Scaling Disk Failure Prediction via Multi-Source Stream Mining

Shujie Han[1,2], Zirui Ou[2], Qun Huang[2], and Patrick P. C. Lee[3]
[1]Northwestern Polytechnical University [2]Peking University [3]The Chinese University of Hong Kong

***Abstract*—Traditional disk failure prediction approaches struggle to scale with data growth, as they treat data as a whole collection to obtain the global data view for preprocessing and training. Existing distributed machine learning and stream mining systems are designed to scale data processing, particularly for training. However, scaling disk failure prediction faces challenges in the scalability of preprocessing, including additional data movements from data collection to training, data inflation during preprocessing, and multiple-to-multiple data allocation. To address these challenges, we present SCALEDFP, a general framework for scaling disk failure prediction via multi-source stream mining based on three techniques: near-data preprocessing, random downsampling, and training data allocation. SCALEDFP scales disk failure prediction with the number of data sources. It achieves significant throughput gains of preprocessing and training with comparable prediction accuracy against a state-of-the-art disk failure prediction approach that collects data in a centralized place.**

I. INTRODUCTION

Prevalent disk failures challenge the reliability of modern data centers. To tolerate disk failures, modern data centers tend to adopt *proactive* approaches by predicting imminent disk failures [9], [27], [48]. Such proactive approaches apply *machine learning* (ML) algorithms to identify soon-to-fail disks and replace them in advance. Conventionally, disk failure prediction approaches [9], [24], [31], [43], [44], [50] preprocess disk logs to extract numerical values that represent disk performance and reliability statistics. Utilizing such numerical values, including the known status of disks (i.e., healthy or failed) as *training data*, they train a prediction model and predict the disk status in the near future (e.g., remaining healthy or becoming failed).

Conventional disk failure prediction approaches, either *offline learning* [9], [24], [29], [31], [43], [44], [50] or *online learning* [17], [41], are difficult to scale with the data volume. Offline learning approaches assume the availability of all data in advance, while online learning approaches regard data as a continuous data stream with partial availability. By aggregating data into a monitoring system (e.g., distributed storage systems [12] or time-series databases [37]), both offline and online learning approaches consider the disk set as a whole collection. This simplifies the process of obtaining the global view of data (e.g., global data distribution) to identify failure patterns for high prediction accuracy. However, attaining a global view of data needs extensive hardware resources as more data is used. For example, Azure deploys hundreds of millions of disks [44], where the status data of each disk is recorded hourly [29], [44]. Each record is identified with more than 400 statistical attributes [44]. Given that the status data is represented as a 64-bit floating-point value, loading all disk logs in one day to obtain the global view would require approximately 6.98 TB of RAM. Thus, scaling disk failure prediction is essential for large-scale disk deployment.

Existing distributed ML systems [6], [22], [34] and distributed stream processing systems [11], [20], [39], [45] are adopted for the scalability of data processing, especially in training. However, scaling disk failure prediction faces several challenges in the scalability of *preprocessing* before training. First, the lack of integration between data collection and training involves additional data transfers. Most existing distributed ML and stream processing systems do not incorporate integrated data collection mechanisms for disk failure prediction, while monitoring systems only provide general-purpose data analytics without fully integrated ML-based prediction models [37]. This results in two types of data transfers: (i) data collection from source locations to a monitoring system for preparation and (ii) data allocation from the monitoring system to external ML-based platforms for training. The second challenge is data inflation in preprocessing. Traditional disk failure prediction approaches often augment training data during preprocessing by generating more statistical attributes [44], [48] and buffering a long period of historical data for online labeling [17], [41]. Such data inflation incurs larger network overhead of allocating the training data than distributing original disk logs for preprocessing. The third challenge pertains to multiple-to-multiple data allocation. Suppose that multiple sources originate the training data which is fed into a distributed prediction model (e.g., an ensemble of *base learners*). The absence of a global data view and the heterogeneity of data sources (e.g., variations in failure occurrences and disk counts) necessitate careful decision-making regarding the allocation of sources to train specific base learners. Allocating a data source independently to train a particular base learner may hinder the development of an accurate prediction model due to insufficient data. Conversely, sharing data sources across base learners can lead to significant data redundancy and overfitting in the training process.

To address the challenges, we propose SCALEDFP, a general framework for scaling disk failure prediction, with specific emphasis on the scalability of preprocessing. Specifically, SCALEDFP scales disk failure prediction to a *multi-source* stream mining problem. By tracing back disk logs originating from multiple data sources, it views disk logs as multiple evolving streams of time-series data. The core design includes three techniques: (i) *near-data preprocessing*, which preprocesses original disk logs near their sources to minimize the data transmission to a remote preprocessing facility, (ii) *random downsampling*, which keeps all positive samples and relevant negative samples for training to mitigate data inflation in preprocessing, and (iii) *training data allocation*,

which ensures the proper assignment of training data from multiple data sources to an ensemble of base learners in a prediction model and optimizes the data transmission.

To summarize, we make the following contributions:

- We present SCALEDFP, a general framework for scaling disk failure prediction through multi-source stream mining. SCALEDFP focuses on the scalability of preprocessing before training, by performing near-data preprocessing, random downsampling, and training data allocation.
- We implement a complete prototype of SCALEDFP based on StreamDFP [17], a state-of-the-art stream mining framework for adaptive disk failure prediction that is executed on a single machine.
- We evaluate SCALEDFP on six public disk models from two independent production environments, i.e., Backblaze and Alibaba, including hard disk drives (HDDs) and solid-state drives (SSDs). To demonstrate the scalability of preprocessing, we evaluate the system performance of SCALEDFP on Alibaba Cloud [1] with up to 130 instances. It achieves the throughput gains of preprocessing and training by up to 41.6× and 9.1×, respectively, while maintaining a comparable prediction accuracy compared to StreamDFP [17].

Our SCALEDFP prototype is open-sourced at **https://github.com/shujiehan/ScaleDFP**.

II. Background and Motivation

We formulate the problem of scaling disk failure prediction and present the challenges of the scalability of preprocessing.

A. Scaling Disk Failure Prediction

Traditional disk failure prediction. Traditional disk failure approaches can be divided into two categories, i.e., *offline learning* and *online learning*. Offline learning approaches [9], [28], [31], [43], [48] assume that all data to use is collected and available in advance. They feed the available data into prediction models for training. In practice, data is collected continuously, such that the distribution of collected data is changing with time (called *concept drift* [14]). Concept drift causes prediction models to age and become inaccurate. Thus, to keep prediction models updated with newly collected data, offline learning approaches have to retrain prediction models periodically from scratch [41]. On the other hand, online learning approaches [17], [41] regard the collected data as a *single* continuous data stream, such that they obtain the global distribution of data and train prediction models incrementally with the updated data.

Problem formulation. Our goal is to scale out disk failure prediction by formulating it as a *multi-source* stream mining problem. We conceptualize disk logs as continuously collected data streams originating from multiple sources, with the total number of data sources denoted by n. Specifically, we consider a typical data center composed of multiple *racks*, each hosting several machines connected via a top-of-rack switch. Each machine is equipped with multiple disks. We regard each rack as a *data source* that generates a data stream consisting of disk logs from the disks attached to the rack. We decompose the scaling disk failure prediction process into three phases:

- **Preprocessing.** We extract the performance and reliability statistics from disk logs into numerical values (called *feature vectors*). For training data, we convert the disk status to numerical values (called *labels*) (see below), while for prediction data, the disk status is unknown. We call a pair of (feature vector, label) as a *labeled sample*. We allocate the training data from multiple streams into a prediction model.
- **Training.** We feed multiple streams of labeled samples simultaneously into the prediction model and update it continuously over time.
- **Prediction.** We use the prediction model to predict the status of a disk within the near future (e.g., the next 30 days).

We model disk failure prediction as a binary classification problem by assigning the values of 0 or 1 to labels to indicate healthy or failed disks, often referred to as *negative samples* and *positive samples*, respectively. The binary classification problem can also be extended to general prediction tasks such as regression and multi-class classification. In regression, continuous values are assigned to labels, whereas in multi-class classification, discrete values are assigned.

Scope of this work. To facilitate stream mining, we employ incremental learning algorithms as our prediction models, which perform real-time predictions over data streams and adapt incrementally based on incoming labeled samples. We consider state-of-the-art incremental learning algorithms grounded in decision tree techniques, particularly *ensemble learning* algorithms, which use single decision trees as *base learners*. Such algorithms combine predictions of multiple decision trees to enhance overall prediction accuracy. To demonstrate the generality of our approach for different algorithms, we focus on two variants of ensemble learning algorithms with concept-drift adaptation, namely (i) Bagging with Adaptive sliding window (BA) [8] and (ii) Adaptive Random Forests (ARF) [15]. Both algorithms support parallel training of base learners.

Although some prior studies [28], [38], [42], [47] also apply deep neural networks (DNNs) to disk failure prediction, we focus on decision tree-based incremental learning algorithms. First, decision tree-based algorithms achieve good performance on disk failure prediction [9], [41], [43], [48], and even a higher prediction accuracy than DNNs [48]. Second, DNNs in the realm of stream mining require buffering a significantly larger number of time-series samples for training, which results in higher memory overhead compared to other incremental learning algorithms [17]. We pose using DNNs for scaling disk failure prediction via multi-source stream mining as our future work.

B. Challenges in Scalability of Preprocessing

Scaling disk failure prediction faces the following challenges in the scalability of preprocessing before training.

(C1) Additional data movements from data collection to training. Recall from Section I that data is collected into a monitoring system. Most existing monitoring systems (e.g., time-series databases [37] or stream processing systems [11],

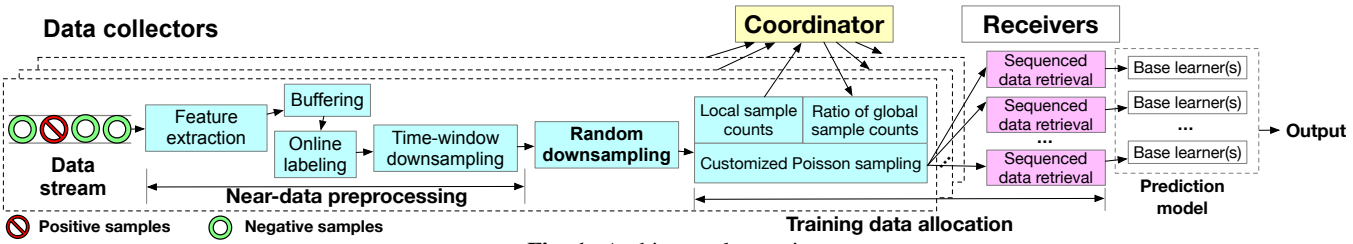


Fig. 1: Architectural overview.

[20], [39], [45]) provide statistical data analytics, but lack specialized capabilities necessary for disk failure prediction. Also, existing distributed ML systems [6], [22], [34] are primarily designed to optimize ML job execution, with limited considerations on data collection. Handling data collection and training separately can introduce additional overheads (e.g., data movements [36] and querying [37]) due to the use of external ML-based platforms.

(C2) Data inflation in preprocessing. Traditional preprocessing approaches inflate the original data via online labeling [17], [41] and feature generation [48], and hence intensify the demand for memory capacity. First, online labeling buffers recent samples for labeling soon-to-fail disks as positive samples to increase the sample count. For example, StreamDFP [17] buffers 30-day samples for online labeling, which results in 30× as many as millions of disks in a large deployment of production data centers (e.g., Azure [29]). Also, feature generation spawns statistical features based on original features, which augments the feature count. For instance, generating 438 additional features based on 26 features expands the training data 17.8× as much as the original data [48].

(C3) Multiple-to-multiple data allocation. To address scalability concerns, preprocessing is deployed in a cluster for parallel processing. We refer to allocating training data from multiple data sources to multiple base learners as *multiple-to-multiple data allocation*. However, state-of-the-art incremental learning algorithms (e.g., BA and ARF) are designed for a *single* continuous data stream and rely on *Poisson sampling* [10], [13] for allocating training data to base learners. Also, to address the *data imbalance* issue [9] in disk failure prediction, where the number of negative samples (i.e., healthy disks) vastly outnumbers that of positive samples (i.e., failed disks), previous studies [17], [41] customize Poisson sampling by configuring different hyper-parameters for negative and positive samples. As multiple data streams can be heterogeneous in terms of disk counts and failure occurrences (Figure 2 in Section III-B), which change over time, applying this allocation approach faces the following open issues: (i) determining the parameters of Poisson sampling for multiple data streams and (ii) adapting the parameters of Poisson sampling over time for multiple data streams.

III. Design

We present the design of ScaleDFP, a multi-source stream mining framework for scaling disk failure prediction. To address the challenges in Section II-B, our design goals are as follows:

- **Scalability.** ScaleDFP should be scalable for disk failure prediction with the number of data streams.
- **System throughput.** ScaleDFP should be efficient for processing data, especially for preprocessing and training. It is expected to improve the throughput of preprocessing and training.
- **Prediction accuracy.** ScaleDFP is expected to provide prediction accuracy guarantees by properly allocating samples from multiple sources for training.

A. Design and Architecture Overview

ScaleDFP is a framework for scaling disk failure prediction, focusing on data preprocessing before training, as shown in Figure 1. We highlight the key elements in the design:

- **Near-data preprocessing.** ScaleDFP considers that data is collected from multiple sources, instead of relying on a prepared dataset that is made available by a monitoring system. It performs immediate data preprocessing at each data source as the data is collected, minimizing the need for extensive transfers of raw data to a remote preprocessing location (**C1 addressed**).
- **Random downsampling.** To mitigate the data inflation issue, ScaleDFP avoids transferring unused negative samples to base learners after preprocessing. It performs random downsampling at each data source, ensuring that all positive samples and only relevant negative samples are allocated to base learners (**C2 addressed**).
- **Training data allocation.** Instead of aggregating all training data into a single stream for allocation, ScaleDFP performs training data allocation on each data source, leveraging lightweight global information. Each data source independently allocates training data to base learners based on the weights of samples to minimize data transmissions. Also, when aggregating multiple data streams into base learners, ScaleDFP maintains a chronological order of time-series samples (**C3 addressed**).

Architecture overview. ScaleDFP consists of three primary components: multiple *data collectors*, a *coordinator*, and *receivers*. Initially, data collectors, each of which is a machine randomly selected within a rack, gather disk logs from connected machines periodically (e.g., daily) (Section II-A), capitalizing on their substantial network bandwidth. These data collectors conduct *near-data preprocessing* on continuous streams of disk logs, including feature extraction, buffering, online labeling, and time-window downsampling (Section III-B). To alleviate data inflation, ScaleDFP performs random downsampling on each data collector to retain all

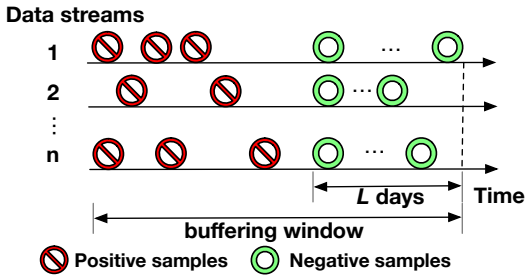


Fig. 2: Time-window downsampling. Here, we deduplicate multiple samples at the same timestamp for one data stream.

positive samples and relevant negative samples (Section III-C). ScaleDFP then allocates the training data from data collectors to base learners (Section III-D). Specifically, the coordinator aggregates local counts of positive and negative samples forwarded from data collectors to derive the global positive-to-negative sample ratio. With the lightweight nature of receiving sample counts, the coordinator avoids becoming a bottleneck. It subsequently disseminates the global positive-to-negative sample ratio to each data collector which leverages such global information to perform the customized Poisson sampling. Also, the data collectors transmit the training data to receivers, each of which maintains a chronological order of time-series samples for a subset of base learners via sequenced data retrieval. Finally, the prediction model conducts training and prediction and then outputs prediction results of disk failures in the near future (Section III-E).

Comparisons with StreamDFP [17]. StreamDFP, designed for single-machine execution, does not address the scalability challenges of preprocessing (Section II-B). It relies on a prepared dataset collected into a monitoring system and preprocesses data on a single stream without parallel capability. In addition, when preprocessing and training are deployed separately, StreamDFP transfers all training data, including many unused negative samples, leading to high network bandwidth consumption. It also computes the weight of Poisson sampling for each base learner on each sample, which incurs significant computational overhead. In contrast, ScaleDFP introduces several enhancements over StreamDFP. It performs near-data preprocessing on multiple data streams, enabling parallel data processing. Also, by implementing random downsampling, ScaleDFP reduces the amount of training data that needs to be allocated, saving network bandwidth and decreasing computational overhead by retaining only useful training data. Furthermore, ScaleDFP optimizes the data transmission process by allocating training data on data collectors in parallel and filtering out samples with zero weights for all base learners before transmission, thereby reducing the network load.

B. Near-Data Preprocessing

Workflow. Starting with the collection of raw data from data collectors, the preprocessing workflow follows the traditional approach in online learning, including feature extraction, buffering, online labeling, and time-window downsampling.

- **Feature extraction.** When dealing with a continuous stream of daily disk logs as input, ScaleDFP extracts disk logs as features for training and prediction (see more details of datasets in Section V). We use *all* features by default since it is difficult to select effective features without a global view of all data streams. We pose the feature selection under multiple data streams as our future work.
- **Buffering.** To label the samples on the fly, ScaleDFP borrows the idea of buffering [17]. It also uses a fixed-size sliding window (called the *buffering window*) (e.g., 30 days) to buffer recent samples for online labeling.
- **Online labeling.** ScaleDFP performs online labeling to alleviate data imbalance. Specifically, it performs online labeling in the buffering window to label soon-to-fail disks as positive samples. We refer to the number of extra labeled days before a failure happens as the *extra labeled days*. If a failure is found, ScaleDFP labels the samples within the extra labeled days (e.g., 20 days) before the failure occurrence as positive samples.
- **Time-window downsampling.** ScaleDFP performs time-window downsampling to further alleviate data imbalance. It selects all positive samples and the last few days of negative samples in the buffering window (Figure 2). The length of the time window for selecting negative samples is denoted by L and set as seven days by default.

Straw-man solutions. To tackle C1, we need to consider the deployment of data collection, preprocessing, and training. We identify two deployment strategies to distribute raw data from multiple data sources. The first strategy involves routing the raw data to a dedicated preprocessing cluster. After preprocessing (or a batch of preprocessing), the preprocessed data is then transferred to another dedicated cluster for training. However, this approach results in *double data movements*, including data transfers from multiple sources to the preprocessing cluster and then to the training cluster. The second strategy opts to distribute the raw data directly to a cluster responsible for both preprocessing and training. While this approach eliminates the need for double data transfers, it introduces the potential for resource contentions (e.g., memory and CPUs).

Observation. We examine the number of data movements from data collection to preprocessing. As the raw data is emitted from multiple sources, it means that the raw data is naturally distributed. Also, preprocessing the raw data from one source is independent of others, implying that preprocessing across different sources can be executed in parallel. Thus, we argue that the data movement from data collection to distributed preprocessing is redundant.

Solution. Based on our observation, we introduce near-data preprocessing to minimize data movements from data collection to preprocessing as well as alleviate resource contention. Specifically, ScaleDFP employs a localized approach to reduce data transfers in the preprocessing. Once each data collector collects disk logs, ScaleDFP executes the preprocessing on each data collector *locally*, including feature extraction, buffering samples into a dedicated local buffering window for online labeling, and time-window downsampling. Also, near-data preprocessing mitigates resource contention by offloading the buffering tasks (e.g., feature extraction and online labeling)

from centralized processing units to distributed data collectors, enabling parallel processing.

C. Random Downsampling

Problem. Recall from Section III-B that the preprocessing workflow includes buffering and time-window downsampling which increase the volume of preprocessed data. As the negative samples are dominant within the buffering window, the volume of preprocessed data depends on the L days of negative samples selected by the time-window downsampling.

Straw-man solution. An intuitive approach is by compression, which mitigates data inflation of preprocessing. While state-of-the-art compressing techniques [40] achieve substantial space savings (e.g., over 90%), there is a trade-off. Decompressing the preprocessed data from multiple data streams may resemble handling numerous small files, potentially leading to inefficiencies when accessed with a single thread. Alternatively, if managed with multiple threads in receivers for decompressing, it could consume excessive computational resources.

Observation. We next pinpoint if such preprocessed data is all needed for training. We observe that StreamDFP [17] performs two-phase downsampling before training, where the first-phase downsampling corresponds to the time-window downsampling and the second-phase downsampling conducts the customized Poisson sampling (Section II-B). Before and after the second-phase downsampling, the actual ratio of negative samples to positive ones is reduced from more than 100:1 to around 10:1 (based on the datasets in Table I). It implies that the preprocessed data can be further downsampled.

Solution. ScaleDFP uses random downsampling to prevent the unused negative samples from transferring. Specifically, after selecting the recent L days of negative samples, each data collector further randomly selects a subset of negative samples, which is referred to as the *downsampling ratio* (currently set as 10%). We evaluate the different percentages of keeping negative samples and find that keeping only 10% does not generally affect the prediction accuracy (Exp#6 in Section V-C). Also, to examine if data inflation still exists, we compare the amount of data after random downsampling with the amount of raw data before preprocessing. Let a denote the total number of disks. As negative samples are dominant after near-data preprocessing, we estimate the transferred training data as $L \times a \times 10\% = 0.1aL$. As L is often selected as the most recent days (e.g., seven days), ScaleDFP mitigates data inflation with the amount of training data less than that of raw data (e.g., equivalent to 70% of raw data) to start allocation.

D. Training Data Allocation

Main idea. To address C3, we extend the customized Poisson sampling [17], [41] from a single data source in ensemble learning algorithms to multiple data sources. This sampling method requires configuring two hyper-parameters, denoted as λ_n and λ_p (where $\lambda_n > 0$ and $\lambda_p > 0$), representing the Poisson distribution for negative and positive samples, respectively. For each sample, a weight is generated using the corresponding Poisson distribution, indicating the frequency of a sample's update during training. To mitigate the data imbalance issue [9] (Section II-B), λ_n is typically set less than λ_p to ensure that positive samples carry more weight than negative ones. One may argue that it is easy to directly integrate the customized Poisson sampling into each data collector. However, we notice that the implementation of Poisson sampling in StreamDFP relies on sample counts of positive and negative samples, which is not mentioned in the paper [17]. Specifically, λ_p remains fixed during training, while λ_n varies with the ratio of the number of positive samples to that of negative samples (denoted by r_g) each day. This adaptation mechanism ensures a proportional allocation of negative samples relative to positive samples. Thus, such global information needs to be aggregated from multiple data sources each day.

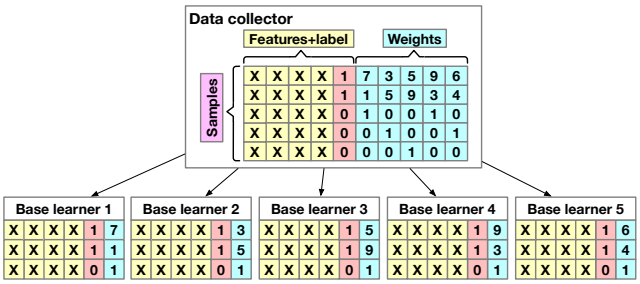


Fig. 3: Example of optimized data transmission.

Coordinator. We introduce a coordinator in ScaleDFP to aggregate the global count of positive and negative samples. The coordinator collects local counts of positive and negative samples each day from all data collectors. As sample counts consist of only two integer values per data collector, the latency associated with receiving local counts is negligible. Hence, for synchronization purposes, the coordinator waits until it receives local counts from all data collectors. The coordinator aggregates the total counts of positive and negative samples, respectively. It then computes the ratio r_g and sends back r_g to each data collector. In case the number of positive samples is zero, we set r_g as a small value (e.g., 0.001) to prevent the update of λ_n from becoming zero in data collectors. Also, as aggregating local counts is lightweight, the coordinator does not become a bottleneck with a large number of data collectors.

Optimized data transmission. We optimize the data transmission in ScaleDFP by ensuring that each base learner receives all positive samples and only relevant negative samples. First, data collectors update λ_n with r_g. Given an initial λ_n (denoted by λ_{n_0}), each data collector updates λ_n by multiplying λ_{n_0} and r_g. Second, for a sample (positive or negative), each data collector uses the corresponding Poisson distribution (λ_p or λ_n) to compute a weight for each base learner. A sequence of weights across base learners is referred to as a *weight vector* concatenated with the feature vector for each sample. Figure 3 shows that such feature and weight vectors from multiple samples are aggregated into a matrix, whose dimension is determined by the number of samples (rows) and the total number of features, weights, and labels (columns). Third, given that each base learner is deployed on a server, each

data collector sends only a subset of the matrix to each base learner. For each base learner, each data collector examines the corresponding weight column, filters out the samples with zero weight, and transmits the remaining samples along with their feature vectors and the corresponding weight column.

Mathematical analysis of the amount of data transmission. We analyze the amount of data transmission before and after optimized data transmission. Considering the samples on a data collector after random downsampling, we denote the count of positive samples and negative samples by c_p and c_n, respectively. The number of negative samples after Poisson sampling is denoted by c'_n, where $c'_n < c_n$. For brevity, we denote the dimension of the feature vector plus one label by m and the number of base learners by T. Suppose that base learners are deployed on q servers, where T is divisible by q. Without optimized data transmission, a data collector needs to send $N := (c_n + c_p) \times m \times q$ data to the prediction model. In contrast, with optimized data transmission, a data collector sends $N' := c_p \times (m + \frac{T}{q}) \times q + c'_n \times (m + \frac{T}{q}) \times q = (c_p + c'_n) \times (m + \frac{T}{q}) \times q$ to the prediction model. Also, the ratio of c_p to c_n is close to r_g. Let $N' < N$, and we derive $\frac{c'_n}{c_n} < \frac{m - \frac{T}{q} r_g}{m + \frac{T}{q}}$. We consider two extreme cases of deployment: i) deploying all base learners on one server (i.e., $q = 1$) and obtaining $\frac{c'_n}{c_n} < \frac{m - r_g T}{m + T}$ and ii) deploying each base learner on one server (i.e., $T/q = 1$) and obtaining $\frac{c'_n}{c_n} < \frac{m - r_g}{m + 1}$. Based on the settings and datasets (Section V-A), these conditions are easy to satisfy and thus $N' < N$. Our evaluation shows the effectiveness of training data allocation (Exp#2 in Section V-B).

Sequenced data retrieval. SCALEDFP uses receivers to maintain the chronological order of training data received from data collectors. Specifically, each receiver is responsible for receiving the relevant training data of $\frac{T}{q}$ base learners, including m columns of features (including one label) and $\frac{T}{q}$ columns of weights. For synchronization, after receiving the training data in one buffering window from all data collectors, it sorts all received samples based on their timestamps in the buffering window. In case there is a straggler in data collectors, SCALEDFP sets the timeout threshold of waiting for the straggler as a short time (e.g., 2 seconds).

E. Training and Prediction

Training. Each base learner uses the weight vector associated with each sample for training, bypassing the original Poisson sampling mechanism in ensemble learning. When a receiver supervises a single base learner, all samples possess non-zero weights, facilitating direct training input. Similarly, if a receiver supervises multiple base learners, each base learner is trained using the samples with non-zero weights accordingly, significantly reducing the number of samples compared to that without training data allocation. Base learners continuously adapt to concept drift via change detectors [8], [15]. Note that change detectors still need to use the zero-weight samples (only feature and label columns) for detecting concept drifts. As change detectors can be deployed on different servers from base learners [23], we leave the optimization of data transmission for change detectors in our future work.

Prediction. SCALEDFP predicts whether a disk will fail in the near future. Specifically, each data collector extracts features and promptly forwards unlabeled samples to each receiver without maintaining the chronological order, as all unlabeled samples are derived from the same timestamp. Each receiver assigns each unlabeled sample to every base learner for prediction. The prediction model collects and combines the predicted values from the base learners for each sample, yielding the final prediction results.

IV. IMPLEMENTATION

We prototyped SCALEDFP in Python, comprising data collectors, a coordinator, and receivers with ∼550 LoC. Among different components, we implement network communications using Remote Procedure Calls (RPCs) via the `gRPC` library [5]. For data collectors, we realize near-data preprocessing based on the preprocessing workflow in StreamDFP and integrate random downsampling. After random downsampling, each data collector sends its local counts of positive and negative samples to the coordinator. The coordinator then returns the ratio of the total number of positive samples to negative samples to each data collector. We also implement the customized Poisson sampling on data collectors. Due to Python's inherent performance limitations in for-loops for computing weights for each sample across all base learners, we optimize the Poisson sampling computation using the `pybind11` library [3]. Additionally, data collectors transmit training data to receivers via RPCs. Receivers ensure receipt of training data from all data collectors and sort samples by their timestamps before writing them into the local file system. Samples are read from the local file system and fed into the ensemble learning algorithms (i.e., BA and ARF). We realize the ensemble learning algorithms based on StreamDFP and replace its original Poisson sampling approach by our training data allocation approach.

V. EVALUATION

We conduct a trace-driven evaluation to evaluate the system performance and prediction accuracy of SCALEDFP. We summarize our findings as follows:

- In preprocessing, SCALEDFP demonstrates the scalability with an increasing number of data collectors, achieving up to a 41.6× throughput gain compared to StreamDFP (Exp#1). Enabling training data allocation boosts the preprocessing throughput gain, showing a 4.3× throughput gain compared to disabling training data allocation (Exp#2). In addition, the training throughput benefits from random downsampling, achieving throughput gains of up to 13.2× and 8.4× for BA and ARF, respectively, compared to disabling random downsampling (Exp#3).
- SCALEDFP attains comparable, and in some cases, higher prediction accuracy than StreamDFP (Exp#4). When dealing with multiple data collectors for training, SCALEDFP ensures that the prediction accuracy remains comparable to that

Dataset	Disk model (ID)	Capacity	Disk count	# failures	# features	Period	Duration
Backblaze	HDD Seagate ST3000DM001 (D1)	3 TB	1,930	203	50	2014-02-15 to 2015-05-20	460 days
	HDD Seagate ST4000DM000 (D2)	4 TB	35,587	797	48	2015-01-01 to 2016-04-04	460 days
	HDD HGST HMS5C4040BLE640 (D3)	4 TB	14,613	65	34	2019-01-01 to 2020-04-04	460 days
Alibaba	SSD MA1 (D4)	480 GB	39,860	1,126	40	2018-01-03 to 2019-04-07	460 days
	SSD MB1 (D5)	1920 GB	42,866	917	38	2018-01-03 to 2019-04-07	460 days
	SSD MC1 (D6)	1920 GB	191,589	1,421	42	2018-01-03 to 2019-04-07	460 days

TABLE I: Overview of datasets.

of a single data collector (Exp#5). In random downsampling, keeping 10% of negative samples generally achieves a comparable prediction accuracy compared to keeping all negative samples (Exp#6).

A. Methodology

Datasets. To validate our evaluation, we employ SMART (Self-Monitoring, Analysis, and Reporting Technology) logs [35] to predict impending disk failures. Within modern data centers [18], [27], [43], [48], a monitoring daemon is often deployed to periodically collect SMART logs, typically on a daily [18], [43] or hourly [27] basis. SMART is an embedded monitoring system found in the firmware of disk drives, encompassing both hard disk drives (HDDs) and solid-state drives (SSDs). It functions by detecting and reporting the performance and reliability statistics of a disk drive in the form of numerical values, referred to as *attributes*. Each attribute comprises two types of values: raw and normalized values. We extract each value of an attribute as a feature.

We use two publicly available datasets collected from distinct production environments on a daily basis as shown in Table I. The first dataset includes HDD failures from Backblaze [4], while the second dataset comprises SSD failures from Alibaba [18], [43]. From each dataset, we select three commonly used disk models, namely D1 to D6, which are evaluated in prior studies [9], [17], [31], [41], [48]. For a disk model, we extract all SMART attributes (including raw and normalized values) as features. Also, to simulate a large-scale disk deployment, we treat *all* disk models in each dataset as a whole collection, considering that the Backblaze and Alibaba datasets encompass around 100 K HDDs [4] and 1 M SSDs [18], respectively. For experiments involving multiple data collectors (Exp#1 and Exp#4), we evenly divide the entire dataset into multiple partitions, aligning with the number of data collectors. Each data collector processes a designated partition as input.

Metrics. We evaluate both the system performance and prediction accuracy on a daily basis. For the system performance, we measure the throughput of preprocessing and training separately, instead of an end-to-end throughput. Specifically, the throughput of preprocessing is calculated by the number of disks for preprocessing over the execution time starting from feature extraction to the end of sequenced data retrieval on receivers (Exp#1). On the other hand, the throughput of training is calculated by the number of disks for preprocessing over the execution time of training (Exp#2).

For the prediction accuracy, we consider three classical metrics (also used in StreamDFP [17]):

- *Precision:* The ratio of actual failed disks that have been correctly predicted as failures to the total number of disks predicted as failures, whether correctly or incorrectly.
- *Recall:* The ratio of actual failed disks that have been accurately predicted as failures to the total number of actual failed disks.
- *F1-score:* $\frac{2\times\text{precision}\times\text{recall}}{\text{precision}+\text{recall}}$.

Testbeds. We evaluate ScaleDFP on a local testbed and Alibaba Cloud [1]. Our local testbed consists of two servers. Each server is equipped with two 20-core 2.10 GHz Intel Xeon(R) Gold 5218R CPUs, 256 GiB of RAM, and a Seagate ST4000NM000A-2HZ100 7200 RPM 4 TiB SATA hard disk. These two servers are connected via a 1 Gbps network. We leverage our local testbed to measure the training throughput (Exp#3) and prediction accuracy (Exp#4 to Exp#6). Also, to evaluate the scalability of preprocessing (Exp#1 and Exp#2), we provision 130 general-purpose instances on Alibaba Cloud for ScaleDFP, which includes multiple data collectors, a coordinator, and one receiver. We fix one receiver for receiving the training data for a fair comparison with StreamDFP and it would not be the bottleneck in ScaleDFP with optimized data transmission (Exp#2). All components are deployed in `ecs.g6.2xlarge` instances with 8 vCPUs and 32 GiB RAM. Each instance is equipped with a 40 GiB enhanced SSD with performance level PL0 [2] and is installed with Ubuntu 20.04. All instances are connected via an 8 Gbps network.

Default setups. We consider StreamDFP [17] as the state-of-the-art baseline without scaling disk failure prediction via stream mining. For a fair comparison with StreamDFP, we configure the same evaluation durations as the same in StreamDFP [17]. Specifically, we choose identical 460 days of samples and set the same start date for each dataset (shown in the "Period" column). For each dataset, we initialize the prediction model from scratch with the initial 30 days of samples. For both ScaleDFP and StreamDFP, we set the extra labeled days as 20 days for online labeling. For ScaleDFP, we set the random downsampling ratio as 10% by default, while we evaluate the impact of the random downsampling ratio in Exp#6. Subsequently, we carry out daily predictions of disk failures for the subsequent 30 days. Our evaluation spans 400 days in total for all datasets for Exp#3 to Exp#6, while our evaluation spans 30 days for Exp#1 and Exp#2 due to the limited capacity of SSDs in our Alibaba Cloud testbed. We run two ensemble learning algorithms, i.e., BA and ARF, and fix 30 base learners as in StreamDFP [17]. To ensure fair comparisons on prediction accuracy, we fix the threshold of the average false positive rate (FPR) over the evaluation period for

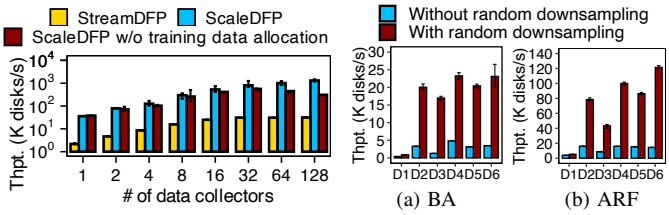


Fig. 4: Exp#1 (Scalability of preprocessing) and Exp#2 (Impact of training data allocation). X-axis and Y-axis are in log scale.

Fig. 5: Exp#3 (Impact of random downsampling on training throughput).

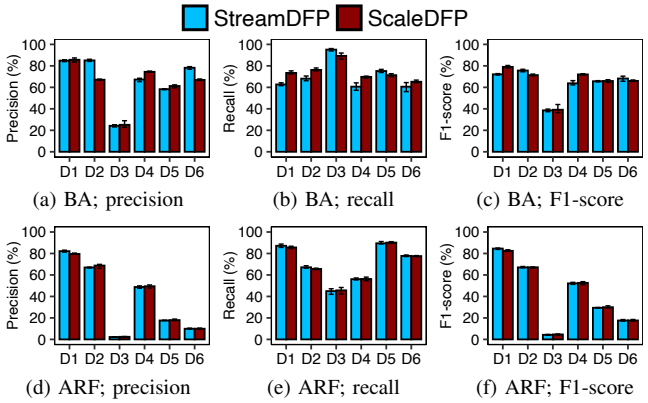


Fig. 6: Exp#4 (Justification of prediction accuracy).

each disk model. Specifically, we calculate the FPR on each day, which is defined as the fraction of falsely predicted failed disks over the total number of healthy disks in the subsequent 30 days. We set the default FPR at 1.0%, which aligns with previous studies [17], [41]. We conduct the experiments across five runs and report the average results over these runs with the maximum and minimum error bars.

B. System Performance

Exp#1 (Scalability of preprocessing). We evaluate the scalability of preprocessing in a large-scale disk deployment, covering the entire process from data collection to preprocessing. For StreamDFP, we emulate data collection from one or multiple data collectors via TCP socket-based file transmission to a single server where StreamDFP executes the preprocessing. We vary the number of data collectors from 1 to 128 for both StreamDFP and ScaleDFP. To ensure a sufficient number of disks for preprocessing on individual data collectors as the number of collectors increases, both ScaleDFP and StreamDFP execute preprocessing on D6, which hosts the largest number of disks.

Figure 4 shows that ScaleDFP scales to a large number of data collectors. We first compare the preprocessing throughput between ScaleDFP and StreamDFP, while we compare the preprocessing throughput with and without training data allocation in Exp#2. For one data collector, the throughput of ScaleDFP is higher than that of StreamDFP by 16.5×, since the execution time of collecting data is dominant, accounting for 94.9% of total execution time for StreamDFP. With increasing the number of data collectors from 2 to 128, the throughput gains of ScaleDFP increase from 17.4× to 41.6× those of StreamDFP. In contrast, although the throughput of StreamDFP also increases with 2 to 16 data collectors, it is throttled by the execution of preprocessing on a single server with 32 to 128 data collectors. This implies that near-data preprocessing is effective for scaling the preprocessing.

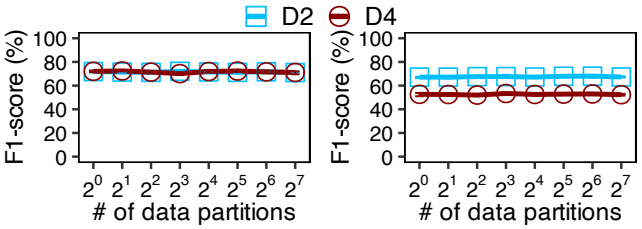
Exp#2 (Impact of training data allocation). We evaluate the preprocessing throughput by enabling and disabling training data allocation. Disabling training data allocation involves deactivating the customized Poisson sampling and optimized data transmission in data collectors, while retaining the sequenced data retrieval to ensure the prediction accuracy. We still use Figure 4 to show the results. For one data collector, the throughput of ScaleDFP is lower than that without training data allocation by 5.3% due to the computation overhead of customized Poisson sampling. However, the throughput gains of ScaleDFP increase from 1.1× to 4.3× those without training data allocation, especially for 64 (2.2×) and 128 (4.3×) data collectors. With optimized data transmission, the receiver does not become a bottleneck even as the number of data collectors increases. In contrast, without optimized data transmission, the receiver is overloaded with training data from numerous data collectors (e.g., 128 data collectors). Moreover, adding more receivers does not alleviate this overload, as each data collector sends the same training data to all receivers (i.e., N in Section III-D).

Exp#3 (Impact of random downsampling on training throughput). Disabling the training data allocation, we evaluate the training throughput before and after enabling random downsampling for BA and ARF. Figure 5 shows that random downsampling improves the training throughput by 2.2× to 13.2× and 1.3× to 8.4× for BA and ARF, respectively, on all disk models, compared to training without random downsampling. The reason is that random downsampling further reduces the number of negative samples by 90%. Although Poisson sampling in training data allocation can also reduce the number of negative samples for training, random downsampling is still efficient for reducing the number of negative samples for computing the weights in Poisson sampling.

C. Prediction Accuracy

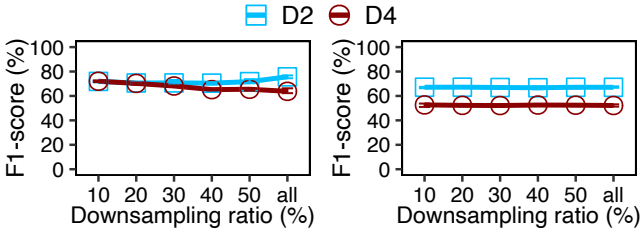
Exp#4 (Justification of prediction accuracy). We first justify the prediction accuracy of ScaleDFP compared to StreamDFP by fixing the number of data collectors as one. Figure 6 shows that ScaleDFP generally achieves a prediction accuracy comparable to StreamDFP for BA and ARF. The differences in F1-scores between ScaleDFP and StreamDFP are within 3.8% and 2.1% for BA and ARF, respectively, except for the cases where ScaleDFP improves the F1-scores of BA by 6.9% on D1 and 8.3% on D4. Overall, ScaleDFP ensures a comparable prediction accuracy with StreamDFP, while the reason for accuracy improvement is that the random downsampling may just drop some unhelpful negative samples.

Exp#5 (Prediction accuracy of multiple data collectors). We evaluate the prediction accuracy by varying the number of data



(a) BA; F1-score (b) ARF; F1-score

Fig. 7: Exp#5 (Prediction accuracy of multiple data collectors). The precision and recall are in the same trend with the F1-score and omitted in plots.



(a) BA; F1-score (b) ARF; F1-score

Fig. 8: Exp#6 (Impact of downsampling ratio). "all" denotes keeping all negative samples with disabling random downsampling. The precision and recall are in the same trend with the F1-score and omitted in plots.

collectors. We focus on two disk models D2 and D4 for HDD and SSD, respectively. D2 has the highest disk count among HDD datasets (D1 to D3), while D4 exhibits the highest failure rate (i.e., failure count to total disk count) in the SSD dataset (D4 to D6). Figure 7 shows that the F1-score remains stable over the number of data collectors. Specifically, for D2 (D4), ScaleDFP keeps the F1-scores of multiple data collectors for BA and ARF close to that of one data collector, i.e., within 0.16% (0.33%) and 0.84% (0.69%), respectively.

Exp#6 (Impact of downsampling ratio). We vary the random downsampling ratio from 10% to 50% and compare the prediction accuracy with disabling random downsampling (i.e., keeping all negative samples after the time-window downsampling). Figure 8 shows the F1-score for D2 and D4 with different downsampling ratios. For ARF, the prediction accuracy with a 10% downsampling ratio is close to that with keeping all negative samples, i.e., the difference of 0.071% for D2 and 0.46% for D4. In contrast, for BA, the prediction accuracy with 10% downsampling ratio is higher than that with keeping all negative samples by 8.3% for D4, while lower than that with keeping all negative samples by 3.8% for D2. The reason is that D4 has more negative samples than D2, which is more crucial to reduce negative samples. It also implies that the optimal downsampling ratio is related to learning algorithms.

VI. Related Work

Disk failure prediction. Extensive studies on disk failure prediction have primarily centered on offline learning, assuming that all training data is readily available. These prediction approaches can be categorized into three types: (i) statistical techniques (e.g., rank sum tests [21], Hidden-Markov models [49], and quantile distribution analysis [30]), (ii) machine learning (e.g., naive Bayesian classifiers [16], multiple-instance naive Bayesian learning [32], backpropagation neural networks [50], decision trees [24], regularized greedy forests [9], random forests [31], [43], [48], and gradient boosting additive regression trees [44]), and (iii) deep neural networks (e.g., recurrent neural networks [42], layerwise perturbation-based adversarial training [47], convolutional neural networks [28], [38], long short-term memory based siamese network [46], long short-term memory for feature encoding [27], and neighborhood-temporal attention model [29]). The closest related studies to ours are [17], [41], which apply online stream-based learning for disk failure prediction and continuously train the prediction model on the incoming data stream. However, ScaleDFP addresses the scalability of preprocessing that is not considered in [17], [41]: (i) preprocessing in parallel for multi-source stream mining (instead of on a single machine), and (ii) reducing computational overhead and saving network bandwidth by random downsampling and training data allocation (instead of no preprocessing optimization).

Anomaly detection. Prior studies explore various anomaly detection techniques (e.g., statistical approaches [19], machine learning [7], and deep learning [33]). While some algorithms (e.g., random forest [26]) are used in both anomaly detection and disk failure prediction (e.g., [31], [43], [48]), we argue that training prediction models in a streaming fashion is crucial for multiple evolving data streams.

Distributed processing systems. Existing distributed processing systems (e.g., Spark [45], Storm [39], Flink [11], and AFStream [20]) primarily serve general-purpose data analytics and ML tasks, but fall short of training prediction models on disk failure prediction. Recent distributed machine learning systems, including the parameter server [25], TensorFlow [6], PyTorch [34], and BytePS [22], focus on optimizing the execution of ML tasks but often lack the integration of data collection, leading to substantial data movements before training. In contrast, ScaleDFP addresses the scalability of preprocessing for disk failure prediction.

VII. Conclusion

We present ScaleDFP, a multi-source stream mining framework for scaling disk failure prediction. ScaleDFP addresses the scalability of preprocessing before training for disk failure prediction via near-data preprocessing, random downsampling, and training data allocation. It achieves the preprocessing throughput gains of up to 41.6× and the training throughput gains of up to 13.2×, while maintaining a comparable prediction accuracy compared to StreamDFP [17].

Acknowledgements

This work was supported in part by the National Key Research and Development Program of China (2023YFB2904600), National Natural Science Foundation of China (62172007), and Fundamental Research Funds for the Central Universities.

REFERENCES

[1] Alibaba Cloud. https://www.alibabacloud.com/product/ecs-pricing-list/en.

[2] Alibaba Cloud - ESSDs. https://www.alibabacloud.com/help/en/elastic-compute-service/latest/essds.

[3] pybind11 — seamless operability between C++11 and python. https://github.com/pybind/pybind11.

[4] Backblaze dataset. https://www.backblaze.com/b2/hard-drive-test-data.html, 2013.

[5] gRPC. https://grpc.io/, 2015.

[6] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al. TensorFlow: a system for large-scale machine learning. In *Proc. of USENIX OSDI*, 2016.

[7] M. Ahmed, A. N. Mahmood, and J. Hu. A survey of network anomaly detection techniques. *Journal of Network and Computer Applications*, 60:19–31, 2016.

[8] A. Bifet and R. Gavaldà. Adaptive learning from evolving data streams. In *Proc. of Springer IDA*, 2009.

[9] M. M. Botezatu, I. Giurgiu, J. Bogojeska, and D. Wiesmann. Predicting disk replacement towards reliable data centers. In *Proc. of ACM SIGKDD*, 2016.

[10] L. Breiman. Bagging predictors. *Machine learning*, 24(2):123–140, 1996.

[11] P. Carbone, A. Katsifodimos, S. Ewen, V. Markl, S. Haridi, and K. Tzoumas. Apache Flink: Stream and batch processing in a single engine. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, 38(4), 2015.

[12] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels. Dynamo: Amazon's highly available key-value store. In *Proc. of ACM SOSP*, 2007.

[13] Y. Freund, R. E. Schapire, et al. Experiments with a new boosting algorithm. In *Proc. of ACM ICML*, volume 96, pages 148–156, 1996.

[14] J. Gama, I. Žliobaitė, A. Bifet, M. Pechenizkiy, and A. Bouchachia. A survey on concept drift adaptation. *ACM Computing Surveys*, 46(4):44, 2014.

[15] H. M. Gomes, A. Bifet, J. Read, J. P. Barddal, F. Enembreck, B. Pfharinger, G. Holmes, and T. Abdessalem. Adaptive random forests for evolving data stream classification. *Machine Learning*, 106(9-10):1469–1495, 2017.

[16] G. Hamerly, C. Elkan, et al. Bayesian approaches to failure prediction for disk drives. In *Proc. of ACM ICML*, 2001.

[17] S. Han, P. P. C. Lee, Z. Shen, C. He, Y. Liu, and T. Huang. StreamDFP: A general stream mining framework for adaptive disk failure prediction. *IEEE Transactions on Computers*, 72(02):520–534, 2023.

[18] S. Han, P. P. C. Lee, F. Xu, Y. Liu, C. He, and J. Liu. An in-depth study of correlated failures in production SSD-based data centers. In *Proc. of USENIX FAST*, 2021.

[19] V. Hodge and J. Austin. A survey of outlier detection methodologies. *Artificial intelligence review*, 22:85–126, 2004.

[20] Q. Huang and P. P. C. Lee. Toward high-performance distributed stream processing via approximate fault tolerance. In *Proc. of VLDB Endowment*, 2016.

[21] G. F. Hughes, J. F. Murray, K. Kreutz-Delgado, and C. Elkan. Improved disk-drive failure warnings. *IEEE Trans. on Reliability*, 51(3):350–357, 2002.

[22] Y. Jiang, Y. Zhu, C. Lan, B. Yi, Y. Cui, and C. Guo. A unified architecture for accelerating distributed DNN training in heterogeneous GPU/CPU clusters. In *Proc. of USENIX OSDI*, 2020.

[23] N. Kourtellis, G. D. F. Morales, A. Bifet, and A. Murdopo. VHT: Vertical hoeffding tree. In *Proc. of IEEE ICBD*, 2016.

[24] J. Li, X. Ji, Y. Jia, B. Zhu, G. Wang, Z. Li, and X. Liu. Hard drive failure prediction using classification and regression trees. In *Proc. of IEEE/IFIP DSN*, 2014.

[25] M. Li, D. G. Andersen, J. W. Park, A. J. Smola, A. Ahmed, V. Josifovski, J. Long, E. J. Shekita, and B.-Y. Su. Scaling distributed machine learning with the parameter server. In *Proc. of OSDI*, 2014.

[26] D. Liu, Y. Zhao, H. Xu, Y. Sun, D. Pei, J. Luo, X. Jing, and M. Feng. Opprentice: Towards practical and automatic anomaly detection through machine learning. In *Proc. of ACM IMC*, 2015.

[27] Y. Liu, H. Yang, P. Zhao, M. Ma, C. Wen, H. Zhang, C. Luo, Q. Lin, C. Yi, J. Wang, et al. Multi-task hierarchical classification for disk failure prediction in online service systems. In *Proc. of ACM SIGKDD*, 2022.

[28] S. Lu, B. Luo, T. Patel, Y. Yao, D. Tiwari, and W. Shi. Making disk failure predictions SMARTer! In *Proc. of USENIX FAST*, 2020.

[29] C. Luo, P. Zhao, B. Qiao, Y. Wu, H. Zhang, W. Wu, W. Lu, Y. Dang, S. Rajmohan, Q. Lin, et al. NTAM: Neighborhood-temporal attention model for disk failure prediction in cloud platforms. In *Proc. of WWW*, 2021.

[30] A. Ma, R. Traylor, F. Douglis, M. Chamness, G. Lu, D. Sawyer, S. Chandra, and W. Hsu. RAIDShield: Characterizing, monitoring, and proactively protecting against disk failures. *ACM Trans. on Storage*, 11(4):17, 2015.

[31] F. Mahdisoltani, I. Stefanovici, and B. Schroeder. Proactive Error Prediction to Improve Storage System Reliability. In *Proc. of USENIX ATC*, 2017.

[32] J. F. Murray, G. F. Hughes, and K. Kreutz-Delgado. Machine learning methods for predicting failures in hard drives: A multiple-instance application. *Journal of Machine Learning Research*, 6(May):783–816, 2005.

[33] G. Pang, C. Shen, L. Cao, and A. V. D. Hengel. Deep learning for anomaly detection: A review. *ACM Computing Surveys*, 54(2):1–38, 2021.

[34] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, et al. PyTorch: An imperative style, high-performance deep learning library. *Proc. of NeurIPS*, 2019.

[35] E. Pinheiro, W.-D. Weber, and L. A. Barroso. Failure trends in a large disk drive population. In *Proc. of USENIX FAST*, 2007.

[36] A. Sandur, C. Park, S. Volos, G. Agha, and M. Jeon. Jarvis: Large-scale server monitoring with adaptive near-data processing. In *Proc. of IEEE ICDE*, 2022.

[37] C. Shen, Q. Ouyang, F. Li, Z. Liu, L. Zhu, Y. Zou, Q. Su, T. Yu, Y. Yi, J. Hu, et al. Lindorm TSDB: A cloud-native time-series database for large-scale monitoring systems. In *Proc. of the VLDB Endowment*, 2023.

[38] X. Sun, K. Chakrabarty, R. Huang, Y. Chen, B. Zhao, H. Cao, Y. Han, X. Liang, and L. Jiang. System-level hardware failure prediction using deep learning. In *Proc. ACM/IEEE DAC*, 2019.

[39] A. Toshniwal, S. Taneja, A. Shukla, K. Ramasamy, J. M. Patel, S. Kulkarni, J. Jackson, K. Gade, M. Fu, J. Donham, et al. Storm@ twitter. In *Proc. of ACM SIGMOD*, 2014.

[40] J. Wei, G. Zhang, Y. Wang, Z. Liu, Z. Zhu, J. Chen, T. Sun, and Q. Zhou. On the feasibility of parser-based log compression in large-scale cloud systems. In *Proc. of USENIX FAST*, 2021.

[41] J. Xiao, Z. Xiong, S. Wu, Y. Yi, H. Jin, and K. Hu. Disk failure prediction in data centers via online learning. In *Proc. of ACM ICPP*, 2018.

[42] C. Xu, G. Wang, X. Liu, D. Guo, and T.-Y. Liu. Health status assessment and failure prediction for hard drives with recurrent neural networks. *IEEE Trans. on Computers*, 65:3502–3508, 2016.

[43] F. Xu, S. Han, P. P. C. Lee, Y. Liu, C. He, and J. Liu. General feature selection for failure prediction in large-scale SSD deployment. In *Proc. of IEEE/IFIP DSN*, 2021.

[44] Y. Xu, K. Sui, R. Yao, H. Zhang, Q. Lin, Y. Dang, P. Li, K. Jiang, W. Zhang, J.-G. Lou, et al. Improving service availability of cloud systems by predicting disk error. In *Proc. of USENIX ATC*, 2018.

[45] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica. Spark: Cluster computing with working sets. In *Proc. of USENIX HotCloud*, 2010.

[46] J. Zhang, P. Huang, K. Zhou, M. Xie, and S. Schelter. HDDse: Enabling high-dimensional disk state embedding for generic failure detection system of heterogeneous disks in large data centers. In *Proc. of USENIX ATC*, 2020.

[47] J. Zhang, J. Wang, L. He, Z. Li, and S. Y. Philip. Layerwise perturbation-based adversarial training for hard drive health degree prediction. In *Proc. of IEEE ICDM*, 2018.

[48] Y. Zhang, W. Hao, B. Niu, K. Liu, S. Wang, N. Liu, X. He, Y. Gwon, and C. Koh. Multi-view feature-based SSD failure prediction: What, when, and why. In *Proc. of USENIX FAST*, 2023.

[49] Y. Zhao, X. Liu, S. Gan, and W. Zheng. Predicting disk failures with HMM-and HSMM-based approaches. In *Proc. of IEEE ICDM*, 2010.

[50] B. Zhu, G. Wang, X. Liu, D. Hu, S. Lin, and J. Ma. Proactive drive failure prediction for large scale storage systems. In *Proc. of IEEE MSST*, 2013.