SEFEE: Lightweight Storage Error Forecasting in Large-Scale Enterprise Storage Systems

Amirhessam Yazdi
University of Nevada, Reno
Reno, NV
ayazdi@nevada.unr.edu

Xing Lin
NetApp
Sunnyvale, CA
xing.lin@netapp.com

Lei Yang
University of Nevada, Reno
Reno, NV
leiy@unr.edu

Feng Yan
University of Nevada, Reno
Reno, NV
fyan@unr.edu

***Abstract*—With the rapid growth in scale and complexity, today's enterprise storage systems need to deal with significant amounts of errors. Existing proactive methods mainly focus on machine learning techniques trained using SMART measurements. However, such methods are usually expensive to use in practice and can only be applied to a limited types of errors with a limited scale. We collected more than 23-million storage events from 87 deployed NetApp-ONTAP systems managing 14,371 disks for two years and propose a lightweight training-free storage error forecasting method SEFEE. SEFEE employs Tensor Decomposition to directly analyze storage error-event logs and perform online error prediction for all error types in all storage nodes. SEFEE explores hidden spatio-temporal information that is deeply embedded in the global scale of storage systems to achieve record breaking error forecasting accuracy with minimal prediction overhead.**

***Index Terms*—Storage failures, error prediction, lightweight forecasting, training-free prediction, tensor decomposition.**

I. Introduction

Enterprise storage system is typically equipped with high-end storage devices and extra reliability schemes, such as RAID, high-availability (HA) pairs, and cross datacenter replication. With the rapid growth in scale and complexity of today's enterprise storage systems, there is a significant increase in the amount and types of storage errors, which poses significant challenges in storage reliability and availability. Our study on 2-year storage error-event logs from the NetApp ONTAP storage systems [1] shows that today's enterprise storage systems can generate up to 1,450 events per day per system and more than 6,000 types of errors ranging from storage hardware, networking, power supply to software stack. Some errors can result in immediate system crash, while other errors are latent but eventually may lead to more severe errors or even system failures.

To deal with the storage errors, many existing solutions rely on reactive detection methods, which debug the errors after these errors have caused issues or failures [2, 3]. One main drawback of reactive detection methods is that these methods usually cause poor system availability and are inefficient to deploy in large-scale storage systems. To enhance the reliability and availability of large-scale storage systems, it is of paramount importance to proactively forecast the errors so that proper prevention and/or fixing actions can be taken timely. State-of-the-art proactive solutions forecast the storage errors based on whether a disk or system is going to fail soon [4, 5] or the ranked failure probabilities of all disks in the near future [6, 7]. These solutions usually adopt machine learning techniques (e.g., Random Forests and Long Short Term Memory (LSTM) networks) to train a model using historical data, such as Self-Monitoring, Analysis, and Reporting Technology (SMART) measurements [4–6, 8–10]. A recent approach [11] combines SMART measurements with disk performance metrics and location data.

However, state-of-the-art approaches are not suitable for today's enterprise storage systems with large amounts of heterogeneous types of errors, as these approaches can predict either *complete disk failure* (i.e., disk operational or not) or whether some severe errors/failures (such as sector reallocation) occur, not to mention the long training and prediction time as well as high demand for computational resources. Due to the high complexity of modern enterprise storage systems, the upper-layer services may experience latent errors, performance instability [12] and slow-downs at different levels (i.e., faulty memory requiring more ECC checks, software bugs, etc.) before a complete failure [13]. Some of these latent errors and slow-downs may cause cascading failures on other components of storage hierarchy and potentially cause slow-downs on cluster scale, which may lead to a complete failure. Therefore, predicting only a complete failure or a few severe error/failure is not sufficient to achieve high system reliability and availability in today's storage systems. In this paper, we aim at developing a lightweight storage forecasting methodology that can jointly predict all types of errors in all storage nodes.

To this end, we first collected and analyzed 2-year storage error-event logs, containing more than 23 million events from 8 enterprise deployments and 4 workloads from tens of customers of NetApp ONTAP storage systems. The error-event logs record errors with different sources (e.g., disk, SSD, controller, network, degraded power supply unit, RAID-level checksum errors, etc.), severity levels (i.e., from mild debug error to severe emergency error), occurrence location, and time. We summarize our key findings below:

- The error occurrence frequencies for different error types are diverse. 48% of error types occur less than 10 times per year and 18% of error types happen only once a year. The error distribution is imbalanced and time-varying.

- Severe error events occur less frequently. The average occurrence (i.e., density) of severe error events during a 2-hour period is 0.18%, which is far less than the average density (0.77%) and accumulated density (1.1%) of all error types.
- The distribution of error event types follows a power law. 4% of error types account for 80% of all error events, while 96% of error types account for 20% of events.
- There is a complex spatio-temporal inter-dependency between the location and the type of errors.

The above findings suggest that it is challenging to handcraft good features for machine learning based forecast models and that the sparsity of error events makes the model training challenging. To address these challenges, we develop a lightweight *training-free* online forecasting methodology, namely SEFEE (Storage Error Forecasting using Tensor decomposition), which can predict all types of errors as well as their locations. SEFEE takes storage error-event logs as streaming input and convert it into a tensor format with three dimensions: error event type, location, and occurrence time stamp. SEFEE employs tensor decomposition together with a contextual information of error severity index to obtain enhanced tensor factorization matrices, which can capture the latent correlation structure associated with each dimension. Based on these factorization matrices, the online storage error prediction is developed by reconstructing the factorization matrix associated with the time dimension, where the heterogeneous time-of-day effect discovered in the storage error-event logs is integrated.

We evaluate SEFEE using the real-world traces collected from NetApp ONTAP storage systems. The results corroborate that SEFEE is lightweight (i.e., training-free), which takes only seconds to predict errors on a number of large-scale storage systems with over 8,600 storage devices. Moreover, SEFEE outperforms state-of-the-art machine learning based methods by more than 15% in F1-score and achieves a very balanced prediction accuracy with 87.2% Recall and Precision of 85.6%. SEFEE has important business impact as it can proactively detect around 90% of failures that could result in potential data loss if they are not handled in a proper and timely fashion.

II. Storage Trace Data and Characterization

A. Trace Data Description

We collected the storage error trace from system-level event logs generated by NetApp ONTAP enterprise storage systems [1] deployed in production by their customers (e.g., Internet companies, manufacturing companies, education institutions, and financial institutions). The storage trace data contains the disk drive information (including its physical location in terms of *stackid.shelfid.diskid* and logical location such as RAID group information) and all error events that the ONTAP system reported. The data was collected in two batches. In the first batch (i.e., batch-2016), we collected the trace from 8 storage clusters, each deployed at a different customer for the year of 2016. This batch includes 23 systems (also referred to as storage nodes or storage controllers), managing 4,655 disks in 284 RAID groups. In the second batch (i.e., batch-2017), we collected information for systems that were used to deploy a certain type of workloads. We were able to collect four types of workloads, Workload A through D, which correspond to Microsoft Exchange, Microsoft SQL server, Oracle Database, and Microsoft Sharepoint workloads, respectively (see Table I). Within each workload, we collected between 10 to 20 deployed systems. In total, the second batch includes 62 systems, with 9716 disks in 455 RAID groups. In the interest of space, this paper mainly presents the results of the trace from batch-2017, as the results from batch-2016 share the similar insights and batch-2017 corresponds to larger-scale systems that contain more types of error events with finer-grained severity level for each event. Below are more concrete details of collected metrics.

TABLE I: Workloads in batch-2017 dataset.

Workload	Application	Nodes	Error types
Workload A	Exchange	16	1369
Workload B	MSSQL	11	890
Workload C	Oracle	18	1243
Workload D	SharePoint	17	938

- **RAID groups**: specifies the logical location of each disk. A RAID group consists of one or more disks, across which user's data is striped and stored. NetApp employs both RAID-DP with double parity disks and RAID-TEC for triple parity disks. RAID-DP is the default RAID type for all NetApp aggregates. RAID-DP is a modified RAID-4, supporting double parity disks. In RAID-DP or RAID-TEC, there are two or three dedicated parity disks, depending on the RAID type. For more details on RAID-DP, please check the technical report from NetApp [15]. It is worth noting that RAID-DP/TEC in ONTAP do not suffer from the problem of frequent parity block updates. There are two main reasons behind this. First, the WAFL file system is a log-structure file system. It does not do in-place updates. Instead, it stores new writes by appending the new data into the free space. Secondly, the RAID layer uses an algorithm to pack writes into full-stripe writes and the parity blocks are written at the same time as the full-stripe write. This significantly reduces partial stripe writes, which requires parity block updates.
- **Disk counts**: specifies, per system (i.e., node), how many disks are from each disk type.
- **Disk placement**: specifies the physical location of each disk in the format of *stackid.shelfid.diskid* that indicates where a disk drive is installed. Thus, the physical storage hierarchy in our data follows disks, shelves, stacks, nodes (systems) and storage cluster. In batch-2017, nodes within a workload can be from different storage clusters and different customers. In this paper, we choose node (i.e., system) as the storage hierarchy. All the nodes within the same workload serve the same application (e.g., SharePoint).
- **Disk type**: specifies the following type of a disk in a RAID group:

TABLE II: Error severity group, index labels, and density measured by binning the trace into 2-hour bins.

Severity					
Group	**index: Label**	**Definition** [14]	**Density**		**# error types**
1	0: Emergency	System is unusable.	0.18%	0.52%	70
	1: Alert	Temporary loss of service. Immediate Action required.		0.09%	
	2: Critical	Critical condition, cause should be determined. Not immediately fatal.		0.13%	
2	3: Error	Software error. Not immediately fatal.	0.6%		201
3	4: Warning	A high-priority message. does not indicate a fault.	0.8%	0.59%	363
	5: Notice	Normal-priority message. does not indicate a fault.		0.89%	
4	6: Informational	Low-priority message. does not indicate a fault.	1.5%	0.95%	735
	7: Debug	A debugging message, typically suppressed from customer.		2.3%	
		Overall:	**1.1%**		**1369**

- **DATA** disks hold data on behalf of clients within RAID groups.
- **PARITY/DPARITY** disks store row/diagonal parity information, which is used for data reconstruction.
- **SPARE** disks that are used when a disk fails within a RAID group. The data of the failed disk is reconstructed on a hot spare from a RAID parity disk. It is recommended to have multiple hot spare disks during steady state operations.
- **null** disks are unassigned.

- **EMS events**: for each node in the data set, the Event Management System (EMS) collects event data from various places of the Data ONTAP kernel and provides a set of filtering and event forwarding mechanisms. Specifically, the following information are collected:
  - Timestamp: records when an event occured.
  - Sequence id: a unique id for each event.
  - EMS event: the name of an event. The full list of EMS events can be found in EMS event catalog [16].
  - Severity index: categorizes each event based on its implication to system availability. The values range from 0 to 7 which translate to EMERGENCY, CRITICAL, ALERT, ERROR, WARNING, NOTICE, DEBUG, and INFO from the most severe to the least severe error event. Table II details each severity level.
  - Node name: the name of the storage node which reported this event. Each Workload consists of several nodes (systems), and the data is collected for each node separately.
  - Device name: represents the disk drive where an event happened.
  - Component name: the system component or service which generated the event (e.g., mgwd, sshd, statd).

Observation #1: The error occurrence frequencies for different error types are diverse. 48% of error types occur less than 10 times per year and 18% of error types happen only once a year. Error events are busrty and the error distribution is imbalanced and time-varying.

B. Error Type Distribution

There are 1,625 unique types of error events captured across the four workloads in the batch-2017 trace. Table I specifies the number of error types per workload in batch-2017. Without loss of generality, we choose Workload A to explain our findings. Workload A has 1,369 unique types of error events across 16 nodes. Out of the 1369 error types, 543 (40%) error types appear in more than 5 nodes, while 398 (29%) types occur in 2 to 5 nodes. The remaining 428 types (31%) only appear in one of the nodes. Only 32 error types (2.3%) are commonly observed in all 16 nodes.

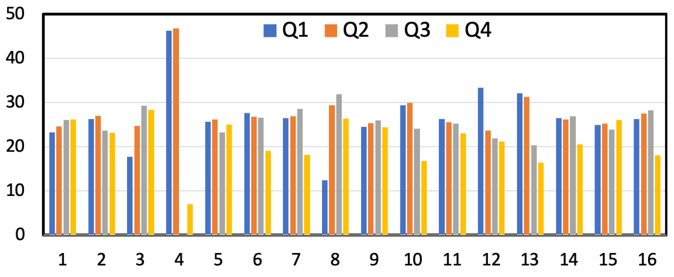


Fig. 1: Distribution of error per node across quarters. Each color represents a quarter of a year. x-axis is the node index and y-axis is the percentage of error.

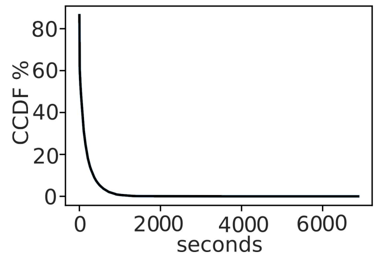


Fig. 2: Complementary Cumulative Distribution Function (CCDF) of the inter-occurrence time of error events in Workload A.

We divide the entire data collection time period into 2-hour time bins to compute the error density, i.e., if an error happens in a 2-hour time bin, count as 1, otherwise 0. The density results are summarized in Table II. The density of all error types in Workload A is 1.1%, which indicates that the error events are sparsely distributed. The error is even sparser for the most severe group of errors with a density of 0.18%. In addition, the more severe the error type, the more sparse and rare they are, i.e., only 70 out of 1,369 error types are from Emergency, Alert, and Critical error levels, belonging to

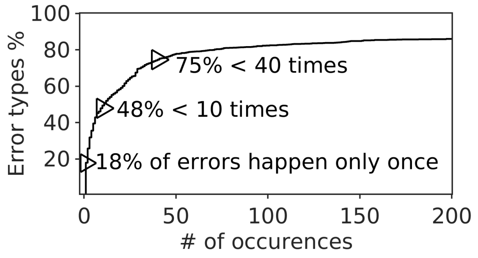


Fig. 3: The occurrence distribution of error types occurred less than 200 times. y-axis is the percentage of error types occurred less than the value represented in x-axis.

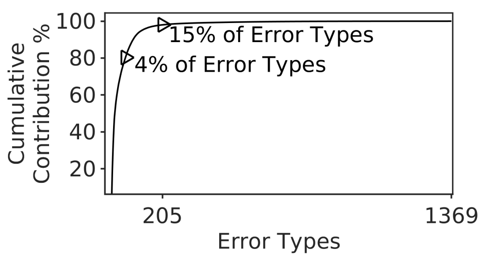


Fig. 4: Contribution of error types to the overall observed errors. 98% of all observed errors in batch-2017-WorkloadA are from 205 error types (15%).

severity Group 1. We also illustrate how the error distribution changes for each quarter of a year in Fig. 1. In Fig. 1, each color represents each quarter (3 months) and the x-axis is the index of node while y-axis presents the percentage distributed in each quarter. Fig. 1 clearly demonstrates that the errors are not evenly distributed across time and nodes. Such dynamics makes the training of a prediction model challenging, which motivate us to develop a training-free approach that is less vulnerable to such dynamics. Furthermore, we analyze the occurrence pattern of error events and plot the inter-occurrence time distribution of error events in Fig. 2, where the long tail distribution indicates that error events tend to occur in bursts.

Some of the most severe error types are very rare. For instance, the Emergency level error *callhome.partner.down* that indicates the storage fail-over partner is down and takeover cannot begin, happened only once across all 16 nodes in one year. In comparison, the most common error type in the trace we collected is *kern.uptime.filer*, which is informational and typically occurs every hour (i.e., more than 66k times in Workload A). Fig. 3 illustrates the occurrence distribution of error types occurred less than 200 times. It is observed that almost half of error types (48%) occur less than 10 times, while 18% of error types occur only once.

Observation #2: Severe error events occur less frequently. The average density of severe error events (i.e., Emergency, Alert, and Critical) during a 2-hour period is 0.18%, which is far less than the average density (0.77%) and the accumulated density (1.1%) of all error types.

Furthermore, we observe from the skewed curve in Fig. 4 that 80% of all error events are only from 4% of error types. This suggests a power law distribution of error types, in which 87 out of 1,369 error types in Workload A are responsible for more than 90% of all events.

Observation #3: The distribution of error event types follows a power law. 4% of error types are correspondence with 80% of all error events, while 96% of error types account for only 20% of events.

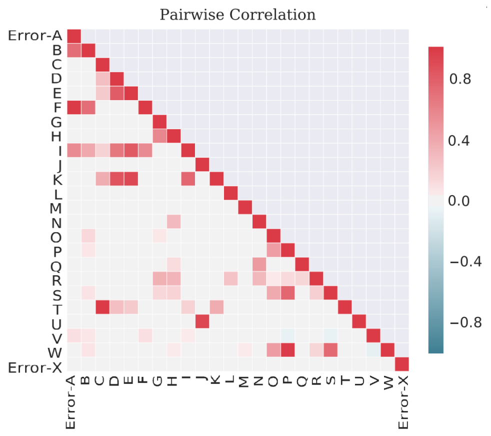
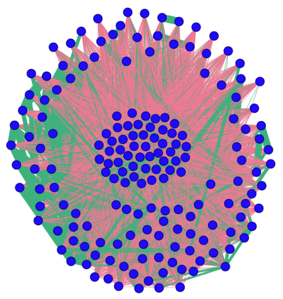


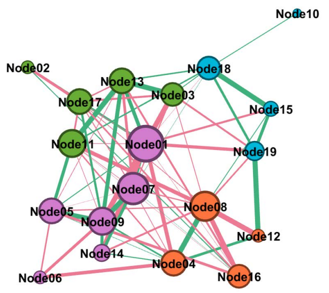
Fig. 5: Cross-correlation heat map for a subset of event types in Workload A that occur most frequently. The x-axis and the y-axis contain the same list of error types and the grid shows the correlation between various error type pairs. As the heat map is symmetric, only half of the map is provided.

C. *Correlation Analysis of Storage Error*

We analyze the complex correlation of the multi-dimensional error data, which is useful to reduce the computational complexity of SEFEE (see Section IV-E). A heatmap of cross-correlation between a subset of error events that occur most frequently is given in Fig. 5. Many error pairs are highly correlated. To visualize the error cross-correlation, we generate a graph as $G_t = (N, E, C)$, where error type $e \in E$ and storage node $n \in N$ are the two sets of vertices, and count C is the edge, which exists if error e has happened at storage node n at time t. To understand the error cross-correlation, we visualize the entire trace as a graph using pairwise cross-correlation (CC) between errors to represent the edge and the errors to represent vertices in a graph $G_e = (E, CC_{e_{ij}})$. To understand the storage node auto-correlation, we perform similar graph generation from a system (i.e., node) viewpoint: $G_n = (N, CC_{n_{ij}})$. Fig. 6a and Fig. 6b show storage error and storage node cross-correlation network for the Cluster A of batch-2016. We first visualize the cluster A of the batch-2016 data as its scale is smaller and thus easier to see the details. We also show the storage node cross correlation network for batch-2017 Workload A in Fig. 7.



(a) Cross-correlation network of Storage errors. Vertices are storage error types.



(b) Storage node cross-correlation network. Vertices are storage nodes.

Fig. 6: Visualization of Cluster A of batch-2016 trace with 175 error types and 19 nodes. Edges are non-zero cross-correlation values and the edge weight (i.e., line width) is the absolute value of cross-correlation. Green and pink edges represent positive and negative cross-correlation values, respectively. Nodes with the same color indicate a community grouped based on Modularity.

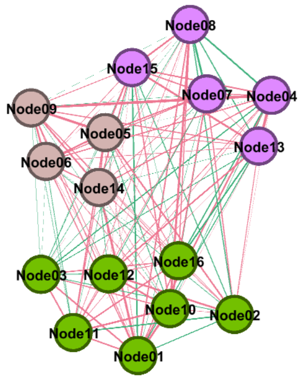


Fig. 7: Storage node cross-correlation network for Workload A of the Batch-2017 trace. Edges are non-zero cross-correlation values and the edge weight (i.e., line width) is the absolute value of cross-correlation. Green and pink edges represent positive and negative cross-correlation values, respectively. Vertices with the same color indicate a community grouped based on Modularity.

We use Modularity [17] to quantify the strength of the community structure among the vertices of the graph (i.e. storage node or system). Modularity compares the connections within a group of vertices with the expected connection to other vertices outside groups in a random network to determine whether there exists a community structure. Modularity measure is defined as

$$\mathbf{Q} = \sum_i (e_{ii} - a_i^2),$$

where e_{ii} gives fraction of within-community edges and a_i gives fraction of edges that connect to vertices in community i. In a fully random network, the number of within-community connections is no better than random, resulting in Q = 0. A network with strong community structure typically fall between 0.3 to 0.7 modularity, while a network with community structure above 0.7 is rare. The storage node cross-correlation network of Workload A has modularity score of Q = 0.102, which indicates weak community structure among the storage nodes. As we picked the systems that run the same workload but each system could be deployed at different customers, we anticipate there is only weak correlation between these storage nodes and the results validate our assumption.

Such community structure provides only coarse-grained information (e.g., which nodes are more related than others) that may not be sufficient to help per error prediction. The fine-grained correlation among nodes are very complex and difficult to be utilized directly for deriving analytical models or extracting features for crafting machine learning models. More importantly, such graph evolves over time, which makes it even more difficult to capture the spatial-temporal inter-dependency among nodes' error-event logs. Despite the challenge of utilizing the community structure directly for error prediction, in Section IV-E, we demonstrate how such community structure can be used to accelerate the computation of SEFEE by decomposing the big problem into parallel-able small problems.

Observation #4: There is a complex spatio-temporal inter-dependency between the location and the type of errors.

III. Methodology

To capture the complex spatio-temporal inter-dependency among different nodes' error-event logs, we construct the tensor model of error-event logs. Based on the analysis in Section II and the tensor model, we propose a lightweight storage error forecasting method SEFEE (Storage Error Forecasting using tEnsor dEcomposition) for online storage error prediction using tensor decomposition.

A. *Tensor Model for Storage Error-Event Logs*

Tensors are multidimensional or multi-way arrays that are higher-order generalization of matrices (i.e., second-order tensor) and vectors (i.e., first-order tensor). In this paper, the error-event logs are constructed as a three-way tensor $\mathcal{X} \in \mathbb{R}^{I\times J\times K}$ with $\mathcal{X}_{ijk}$ denoting the (i, j, k)-th entry of $\mathcal{X}$ such as

$$\mathcal{X}_{ijk} = \begin{cases} 1 & \text{if error event } j \text{ happened at node } i \text{ at time } k \\ 0 & \text{otherwise} \end{cases}.$$

As illustrated in Fig. 8, I corresponds to the number of nodes in the system, J corresponds to the types of errors, and K corresponds to the time period of the logs.

At a given time t, the observed error-event logs across all the nodes can be denoted as $\mathcal{X}_t$, which is a slice of $\mathcal{X}$ at time t. The error-event logs can be treated as a time-series of these slices. To predict storage error, we can use the observed tensor $\mathcal{X}^{obs}$ to predict future slices.

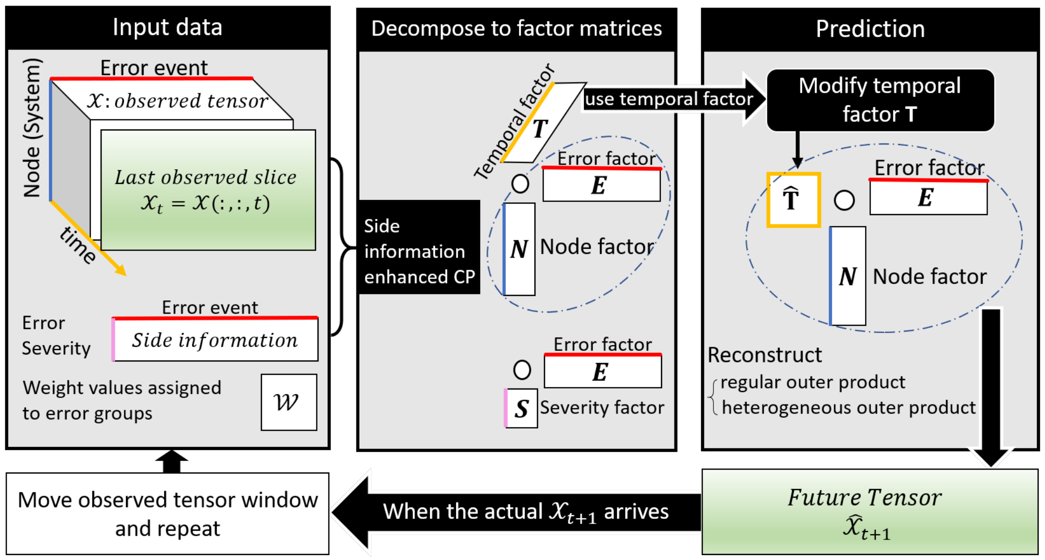


Fig. 8: SEFEE overview.

B. SEFEE

Based on the tensor model, we propose a lightweight storage error forecasting method SEFEE for online storage error prediction using tensor decomposition. The overview of the proposed method is illustrated in Fig. 8. The basic idea is to construct the observed tensor $\mathcal{X}^{obs}$ using recent error-event logs and then predict the future errors using the tensor decomposition of $\mathcal{X}$ enhanced by side information (i.e., error severity).

Specifically, let $\mathcal{X} \in \mathbb{R}^{I\times J\times K}$ denote the tensor containing both the observed tensor $\mathcal{X}^{obs} \in \mathbb{R}^{I\times J\times G}$ where $G < K$ and the slices to be predicted. The prediction horizon h is $K-G$. For example, when $h = K-G = 1$, it means that the method predicts the errors in the next slice, i.e., 2 hour ahead. Let $\mathbf{Y} \in \mathbb{R}^{J}$ denote the error severity vector and $\mathcal{W} \in \mathbb{R}^{I\times J\times K}$ denote the weight tensor that is set according to the error severity, where the weights associated with a specific error type have the same weight, i.e., for a given error type j, $\mathcal{W}_{ijk}$ is the same for any i and k. The online storage error prediction is formulated as a side information enhanced weighted tensor decomposition problem:

$$\begin{array}{ll} \text{minimize} & ||\mathcal{W} * (\mathcal{X} - \hat{\mathcal{X}})||_{\mathbf{F}}^2 + ||\mathbf{Y} - \mathbf{E}\mathbf{S}||_{\mathbf{F}}^2 \\ \text{subject to} & \hat{\mathcal{X}} = [\![\mathbf{N}, \mathbf{E}, \mathbf{T}]\!], \\ & \mathcal{X}_{ijk} = \mathcal{X}_{ijk}^{obs}, i = 1, ..., I, j = 1, ..., J, k = 1, ..., G, \\ \text{variables} & \{\mathbf{N}, \mathbf{E}, \mathbf{T}, \mathbf{S}, \mathcal{X}\} \end{array} \tag{1}$$

where $*$ corresponds to item-wise multiplication, $||\cdot||_F$ is the Frobenius norm, and $[\![\cdot]\!]$ corresponds to CANDECOMP/PARAFAC (CP) decomposition operator [18].

In the problem (1), $\hat{\mathcal{X}}$ denotes the predicted tenor using CP decomposition with factorization matrices $\mathbf{N} = [\mathbf{n}_1, \mathbf{n}_2, ..., \mathbf{n}_R] \in \mathbb{R}^{I\times R}$, $\mathbf{E} = [\mathbf{e}_1, \mathbf{e}_2, ..., \mathbf{e}_R] \in \mathbb{R}^{J\times R}$, and $\mathbf{T} = [\mathbf{t}_1, \mathbf{t}_2, ..., \mathbf{t}_R] \in \mathbb{R}^{K\times R}$ such that

$$\hat{\mathcal{X}} = [\![\mathbf{N}, \mathbf{E}, \mathbf{T}]\!] = \sum_{r=1}^{R} \mathbf{n}_r \circ \mathbf{e}_r \circ \mathbf{t}_r, \tag{2}$$

where $\circ$ corresponds to the vector outer product and $R > 0$ is a positive integer representing the rank of $\hat{\mathcal{X}}$. $\mathbf{N}$ and $\mathbf{E}$ capture the spatial correlation of errors across different nodes, while temporal profiles are captured by $\mathbf{T}$. The error severity $\mathbf{Y}$ is introduced to enhance the CP decomposition such that the factorization matrix $\mathbf{E}$ should also capture the error severity $\mathbf{Y}$, which is captured in $||\mathbf{Y} - \mathbf{E}\mathbf{S}||_{\mathbf{F}}^2$, where $\mathbf{S} \in \mathbb{R}^{R}$ is a factorization vector for $\mathbf{Y}$. The problem (1) can be solved using CP-ALS methods [19], where the factorization matrices are alternatively updated based on $\mathcal{X}^{obs}$. It is worth noting that the complexity of solving (1) depends on the size of tensor $\mathcal{X}$, which can be large in practice. To reduce the computational complexity, we leverage the correlation analysis in Section II-C to decompose the whole tensor into multiple subtensors without sacrificing the prediction performance, which is validated by the experimental results in Section IV-E.

Remarks: SEFEE does not require the label intensive and computational expensive pre-processing and training tasks (e.g., data cleaning, pruning, feature selection, model architecture crafting, hyper-parameter tuning, iterative training) that are typically required by machine learning algorithms. Thus, it offers a lightweight forecasting method that is training free and can be used in a plug-n-play fashion.

C. Online Storage Error Prediction

Based on the factorization matrices solved in (1), we propose an online storage error prediction:

$$\hat{\mathcal{X}}^h = \sum_{r=1}^{R} \mathbf{n}_r \circ \mathbf{e}_r \circ \hat{\mathbf{t}}_r, \tag{3}$$

where $\hat{\mathcal{X}}^h \in \mathbb{R}^{I\times J\times h}$ denotes the predicted tensor with the prediction horizon h and $\hat{\mathbf{T}} = [\hat{\mathbf{t}}_1, \hat{\mathbf{t}}_2, ..., \hat{\mathbf{t}}_R] \in \mathbb{R}^{h\times R}$ is a modified factorization matrix based on $\mathbf{T}$. In the following, we propose different ways to obtain $\hat{\mathbf{T}}$ in an online manner, as applying a single prediction strategy globally may not well capture the heterogeneity of the error-event logs.

1) Persistence Prediction: The persistence prediction assumes that the temporal profiles do not change, i.e., the future tensor slice is the same as the last observed slice. To ease the presentation, let $\mathbf{T}_{j:}$ denote the j-th row of $\mathbf{T}$. Under the persistence prediction, each row of $\hat{\mathbf{T}}$ is equal to the G-th row of $\mathbf{T}$, i.e.,

$$\hat{\mathbf{T}}_{j:} = \mathbf{T}_{G:},\ j = 1, ..., h. \quad (4)$$

2) Moving Average: The moving average method leverages the average of the temporal component $\mathbf{T}$ in a time window W to calculate each row of $\hat{\mathbf{T}}$. Specifically, we calculate the first row of $\hat{\mathbf{T}}_{1:}$ by

$$\hat{\mathbf{T}}_{1:} = \frac{1}{W}(\mathbf{T}_{G:} + \mathbf{T}_{G-1:} + \cdots + \mathbf{T}_{G-W+1:}). \quad (5)$$

Then we can recursively solve the remaining rows of $\hat{\mathbf{T}}$ by

$$\hat{\mathbf{T}}_{j+1:} = \frac{1}{W}(\sum_{i=1}^{W-j} \mathbf{T}_{G-i+1:} + \sum_{i=1}^{j} \hat{\mathbf{T}}_{i:}),\ j = 1, ..., h-1. \quad (6)$$

3) Moving Average under Time-of-day Effect: This method accounts for time-of-day effect in the moving average method, as we observe time-of-day effect in the error-event logs. For example, some benign error types such as licence renewal notifications might have a daily routine at a particular time of day, while other error types might be very sporadic and random. Specifically, we introduce a time-of-day parameter t_d. When taking average over the last W time steps, we average over W time steps according to t_d and calculate the first row of $\hat{\mathbf{T}}_{1:}$ by

$$\hat{\mathbf{T}}_{1:} = \frac{1}{W}(\mathbf{T}_{G:} + \mathbf{T}_{G-t_d:} + \mathbf{T}_{G-2t_d:} + \cdots + \mathbf{T}_{G-Wt_d+1:}). \quad (7)$$

Then we can recursively solve the remaining rows of $\hat{\mathbf{T}}$ by

$$\hat{\mathbf{T}}_{j+1:} = \frac{1}{W}(\sum_{i=1}^{W-j} \mathbf{T}_{G-it_d+1:} + \sum_{i=1}^{j} \hat{\mathbf{T}}_{i:}),\ j = 1, ..., h-1. \quad (8)$$

Heterogeneous Time-of-day Effect. Due to the heterogeneity of the error-event logs, we compute t_d for each ⟨node, error⟩pair. Let $\mathbf{T_d}$ denote a time-of-day matrix where $T_d(i,j)$ is the time-of-day coefficient associated with error type j at node i. We treat the error-event logs for each ⟨node, error⟩pair as a time series and use auto-correlation function (ACF) to determine $T_d(i,j)$ for each pair. Based on our experiments, $\mathbf{T_d}$ does not change much in a short period and therefore we do not need to update $\mathbf{T_d}$ at every single prediction step. In our experiments, $\mathbf{T_d}$ is updated every week.

In order to accommodate individually computed time-of-day effects in reconstruction of predicted tensor from the factorization matrices, we need to modify the vector outer product used to yield the predicted tensor as in (3). The details are given in Algorithm 1, where we modify the regular outer product to treat each ⟨node, error⟩pair differently using its time-of-day effect.

Algorithm 1: Moving Average under Heterogeneous Time-of-day Effect

```
input : N, E, T, and Time-of-day matrix $\mathbf{T_d}$.
output: $\hat{\mathcal{X}}^h$
1 foreach node $i$ in $size(\boldsymbol{N}, 1)$ do
2     foreach error $j$ in $size(\boldsymbol{E}, 1)$ do
          // compute temporal factor for
             node i and error j
3         $t_{last} = size(\mathbf{T}, 1)$;
4         $u = t_{last} + 1 - T_d(ij) * W$;
5         for $k = t_{last} + 1 - T_d(ij) : -T_d(ij) : u$ do
6             $sum(\mathbf{T}(k))$;
7         $\hat{t}_{ij} = \frac{1}{W} sum(\mathbf{T}(k))$;
          // reconstruct each element in
             the predicted tensor
8         $\hat{\mathcal{X}}^h_{ij} = outerProduct\ (N[i,:]\ ,\ E[j,:]\ ,\ \hat{t}_{ij})$;
```

IV. Error Prediction Results

A. Experimental Setup

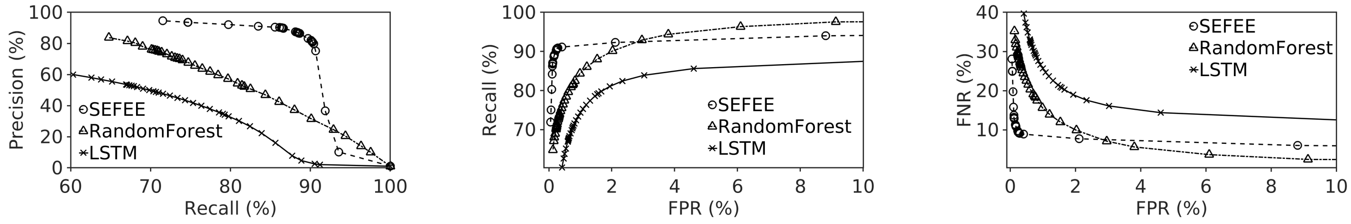
1) SEFEE: We prototype SEFEE and use the collected batch-2017 trace for evaluation. As SEFEE is training-free, we use only the last week of the first 6 months as the initial observations to warm up our method. We use the rest 6 months for live predictions (i.e., 2195 predictions for two-hour time bin). We set $R = 300$ as the rank to decompose the observed tensor with dimension $(16 \times 1369 \times 84)$.

2) Baselines: We use LSTM and Random Forests as baselines because they are consistently found to have the highest storage error prediction accuracy in recent literature [9, 20, 21] For LSTM, we use Keras library [22] with tensorflow backend. For Random Forests, we use Multivariate Random Forests [23]. We use the first 6 months to train LSTM and Random Forests models and the rest 6 months for evaluation. For Random Forests, we experiment with different numbers of trees from 10 up to 120 trees and observe no significant improvement beyond 30 trees. The LSTM model uses stacked bidirectional cells, as we found that it achieves higher accuracy than the vanilla LSTM. As LSTM and Random Forests have lots of hyperparameters, to make a fair comparison, we perform extensive fine-tuning on the hyperparameters of LSTM and Random Forests to ensure their prediction performance. We also use Hidden Markov model (HMM) and ARIMA with moving windows as reference approaches for the fine-tuning.

3) Evaluation Metrics: In the literature of storage failure prediction, it is common to use error detection rate (i.e., True Positive Rate (TPR)) along with false alarm rate (i.e., False Positive Rate (FPR)) as prediction accuracy metric. More recent papers using machine learning based approaches often use Precision, Recall, and F1-measure as alternative measures.

TABLE III: Storage error prediction performance comparison among SEFEE, LSTM, and Random Forests for Worklaod A in batch-2017. Group is the error severity level defined in Table II.

	SEFEE				LSTM				Random Forests			
	Precision	Recall	F1	FPR	Precision	Recall	F1	FPR	Precision	Recall	F1	FPR
Group 1	90.3%	89.9%	90.1%	0.02%	27.4%	37.7%	31.7%	0.24%	59.2%	31.7%	41.3%	0.05%
Group 2	81.6%	85.3%	83.4%	0.12%	43.6%	57%	49.4%	0.47%	57.9%	73%	64.6%	0.34%
Group 3	74.5%	74.4%	74.4%	0.2%	33.4%	48.2%	40%	0.75%	52.2%	51.4%	51.8%	0.36%
Group 4	89%	91%	90%	0.15%	57.9%	77.4%	66.2%	0.75%	78.5%	80%	79.2%	0.3%
Overall	85.6%	87.2%	86.4%	0.15%	51%	69%	58.6%	1%	70%	73.6%	71.8%	0.33%



(a) PRC curve. (b) ROC curve. (c) FPR versus FNR.

Fig. 9: Prediction performance comparison between SEFEE, Random Forests, and LSTM in terms of: (a) PRC curve (high precision and high recall is desired), (b) ROC curve (high recall and low FPR is desired), and (c) FPR versus FNR (low FNR and low FPR is desired).

TABLE IV: Prediction performance comparison between SEFEE and Random Forests for four different workloads of batch-2017.

	SEFEE			Random Forests		
	Precision	***Recall***	***F1***	***Precision***	***Recall***	***F1***
Workload A	85.6%	87.2%	86.4%	70%	73.6%	71.8%
Workload B	88.8%	90.6%	89.7%	72.5%	80.8%	76.4%
Workload C	80.4%	83.5%	81.9%	52.5%	72.8%	61%
Workload D	90.1%	90.9%	90.5%	75.6%	85.1%	80.1%
Overall	86.2%	88%	87.1%	67.65%	78.1%	72.3%

Therefore, we mainly use Precision, Recall (i.e., TPR), F1-measure along with FPR in our evaluation. The metrics are defined below:

$$\begin{aligned} Precision &= \frac{\#truepositives}{\#truepositives + \#falsepositives}, \\ Recall &= \frac{\#truepositives}{\#truepositives + \#falsenegatives}, \\ F1\ score &= 2 \times \frac{precision \times recall}{precision + recall}, \\ FPR &= \frac{\#falsepositives}{\#falsepositives + \#truenegatives}. \end{aligned} \tag{9}$$

B. Accuracy of Storage Error Forecasting

The prediction results of Workload A are given in Table III, which corroborates that SEFEE outperforms LSTM and Random Forests in terms of Precision, Recall, F1, and FPR. For example, in the per error group break down results, SEFEE is at least 31% and 11% better than LSTM and Random Forests, respectively. For the overall performance, SEFEE is at least 18% and 14% better than LSTM and Random Forests, respectively. SEFEE also excels in achieving a balanced prediction accuracy across error severity groups. In practice, recall is a very important measure for groups of higher severity, as immediate actions are required to handle these errors. However, groups of higher severity are also of lower data density, which makes it more challenging to train good LSTM and Random Forests models.

The recall results in Table III shows that SEFEE achieves much higher prediction accuracy on Group 1, compared to LSTM and Random Forests. We also compare the results between SEFEE and Random Forests for all the workloads (Workload A - D) in Table IV, where SEFEE consistently outperforms Random Forests. In the interest of space, we omit the results for LSTM as the LSTM prediction performance is consistently worse than Random Forests.

Prediction methods provide the occurrence probability of an error event. To determine whether there is an error event, a triggering probability threshold needs to be set. This threshold can be considered as a hyperparameter to control the aggressiveness of prediction, which can be set based on the use scenarios. For example, a more aggressive prediction (lower triggering probability threshold) would result in a higher recall (i.e., more errors are captured) while a lower precision (i.e., more wrong guess in the forecast errors), as well as higher FPR (i.e., more false alarm) and a lower FNR (i.e., less fraction of errors missed). To demonstrate the effectiveness of SEFEE under different prediction aggressiveness, we plot three curves in Fig. 9: Precision/Recall (PRC), Receiver Operator

TABLE V: Improvement in % with side information compared to without side information. Negative value means degraded performance. Group is the error severity level defined in Table II.

	Improvement			
	Precision	Recall	F1	FPR
Group 1	+2.1%	+5.9%	+4%	-0.01%
Group 2	+2%	-0.4%	+0.9%	-0.02%
Group 3	-0.2%	+1.1%	+0.4%	+0.01%
Group 4	-0.3%	+0.4%	+0.1%	0%
Overall	+0.3%	+0.5%	+0.4%	0%

TABLE VI: Prediction accuracy comparison for "Alert" severity level of errors between "before" using weighted approach and "after" using weighted approach. S indicates the error severity index as outlined in Table II, and W denotes the weights of each severity level.

S	Recall		#False alarms		F1-score		W
	before	after	before	after	before	after	
0	93.6%	91.1%	17	21	92.6%	90.5%	1
1	**82.6%**	**85.7%**	**18**	**18**	**86.2%**	**87.9%**	**4**
2	NaN	NaN	2	6	NaN	NaN	1
3	85.3%	84.1%	416	514	82.8%	80.3%	1
4	85.2%	83.6%	166	189	83.7%	81.8%	1
5	69.5%	68.2%	1070	1154	69.5%	67.8%	1
6	88.5%	87.3%	1050	1102	85.7%	84.7%	1
7	92.8%	91.8%	935	1130	91.7%	90.3%	1

Characteristics (ROC), and FPR versus FNR [24], all of which are often employed by the literature for evaluating the quality of prediction methods. Fig. 9 (a) illustrates the PRC curve, where only SEFEE can achieve both high precision and high recall. Fig. 9 (b) illustrates the ROC curve, where it is desirable to detect more errors while not introducing many false alarms. When FPR is less than 3%, SEFEE yields much higher recall and significantly outperforms LSTM and Random Forests. Fig. 9 (c) illustrates the results of FPR versus FNR, where it is desirable to have both low FPR and low FNR. When FPR is less than 3%, SEFEE yields much lower FNR than LSTM and Random Forests.

C. Effectiveness of Side Information

In Table V, we evaluate the effectiveness of side information by comparing the prediction results with and without side information. It is observed that the side information is especially useful for the most severe error types (i.e., Group 1 with Emergency, Alert, and Critical level) as there is about 6% improvement in recall. This is because the side information can provide useful information to enhance the tensor decomposition and thereby improve the prediction accuracy, especially for groups of higher severity that occur less frequently.

D. Impact of Weights on Prediction Accuracy

In this section, we evaluate the impact of weights on the prediction accuracy, which can provide prioritized improvements of prediction performance for target groups. We perform experiments for a 200-hour time period (i.e., 100 time-steps) and weighted the severe error type (i.e., "Alert" level error events) by a weight of 4 compared to others as 1. Table VI shows the prediction performance.

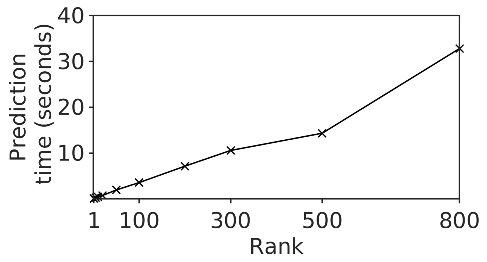


Fig. 10: Prediction time under different tensor ranks.

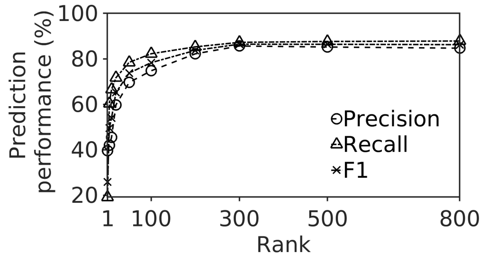


Fig. 11: Prediction performance under different tensor ranks.

The results show that the recall and F1-score of the "Alert" error type has been improved, while false alarm keeps the same. However, to achieve this, the prediction performance of other severity level has been slightly scarified. In practice, the system operator can adjust the weights based on the needs to improve the prediction accuracy for certain error types.

E. Prediction Overhead

To use the proactive error forecasting approach in practice, it is critical to have low computational resource consumption as well as low training and prediction time. In this section, we compare the prediction overhead of SEFEE with LSTM and Random Forests.

First of all, we evaluate SEFEE on a very outdated personal computer with Intel Xeon E3-1225 v3 3.2GHz 4C 84W CPU and 4 GB RAM. SEFEE takes on average 26 seconds per prediction utilizing between 1.8-2.5 GB of RAM. These results show that SEFEE is lightweight and does not demand a significant amount of CPU or memory resources for making prediction, which allows it to be easily deployed in a real storage system.

For example, a storage controller such as NetApp A700 has a powerful 36-core CPU. Though the main priority of storage controller is to serve storage workloads, running SEFEE would not put much pressure on the controller.

As LSTM and Random Forests are very computation resource demanding, our outdated personal computer could not finish the experiments in a reasonable time. Therefore, we perform experiments using an Amazon Web Services instance

TABLE VII: Prediction time comparison between using the whole tensor versus using subtensors. "Subtensors Combined" computes subtensors in a sequential order, while "Parallel Subtensors" computes subtensors in parallel.

	Whole Tensor	Subtensor1	Subtensor2	Subtensor3	Subtensors Combined	Parallel Subtensors
Dimension	(16,1369,84)	(7,1043,84)	(4,752,84)	(5,1064,84)		
Rank	300	180	120	150		
Precision	89.2%	89.25%	91.65%	87.6%	89.5%	89.5%
Recall	90.2%	90.7%	93.8%	87.3%	90.6%	90.6%
Time (seconds)	26	5.37	2.4	3.1	10.9	5.37

with 36 Intel Xeon vCPUs and 72 GiB of RAM, which has very similar specs as the storage controller of NetApp A700. The LSTM took more than 8 hours to finish training, while Random Forests took less than 7 hours. Once these models are trained, they can be used for online prediction. In comparison, SEFEE makes the prediction in less than 10 seconds without training using the same instance.

The most time consuming components of SEFEE are tensor decomposition and reconstruction of future tensor, the time complexity of which mainly depends on the tensor rank used for the decomposition. In addition, the rank also impacts the prediction accuracy. To better understand how the rank impacts the prediction time and prediction accuracy, we demonstrate the average time for each prediction as a function of rank in Fig. 10 and prediction accuracy as a function of rank in Fig. 11. In a nutshell, the prediction time increases linearly with the rank, while the accuracy increases significantly at small ranks but flatten out when the rank is around 300. In our experiments, we choose the rank equal to 300 for striking a balance between computation time and prediction accuracy.

To further reduce the computation time, we divide the tensor data into three different subtensors using the node groups in Fig. 7, which are generated using Gephi [25] that employs Modularity [26]. Specifically, the 16 nodes are divided into three groups with 4, 5, and 7 nodes, respectively; the error data for each group is constructed as a subtensor. For each subtensor, we carry out the same analysis as in Fig. 11 to get an estimate of tensor rank. We compare the performance of using these subtensors with the whole tensor in Table VII. The results show that it takes more than 58% less time (i.e., 10.9 seconds) on outdated personal computer for each prediction when using these subtensors compared to the whole tensor. By running the three subtensors in parallel on separate machines, the prediction time can be further reduced to 5.37 seconds per prediction.

F. Intuition of Why Tensor Approach is Effective

The intuition behind why the tensor approach is effective is that it can automatically discover and utilize the complex latent spatio-temporal inter-dependency between the location and the type of errors. To further understand why the prediction is more accurate in some cases than others, we plot the auto-correlation (ACF) of error trace for each (node, error) pair against the accuracy (recall) of that pair in Fig. 12.

We divide Fig. 12 into nine regions based on the low, medium, and high value of accuracy and auto-correlation. We

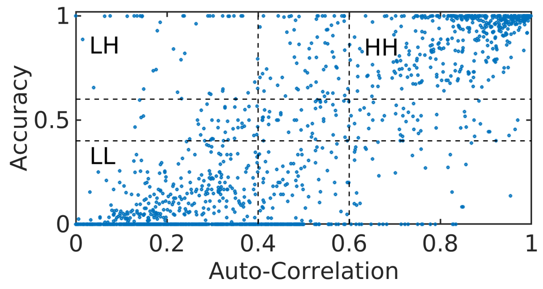


Fig. 12: Auto-correlation vs. Accuracy (recall). Each data point is generated by a (node, error) pair. x-axis represents auto-correlation (ACF) and y-axis represents Accuracy (recall).

can see there are two regions with the highest density of dots, namely HH region *(high ACF, high Recall)* (i.e., the top right region) and LL region *(low ACF, low Recall)* (i.e., the bottom left region). This indicates that the tensor approach essentially explores the complex latent spatio-temporal inter-dependency for prediction. When there is relatively high dependency, the prediction is more accurate, vice versa. Another interesting observation is that there are some points fall in the LH region *(low ACF, high Recall)* (i.e., the top left region). We found that most of the points in this region have some spatial or logical connection with points in HH region. For example, for every LH point there is usually at least one HH point on the same node but of different error types, or the same error type on a different node. This suggests that the tensor approach is able to capture and utilize such deeply latent dependency information to improve the prediction performance.

G. Generalizability

Our tensor-based prediction method can effectively discover the complex spatio-temporal correlation among a significant number of elements. With the assist of domain specific knowledge, the computation of tensor decomposition can be significantly reduced using the proposed approach. Therefore, it is generalizable to other use cases with similar properties. Take HPC storage systems as an example, since the main observations that drive our design is based on the correlation analysis in Section II, such as the heterogeneous correlations among different error types at different storage nodes across time. If HPC storage systems share such similar characteristics, the proposed approach can be applied to HPC storage systems as well.

V. Related Work

A. Storage Failure

1) Reactive approaches: Kubo et al. [3] proposed a threshold-based reactive approach, in which a functioning storage device is taken offline when the number of times that the device fails to perform a storage operation exceeds a predetermined threshold. [27] identifies RAID groups that are at risk of failure and moves them in advance based on the observation that the high count of reallocated sectors is a sign of impending failure. Xu et al. [28] proposed to prevent storage failures in distributed systems by proactively detecting latent configuration errors early through analyzing the source code of the relevant system components. Kadekodi et al. [29] proposed an online tuning tool, namely HeART, to help make cost-effective redundancy settings for long-term data reliability. HeART reduces HDDs in duty by 11-33% fewer HDDs to reduce the likelihood of failures. This line of work usually results in relatively long system down time and is not suitable for large-scale enterprise storage systems.

2) Statistical and machine learning based proactive approaches: Hamerly and Elkan [30] proposed prediction models using NBEM and naive-Bayes based classifier to improve the detection accuracy of the SMART monitoring system. Hughes et al. [31] proposed Wilcoxon rank-sum test based models to improve disk-drive failure prediction. [32] found that rank-sum tests outperform SVM when certain small subset of SMART attributes are used, with accuracy of 25% and False alarm rate of 0, while SVM performs better (43% failure detection rate with 0.6% FAR) when more than four attributes are used. Murray and Hughes [33] found that SVM performs better than statistical hypothesis tests, mi-NB, and unsupervised clustering with failure detection rate of 50.6% and 0% false alarm.

Mahdi Soltani et al. [9] proposed different machine learning techniques such as SVM, CART, and neural network to predict partial drive failures. Their results show that Random Forests outperform the rest by correctly predicting 70-90% of the errors at false positive rate of 2%. Xu et al. [6] proposed an online prediction based on Fast-Tree algorithm which ranks disks based on the degree of error-proneness. Their results show that their approach outperforms SVM and Random Forests by achieving 41.2% Recall at FPR of 0.1%. After evaluating various classification models, [21] employed Random Forests to predict fail-stop events in SSDs due to its better performance and robustness in presence of noisy inputs. Zhang et al. [34] developed DeepView to localize Virtual Hard disk failures to tackle VM availability issues in Microsoft Azure. A recent work [20] tried various machine learning approaches to predict SSD failures and concluded that Random Forests outperform the other approaches by achieving 90% ROC AUC. Lin et al. [7] proposed MING as a node failure prediction approach that uses LSTM to incorporate temporal features and Random Forests for spatial features.

From these literature, we conclude that Random Forests and LSTM are the most promising machine learning techniques for predicting storage failures and we choose these two models as our main baselines in experimental evaluation. In addition, our work is different from these literature as SEFEE is a lightweight training-free online approach and can predict all types of errors in terms of their expected occurrence time and location.

B. Tensor Decomposition

Papalexakis et al. [35] presented a survey of using tensors and tensor decomposition for data mining and data fusion. [18] is another review on tensor decomposition and its applications. Tensor related approaches have been used in Twitter interactions [36], wearable sensor data for assessing student performance [37], image processing [38], author collaboration history [39], urban computing [40], computer networks [41], and machine learning approaches [42]. Compared to matrix factorization based methods (e.g., [43]), the tensor decomposition method can explore more complicated correlation structures in high dimensional data, i.e., error event type, location, and occurrence timestamp in our application. Moreover, we leverage a contextual information of error severity index to obtain enhanced tensor decomposition. To the best of our knowledge, SEFEE is the first work exploring tensor decomposition for prediction of storage error events and leveraging the severity level as contextual information for improving the prediction accuracy.

VI. Conclusion

In this paper, we study the 2-year long error-event logs collected from the production NetApp ONTAP enterprise storage systems that manage 14,371 disks. By analyzing more than 23-million storage error events, we present SEFEE– a lightweight storage error forecasting method for large-scale enterprise storage systems. Different from literature, we propose a training-free methodology driven by tensor decomposition techniques for achieving superior prediction performance while also keeping the prediction overhead minimal. Our evaluation shows SEFEE significantly outperforms state-of-the-art machine learning based methods such as LSTM and Random Forests. Yet, SEFEE can provide prediction in just 5.73 seconds in legacy hardware, compared to 8 hours and 7 hours training time on high spec hardware using LSTM and Random Forests, respectively.

Acknowledgement

This work is supported in part by the following grants: National Science Foundation CCF-1756013, IIS-1838024 (using resources provided by Amazon Web Services as part of the NSF BIGDATA program), CNS-1950485, and EEC-1801727. We thank Shankar Pasupathy and Art Harkin from NetApp for their continuous support and feedback in making this project possible. We thank Gautham Yerroju for his help with the project. We thank also the anonymous reviewers for their insightful comments and suggestions that significantly improved the paper.

References

[1] NetApp. (2020) Netapp inc. [Online]. Available: https://www.netapp.com/us/products/storage-systems/hybrid-flash-array/fas8000.aspx

[2] 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 Transactions on Computers*, vol. 65, no. 11, pp. 3502–3508, 2016.

[3] R. A. Kubo, D. F. Mannenbach, and K. A. Nielsen, "Apparatus, system, and method for predicting storage device failure," Feb. 24 2009, uS Patent 7,496,796.

[4] "Finding soon-to-fail disks in a haystack," in *4th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage 12)*. Boston, MA: USENIX Association, Jun. 2012. [Online]. Available: https://www.usenix.org/conference/hotstorage12/workshop-program/presentation/Goldszmidt

[5] T. Pitakrat, A. van Hoorn, and L. Grunske, "A comparison of machine learning algorithms for proactive hard disk drive failure detection," in *Proceedings of the 4th International ACM Sigsoft Symposium on Architecting Critical Systems*, ser. ISARCS '13. New York, NY, USA: Association for Computing Machinery, 2013, p. 1–10. [Online]. Available: https://doi.org/10.1145/2465470.2465473

[6] 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 *2018 {USENIX} Annual Technical Conference ({USENIX}{ATC} 18)*, 2018, pp. 481–494.

[7] Q. Lin, K. Hsieh, Y. Dang, H. Zhang, K. Sui, Y. Xu, J.-G. Lou, C. Li, Y. Wu, R. Yao *et al.*, "Predicting node failure in cloud service systems," in *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. ACM, 2018, pp. 480–490.

[8] Y. Wang, Q. Miao, E. W. Ma, K.-L. Tsui, and M. G. Pecht, "Online anomaly detection for hard disk drives based on mahalanobis distance," *IEEE Transactions on Reliability*, vol. 62, no. 1, pp. 136–145, 2013.

[9] F. Mahdisoltani, I. Stefanovici, and B. Schroeder, "Proactive error prediction to improve storage system reliability," in *2017 {USENIX} Annual Technical Conference ({USENIX}{ATC} 17)*, 2017, pp. 391–402.

[10] B. Zhu, G. Wang, X. Liu, D. Hu, S. Lin, and J. Ma, "Proactive drive failure prediction for large scale storage systems," in *2013 IEEE 29th Symposium on Mass Storage Systems and Technologies (MSST)*. IEEE, 2013, pp. 1–5.

[11] S. Lu, B. Luo, T. Patel, Y. Yao, D. Tiwari, and W. Shi, "Making disk failure predictions smarter!" in *18th USENIX Conference on File and Storage Technologies (FAST 20)*. Santa Clara, CA: USENIX Association, Feb. 2020, pp. 151–167. [Online]. Available: https://www.usenix.org/conference/fast20/presentation/lu

[12] M. Hao, G. Soundararajan, D. Kenchammana-Hosekote, A. A. Chien, and H. S. Gunawi, "The tail at store: A revelation from millions of hours of disk and {SSD} deployments," in *14th {USENIX} Conference on File and Storage Technologies ({FAST} 16)*, 2016, pp. 263–276.

[13] H. S. Gunawi, R. O. Suminto, R. Sears, C. Golliher, S. Sundararaman, X. Lin, T. Emami, W. Sheng, N. Bidokhti, C. McCaffrey *et al.*, "Fail-slow at scale: Evidence of hardware performance faults in large production systems," *ACM Transactions on Storage (TOS)*, vol. 14, no. 3, p. 23, 2018.

[14] NetAppSupport. (2014) What syslog messages are. [Online]. Available: https://library.netapp.com/ecmdocs/ECMP1516135/html/GUID-363BF90A-FC09-4EF1-BAC6-43F315DFED68.html

[15] S. Maneas, K. Mahdaviani, T. Emami, and B. Schroeder, "A study of SSD reliability in large scale enterprise storage deployments," in *18th USENIX Conference on File and Storage Technologies (FAST 20)*. Santa Clara, CA: USENIX Association, Feb. 2020, pp. 137–149. [Online]. Available: https://www.usenix.org/conference/fast20/presentation/maneas

[16] NetAppKB. (2018) Ontap 9.3 ems event catalog. [Online]. Available: https://library.netapp.com/ecm/ecm_get_file/ECMLP2843561

[17] M. E. J. Newman and M. Girvan, "Finding and evaluating community structure in networks," *Phys. Rev. E*, vol. 69, p. 026113, Feb 2004. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevE.69.026113

[18] T. G. Kolda and B. W. Bader, "Tensor decompositions and applications," *SIAM review*, vol. 51, no. 3, pp. 455–500, 2009.

[19] B. W. Bader, T. G. Kolda *et al.*, "Matlab tensor toolbox version 3.1," Available online, Jun. 2019. [Online]. Available: https://www.tensortoolbox.org

[20] J. Alter, J. Xue, A. Dimnaku, and E. Smirni, "Ssd failures in the field: Symptoms, causes, and prediction models," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '19. New York, NY, USA: ACM, 2019, pp. 75:1–75:14. [Online]. Available: http://doi.acm.org/10.1145/3295500.3356172

[21] I. Narayanan, D. Wang, M. Jeon, B. Sharma, L. Caulfield, A. Sivasubramaniam, B. Cutler, J. Liu, B. Khessib, and K. Vaid, "Ssd failures in datacenters: What? when? and why?" in *Proceedings of the 9th ACM International on Systems and Storage Conference*, ser. SYSTOR '16. New York, NY, USA: Association for Computing Machinery, 2016. [Online]. Available: https://doi.org/10.1145/2928275.2928278

[22] F. Chollet, "keras," https://github.com/fchollet/keras, 2015.

[23] M. Segal and Y. Xiao, "Multivariate random forests," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 1, no. 1, pp. 80–87, 2011.

[24] J. Davis and M. Goadrich, "The relationship between precision-recall and roc curves," in *Proceedings of the 23rd international conference on Machine learning*. ACM, 2006, pp. 233–240.

[25] M. Bastian, S. Heymann, and M. Jacomy, "Gephi: An open source software for exploring and manipulating networks," 2009. [Online]. Available: http://www.aaai.org/ocs/index.php/ICWSM/09/paper/view/154

[26] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre, "Fast unfolding of communities in large networks," *Journal of Statistical Mechanics: Theory and Experiment*, vol. 2008, no. 10, p. P10008, oct 2008.

[27] 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 Transactions on Storage (TOS)*, vol. 11, no. 4, p. 17, 2015.

[28] T. Xu, X. Jin, P. Huang, Y. Zhou, S. Lu, L. Jin, and S. Pasupathy, "Early detection of configuration errors to reduce failure damage," in *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, 2016, pp. 619–634.

[29] S. Kadekodi, K. Rashmi, and G. R. Ganger, "Cluster storage systems gotta have heart: improving storage efficiency by exploiting disk-reliability heterogeneity," in *17th {USENIX} Conference on File and Storage Technologies ({FAST} 19)*, 2019, pp. 345–358.

[30] G. Hamerly, C. Elkan *et al.*, "Bayesian approaches to failure prediction for disk drives."

[31] G. F. Hughes, J. F. Murray, K. Kreutz-Delgado, and C. Elkan, "Improved disk-drive failure warnings," *IEEE transactions on reliability*, vol. 51, no. 3, pp. 350–357, 2002.

[32] J. F. Murray, G. F. Hughes, and K. Kreutz-Delgado, "Hard drive failure prediction using non-parametric statistical methods," in *Proceedings of ICANN/ICONIP*, 2003.

[33] 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*, vol. 6, no. May, pp. 783–816, 2005.

[34] Q. Zhang, G. Yu, C. Guo, Y. Dang, N. Swanson, X. Yang, R. Yao, M. Chintalapati, A. Krishnamurthy, and T. Anderson, "Deepview: Virtual disk failure diagnosis and pattern detection for azure," in *15th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 18)*, 2018, pp. 519–532.

[35] E. E. Papalexakis, C. Faloutsos, and N. D. Sidiropoulos, "Tensors for data mining and data fusion: Models, applications, and scalable algorithms," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 8, no. 2, p. 16, 2017.

[36] M. Araujo, P. Ribeiro, H. A. Song, and C. Faloutsos, "Tensorcast: forecasting and mining with coupled tensors," *Knowledge and Information Systems*, vol. 59, no. 3, pp. 497–522, Jun 2019. [Online]. Available: https://doi.org/10.1007/s10115-018-1223-9

[37] H. Hosseinmardi, H.-T. Kao, K. Lerman, and E. Ferrara, "Discovering hidden structure in high dimensional human behavioral data via tensor factorization," *arXiv preprint arXiv:1905.08846*, 2019.

[38] J. Liu, P. Musialski, P. Wonka, and J. Ye, "Tensor completion for estimating missing values in visual data," *IEEE transactions on pattern analysis and machine intelligence*, vol. 35, no. 1, pp. 208–220, 2012.

[39] D. M. Dunlavy, T. G. Kolda, and E. Acar, “Temporal link prediction using matrix and tensor factorizations,” *ACM Transactions on Knowledge Discovery from Data (TKDD)*, vol. 5, no. 2, p. 10, 2011.

[40] F. Zhang, N. J. Yuan, D. Wilkie, Y. Zheng, and X. Xie, “Sensing the pulse of urban refueling behavior: A perspective from taxi mobility,” *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 6, no. 3, p. 37, 2015.

[41] K. Xie, L. Wang, X. Wang, G. Xie, J. Wen, G. Zhang, J. Cao, D. Zhang, K. Xie, X. Wang *et al.*, “Accurate recovery of internet traffic data: A sequential tensor completion approach,” *IEEE/ACM Transactions on Networking (TON)*, vol. 26, no. 2, pp. 793–806, 2018.

[42] S. Rabanser, O. Shchur, and S. Günnemann, “Introduction to tensor decompositions and their applications in machine learning,” 2017.

[43] N. Sorkunlu, D. T. Anh Luong, and V. Chandola, “dynamicmf: A matrix factorization approach to monitor resource usage in high performance computing systems,” in *2018 IEEE International Conference on Big Data (Big Data)*, 2018, pp. 1302–1307.

Appendix: Artifact Description/Artifact Evaluation

SUMMARY OF THE EXPERIMENTS REPORTED

(1) We ran experiments on two different systems. The main comparative results provided in Table III: "Storage error prediction performance comparison among SEFEE, LSTM, and Random Forests for Worklaod A in batch-2017" and Table IV were done on AWS c5.9xlarge instance with 141 ECUs, 36 vCPUs, 3.4 GHz, Intel Xeon Platinum 8124M, 72 GiB memory, EBS only. The environment variables are collected from this system and attached below. However, since AWS C5.9Xlarge is a high-end compute optimized system, we set out to evaluate SEFEE's computation time on a typical system as well and that brings us to the 2nd system which was author's PC: PowerEdge T20 with 3.5" 1TB SATA HDD, 4GB DIMM, Xeon E3-1225 v3 3.2GHz 4C 84W CPU, DVD+/-RW. The results from this experiment is reported in Table VII: "Prediction time comparison between using the whole tensor versus using subtensors." The environment variables from this system is also collected and attached below. Both these systems are mentioned in the paper as well.

(2) We had raw proprietary data that was critical event logs from enterprise storage systems. We first aggregated the data into 2 hour time bins and formed a tensor from the main data. We then used MATLAB's Tensor Toolbox (a publicly available tool) , poblano toolbox, and CMTF (All publicly available tools) to run prediction experiments. We have publicly released our code in a github repository where you can find more details about the libraries (and versions) used.

(3) SEFEE attempts multivariate time-series prediction, thus for comparison we used multivariate models for LSTM and Random Forests as well. More details can be found in our github repo's ReadMe file.

ARTIFACT AVAILABILITY

Software Artifact Availability: All author-created software artifacts are maintained in a public repository under an OSI-approved license.

Hardware Artifact Availability: There are no author-created hardware artifacts.

Data Artifact Availability: Some author-created data artifacts are NOT maintained in a public repository or are NOT available under an OSI-approved license.

Proprietary Artifacts: There are associated proprietary artifacts that are not created by the authors. Some author-created artifacts are proprietary.

Author-Created or Modified Artifacts:

```
Persistent ID: https://github.com/SayBender/SEFEE
Artifact name: SEFEE
```

BASELINE EXPERIMENTAL SETUP, AND MODIFICATIONS MADE FOR THE PAPER

Relevant hardware details: 141 ECUs, 36 vCPUs, 3.4 GHz, Intel Xeon Platinum 8124M, 72 GiB memory, EBS only

Operating systems and versions: Ubuntu 18.04

Applications and versions: MATLAB 2019

Libraries and versions: Tensor Toolbox v2.6, CMTF_toolbox v1.1, Poblano_toolbox v1.1, keras 2.2.4, tensorflow 1.14

Key algorithms: Alternating least squares, adam optimizer, TreeBagger, non-linear conjugate gradient descent

URL to output from scripts that gathers execution environment information.

`https://github.com/SayBender/executionenvironment`

ARTIFACT EVALUATION

Verification and validation studies: We verified our approach through experimental evaluations. The experimental evaluations include extensive performance comparison (Precision, Recall, F1-measure, False Positive rate, ROC curve, PRC curve) and computation time. We also ran sensitivity analysis on time and prediction performance (Figure 10 and 11). We specifically compared our approach with 2 state-of-the-art approaches in terms of time taken to produce prediction and prediction accuracy in terms of precision, recall, F1-measure and FPR. The experiments of the baseline approaches are done in the same environment as the proposed approach. On top of those experiments, we further validated the feasibility of using our approach on systems with typical specs. Thus, we evaluated the prediction time of SEFEE on Author's PC with 1, 2 and 4 CPU cores. This validated that, in a typical system with 4 GB RAM and 4 core CPU, our approach is able to produce prediction in reasonable time as discussed in the paper.

Accuracy and precision of timings: The tensor-based SEFEE approach and RandomForests were both conducted on MATLAB and we used MATLAB's built-in tic toc library to time the experiments. However, LSTM experiment was conducted in python using keras and we approximated the total time taken by using the built in time per epoch in training output. This might not be totally precise but a close enough approximation.

Used manufactured solutions or spectral properties: No

Quantified the sensitivity of results to initial conditions and/or parameters of the computational environment: Our algorithm is not sensitive to initial conditions, but the system randomness might slightly change the results.

Controls, statistics, or other steps taken to make the measurements and analyses robust to variability and unknowns in the system. We have run the experiments multiple times and made sure that the results are not unusually anomalous.