision forests. In *Proceedings of 3rd international conference on document analysis and recognition*, volume 1, pages 278–282. IEEE, 1995.

[21] A. Hyvärinen, J. Karhunen, and E. Oja. *Independent component analysis*, volume 46. John Wiley & Sons, 2004.

[22] Z. Jia, L. Wang, J. Zhan, L. Zhang, and C. Luo. Characterizing data analysis workloads in data centers. In *2013 IEEE International Symposium on Workload Characterization (IISWC)*, pages 66–76. IEEE, 2013.

[23] S. Kavalanekar, B. Worthington, Q. Zhang, and V. Sharda. Characterization of storage workload traces from production windows servers. In *2008 IEEE International Symposium on Workload Characterization*, pages 119–128. IEEE, 2008.

[24] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu. Lightgbm: A highly efficient gradient boosting decision tree. In *Advances in Neural Information Processing Systems*, pages 3146–3154, 2017.

[25] K. Keeton, G. Alvarez, E. Riedel, and M. Uysal. Characterizing i/o-intensive workload sequentiality on modern disk arrays. In *Proceedings of the 4th Workshop on Computer Architecture Evaluation using Commercial Workloads (CAECW-01)*, 2001.

[26] R. Koller and R. Rangaswami. I/o deduplication: Utilizing content similarity to improve i/o performance. *ACM Transactions on Storage (TOS)*, 6(3):13, 2010.

[27] W. Lang, S. Shankar, J. M. Patel, and A. Kalhan. Towards multi-tenant performance slos. *IEEE Transactions on Knowledge and Data Engineering*, 26(6):1447–1463, 2013.

[28] X. Li, B. Dong, L. Xiao, L. Ruan, and D. Liu. Hccache: a hybrid client-side cache management scheme for i/o-intensive workloads in network-based file systems. In *2012 13th International Conference on Parallel and Distributed Computing, Applications and Technologies*, pages 467–473. IEEE, 2012.

[29] Y. Liu, R. Gunasekaran, X. Ma, and S. S. Vazhkudai. Automatic identification of application i/o signatures from noisy server-side traces. In *FAST14*, pages 213–228, 2014.

[30] P. Mandot. What is lightgbm, how to implement it? how to fine tune the parameters?, 2017.

[31] M. Müller. Dynamic time warping. *Information retrieval for music and motion*, pages 69–84, 2007.

[32] D. Narayanan, A. Donnelly, and A. Rowstron. Write off-loading: Practical power management for enterprise storage. *ACM Transactions on Storage (TOS)*, 4(3):10, 2008.

[33] J. S. Oh, K. S. Choi, J. R. Kwon, and S. H. Lee. Finding the near workload type between tpc-c and tpc-w environments. In *2008 International Conference on Convergence and Hybrid Information Technology*, pages 334–337. IEEE, 2008.

[34] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[35] M. A. Rodriguez and R. Buyya. Scheduling dynamic workloads in multi-tenant scientific workflow as a service platforms. *Future Generation Computer Systems*, 79:739–750, 2018.

[36] S. Sakr and M. Gaber. *Large scale and big data: Processing and management*. Auerbach Publications, 2014.

[37] B. Seo, S. Kang, J. Choi, J. Cha, Y. Won, and S. Yoon. Io workload characterization revisited: A data-mining approach. *IEEE Trans. Comput.*, 63(12):3026–3038, Dec. 2014.

[38] C. Vazquez, R. Krishnan, and E. John. Time series forecasting of cloud data center workloads for dynamic resource provisioning. *JoWUA*, 6(3):87–110, 2015.

[39] M. Wachs, M. Abd-El-Malek, E. Thereska, and G. R. Ganger. Argon: Performance insulation for shared storage servers. In *FAST07*, 2007.

[40] X. Wang, K. Smith, and R. Hyndman. Characteristic-based clustering for time series data. *Data Mining and Knowledge Discovery*, 13(3):335–364, 2006.

[41] Y. Wang. Frequencies of the ricker wavelet. *Geophysics*, 80(2):A31–A37, 2015.

[42] A. Wildani and I. F. Adams. A case for rigorous workload classification. In *Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS), 2015 IEEE 23rd International Symposium on*, pages 146–149. IEEE, 2015.

[43] W. Xiao, R. Bhardwaj, R. Ramjee, M. Sivathanu, N. Kwatra, Z. Han, P. Patel, X. Peng, H. Zhao, Q. Zhang, et al. Gandiva: Introspective cluster scheduling for deep learning. In *13th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 18)*, pages 595–610, 2018.

[44] F. Yan and E. Smirni. Workload interleaving with performance guarantees in data centers. In *NOMS 2016-2016 IEEE/IFIP Network Operations and Management Symposium*, pages 967–972. IEEE, 2016.

[45] W. Yin, K. Kann, M. Yu, and H. Schütze. Comparative study of cnn and rnn for natural language processing. *arXiv preprint arXiv:1702.01923*, 2017.

[46] Q. Yu, C. Wang, X. Ma, X. Li, and X. Zhou. A deep learning prediction process accelerator based fpga. In *2015 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, pages 1159–1162. IEEE, 2015.

[47] J. Zeng and B. Plale. Workload-aware resource reservation for multi-tenant nosql. In *2015 IEEE International Conference on Cluster Computing*, pages 32–41. IEEE, 2015.

[48] H. Zhang, S. Si, and C.-J. Hsieh. Gpu-acceleration for large-scale tree boosting, 2017.

[49] C. Zhu, F. Wang, and B. Hou. Bpp: A realtime block access pattern mining scheme for i/o prediction. In *Proceedings of the 48th International Conference on Parallel Processing*, ICPP 2019, New York, NY, USA, 2019. Association for Computing Machinery.

## VII. Appendix

### A. Hyperparameters

TABLE I
ADDITIONAL LIGHTGBM HYPERPARAMETERS SETTING

| Name | value | Description |
|---|---|---|
| boosting | gbdt | Gradient Boosting Decision Tree |
| num_class | 31 | # of the possible classes |
| learning rate | 0.01 | shrinkage rate |
| feature_fraction | 0.8 | feature sub-sampling percentage |
| bagging_fraction | 0.8 | bagging percentage |
| max_depth | 6 | max tree depth |
| num_leaves | 30 | max # of leaves per tree |
| num_trees | 557 | # of boosting iterations |
| reg_alpha | 0.6 | regularization l1 |
| reg_lambda | 0.6 | regularization l2 |

TABLE II
ADDITIONAL RF HYPERPARAMETERS

| Name | value | Description |
|---|---|---|
| n_estimators | 482 | # of trees |
| min_samples_split | 15 | min. # of samples to split internal node |
| min_samples_leaf | 30 | min. # of samples to leaf node |
| max_features | 0.8 | ratio of features considered at each split |
| max_depth | 5 | max. tree depth |
| max_leaf_nodes | 11 | max # of leaves per tree |
| bootstrap | True | bootstrap samples when building trees |
| oob_score | True | Use out-of-bag samples for generalization accuracy |

### B. Additional Extracted Features

- *address_count_above_mean* calculates the number of values in the address series that higher than the mean. A high feature value may indicate more accesses in the trace, implying more concurrent fworkloads in that time window.
- *address_abs_energy* calculates the absolute energy, which is the sum over the squared values of the address series. The energy formula is

$$E = \sum_{i=1,\ldots,n} x_i^2$$

  For example, the average absolute energy value is 4.119117e+20 when the number of running processes is 30, however, when the class number is 1, the average energy value changes to 1.312363e+19. A high feature value indicates more accesses in the trace, suggesting more concurrent fworkloads in that time window.
- *address_fft_coefficient_a* calculates the fast Fourier transformation coefficients of the one-dimensional discrete Fourier transform for address series.
- *address_ar_coefficient_a* returns the coefficients of an auto-regressive process for the address series given parameter a.
- *address_binned_entropy* calculates the entropy of the equal binned address series.
- *time_quantile_a* calculates the "a" ($0 < a < 1$) quantile of the delta-time series, and returns the value of delta-time which is greater than "a" of the ordered delta-time series. A high feature value means the eligible delta-time is big, indicating sparse accesses and less concurrent fworkloads.
- *address_number_peaks* returns the numbers of peaks in a given range of left and right neighbors of the address subseries. A high feature value means more eligible peaks in the address series, which suggests more random accesses in that time window. For example, the address series is [3, 0, 1, 4, 2, 3, 13, 2, 3, 4, 5, 2, 3, 13], if we set the range to be 3, then the feature value is 3 because we have three peaks ( 4, 13, 5).
- *address_standard_deviation* returns the standard deviation of the address series. It measures the amount of variation of address series. A high feature value means that the address series are more spread out and unstable, which represents more concurrent fworkloads in that time window.
- *delta_time_number_cwt_peaks_n_5* detects the number of different peaks in the delta-time series. First, it smooths the delta-time series by convolving it with a Ricker wavelet for supplied widths ranging from 1 to 5. Then, this feature returns the number of peaks with high Signal-to-Noise-Ratio(SNR), and enough width scales accepted. In this experiment, we find this Ricker wavelet is useful for block I/O analysis by smoothing the delta-time series. A high feature value means more peaks, which indicates the accesses of this trace vary and have less routine, suggesting more concurrent fworkloads.