University of Nevada, Reno

Proactively Handling Failures in Extreme-Scale Big Data Storage: A Data Driven Approach

A thesis submitted in partial fulfillment of the
requirements for the degree of Master of Science
in Computer Science and Engineering

by

Gautham Yerroju

Dr. Feng Yan, Thesis Advisor

August, 2018

THE GRADUATE SCHOOL

We recommend that the thesis
prepared under our supervision by

GAUTHAM YERROJU

Entitled

Proactively Handling Failures in Extreme-Scale Big Data Storage: A Data Driven Approach

be accepted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

Dr. Feng Yan, Advisor

Dr. Lei Yang, Committee Member

Dr. Hao Xu, Graduate School Representative

David W. Zeh, Ph.D., Dean, Graduate School

August, 2018

Abstract

With the prosperity of Big Data, the performance and robustness of storage systems have become ever more important. Today's Big Data enterprise storage systems operate at petabyte scale and are composed of thousands to tens of thousands of data drives organized in a hierarchical architecture. These storage systems usually host many applications concurrently and perform continuous data collection and analysis. Such complex hardware and software stacks under intensive usage can easily result in hundreds of critical storage failures in a daily manner and pose significant challenges on the robustness of these systems. State of the practice solution reactively moves data from failed disks to spare disks, which usually leads to severe performance degradation and unrecoverable data loss. This thesis targets at developing a proactive methodology to predict critical storage failures in advance so that the backup process can start even before a failure occurs to reduce the performance impact and the risk of data loss. Given there are hundreds of critical failures in modern Big Data storage systems and there are complex dependencies among them, we propose a data-driven approach by characterizing an extreme-scale storage system that is composed of 8 storage clusters with 462,578 data drives over a one-year period, which adds up to 857,183,442 data drive hours. Considering the lightweight requirement of practical systems and the sparsity of the data, we opt for a statistical approach instead of heavy overhead machine learning based methods that usually require tremendous computing power with a large number of training samples. The proposed spatial-temporal prediction methodology builds on the findings from characterization results, especially the spatial-temporal dependencies in each failure type as well as cross failure types. The extensive evaluation suggests that the proposed prediction methodology can outperform state of the art Markov Chain based methods by more than 20% and it can also adapt swiftly in dynamic environments.

Dedication

I dedicate this thesis to all the researchers in academia whose collective hard work continues to push the boundaries of human knowledge and improve lives.

Acknowledgments

I would like to thank my adviser Dr. Feng Yan and my committee members Dr. Lei Yang and Dr. Hao Xu for their time and support.

I am grateful to our collaborators from NetApp, in particular Xing Lin, Art Harkin, Shankar Pasupathy, and Jeffrey Heller for providing the data and support, which are essential to this project.

I would also like to thank Dr. Mai Zheng for contributing to the spatial analysis part of the project.

Last but not least, I would like to thank my family for their unconditional love and support.

This material is based upon work supported by the NSF under Award NO. CCF-1756013 and IIS-1838024.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

In the last decade, the blooming of Internet-based technologies, Internet of Things, and Cloud Computing has brought the rapid rise of Big Data [4, 15, 17]. Today's Big Data systems are built for capacities of petabyte scales and rely on extreme-scale storage systems containing thousands to tens of thousands of data drives running in concert, organized in a hierarchical architecture [29]. These extreme-scale storage systems are expected to offer good robustness and resilience capabilities to deal with irresistible external incidents like power irregularities and natural disasters, as well as implicit factors such as the lifespan of devices, electronic and mechanical failures, data corruption, and software bugs. The ever-growing complexity in hardware and software stacks which undertake continuous and intensive usage due to the restless data generation and analysis demands, can easily result in hundreds of critical storage failures on a daily basis [27]. This poses significant challenges to the robustness and resilience of the storage systems. Furthermore, failures can occur at any level of the hardware and software stack, so their root causes and dependencies are often difficult to track and analyze [14, 5]. The state-of-the-practice solution is to reactively move data from failed disks to spare disks until they are eventually replaced [21, 30, 13, 19]. However, this may lead to severely degraded availability and performance, and even cause unrecoverable data loss.

This thesis aims to develop a proactive methodology for forecasting critical storage failures so that the backup process can start even before the failure occurs. This way, the availability and performance impacts as well as the risk of permanent data

loss can be minimized. However, there are hundreds of critical failure types in today's Big Data systems and there are also complex dependencies among them, which makes identifying them and their root causes extremely challenging, especially during the runtime. To solve these challenges, we propose a data-driven approach by characterizing failures on an extreme-scale storage system that consists of 8 storage clusters housing 462,578 data drives over a period of one year, which adds up to 857,183,44 data drive hours. We performed temporal, spatial, and logical characterization and the results revealed that failure events tend to happen in a very bursty manner in the temporal dimension. Failure type distribution follows the 80/20 rule: most failures come from only a few failure types. The 80/20 rule is also observed in the spatial distribution of failure events at the shelf and device levels, as well as in the logical RAID group dimension. Similarly, in the spatial-temporal dimension, most groups of successive failure events tend to happen on the same device. Isolation implies that a large number of errors can happen on isolated devices while the neighboring devices remain error free. It is also shown that failure types are often temporally dependent with themselves and with other failure types. These findings indicate a strong potential for predictability.

Performing real-time analysis and prediction in large-scale production systems is known to be challenging as the managing hardware often uses embedded, low-power, or purpose-built processing units that do not have rich processing power to spare. Considering the scarcity of computing power and the sparsity of available data, we opt for a statistical approach for prediction instead of heavy overhead machine learning based methods, as they usually require a large amount of training samples to be effective and also consume lots of computing power. The proposed spatial-temporal methodology relies on findings from characterization results, especially the spatial-temporal dependencies in each failure type and across failure types. The proposed prediction algorithm builds on conditional probability approach and is being enhanced with the domain knowledge from the characterization results, such as burstiness, correlation, and propagation, to achieve better prediction accuracy with lower false alarm

rate. The improvements include combining probabilities of correlated failures, chaining predictions to capture propagation effects, exploring spatial retention, and using moving window to better reflect the short-term changes and be robust to dynamic behaviors. Results from extensive evaluation show that the proposed methods can achieve a success rate of over 90% in prediction, which outperforms state-of-the-art Markov chain-based methods by more than 20% and also be much more lightweight.

The rest of the thesis is organized as follows: we first discuss related work and give an overview of the data set in Chapter 2; then we conduct a detailed data characterization in Chapter 3; we introduce our prediction methodology in Chapter 4; we show extensive experimental evaluation results in Chapter 5; finally, we conclude our work and discuss the future work in Chapter 6.

Chapter 2

Background

2.1 Related Work

Much research has gone into workload and error characterization of data centers in the last decade, and many of them use similar techniques that are used in this project. For instance, Xue, Yan, Birke, Chen, Scherer, and Smirni have analyzed VM usage patterns in CPU, memory, disk and network bandwidth from workload traces, and proposed a neural-net based framework, PRACTISE, which is used to predict periods of future peak loads [32]. The main contribution of this work is using autocorrelation for automating feature selection for training the neural network. This enables automatically re-training the model when the existing model has too many prediction errors. The methodology used in this project (for the dynamic window method) is similar to the methodology used in this paper: we keep track of recent history and occasionally recalculate the probabilities. While PRACTISE enables easier automated re-training of the neural network, the conditional probability maps used in our method might be even cheaper to calculate. Moreover, we only use autocorrelation in our work for characterizing the data, not to generate the prediction model. Yan, Mountrouidou, Riska, and Smirni have presented a method to estimate performance delays on disk operations caused due to power saving mode, taking into account the propagation delay effects [33]. The CDF of idle times is used to predict the probability of different lengths of penalty, allowing more insight when selecting the idle time and desired penalty for a given system. This is similar to our work, in that we use a

table of conditional probabilities to make predictions of failures. However, this does not directly apply to what we are doing because the probabilities we calculate are of failures occurring, conditioned on other failures before them, so the predictions are binary in nature. Riska and Riedel have analyzed drive data from three data sets of different granularities: millisecond, hour and lifetime, for disk utilization, idleness availability and read/write ratio over time [23]. This study mainly looks at the distributions of various metrics and attempts to draw conclusions about them. Different granularities offer different types of insights into data. In our work, the source data is both highly granular (second-level granularity) and highly abundant (range of one year). But the data points themselves are only binary (representing whether a failure event occurred or not), instead of quantitative (such as read and write times, wait times, etc.). The data is binned into larger granularities (minute, hour, etc.) so that the range and specificity of the predictions can be controlled. Ros, Chen, and Binder have analyzed job traces from the Google cloud clusters, in an attempt to: (1) do a detailed failure analysis on event types leading to unsuccessful termination of jobs, and (2) machine learning based on-line prediction models to predict failing jobs [25]. Conditional probability is used in their characterization work to calculate the probability of a job completing successfully given the history of unsuccessful events. Similarly, the rate of job success dependent on various attributes is calculated, such as job size, arrival time, etc. But these probabilities are not directly used in their on-line prediction models. The attributes are instead only used as features for their machine learning models. Our work directly uses conditional probabilities in our on-line prediction algorithms with the goal of being computationally efficient. Hao, Soundararajan, Kenchammana-Hosekote, Chien, and Gunawi have analyzed data storage performance from a large-scale data set focusing on storage performance instabilities, in particular due to slow disks (storage tails). Their main contribution is the finding that internal characteristics and idiosyncrasies of disks and SSDs are the root cause for instability, and the design of a tail-tolerant RAID [12]. While their focus was to investigate root causes for instability in storage performance, our project

ignores root causes and focuses instead on a data-driven approach to characterizing and predicting failures. Fail-slow [10] is a failure mode where hardware is functional but performs well below its expected performance. This work studies 101 reports and presents findings, including: fail-slow can affect different types of hardware, faults convert from one form to another, the cascading root causes and impacts can be long and fail-slow can have different symptoms. In addition, suggestions are presented to vendors and manufacturers based on these findings.

The main focus of these papers and similar ones is usually root cause analysis [22, 6, 7, 9], and where prediction is involved, machine learning or neural network-based methods are used [34, 18, 11]. Additionally, when analyzing failures, more contextual information is usually available (such as CPU performance, disk speeds, etc.). This project focuses on characterizing and predicting storage failures in extreme-scale data centers with a data-driven approach.

2.2 Data Set Description

The data set is provided by NetApp Inc. and comes from their NetApp ONTAP data storage systems [20]. Some of their systems can support up to 17,820 drives with a maximum storage capacity of 172 petabytes.

The hierarchy of storage is as follows: multiple disks exists in a shelf, multiple shelves make a stack, multiple stacks make a node, and multiple nodes make a cluster.

Two sets of data are provided. The first data set includes system errors and warnings from eight different clusters, A through H. Each cluster belongs to a different customer, and together they represent ONTAP system deployments running real production workloads from multiple industries. The second data set is grouped by workloads A through D each running on production clusters from multiple customers. For data privacy reasons, further information cannot be shared.

The data consists of failure events for a large number of disks: 462,578 disks split into 38,601 raid groups. The data set contains a rich information for analyzing spatial temporal patterns, such as the failure type, the time stamp of the failure event, the

specific location at the storage hierarchy, and the RAID group ID.

Chapter 3

Data Characterization

3.1 Failure Type Distribution

There are on average 33 failure types in each cluster and 163 failure types in each workload. Specific counts are shown in 3.1. Within all clusters, there are 126 failure types in total, but only a few failure types are common among more than one cluster. Only 33 types (26.2%) appear in more than three clusters and only 32 types (25.4%) appear in any two clusters. The remaining 61 types (48.4%) only appear in any one cluster. Since workloads contain data from multiple clusters, they have more failure types than clusters and more failure types occur in more than one workload. Within all workloads, there are 286 failure types in total. 109 types (38.1%) occur only in any one workload, 57 types (19.9%) occur in any two workloads, 48 types (16.8%) occur in any three workloads and 72 types (25.17%) occur in all workloads.

Examining histograms of failure types revealed that their distribution follows the 80/20 rule (also known as the Pareto Principle), which states that the majority of effects come from a minority of causes [26]. For example, across all workloads, 6% of failure types contribute to 80% of total failure events, and 9.3% of failure types contribute to 90% of total failure events. This can be seen in Figure 3.1.

Finding 1: *A small number of failure types contribute to a large percentage of failure events.*

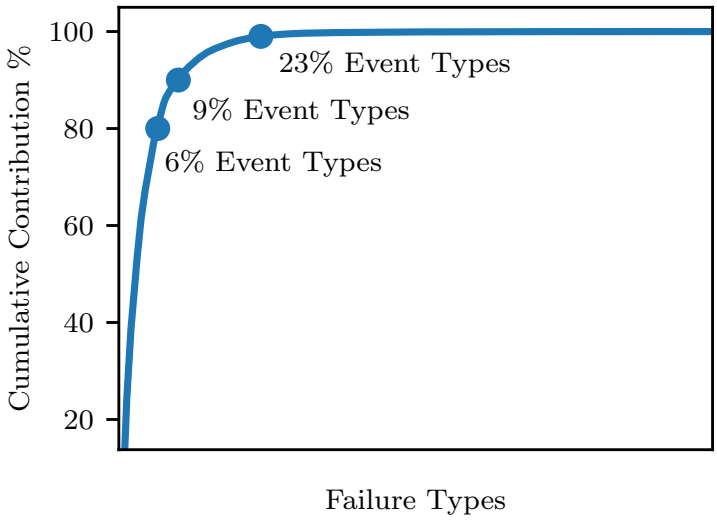

Figure 3.1: Contribution of failure types to the volume of failure events. x-axis represents individual failure types, and y-axis represents the contribution in percentage.

Data set	# of failure types
clusterA	47
clusterB	32
clusterC	29
clusterD	35
clusterE	22
clusterF	19
clusterG	43
clusterH	38
WorkloadA	175
WorkloadB	200
WorkloadC	164
WorkloadD	116

Table 3.1: Failure type counts in each data set.

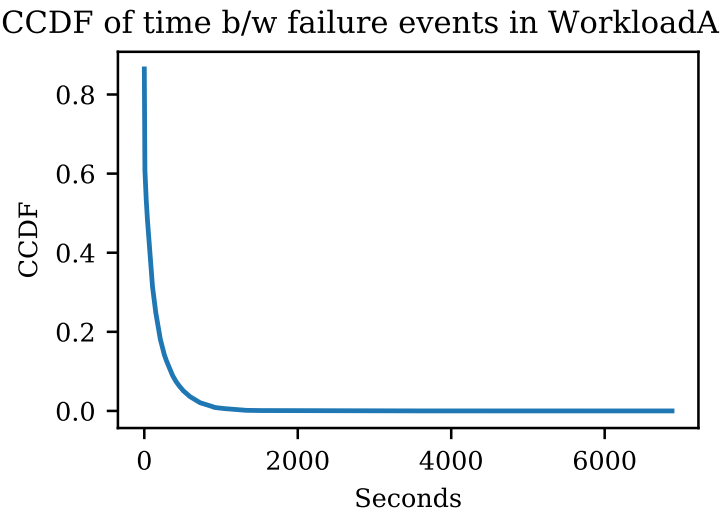

Figure 3.2: Complementary CDF of time between failure events of WorkloadA.

3.2 Temporal Analysis

3.2.1 Time Between Failure Events

To get a sense of how failure events are spread out across time, the distribution of time between failure events is analyzed. The complementary CDF of the distribution of the most significant failure types (contributing to 99% of the overall failure events) shows long tails, which indicates that very short time between failure events are most common. Very long time between failure events are also present in the data set. This indicates that the data is temporally sparse. This behavior is consistent across all the data sets except clusterE, which also has commonly occurring large time between failure events.

Finding 2: *Failure events tend to occur in bursts.* This is shown by the long tails in the Complementary CDFs (CCDFs) of time between failure events. Figure 3.2 shows the CCDF of WorkloadA and Figure 3.3 shows the CCDF for clusterE.

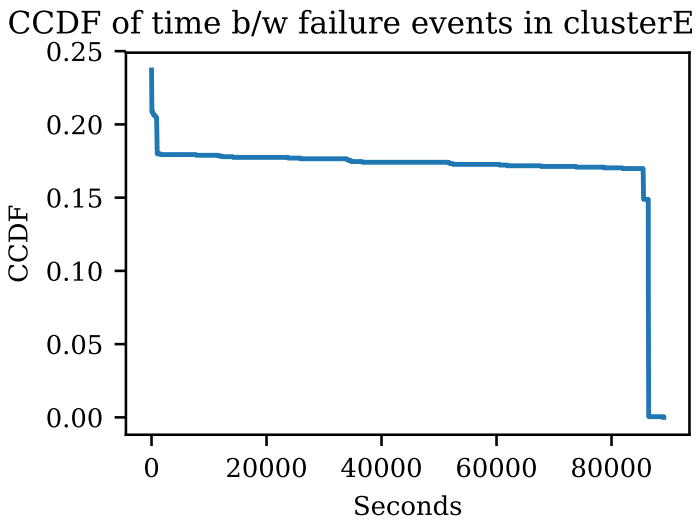

Figure 3.3: Complementary CDF of time between failure events of clusterE.

3.2.2 Time binning

The source data is converted to into a uniform time series by binning failure events into fixed-size time slots. This is called time-binning. This converts points on a time line into a curve, with each point on the x-axis representing the bin number and y-axis representing the number of failure events in that bin. This allows us to analyze the rate of failure events instead of individual failure events. The choice of bin size depends upon the required granularity of the characterization. Analysis is done on different scales including one-minute bins, one-hour bins, and even one-day bins. This is done separately for each failure type. Correlation analysis is done on the binned time series.

Correlation is a measure of linear dependency between two signals [1] [28], and is a common method used in time series analysis.

3.2.3 Autocorrelation

Autocorrelation is the correlation between a signal and a delayed copy of itself as a function of delay. It is used to detect non-randomness of data [31]. In other words, it detects self-repeating patterns in a time series. Autocorrelation for each failure type

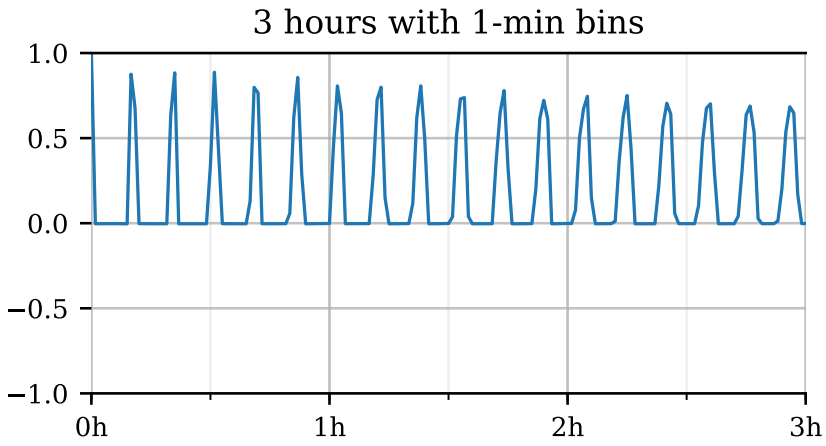

Figure 3.4: Failure type *ClientAccessIndeterminated* showing high autocorrelation. The x-axis is lag in time bins, and the y-axis is the correlation value.

was calculated at multiple lag values (in number of bins) in an attempt to find any repeating patterns or periodicity. The main findings are listed below.

Finding 3: *A significant number of failure types show high autocorrelation at non-zero lags.* This implies that it is possible to predict failure occurrences in advance with reasonable accuracy. For example, out of 175 failure types in WorkloadA, 13 have daily patterns and 4 have 4-hourly patterns. Figure 3.4 shows an example of this.

Finding 4: *A few failure types show strong periodicity.* Figure 3.5 shows an example of a failure type with strong periodicity.

It is interesting to note that failure types with high occurrence do not necessarily have a strong pattern. An example of this is shown in Figure 3.6.

3.2.4 Cross-correlation

Cross-correlation is the correlation between two signals as a function of delay. It is used to estimate the degree to which two series are correlated [2]. For each failure type pair, cross-correlation is calculated at different time scales (i.e., bin sizes) to find temporal dependencies between them. The findings are listed below.

Finding 5: *High cross-correlation can occur at multiple time scales.* This implies

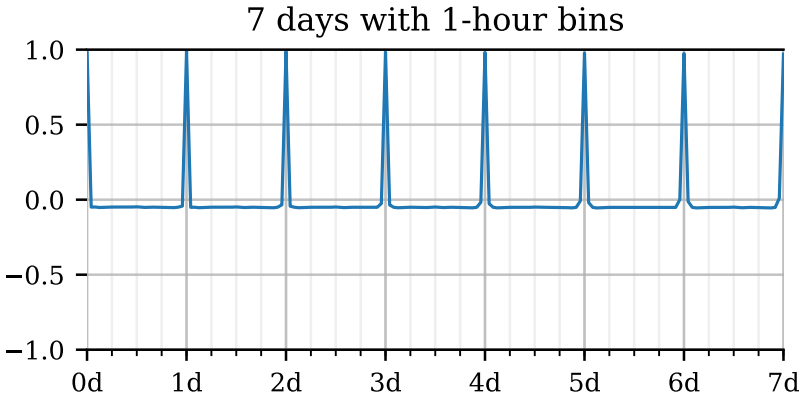

Figure 3.5: Autocorrelation of the failure type *VserverCertificateExpired* for the 1-hour bin size. The x-axis is lag in time bins, and the y-axis is the correlation value. This failure type occurs at the beginning of each day every day, which is revealed in the autocorrelation plot.

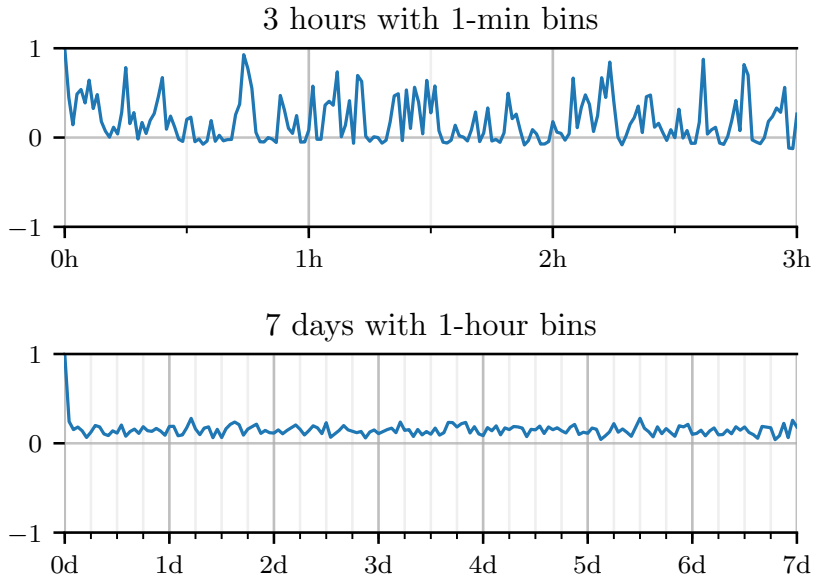

Figure 3.6: Failure type *UncompletedDiskIO* showing high autocorrelation (due to high density of data), but too irregular for a repeating pattern. The x-axis is lag in time bins, and the y-axis is the correlation value.

that failure types can be temporally dependent in the short term as well as the long term. Figure 3.7 shows an example of this.

Finding 6: *About 35.37% of failure type pairs in clusters A-H show significant cross-correlation at multiple non-zero lags.* This implies that it is possible to predict failure occurrences in advance with reasonable accuracy. A heat map of cross-correlation between a subset of failure type pairs in clusterA is shown in Figure 3.8, which shows that a large number of failure type pairs are correlated.

Finding 7: *At least 80% of failure types in every cluster are cross-correlated with at least one other failure type within that cluster.* This implies that most failure types in any data set are temporally dependent with other failure types. This is shown in Figure 3.9.

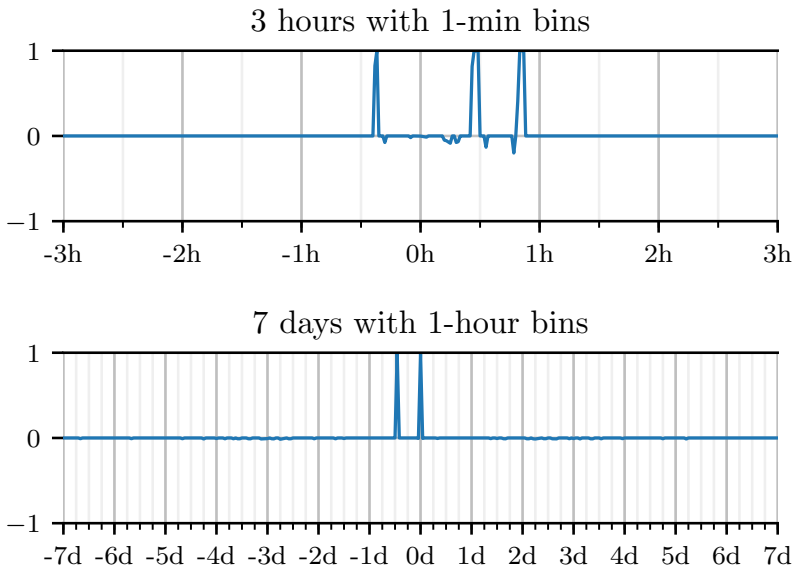

Figure 3.7: Cross-correlation between failure types *SASDeviceTimeout* and *SCSI-AdapterHardwareError*. The plots show high non-zero correlation values between these two failure types in two time scales: within an hour and within a day.

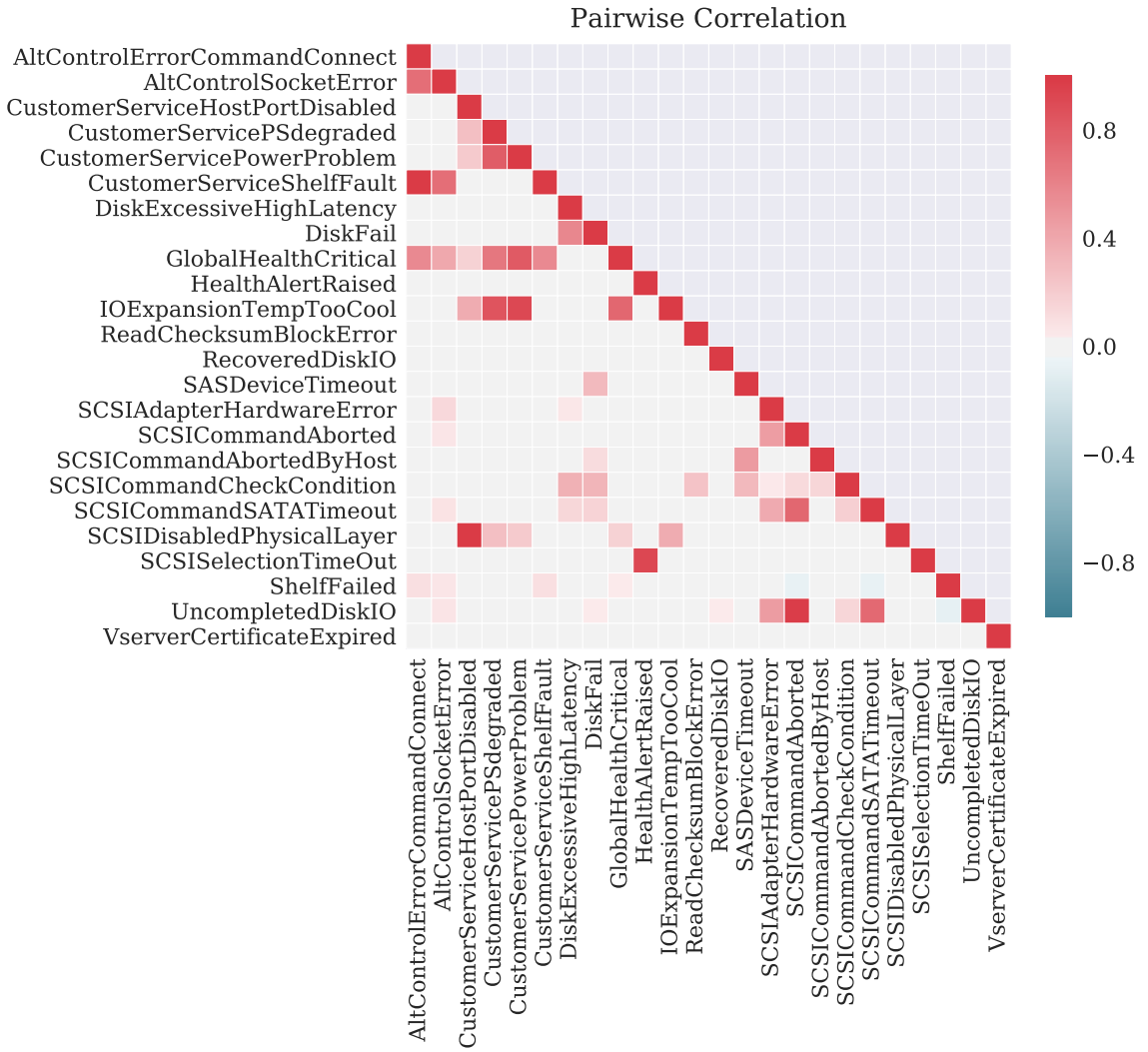

Figure 3.8: Cross-correlation heat map for a subset of failure types in clusterA for the 2-hour bin size, lag 0. Darker squares indicate higher correlation. The range of correlation is -1 to 1. The x-axis and the y-axis contain the same list of failure types and the grid shows the correlation between various failure type pairs. Values on the diagonal are always 1 because at lag 0, diagonals represent correlation between the same failure type.

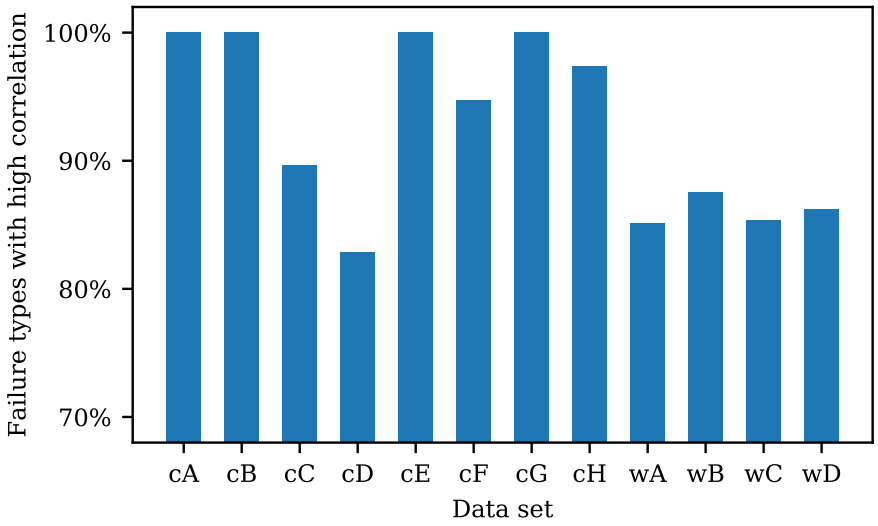

Figure 3.9: Percentage of failure types in each data set which are significantly cross-correlated with at least one other failure type. Prefix "c" indicates a cluster, prefix "w" indicates a Workload.

3.2.5 Conditional Probability

Conditional probability is the probability of a failure occurring given that another failure has occurred [24]. This is used as a way to express temporal dependency between failure type pairs. Using the time series for each failure type, the conditional probability of two failure types A and B is calculated such that $P(A|B)$ is the probability of failure type A occurring in a time bin given that failure type B occurred in the same time bin. This is the conditional probability for lag 0 (as it refers to the same time bin). Probabilities for non-zero time bins are also calculated, such that for lag x, $P(A|B)$ is the probability of failure type A occurring in time bin $c + x$ given that failure type B occurred in time bin c. An example of a conditional probability heat map is shown in Figure 3.10.

Overall, we find that many failure type pairs have high conditional probabilities at both zero and non-zero lags.

An effect of using conditional probability on larger time bins is that, since more failure events fall within each bin, conditional probability is higher. This means that probabilities for large window sizes, though high, may not be temporally specific.

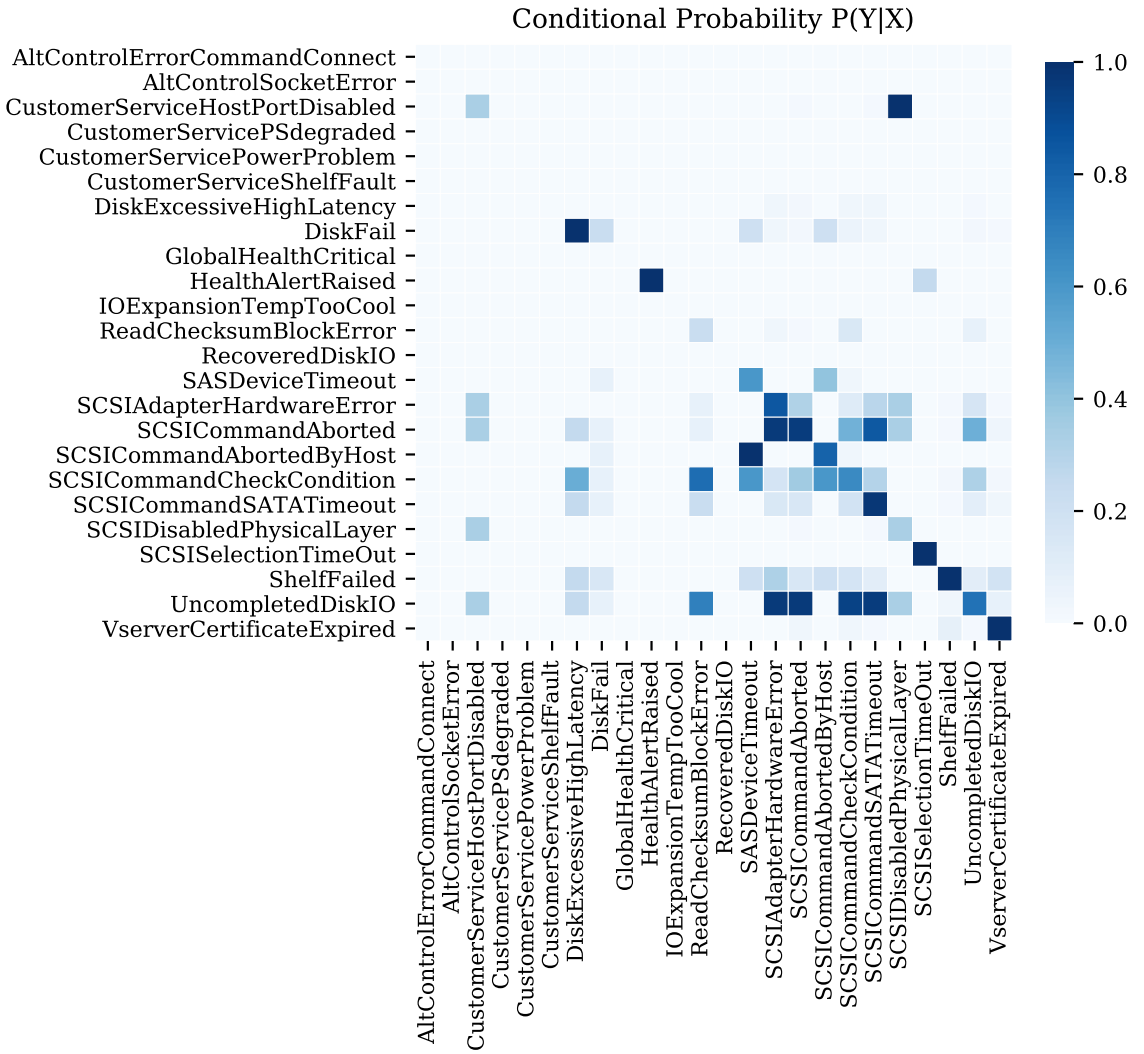

Figure 3.10: Conditional probability heat map for a subset of failure types in clusterA for the 2-hour bin size, lag 0. The heat map shows the probability of a failure type on the y-axis given a failure type on the x-axis occurred, i.e., $P(Y|X)$. Dense horizontal lines indicate that those failure types on the y-axis are dependent on many failure types on the x-axis. Similarly, dense vertical lines indicate that those failure types on the x-axis are followed by many failure types on the y-axis.

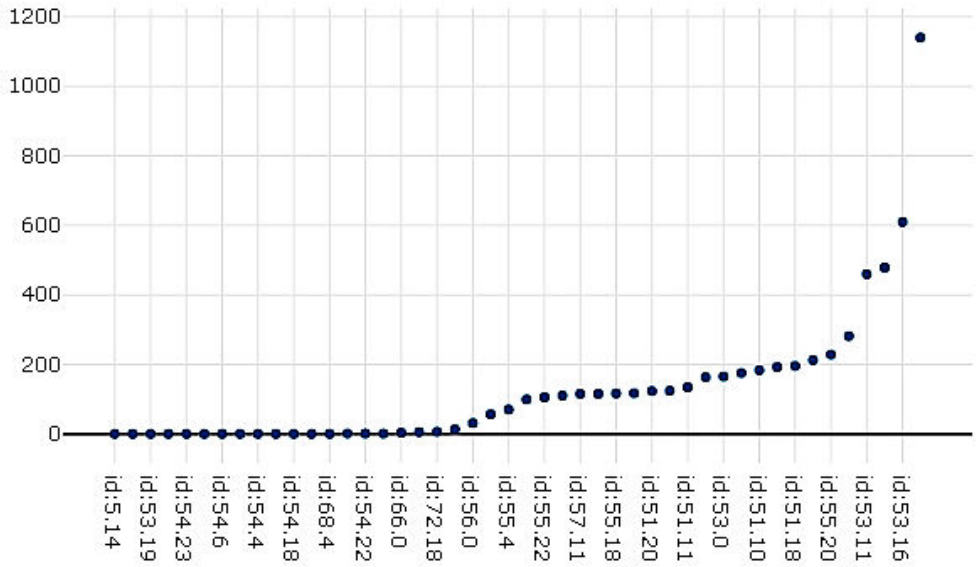

Figure 3.11: **Example of the 80/20 rule at the device level.** *This figure shows that the majority of failure events occurred on a minority of devices within a shelf. The y-axis is the number of failure events, the x-axis is the device IDs ordered by the increasing number of failure events.*

3.3 Spatial Analysis

In this section, we analyze the spatial patterns and correlation of the failure events. Overall, we find that there is strong spatial locality of failure events at different levels of systems. More specially, we find that the majority of failure events tend to occur at a minority subset of units (80/20 rule), and failure events tend to occur at isolated devices where neighboring devices are failure-free (isolation).

3.3.1 80/20 Rule

The 80/20 rule states that, for many failures, the majority of the effects come from a minority of the causes. Interestingly, we find that failure events exhibit such a rule at different levels (i.e., shelf, device, and RAID group).

Finding 8: *At the shelf level, 80% failure events happen at 21.4% shelves, and 60% failure events happen at 13.9% shelves.*

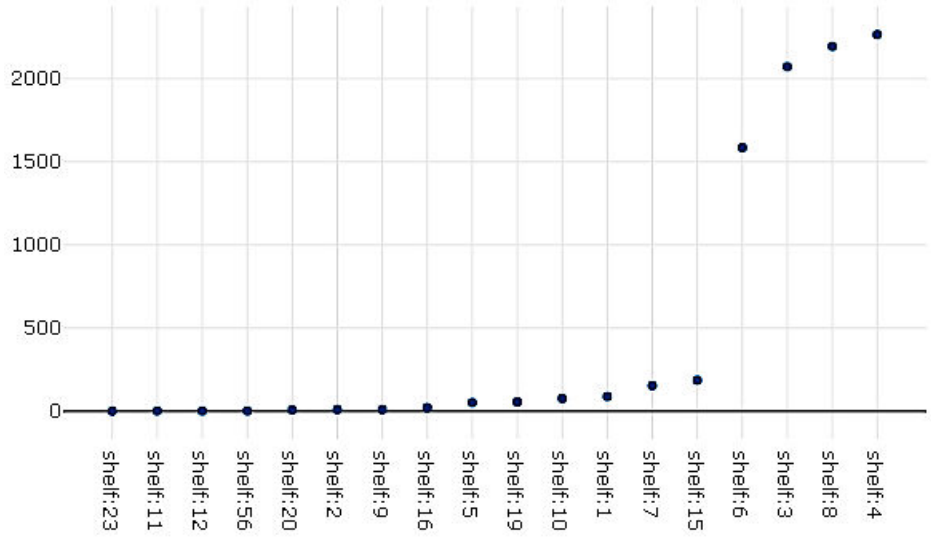

Figure 3.12: **Example of the 80/20 rule at the shelf level.** *This figure shows that the majority of failure events occurred on a minority of shelves within a stack. The y-axis is the number of failure events, the x-axis is the shelf IDs ordered by the increasing number of failure events.*

Figure 3.12 shows an example of shelf-level 80/20 rule observed on one node. The y-axis is the number of failure events, the x-axis is the shelf IDs ordered by the increasing number of failure events (shelves without failure events are not shown). Clearly, the last four shelves (i.e., 4, 8, 3, 6) account for the majority of failure events on this node.

Table 3.3 shows the shelf-level 80/20 rule on one system with four nodes. The second column shows the percentage of shelves containing 80% failure events on each node, while the third column shows the percentage of shelves containing 60% failure events. For instance, on node #1, 80% failure events occur on 32.91% shelves, and 60% failure events occur on 19.91% shelves. That last row shows that on average, 21.4% shelves account for 80% failure events, and 13.9% shelves account for 60% failure events, which match the 80/20 rule well.

Similarly, at the device level and the RAID group level, we have the following findings:

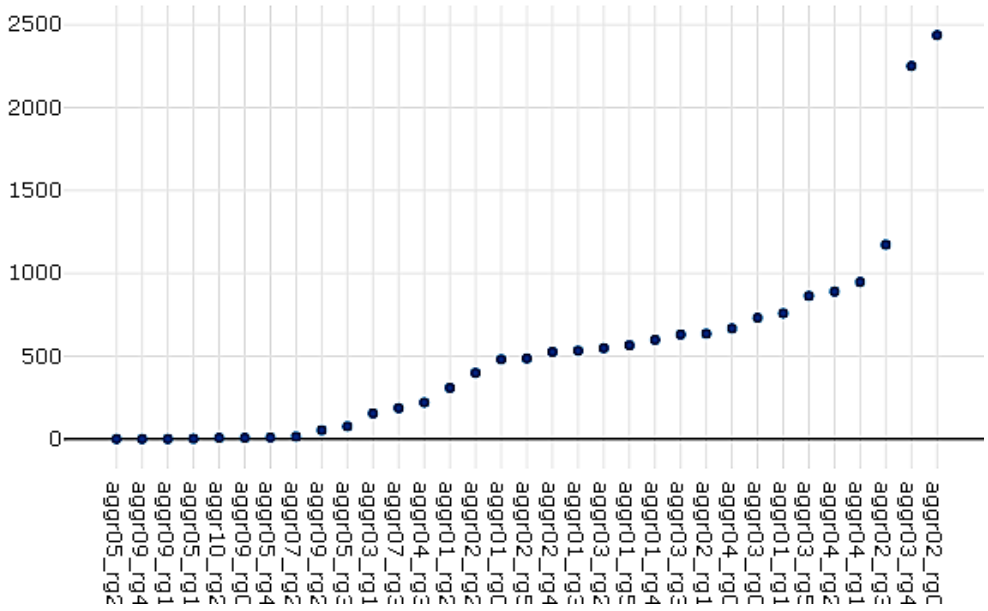

Figure 3.13: **Example of the 80/20 rule at the RAID group level.** *This figure shows that the majority of failure events occurred on a minority of RAID groups within a system. The y-axis is the number of failure events, the x-axis is the RAID group IDs ordered by the increasing number of failure events.*

Finding 9: *At the device level, 80% failure events happen at 37.4% devices, and 60% failure events happen at 21.9% devices.* An example is shown in Figure 3.11. Table 3.3 further shows the device-level 80/20 rule on one system with four nodes.

Finding 10: *At the RAID group level, 80% failure events happen at 43.5% RAID groups, and 60% failure events happen at 28.5% RAID groups.* An example is shown in Figure 3.13. Table 3.3 further shows the RAID-group-level 80/20 rule on one system with four nodes.

3.3.2 Isolation

One common assumption is that failure events tend to occur at neighboring devices due to the share of hardware. Surprisingly, we find that this is not true at the device level.

Finding 11: *A large number of failure events occur at isolated devices where all neighboring devices are failure-free.*

Node ID	80% failures	60% failures
1	51.7	31.05
2	26.88	15.60
3	31.51	16.32
4	39.79	24.47
Average	37.47	21.86

Table 3.2: **Device-level 80/20 rule on one system.** *This table shows that on one system with four nodes, the percentage of devices containing 80% failure events (2nd column), and the percentage of devices containing 60% failure events (3rd column) on each node respectively. The last row shows the average percentage across four nodes.*

Node ID	80% failures	60% failures
1	32.91	19.91
2	18.43	13.03
3	17.14	11.12
4	17.18	11.39
Average	21.41	13.86

Table 3.3: **Shelf-level 80/20 rule on one system.** *This table shows that on one system with four nodes, the percentage of shelves containing 80% failure events (2nd column), and the percentage of shelves containing 60% failure events (3rd column) on each node respectively. The last row shows the average percentage across four nodes.*

Figure 3.14 shows an example of the pattern. In the shelf with 24 devices (0-23), the devices with failure events are marked in red. Most failure events in this shelf occur at two isolated devices (12 and 14), while the remaining failure events occur at 7 continuous devices (2, 4, 5, 6, 7, 9, 10).

Table 3.5 shows the numbers of continuous devices that contain failure events, and the corresponding percentages of failure events. We can see that the maximum number of continuous devices with failure events is 17. On the other hand, many failure events occur at isolated devices, which is more than the failure events occurred at any other continuous devices.

Node ID	80% failures	60% failures
1	50.14	32.03
2	48.83	27.25
3	44.32	24.92
4	45.75	29.92
Average	43.51	28.53

Table 3.4: **Raid-group-level 80/20 rule on one system.** *This table shows that on one system with four nodes, the percentage of RAID groups containing 80% failure events (2nd column), and the percentage of RAID groups containing 60% failure events (3rd column) on each node respectively. The last row shows the average percentage across four nodes.*

# of C.D.	1 (isolated)	2	3	4	7	14	17
Node 1	55.46%	11.11%	10.58%	7.60%	16.25%	–	–
Node 2	30.65%	13.58%	10.30%	–	0.53%	21.36%	23.57%
Node 3	35.23%	36.32%	14.50%	13.95%	–	–	–
Node 4	39.58%	24.57%	8.05%	7.29%	24.57%	–	–

Table 3.5: **Isolation and Continuity.** *This table shows the percentages of failure events that occur at different number of continuous devices. The top row denotes the # of continuous devices. The remaining rows denote the % of failure events on each node. The majority of failure events occur at isolated devices, while the remaining failure events occur on 17 continuous devices.*

3.4 Spatial-Temporal Analysis

3.4.1 Logical Locality

In this section, we analyze groups of failure events to see if there is strong logical locality, which refers to disks belonging to the same RAID group. Failure events are clustered into groups based on the time they occurred. To do this, a histogram of the time between failure events is calculated and Kernel Density Estimation (KDE) [3] is performed to smooth the histogram. Then, local minima are used as splitting points to cluster time between failure events. The highest value from the largest cluster (having the highest peak) is used as a threshold duration to split failure events into clusters (i.e., if the distance between two failure events is larger than the selected

Figure 3.14: **Example of isolation pattern.** *This figure shows that in a shelf with 24 devices, the majority of failure events occurred on an isolated device (14), while the neighboring devices (10,13,15,18) do not have any failure events.*

threshold, then we consider that as a new group of failure events). An example for clusterA is shown in Figure 3.15

We found that grouping failure events by KDE was too aggressive: multiple bursts of failure events spanning days were grouped together in instances where the data was sparse. But from previous analysis, we know that failure events happen in shorter bursts. To get a more accurate grouping, a threshold of five minutes (based on the time between failures analysis) is used as the cut-off for time between failure events. That is, failure events more than 5 minutes apart are considered to be from different groups. With this threshold, we find that most of the time, groups of failure events happen mostly within the same disk. An example is shown in Figure 3.16. However, clusterA and WorkloadD deviate from this finding. clusterA has 43.83% of the failure event groups happening in different RAID groups, and WorkloadD has 47.32% of its failure event groups happening in different disks but the same RAID groups. The detailed composition from the data sets is shown in Table 3.6.

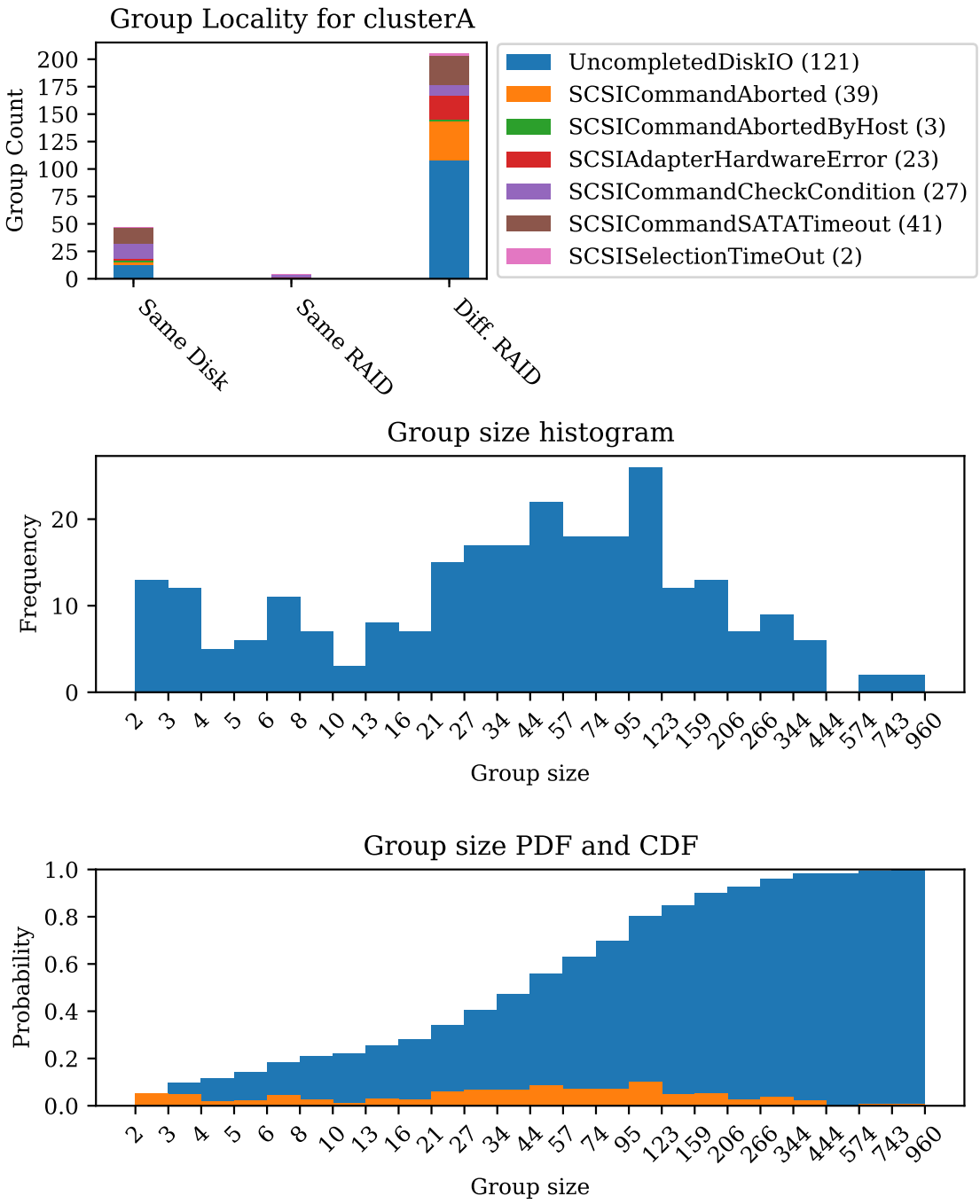

Figure 3.15: Logical locality of failure events in clusterA using KDE-based grouping of events. The first graph shows the number of groups of failure events that happened on the same disk, same RAID group, or different RAID groups. The legend shows the failure types and the number of groups for each failure type. The second graph is a histogram of the group sizes across all failure types. This helps visualize the distribution of group sizes. The bin boundaries for the histogram are designed so that smaller numbers are more distinct, whereas larger numbers are lumped together. The third graph shows the PDF (orange) and CDF (blue) of the group sizes.

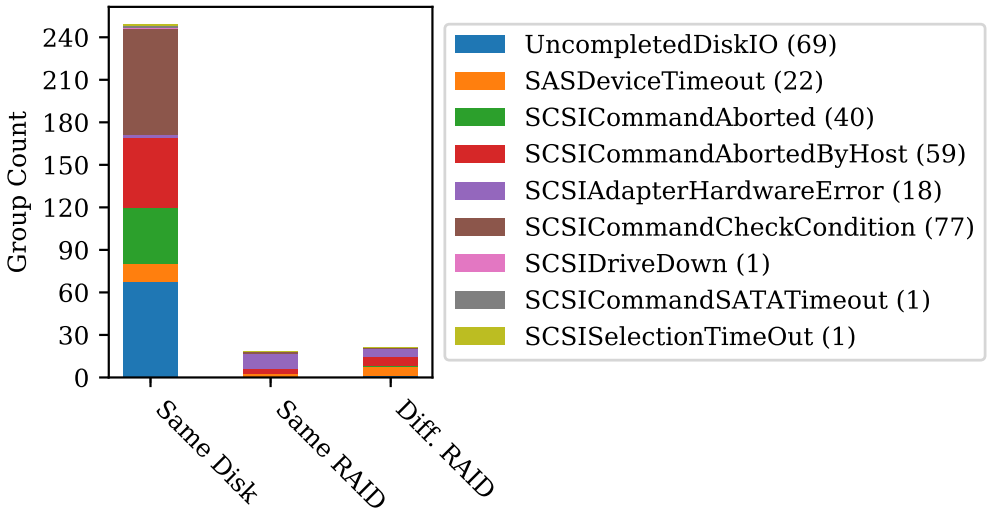

Figure 3.16: Logical locality of failure events in WorkloadC. The legend shows the name of each failure type and the number of groups in that failure type.

Data set	Same Disk	Same RAID	Different RAID
clusterA	49.5%	6.68%	43.83%
clusterB	100%	0%	0%
clusterC	100%	0%	0%
clusterE	100%	0%	0%
clusterF	0%	100%	0%
clusterG	100%	0%	0%
clusterH	100%	0%	0%
WorkloadA	91.3%	2.32%	6.38%
WorkloadB	64.17%	9.29%	26.54%
WorkloadC	86.46%	6.25%	7.29%
WorkloadD	51.86%	47.32%	0.82%

Table 3.6: Logical locality of failure event groups

Finding 12: *Groups of failure events are highly isolated, i.e., tend to happen on the same disk.*

3.4.2 Spatial Locality

Spatial locality describes the number of failure event groups occurring in different levels of spatial hierarchy. We find that with the exception of clusterA and Work-

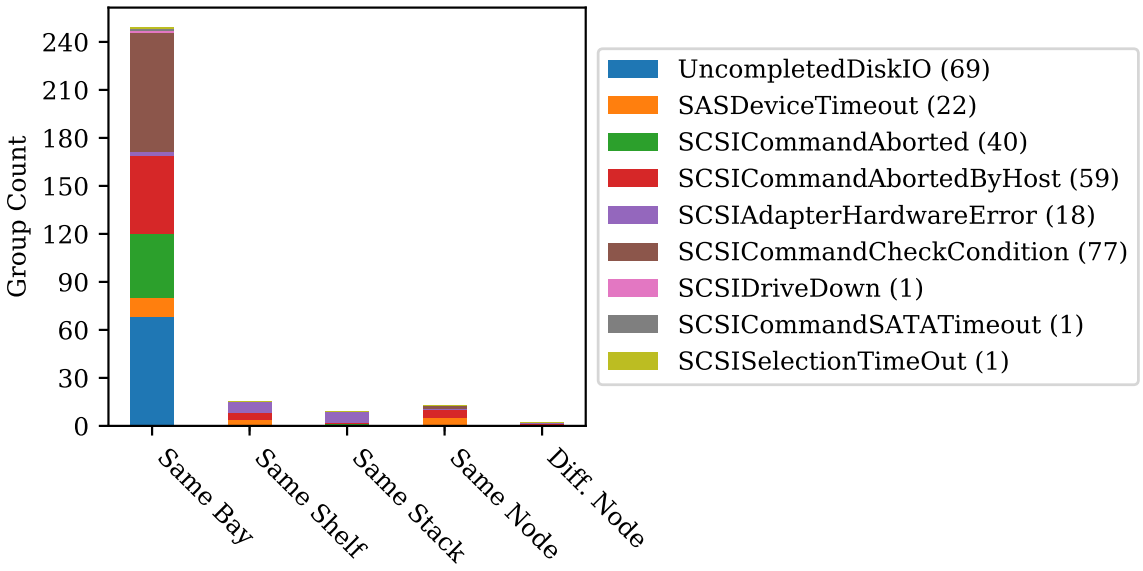

Figure 3.17: Spatial locality of failure events in WorkloadC. The legend shows the name of each failure type and the number of groups of that failure type.

loadA, most of the failure event groups happen on the same bay in all data sets. An example of this is shown in Figure 3.17. For clusterA, 42.44% of groups happen on different bays but in the same shelf. For WorkloadD, 21.79% of groups happen in the same shelf, and 22.26% of groups happen in different nodes.

Finding 13: *Groups of failure events show strong spatial locality, i.e., they tend to happen in the same bay.* This follows from the previous finding from logical locality which shows that most failure event groups tend to happen on the same disk.

3.4.3 Node Jumping

Histogram Representation

Plotting failure events across time on different nodes in each cluster or workload reveals that for some failure types, failure events do not happen on multiple nodes at the same time. Instead, they tend to occur on one node, then change to a different node. This is termed as *node jumping*. It is the tendency of failure events to switch to different nodes. To analyze this, failure events are grouped based on consecutive events occurring on the same node. This means that each group of failure events are

consecutive events that happened on the same node, regardless of the time between them. More number of larger groups mean that there is low node jumping, and more number of smaller groups mean that there is high node jumping. Figure 3.18 shows an example of high node jumping and Figure 3.19 shows an example of low node jumping.

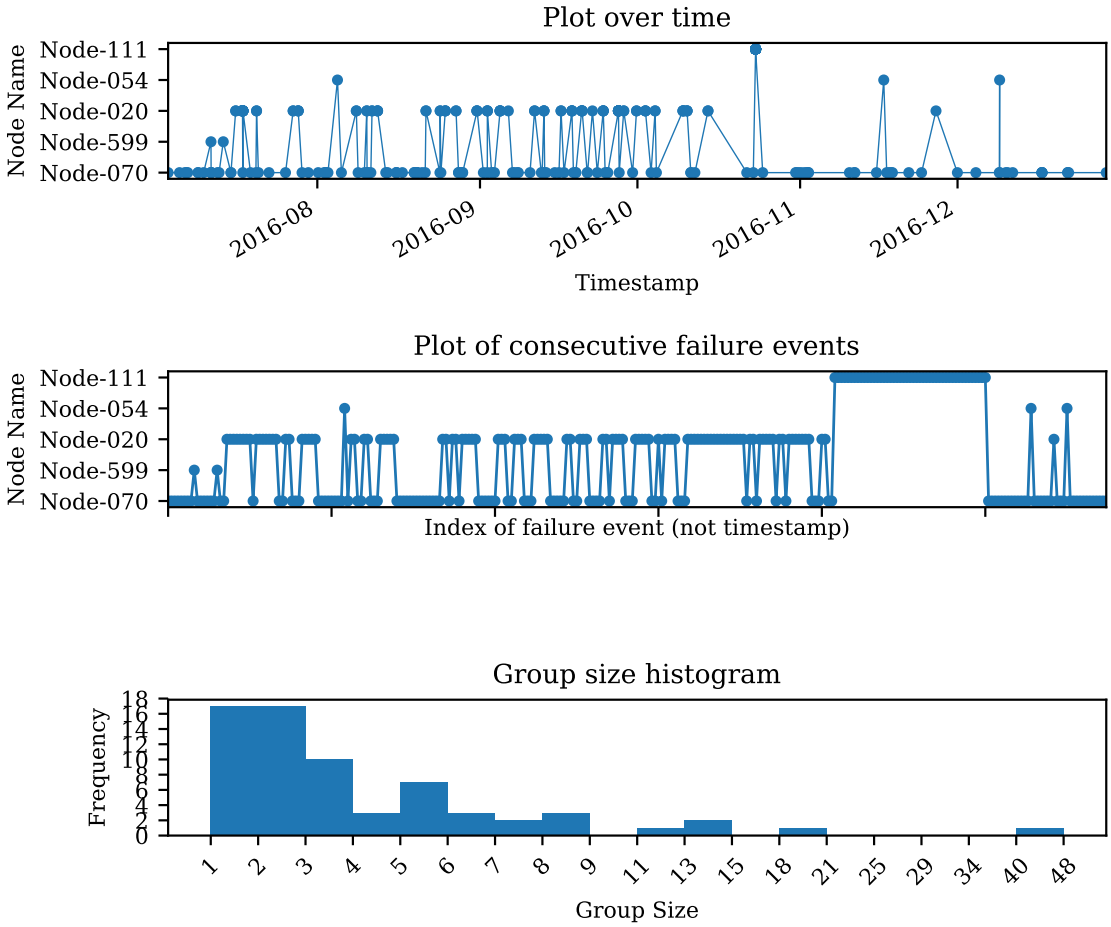

Figure 3.18: Node jumping analysis of failure type *SASDeviceTimeout*. The first graph is a plot of failure events over time (x-axis) and the node in which they occurred (y-axis). The second graph is a sequential plot of all the failure events in chronological order. x-axis is an index (sorted by Timestamp and SequenceID), and y-axis is the node on which the failure event occurred. Note that the X axes on the first and second graphs are different. In the first graph, many failure events may occur at a single X, but they will be spread out in the second graph (as each individual failure event has a separate X value). The third graph contains a histogram of lengths of failure event groups. The x-axis is the size of groups and y-axis is the frequency. The bin size of histogram is fixed at 25. Note the large number of small failure event groups in the histogram, indicating high node jumping.

Probability Representation

While the histogram representation of node jumping is detailed, it does not describe the phenomenon in a simple metric. We used probability to represent node jumping

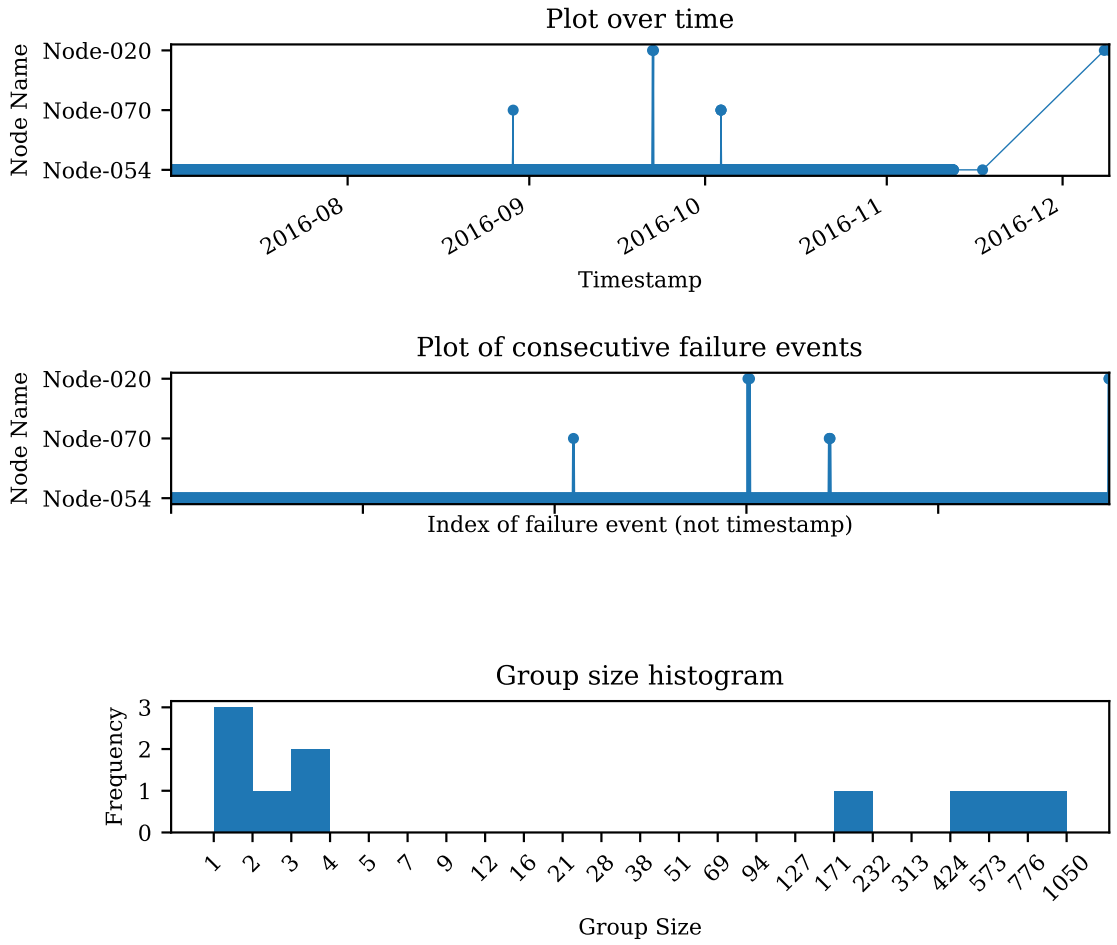

Figure 3.19: Node jumping analysis of failure type *LDAPConnectionFailed*. The first graph is a plot of failure events over time (x-axis) and the node in which they occurred (y-axis). The second graph is a sequential plot of all the failure events in chronological order. x-axis is an index (sorted by Timestamp and SequenceID from ems-events-*.csv), and y-axis is the node on which the failure event occurred. Note that the X axes on the first and second graphs are different. In the first graph, many failure events may occur at a single X, but they will be spread out in the second graph (as individual failure eventa have a separate X values). The third graph contains a histogram of lengths of failure event groups. The x-axis is the size of groups and y-axis is the frequency. The bin size of histogram is fixed at 25.

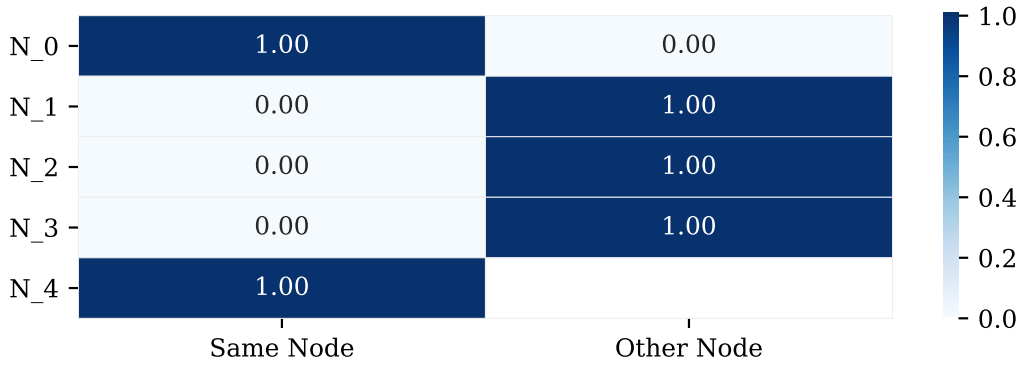

Figure 3.20: Node jumping probability of failure type *UncompletedDiskIO*. Node_0 is a pseudo-node which represents multiple nodes.

in a single metric. Instead of the raw data (list of failure events with time stamps), time-binned failure event data is used and node jumping is expressed as conditional probability: given that a failure type A occurred in a node in time bin x, $P(B|A)$ is the probability of a failure type B occurring on a different node in time bin $x + 1$. This is calculated with a bin size of one second, the lowest granularity available in the data, so that the ground truth is represented as accurately as possible. When using binned data. multiple failure events can also happen on multiple nodes within the same time bin, which is represented in a pseudo-node *Node_0*. Figure 3.20 shows an example of high node jumping and Figure 3.19 shows an example of low node jumping.

Finding 14: *Node jumping is rare but strong.* Most failure events have low node jumping but some failure events have strong node jumping.

It should be noted that this metric is sensitive to the number of failure events in each failure type. For failure types with small number of failure events (which is the case for many failure types), it is typical for the probabilities to be very strong (1 or 0), whereas for failure types with large number of failure events, the probabilities are more spread out.

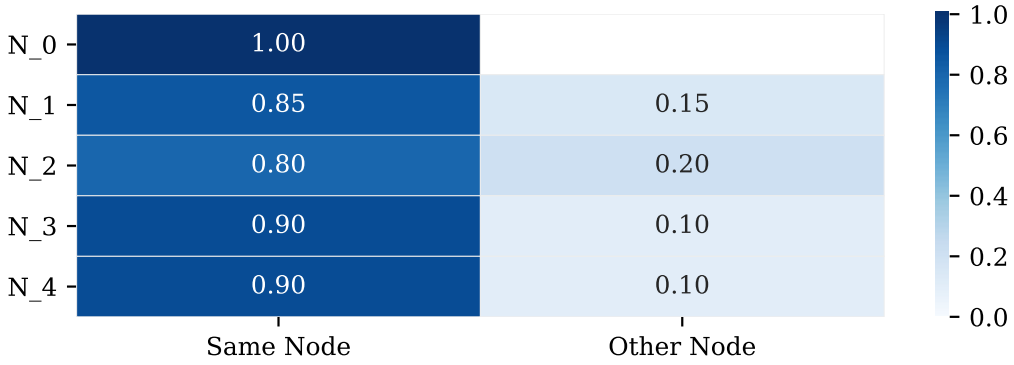

Figure 3.21: Node jumping probability of failure type *ShelfFailed*. Node_0 is a pseudo-node which represents multiple nodes.

Chapter 4

Prediction

Since data center hardware typically uses embedded low-power purpose-built processing units, there is not a lot of computing power to spare. Prediction algorithms based on machine learning or neural networks are becoming popular in recent years, but they still have training overhead. Moreover, machine learning based methods require a large amount of training data to be most effective. Considering the lightweight requirement of practical systems and the sparseness of data, we opted for a statistical approach in designing the prediction methodology which uses the spatial-temporal dependencies between various failure types. This section details the algorithm and its improvements which are designed to progressively increase accuracy.

4.1 Conditional Probability Methods

A prediction algorithm was developed based on the conditional probability maps discussed in Section 3.2.5. Each of the sections below explain the base algorithm and its incremental improvements.

4.1.1 Strawman

Pairwise conditional probability maps are created as discussed in Section 3.2.5 for lags of zero to nine. Then a *maximum probability map* is created, where for each failure type pair, the lag with the highest probability is picked. This map encodes information about which lag produces the largest conditional probability for each

failure type pair for a given window size. An overview of the prediction algorithm is as follows: With the maximum probability maps used as look up tables, incoming failure events are processed one at a time. For each failure event:

- Look up probability maps for the lag at which other failure types happen with $P(X \geq threshold)$, where threshold is set at 0.7
- Generate predictions, where each prediction states which failure type will happen on which time bin
- As time passes, validate predictions based on ground truth to determine accuracy

The algorithm is described in detail in Algorithm 4.2. It should be noted that the algorithm predicts whether or not a failure of a certain type will occur in a time bin, not individual failure events.

4.1.2 Combining lower probabilities

A large number of lower probability predictions might indicate the occurrence of a failure event, even if each individual probability is lower than *threshold*. To take advantage of this, lower probability predictions of the same failure type and time bin are combined to improve their overall probability. The algorithm for combining probabilities into a metric called $confidence$ is described in Algorithm 4.1.

The idea is to take the remaining chance of a perfect probability (1 - confidence) and multiply it with the probabilities to increase the confidence. Probabilities are processed in descending order so that higher probabilities would have a larger weight in the overall confidence.

4.1.3 Minimum threshold for combining lower probabilities

Although combining lower probabilities increased the success rate, the number of false positives also increased greatly. Large number of false positives are undesirable, as they might prompt unnecessary backups wasting resources. To combat this, we set

Algorithm 4.1 Calculate Confidence

```
 1: procedure Prediction.CalculateConfidence
 2:     if not combinePredictions then self.Confidence = max(self.Probabilities)
 3:     else
 4:         confidence = 0.0
 5:         for prob in sorted(self.Probabilities, reverse=True) do
 6:             confidence += (1 - confidence) * prob
 7:             if confidence >= 1.0 then
 8:                 break
 9:             end if
10:         end for
11:         self.Confidence = confidence
12:     end if
13: end procedure
```

another threshold, a lower bound, for combining lower probabilities. This variant of the algorithm combines only those lower probabilities which are higher than a minimum threshold (of 0.5).

4.1.4 Prediction Chaining

So far, new predictions are generated only from failure events which are not predicted successfully. *Prediction chaining* also uses the failure events that are successfully predicted to make further predictions.

For this method, a parameter called *hop limit* is used. Each time a prediction is made (say $P1$) based on a successful prediction ($P0$), it is called one hop. If that predicted failure event ($P1$) is used to make a new prediction ($P2$), that is the second hop, and so on. Without a hop limit (keep making predictions based on predicted failure events indefinitely), while the success rate increased, the number of false alarms also greatly increased. Limiting the number of hops, the number of false positives is greatly reduced while still achieving higher success rates. Understandably, the higher the hop limit, the higher the success rate but also the number of false positives.

4.1.5 Processing nodes separately

The conditional probability maps used so far include data from all the nodes in a cluster. This makes it impossible to specify on which node a failure type might occur. To address this, probability maps are created for each node separately and an additional property, the node name, is added to the predictions. This allows the predictions to be specific to each node within the cluster, adding spatial specificity.

4.1.6 Using node jumping maps

To take advantage of node jumping described in Section 3.4.3, additional probability maps are created which capture $P(N|M)$, where N and M are any two nodes, and $P(N|M)$ is the probability of a failure event jumping from node M to node N in the next time bin. These additional maps are used to determine on which node each prediction is made. For example, if the node jumping map indicates a jump to different node with a probability greater than 0.5, i.e., $P(N|M) \geq 0.5$ and $N \neq M$, the following steps are performed:

- Reduce the confidence of prediction on the current node by half
- Increase confidence of prediction on the destination node:
  - If a prediction for destination node doesn't exist, create a new prediction
  - If a prediction for destination node exists, increase its confidence

4.1.7 Algorithms

This section describes in detail the prediction algorithm and related functions.

Algorithm 4.2 Predict

```
1: procedure PREDICT(inputEvents, minThreshold, hopLimit)
2:     actuals = set()
3:     predictions = []
4:     lastBinNumber = None
5:     results = [0, 0, 0, 0, 0, 0]
6:     for event in inputEvents do
7:         isEventPredicted = False
8:         currentBinNumber = getCurrentTimeBin()
9:         nonEmptyBinsCount += 1
10:        if lastBinNumber != currentBinNumber then
11:            for pred in predictions do
12:                pred.Lag -= currentBinNumber - lastBinNumber
13:            end for
14:            evaluatePredictions()
15:            actuals = set()
16:            lastBinNumber = currentBinNumber
17:        end if
18:        actuals.add(event.EventType)
19:        pred = findPrediction(lag=0, resultantError=event.EventType)
20:        if pred is not None then
21:            pred.PredictionSuccessful = True
22:            isEventPredicted = True
23:        end if
24:        if not isEventPredicted then generatePredictions(event.EventType,
   event.NodeName, 0)
25:        else
26:            hopCount = pred.HopCount
27:            if hopLimit == None or hopCount < hopLimit then
28:                generatePredictions(event.EventType, event.NodeName, hopCount
   + 1)
29:            end if
30:        end if
31:    end for
32:    evaluatePredictions(finalEvaluation=True)
33:    fillPredictionsForMissingBins()
34:    return results
35: end procedure
```

Algorithm 4.3 Generate Predictions Part 1

```
1: procedure GeneratePredictions(causeEvent, nodeName, hopCount)
2:     preds = getPredictionsFromMap(nodeName, causeEvent)
3:     for (resultantError, conditions) in preds do
4:         timebin = conditions[0]
5:         lag = conditions[1]
6:         prob = conditions[2]
7:         if not combinePredictions and prob < minThreshold then
8:             continue
9:         end if
10:        if combinePredictions and prob ≤ combinedMinThreshold then
11:            continue
12:        end if
13:        pred = findPrediction(lag, resultantError, nodeName)
14:        if pred is not None then
15:            pred.addProb(prob)
16:            pred.calculateConfidence()
17:        else
18:            pred = new Prediction()
19:            pred.resultantError = resultantError
20:            pred.probabilities = [prob]
21:            pred.confidence = prob
22:            pred.lag = lag
23:            pred.hopCount = hopCount
24:            pred.nodeName = nodeName
25:            pred.predictionSuccessful = False
26:            pred.nodeProbability = None
27:            predictions.append(pred)
28:        end if
29:    end for
```

Algorithm 4.4 Generate Predictions Part 2

```
30:     if processPerNode = True then
31:         possibleDestsWithProbs = nodeJumpingMap[causeEvent][nodeFrom]
32:         nodeFrom = nodeName
33:         nodeTo = possibleDestsWithProbs.idxmax()
34:         nodeToProb = possibleDestsWithProbs.max()
35:         if nodeToProb ≥ 0.5 then
36:             pred = findPrediction(resultantError=causeEvent, lag=1, nodeName=nodeFrom)
37:             if nodeFrom != nodeTo then pred.Confidence *= 0.5
38:             end if
39:             pred = findPrediction(resultantError=causeEvent, lag=1, nodeName=nodeTo)
40:             if pred is None then
41:                 pred = new Prediction()
42:                 pred.resultantError = causeEvent
43:                 pred.probabilities = [nodeToProb]
44:                 pred.confidence = None
45:                 pred.lag = 1
46:                 pred.hopCount = hopCount
47:                 pred.nodeName = nodeTo
48:                 pred.predictionSuccessful = False
49:                 pred.nodeProbability = nodeToProb
50:                 pred.calculateConfidence()
51:                 predictions.append(pred)
52:             else
53:                 pred.addProb(nodeToProb)
54:                 pred.calculateConfidence()
55:             end if
56:         end if
57:     end if
58: end procedure
```

Algorithm 4.5 Evaluate Predictions

```
1: procedure EvaluatePredictions
2:     setActualPositives = actuals
3:     setActualNegatives = allEventTypes - setActualPositives
4:     truePositives = set()
5:     falsePositives = set()
6:     for indx, pred in enumerate(predictions) do
7:         if pred.Lag ≥ 0 then
8:             if not finalEvaluation then
9:                 continue
10:            end if
11:        end if
12:        if pred.Confidence ≥ minThreshold then
13:            if pred.PredictionSuccesful then
14:                truePositives.add( pred.ResultantError )
15:            else
16:                falsePositives.add( pred.ResultantError )
17:            end if
18:        end if
19:        deletePredictionFromList(pred)
20:    end for
21:    truePositives |= falsePositives & setActualPositives
22:    falsePositives -= setActualPositives
23:    setAllPositivePredictions = truePositives | falsePositives
24:    setAllNegativePredictions = allEventTypes - setAllPositivePredictions
25:    trueNegatives = setAllNegativePredictions & setActualNegatives
26:    falseNegatives = setAllNegativePredictions - setActualNegatives
27:    results[0] += len(truePositives)
28:    results[1] += len(trueNegatives)
29:    results[2] += len(falseNegatives)
30:    results[3] += len(falsePositives)
31:    results[4] += len(setActualPositives)
32:    results[5] += len(setActualNegatives)
33: end procedure
```

Chapter 5

Experimental Evaluation

The results of evaluation of the prediction algorithm and its improved versions are discussed in this section. Unless specified otherwise, the first half of data set (the first half year) is used to build conditional probability maps, and the algorithm is evaluated using the second half of the data set.

5.1 Metrics

A definition of some metrics and terminology is in order before the results are discussed:

- **+ve Success %**: The percentage of failure types that were predicted to occur, and did occur (true positive).
- **-ve Success %**: The percentage of failure types that were not predicted to occur, and did not occur (true negative).
- **Misses %**: The percentage of failure types that were not predicted to occur, but did occur (false negative).
- **False Alarms %**: The percentage of failure types that were predicted to occur, but did not occur (false positive).

The term "positive" means that the prediction is for a failure event occurring, and "negative" means that the prediction is for a failure event *not* occurring.

The denominator for each of the four metrics is the total number of evaluated time bins multiplied by the number of failure types that could happen per time bin (i.e., the number of failure types in the cluster or workload). As a sanity check, adding the number of failure type occurrences and non-occurrences should add up to the total possible slots. This is used to test that the algorithms are counting correctly.

The goal is to maximize the positive success percentage and the overall success percentage (percentage of predictions that were correct, regardless of whether the prediction was positive or negative) and minimize the false alarm and miss percentages.

The results discussed in the following sections are for WorkloadA, with a bin size of 2 hours.

5.2 Markov Chain Methods

Markov chains are a combination of states and transitions, with probabilities assigned to each transition [16]. An additional constraint is that the sum of all outgoing transitions should be 1. Three methods based on Markov chains are used as a baseline to compare with the algorithm.

5.2.1 2-state Markov chain

For each failure type, Markov chains are created with two states, one for the failure type occurring in the current time bin and the other for the failure type not occurring. Markov chains for each failure type are calculated and used to predict whether they occur in the next time bin. Higher order Markov chains up to order 3 were also tested, but the results did not vary from order 1 Markov chains.

5.2.2 Hidden Markov Model

Hidden Markov Models (HMM) [8] are also trained and used for each failure type. The +ve and -ve success percentages were less than 5% higher than Markov chains. Varying the number of hidden states did not seem to affect the results after 3 hidden

states.

5.2.3 Combining Sub-Windows

For both the methods discussed above, the results of using smaller bin sizes were combined into the target bin size. For example, for the 2-hour target bin size, the results of both 5-minute and 15-minute bin sizes were calculated and down-sampled to 2-hour bin sizes. For instance, eight 15-minute bins form one 2-hour bin. If a failure type occurred at least once in the 8 smaller windows, the failure type is considered to occur in the 2-hour bin. The results were 10-15% better using this method, but the run time was also higher due to processing a larger number of time bins. The results for Markov chain-based methods are shown in Figure 5.1. We see that Markov-Hi, i.e., combining smaller bins into target bins for the simple Markov chain method, performs the best, with a +ve success percent (amount of errors captured) of 58.72%.

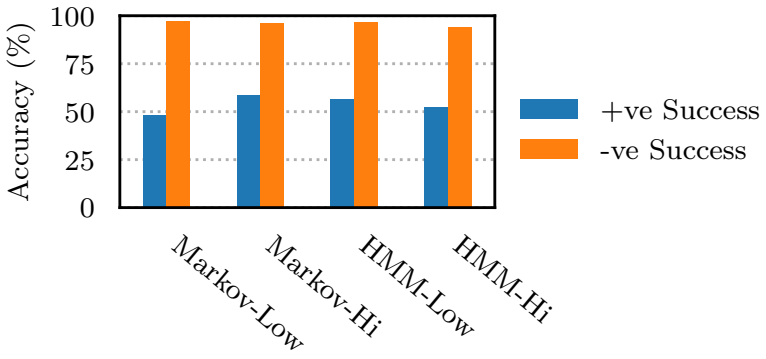

Figure 5.1: Positive and negative success percentages for Markov chain-based methods. The suffix -Low refers to using the target bin size. The suffix -Hi refers to using smaller bins and down-sampling them to the target bin size.

5.3 Conditional Probability Results

Figure 5.2 shows the results for the conditional probability algorithms described in Section 4. In the interest of space, some of the improvements are re-labelled as follows:

1. Strawman 1: Based on conditional probability maps (Section 4.1.1).
2. Strawman 2: Combining probabilities below *threshold* (Section 4.1.2).
3. Strawman 3: Combining only probabilities below *threshold* and above 0.5% (Section 4.1.3).
4. Prediction Chaining: Using predicted failure events to make further predictions (Section 4.1.4).

The results show that the positive success percentage for all of the conditional probability-based algorithms (starting at 69.62% for Strawman 1 and 87.2% for prediction chaining) are better than the best-case Markov chain result (58.72% for Markov-Hi).

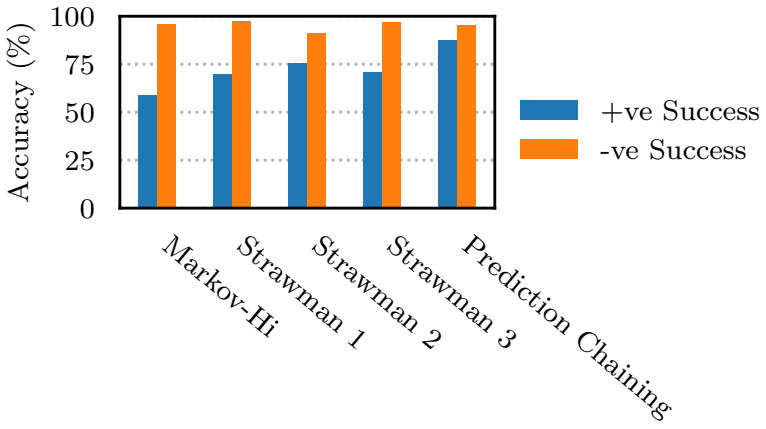

Figure 5.2: Positive and negative success percentages for conditional probability methods.

We see that combining probabilities below threshold (Strawman 2) captured more failure events (increased +ve success) but also produced many false alarms (decreased -ve success). Then, applying a minimum threshold for combining low confidence predictions (Strawman 3) reduces the false alarms (increases -ve success) but reduces the positive success rate. Prediction chaining has a tune-able parameter *hop_control* which controls the trade-off between positive success rate and false alarms. Figure 5.3 shows the results for prediction chaining with different hop limits. The graph shows

that the benefit of increasing the hop limit becomes less useful after a certain point, as there is not much difference between 25 hops and infinite hops.

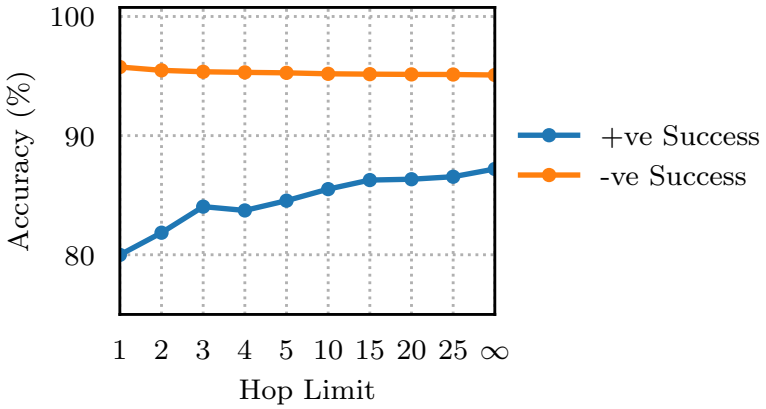

Figure 5.3: Positive and negative success percentages for Prediction Chaining with varying hop limits.

5.4 Per Node Results

The Per-Node algorithms use separate probability maps for each node, where each probability map is created using failure events only from that node. This localizes predictions to specific nodes. This is described in detail in section 4.1.5.

Figure 5.4 shows the results for Markov-chain methods and Figures 5.5 and 5.6 show the results for conditional probability methods.

We see that the success rates are lower for per-node versions when compared to running on all nodes by up to 19%. This is understandable, because the uncertainty from adding the spatial dimension reduces the overall success rate of predictions. Additionally, since probability maps are calculated for each node separately, less data is available for training conditional probability maps for each node, leading to less accurate maps. We also see that the differences between different versions of conditional probability-based algorithms are similar to running on all nodes together, with prediction chaining achieving the highest +ve success rate of 75.92%.

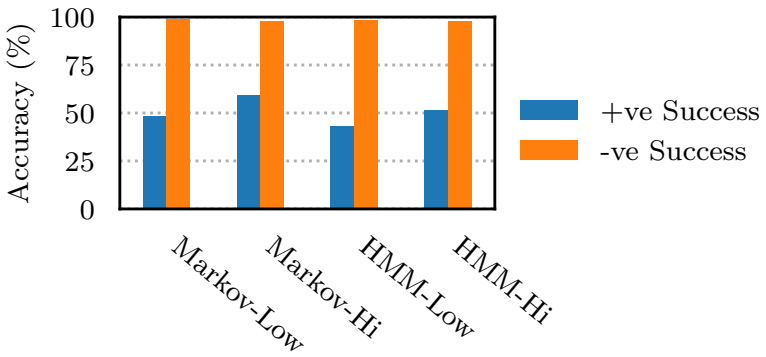

Figure 5.4: Positive and negative success percentages for Markov chain-based methods processed separately for each node.

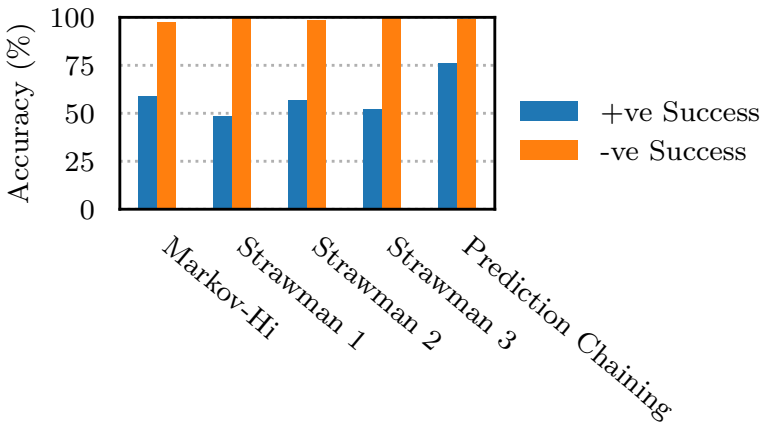

Figure 5.5: Positive and negative success percentages for Strawman methods, processing each node separately.

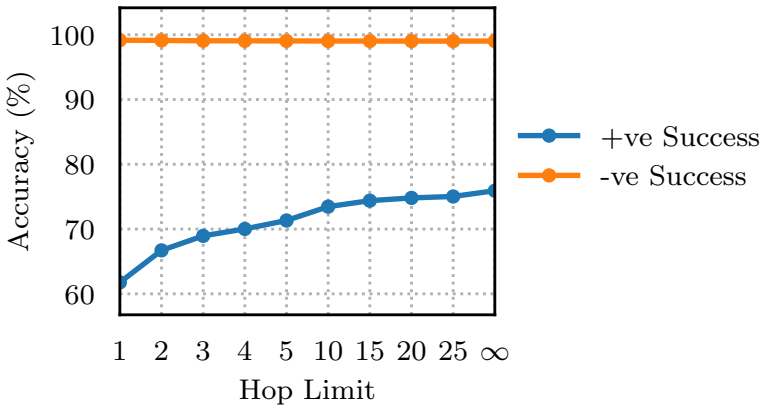

Figure 5.6: Positive and negative success percentages for Prediction Chaining with varying hop limits, processing each node separately.

5.5 Per Node with Node Jumping Results

Node Jumping probability maps are used to better predict the node in which a failure type might occur. This is described in Section 4.1.6. While most failure types are unaffected by this method, a few failure types showed significant improvement in success rates. The improvement percentage in positive success rates are shown in Figure 5.7.

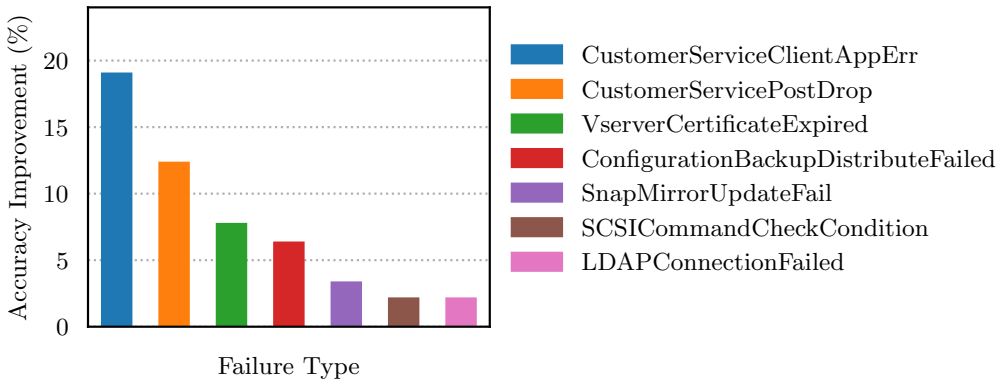

Figure 5.7: Improvement of positive success rate for failure types when using node jumping.

5.6 Dynamic Maps with Moving Window

So far, one large chunk of data is used to build conditional probability maps which are used as look up tables to make predictions on a test data set. While building maps using data from a long time period captures the overall conditional probability, it may not work well in unstable periods which deviate from the global pattern. A moving window methodology is implemented to help adapt to short-term changes. The main idea is to keep track of some recent history, then periodically re-train the probability maps based on it. This is implemented using the following two parameters:

- *window_size*: The size of the moving window in number of time bins
- *retrain_frequency*: Number of time bins after which probability maps are re-calculated using the data inside moving window

While using probabilities only from the moving window might adapt well changes, it might not always be desirable. This technique was used to deal with outlier groups of failure events which don't conform to the overall "default" behavior. Using only recent history would therefore be too sensitive to local changes and might miss global patterns. To address this, we implemented a weighting of probabilities from the fixed global probability maps and the dynamic moving window maps. A weighting of 70% for the moving window probabilities and 30% for the overall probabilities is used for testing. Graphs comparing results using fixed and dynamic maps for all nodes combined is shown in Figure 5.8, and for running per-node is shown in Figure 5.9.

It is clear from the graphs that using a dynamic window results in a higher positive success rate for both Markov-based and conditional probability-based methods. However, weighting global and dynamic probability maps does not seem to have a large effect on the positive success rate. With moving window, the +ve success rate for prediction chaining increased by 4.28%, and for per-node prediction chaining increased by about 8%.

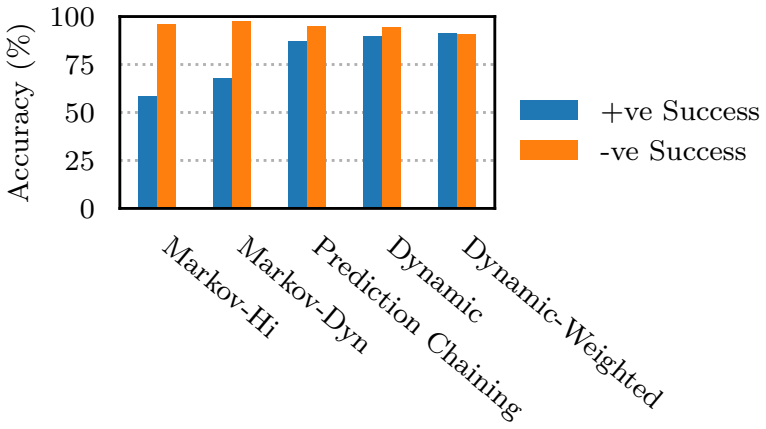

Figure 5.8: Comparison of using fixed maps and dynamic maps for all nodes combined. For the Markov-Dyn, the window size and retrain frequency are 100 and 10 respectively. For Dynamic and Dynamic-Weighted, the window_size and retrain_frequency are 10 and 10 respectively.

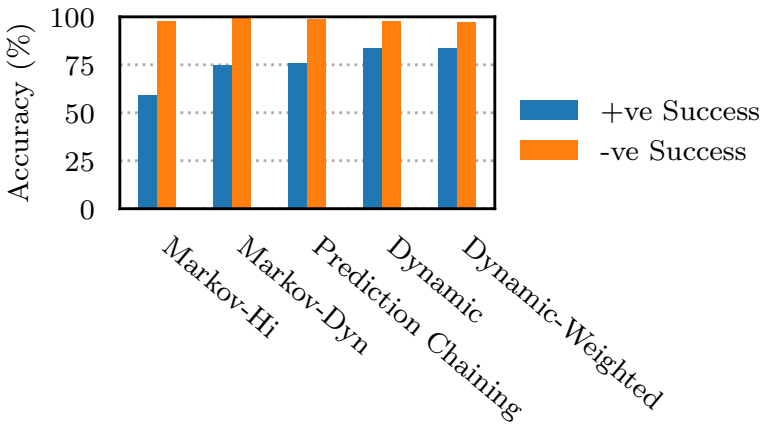

Figure 5.9: Comparison of using fixed maps and dynamic maps for separate nodes. For the Markov-Dyn, the window size and retrain frequency are 100 and 10 respectively. For Dynamic and Dynamic-Weighted, the window_size and retrain_frequency are 10 and 10 respectively.

Chapter 6

Conclusions and Future Work

6.1 Conclusion

This thesis characterized critical failure events of extreme-scale data storage systems. The characterization results show that failure events are temporally sparse, but the distribution of time between failure events exhibits a long tail, showing that quick bursts of failure events are very common. The overall failure pattern follows the 80/20 rule and demonstrates isolation. The 80/20 rule suggests that the majority of failures (80%) are caused by a small set of critical failure types (6%). In addition, the spatial distribution of failures at the shelf and device levels as well as the RAID group in logical dimension also follow the 80/20 rule. In the spatial-temporal dimension, most groups of successive failure events happen on individual devices. Isolation implies that a large number of failures can happen on isolated devices while the neighboring devices remain failure-free. This is counter-intuitive to what is commonly observed in storage systems, where data drives in close proximity tend to be affected by failures simultaneously. Temporal analysis revealed that many failure types are strongly autocorrelated and at least 80% of failure types are strongly cross-correlated with at least one other failure type within each cluster. This shows that temporal dependencies within and across failure types are common. These findings indicate the predictability of critical failure events in extreme-scale storage systems. We propose a failure prediction approach based on the above findings. The prediction is built on the conditional probabilities that reflect the temporal dependencies between fail-

ure types. We made several enhancements based on the characterization results to improve the prediction accuracy: 1) combining probabilities of correlated failures, 2) chaining predictions to capture propagation effects, 3) exploring spatial retention, 4) using moving window to better reflect the short-term changes and be robust to dynamic behaviors. The results show that the proposed prediction methods outperform state-of-the-art Markov chain-based methods by more than 20%. Moving window further improves the prediction of our method by up to 8%. The overall prediction accuracy of our method is 91.48%.

6.2 Future Work

The most important opportunity for future work lies in better leveraging on the findings of spatial as well as logical dimensions. While we utilized some spatial properties in this thesis, the main focus of this work is on the temporal dimension. In the future, we plan to apply more fine-grained spatial findings to make the predictions more accurate and robust. We also plan to develop a more sophisticated way to tune the parameters in our prediction algorithm.

Bibliography

[1] Jacob Benesty, Jingdong Chen, Yiteng Huang, and Israel Cohen. *Pearson correlation coefficient*. In *Noise Reduction in Speech Processing*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009, pages 1–4. ISBN: 978-3-642-00296-0. DOI: 10.1007/978-3-642-00296-0_5. URL: https://doi.org/10.1007/978-3-642-00296-0_5.

[2] Paul Bourke. Cross correlation. URL: http://paulbourke.net/miscellaneous/correlate/ (visited on 08/07/2018).

[3] Matthew Conlen. Kernel density estimation. URL: https://mathisonian.github.io/kde/ (visited on 08/08/2018).

[4] Dealing with data: using nvivo in the qualitative data analysis process. URL: http://www.qualitative-research.net/index.php/fqs/article/view/865/1880&q=nvivo+manual&sa=x&ei=zah_t5pqoyubhqfe9swgbq&ved=0cc4qfjaj (visited on 08/06/2018).

[5] Different types of hard drive failure and data recovery solutions. 2017. URL: https://www.stellarinfo.co.in/blog/hard-drive-failure-and-data-recovery-solutions/ (visited on 08/06/2018).

[6] J. G. Elerath and S. Shah. Disk drive reliability case study: dependence upon head fly-height and quantity of heads. In *Annual Reliability and Maintainability Symposium, 2003.* Pages 608–612, 2003. DOI: 10.1109/RAMS.2003.1182057.

[7] J. G. Elerath and S. Shah. Server class disk drives: how reliable are they? In *Annual Symposium Reliability and Maintainability, 2004 - RAMS*, pages 151–156, 2004. DOI: 10.1109/RAMS.2004.1285439.

[8] Zoubin Ghahramani. An introduction to hidden markov models and bayesian networks. *International Journal of Pattern Recognition and Artificial Intelligence*, 15(01):9–42, 2001. DOI: 10.1142/S0218001401000836. eprint: https://doi.org/10.1142/S0218001401000836. URL: https://doi.org/10.1142/S0218001401000836.

[9] Jim Gray. Empirical Measurements of Disk Failure Rates and Error Rates. Technical report, 2005, page 3. URL: https://www.microsoft.com/en-us/research/publication/empirical-measurements-of-disk-failure-rates-and-error-rates/.

[10] Haryadi S. Gunawi, Riza O. Suminto, Russell Sears, Casey Golliher, Swaminathan Sundararaman, Xing Lin, Tim Emami, Weiguang Sheng, Nematollah Bidokhti, Caitie McCaffrey, Gary Grider, Parks M. Fields, Kevin Harms, Robert B. Ross, Andree Jacobson, Robert Ricci, Kirk Webb, Peter Alvaro, H. Birali Runesha, Mingzhe Hao, and Huaicheng Li. Fail-slow at scale: evidence of hardware performance faults in large production systems. In *16th USENIX Conference on File and Storage Technologies (FAST 18)*, pages 1–14, Oakland, CA. USENIX Association, 2018. ISBN: 978-1-931971-42-3. URL: `https://www.usenix.org/conference/fast18/presentation/gunawi`.

[11] Greg Hamerly and Charles Elkan. Bayesian approaches to failure prediction for disk drives. In *ICML*, 2001.

[12] Mingzhe Hao, Gokul Soundararajan, Deepak Kenchammana-Hosekote, Andrew A. Chien, and Haryadi 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)*, pages 263–276, Santa Clara, CA. USENIX Association, 2016. ISBN: 978-1-931971-28-7. URL: `https://www.usenix.org/conference/fast16/technical-sessions/presentation/hao`.

[13] Kazuhiko Ikeuchi, Hidejirou Daikokuya, Chikashi Maeda, Norihide Kubota, Atsushi Igashira, Kenji Kobayashi, and Ryota Tsukahara. Storage system, apparatus, and method for failure recovery during unsuccessful rebuild process, 2015. US Patent 8,943,358.

[14] Weihang Jiang, Chongfeng Hu, Yuanyuan Zhou, and Arkady Kanevsky. Are disks the dominant contributor for storage failures?: a comprehensive study of storage subsystem failure characteristics. *Trans. Storage*, 4(3):7:1–7:25, 2008. ISSN: 1553-3077. DOI: `10.1145/1416944.1416946`. URL: `http://doi.acm.org/10.1145/1416944.1416946`.

[15] Xiaolong Jin, Benjamin W. Wah, Xueqi Cheng, and Yuanzhuo Wang. Significance and challenges of big data research. *Big Data Res.*, 2(2):59–64, June 2015. ISSN: 2214-5796. DOI: `10.1016/j.bdr.2015.01.006`. URL: `http://dx.doi.org/10.1016/j.bdr.2015.01.006`.

[16] Markov chains explained visually. 2014. URL: `http://setosa.io/ev/markov-chains`. [Online; accessed 8. Aug. 2018].

[17] Viktor Mayer-Schnberger. *Big Data: A Revolution That Will Transform How We Live, Work and Think. Viktor Mayer-Schnberger and Kenneth Cukier*. John Murray Publishers, UK, 2013. ISBN: 1848547927, 9781848547926.

[18] Joseph F. Murray, Gordon F. Hughes, and Kenneth Kreutz-Delgado. Machine learning methods for predicting failures in hard drives: a multiple-instance application. *J. Mach. Learn. Res.*, 6:783–816, December 2005. ISSN: 1532-4435. URL: `http://dl.acm.org/citation.cfm?id=1046920.1088699`.

[19] Richard C Myers and Randolph E Stiarwalt. Disk preservation and failure prevention in a raid array, 2016. US Patent App. 14/621,891.

[20] Netapp fas 8000. URL: https://www.netapp.com/us/products/storage-systems/hybrid-flash-array/fas8000.aspx (visited on 08/06/2018).

[21] Jehan-François Pâris, Ahmed Amer, Darrell DE Long, and Thomas JE Schwarz. Self-repairing disk arrays. *arXiv preprint arXiv:1501.00513*, 2015.

[22] Eduardo Pinheiro, Wolf-Dietrich Weber, and Luiz André Barroso. Failure trends in a large disk drive population. In *FAST*, volume 7 of number 1, pages 17–23, 2007.

[23] Alma Riska and Erik Riedel. Evaluation of disk-level workloads at different time-scales. In *Proceedings of the 2009 IEEE International Symposium on Workload Characterization (IISWC)*, IISWC '09, pages 158–167, Washington, DC, USA. IEEE Computer Society, 2009. ISBN: 978-1-4244-5156-2. DOI: 10.1109/IISWC.2009.5306787. URL: https://doi.org/10.1109/IISWC.2009.5306787.

[24] S.M. Ross. *A First Course in Probability*. Pearson Prentice Hall, 2010. ISBN: 9780136033134. URL: https://books.google.com/books?id=Bc1FAQAAIAAJ.

[25] A. Ros, L. Y. Chen, and W. Binder. Failure analysis and prediction for big-data systems. *IEEE Transactions on Services Computing*, 10(6):984–998, 2017. ISSN: 1939-1374. DOI: 10.1109/TSC.2016.2543718.

[26] Robert Sanders. The pareto principle: its use and abuse. *Journal of Services Marketing*, 1(2):37–40, 1987. DOI: 10.1108/eb024706. eprint: https://doi.org/10.1108/eb024706. URL: https://doi.org/10.1108/eb024706.

[27] Bianca Schroeder and Garth A Gibson. Understanding failures in petascale computers. *Journal of Physics: Conference Series*, 78(1):012022, 2007. URL: http://stacks.iop.org/1742-6596/78/i=1/a=012022.

[28] Stephanie. What is correlation in statistics? correlation analysis explained. URL: http://www.statisticshowto.com/what-is-correlation/ (visited on 08/07/2018).

[29] Liz Tay. Inside ebay's 90pb data warehouse. 2013. URL: https://www.itnews.com.au/news/inside-ebay8217s-90pb-data-warehouse-342615 (visited on 08/06/2018).

[30] Vamsi Vankamamidi, Ryan Gadsby, Thomas E Linnell, David W Harvey, Daniel Cummins, and Steven Morley. Using spare disk drives to overprovision raid groups, 2018. US Patent 9,921,912.

[31] www.itl.nist.gov. Autocorrelation. URL: https://www.itl.nist.gov/div898/handbook/eda/section3/eda35c.htm (visited on 08/07/2018).

[32] Ji Xue, Feng Yan, Robert Birke, Lydia Y. Chen, Thomas Scherer, and Evgenia Smirni. Practise: robust prediction of data center time series. *2015 11th International Conference on Network and Service Management (CNSM)*:126–134, 2015.

[33] Feng Yan, Xenia Mountrouidou, Alma Riska, and Evgenia Smirni. Quantitative estimation of the performance delay with propagation effects in disk power savings. In *Proceedings of the 2012 USENIX Conference on Power-Aware Computing and Systems*, HotPower'12, pages 5–5, Hollywood, CA. USENIX Association, 2012. URL: http://dl.acm.org/citation.cfm?id=2387869.2387874.

[34] 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 2013)(MSST)*, volume 00, pages 1–5, May 2013. DOI: 10.1109/MSST.2013.6558427. URL: doi.ieeecomputersociety.org/10.1109/MSST.2013.6558427.