Toward Resilience in High Performance Computing: A Prototype to Analyze and Predict System Behavior

Dissertation

zur Erlangung des akademischen Grades
Doktoringenieur (Dr.-Ing.)

vorgelegt an der
Technischen Universität Dresden
Fakultät Informatik

eingereicht von
Siavash Ghiasvand

Gutachter:	Prof. Dr. rer. nat. Wolgang E. Nagel Technische Universität Dresden
	Prof. Dr. rer. nat. Martin Schulz Technische Universität München

16 September 2020

Abstract

Following the growth of high performance computing systems (HPC) in size and complexity, and the advent of faster and more complex Exascale systems, failures became the norm rather than the exception. Hence, the protection mechanisms need to be improved. The most de facto mechanisms such as checkpoint/restart or redundancy may also fail to support the continuous operation of future HPC systems in the presence of failures. Failure prediction is a new protection approach that is beneficial for HPC systems with a short mean time between failure. The failure prediction mechanism extends the existing protection mechanisms via the dynamic adjustment of the protection level. This work provides a prototype to analyze and predict system behavior using statistical analysis to pave the path toward resilience in HPC systems. The proposed anomaly detection method is noise-tolerant by design and produces accurate results with as little as 30 minutes of historical data. Machine learning models complement the main approach and further improve the accuracy of failure predictions up to 85%. The fully automatic unsupervised behavior analysis approach, proposed in this work, is a novel solution to protect future extreme-scale systems against failures.

Contents

1 Introduction

The increasing demand for higher computation power, leads to production of more complex computing units. According to Moore's Law [1], the number of transistors per square centimeter on integrated circuits are doubled every two years thus, the computational performance. The Moore's Law holds true since 1965, and it is expected to remain valid in near future[1]. However, despite the higher computation power of complex computing units, this complexity contributes to instability and error proneness of components as well as the entire computing system.

On a higher granularity, failure rate in high performance computing (HPC) systems rapidly increases due to the growth in system size and complexity. Hence, failures became the norm rather than the exception [2, 3, 4, 5, 6]. The efficiency of failure recovery mechanisms, e.g., checkpoint-restart, is highly dependent on the mean time between failure (MTBF). With the arrival of Exascale computers in the near future [7], the MTBF of HPC systems is expected to become too short, and that current failure recovery mechanisms will no longer be able to efficiently protect the systems against failures [8, 9, 10, 11, 12, 13].

Early failure detection is a new class of failure recovery methods which is in particular beneficial for large computing systems with short MTBF [14, 15]. Detecting failures in their early stage can reduce their negative effects via barricading their propagation [16]. This work provides a prototype to analyze and predict system behavior. The behavioral analysis is then used to detect node-level failures as early as possible which paves the way toward resilience in Exascale high performance computing.

1.1 Background and Statement of the Problem

Although the distribution, origin, and cause of failures have changed throughout the years, addressing failures remained an important challenge in computing systems [17, 18]. Reliability of computing systems is becoming more and more important as the demand for higher computing performance is increasing. To fulfill the performance requirements of new algorithms and software, the computing units became complex and dense. Furthermore, additional computing units are employed by HPC systems. High complexity and density of computing units and higher number of components in computing systems, as well as aggressive power management approaches such as dynamic frequency scaling (CPU throttling) highly contribute to HPC systems error proneness [19].

Various approaches are proposed to address failures in HPC systems. Regardless of the system layer of application, existing general-purpose approaches can be categorized

[1]Although the transistors may not shrink their size anymore, the computational performance tends to double every two years.

in three main categories of: (1) checkpointing, (2) replication, and (3) failure prediction[2]. The first two categories (checkpointing and replication) are currently de facto approaches to address failures in production HPC systems. Although these categories are shown separately, in most cases an integrated approach consisting both checkpointing and replication is being used [20].

Statistics indicate a persistent number of critical failures in major production HPC systems. Table 2.1 on page 8 summarizes the failure statistics of various HPC systems over the course of 37 years (1948-2020). Hardware unreliability is one of the main sources of system failures in both industrial and commodity hardware [21, 22, 23]. The failure rate of commodity components in most computing systems can be described using the Bathtub Curve [24] hazard function shown in Figure 1.1 [25]. Furthermore, the same relation is observed between failure rate and the system up-time. In Chapter 4 the bathtub curve concept is used to adjust the anomaly detection method.

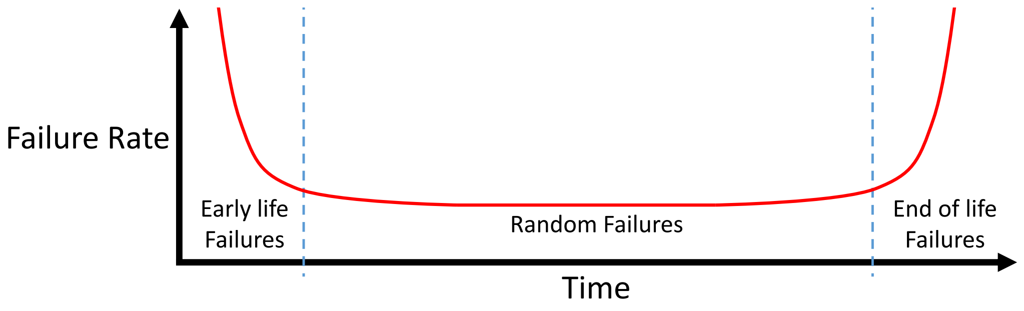


Figure 1.1: The *Bathtub Curve* hazard function

Exascale HPC systems are expected to arrive by 2020 [7]. Current failure statistics indicate the dire need of failure mitigation mechanisms in Exascale systems. In various researches the MTBF is projected to be in the range of seconds and minutes instead of days[3] [2, 3, 27, 28, 29, 30, 31, 32].

Considering the short mean time to interrupt (MTTI) and the large size of Exascale HPC systems, none of the existing approaches in their current form may remain beneficial to address failures in the future. The checkpointing mechanisms require a certain amount of time to generate snapshots of the current system status ($T_{Checkpoint}$) and to restore the snapshot after each failure ($T_{Restore}$)[4]. Decreasing the MTTI to less that $T_{Checkpoint}$ or $T_{Restore}$ prevents further progress of applications. Full replication of the system status is also not cost-effective regarding the large size of the future Exascale HPC systems. Additional challenges such as consistency, synchronization, and network congestion which are side effects of checkpointing and replication approaches may in turn introduce additional failures in HPC systems.

[2]This categorization considers general purpose approaches and does not include domain-specific and special-purpose methods such as algorithm-based fault tolerance (ABFT).

[3]Exponential distribution is often used to predict the time interval until future events (failure). This distribution predicts the time until the first failure. Gamma distribution, on the other hand, predicts the time interval until the i$_{th}$ failure occurrence. Weibull, Gamma, and Pearson6 are the best-fit distributions for the MTBF [26].

[4]Asynchronous (live) methods can reduce the required checkpoint/restart time.

It is important to note that most failure statistics shown in Table 2.1 belong to HPC systems that operate in highly controlled environments. These HPC systems enforce strict usage regulations and are under constant monitoring and maintenance by administration personnel which regardless of the high imposed expenses, greatly contribute to systems reliability. The aim is to fully automate the system monitoring and optimize the maintenance periods according to systems behavior, and yet preserve the systems functionality.

1.2 Purpose and Significance of the Study

Operational HPC systems are facing failures on a daily basis, regardless of their size and application. Failures in HPC systems are often correlated. In many cases these correlations cause a failure chain[5] to form. Regular maintenance as well as hardware and software upgrades constantly change the system's behavior. Analyzing and predicting the system behavior can effectively improve the HPC systems uptime, and reduce penalties imposed by regular failures[6]. Furthermore, automatic monitoring of the system reduces the maintenance costs and increases the system's functionality. The arrival of Exascale HPC systems with their massive number of components further reveals the importance of failure prediction for current and future computing systems.

This work provides a prototype to analyze and predict system behavior in order to detect node-level failures as early as possible. The proposed approach respects the users and system's privacy and is applicable on operational HPC systems without further modifications. Behavioral analysis enables the system to consider protective measures such as checkpointing, redundancy, and migration in useful time to stabilize the systems status and prevent further damages. Due to the large size of computing systems and the high volume of monitoring data, the proposed approach performs the main tasks fully automatic or with minimum interactions.

It is important to emphasis that this work does not intend to introduce a replacement for de facto failure protection mechanisms. Instead, this work provides a prototype to analyze and predict systems behavior, to improve the functionality of available mechanisms. In another word, this work proposes an adaptive resilience approach that employs appropriate protective measures according to the system's behavior.

The results of this work contribute to answer questions including the followings: Are the failures in HPC systems predictable? Which failures can be detected or predicted? Is it possible to provide a general behavioral analysis method? How to expand this approach to protect the Exascale computing systems? Which monitoring data has the required information for behavioral analysis? How to comply with data protection regulations (GDPR)? How to process and store the monitoring data for longer periods?

[5]Series of correlated failures (Section 3.1).
[6]An unintential failure caused by internal factors (Section 3.4).

1.3 *Jam–e Jam*: A System Behavior Analyzer

This work introduces *Jam–e Jam*[7], a prototype to analyze and predict system behavior to detect node-level failures as early as possible, in order to employ appropriate protective measures in useful time. The system log (syslog) entries are chosen as the main source of monitoring data in this work, due to their availability and information richness. A comprehensive anonymization technique is proposed to address privacy concerns raised by syslog analysis. The proposed approach is applicable to HPC systems without any further modifications. To provide an automatic approach suitable for extremely large HPC systems, *Jam–e Jam* utilizes statistical analysis and failure correlations among nodes with similar characteristics (node vicinity) to detect abnormal behaviors and predict upcoming node failures.

Figure 1.2 illustrates an example of system behavior analysis performed on Taurus. Timely detection of *anomalies* could have reduced the damages caused by failures via activating the protection mechanisms before the occurrence of the *failure*.

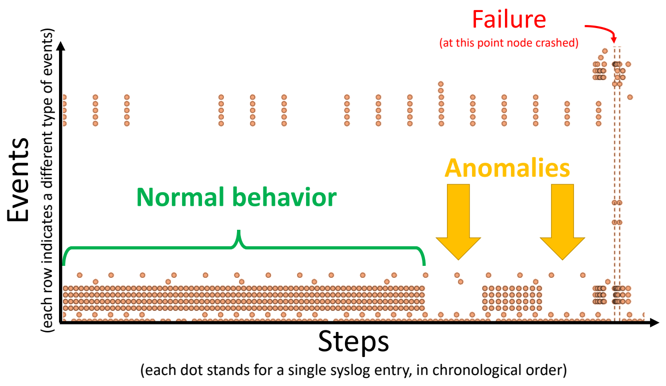

Failure
(at this point node crashed)
Events
(each row indicates a different type of events)
Anomalies
Normal behavior
Steps
(each dot stands for a single syslog entry, in chronological order)

Figure 1.2: The time interval between detection of an anomaly and occurrence of the subsequent failure (golden interval). The goal is to react during the golden interval to reduce the damages caused by failures via timely activation of the failure recovery mechanisms.

Jam–e Jam's approach can be likened to the function of smoke detectors in buildings. The smoke detectors activate protection mechanisms upon detection of smoke (the anomaly) to reduce the damages of a potential fire (the failure).

Although the behavior analysis in this work is aimed for early detection of node-level failures, the proposed mechanism can be used to detect other anomalies too. Furthermore, syslog entries, that are used as the main monitoring data in this work, can be replaced with any other data source as long as it provides required information relevant to the granularity of the expected anomalies.

The remainder of this work is structured as follows. Chapter 2 provides a review of the literature. Details of the data collection and preparation process are provided in Chapter 3. The methods are described in Chapter 4 and the major findings are explained in Chapter 5. Chapter 6 concludes the work and specifies the important future work directions.

[7]Proper noun; a cup of divination in ancient mythology. Pronounced as Jām-e Jam.

2 Review of the Literature

With drastic increase in the number of HPC system components, it is expected to observe a sudden increase in the number of failures, which consequently poses a threat to the continuous operation of the HPC systems. Current statistics show a persistent number of critical failures in major production HPC systems. Hardware failures are responsible for the majority of long-lasting down times in computing systems. As an example, the root causes of network failures in data centers are shown in Figure 2.1.

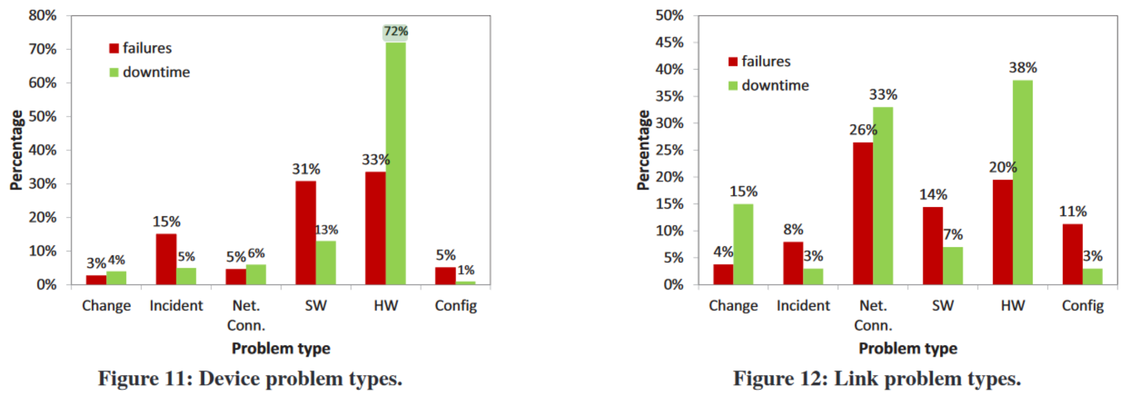


Figure 2.1: Root causes of network failures in data centers [33]. Failures are defined as any unsuccessful link in the network. HW and SW stand for hardware and software respectively.

Table 2.1 summarizes 37 years of HPC systems failure statistics. The inconsistency of failure assessments and the heterogeneity of HPC systems shown in Table 2.1 make it difficult to perform a one-to-one comparison. However, it can be concluded that failures are integral part of HPC systems. Therefore, failures must be addressed in order to guarantee efficient progress of computations on HPC systems. The studies summarized in Table 2.1 conclude that large scale regular failures are mostly caused by file systems however, the main cause of failures is hardware. The resiliency in HPC systems plays a vital role in comparison to other computing systems due to the complexity of workflows, applications and the requirements of HPC systems. However, similar techniques can be applied on all types of computing systems.

In summary, failures became an integral part of computing systems. Although, despite imposing excessive overheads, existing failure recovery mechanisms such as checkpointing-restart and redundancy are still useful for current HPC systems, it is predicted that future HPC systems will not be able to perform efficient forward progress while running large applications due to their ever reducing MTBF. To ensure the aliveness of HPC systems in the existence of failures, additional protective measures must be taken.

Table 2.1: High performance computing systems failure statistics from 1984 to 2020

Date	Environment	Observation
1984-1986	IBM Mainframe [34]	456 failures
1988-1990	Tandem [35, 36]	800 failures
1990	VAX [37]	364 failures
1989-1990	VICE [38]	300 failures
1999	70 Windows NT nodes [39]	1100 failures
1999	503 nodes [40]	2127 failures
2003	3000 nodes [41]	501 failures
2003-2004	395 nodes [42]	1285 failures
2004-2005	Liberty [43]	7.8 alerts per day
2005-2006	Blue Gene/L [43]	1,620 alerts per day
2005-2006	Thunderbird [43]	13,312 alerts per day
2006	Red Storm [43]	16,016 alerts per day
2005-2007	Spirit (ICC2) [43]	309,707 alerts per day
2007	BlueGene/L Coastal [44]	MTBF 7-10 days
2007-2009	Unknown [45]	MTBF 3-37 minutes
2005-2010	CENIC [46]	16-302 failures per link
2008-2010	Jaguar [47]	2.33 failures per day
2008-2011	Jaguar XT4 [18]	MTBF 36.91 hours
2008-2011	Jaguar XT5 [18]	MTBF 22.67 hours
2008-2011	Jaguar XK6 [18]	MTBF 8.93 hours
2012-2013	K Computer [48]	Failures rate 1.6%
2013	Blue Waters [49]	MTBF 4.2 hours
2014	Titan [50]	317 HW and 270 SW failures
2014	Titan [51]	9 failures per day
2013-2015	EOS XC 30 [18]	MTBF 189.04 hours
2013-2015	Titan XK7 [18]	MTBF 14.51 hours
2015	BlueGene/Q Mira [52]	MTBF 5.5 hours
2015	Petascale systems [53]	MTBF 7-10 hours
2015	Cielo [54, 55]	MTBF 24 hours
2011-2017	Facebook network [56]	MTBF 1.8 months
2016-2017	Beocat [57]	10% job failure
2017	Argonne FUSION [58]	MTBF 3-52 minutes
2013-2018	IBM Blue Gene/Q Mira [5]	MTBF 1.3 days (99,245 job failure)
2014-2018	Titan [59]	MTBF few minutes
2015-2018	Titan [60]	164,593 alerts per day
2020	TaihuLight [61]	MTBF relatively short
2020	Summit [62]	MTBF several hours

2.1 Syslog Analysis

Syslog was developed in the 1980s and since then became the de facto standard for logging systems activities. RFC 3164 [63] first documented the details of the syslog protocol, and later RFC 5424 [64] standardized it. All data fields in syslog entries are structured, except the message field, which contains free-form text. Each syslog entry begins with a `PRIVAL` (priority value) that represents the `severity` and the `facility` (origin) of the message. The PRIVAL is calculated by multiplying the facility number by 8 and adding the severity value to it. Table 2.2 provides an overview of syslog de facto facility codes and severity levels used in Unix-like operating systems.

Table 2.2: syslog facility names and severity levels as described in RFC 5424

Code	Keyword	Facility	Keyword	Severity
0	kern	kernel messages	emerg	Emergency: system is unusable
1	user	user-level messages	alert	Alert: action must be taken immediately
2	mail	mail system	crit	Critical: critical conditions
3	daemon	system daemons	err	Error: error conditions
4	auth	security/authorization messages	warning	Warning: warning conditions
5	syslog	messages generated internally by syslogd	notice	Notice: normal but significant condition
6	lpr	line printer subsystem	info	Informational: informational messages
7	news	network news subsystem	debug	Debug: debug-level messages
8	uucp	UUCP subsystem		
9	cron	clock daemon		
10	authpriv	security/authorization messages		
11	ftp	FTP daemon		
12	ntp	NTP subsystem		
13	security	log audit		
14	console	log alert		
15	solaris-cron	clock daemon		
16-23	local0-local7	local use 0-7		

Due to the simplicity of the syslog protocol and its usefulness, wide range of computing systems generate their system logs according to the syslog protocol. The information richness of syslog entries and the availability of syslogs on various computing systems made them a good candidate for systems monitoring. Apart from the operating systems and system-level services, the majority of the commodity applications also report their execution status in detail through log generation. On average, between 1% and 5% of the software source code is dedicated to log generation [65].

Various studies utilize syslog entries for behavioral analysis and anomaly detection. There are available platforms such as Elastic Stack [66], CloudSeer [67], and LogSCAN [68] which provide comprehensive analytics toolkit for administrators in order to have a better accessibility to system logs.

Parsing the syslog entries is a challenging task due to the existence of free-form text within the message part of the syslog entries. Even though that the syslog entries produced by each software are generated via predefined message templates, these templates are not uniform across various software. Furthermore, the large number of variables used in message templates contributes to the complexity of the pattern detection, e.g., the `Ubuntu` operating system reports 70,506 variables in its log entries and `Openssh` reports 3,290 variables [65]. Therefore, throughout the years various tools and platforms were developed to collect, digest, and analyze the log messages.

Several research projects attempt to automate the parsing of unstructured log messages. Most of the proposed algorithms where exclusively designed for handling system logs. SLCT [69] and LFA [70] algorithms use frequent item set mining to detect similar patterns among syslog entries. AEL [71] and LKE [72] use heuristic methods to detect variables. IPLoM [73] utilizes iterative partitioning based on word count. LogSig [74] generates system events from textual log messages via detecting the most representative message signatures. SHISO [75] proposes a structured tree capable of refining log formats in realtime. LogCluster [69] applies hierarchical clustering to cluster system logs. LenMa [76] detects message patterns using the length of words in each message. LogMine [77] employs a multiple sequence alignment algorithm to group similar log messages. Spell [78] and Drain [79] provide online parsing methods, Spell utilizes a longest common sub-sequence based approach to detect log patterns, while Drain uses a fixed depth parse tree. MoLFI [80] in other hand uses the domain knowledge and performs a reverse search in the space of solutions for a Pareto optimal set of message templates.

Among these algorithms the source code of Drain and LenMa as online log parsers are available. Some other log parsing algorithms are re-implemented in other studies[1]. Comparison of the parsing accuracy of these 13 algorithms on different log datasets is shown in Table 2.3 [81, 82]. Despite the high parsing accuracy on certain software logs such as Apache web server, all algorithms perform poorly on Linux system log (syslog)s which is the main source of monitoring data in this work [83].

Table 2.3: Accuracy of log parsers on different datasets [82]

Dataset	SLCT	AEL	IPLoM	LKE	LFA	LogSig	SHISO	LogCluster	LenMa	LogMine	Spell	Drain	MoLFI
HDFS	0.545	0.998	1	1	0.885	0.850	0.998	0.546	0.998	0.851	1	0.998	0.998
Hadoop	0.423	0.538	0.954	0.670	0.900	0.633	0.867	0.563	0.885	0.870	0.778	0.948	0.957
Spark	0.685	0.905	0.920	0.634	0.994	0.544	0.906	0.799	0.884	0.576	0.905	0.920	0.418
Zookeeper	0.726	0.921	0.962	0.438	0.839	0.738	0.660	0.732	0.841	0.688	0.964	0.967	0.839
OpenStack	0.867	0.758	0.871	0.787	0.200	0.200	0.722	0.696	0.743	0.743	0.764	0.733	0.213
BGL	0.573	0.758	0.939	0.128	0.854	0.227	0.711	0.835	0.69	0.723	0.787	0.963	0.960
HPC	0.839	0.903	0.824	0.574	0.817	0.354	0.325	0.788	0.830	0.784	0.654	0.887	0.824
Thunderb.	0.882	0.941	0.663	0.813	0.649	0.694	0.576	0.599	0.943	0.919	0.844	0.955	0.646
Windows	0.697	0.690	0.567	0.990	0.588	0.689	0.701	0.713	0.566	0.993	0.989	0.997	0.406
Linux	0.297	0.673	0.672	0.519	0.279	0.169	0.701	0.629	0.701	0.612	0.605	0.690	0.284
Mac	0.558	0.764	0.673	0.369	0.599	0.478	0.595	0.604	0.698	0.872	0.757	0.787	0.636
Android	0.882	0.682	0.712	0.909	0.616	0.548	0.585	0.798	0.880	0.504	0.919	0.911	0.788
HealthApp	0.331	0.568	0.822	0.592	0.549	0.235	0.397	0.531	0.174	0.684	0.639	0.780	0.440
Apache	0.731	1	1	1	1	0.582	1	0.709	1	1	1	1	1
OpenSSH	0.521	0.538	0.802	0.426	0.501	0.373	0.619	0.426	0.925	0.431	0.554	0.788	0.500
Proxifier	0.518	0.518	0.515	0.495	0.026	0.967	0.517	0.951	0.508	0.517	0.527	0.527	0.013
Average	0.637	0.754	0.777	0.563	0.652	0.482	0.669	0.665	0.721	0.694	0.751	0.865	0.605

Beside research projects, various industrial tools for parsing and analyzing log messages are available. Table 2.4 provides a list of 31 log management and analyzing tools which are capable of analyzing system logs. Other industrial platforms such as AlienVault, BLËSK, Bugfender, Chart.io, GoAccess, jKool, Knowi, Logary, Looker, ManageEngine, Octopussy, PagerDuty, Papertrail, Pentaho, Prometheus, Qlik, Retrace, Rocana, ScoutApp, Sentine, Sentry, Seq, Sisense, and Tableau are also able to perform basic syslog analysis with the help of additional plugins.

Identifying the variable terms within system logs is a common practice for log pattern extraction before performing log analysis. The goal is to extract of common patterns from similar log messages. Direct extraction of log patterns from software source code provides the highest accuracy [65]. However, in many cases the software source code is not available or accessible. Furthermore, the efficiency of direct pattern extraction from software source code is significantly low due to the inconsistency of log messages generated using various versions of a software. Therefore, log patterns must be re-extracted after each software

[1]https://github.com/logpai/logparser

Table 2.4: Names and description of 31 log management and analyzing tool

Tool Name	**Description by the producer**
Alert Logic [84]	Collect, aggregate, and search log data
Cloudlytics [85]	Orchestration for log analysis and monitoring
EventSentry [86]	Log Monitoring and beyond
EventTracker [87]	Monitor, search, alert and report on any log or any format
Fluentd [88]	Data collector for unified logging layer
Apache Flume [89]	Efficiently collecting, aggregating and moving large amounts of log data
Graylog [90]	Industry leading log management
InTrust [91]	Smart and scalable event log management
IPSwitch [92]	An automated tool that collects, stores, archives and backs-up Syslog
lnav [93]	The Log File Navigator
LOGalyze [94]	The best way to collect, analyze, report and alert log data
LogDNA [95]	Instantly centralize, monitor, and analyze logs in real-time
Logentries [96]	The Fastest Way to Analyze Your Log Data
Loggly [97]	Perform log analysis on text based logs
Logmatic [98]	Log Centralization, Analytics and Visualization
LogRhythm [99]	Log Management and Log Analysis
Logsign [100]	Real-Time Analysis
Logstash [101]	Collect, Parse, Transform Logs
LOGStorm [102]	Complete log management with powerful correlation technology
Logsurfer [103]	Monitoring system logs in real-time
Logz.io [104]	AI-Powered Log Analysis
Loom [105]	Predict and Prevent IT incidents
Motadata [106]	Find Actionable Context in Log Data
Nagios [107]	Centralized Log Management, Monitoring and Analysis Software
NXLog [108]	Log manarement solutions for everyone
OVIS (Baler) [109, 110]	HPC data collection, transport, storage, analysis, visualization, and response
Scalyr [111]	Log management and visibility for modern applications
Splunk [112]	Predict and prevent with an AI-powered monitoring and analytics solution
Sumo Logic [113]	Proactive and predictive
Swatch [114]	Monitoring events on a large number of servers andworkstations
XpoLog [115]	Fully Automated Log Management

update. Alternative approach is the extraction of log patterns through reverse analyzing of collected log messages. This method is used in this work. More details are provided in Section 3.3.

A common approach proposed by several algorithms such as LogSig [74], IPLoM [73], LogCluster [69], LogMine [77], and MoLFI [80] is to convert the syslog entries into system events via multiple passes of clustering and categorization. In several cases, the timestamps of log entries are discarded as a side effect of such clustering methods, resulting in the elimination of temporal correlations among system logs [116, 65, 117]. Although few approaches preserve the temporal correlations of log entries [118], most of the existing approaches utilize static sets of rules to differentiate among various classes of system logs which negatively affects the accuracy of these approaches in processing of unseen and sophisticated log entries.

The main challenge in all available parsing methods is the analyzing of the unstructured part of syslog messages. Due to the unreliable nature of the syslog protocol, noises and inconsistencies also exist among the collected system log entries. Majority of the existing log parsers require multiple passes of processing which makes them inadequate for real-time monitoring and analysis of large HPC systems. Due to the unavailability of the original implementation of the existing algorithms, most features of the proposed algorithms cannot be evaluated. None of the existing tools are able to automatically analyze unstructured free-form log messages. User privacy remains a great concern since the industrial platforms provide the log analyzing process as a remote service, and the log analysis may take place on the third-party servers. Reliance on static rule sets restricts the flexibility of pattern detection. Due to the dynamic nature of HPC systems, manual adjustment of rule sets is also not preferred.

2.2 Users and Systems Privacy

Conducting any form of behavior analysis, for the purpose of failure detection and prediction, requires in-depth details about the actual state of the computing system. System logs readily contain such information. The usage of an HPC system is regulated by the privacy guidelines in force, according to its functionality, production environment, and administration domain. Depending on the applicable privacy regulations, certain information within the system logs may be considered as sensitive information. Examples include usernames and IP addresses. Information deemed sensitive on one HPC system can be considered not sensitive to another HPC system. Analyzing and distributing raw system logs, that may contain sensitive information, endangers the privacy of data subjects such as users, system owners, and system vendors. Therefore, data anonymization is required *before* the analysis and distribution of (raw) system logs. However, due to the uncertainties about the imperfection of the existing anonymization methods, the owners of HPC systems are reluctant to publish their monitoring data [119]. A list of publicly available HPC monitoring data[2] is shown in Chapter E.

The data protection and privacy guidelines of each computing system mandate the removal of certain sensitive information from system logs. Therefore, a certain amount of information loss cannot be prevented during the anonymization phase. After anonymization, the system logs may have already lost their usefulness for certain types of analyses. For example, the anonymized system logs of a computing system, with a privacy guideline that mandates complete removal of all usernames from system logs before any analysis, are not useful for user accounting purposes.

In March 2014, European Parliament approved the new privacy legislation. According to these regulations, *personal data* is defined as "any information relating to an identified or identifiable natural person (‘data subject.’)" [120]. This information must remain private to ensure a person’s privacy. Based on this definition, syslog entries contain numerous terms which represent personal data and must, therefore, be protected.

Encryption and de-identification are the most common approaches to protect personal data in log entries. Encryption reduces the risk of unauthorized access to personal data. However, encryption is reversible. Any form of encryption is theoretically breakable, pro-

[2]To the best knowledge of author, at the time of writing, these are the only existing large-scale publicly-available HPC monitoring data.

vided enough time and computational power. The encryption key having to be securely preserved yet also shared in order to make further analysis possible. Therefore, encryption can only be used within a trusted environment and the encrypted syslog entries cannot be freely used or distributed in the public domain. Therefore, log encryption is not a suitable approach for analyzing and distributing system logs. In contrast, de-identification eliminates the sensitive data and only preserves the nonsensitive (cleansed) data. As such, de-identification provides the possibility of distributing de-identified data in the public domain. However, due to the potential excessive information loss, the de-identified data may turn out to no longer be of real use.

Pseudonymization and anonymization are two different forms of de-identification. In pseudonymization, the sensitive terms are replaced by dummy values to minimize the risk of disclosure of the *data subject* identity. Nevertheless, with pseudonymization the *data subject* can potentially be re-identified using supplementary information [121]. Anonymization, in contrast, refers to protecting user privacy via irreversible de-identification of personal data.

Generalization and suppression are two well-known methods for data anonymization. These methods either group or remove data to reduce uniqueness, and thus, the chance of identification of individual data subjects from the records in the dataset. In 2002 k-anonymity [121] was introduced as a model for protecting privacy via generalization. Although k-anonymity addressed the main challenge of data privacy in anonymized datasets, it had several shortcomings such as attribute disclosure, and complementary data release. To overcome these shortcomings, several models such as l-diversity [122] and t-closeness [123] were introduced. These models reduced the data representation granularity (grouping) beyond the level used in k-anonymity, which could result in decreased data usefulness. The l-diversity models are also potentially vulnerable to algorithm-based attacks. Some studies considered an integration of both, the l-diversity and t-closeness models [124]. In 2006 the formal principle of differential privacy was introduced [125]. The differential privacy principle addresses the vulnerability to algorithm-based attacks and provides a strong privacy model. However, identifying a good strategy to implement differential privacy is difficult, it further decreases the data utility, imposes high overhead, and cannot be automated. According to the articles 2, 4(1) and (5) and recitals (14), (15), (26), (27), (29) and (30) of the GDPR[3], in order to analyze sensitive information, an irreversible anonymization of personal data must be guaranteed [126]. To the best knowledge of the author, no existing model can guarantee the data privacy and provide useful data.

Various tools have been developed to address the privacy concerns of using syslog information. Most of these tools provide log encryption as the main feature, while certain tools also provide de-identification as an additional feature. Syslog-ng and Rsyslog are two open-source centralized logging infrastructures that provide *out of the box* encryption and message secrecy for syslogs, as well as de-identification of syslog entries [127, 128]. Both tools provide a *pattern database* feature, which can identify and rewrite personal data based on pre-defined text patterns. Logstash [129] is another open-source and reliable

[3]The European General Data Protection Regulation.

tool to parse, unify, and interpret syslog entries. Logstash provides a text filtering engine which can search for the text patterns in live streams of syslog entries and replace them with predefined strings [130]. In addition to the off-line tools (local installation), such as syslog-ng and Logstash, there is a growing number of online tools (remote services) such as Loggy [131], Logsign [100], and Scalyr [111], that offer a comprehensive package of syslog analysis services. The existence of sensitive data in the system logs barricades the usage of such remote services.

Alongside these industrial-oriented tools, several research groups have developed scientific-oriented toolkits to address the syslog anonymization challenge. eCPC toolkit [132], sdcMicro [133], TIAMAT [134], ANON [135], UTD Anonymization Toolbox [136], and Cornell Anonymization Toolkit [137] are selected examples of such toolkits. These tools apply various forms of *k-anonymity* [121] and *l-diversity* [122] to ensure data anonymization. Achieving an optimal *k-anonymity* is an NP-hard problem [138]. Heuristic methods, such as k-Optimize, can provide effective results at the cost of a longer time [139].

The process of data anonymization incurs a certain degree of information loss. With significant information loss comes decreased usefulness of the anonymized data. Various studies attempted to address the problem of achieving k-anonymity protection with minimal information loss. Gionis and Tassa proved that solving the problem for the two conflicting goals above is NP-hard [140]. Later, it has been shown that dynamic optimization of the anonymization process considerably reduces the loss of information [141]. In another attempt to address the high information loss during data anonymization, *utility-based anonymization* methods were proposed. Xu et al. [142] introduced an approach which first, specifies the utility of each attribute, and second, proposes two heuristic *local recording*-based anonymization methods [143] to boost the quality of the analysis later. A *data relocation* mechanism has also been applied to reduce granularity and populate small groups of tuples to increase data usefulness [144]. In another similar effort, quasi-identifiers have been divided into two groups of ordered and unordered attributes [145]. To reduce the information loss, more flexible strategies for data generalization have been applied on the unordered attributes. More recently, *co-utility* [146] has been introduced as a global distributed mechanism for data anonymization, such that a balance between data utility and data privacy is achieved. Although these efforts decrease the information loss during data anonymization, they still cannot guarantee data privacy. The non-zero probability of privacy breaches through anonymization by the use of approaches such as those mentioned in this section has been experimentally determined [147].

Quantifying data utility[4], which is a qualitative property of data, provides a measure to control the balance between privacy and utility of data. A number of studies proposed such measures to quantify the utility of anonymized data. Data utility is mostly described as the amount of information loss. Information loss, in general, can be quantified according to the uncertain change in attribute values during the anonymization [145], via result-driven approaches to compare the data before and after anonymization [148], or even according to the data entropy in the dataset [149]. These measures are generally divided into two cat-

[4] *Utility* and *usefulness* are used interchangeably in this work.

egories: (1) *entropy-based* and (2) *distance-based* (e.g., the Hellinger distance). Furthermore, most of the above mentioned usefulness quantification approaches are implemented into data anonymization tools, such as ARX [150].

Existing approaches for quantification of data usefulness aim to increase data privacy, data utility, or both, in anonymized datasets. However, these approaches implicitly make the fundamental assumption of having a structured format for the data entries. In reality, system log entries are of mixed *structured* and *unstructured* data formats. The structured part contains the meta-data (e.g., time or location related to the particular syslog entry), and the unstructured part contains the detailed event information. Sensitive information mainly resides within the unstructured part of the data. In the author's best knowledge, due to the unstructured nature of the detailed event information (no two distinct events generate the same information pattern), none of the existing approaches provide utility-based anonymization of system logs. Moreover, although a few studies addressed the de-identification of unstructured datasets [151], none of these studies, nor the known privacy models guarantee data privacy at an acceptable overhead (time and complexity).

Using the existing anonymization approaches, in general: (1) The quality of the anonymized data dramatically degrades, and (2) The size of the anonymized syslogs remains almost unchanged. The industrial-oriented approaches are unable to attain full anonymization at micro-data level [121]. Even though scientific-oriented approaches can guarantee a high level of anonymization, they are mainly not capable of applying effective anonymization in an online manner. Certain scientific-oriented methods, which can effectively anonymize online streams of syslogs, need to manipulate log entries at their origin [152, 153].

The data application is not considered in existing methods of data usefulness assessment. Although data may lose its usefulness for certain analysis, it may remain useful for other applications. Therefore, it is important to assess the data usefulness in respect to the data application. While the anonymization methods try to uniform the data variance in order to protect users privacy, anomaly detection techniques on the other hand require those differences in input data to detect anomalies. Thus, *useful data* from this perspective must project a balance between variation and uniformity.

2.3 Failure Detection and Prediction

When defects within the system's components (fault) are triggered (error), they might prevent system's components to perform their expected functionalities, thus cause failure. Although failures are run-time events, failure prediction can be performed using both on-line and offline approaches. Offline failure prediction approaches employ the knowledge gained during previous executions to predict the probability of future failure occurrences. Due to the static nature of offline prediction approaches and their reliance on historic data, dynamic behaviors cannot be predicted using offline approaches. Online failure prediction approaches consider run-time information beside the historic executions' knowledge to predict the probability of future failure occurrences [154]. Given the dynamic nature of general-purpose HPC systems, online approaches are more suitable for failure prediction.

During the past decade, multiple studies investigated the existing failure detection and prediction methods in the context of HPC [155, 156, 157, 158]. This sub-section, along with an overview of existing literature, fills the gap via investigating failure prediction methods that are based on log processing and are not covered in the existing studies.

2.3.1 Failure Correlation

Studies have shown different forms of correlations among failures in large computing systems [159, 68, 160]. In most cases, these correlations proved to be beneficial for syslog analysis. It is expected to observe similarities among the footprints of correlated failures in system logs. These similarities are captured and utilized to detect correlated failures. Two groups of correlations exist: static (i.e., permanent) and dynamic (i.e., temporary) [161, 162, 163, 164].

Majority of correlations are static correlations that are part of the system characteristics and will remain unchanged during the lifetime of the computing system. Simultaneous failure of multiple computing nodes located in a single rack caused by malfunctioning of the rack's cooling system is an example of static (spatial) correlation among failures. Dynamic correlations are mainly appearing due to the dynamic assignment of shared resources and user interactions. The simultaneous failure of multiple computing nodes that are accessing a certain file on an unstable distributed file system is an example of a dynamic failure correlation.

2.3.2 Anomaly Detection

Anomaly detection is the main building block of failure detection mechanisms. The goal of anomaly detection is detecting irregularities in normal behavior. To achieve this goal, (1) the *normal* behavior and (2) the acceptable deviation threshold from this norm should be defined. Any deviation more than the acceptable threshold is considered *abnormal* behavior. Since the behavior of computing systems is constantly changing in response to their users and environment, static models are not sufficient to describe the dynamic behavior of modern HPC systems. Therefore, the model that describes the normal system's behavior should be constantly updated according to the new behavior of the computing system.

Various methods are proposed to extract the behavioral patterns of computing systems. Estimating probability of upcoming failures based on system's load [34, 42], calculating failure frequency via word counting, using time series analysis, generating hidden Markov models [165], anomaly detection via predefined rule-sets [103], automatic pattern mining [166], various forms of clustering [167], and decision trees [168] are examples of major anomaly detection methods. Furthermore, several studies suggested to form blocks of correlated syslog entries using the semantic correlation among system logs before further analysis [169, 116]. Directed acyclic graphs (DAG) have been also used in anomaly detection methods to preserve the correlation of log entries [170].

However, system logs are not the only monitoring data which are used for detecting anomalies in computing systems. CASPER [171] monitors the network activities, TIRESIAS [172] observes CPU, memory, and context switches, SEAD [173] monitors the hypervisor, and ALERT [174] collects various metrics including CPU load, memory usage, input/output data rate and buffer queue length (more information in Table 2.7). This section briefly introduces the 3 most used methods of anomaly detection that are based on system log analysis.

Time Series Analysis

System logs by default are discrete series of timestamped events. Therefore, they can be best described using time series. Time series analysis are mainly focused on detecting auto-correlations, trends or seasonal variations among the input data.

The first steps in time series analysis is data discretization and assignment of an alphabet according to the potential input values. Assuming that the number of syslog entries received during a 5-minute time window is a value between 0 and 1000 depending on the computing system's condition, the alphabet Σ can be defined as $\Sigma = a, b, c, d$ such that $a = \{0 : 0\}$, $b = \{1 : 10\}$, $c = \{11 : 100\}$ and $d = \{101 : 1000\}$. According to the newly defined alphabet, the set of syslog records $R = \{10, 34, 0, 512, 23, 12\}$ which is collected during a 30-minute interval (5-minute time window) can be written as $R = \{b, c, a, d, c, c\}$, or simply $R = "bcadcc"$.

Using time series, 3 types of periodicity can be defined: (a) symbol periodicity: when one symbol repeats periodically e.g., a in $"\underline{a}bc\underline{a}ee\underline{a}be\underline{a}bd\underline{a}ce\underline{a}"$, (b) partial periodicity: when a pattern consists of more than one symbol repeats periodically e.g., abc in $"\underline{abc}aeed\underline{abc}beab\underline{abc}d"$, and (3) full periodicity: when the time series is mostly represented by a periodic pattern e.g., $abdc$ in $"\underline{abdc}\overline{abdc}\underline{abdc}"$.

The definition of alphabet has a significant impact on the results of time series analysis. Assigning a symbol to a range of values eliminates potential noises in data, and improves the consistency of results. However, the semantic of values must be considered in choosing the value ranges. An uninformed assignment may hide anomalies via an unintentional flattening of the outliers.

Suffix trees and suffix arrays are powerful data structures for analyzing time series [175, 176]. Suffix trees can be generated in linear time [177]. A bottom-up traverse of suffix tree results in generation of the *occurrence vector*. Occurrence vectors are common data structures to store the position of repetitive patterns in time series. The sorted list of all suffixes of a string, forms the suffix array of that string. Suffix arrays proved to be extremely efficient data structures for anomaly detection in very large time series [178].

Pattern and Rule Mining

Pattern mining is a powerful tool to discover sequential and periodic recurring patterns in various sequences (e.g., time series). In Table 2.5a, 5 sequences of events have been shown. The sequence of $< b, c, g >$ is a recurring sequential pattern with support value of

4, since it has been repeated in 4 sequences. Significant changes of the recurring patterns' support value during fixed-size time windows can be an indication of abnormal behavior.

Main drawback of this approach is the lack of probability assessment. Therefore, it is not known how plausible is the re-occurrence of an upcoming event. For example in Table 2.5a, it is indicated that the <b,c,g> sequence is recurring with a support value of 4. However, it is not known that this sequence has a re-occurrence probability of 80% (4 out of 5 sequences; the order of appearance is not considered).

(a) Pattern mining

Name	Sequence
Sq1	<a,**b**,**c**,d,e,e,**g**,h>
Sq2	<f,**b**,e,g,**c**,f,**g**,b>
Sq3	<g,b,f,d,h,g,b,a>
Sq4	<**b**,**c**,a,a,d,f,**g**,a>
Sq5	<**b**,a,**c**,e,f,h,c,**g**>

(b) Rule mining

Name	Sequence
Sq1	<a,d,c,d,e,e,g,h>
Sq2	<f,**b**,e,g,c,**f**,g,**a**>
Sq3	<g,**b**,**f**,d,h,g,b,**a**>
Sq4	<**b**,c,a,**a**,d,**f**,g,a>
Sq5	<a,b,c,e,f,h,c,g>

Table 2.5: Examples of pattern and rule mining in sequences

Various algorithms such as PrefixSpan (2004), LAPIN (2005), CM-SPADE (2014), VMSP (2014), CM-SPAM, FCloSM, FGenSM, Spade, SPAM and GSP have been introduced for pattern mining. However, the lack of probability assessment is one of the main limitations to use these algorithms in failure prediction.

To overcome this limitation, the *rule mining* has been introduced. A (sequential) *rule* is defined as $A => B$, where both A and B are itemsets. The $A => B$ rule is interpreted as if items in itemset A occur, then they will be followed by the items in itemset B. The items within A and B can occur in any order, but it is required that the items of B occur only after items of A. Considering Table 2.5b as sets of sequences and $\{b\} => \{a, f\}$ as the sequential rule, it can be concluded that the support of sequential rule $\{b\} => \{a, f\}$ is 3. Since, in 3 sequences the events a and f are occurring after event b.

To calculate the probability of a sequential rule, another measure called confidence will be considered. Confidence of the rule $A => B$ is the support of the rule divided by the number of sequences containing the items of A. In the given example, since in 4 out of 5 sequences $\{b\}$ occurs but only in 3 of them it is followed by $\{a, f\}$ therefore, the confidence of rule $\{b\} => \{a, f\}$ is 75%.

Various algorithms such as CMRules (2010), TopSeqRules (2011), RuleGrowth (2011), TNS (2013), ERMiner (2014), TRuleGrowth, PFP-Tree, HUSRM, MKTPP, ITL-Tree, PF-tree and Max-CPF are proposed for sequential and periodic rule mining. Studies show that rule mining provides higher detection accuracy than pattern mining [179].

Machine Learning

From another perspective, anomaly detection methods are divided into three main categories of rule-based, supervised, and unsupervised [180]. Although many rule-based methods have been proposed, the unstructured nature of syslog messages extremely limits the functionalities and the detection domain of rule-based approaches. Supervised methods on the other hand require both normal and abnormal patterns to train a functional behavioral classifier. Therefore, rule-based approaches as well as supervised approaches are not able to detect anomalies which are not seen before. Furthermore, extraction of rules and patterns for rule-based and supervised approaches are time-consuming and inaccurate, respectively [181].

In contrast, unsupervised approaches are able to automatically extract the system's behavioral pattern from the monitoring data. However, most of the unsupervised approaches and available tools are domain specific. They are built specifically for a certain class of problems e.g., security threads detection [182], DNS poisoning attacks identification [183], or performance bottleneck detection [184]. Furthermore, general approaches such as invariant log mining [185] and principal component analysis (PCA) [186] only consider the chronological order of the events, discarding the temporal correlation among log entries.

Unsupervised syslog-based anomaly detection methods were further improved via recent advances in machine learning techniques [118, 166, 187, 188]. Despite the rapid improvements in performance and accuracy of unsupervised anomaly detection approaches using machine learning, certain challenges remained unsolved. As the volume of generated system logs on HPC systems is rapidly increasing, the storage and processing of syslog entries becomes challenging. Due to the diversity of HPC systems characteristics, in contrast to many use cases of machine learning, reuse of pre-trained models for anomaly detection in HPC systems is not an effective alternative. Processing system logs that contain various personal data, raises serious concerns regarding users privacy. Due to the components' heterogeneity in modern HPC systems, each component projects a different behavior which cannot be accurately modeled via a single system-wide general model. Software and hardware updates, various applications and the multi-user environment of HPC systems, continuously change the system's behavior. Therefore, a static behavioral model of the HPC system is not sufficient to accurately model the dynamic behavior of modern HPC systems. In addition, system logs are generated by individual computing nodes, thus, any failure directly affects syslog entries via introducing noises, interrupting log generation mechanisms, or impeding the log collection procedure. Furthermore, harmless errors that are not causing failures may also introduce noises in syslog entries.

2.3.3 Prediction Methods

Salfner et al. [155] divide the procedure of failure prediction into 5 steps of (1) testing, (2) auditing, (3) monitoring, (4) reporting and (5) tracking. Figure 2.2 shows 3 main steps that are performed by most online failure prediction mechanisms. In a recent study, Jauk et al. classified more than 30 existing prediction methods. Table 2.6 shows the two-dimensional classification of prediction methods as originally proposed in [158].

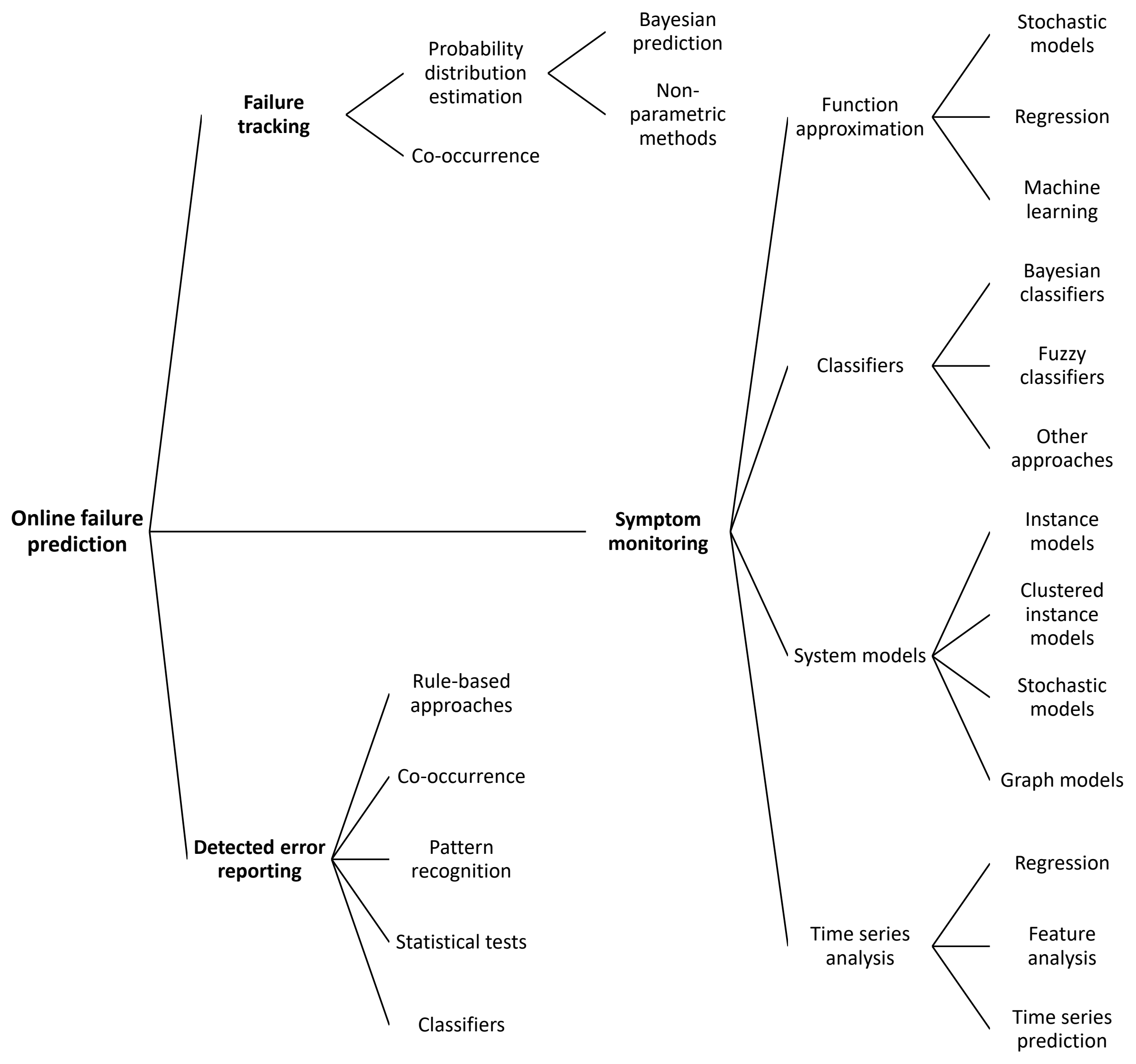


Figure 2.2: A taxonomy for online failure prediction approaches [155]

Failure tracking

Failure tracking uses the recorded failures of the past to predict potential future failures. The prediction is made either via estimating the probability distribution of failures, or via analyzing the correlation between failures.

Symptom monitoring

Symptoms are side effects of failures which can be directly or indirectly related to the cause of failure. Sudden increase of network traffic or higher CPU consumption are direct and indirect side effects of network driver failure, respectively. Constant monitoring of the computing system parameters can reveal the symptoms and help to identify the causing failures. Common methods of symptom detection are: (1) function approximation that compares the actual and the expected output of unknown functions fed with system

Table 2.6: A classification of literature on failure prediction in HPC [158]

Class		SW/S	Node	Disk	Mem.	Net.	Log
Root Cause		unspecified		pinpoint-able			
Correlation Probability	[189]						🟦 - / 82
	[190]	🟦 - / -	🟦				
	[191]	🟦 93 / 43	🟦			🟦	
	[192]				🟦 - / -		
	[83]						🟦 78 / 75 69 / 58 88 / 46
	[193]	🟦 91.2 / 45.8	🟦	🟦	🟦	🟦	🟦
	[194]	🟦 88 / 75 77 / 69 81 / 85	🟦				
Rule	[195]		🟦 - / -				
	[196]						🟦 80 / 90
	[197]						🟦 58 / 74
	[198]	🟦 - / -					
	[199]		🟦 98 / 93				
	[200]			🟦 - / -			
Mathematical, Analytical	[201]						🟦 40 / 80
	[202]						🟦 - / -
Decision Trees / Forest	[203]			🟦 - / -			
	[204]			🟦 * / *			
	[205]		🟦 74 / 81				
	[206]		🟦 98 / 91				
	[207]	🟦 94.2 / 85.9					
	[208]	🟦 79.5 / 50 95 / 94					
	[209]	🟦 53 / -	🟦				
	[210]	🟦 - / -	🟦				
	[211]		🟦 72 / 87				
Regression	[212]	🟦 - / -	🟦				
Classification	[213]						🟦 55* / 80*
	[214]		🟦 66.4 / 59.3		🟦	🟦	
	[215]			🟦 - / -			
	[216]	🟦 * / *	🟦 * / *				
	[202]						🟦 - / -
Bayesian Network, Markov	[217]		🟦 93 / 91				
	[218]						🟦 82.3 / 85.4 64.8 / 65.2
Neural Network	[215]			🟦 - / -			
	[219]	🟦 - / -					
	[206]	🟦 89 / - 80 / -					
Meta-Learning	[220]						🟦 90* / 70*
	[221]						🟦 - / -

(with precision/recall values in %)
Blue shows ability to predict this type of failure.
A : Results for different data sets.
B : Results for different training parameters.
C : Results varies greatly with different parameters.
D : Paper lists several methods or setups.
* : Many results provided, see reference.
- : No numerical result is given.

parameters, (2) classifiers that identify the value of system parameters as normal or abnormal based on predefined thresholds, (3) system models that build a model according to the normal state of the system and search for significant deviations from this model and (4) time series analysis that detects deviations within the chronological sequence of system parameters.

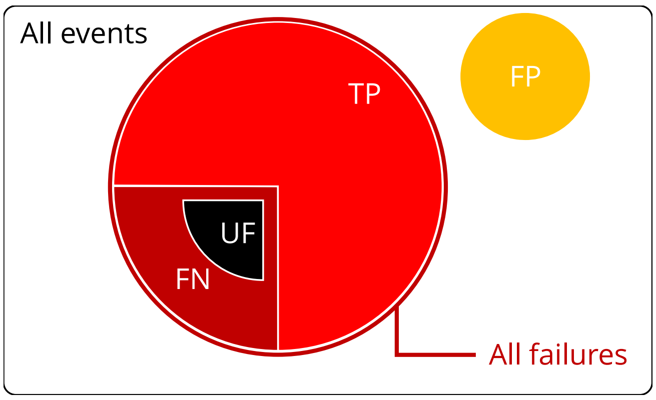
Detected Error reporting

In contrast to symptom monitoring, that constantly monitors certain system parameters to detect potential deviations in their values, this approach only analyzes the detected and reported errors. Undetectable errors are hidden to *detected error reporting* as an online event-driven approach, thus, certain failures are out of the scope of this prediction method. The reported errors can be compared against a set of predefined rules or patterns (rule-based systems and pattern recognition), searched for detecting correlations between reported errors (co-occurrence), statistically compared to previous records (statistical tests) or labeled according to their importance and impact (classifiers).

2.3.4 Prediction Accuracy and Lead Time

Among the available prediction methods, regardless of the chosen failure prediction approach, the final reported assessments are either (1) the location and time of potential future failures (i.e., when a certain node is expected to fail) or (2) continuous reporting of the stability status of computing nodes (i.e., no conclusion is provided). Both formats may include a certainty value that indicates the confidence of assessment. Table 2.7 provides a list of recent failure prediction methods and their respective accuracy and recall values. Each method is evaluated using a different set of input data.

The $Precision$, $Recall$, $Accuracy$, and F_1Score are calculated as shown in Figure 2.3. True positives (TP) indicate failures that are correctly identified. Higher precision value in Table 2.7 indicates less false positives (FP), that represent normal events that mistakenly identified as failure. Higher recall value indicates less false negatives (FN), that represent failures that remained unidentified[5].



$$Precision = \frac{TP}{TP+FP}$$
$$Recall = \frac{TP}{TP+FN}$$
$$Accuracy = \frac{(TP+TN)}{(TP+FP+FN+TN)}$$
$$F_1Score = 2\frac{Precision.Recall}{Precision+Recall}$$

Figure 2.3: Calculation of precision and recall values. TP, FP, FN, and TN stand for true positive, false positive, false negative, and true negative respectively. UF represents the unpredictable failures explained in Section 5.4. Unpredictable failures are subset of false negatives.

[5]Some studies suggest additional metrics to evaluate the usefulness of prediction methods [15]. However, in the context of this work, precision, recall, and the F_1Score are the most relevant metrics to describe the usefulness of prediction methods.

Table 2.7: Failure prediction methods

Year	Name	Method	Source of Monitoring Data	**Based on Syslog**	**Precision**	**Recall**	**F1Score**	**Requires Additional Data**	**Automaticity**	**Privacy**
2010	Zheng et al. [201]	Genetic Algorithm	IBM Blue Gene/P	No	10-40%	60-80%	17% - 53%	RAS, Job	No	Partial
2011	ELSA/HELO [222]	Signal Processing, Offline/Online Clustering	IBM Blue Gene/L, Mecury, PNNL, Cray XT4, LANL	Yes	90%	50%	64%	Event, Outages	No	Partial
2011	Nakka et al. [205]	Decision Tree	LANL System 23	No	73%	80%	76%	Failure, Usage	No	Partial
2012	LogMaster [83]	Apriori-LIS, Event Correlation Graphs	IBM Blue Gene/L, Hadoop	Yes	79%-84%	38-70%	51% - 76%	Event	No	Partial
2014	Gainaru et al. [223]	Signal Processing, Rule Based	Blue Waters*	Yes	85%	60%	70%	Moab, Storage	No	No
2016	CloudSeer [67]	Workflow Graph, DAG, Automaton	5 node test cluster*	No	84%$^+$	90%$^+$	86%$^+$	OpenStack Events	No	No
2016	He et al. [157]	PCA, Logistic Regression, Decision Tree, and SVM	Amazon EC2*, BlueGene/L	Yes	50%-98%	61%-67%	54% - 79%	HDFS	No	No
2017	DeepLog [118]	RNN (LSTM)	Cloud-based VM, HDFS, OpenStack, Blue Gene/L	Yes	96%$^+$	96%$^+$	96%$^+$	Event	No	No
2017	Hora [224]	Bayesian Networks, Architectural Knowledge	Netflix*	No	41.9%	83.3%	55%	System metrics, Event, Fault injection	No	No
2017	Islam et al. [159]	RNN (LSTM)	Google cluster	No	85%	89%	86%	Workload	No	Partial
2017	Klinkenberg et al. [206]	Descriptive Statistics, Supervised ML	HPC at RWTH*	No	98%	91%	94%	System metrics, Power, Fan, Temperatures, Network	No	No
2018	Desh [225]	RNN (LSTM)	Cray XC30, XE6, XC40, XC40/XC30	Yes	83%	85%	83%	-	No	No
2018	Zasadzinski [226]	Decision Trees, CNN	DKRZ Mistral*	No	85%	60%	70%	Job, Power, Topology, Hardware	Yes	No
2018	TBP [199]	Topic Modeling, Time-Based Phrase	Cray systems	Yes	98%	83%	89%	Job, Console, Power, Topology, Hardware, Network	Yes	No
2018	Kimura et al. [167]	Log Clustering, Text Processing	A large production network	Yes	68-75%	71-81%	69% - 77%	-	No	No
2018	Landauer et al. [227]	Log Clustering	Web Server, SQL, Reverse proxy*	No	61.8%	99.3%	76%	Semi-synthetically, Fault injection	Yes	No
2019	Borghesi et al. [228]	Auto-encoders	D.A.V.I.D.E. HPC cluster	No	88%-96%	N/A	N/A	166 metrics aggregated each 5-minute	Yes	No
2019	Frank et al. [15]	DNN	Mogon I	No	99.4%	73%	84%	Hardware Sensors (fan RPM and temperature) and Software info	No	No
2020	Jam-e-Jam	Vicinity-based Model, HTM	Taurus II	Yes	85%	80%	82%	-	Yes	Guaranteed

* Data is not publicly available.
$^+$ Only achievable on structured logs.
RNN: Recurrent Neural Network
CNN: Convolutional Neural Network
DNN: Deep Neural Network
LSTM: Long Short-Term Memory
HTM: Hierarchical Temporal Memory
PCA: Principal Component Analysis
DAG: Directed Acyclic Graph

Since each method is evaluated using a different set of input data, it is not possible to directly compare the performance of listed methods solely based on their precision and recall values. For example, DeepLog reports a surprisingly high precision and recall rate of 96% in comparison to Hora with the precision rate of 42%. The main distinction lies in the datasets that were used to evaluate each of these methods. DeepLog uses HDFS, OpenStack and Blue Gene/L logs while Hora analyzes Netflix's server logs. HDFS dataset contains 11.1 million log messages collected from Amazon EC2 platform, manually labeled by the original domain experts [65]. BGL data contains 4.7 million log messages recorded by the BlueGene/L supercomputer system at Lawrence Livermore National Labs (LLNL), manually labeled by the original domain experts [43]. The HDFS and OpenStack datasets that were used to evaluate DeepLog contain semi-structured log entries. The diversity of log entries is very limited and log entries are directly related to the reported events. On the other hand Netflix's server logs are more diverse, less structured and the log entries do not directly relate to the reported events, which negatively influence the failure prediction accuracy. Looking at the lower precision of DeepLog in predicting Blue Gene/L failures[6] (less structured) reveals the importance of data preparation for failure analysis.

Among the listed methods in Table 2.7 Desh's input data is the most similar to the data used in this work, unstructured and low level. Although there are similarities between the prediction methods used in this work and Desh, the main difference is the use of a fully unsupervised approach and utilizing fully anonymized monitoring data in this work. The data labeling step (phrase labeling), greatly contributes to the high precision and recall rates of Desh's predictions.

Useful failure predictors must predict failures as early as possible to provide enough time for protective measures to take place. The current state-of-the-art lead time is below 10 minutes [225, 15] (although with the prediction precision of 42%). Recent recovery techniques require about 3 minutes lead-time to perform a complete cycle of checkpointing [229].

In summary, detection and analysis of dynamic correlations are more complicated due to constant changes in their characteristics. However, due to their significant influence, it is required to consider the impact of both static and dynamic correlations in failure analysis. The syslogs of real-world production HPC systems contain unstructured, noisy, and erroneous entries. Therefore, a failure predictor should work in the existence of all these imperfections. Modern HPC systems are dynamic and multi-purpose, therefore, any prediction model must adapt itself to the dynamic nature of such environments. The large number of syslog entries generated by modern HPC systems as well as the dynamic nature of large computing systems, demand automatic analysis approaches. Therefore, supervised approaches are not good candidates as long-term solutions.

[6] Originally 16%. With feedback from experts (online training) precision increased to 88%.

3 Data Collection and Preparation

Chapter 3 describes the methodology, workflow, and tools which were used and developed in this work to collect and prepare the monitoring data for the next steps. Figure 3.1 illustrates the major building blocks of the proposed approach as well as the workflow of the analysis. To maintain the readability of Figure 3.1, overlapping blocks and flows are not shown. The colored building blocks are covered in this chapter and the white blocks are covered in Chapter 4.

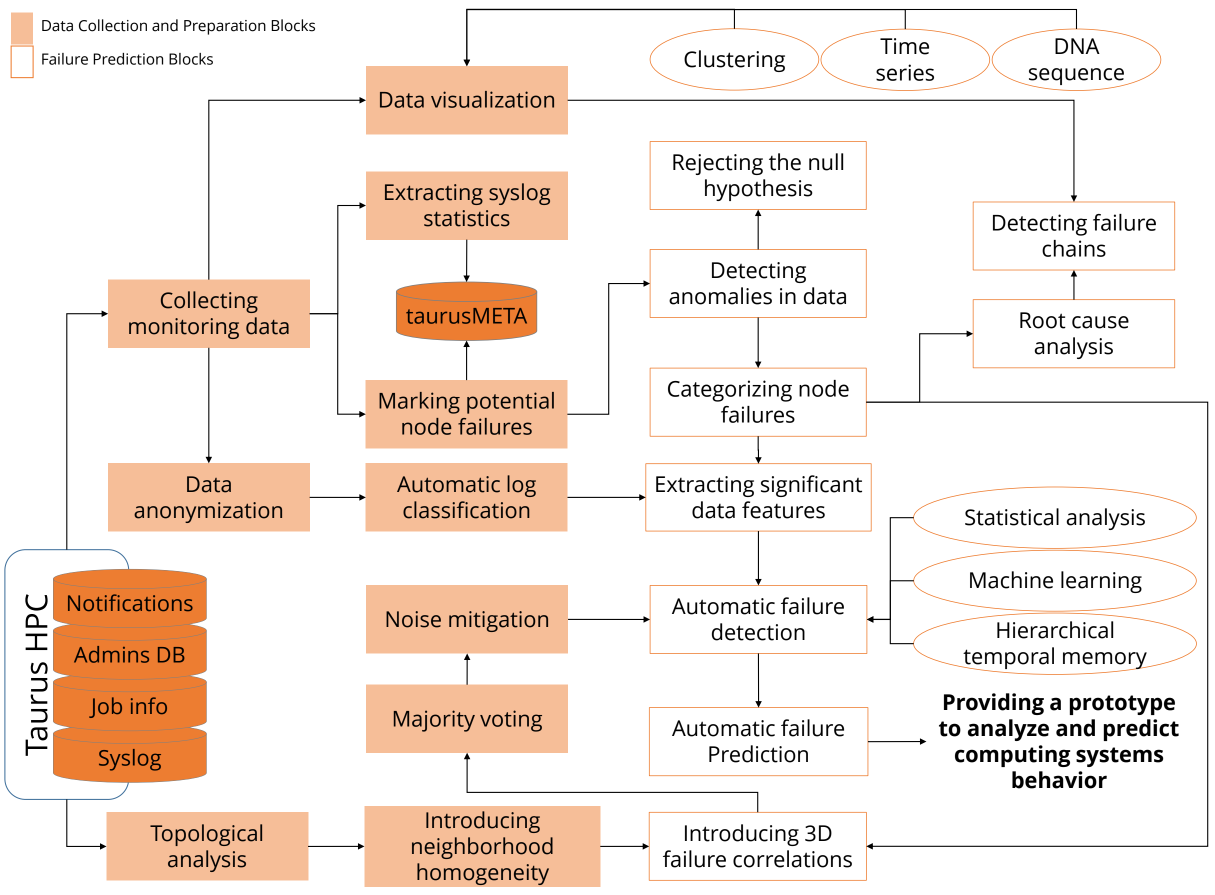

Data Collection and Preparation Blocks
Failure Prediction Blocks
Clustering
Time series
DNA sequence
Data visualization
Rejecting the null hypothesis
Extracting syslog statistics
Detecting failure chains
Detecting anomalies in data
taurusMETA
Collecting monitoring data
Root cause analysis
Marking potential node failures
Categorizing node failures
Data anonymization
Automatic log classification
Extracting significant data features
Statistical analysis
Machine learning
Hierarchical temporal memory
Taurus HPC
Notifications
Admins DB
Job info
Syslog
Noise mitigation
Automatic failure detection
Majority voting
Automatic failure Prediction
Providing a prototype to analyze and predict computing systems behavior
Topological analysis
Introducing neighborhood homogeneity
Introducing 3D failure correlations

Figure 3.1: Major building blocks and the workflow of *Jam–e Jam*. Building blocks shown in orange are covered in Chapter 3.

This work addresses the system failure, a main challenge of production HPC systems. Due to the nature of HPC systems, as well as their special production environment and conditions, certain assumption and limitation must be considered. Below is a short list of the most important assumptions and limitations which are considered in this work:

- Taurus is used as the main use case in this work (Section 3.1).
- Fault injection tools can be used to evaluate the resilience of HPC systems [119]. However, fault injection tests are intrusive and may interrupt the systems progress [230, 231]. Taurus is a production computing system with a large group of active users. Therefore, this work refrains to use intrusive approaches such as fault injection.
- To propose an approach that is generally applicable to other HPC system, no changes should be required to be made on the target systems (Section 3.2.1).
- The failure analysis will be conducted at the node-level (Section 3.4).
- To avoid overheads and prevent unwanted side effects on the target system, *active* monitoring techniques will be avoided. This limitation becomes vital during nodes instability conditions (Section 3.2.1).
- The source of monitoring data should be available on the majority of HPC systems (Section 3.2).
- There are only a few publicly available sources of monitoring data (Chapter E). User privacy is the main challenge to publish such monitoring data. Therefore, data anonymization is required before further analysis (Section 3.3.1).
- The collected monitoring data may have extensive amount of noises. Therefore, data cleaning is required before further analysis (Section 3.3.4).
- Due to the large size of HPC systems and high volume of monitoring data, a high degree of automation is required (Section 3.3.3).
- The exact time and count of node failures on Taurus is unknown. Therefore, data preprocessing for marking the potential failures is required before analysis (Section 3.4).

3.1 Taurus HPC Cluster

Taurus is a production high performance computing system located in Dresden, Germany. Taurus (at the time of writing) consists of 2046 computing nodes located in 6 islands. Taurus monitoring data is the main use case of this work. This section provides detailed information about Taurus' hardware and software specifications. Additional details, live system status, list of services and future upgrades are accessible via Taurus information page[1].

The 2046 computing nodes on Taurus consist of four different processor architectures: Intel Haswell, Broadwell, Sandy Bridge, and Westmere. The 108 nodes with Sandy Bridge and Haswell processors are also equipped with NVIDIA Tesla GPUs, out of them 44 nodes are each equipped with two NVIDIA Tesla K20x, and another 64 nodes are powered by each four NVIDIA Tesla K80. 32 additional nodes are equipped with Intel Xeon Phi manycore processors[2] and 14 servers are also equipped with Intel Xeon E5-2603 CPU and NVIDIA GTX1080 GPU cards.

[1]https://doc.zih.tu-dresden.de/hpc-wiki/bin/view/Compendium/SystemTaurus
[2]https://doc.zih.tu-dresden.de/hpc-wiki/bin/view/Compendium/KnlNodes

This work uses the data collected from 2046 regular computing nodes as the main use case. Considering the hardware architecture, Taurus nodes are divided into five categories. The number of nodes in each category and the node's dominant processor architecture are shown in Table 3.1.

Table 3.1: Hardware architecture of Taurus computing nodes

Architecture	Haswell	Sandy Bridge	Westmere	Broadwell	GPU (K20X/k80)
Nodes count	1456	270	180	32	108
Island	4, 5, 6	1	3	4	2

Figure 3.2 provides a schematic illustration of Taurus islands. Each O represents a single computing node. Identical colors represent identical hardware architectures (processing units).

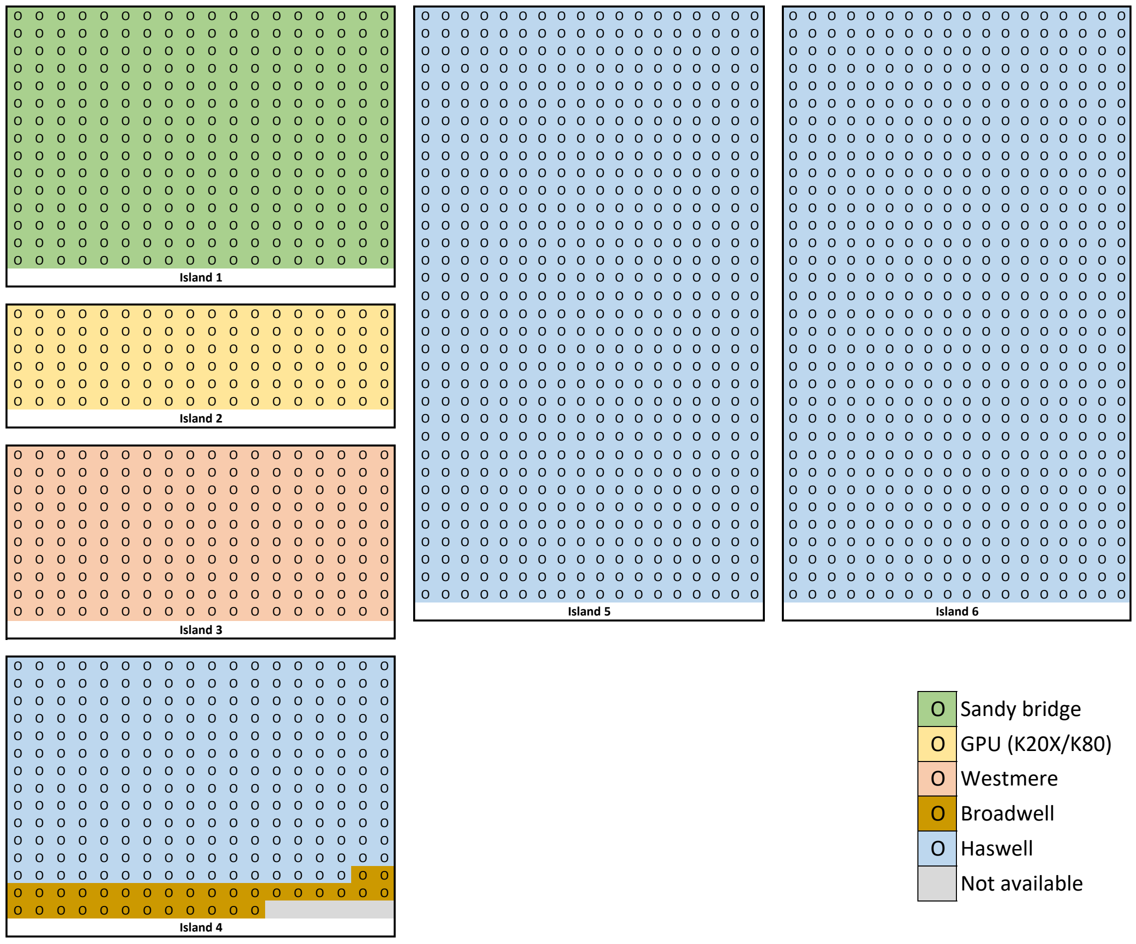


Figure 3.2: Schematic island topology of Taurus. Node colors represent the dominant processing unit type of a node. Thick border lines indicate the 6 islands of Taurus.

All nodes in Island 2 (108 nodes) beside their Sandy bridge or Haswell CPUs are equipped with graphical processing units (GPU). Since the majority of jobs submitted to Island 2 mainly utilize GPUs rather than CPUs, GPUs are considered as dominant processing units of these nodes. Therefore, in this work, Island 2 is considered as a homogeneous GPU island, despite other heterogeneity of its nodes. Taurus is powered by Linux and employs Slurm [232] as its job scheduler. The parallel filesystem of Taurus is powered by Lustre [233]. The syslog-ng daemons are running on all nodes.

During the first stages of the analysis, three important phenomena were observed, namely (1) failure propagation, (2) failure chains and (3) side effects of protection mechanism. Each of these phenomena highly contribute in providing a better understanding of the system's behavior. Thus, they play an important role in detection and prediction of upcoming failures.

Failure propagation

Analyzing Taurus behavior revealed that certain failures are propagating through the system. This propagation may happen within local components of a single node, or may affect other nodes and remote components. Failures originate in different layers, and they may be of different types. Failures always propagate horizontally within a single layer, as well as from bottom to top across the system layers. For example, as shown in Figure 3.3 those failures which occur at the hardware layer will have influence on the application layer.

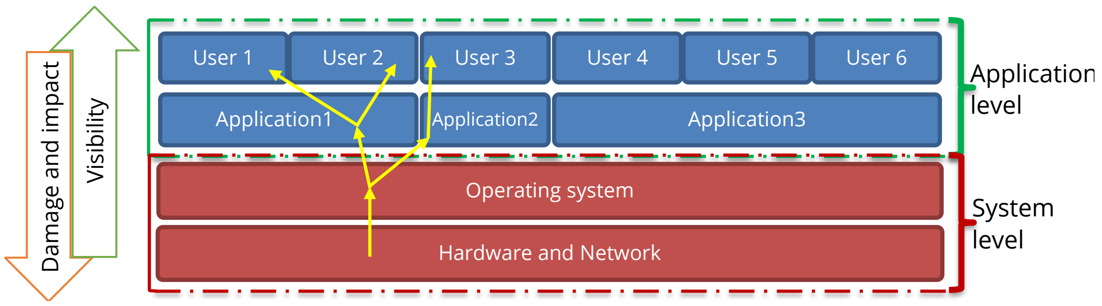


Figure 3.3: Propagation of failures within a computing node. The yellow arrows indicate a potential propagation of failures among layers.

The failure propagation in an HPC system can be analogized to a tree which has its root in the lowest layer and its leaves in the highest layer. Going from top to bottom, the diversity of failures decreases while their impacts are greater. Majority of propagated failures, cause new forms of failure in comparison to their original form. A sample of failure propagation on Taurus is shown in Figure 3.4.

In the example shown in Figure 3.4, early detection of the original failure could have prevented the propagation of the failure into vital components of the computing node which finally caused the complete failure of node taurusi5071 at $14:10:01$ on January 1st. The lead time in this example is surprisingly large, which makes it an interesting case for further analysis.

Out of 878 failures on Taurus in the year 2017, 213 failures were simultaneous node failures[3] that affected between 2 and 83 nodes. 665 failures occurred on a single node. Majority of the propagated failures into the kernel space first appeared as user space problems[4].

[3]Confirmed failures which occurred on more than one node during a one-second time window.
[4]At the beginning this appears counterintuitive since failures would normally propagate from bottom to top

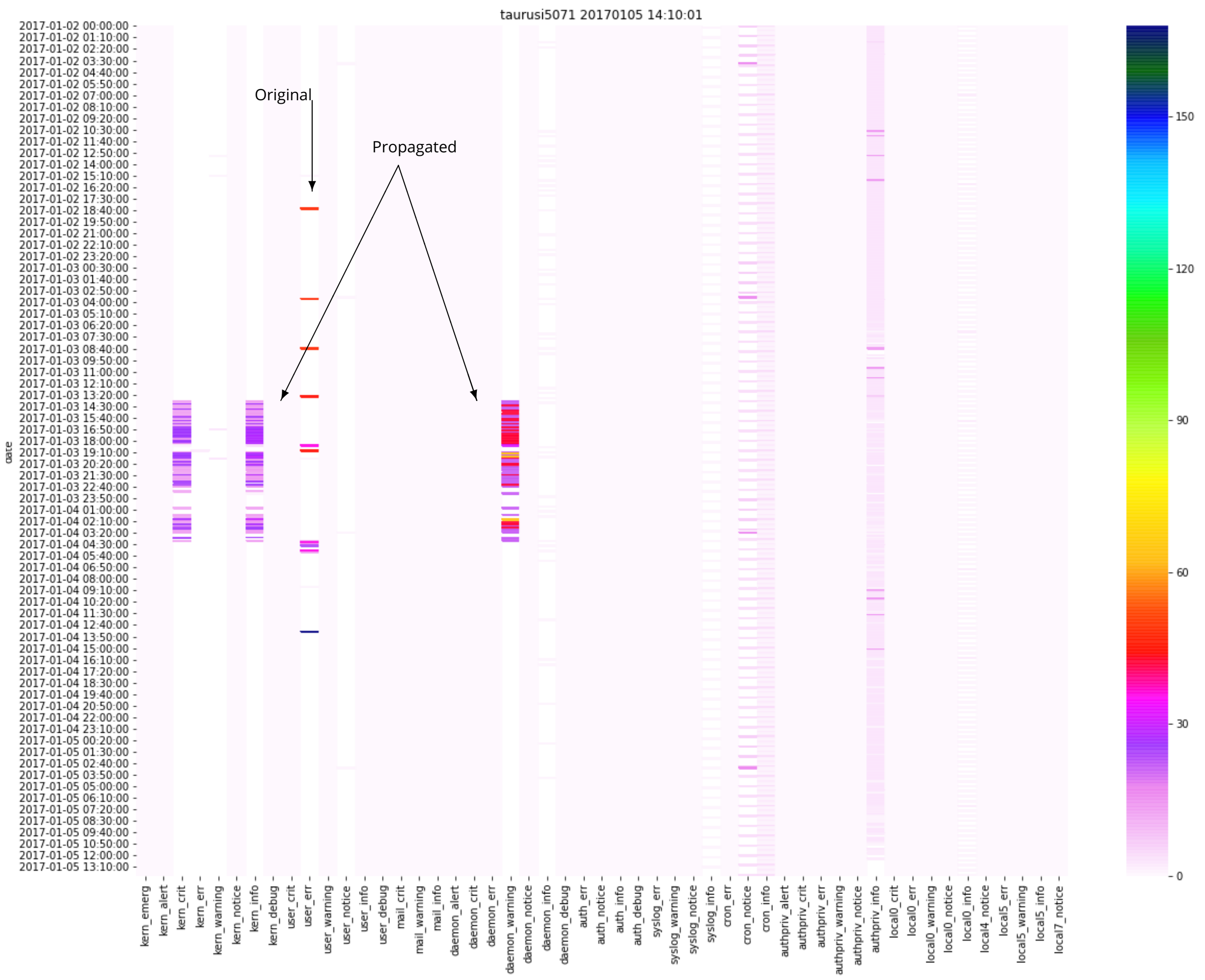
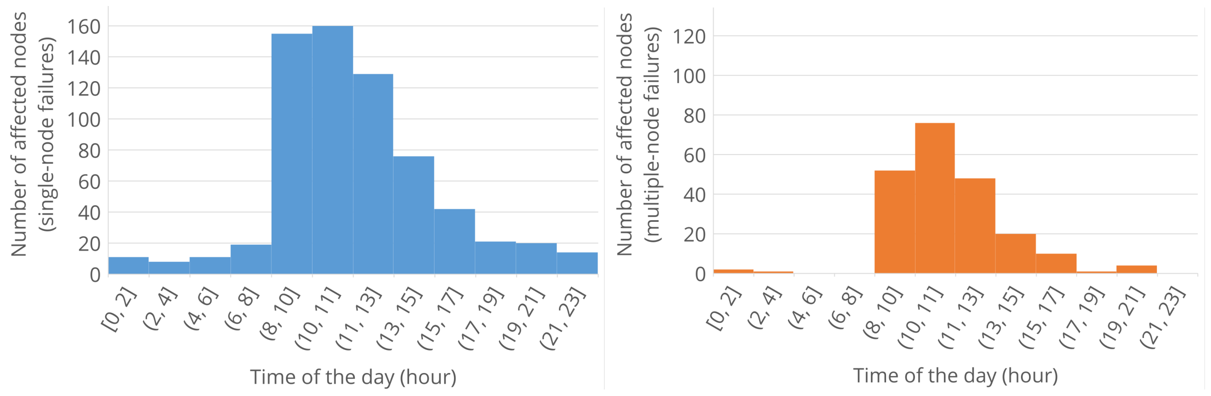


Figure 3.4: Failure propagation in a single node. Vertical axis represents the time in reverse order. Each value on the horizontal axis represents a class of events. The original event was a user error, that later propagated to other parts of the system and caused kernel and daemon errors.



(a) Single-node failures (b) Multiple-node failures

Figure 3.5: Correlation among users behavior and node failures. Most single-node failures and almost all multiple-node failures occurred during the daily working hours, from $08:00$ to $17:00$.

however, although the problem first appears in the user space the root cause lies in the kernel space. Therefore, the failure itself is propagating from bottom to top and the symptoms are first observable once it reaches the user space.

Most single-node failures and almost all multiple-node failures, as shown in Figure 3.5, occurred during the daily working hours ($08:00$ to $17:00$) implying a strong correlation between users activities and node failures[5].

Failure Chains

A sequence of successive failures is called a failure chain. Failure chains are special form of failure propagation. Figure 3.6 illustrates a potential scenario of failure propagation forming a failure chain.

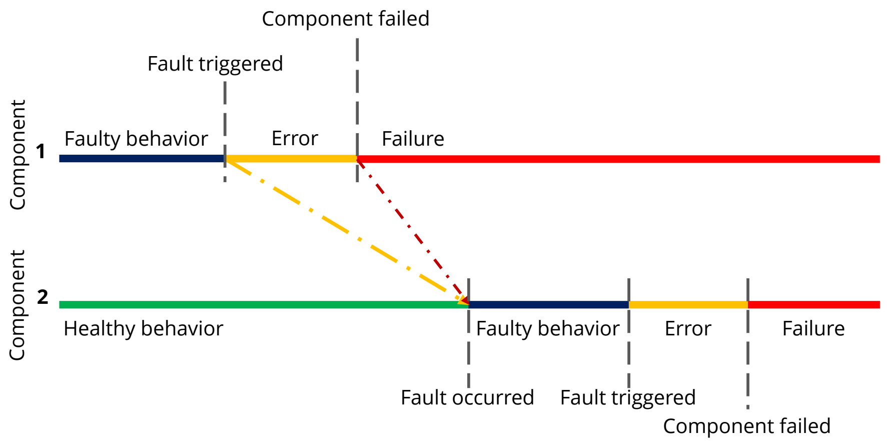


Figure 3.6: Propagation of failures and formation of Failure chains

Failures of a failure chain are identical. It is more probable to detect failure chains as well as their root cause in comparison to mutated propagated failures. Figure 3.7 illustrates a detailed sample of a failure chain in Taurus.

In Figure 3.7 events shown using dotted outline are contributing to the expansion of failure chain, although they may not experience the failure themselves (*interface* nodes). In general failure chains are local phenomena therefore, the majority of failures involved in a chain of failures are physically co-located. However, there are multiple counter examples such as the failure chain shown in Figure 3.7. Detection of failure chains, in useful time and in the presence of interface nodes, is practically infeasible. Omitting the interface nodes divides the failure chain into shorter sub-chains. These sub-chains are detectable and provide adequate functionality to prevent excessive damages.

Side Effects of Protection Mechanism

HPC systems are equipped with various protection mechanisms to prevent hardware damages in emergency situations such as overheating. Although protection mechanisms prevent fatal damages to the system, their interference can unexpectedly change the behavior of HPC systems and introduce additional failures. As shown in Figure 3.8, a major system-wide node failure in October 2016 was caused by the automatic overheating protection mechanism on Taurus.

[5]Similar correlations are observed on DKRZ cluster [4].

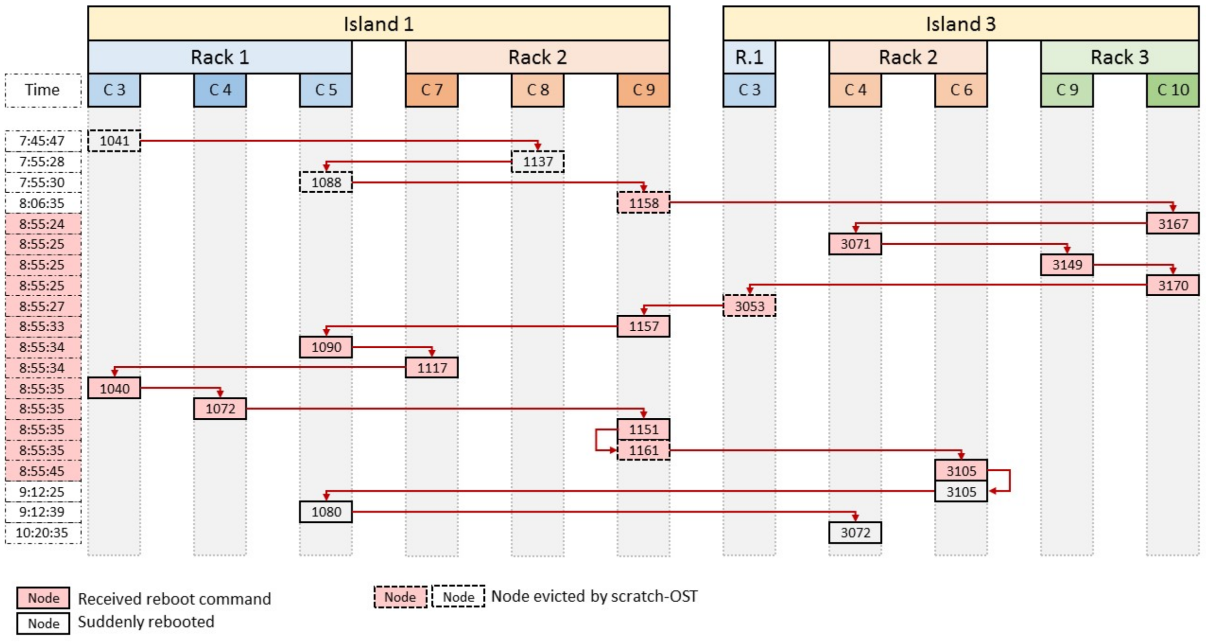


Figure 3.7: Sample of a failure chain on Taurus, including the *interface* nodes that are contributing to the expansion of the failure chain.

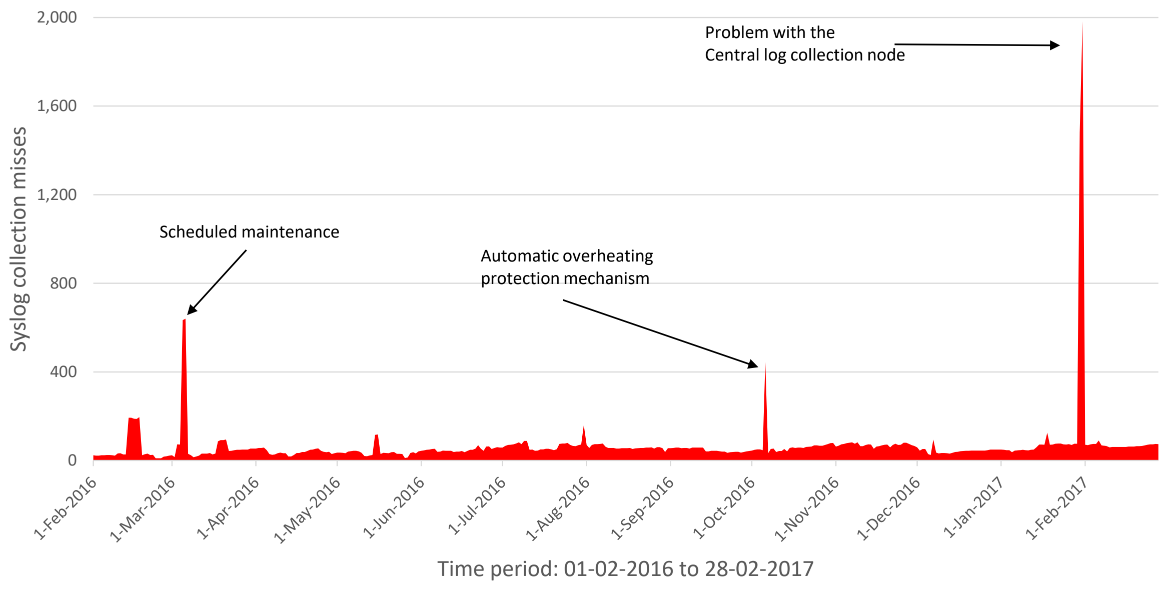


Figure 3.8: Major system-wide failures caused by activation of the automatic overheating protection mechanism on Taurus.

Another example of such interference is shown in Figure 3.20 on page 52. Although the independent automatic protection mechanisms are required, in certain cases (e.g., Figure 3.20) their radical reactions can be delayed or avoided via timely prediction of the upcoming emergency situations. During the period of this work, two major system-wide failures on Taurus were caused by automatic overheating protection mechanisms which led to job loss. Both incidents could have been predicted. Timely prediction of major problems of cooling systems via analyzing system logs is possible. However, due to the small footprint of anomalies, timely prediction of such problems using fully anonymized system logs is extremely complicated.

3.2 Monitoring Data

System logs[6] are the main source of monitoring data in this work. Syslog entries provide a wide range of information about the behavior of the underlying hardware, software, and users. All current TOP500 [234] HPC systems are powered by Linux. The Linux logging protocol is implemented according to RFC 5424 [64] thus, there is a high degree of consistency among system log entries generated by all TOP500 HPC systems[7]. Therefore, the results of syslog-based analysis as a general approach can be applied to all other TOP500 HPC systems without further modifications.

In addition to syslog entries, three other data sources were employed to assist the analysis in this work: *outages database*, *service notifications*, and *job status reports*. The outages database reports all system outages from the users perspective, the service notifications notify users regarding scheduled maintenance and system-wide outages, and the job status reports indicate the final status of submitted jobs after their allocation. Table 3.2 provides an overview of the four main data sources used in this work.

Table 3.2: Four main data sources used as monitoring data

Data source	**Data collection**	**Information**	**Granularity**
System logs	Automatic	Software and hardware	Component
Outage database	Semi-automatic	Service availability	Entire system
Service notifications	Manual	Service availability	Entire system
Job status report	Automatic	Job completion status	Node

System Log Entries

Syslog entries typically consist of four parts: `PRIVAL`, `timestamp`, `source` and `message`. `PRIVAL`, `timestamp` and `source` contain structured data while `message` is unstructured. The `timestamp` denotes the time at which an event occurs. The `source` provides information about the location of the event occurrence, and the `message` describes the event properties. Table 3.3 illustrates three syslog entries divided into their `timestamp`, `source` and `message` fields. In this example, the `timestamp`s are in the UNIX time format[8], the `source`s are node IDs, and the `message`s contain event details.

3.2.1 Data Collection

Taurus syslog entries were collected since September 2014. During this time two fundamental hardware and software upgrades affected the data collection process. From the

[6]System log and syslog are used interchangeably in this work

[7]It worth to mention that /proc/kmsg and /dev/kmsg provide the kernel ring buffer, thus their entries are not in RFC 5424 format.

[8]Also known as POSIX time or UNIX Epoch time.

Table 3.3: Sample of raw syslog entries

Timestamp	Source	Message
1515625261	taurusi1230	(siavash) CMD (/home/config.sh > output.stat)
1515625370	taurusi3417	pam_unix: session closed for siavash
1515625713	taurusi6201	disabling lock debugging due to kernel taint

available collection of syslog entries, the period of January to December 2017 is chosen as the main data source for this work. The main reasons for focusing on the time frame of 01-01-2017 to 31-12-2017 are (1) experiencing the lowest number of updates and unplanned maintenance, (2) having the most reliable data sub-collection, (3) constant monitoring of data consistency, and (4) availability of other monitoring data (i.e. Table 3.2). Figure 3.9 provides an overview of the Taurus syslog collection timeline since 2014.

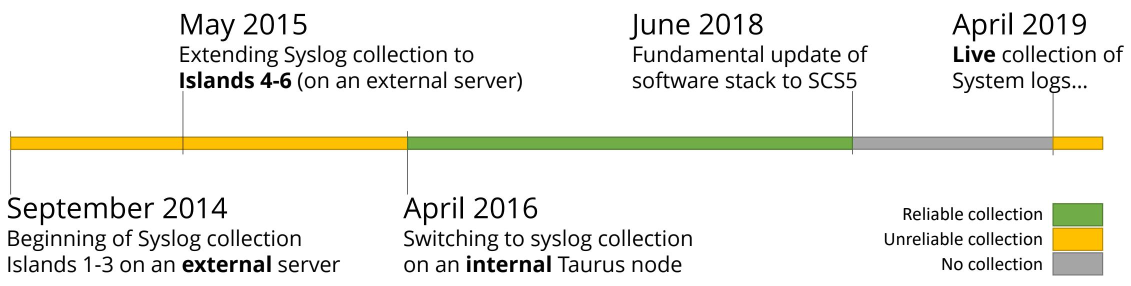
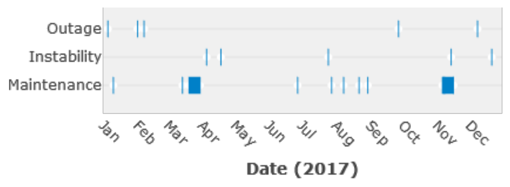

May 2015
Extending Syslog collection to **Islands 4-6** (on an external server)
June 2018
Fundamental update of software stack to SCS5
April 2019
Live collection of System logs...
September 2014
Beginning of Syslog collection Islands 1-3 on an **external** server
April 2016
Switching to syslog collection on an **internal** Taurus node
Reliable collection
Unreliable collection
No collection

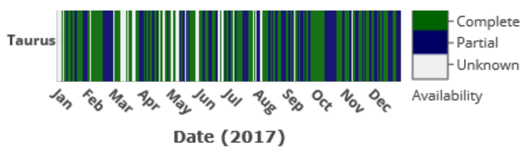
Figure 3.9: Timeline of syslog collection on Taurus

The collection mechanism is passive. Syslog daemons, on each node, collect the syslog entries and forward them to a central log collector. To maintain the neutrality of the results and providing a general approach suitable for all HPC systems, the pre-configured syslog daemons were used without any modifications.

Although the most reliable period of data collection is chosen for the purpose of this study, there are certain gaps in the data. These gaps are mainly incurred due to the interruption of the data collection mechanism. Syslog entries cover the entire period of the year 2017. The job status reports generated by Slurm covers the period of 28-02-2017 to 14-11-2017. Service notifications and outages database, shown in Figure 3.10a and Figure 3.10b respectively, provide information from a higher perspective and are available for the entire period of one year from January to December 2017.



(a) Service notifications



(b) Outages database

Figure 3.10: Availability and Maintenance Notifications of Taurus in 2017

In each section of this work, different subsets of the collected Taurus syslog entries (2014-2018) are used according to the requirements of the respective analysis. However, the behavior of Taurus is generally analyzed for year 2017 with the focus being on the period of 01-03-2017 to 31-10-2017. Existence of gaps in data sources is a common challenge in similar works [163, 15].

During the first round of data collection from September 2014 until April 2016, an external server was used as the central data collector. The external data collector was passively receiving syslog entries over UDP[9] protocol. The UDP protocol was chosen because of its higher transmission speed and lower overhead. However, further analysis revealed various inconsistencies between the remotely collected syslog entries and their local copy on Taurus computing nodes. Most inconsistencies were observed during the high workload periods of the HPC system. Therefore, the data transfer protocol was switched to TCP[10]. Using the TCP protocol, fewer inconsistencies between the remotely collected syslog entries and their local copy were detected. However, high network congestion during major system-wide failures still introduced inconsistencies and delays.

Since April 2016, the central syslog collector resides inside the Taurus HPC system, and is using TCP protocol to collect syslog entries. Using an internal syslog collector, the syslog collection may not be accessible during a major system-wide failure, however, the probability of such major system-wide failures are significantly low and does not impair the functionality of the current setup. Additionally, the collected syslog entries can be backed up on external storage for further analysis and long-term archiving.

Beside the noises and gaps among collected syslog entries, several irregularities were identified. These irregularities are part of the system's normal behavior. Thus, they should not be considered as abnormal behavior. Due to the insignificant difference of these irregular system logs and regular syslog entries, in most cases they can not be easily detected. The proposed approach in this work is highly noise-tolerant. Therefore, syslog irregularities are not affecting the final results. The most frequent irregularities observed on Taurus are (1) out-of-order entries, (2) out of sync clock, (3) daylight saving, and (4) invalid entries.

The sequential arrangement of syslog entries is one of the most important characteristics that make them a highly adequate data source for behavioral analysis. However, among the collected syslog entries various instances of irregular appearance of syslog entries were observed. In Table 3.4 three occurrences of such timestamp confusions are marked in red. Further analysis revealed that this problem is mainly caused by unsuccessful timezone detection. The problem can be solved by restarting the problematic daemons on the HPC system. However, the occurrence of such timestamp confusions on Taurus is so few that they can safely be ignored without significant negative impacts.

During cold boots, similar confusions may happen due to delayed clock synchronization, as shown in Table 3.5. Data analyzing approach in this work skips a short interval of instability after each reboot (and failure) to address the early-life failures suggested by Bathtub Curve (Figure 1.1 on page 4), which additionally addresses the out-of-sync clock phenomena.

Another important point to consider is the missing and overlapping hours caused by

[9]https://tools.ietf.org/html/rfc768
[10]https://tools.ietf.org/html/rfc7805

Table 3.4: System logs with out-of-order timestamps

Example 1	Example 2	Example 3
1514974801 2018-01-03 11:20:01 taurusi5003	1516901631 2018-01-25 18:33:51 taurusi5003	1516344245 2018-01-19 07:44:05 taurusi5003
1514974835 2018-01-03 11:20:35 taurusi5003	1516902001 2018-01-25 18:40:01 taurusi5003	1516344286 2018-01-19 07:44:46 taurusi5003
1514974835 2018-01-03 11:20:35 taurusi5003	1516902241 2018-01-25 18:44:01 taurusi5003	1516344601 2018-01-19 07:50:01 taurusi5003
1514974852 2018-01-03 11:20:52 taurusi5003	1516898641 2018-01-25 17:44:01 taurusi5003	1516345017 2018-01-19 07:56:57 taurusi5003
1514971326 2018-01-03 10:22:06 taurusi5003	1516902601 2018-01-25 18:50:01 taurusi5003	1516341419 2018-01-19 06:56:59 taurusi5003
1514974951 2018-01-03 11:22:31 taurusi5003	1516902834 2018-01-25 18:53:54 taurusi5003	1516345201 2018-01-19 08:00:01 taurusi5003
1514974993 2018-01-03 11:23:13 taurusi5003	1516902834 2018-01-25 18:53:54 taurusi5003	1516345201 2018-01-19 08:00:01 taurusi5003
1514975032 2018-01-03 11:23:52 taurusi5003		1516345201 2018-01-19 08:00:01 taurusi5003
		1516345261 2018-01-19 08:01:01 taurusi5003

Table 3.5: Invalid syslog entries timestamp caused be out of sync system clock

Example 1	Syslog message
1486462610 2017-02-07 11:16:50 taurusi6086	Disabling lock debugging due to kernel taint
1486462610 2017-02-07 11:16:50 taurusi6086	Succesfully Started x86 Adapt Processor Feature Device Driver
1486462610 2017-02-07 11:16:50 taurusi6086	0.0.0.0 c61c 0c clock_step -3459.669479 s
1486459151 2017-02-07 10:19:11 taurusi6086	0.0.0.0 c615 05 clock_sync
1486459152 2017-02-07 10:19:12 taurusi6086	0.0.0.0 c618 08 no_sys_peer
1486459154 2017-02-07 10:19:14 taurusi6086	br0: no IPv6 routers present
1486459154 2017-02-07 10:19:14 taurusi6086	eth0: no IPv6 routers present

daylight saving time changes. Therefore, for the data used in this work (collected in 2017), one missing hour of information on March 26th from 02:00 to 03:00, as well as one overlapping hour of information on October 29th from 02:00 to 03:00 is considered.

The TCP connections guarantee the correct transmission of syslog entries from their source to the central syslog collector. However, due to component failures at the source as well as the silent errors, syslog messages may be incorrectly generated or incompletely stored. Therefore, each syslog entry which does not contain all the expected data fields is considered invalid and is excluded from further analysis.

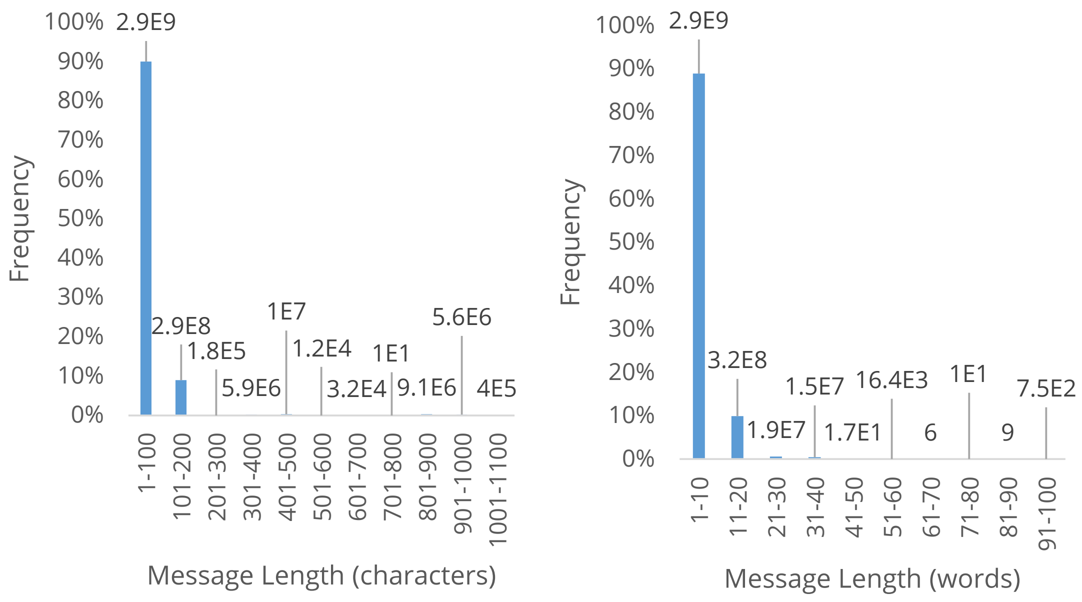
3.2.2 Taurus System Log Dataset

During the year 2017, in total more than 3.2 billion syslog entries with a total size of 344 GiB were collected. Less than 0.33% of the collected syslog entries were incorrect or provided no useful information for the purpose of this work. Detailed statistics regarding the number of syslog entries collected on Taurus divided by their originating island are shown in Table 3.6. The column `/Node/Day` shows the average number of entries generated by a single node per day in its respective island. The `#Event patterns` column indicates the number of extracted unique event patterns per island. The event pattern extraction process is further explained in Section 3.3.1.

Table 3.6: Statistics of Taurus syslog Entries in year 2017

		#Log entries			#Event patterns	
Island	#Nodes	Total	Invalid	/Node/Day	Total	Exclusive
1	270	299,046,032	535,719	3,034	1,601	291
2	108	58,106,243	603,848	1,474	1,360	357
3	180	56,778,161	16,443	864	1,254	215
4	264	490,299,895	4,585,815	5,088	1,667	285
5	612	1,034,193,160	1,846,396	4,629	1,945	340
6	612	1,328,650,822	3,264,407	5,947	2,168	483
All islands	2046	3,267,074,313	10,852,628	4,374	4,026	1,488

Since the `message` field of syslog entries is an unstructured free-format text, the message length has a great impact on the performance and implementation of log processing methods. The message field of Taurus syslog entries has a diverse length of 1 to 1108 characters. Most of this variation is caused by variables such as hardware addresses and file paths. Although there are extremely long messages with 1108 characters, the majority of Taurus syslog messages are shorter than 100 characters. Figure 3.11a illustrates the frequency of syslog messages with their lengths count in characters.



(a) 90% of syslog messages are shorter than 100 characters

(b) 89% of syslog messages are shorter than 10 words

Figure 3.11: Frequency of Taurus syslog messages with various lengths

The impact of variables on diversifying the length of syslog messages can be reduced via performing word processing rather than character processing. Considering words as the measurement unit decreases the length diversity by the factor of 10. Although Taurus syslog messages consist of 1 to 99 words, using the new measurement unit, the majority of Taurus syslog messages are shorter than 10 words. Which is an acceptable message length for achieving a near real-time processing performance. Figure 3.11b illustrates the frequency of syslog messages with their length measured in words. Converting syslog messages into event patterns (Section 3.3.1), reduces the size of syslog entries to a constant length that significantly improves the syslog processing performance.

3.3 Data Preparation

The entire procedure of syslog generation, transmission, and collection is prone to errors caused by software and hardware failures. Same principle applies to all other monitoring data collected from computing nodes such as power consumption, CPU load, or memory utilization. Therefore, data preparation is necessary.

The goal of this step is the conversion of monitoring data to noise-less anonymized discrete time series (reasons are explained in Chapter 2). Furthermore, the relative semantic of anonymized data entries must be preserved. The relative semantic refers to the in-context semantic of an entry in comparison to other entries in that context. As an instance, both pairs of log entries in Example 3.1 deliver similar relative semantics that a user was connected and disconnected correctly, although, the second pair is fully anonymized.

Example 3.1: Original and anonymized system logs with similar relative semantics

```
A1 Accepted publickey for root from 192.64.12.13 port 32431 ssh2
A2 Received disconnect from 192.64.12.13: 12: disconnected by user

B1 Accepted publickey [...]
B2 Received disconnect [...]
```

Among the available monitoring data in HPC systems such as power consumption, memory usage, cpu workload, jobs status, user behaviors, and components aging, system logs are the most sophisticated input data due to their free-form and qualitative nature. Therefore, system logs require additional steps of data preparation in comparison to most other monitoring data. Since syslog entries are the main source of monitoring data in this work, the proposed data preparation methods in this section are exemplified using Taurus syslog entries. However, these methods are applicable to any other forms of monitoring data.

3.3.1 Users and Systems Privacy

The existence of sensitive data within the system logs as explained in Chapter 2 raises serious concerns about their storage, analysis, dissemination, and publication. The anonymization of system logs is a mean to address the data privacy challenge. During the process of anonymization, the sensitive data will be eliminated. The remaining data is considered as cleansed data. However, there is the probability that sensitive data passes through filters of anonymizers and leaks into the cleansed data. To the best of author’s knowledge, no existing automatic anonymization method guarantees full user privacy.

It is worth mentioning that common data and output perturbation[11] methods that are beneficial to provide differential privacy [125] are not suitable for fulfilling the goals of this work. Since in this work (1) the entire dataset is intended to be publicly available, (2) it is aimed to reduce the size of dataset as much as possible, (3) there are no trusted parties out of the HPC systems, and (4) the sensitive data cannot be kept for long-term storage and analysis, the hybrid techniques such as PrivApprox [235] are also not a good fit for the purpose of this work.

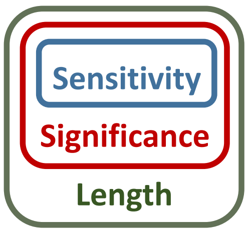
Anonymization of system logs, as a side effect, reduces the usability of data for further analysis. After a certain degree of anonymization, the cleansed syslog entries lose their general semantic, however, they remain useful for certain statistical analysis, such as time series analysis and anomaly detection. At this stage, it is possible to encode syslog entries

[11]Providing noisy or approximate answers to data queries in order to prevent revealing individual records in a database. Swapping, random sampling, varying perturbation, and randomization are some of the most common methods of data and output perturbation.

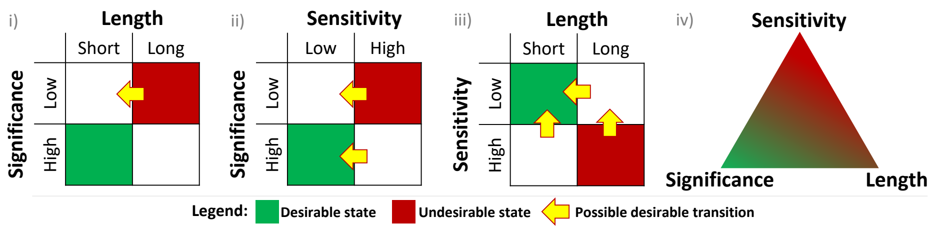
into shorter strings. Such encoding reduces the required capacity for the storage of system logs. Shortening the log entries' length also reduces their processing complexity and, therefore, improves the performance of further analysis on syslogs. Furthermore, an irreversible encoding guarantees full user privacy via masking any potential sensitive data leakage. It is important to note that the sensitivity and the significance of syslog entries are relative to HPC system's policies.

A *term* is a string of characters with certain semantics (e.g., root, 2, CMD). Each term is either *constant* or *variable*. A constant term remains identical in all syslog entries. A variable term, in contrast, takes different values across different syslog entries. Each term in a syslog entry, depending on the policies of the computing system that it originates from, may (or may not) be considered as sensitive data. The same degree of relativity applies to the significance of syslog entry terms. The significance of syslogs entries is assessed depending on the chosen data analysis method. Even though the classification of each term as sensitive or significant is relative (e.g., not, semi, or highly significant), the final assessment of sensitivity and significance of a term is a binary value of true or false. Therefore, every single term in a syslog entry can only be sensitive/significant or nonsensitive/nonsignificant (e.g., a username).

A zero-length term is not significant. A significant term has a nonzero length and can be either sensitive or nonsensitive. A sensitive term is significant. Figure 3.12a illustrates the relation between the sensitivity, the significance and the length of syslog terms.



(a)



(b)

Figure 3.12: (a) The sensitivity, significance, and length of terms in syslog entries and their relation. (b) Trade-off scenarios between the significance, sensitivity, and length of a system log entry. Each of the i, ii, and iii illustrations depicts the four possible states of a syslog entry based on its sensitivity, significance, and length. The trade-off triangle in illustration iv shows the trade-off between the three parameters (sensitivity, significance, length) in a single unified view.

A triple trade-off exists between sensitivity, significance, and length of syslog entries. The preferred input for data analysis should contain significant content to provide accurate information, nonsensitive data to comply with users privacy, and shorter length to facilitate online analysis using less resources and higher performance. Figure 3.12b, regardless of the system policies and syslog analysis methods, schematically illustrates this trade-off. This illustration shows that a syslog entry can be in four distinct states. Green color states denote best conditions while red color states denote undesirable conditions. White color states represent median conditions. The yellow arrows in Figure 3.12b indicate the possible preferable actions to improve the overall condition. The significance of syslog entries cannot be increased, and reducing the length of syslog entries decreases their significance, which is an undesirable transition. Therefore, the remaining possibilities to improve the condition are (1) decreasing the sensitivity of syslog entries or (2) reducing their length.

Syslog entries are generated using static templates predefined in components of the computing system. Assuming the sample syslog entry E_1: "`1462053899 taurusi1013 Accepted publickey for Siavash from 4.3.2.1`". In this entry, "`146205389`" is the timestamp, "`taurusi1013`" is the source, and the rest of the line "`Accepted publickey for Siavash from 4.3.2.1`" is the message. In the message part, the terms `Accepted`, `publickey`, `for`, and `from` are constant terms, while `Siavash` and `4.3.2.1` are variable terms, in the sense that for the above variable terms, the user name and IP can vary among users and machines. The syslog entries shown in Example 3.2 are generated using a similar template. The constant terms are marked in red.

Example 3.2: Constant and variable terms in system logs

```
1462053899 taurusi1013 Accepted publickey for siavash from 4.3.2.1
1462053909 taurusi4124 Accepted publickey for root from 192.168.1.15
1462054899 taurusi6312 Accepted publickey for parya from 12.38.121.49
```

This step attempts to transit current condition towards a better condition as shown in Figure 3.12b via eliminating sensitive terms (anonymization) or reducing the syslog entries length. However, each sensitive term is also a significant term thus, carries information. Therefore, certain information may be lost during the anonymization process. The goal is to preserve the highest possible (permissible) *quality* of syslog entries throughout the anonymization process while preserving user privacy. To achieve this goal PαRS is proposed [236, 237].

The PαRS anonymization approach consists of 8 steps. The input data is a stream of log entries split into content and metadata parts. For the case of syslog entries, the content is the `message` field of syslog entry and the metadata covers the rest of the entry (e.g., `timestamp` and `source`). Since metadata contains no sensitive data, PαRS targets the sensitive data within the content part of the input data as following[12]:

Step 1) The variable terms in the syslog entries are divided into 3 groups:

(a) sensitive (e.g., username, IP address),

(b) significant (e.g., temperature, memory address),

(c) nonsignificant (e.g., cron job name, path).

Step 2) The sensitive terms are eliminated to comply with the privacy policies.

Step 3) The nonsignificant terms are replaced with predefined constants.

Step 4) Every syslog entry that does not have any remaining variable terms (event pattern[13]), is mapped to a hash key, via a collision-resistant hash function. The hashing step is called encoding.

[12]Figure 3.14 on page 43 provides a graphical illustration of the PαRS workflow.
[13]Also known as log key or message type; examples are shown in Table 3.9c.

Step 5) The quality of the remaining syslog entries is measured with a utility function.

Step 6) When it is revealed that removing a significant term from the syslog entry improves the quality of syslog, that particular term is replaced with a predefined constant.

Step 7) The remaining processed syslog entries that do not contain additional *variable* terms, are mapped into hash keys (similar to step (4) above).

Step 8) Upon completion of steps (4) and (7), the hash key codes can be optimized based on their frequency of appearance.

Regular expressions are used for the automatic detection of variable terms within syslog entries. Categorization of automatically detected terms into sensitive and/or significant is performed based on the information in Table 3.7.

Table 3.7: Classification of syslog entry terms into sensitive and/or significant. Severity denotes the importance of the characteristics for the respective terms.

Term	**Sensitivity**	**Severity**	**Term**	**Significance**	**Severity**
`User Name`	Y	10	`accept⋆`	Y	07
`IP Address`	Y	08	`reject⋆`	Y	10
`Port Number`	Y	01	`close⋆`	Y	08
`Node Name`	Y	03	`⋆connect⋆`	Y	09
`Node ID`	Y	03	`start⋆`	Y	02
`Public Key`	Y	10	`⋆key⋆`	Y	01
`App Name`	N	00	`session`	Y	07
`Path / URL`	N	00	`user⋆`	Y	05

This information is manually inferred from the policies and conditions of the host high-performance computing system. Automatically detected variable terms which do not belong to any of the sensitive and significant categories are considered as nonsignificant. PαRS uses the variable length, collision resistant hash algorithm SHAKE-128 [238, 239] to encode the syslog entries.

During the early stages of analysis at the beginning of this work, manually extracted regular expressions were used to detect sensitive terms in syslog entries. Table 3.8 contains 15 (out of 38) manually extracted regular expressions which were used to detect variable terms in syslog entries of Taurus. Regular expressions shown in Table 3.8 are dedicatedly generated to match Taurus syslog entries. The order of their application is important since certain patterns are subsets of other patterns. Although these regular expressions are compatible with log entries of many HPC systems, to address other sources of monitoring data as well as potential major changes in the templates of log entries, an automatic approach was required. Section 3.3.3 provides detailed information regarding the automatic approach for extracting regular expressions from general log entries.

Even though most variables can be detected with these basic regular expressions, in an unlikely case of similarity between variables and constants, the regular expression may not be able to differentiate between constants and variables correctly. For example the username `panic` may be misinterpreted as a constant value like `kernel panic`. Various possibilities for such misinterpretations are imaginable, yet unlikely. Therefore, the

Table 3.8: Regular expressions used to detect certain terms within Taurus syslogs

Variable	**Regular expression**
Path	`([\(\s\,\>\:\=])([\/][a-z0-9_\.\-\:]*)+`
Version	`([\w\.\-]+x86_64)`
Email	`([a-z0-9_\-\.]+@([a-z0-9_-]+\.)+[a-z]+)`
DateTime	`(\d{4}-\d{2}-\d{2})T(\d{2}:\d{2}:\d{2})`
IPv4	`(\d+\.\d+\.\d+\.\d+)`
Port	`([\W])(port \d+)`
Parameter	`(\$[a-z0-9_]+)`
URID	`(uid=[\w\-]+)`
User	`(for )((user\ )*[a-z0-9_-]+)`
Library	`([a-z0-9_\-]+\.so(\.\d*)*)`
Hardware address	`(0[x][a-f0-9]+\-0[x][a-f0-9]+)`
Hex Number	`(0[x][a-f0-9]+)`
Percentage	`(\d+\.*[\d]*\%)`
Serial number	`((\s)([a-f0-9\.\-]+\:)+(\s))`
Size	`([^a-z0-9])(\d+[bkmg])([^a-z0-9])`

overhead imposed by employing sophisticated methods, such as named entity recognizer (NER) to detect misinterpretations, is not justifiable. In contrast, encoding is a robust and lightweight approach to address all forms of misinterpretations. In such scenarios, the undetected variables are considered as constants and will be eliminated through the encoding step. The final encoding step masks any potential data-leakage and guarantees the highest attainable level of anonymization.

Table 3.9 shows an example of applying PαRS on Taurus syslog entries. Table 3.9a contains the original syslog entries (the input data). De-identified entries (event patterns) are shown in Table 3.9b, and Table 3.9c contains the fully anonymized (encoded) syslog entries. While the message part of syslog entries are fully anonymized, the metadata remains unchanged.

Through the anonymization phase, sensitive terms of syslog messages were de-identified (page 39, step (4)). The sensitive terms can be de-identified in various forms according to the intended usage. In contrast, the nonsignificant terms are de-identified always via substitution by an identical symbol (e.g., all paths such as `/usr/bin/` will be substituted by `#PATH#`). This form of de-identification in which, all instances of a variable term is substituted with an identical symbol is referred to as global de-identification. On the other hand, the substitution of each instance of a variable term by an individual symbol is called individual de-identification. The global de-identification, by default, applies to all nonsignificant variable terms. Global de-identification provides the highest degree of generalization, while individual de-identification prevents any generalization. Categorizing syslog terms into multiple groups and substituting all terms of each group with an identical group symbol is called group de-identification. The group de-identification provides various degrees of generalization according to the grouping granularity (variable granulation). The trade-off between privacy and distinguishability of de-identified syslog entries via individual, group, and global de-identification methods is shown in Figure 3.13.

(a) Raw system log entries

Timestamp	Source	Message
1515625261	taurusi1230	(siavash) CMD (/home/config.sh > output.stat)
1515625370	taurusi3417	pam_unix: session closed for siavash
1515625390	taurusi4023	(parya) CMD (/usr/bin/cmon > mon-1.log)
1515625713	taurusi6201	disabling lock debugging due to kernel taint

(b) Event patterns

Timestamp	Source	Message
1515625261	taurusi1230	(#USER#) CMD (#PATH# > #PATH#)
1515625370	taurusi3417	pam_unix: session closed for #USER#
1515625390	taurusi4023	(#USER#) CMD (#PATH# > #PATH#)
1515625713	taurusi6201	disabling lock debugging due to kernel taint

(c) Encoded system log entries

Timestamp	Source	Message
1515625261	taurusi1230	1808e388
1515625370	taurusi3417	0964de42
1515625390	taurusi4023	1808e388
1515625713	taurusi6201	59f2da35

Table 3.9: Anonymization of syslog entries via PαRS

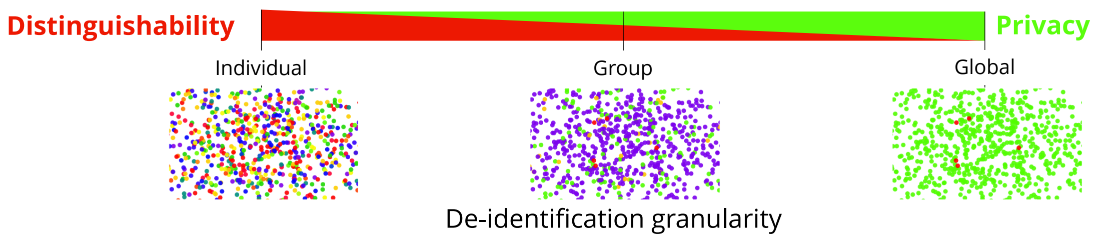
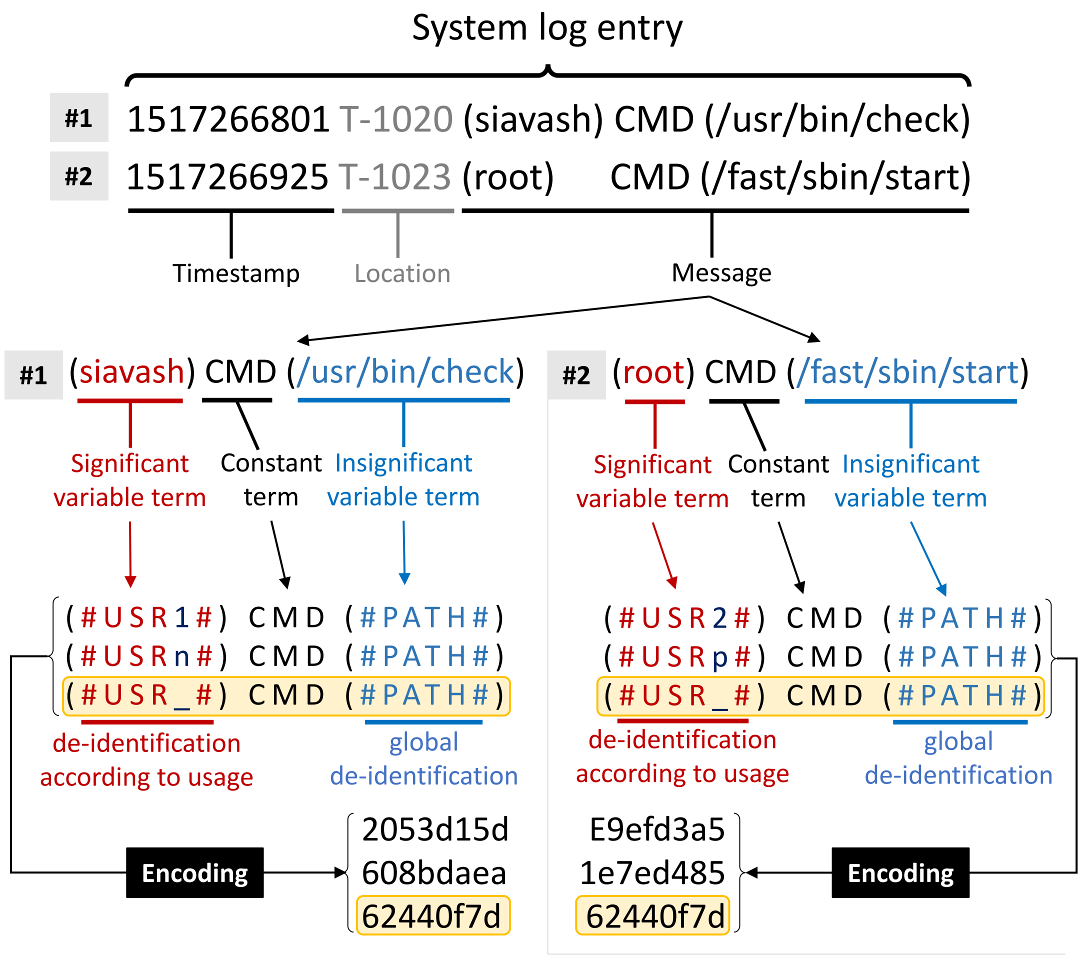


Figure 3.13: Distinguishability trade-off. Better privacy is achieved via increasing the granularity towards global de-identification in exchange for worse distinguishability.

Considering that on `January 29, 2018 11:00:01 PM` user `siavash` executed the command `/usr/bin/check` on computing node `taurusi1020`. This event in system logs is shown as entry E_2: `1517266801 taurusi1020 (siavash) CMD (/usr/bin/check)`. In this entry `siavash` and `/usr/bin/check` are variable terms, and `CMD` is a constant term. In accordance to the intended future usage of the anonymized system logs in this work, namely behavior analysis, `siavash` is considered as a significant variable term and `/usr/bin/check` as a nonsignificant variable term. The entries of an ideal dataset for behavioral analysis should have a certain degree of distinguishability as well as similarity. Based on the entries' similarity a majority may form (the normal behavior), while the outliers (abnormal behavior) could be still distinguished from this majority. To address this challenge, the event patterns where introduced.

Figure 3.14 illustrates the PαRS anonymization workflow. The event pattern of a syslog entry is generated through global de-identification of all variable terms in the message part of the respective syslog entry. Global de-identification extracts identical event patterns from syslog entries with similar messages. Therefore, the similarity between syslog entries is preserved such that the results of further data analysis are not skewed.


System log entry
#1 1517266801 T-1020 (siavash) CMD (/usr/bin/check)
#2 1517266925 T-1023 (root) CMD (/fast/sbin/start)
Timestamp
Location
Message
#1 (siavash) CMD (/usr/bin/check)
#2 (root) CMD (/fast/sbin/start)
Significant variable term
Constant term
Insignificant variable term
Significant variable term
Constant term
Insignificant variable term
(#USR1#) CMD (#PATH#)
(#USRn#) CMD (#PATH#)
(#USR_#) CMD (#PATH#)
(#USR2#) CMD (#PATH#)
(#USRp#) CMD (#PATH#)
(#USR_#) CMD (#PATH#)
de-identification according to usage
global de-identification
de-identification according to usage
global de-identification
Encoding
2053d15d
608bdaea
62440f7d
E9efd3a5
1e7ed485
62440f7d
Encoding

#USR1#, #USR2#: Individual de-identification; replacing each username with a new symbol
#USRn#, #USRp#: Group de-identification; replacing usernames according to the user groups
#USR_#, #USR_#: Global de-identification; replacing all usernames with an identical symbol
🟨 : *Event pattern*, the result of global de-identification of all variable terms

Figure 3.14: The *event pattern* is the result of a full de-identification of syslog entries. Note that other forms of de-identification are also possible. The final encoding step guarantees full data privacy.

Although a hash key might appear devoid of semantics, given the one-to-one relation between hash keys and event patterns, it is always possible to reaccredit the original semantics to the pattern denoted by a hash key. This accreditation can only be done by the owners of the adequate information about the event patterns and the hashing function. However, regardless of the reaccreditation of the original semantics to the pattern, it is always possible to track similar events according to the similarity of their event patterns without endangering the users privacy.

The final output of PαRS consists of the log entries metadata in its original format and the anonymized message as hash key. Depending on the strictness of the applicable privacy guidelines, the relative semantics of each hash key can also be added to the final output. Table 3.10 contains the final output of applying PαRS on the given sample syslog entries in Table 3.9a, accompanied by the semantics of hash keys.

Table 3.10: Final output of PαRS

Timestamp	Source	Hash key	Semantics
`1515625261`	`taurusi1230`	`1808e388`	`A command executed by a user`
`1515625370`	`taurusi3417`	`0964de42`	`A user logged out`
`1515625713`	`taurusi6201`	`59f2da35`	`Kernel is in taint mode`

The anonymization phase may severely affect the quality and usefulness of the input data. The PαRS anonymization approach proposes a utility function to quantify the quality of anonymized system log entries. It has been shown that the quality of anonymized syslog entries for conducting behavioral analysis for failure detection remains at an adequate degree, such that the anonymized system logs are useful for further analysis [240].

Since 25th of May 2018 the general data protection regulation (GDPR) is enforced [120]. Complying with the GDPR and the current Technische Universität Dresden (TUD) privacy regulations [241], syslog entries must be excessively anonymized such that the remaining significant terms among system logs are so few that it is not worth to preserve them (e.g., "`failed password for #USER# from #IPv4# port 32134 ssh2`"). The only remaining useful information in these cases is the relative semantics (meaning) of the event pattern itself. For the above example the semantics is `authentication via ssh failed`.

Further analysis on Taurus syslog entries anonymization indicated that: (1) The cleansed system logs consist of approximately 90% nonsignificant entries (after performing the mandatory de-identification), (2) Approximately 5% of the entries are constant (without any variable terms), (3) approximately 5% are entries with significance (retained their useful properties even after de-identification). Following the necessary de-identification enforced by policy guidelines, 95% of syslog entries no longer have any significance and therefore, can be directly converted to hash keys. The 5% of syslog entries which still had a certain degree of significance even after de-identification, may remain untouched. However, statistical analysis that focus on anomaly detection[14] does not require such information.

Furthermore, the overhead of calculations for preserving these 5% semantics is not justifiable. Therefore, according to the chosen analysis method in this work the remaining 5% of the syslog entries will be also encoded to hash keys. Thus, regardless of the quality measurements, PαRS applies the global de-identification on all Taurus syslog entries.

3.3.2 Storage and Size Reduction

The volume of generated system log entries is in proportion to the system size. The storage of syslog entries, produced by large parallel computing systems, in view of their analysis requires high storage capacity. The number of syslog entries generated by each node of Taurus during 2017 is shown in Figure 3.15.

Size reduction can be achieved via any general lossy or lossless compression algorithm. When the applied compression method does not change the significance and sensitivity of

[14] e.g., PrefixSpan, Spade, SPAM, GSP, CM-SPADE, CM-SPAM, FCloSM, FGenSM, PFP-Tree, MKTPP, ITL-Tree, PF-tree, and MaxCPF.

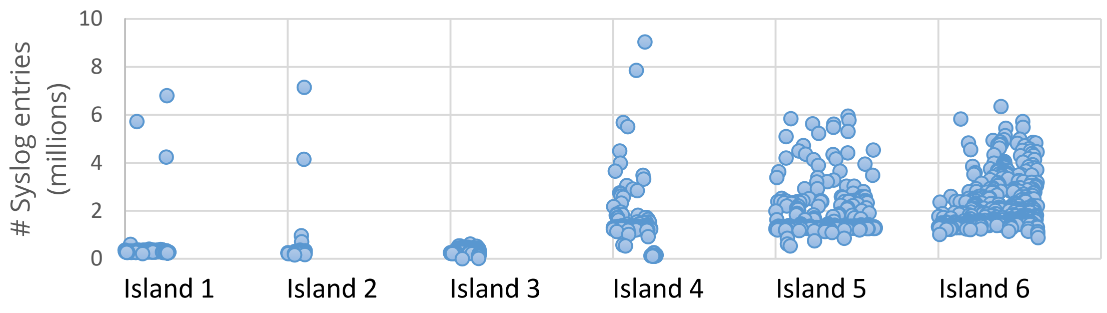


Figure 3.15: Number of Taurus syslog entries per node in 2017

syslog entries, from the perspective of this work, it is considered as lossless. If the compression method modifies the significance or sensitivity, it is considered as lossy and is taken as an additional level of anonymization rather than compression. A careful consideration of various effective compression algorithms, including Brotli, Deflate, Zopfli, LZMA, LZHAM, and Bzip2 revealed that in affordable time, compression could reduce the size of non-anonymized system logs to 25% of their original size (75% reduction) [242, 242, 243, 244]. In contrast, the size of anonymized system logs after anonymization via PαRS is only 5% to 10% of the original size (90%-95% reduction) [245].

Log aggregation can also significantly reduce the size of the final log collection. However, since entries will be merged or discarded during the aggregation process, this work considers log aggregation as a form of lossy compression thus, an additional level of anonymization. Furthermore, the log aggregation is affecting entries prior to the log collection phase. Therefore, it is considered similar to any other configuration which is part of the HPC system characteristics. As a side note, log compression during the transfer phase is also not recommended [127].

In contrast to the compressed syslog entries which require a decompression phase before any further analysis, the syslog entries that are anonymized using PαRS are ready to be processed in their current hash key format. In addition, compression (as well as encryption) are bidirectional and may endanger the users privacy. However, the unidirectional anonymization approach of PαRS eliminates all potential privacy threats and also preserves the required distinguishability among log entries. Table 3.11 denotes the size reduction achieved via anonymization of a syslog dataset consisting of more than 8.6 billion entries.

Table 3.11: Size reduction by PαRS, applied on syslog entries collected from Taurus in the years 2016 and 2017.

	Original	**Anonymized**
Syslog size	984.2 GiB	163.4 - 49 GiB[15]
No. of entries	$8.6 * 10^9$	$8.6 * 10^9$
Unique entries	$> 10^8$	< 3000

[15]The output size with and without metadata.

3.3.3 Automation and Improvements

This section further improves the previously proposed methods to achieve a higher accuracy and automation. There are available methods for automatic generation of regular expressions [246] However, automatic generation of regular expressions for natural languages is extremely inefficient [247]. As mentioned in Section 3.3.1 regular expressions are used to identify various terms in system log entries and to substitute them with relevant invariants (constants). For this purpose a set of 38 regular expressions, partly shown in Table 3.8, were extracted from Taurus syslog entries. Although these 38 regular expressions are fulfilling the expected tasks on Taurus syslog entries, some of these expressions are dependent on specific log patterns which prevent proper generalization of the proposed approach. Therefore, an automatic regular expression generator has been developed [248].

Automatic generation of regular expressions for a given text is a complex and time-consuming task [247]. However, restricting the input text to a set of syslog entries significantly reduces the time and computational complexities. Each application has a limited number of message templates which are used to generate syslog messages. Therefore, although the message part of syslog entries is unstructured and may hold messages with any pattern, the number of these message patterns is limited by the number of running applications and daemons (syslog message generators).

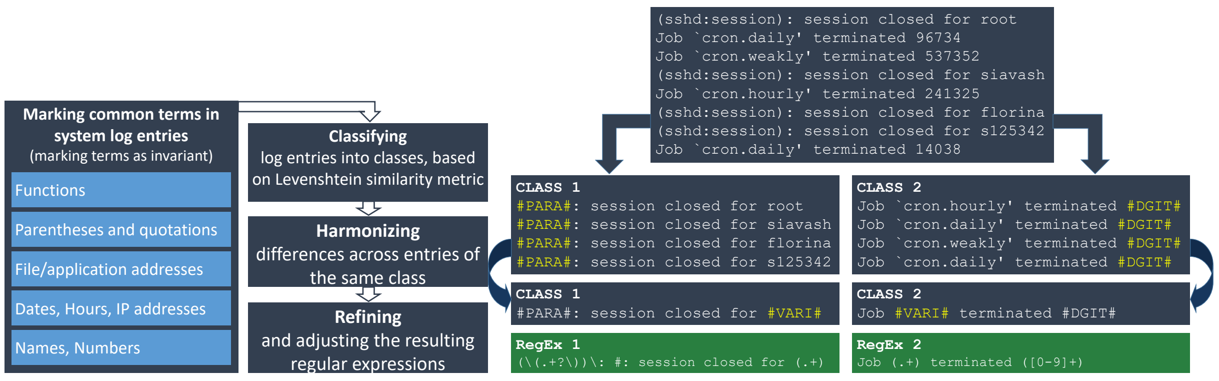
Considering Example 3.3 with four sample syslog entries. Although the `username`, `IP address`, and `port number` (shown in red) are different in each syslog entry, the general semantics for these particular events is identical. The similarity among syslog entries generated for a particular event facilitates the automatic generation of regular expressions for system log entries. Figure 3.16a illustrates the workflow of automatic regular expression generation. The last line in Example 3.3 (marked as `RE`) denotes the automatically generated regular expression. An example of automatic regular expression generation for multiple syslog entries is shown in Figure 3.16b. The same method can be applied on any other monitoring data that contains unstructured free-form text.

Example 3.3: Similar syslog entries

```
A] failed password for siavash from 192.168.3.5 port 5734 ssh2
B] failed password for u7754 from 192.168.4.35 port 30740 ssh2
C] failed password for parya from 192.168.7.43 port 3405 ssh2
D] failed password for root from 192.168.5.74 port 5407 ssh2

RE] failed password for (\w+) from 192.168.(\d+).(\d+) port (\d+) ssh2
```

As shown in Figure 3.16a, five detection classes are used to identify textual structures such as functions, parentheses, dates, IP addresses, and so forth in system logs. Further analysis of system logs revealed that only two detection sub-classes of *parentheses* and *numbers* are sufficient for correct detection of textual structures in syslog entries.

Two main tasks must be fulfilled in view of understanding system behavior via syslog entries. First, the identification of similar events and second, the distinction of the differences between similar events. Table 3.12 provides a set of syslog entries with their hash keys (in-


Marking common terms in system log entries
(marking terms as invariant)
Functions
Parentheses and quotations
File/application addresses
Dates, Hours, IP addresses
Names, Numbers
Classifying
log entries into classes, based on Levenshtein similarity metric
Harmonizing
differences across entries of the same class
Refining
and adjusting the resulting regular expressions

```
(sshd:session): session closed for root
Job `cron.daily' terminated 96734
Job `cron.weakly' terminated 537352
(sshd:session): session closed for siavash
Job `cron.hourly' terminated 241325
(sshd:session): session closed for florina
(sshd:session): session closed for s125342
Job `cron.daily' terminated 14038
```

```
CLASS 1
#PARA#: session closed for root
#PARA#: session closed for siavash
#PARA#: session closed for florina
#PARA#: session closed for s125342
```

```
CLASS 2
Job `cron.hourly' terminated #DGIT#
Job `cron.daily' terminated #DGIT#
Job `cron.weakly' terminated #DGIT#
Job `cron.daily' terminated #DGIT#
```

```
CLASS 1
#PARA#: session closed for #VARI#
```

```
CLASS 2
Job #VARI# terminated #DGIT#
```

```
RegEx 1
(\(.+?\))\: #: session closed for (.+)
```

```
RegEx 2
Job (.+) terminated ([0-9]+)
```

(a) Workflow (b) Example

Figure 3.16: Automatic generation of regular expressions for syslog entries. A variation of Levenshtein similarity metric [249] is used for the classification of log entries.

dividual de-identification) and event patterns (global de-identification). The log entries #2 and #9 from Table 3.12 report the occurrence of similar events. However, different users triggered each of these similar events. For a detection mechanism it is important to understand that the same type of event occurred by different users of the system. The event patterns in Table 3.12, enables the detection mechanism to identify similarities between events, while the hash keys are employed to represent the differences.

Table 3.12: Pre-anonymized entries; nonsignificant terms are de-identified

#	Message	Significant term	Hash key	Event pattern
1	`(siavash) CMD (#PATH#)`	siavash	bb2d95d2	66dc2742
2	`(parya) CMD (#PATH#)`	parya	23343ad0	66dc2742
3	`(siavash) CMD (#PATH#)`	siavash	bb2d95d2	66dc2742
4	`starting 0anacron`	oanacron	47c6b01d	dd740712
5	`Anacron started on #TIME#`	Anacron	22bb4f1a	e5a59462
6	`Jobs will be executed sequentially`	-	f1e7eac3	f1e7eac3
7	`Normal exit (0 jobs run)`	0	e46c1bdb	eac7924f
8	`finished 0anacron`	oanacron	76690e70	a5803a8a
9	`(siavash) CMD (#PATH#)`	siavash	bb2d95d2	66dc2742
10	`(root) CMD (#PATH#)`	root	752d8638	66dc2742
11	`(root) CMD (#PATH#)`	root	752d8638	66dc2742
12	`(siavash) CMD (#PATH#)`	siavash	bb2d95d2	66dc2742
13	`(parya) CMD (#PATH#)`	parya	23343ad0	66dc2742
14	`(siavash) CMD (#PATH#)`	siavash	bb2d95d2	66dc2742
15	`(siavash) CMD (#PATH#)`	siavash	bb2d95d2	66dc2742
16	`starting 0anacron`	oanacron	n 47c6b01d	dd740712
17	`Anacron started on #TIME#`	Anacron	22bb4f1a	e5a59462
18	`Jobs will be executed sequentially`	-	f1e7eac3	f1e7eac3
19	`Normal exit (4 jobs run)`	4	0c3b639c	eac7924f
20	`finished 0anacron`	oanacron	76690e70	a5803a8a

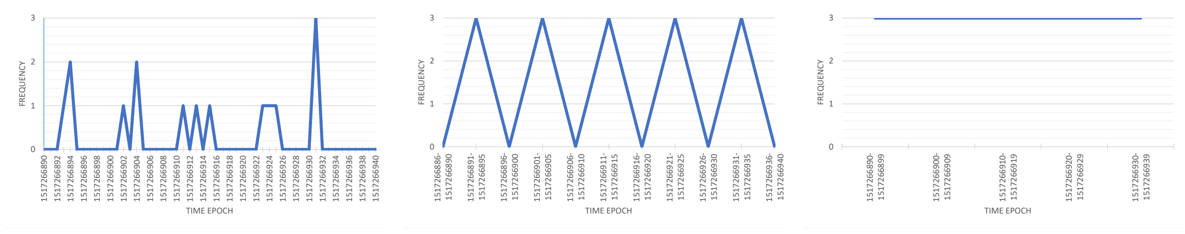
System logs are either periodic or event-driven. The periodic syslog entries reappear with a constant frequency (e.g., once every 10 minutes). The event-driven entries on the other hand, only appear after the occurrence of a certain event. The absence of periodic syslog entries, as well as changes in their frequency may indicate an abnormal behavior. Furthermore, differentiating between periodic and event-driven entries reduces the complexity of pattern detection among event-driven entries.

The automatic categorization method in this work employs the majority voting approach among a homogeneous neighborhood[16] of computing nodes. Since general purpose HPC systems such as Taurus are used by a large community of users, their events mostly follow a daily pattern (Figure 3.5 on page 29). Therefore, analyzing 24 hours of monitoring data (i.e. system logs) reveals both periodic and common event-driven patterns. The majority voting among homogeneous neighborhood of computing nodes mitigates potential inconsistencies and noises among monitoring data. Therefore, an accurate pattern for periodic and common event-driven entries can be extracted. Then any log entry can be automatically assigned to any of these two categories. A log entry which does not follow the known patterns can be the sign of a potential anomaly.

3.3.4 Data Discretization and Noise Mitigation

The proposed approach in this work is based on time-series analysis. Therefore, prior to analysis the continuous monitoring data collected from computing systems should be discretized. System logs are discrete time-series by nature, therefore, further discretization is not required. However, discrete binning of syslog entries can be used to mitigate noises. In this work a dynamic time binning is applied on syslog entries. Each bin contains the accumulated number of events occurred in a certain time window per node and per event class.

The dynamic binning significantly contributes to detection of periodic patterns in monitoring data. Many patterns are only visible using a certain (binning) bucket size. Larger bucket sizes may completely hide a pattern and smaller bucket sizes may reduce the significance of patterns. A sample binning of system logs using three different bucket sizes is shown in Figure 3.17. Only in Figure 3.17b a significant periodic pattern can be detected.



(a) Bucket size = 1 second: Patterns are not significant.

(b) Bucket size = 5 seconds: Significant periodic patterns.

(c) Bucket size = 10 seconds: No detectable patterns.

Figure 3.17: Significance of data binning bucket size on detectability of periodic patterns

To calculate the suitable bucket size, syslog entries of correlated nodes[17], collected in the period of one hour, are re-sampled using multiple bucket sizes. The size of each bucket varies from 60 to 3600 seconds with 60-second steps. Each bucket holds the average number of syslog entries generated during that period per second. The standard deviation of values in buckets with a similar size are calculated.

[16] Node vicinity defined in Section 4.2.1 further expands the concept of neighborhood homogeneity.
[17] Refer to Section 4.2.1 for more information.

The smallest bucket size that is (1) a local minimum in comparison to the nearest smaller and larger buckets, (2) is less than a certain threshold[18], and (3) projects a descending trend is chosen as the suitable bucket size. Figure 3.18 illustrates the final step of this calculation for Taurus. Green dots are potential suitable bucket sizes (local minimums). The horizontal yellow line indicates the threshold (standard deviation = 1) and the vertical red line represents the automatically chosen bucket size (600 seconds = 10 minutes) for data binning. This calculation will be repeated after each major change in syslog generation pattern.

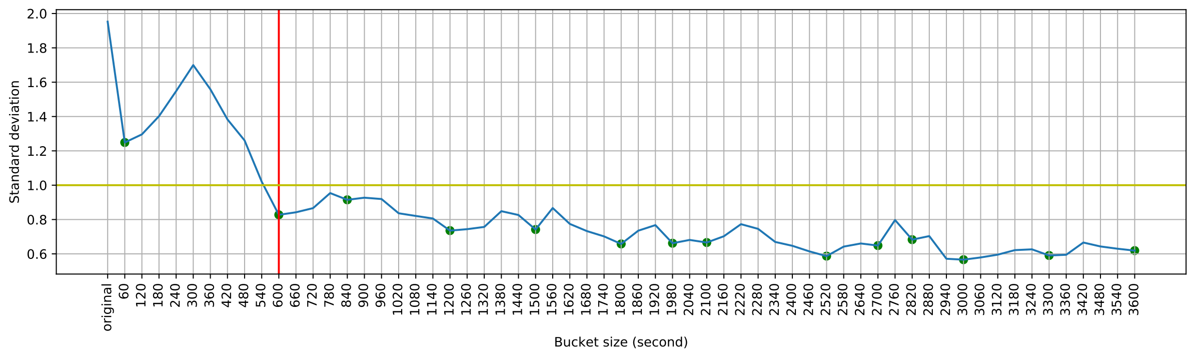


Figure 3.18: Calculation of suitable bucket size for data binning

Noise is an erroneous presence or absence of entries within the monitoring data. Syslogs are generated by applications on individual computing nodes, thus, any failure directly affects syslog entries via introducing random noises, interrupting log generation, or impeding log collection. Furthermore, even harmless errors may introduce random noises in syslog entries.

To identify the normal behavior of computing systems, it is necessary to remove the random noises. Beside software and hardware failures which may inject random noises into the monitoring data, other actions such as software updates, administration activities, and system maintenance can also introduce noises. In addition, most production HPC systems are used by various groups of users and for different applications. Therefore, existence of random noises in monitoring data is highly plausible due to human errors and applications misbehavior [57, 5]. Part of these noises can be removed via discrete binning of the monitoring data.However, an extreme discrete binning can decrease the accuracy of anomaly detection by decreasing the monitoring data precision and hiding the existing patterns.

This work utilizes the *neighborhood homogeneity* of HPC systems to mitigate random noises. Computing nodes in HPC systems are divided into smaller subsets such as chassis or racks. Majority of these small subsets consist of homogeneous computing nodes which share various physical resources such as power supply, cooling system, and network infrastructure. Homogeneous computing nodes which are physically collocated (adjacent) and share similar physical resources tend to project similar behaviors [250]. Therefore, in a homogeneous subset of computing nodes, common behavior of the majority can be considered as the normal behavior in that particular subset.

[18]The threshold is system dependent and should be adjusted once for each computing system.

Figure 3.19 shows the extraction of common node behavior from noisy syslog entries on Taurus in a subset consisting of 8 homogeneous computing nodes. Colored cells mark the occurrences of event `a5803a8a` (event pattern) on 8 adjacent nodes during 32 minutes. The bucket size is 60 seconds. The bottom row indicates the normal pattern of event occurrences, extracted via majority voting among the 8 computing nodes. Events are placed in each bin according to their relative time passed since midnight. Further time synchronization is not required.

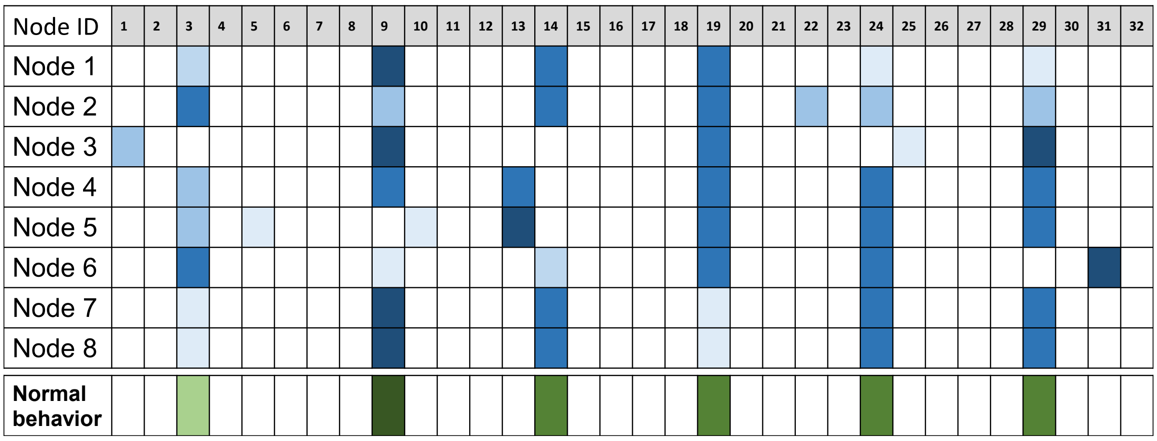


Figure 3.19: A sample of normal behavior extraction using majority voting. Occurrence pattern of one event class (`a5803a8a`) on 8 nodes, in 32 minutes with bucket size of 60 seconds. The bottom row, shown in green, holds the result of majority voting on all 8 nodes. The darker shades indicate higher number of entries. Since each event class has its own pattern, to emphasis on the important details only one event class is shown.

Outliers are valid events which distant from the norm. Outliers can impede correct analysis of systems behavior. However, in contrast to noises, outliers are part of the systems behavior, thus, they should not be removed. Outliers are not always indicators of abnormal behaviors. It is worth to emphasize that the goal of this stage is extracting the pattern of normal (healthy) system behavior. Therefore, standardizing the data range (scaling) is sufficient to omit the negative effect of outliers.

Considering the noise mitigation approach shown in Figure 3.19, when the majority of computing nodes project abnormal behavior, the extracted behavior pattern will be incorrect. However, analyzing Taurus behavior revealed that except major system failures, that affect the majority of computing nodes, utilizing the *neighborhood homogeneity* and majority voting extracts the common event patterns correctly. Nevertheless, the system log entries collected during major system failures must be excluded from the training data (ground truth) to prevent unexpected results.

3.3.5 Cleansed Taurus System Log Dataset

Syslog entries collected from 2046 Taurus nodes were processed using PαRS and encoded into their respective anonymized format (event pattern). Out of the total number of 3.26 billion collected syslog entries in the year 2017, 10.8 million entries were marked as invalid entries and were removed from dataset. $1,488$ unique event patterns were extracted from the remaining 3.25 billion syslog entries forming the cleansed Taurus Syslog dataset; the `TaurusCleansed`.

Each record of `TaurusCleansed` consists of five fields: timestamp, node ID, facility, severity and the message hash key (event pattern). A sample of `TaurusCleansed` records is shown in Example 3.4. It is important to note that this dataset is only created for the purpose of this work. In production environment all pre-processing steps are performed online on the stream of incoming monitoring data.

Example 3.4: Sample of `TaurusCleansed` records

```
timestamp   node  facility Severity hash key
----------  ----  -------- -------- --------
1488424393  5103  4        4        23666bbc
1488424483  5101  3        6        23666bbc
1488424501  5104  0        3        85f5c18b
1488424573  5102  3        4        23666bbc
1488424657  5101  4        6        760c5208
1488424657  5103  4        6        bba3d47c
1488424657  5102  4        5        f9cfa0b9
```

3.4 Marking Potential Failures

Failures can be observed and analyzed in different granularities, from a single transistor to the entire HPC system. Nodes are the smallest units in HPC systems which have a fully functional computational stack, yet are independent and can be added to or removed from HPC systems with minimum side-effects [6]. Therefore, the granularity of failure detection in this work is set at the node level[19].

Due to various technical reasons, a complete list of all node failures on Taurus for the period of this work is not available. Therefore, this section retrieves a complete list of Taurus node failures as *ground truth* for further analysis. The available collection of Taurus system log entries contains data gaps, explained in Section 3.2.1, which are mainly incurred due to interruption of data collection mechanism. These gaps do not necessarily indicate node outages. On the other hand, some failures, such as failures caused by power outage, leave no traces in syslog entries. Therefore, one of the first challenges is to identify real node failures on Taurus using the existing imperfect monitoring data [251].

Node failures in computing systems can be divided into two main categories according to their causes. Some failures are occurring during the normal operation of the HPC system and are caused by internal factors such as software and hardware errors or racing conditions. While other failures are occurring due to external causes such as power outages and human errors. Analyzing the impact of external causes on node failures requires additional

[19] *Node failure* and *failure* are used interchangeably in the rest of this work.

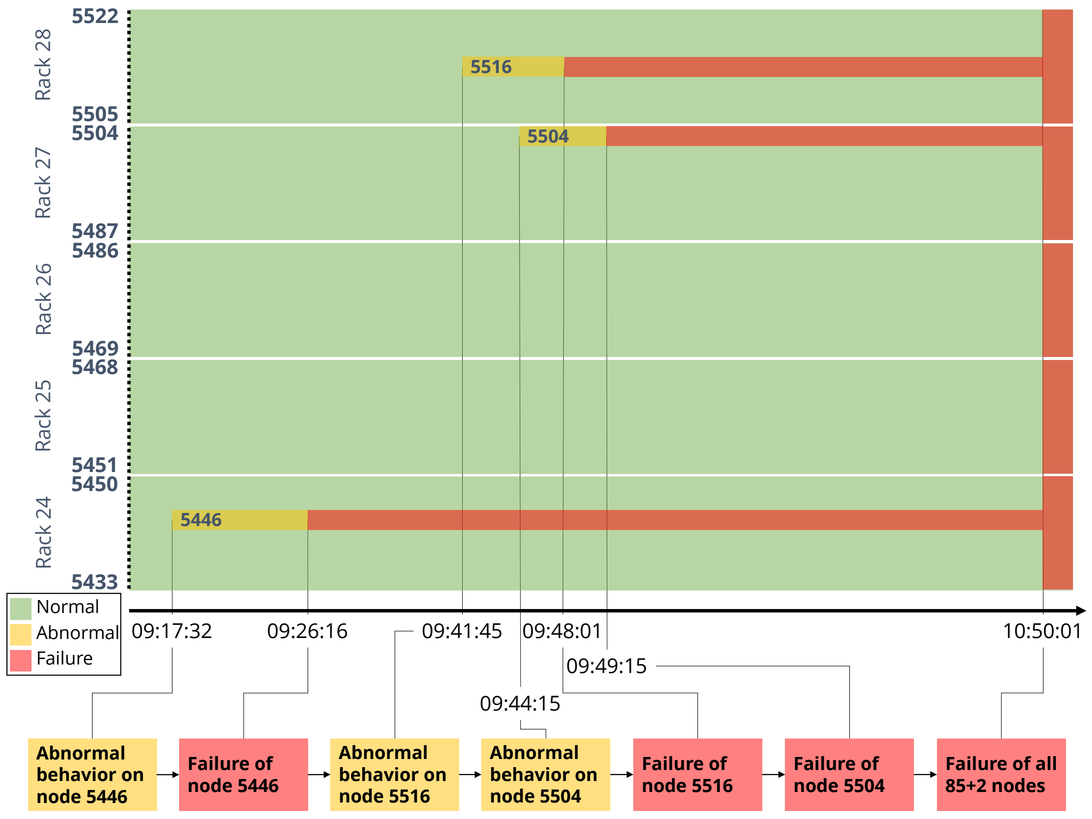


Figure 3.20: Timeline of the major failure on March 16th

information regarding the external factors, such as detailed information about the behavior of the power supplier company, which is currently not available. Furthermore, multiple independent protection mechanisms protect Taurus against severe damages such as overheating. Therefore, beside the human interactions and external causes of failure such as power outage, the protection mechanism is also the cause of certain outages. As an example, the large failure shown in Figure 3.20 was caused by the overheating protection mechanism to prevent further damages to computing nodes.

In this particular example 3 nodes (out of 90) were able to detect and report the unusual temperature increase before activation of the overheating protection mechanism. Therefore, it was possible to detect the reason behind this major failure via analyzing syslog entries. However, in most cases the protection mechanism detects the unusual behavior earlier than computing nodes and will be activated. Thus, no footprints of the failure's cause exist in Taurus syslog entries. In such cases, the activity logs of protection mechanisms should be included. Similar analysis methods, as used for syslog analysis in this work, can be applied to any other form of activity logs. However, to improve the generality of the proposed approach, the protection mechanism is considered an external cause similar to power outages and human interactions. Therefore, in this work the focus is on the first group of node failures which are referred to as *regular failures* and are caused by internal factors.

The first step to identify such failures is to retrieve all node outages and afterwards distinguish regular failures from those which may happen as a result of external factors such as maintenance, human errors, and so forth. Figure 3.21 illustrates the main root causes of Taurus node outages in 2017.

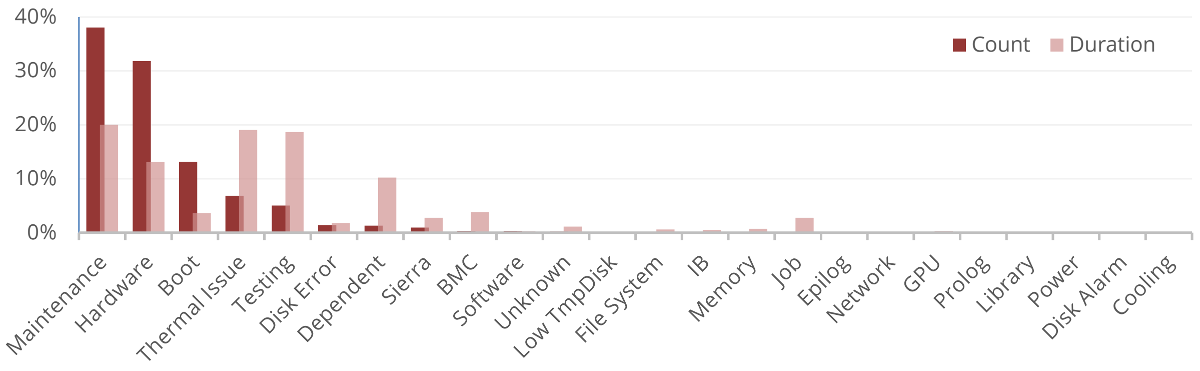


Figure 3.21: Main root causes of Taurus node outages in 2017

Taurus computing nodes generate and send syslog entries to a central log collector which stores them for future analysis. This passive log collection mechanism is chosen since it imposes no additional overheads, and is applicable to all HPC systems. However, this approach leads to a common problem shared by all remote-access systems: it is not possible to truly indicate whether a node is down, too busy to respond, or simply lost its connectivity. Therefore, the failure identification process becomes more challenging. Due to employment of the passive log collection mechanism, a node outage can be confidently detected only when a direct indication in form of an entry in monitoring data (e.g., syslog) is generated by the failing node and correctly received and stored by the central log collector, e.g., "`Kernel panic - not syncing: Fatal Exception`." However, in many cases a node outage leaves no direct indication in system logs. A workaround is to assume the absence of log entries, for longer than a certain time interval, an indication of a potential outage. Nonetheless, this assumption is not accurate. For various reasons the flow of system log entries from computing nodes to the central log collector might be interrupted or delayed, which appears as log absence interval and therefore, might be interpreted as outage, although, the computing nodes are functional. Also, in many cases immediately after the occurrence of an outage, protection mechanisms recover the node. In the both later scenarios, an active node probing approach may also fail to detect all node outages correctly. Therefore, in this work two overlapping methods of (1) analyzing syslog-ng internal metrics and (2) back tracking and cross-checking are used to retrieve Taurus node outages and provide a reliable set of ground truth for the next steps.

The syslog-ng daemons periodically report statistics of collected system logs[20]. On Taurus, under current configuration, once per hour syslog-ng's internal metrics are recorded as a *log statistics* entry. These metrics are mostly based on various message counters such as the number of messages that successfully reached their destination driver (`processed`), the number of dropped messages (`dropped`), and the number of messages passed to the message queue of the destination driver that are waiting to be sent to the destination (`queued`). To retrieve the potential time frame of a node failure, three syslog-ng metrics

[20]syslog-ng.com/technical-documents/doc/syslog-ng-open-source-edition/3.16/administration-guide/80

are used: timestamp, stamp and processed. The timestamp is the UNIX timestamp of the recorded *log statistics* entry, while the stamp is the UNIX timestamp of the last message sent to the destination.

Two consecutive Syslog-ng log statistics entry are shown in Example 3.5. In the first entry the timestamp is 1485965346, the stamp is 1485961746, and the processed counter is 29954. The unused metrics are removed to increase readability.

Example 3.5: Syslog-ng log statistics entry

```
1485965346 [...] Log statistics; processed='src.internal(s_sys#1)=29954',
    stamp='src.internal(s_sys#1)=1485961746', [...]
1485968946 [...] Log statistics; processed='src.internal(s_sys#1)=29955',
    stamp='src.internal(s_sys#1)=1485965346', [...]
```

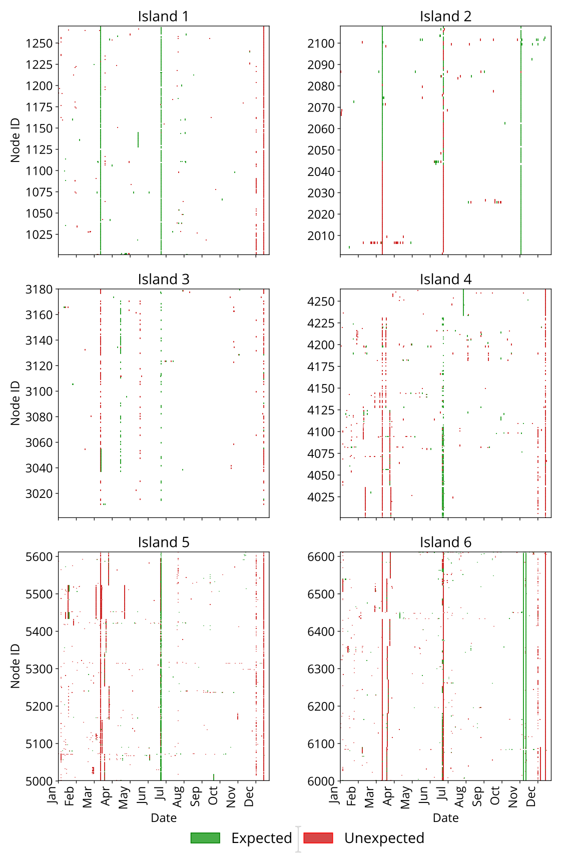
To retrieve the potential time frame of a node instability on Taurus, two following conditions are checked. First, the time difference between timestamp and stamp of each entry on a healthy Taurus node should always remain equal to 3600 seconds (1 hour). Second, the processed counter for s_sys in each entry must be incremented by exactly 1 unit in comparison to the previous entry. For both entries shown in Example 3.5, both conditions hold true. These two conditions for the entire Taurus syslog dataset were controlled. All intervals of node instability were retrieved and marked. It is expected that each node had experienced at least one failure during its instability interval.

Through analyzing Taurus system logs it has become experimentally evident that all nodes during a healthy boot, leave similar footprints in their syslog entries. If a node fails to generate the expected footprint at boot time, it is an indication of a faulty boot process and thus the node will be either automatically rebooted again or fails shortly afterwards. The higher frequency of log generation at boot time in comparison to the normal operation time is another indicator of a boot event, which can be used to identify a boot process as well as, distinguishing between healthy and problematic boot processes.

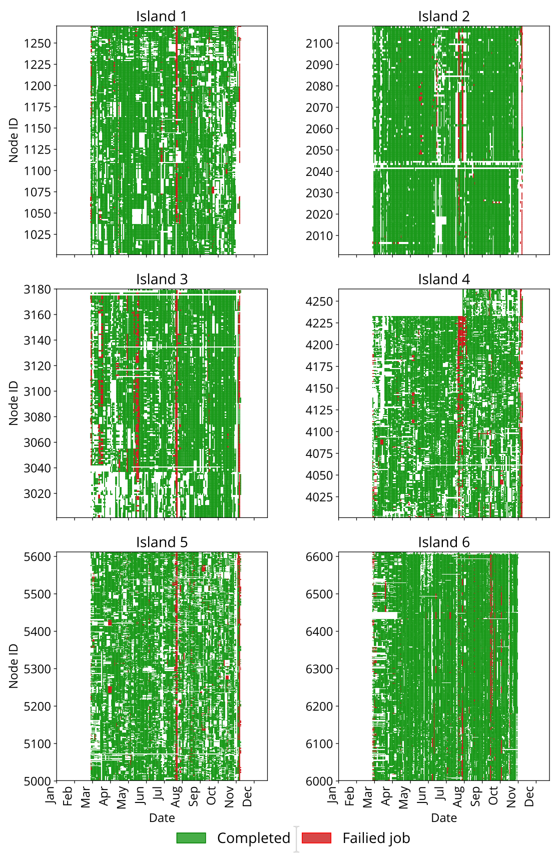
The proposed node outage retrieval method also searches for the footprint of boot events among system log entries. Upon detection of a boot event, syslog entries are backtracked to extract the last syslog entry before the boot event. The timestamp of last syslog entry before the boot event is considered as the point of outage[21]. Using the proposed method, all node outages will be identified. The only exception is when a node fails and has no further successful boot process. In such cases, comparing the timestamp of the last available syslog entry with the current time (31-12-2017 23:59:59 in this work) reveals the missing outage. Figure 3.22a illustrates all detected boot events on Taurus within the period of one year. Unexpected events (shown in red) indicate the absence of information in system logs which might be signs of potential crashes. Expected events indicate scheduled boot events which are due to maintenance or intentionally caused via protection mechanisms[22].

Retrieved node outages are then compared against the other available data sources

[21]The last entry might also be a shutdown or reboot command.
[22]In such cases, rather than a sudden power off, a shutdown command is executed.



(a) Node outages retrieved via syslog analysis.



(b) Slurm job status report

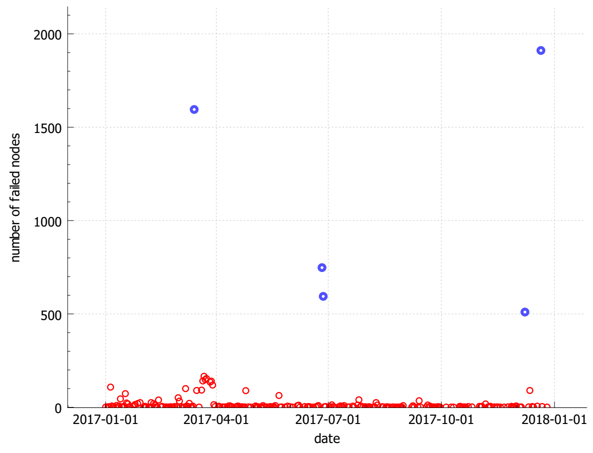
Figure 3.22: Node outages and job reports on Taurus. Intervals of unavailability of job reports do not necessarily specify node outages.

described in Table 3.2 on page 32. When a node outage is happened out of the scheduled maintenance period, and no job could be accomplished on that particular node at the time of the detected outage, the outage is marked as a regular failure. As Figure 3.22b illustrates, it is common that certain jobs on a specific node fail although other jobs on the same node are accomplished simultaneously. The red dots indicate jobs that are failed due to node failures. Node outages that are recorded in the outages database, which monitors the availability of the HPC system from users perspective, are also considered as regular failures.

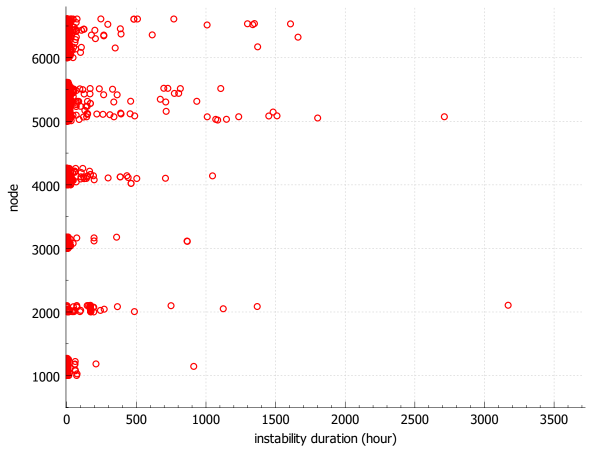
Using the procedures introduced in this section, a set of ground truth is built for Taurus failure statistics. This dataset together with the information obtained in previous steps are stored in an SQLite database, namely `taurusMETA`. The schema of `taurusMETA` database is shown in Figure C.1 on page 145.

In total $11,463$ intervals of instability were detected via analyzing Syslog-ng internal metrics stored in syslog entries and back tracking of boot events, out of which, $3,332$ incidents were either initiated or interrupted by system moderators. Figure 3.23 shows an overview of Taurus instability intervals. Among instability intervals, Figure 3.23a indicates five system-wide instabilities. Further analysis revealed that on March 14th, as well as on June 26th and 27th, system-wide maintenance was the root cause of nodes instability.

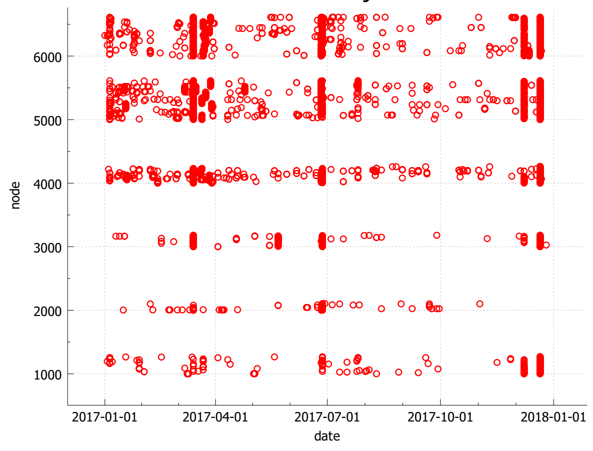
In addition, on 8th and 21st of December 2017, the system-wide failures were caused by human error and filesystem bug respectively. Therefore, all incidents related to these five days ($5,358$ incidents) were removed from the potential Taurus node failures dataset. Figure 3.23d shows the impact of these five incidents on individual Taurus nodes. The remaining $2,773$ incidents (node outages) are considered as *potential node failures*. As shown in Figure 3.23b, the majority of nodes were recovered to stable conditions in less than five hours. Out of $2,046$ Taurus computing nodes, 29 nodes did not have any useful failure information[23].



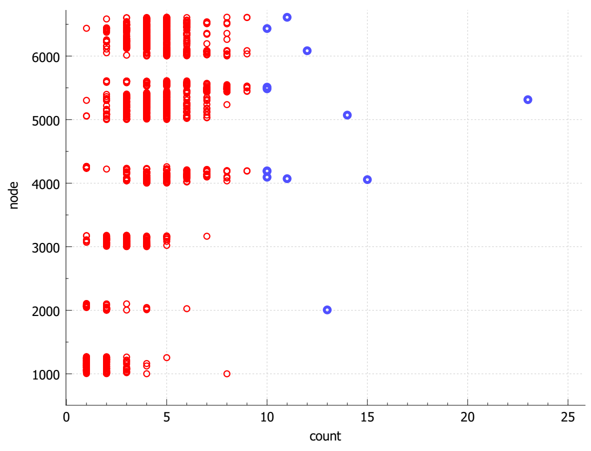
(a) Total number of node instabilities per day. Large number of unstable nodes on 5 days (marked in blue) indicate system wide failures



(b) Duration of each instabilities per node. Some nodes were powered down for several months



(c) Distribution of node instabilities.



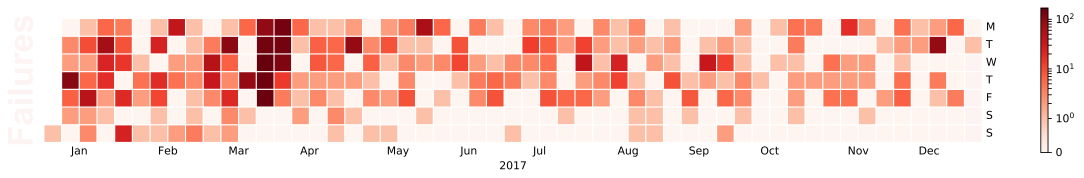
(d) Number of instabilities per node. 13 nodes (marked in blue) experienced 10 or more incidents

Figure 3.23: Taurus node instability intervals in year 2017

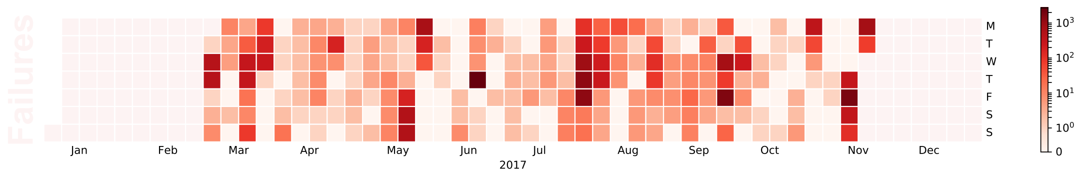
Figure 3.24 illustrates the number of potential Taurus node failures per day in 2017. The pattern of node failures over the year 2017, shown in Figure 3.24a, has no linear correlation to the pattern of job failures shown in Figure 3.24b, implying that the number of jobs running on some nodes are extremely larger than the others. However, the low number

[23]The 29 nodes are: 1005, 1125, 1149, 1160, and 1172 from Island 1, 2045-2047, 2050, 2055, 2063, 2069-2070, 2078, 2081-2082, 2085, 2087, 2089-2091, 2097, 2098 and 2102-2104 from Island 2, and 3073 and 3175 from Island 3.

of node failures on weekends, as shown in Figure 3.24a, implies a direct relation between user activities and node failures. Figure 3.5 on page 29 indicates similar correlations with a different granularity.



(a) Potential node failures



(b) Job failures caused by node failures

Figure 3.24: Number of potential failures per day in year 2017

The taurusMETA dataset provides a list of all regular failures[24] of Taurus HPC cluster. Figure 3.25 shows a node-level overview of potential Taurus node failures.

During the year 2017, 93.96% of all jobs submitted to Taurus were executed on single nodes, 4.43% on 2 to 4 nodes, 1.33% on 5 to 16 nodes, and 0.27% on 16 to 512 nodes. In total 1.41% of jobs that were running on multiple nodes, as reported by Slurm, failed due to node failures. Out of which, 10.77% of job failures were directly caused by node failures. Resulting the interruption of 1857 tasks on various nodes. The distribution of accomplished and failed jobs, as reported by Slurm, is shown in Figure 3.26.

No significant pattern or correlation can be observed between node and job failures[25]. However, this might be due to the low number of multiple-node failures on Taurus in the year 2017. The independence of job failures and node failures can be better illustrated via a side by side comparison of daily pattern of job accomplishments versus job failures. As shown in Figure 3.27, although there are days with multiple job failures, the number of job accomplishments remains almost identical throughout the year. Furthermore, even those job failures that are reportedly caused by node failures do not have any direct correlation with the number of node failures.

Analyzing taurusMETA provides better understanding of patterns in Taurus behavior. The frequency of all potential failures according to the duration of each failure (in hours) is listed in Figure 3.28. The majority of failures last longer than 1 hour and recover in less than 6 hours. However, there are three exceptions: (a) 91 failures with the length of 22 hours, (b) 43 failures with the length of 28 hours and (c) 47 failures with the length of 48 hours. Considering the large differences between these peak values and their neighboring values, it is concluded that failures in each group (a, b, and c) are strongly correlated.

[24]Failures that are caused by internal factors during the normal production interval.
[25]Two other large-scale studies on IBM BlueGene/Q Mira and Blue Waters also conclude that 99.4% and 98.5% of job failures were due to user behaviors, respectively [5, 252].

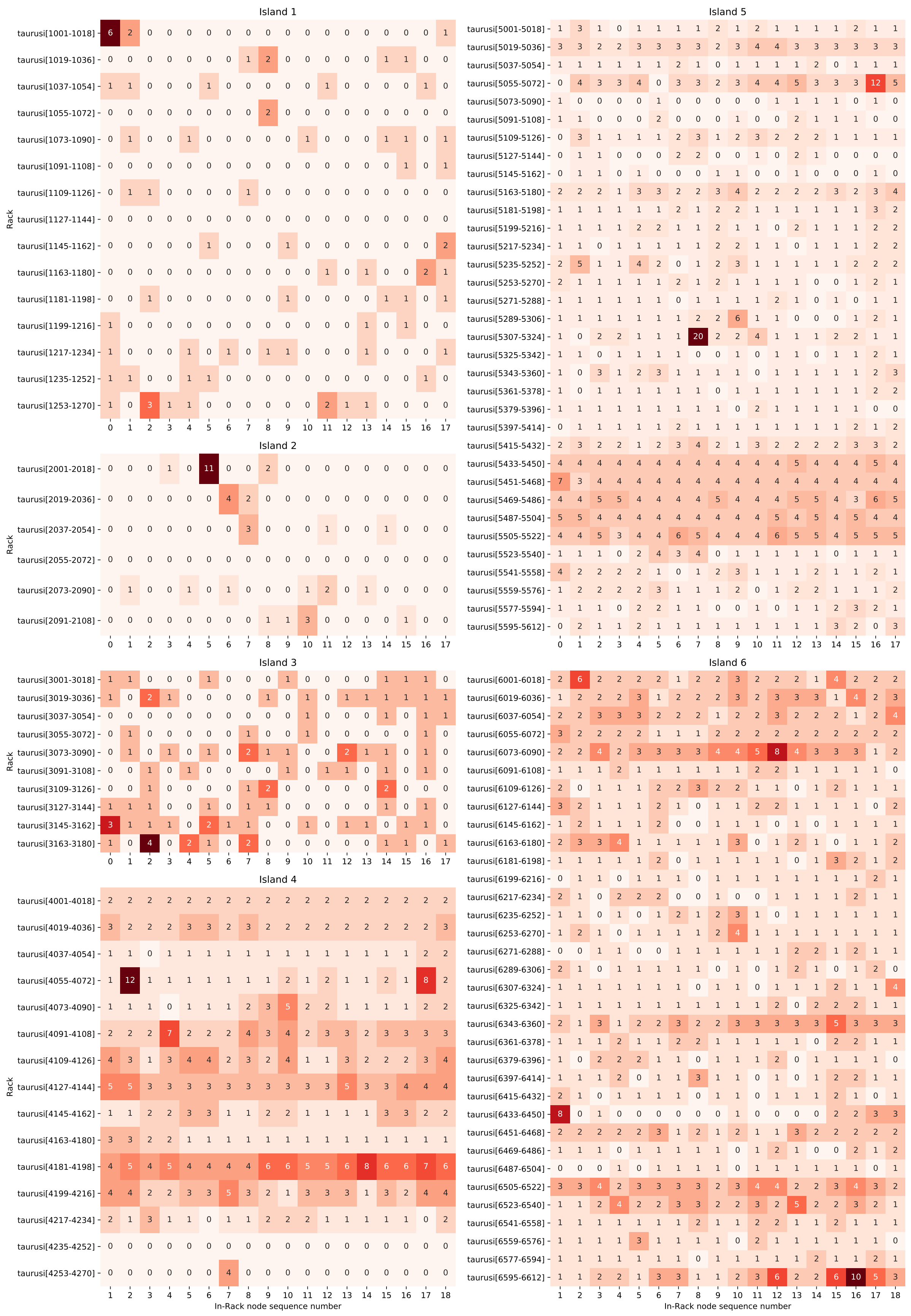
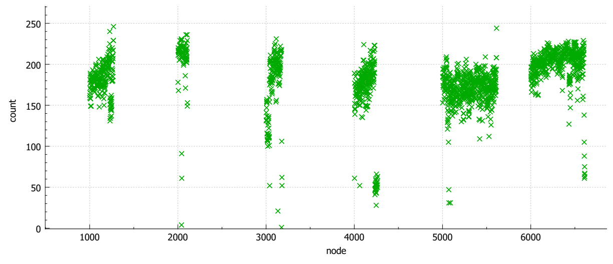
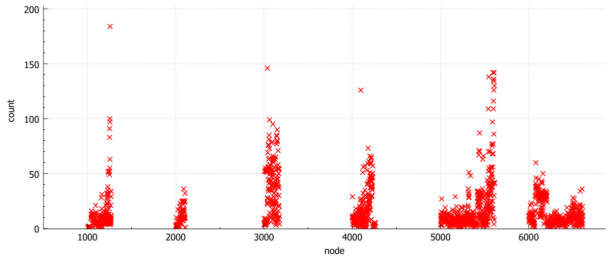


Figure 3.25: Potential node failures occurred in 2017. Each cell represents a single node and each row represents a rack. The total number of potential node failures is shown in each cell.

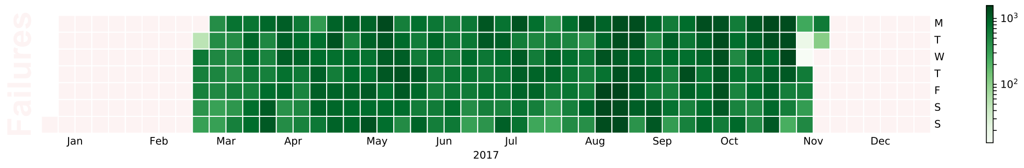


(a) Total number of jobs that has terminated all its processes on all nodes with exit code of zero

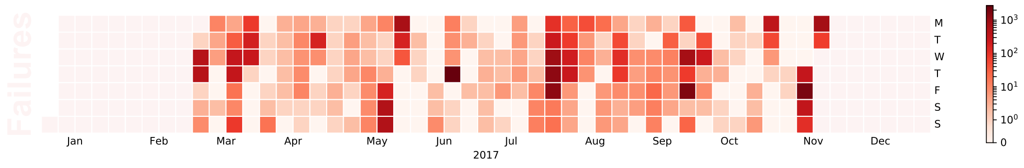


(b) Total number of jobs terminated due to failure of one or more allocated nodes

Figure 3.26: Jobs status reported by Slurm per node in year 2017



(a) Job accomplishments



(b) Job failures caused by node failures

Figure 3.27: Number of jobs accomplished or failed per day in year 2017

Analyzing the failures in group (a), confirms the correlation hypothesis. 85 out of 91 failures with the duration of 22 hours were simultaneous failures occurred on March 16th. Analyzing the syslog entries of these 85 nodes does not provide any information regarding the cause of this failure. Looking at the list of all node failures on March 16th reveals 5 additional node failures on the same day with different failure duration. More interestingly all of these 90 failures (85 + 5) were occurred on five neighboring racks[26]. Fortunately, 3 of the newly discovered node failures had traces of the cause of this major failure. Analyzing the syslog entries of node 5446, 5504 and 5516 revealed that this major failure was caused by malfunctioning of the central cooling system in those five neighboring racks which consequently activated the overheating protection mechanism and resulted in the sudden power down of all five racks. The timeline of significant events from the first detectable anomaly on node 5446 until the failure of all 90 nodes is shown in Figure 3.20 on page 52. Early detection of the abnormal behavior at 09:17 could have provided more than 1.5 hours lead time to prevent the major failure at 10:50.

Analyzing the failures in group (b) and (c) confirms the existence of similar form of temporal correlations. Out of the 43 failures with the duration of 28 hours in group (b), the 35 failures that occurred on February 13th were most likely caused by network problems. Out of 47 failures with the duration of 48 hours in group (c), the 46 failures that occurred on January 17th were also caused by overheating protection mechanism.

[26] Racks: 24, 25, 26, 27, and 28

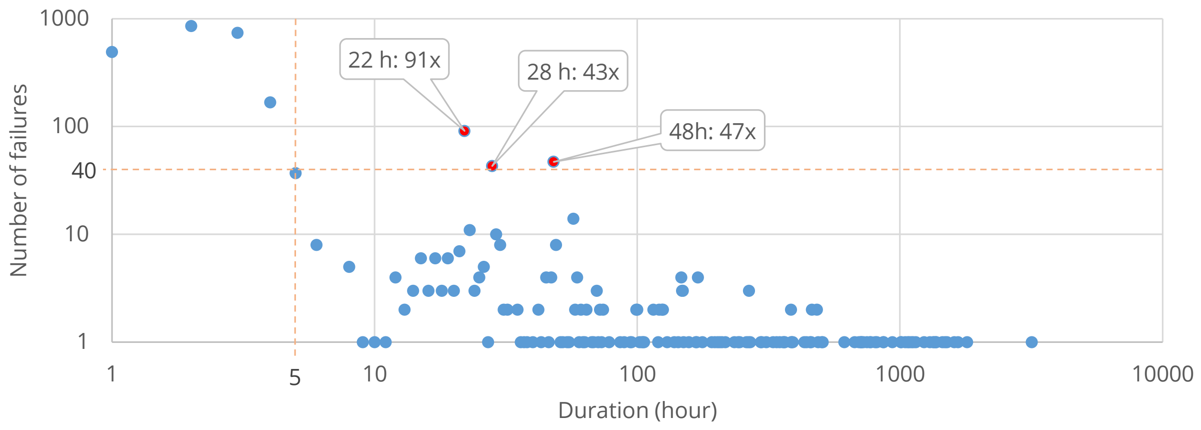


Figure 3.28: Number of potential node failures classified based on the duration of each failure. More than 81% of failures were recovered in less than 5 hours.

The reasons behind each of these 3 major failures, namely, malfunctioning of cooling system and problem in network connectivity, reveals the unexpectedly long failure duration as well as the large number of simultaneous node failures. Both cases which were caused by malfunctioning of cooling system could have been prevented with few hours lead time, providing enough time to prevent the upcoming failures. Therefore, it can be concluded that certain failures are predictable. However, to evaluate the generality of this conclusion, first the null hypothesis should be rejected.

4 Failure Prediction

Chapter 4 describes the methodology and tools which were used and developed in this work to analyze computing systems behavior. Chapter 4 covers the orange building blocks shown in Figure 4.1. The output of this chapter is a comprehensive failure predictor for HPC systems. Behavior analysis in this work are conducted through 3 phases: failure correlation, pattern detection, and anomaly detection. Methods of the first phase manually analyze HPC system monitoring data to identify potential correlations among failures. During the second phase semi-automatic methods detect potential patterns in HPC system monitoring data prior failure events. The third phase attempts to automate the entire failure prediction process.

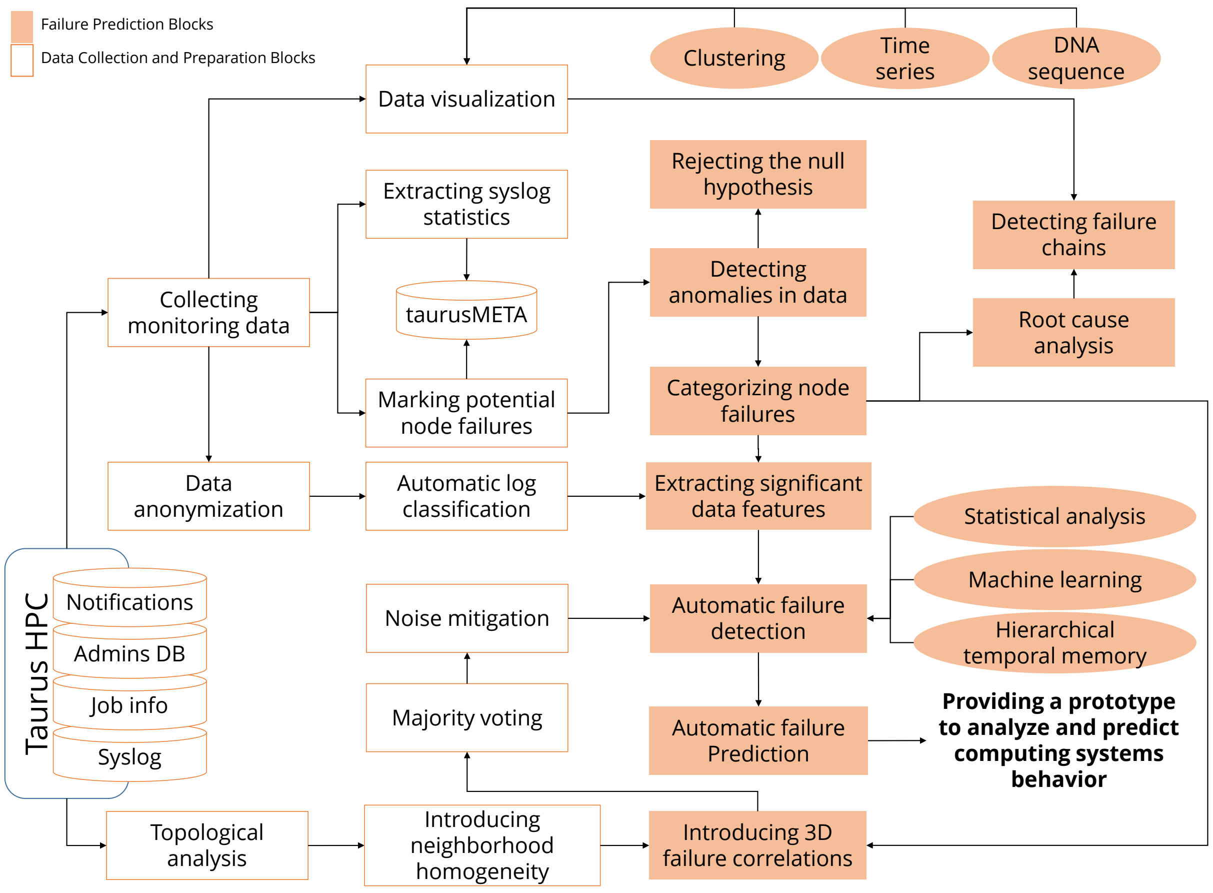

Failure Prediction Blocks
Data Collection and Preparation Blocks
Clustering
Time series
DNA sequence
Data visualization
Rejecting the null hypothesis
Extracting syslog statistics
Detecting failure chains
Detecting anomalies in data
Collecting monitoring data
taurusMETA
Root cause analysis
Marking potential node failures
Categorizing node failures
Data anonymization
Automatic log classification
Extracting significant data features
Statistical analysis
Machine learning
Hierarchical temporal memory
Taurus HPC
Notifications
Admins DB
Job info
Syslog
Noise mitigation
Automatic failure detection
Majority voting
Automatic failure Prediction
Providing a prototype to analyze and predict computing systems behavior
Topological analysis
Introducing neighborhood homogeneity
Introducing 3D failure correlations

Figure 4.1: Major building blocks and the workflow of *Jam–e Jam*. Building blocks shown in orange are covered in Chapter 4.

4.1 Null Hypothesis

This work uses syslog entries as the main monitoring data for detecting abnormal behaviors in HPC systems. Prior to practical syslog analysis, the fitness of syslog entries for behavioral analysis should be proved. It should become evident that syslog entries, that are collected before a node failure, project different patterns than the patterns observed during the normal operation of HPC systems. To achieve this goal, the HPC system monitoring timeline is divided into 4 sections of pre-event time (PET), event time (ET), post-event time (PoET), and inter-event time (IET) around each identified node failure (ground truth). A schematic illustration of the monitoring timeline is shown in Figure 4.2.

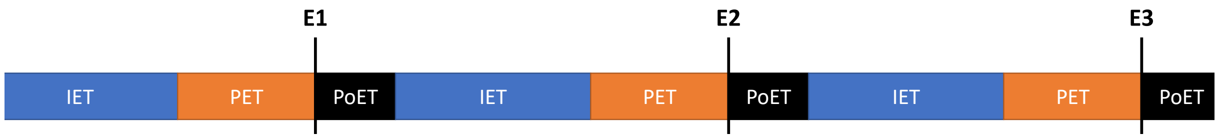


Figure 4.2: Division of events timeline into inter-event time, pre-event time, event time, and post-event time.

A node failure may last from seconds to hours or days, ET refers to the entire duration of a node failure[1]. PET denotes a time window before the failure occurrence. PoET marks a short time window after node recovery. During the PoET interval, the system is unstable and the collected syslog entries are not reliable. IET on the other hand, specifies a time window that has no overlap with PET, ET, and PoET intervals. During the IET intervals, the system is stable and operates normally.

The null hypothesis assumes that the patterns observed before node failures (PET) are similar to patterns observed during the normal operation (IET). If this hypothesis holds true, it is concluded that there is no correlation among syslog frequency and the system status thus, the syslog frequency is not a useful measure to predict node failures. To reject the null hypothesis, it must be shown that the distribution of certain data features, prior to a failure occurrence, significantly changes. For this purpose the distribution of syslog entries' metadata, namely `facility` and `severity`, is analyzed. The `facility` and `severity` values of `Taurusi1235` syslog entries for a duration of 24 hours before the failure are shown in Figure 4.3. Horizontal axis in Figure 4.3 shows `facility-severity` pairs such as `usr-err` that stands for syslog entries produced by a *user* facility with the *error* severity level[2]. Sudden changes in the appearance of syslog entries regardless of the facility name and severity level may be the indication of an upcoming failure. In Figure 4.3 four sudden changes are marked with black circles, namely `kern-err`, `kern-warning`, `kern-info`, and `authpriv-info`. Out of 2,773 potential failures on Taurus in 2017, 127 failures had overlapping pre-/post-event intervals. Due to these overlaps the PET and PoET of respective failures cannot be correctly calculated. Therefore, these 127 failures were excluded from the analysis in this section.

[1]E1, E2, and E3, in Figure 4.2, are examples of ET. Since no syslog entries are available during event times, ETs are shown as solid vertical lines rather than time intervals.
[2]Table 2.2 on page 9 provides a complete list of syslog facilities and severity levels.

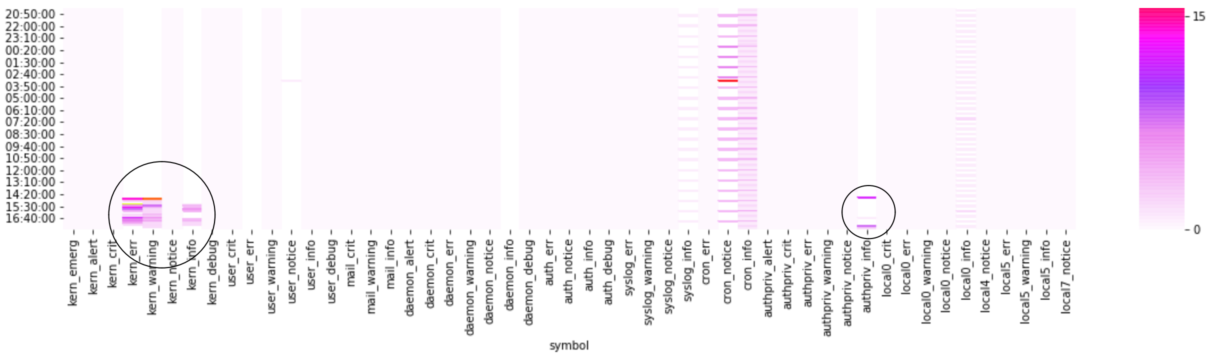


Figure 4.3: A sample of sudden increase in the frequency (marked with black circles) of certain syslog entries shortly before the failure of node Taurusi1235 at 17:42:22.

Upon observation of a sudden change in the node behavior, node enters an instability interval. A useful prediction should provide enough lead time that allows required protective measures to be taken. Therefore, the feasibility of performing node failure predictions with useful lead time should also be verified. To calculate the potential lead time, the instability period before each failure, using the method introduced in Section 3.4, was calculated. 97% of the failures had an instability period of 60 minutes or shorter and the respective nodes were behaving normal (at least) during the last 24 hours before each failure. The distribution of instability periods for these 2,536 failures are shown in Figure 4.4. The lead time of failure predictions at most can be equal to the duration of node instability interval. According to Figure 4.4 it is concluded that for majority of failures, upon an early detection of node's instability, a node failure prediction with useful lead time can be made.

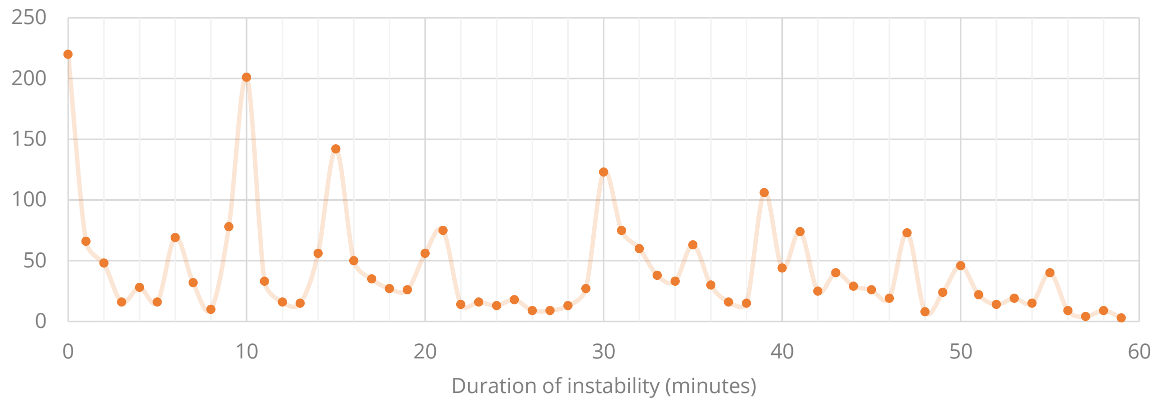
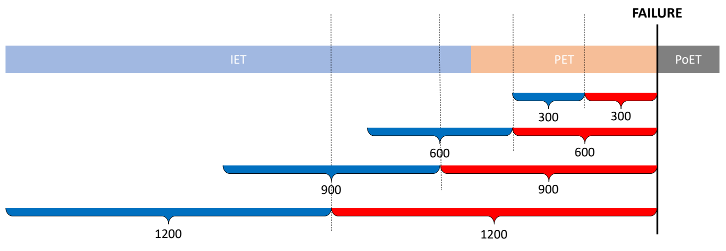
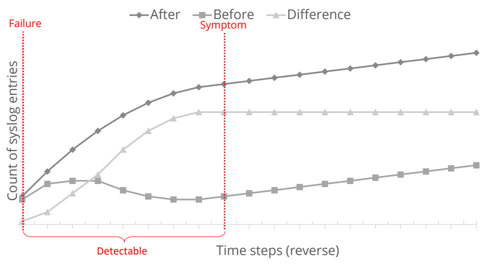


Figure 4.4: Distribution of the nodes instability interval before each failure. Out of 2,624 total failures, 88 failures with instability periods of longer than 1 hour are not shown in this figure. Instability duration of 0 indicates failures that occurred immediately after a system boot.

To extract the effective features that reveal abnormal behavior, the frequency of symbol occurrences for all $2,536$ failures has been calculated. For this calculation the base distance of 300 seconds (five minutes) is chosen. In each step the new distance is calculated as the summation of previous distance and the base distance. Therefore, with the base distance of 300 seconds, the first four distances will be 300, 600, 900 and 1200 seconds. In each step the starting-point of collection is the point of failure minus the calculated distance. This starting-point is assumed to be the point of instability. The total number of symbols before and after the starting-point within an equal distance are counted. Figure 4.5a shows the first four steps of counting symbols before a known node failure. In normal situations, the ratio of counted symbols within the intervals before and after the moving starting-point should remain equal. However, in an abnormal situation the ratio is expected to change as shown in Figure 4.5b.



(a) Sampling of syslog frequency



(b) Expected ratio of syslog frequency

Figure 4.5: Sampling and the expected ratio of syslog entry symbols to assess the null hypothesis. A continuous change in the difference of syslog entries collected before and after a symptom indicates a potential problem.

The frequency of symbols counted in various distances from the point of failures is shown in Figure 4.6. Among the available parameters and based on the amount of changes between these values the `user-err` is chosen to assess the null hypothesis.

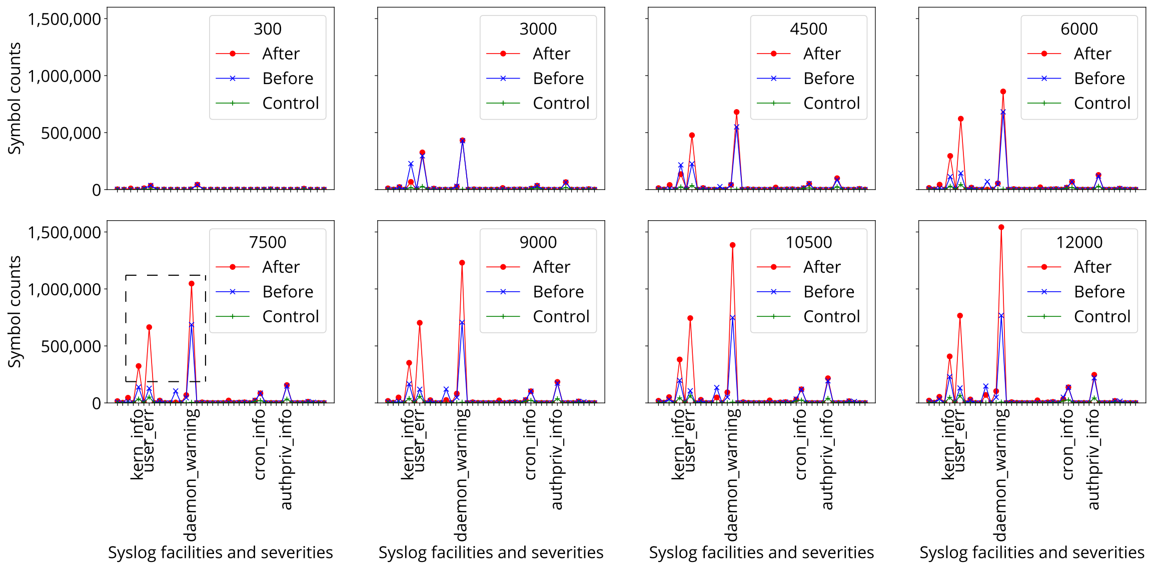


Figure 4.6: The frequency of syslog entries metadata of all Taurus nodes prior to occurrence of a failure. Control is a semi-random generated series of values representing a normal distribution.

The frequency of `user-err` symbols before and after the moving starting-point is shown in Figure 4.7. The ratio shown in Figure 4.7 closely follows the expected ratio shown in Figure 4.5b. The frequency of syslog entries closer to the point of failure significantly differs from the normal operation. This significant deviation rejects the null hypothesis. Based on this observation the fitness of syslog entries for behavioral analysis is concluded.

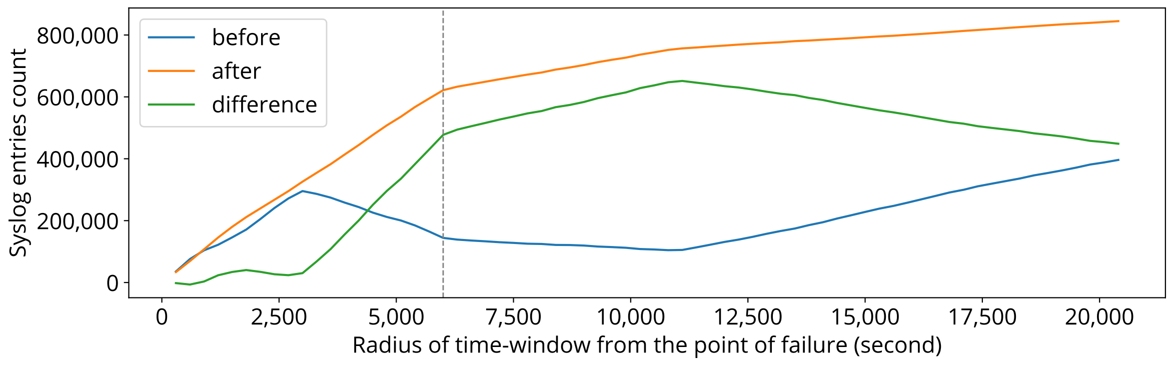


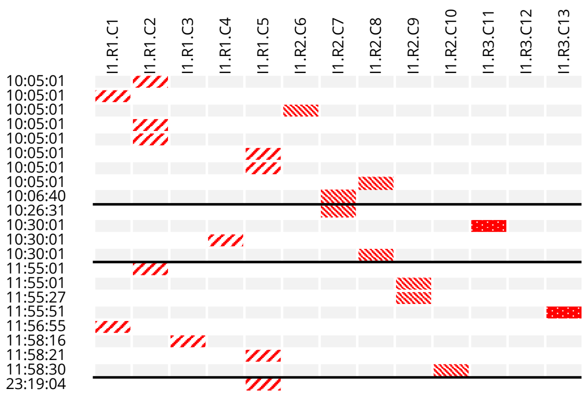
Figure 4.7: The frequency of syslog entries metadata (`usr-err`) prior to occurrence of failures.

4.2 Failure Correlation

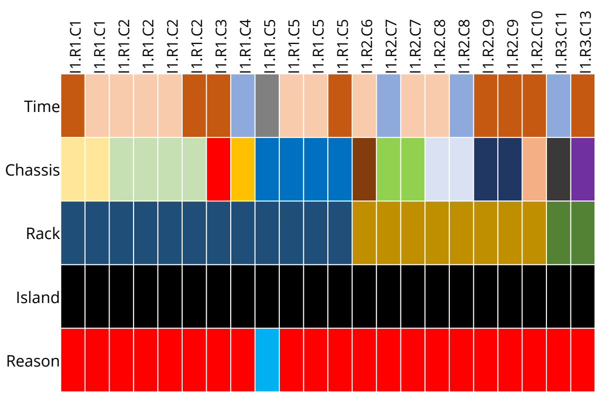
Early detection of failure chains prevents their propagation. Correlations among node failures, leading to formation of failure chains, derive along three dimensions: (1) *Temporal* denotes cases when the time interval between consecutive failures falls below a certain threshold. (2) *Spatial* denotes cases when the failed nodes share a physical resource (e.g., chassis). (3) *Logical* denotes cases when the failed nodes share a logical resource (e.g., batch job).

Among the three dimensions of failure correlation (time, space, and logic), the most relevant one is the logical dimension as it can help to prevent re-occurrence of the same failure in the future. However, this correlation is the hardest to infer. Oftentimes the logic behind a group of failures is so complex that the correlation can easily be overlooked. Whenever the logical correlation between node failures is not immediately visible, analyzing the other two dimensions (time and space) of failure correlation might help to infer the existence or absence of a logical correlation between failures. By definition, logical correlations are derived from strong correlations in space and time [253].

Figure 4.8a shows a group of 22 failures over 24 hours in Taurus. Each column shows the physical location of nodes in an *Island.Rack.Chassis* format (e.g., I1.R2.C3 indicates a node located in Chassis 3 of Rack 2 in Island 1). The thick horizontal lines divide failures into four sections according to the time of failure, i.e., temporal correlations. Same temporal correlations are represented by four different colors in the `Time` row of Figure 4.8b. In Fig. 4.8b, color coding is used to show the correlation between failures in each row. Failures with identical color in the *time* row are occurred in a certain time window (e.g., 10 minutes). Failures with identical color in the *chassis*, *rack*, or *island* rows, occurred in the same chassis, rack, or island, respectively, as the failures preceding them in these locations. Failures with identical color in the *reason* row, occurred due to the same logic as other failures on this day.



(a) Node failures over 24 hours on 21-04-2015. Temporally correlated failures are shown in three different horizontally divided sections. All failures with same color pattern are also spatially correlated, regardless of their section.



(b) Correlation of node failures over 24 hours on 21-04-2015. The number of correlated failures plays an important role. In each group of correlated failures, the dimension which has more correlations is the dominant one.

Figure 4.8: Temporal and spatial correlation among failures

The bottom section in Figure 4.8a contains only one failure and is, therefore, excluded from further correlation analysis. The remaining three sections indicate strong temporal correlations between failures. An identical color pattern, in Figure 4.8a, indicates spatial correlation among failures i.e., failures that occurred in the same rack. All failures shown in Figure 4.8 were occurred in one island (Island 1). According to Figure 4.8b except one single failure, the remaining 21 failures are logically correlated. Backtracking the system logs revealed that the 21 failures (shown in red) were raised by a problem in the distributed file system[3], and the failure with a different reason (shown in blue) was due to an *out of memory* problem.

Temporal and spatial correlations are direct and simple correlations. There are more complicated correlations among computing nodes in HPC systems that can better explain the cause of failures and their propagation. Thus far, the complicated correlations were referred to as the logical dimension. These three dimensions can be extended to arbitrary number of dimensions based on the relations among computing nodes. This work proposes the concept of node vicinity to extend the failure correlation dimensions on HPC systems.

4.2.1 Node Vicinities

Computing nodes with similar characteristics are considered to be in the *vicinity* of each other. Node characteristics include any physical, spatial, temporal, or logical properties of the computing nodes. A group of computing nodes located in the same rack, performing

[3]Lustre file system maintains a large amount of POSIX-related metadata that are highly error-prone. The implementation of Lustre file system checker(LFSCK) that is designed to detect metadata inconsistencies is also sub-optimal. Therefore, the administrators reluctant to use it as regular maintenance tool [254].

different tasks of the same job, or sharing a common resource (e.g., file system, power supply unit), are all examples of nodes in the vicinity of each other.

The concept of vicinity defines new dimensions of nodes correlation, beyond the natural temporal and spatial correlations. Each vicinity can be imagined as a new dimension in which two separated entities (nodes) become correlated. For example, points $A : (1, 10)$ and $B : (4, 6)$ in a 2D Cartesian representation are separated by the distance of $4 - 1 = 3$ on the X axis and $10 - 6 = 4$ on the Y axis, respectively. Defining the new dimension Z, according to a common (but so far unseen) feature of A and B would result in a 3D representation of $A : (1, 10, 5)$ and $B : (4, 6, 5)$. Here '5' denotes that common feature. In the new 3D representation, even though A and B are still separated on X and Y, their distance on the dimension Z will be $5 - 5 = 0$. In another word, A and B will be in the vicinity of each other from the Z axis perspective.

Among the three dimensions of failure correlation, discussed in Section 4.2, the logical dimension is an indirect and complex dimension, which is hard to infer. The logical dimension can be further divided into more direct dimensions such as hardware architecture and resource allocation. Considering this division, node vicinities are observed from four different perspectives: (1) *hardware architecture*, (2) *resource allocation*, (3) *physical location*, and (4) *time of failure*. The first perspective denotes a node vicinity according to the node's physical properties, the second perspective emphasizes the node's logical properties, while the third and fourth perspectives denote spatial and temporal properties, respectively. Similar to the three-dimensional correlation approach, all correlations among nodes can be mapped onto these four proposed vicinities, e.g., nodes connected to a single switch can be mapped onto the *physical location* vicinity. In this subsection these four vicinities are explained in more detail, based on the Taurus architecture.

The node vicinities are intended to mitigate the major characteristic differences between nodes. Therefore, in cases that several parameters influence a certain node's characteristic, the most dominant parameter is considered to identify the node's vicinity. All nodes in Island 2 beside their Sandy bridge or Haswell CPUs are equipped with graphical processing units (GPU). Since the majority of jobs submitted to Island 2 mainly utilize GPUs rather than CPUs, GPUs are considered as dominant processing units of these nodes. Therefore, in this work, Island 2 is considered as a homogeneous GPU island, despite the heterogeneity of its nodes' CPUs.

It is important to emphasize that in the context of this work, two nodes in the *vicinity* of each other are not necessarily physically co-located. In fact, they may even belong to physically separated partitions of the HPC system.

Hardware Architecture Vicinity

Computing nodes on Taurus may be of four different processors architectures: Intel Haswell, Broadwell, Sandy Bridge, and Westmere. 108 nodes with Sandy Bridge and Haswell processors are also equipped with GPUs (NVIDIA Tesla K20X and K80). According to their hardware architecture, the $2,046$ computing nodes on Taurus can be divided into five categories. The node's dominant processor architecture and the number of nodes in each architecture category are shown in Table 3.1 on page 27. A schematic illustration of the Taurus topology, including the type of each node's hardware architecture is provided in Figure 3.2 on page 27. Nodes with identical colors in Figure 3.2 are in the vicinity of each other from the hardware architecture perspective.

It is important to note that beside the processor architecture, other hardware characteristics (e.g., the amount of physical memory) were also considered and analyzed. However, the final results indicated that the node's main processor plays the dominant role for the purpose of this work thus, significantly outperforms the impact of other hardware characteristics. Therefore, the hardware architecture vicinity considers only the node processor architecture.

Resource Allocation Vicinity

Slurm [232] schedules the jobs in Taurus. The resources are allocated to each submitted job according to the direct request of user, system policies, and the status of available resources. All nodes that execute tasks of the same job are in the vicinity of each other from the *resource allocation* perspective. In contrast to the static nature of the hardware architecture perspective, the resource allocation vicinity is fully dynamic and may change frequently as the running jobs are completed and new jobs are submitted to the cluster.

Physical Location Vicinity

Various granularities can be used to express the physical location of a node in Taurus, e.g., chassis, rack, or island. Since the power, temperature, and connectivity of all nodes located in a single rack are controlled together, this work considers racks as the physical location granularity. Each row of an island shown in Figure 3.2 on page 27 represents one rack of nodes. All nodes located in the same rack are in the vicinity of each other from the *physical location* perspective[4] [5].

Time of Failure Vicinity

Often failure is a consequence of various node-level events on and of properties of several nodes. However, a failure in itself is observable on a particular node at a specific moment. Therefore, the time of failure is considered as a temporal property of that particular node even though, several nodes may fail due to the same reason. From this perspective, all nodes that experience a failure within the same predefined time interval, fall into the same vicinity category. In this work, the time of failure interval is 10 minutes. The 10-minute time interval is chosen according to the results of the previous study on Taurus failure correlations [250]. That study revealed that the majority of failures correlated on Taurus occurred within 10 minutes of each other. Therefore, failures that occur across the

[4]Other studies also confirm the "small-region locality correlation" [26].
[5]The phisycal localities are shown to be beneficial also for load balancing and resource allocation [255].

entire system within 10 minutes of each other are assumed to be in the same temporal vicinity from the *time of failure* perspective.

4.2.2 Impact of Vicinities

Taurus nodes are located in 6 islands. As shown in Figure 3.2, on page 27, Island 4 hosts nodes with two different processor types, while Islands 1, 2, 3, 5, and 6 are homogeneous. Although the nodes' hardware architecture influences the job allocation, as Figure 3.22b on page 55 illustrates there is no noticeable difference among job allocation patterns of Taurus islands. However, as shown in Figure 3.22a, except Island 5 and Island 6 that consist of identical processor types, the node outages have different distribution pattern on each island.

Figure 4.9 illustrates a one-to-one comparison of syslog generation patterns[6] in all Taurus islands. This figure visualizes the temporal and spatial patterns among more than 46K, 82K, 45K, 968K, 940K, and 1M syslog entries generated by Islands 1 to 6, respectively. Islands 5 and Island 6 present an almost identical pattern, which is also very similar to Island 4. In contrast, Island 1, Island 2, and Island 3 have a completely different system log generation pattern.

The comparison shown in Figure 4.9 indicates that the processor architecture has a direct impact on node behavior. Therefore, the behavior of nodes in Island 1 (Sandy Bridge) should not be predicted based on the behavior of nodes in Island 5 (Haswell), while a similar behavior is expected from nodes in Island 5 (Haswell) and Island 6 (Haswell).

Node vicinities are intended to improve the accuracy of the anomaly detection method (Section 4.3) by identifying the most relevant domain among the four vicinities considered in this work. Furthermore, employing node vicinities enables the anomaly detection methods to analyze fully anonymized syslogs. To the best of author's knowledge, there is no similar approach for detecting anomalies using fully anonymized system logs. Therefore, a quantitative comparison cannot be conducted. However, Table 4.1 shows a qualitative comparison of node vicinity impact on anomaly detection.

Table 4.1: The accuracy of anomaly detection inside node vicinities

Anomaly detection	**Hardware architecture**	**Resource allocation**	**Physical location**	**Time of failure**
Inside node vicinity	Fair	Low	High	Fair (certain failures)
Outside node vicinity	No	Low	Low	No

Therefore, it can be concluded that the impact of anomaly detection inside *resource allocation* and *time of failure* vicinities of Taurus is negligible. Anomaly detection inside the *Physical location* vicinity, on the other hand, has a high impact on the accuracy of the final

[6]The dynamic frequency of syslog generation by each node during a time interval. This parameter is explained in Section 4.3.1 and referred to as SG in the rest of document.

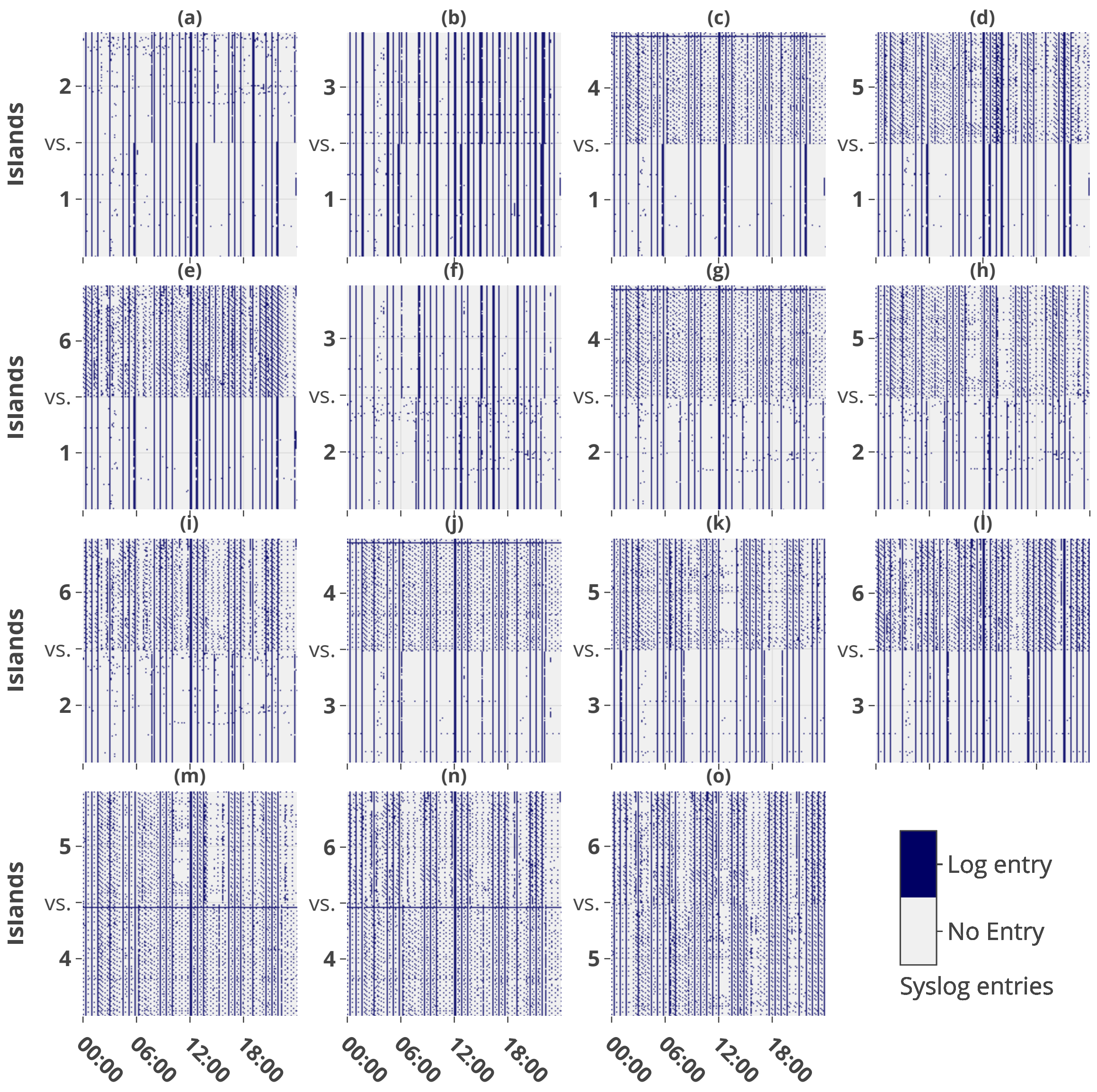


Figure 4.9: Syslog generation patterns of Taurus islands. Each sub-diagram is vertically divided into two sections. Each section illustrates the syslog generation pattern of 100 nodes of the respective island during 24 hours. e.g., sub-diagram (e) illustrates the syslog generation pattern of Island 1 (bottom) versus Island 6 (top).

results. It also became evident that the nodes behavior outside their *hardware architecture* vicinity varies significantly and is not suitable for anomaly detection.

4.3 Anomaly Detection

This work considers two behavioral patterns to model the behavior of HPC systems using discrete time series of monitoring data: (1) the order of events and (2) the frequency of events. Both patterns are automatically inferred from the behavior of the majority (majority voting) in each homogeneous neighborhood (node vicinity) and is constantly updated according to the current system status [256].

Order of Events

The behavioral pattern can be inferred from the chronological order of events recorded in monitoring data. Two different variants of this pattern are defined: (1) time-based (i.e., periodic) and (2) event-based. The former preserves the time interval between two consecutive events, and the later only considers the chronological order of events regardless of the inter-event intervals. As described earlier, system logs contain both periodic and event-driven entries. The time-based patterns are suitable for periodic entries (e.g., cron jobs) while the event-based patterns are more suitable for event-driven entries (e.g., users interaction). The behavioral patterns based on events' order are extracted in different granularities. A subset of potential granularities is shown in Figure 4.10.

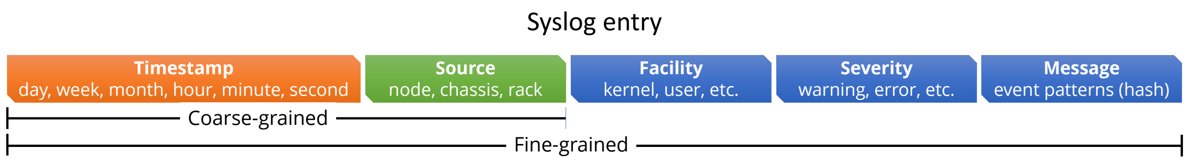


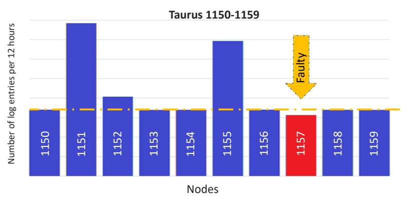
Figure 4.10: Main components of syslog entries. Each entry may perceived in different granularities.

For the purpose of this work, the preferred granularity of `source` is node ID and the preferred granularity of `timestamp` is the full timestamp including both date and time. Both `facility` and `severity` fields (metadata) of system logs are considered.

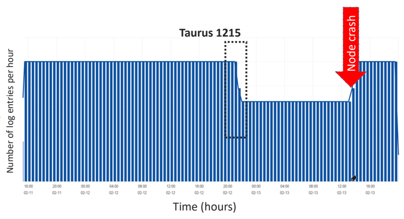
Frequency of Events

The behavioral pattern can also be inferred from the number of syslog entries received during a certain time interval, regardless of the events order. A filtering mechanism suppresses the irrelevant entries. Similar to the patterns that are extracted based on events order, the patterns of events frequency can be extracted in different granularities.

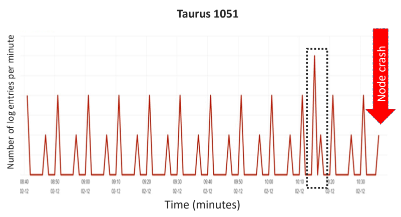
Figure 4.11 illustrates three anomaly patterns that are extracted using the events frequency of Taurus syslog entries. The first pattern shown in Figure 4.11a identifies the faulty node 1157 using its lower frequency of entries in comparison to the other nodes in its vicinity. The second and third patterns shown in Figure 4.11b and 4.11c illustrate two common failure patterns in form of valley and peak over time, respectively.



(a) Majority voting



(b) Valley



(c) Peak

Figure 4.11: Failure patterns inferred based on the frequency of events on Taurus

These two forms of behavioral patterns (order/frequency of events), extracted from monitoring data, are the input data of various anomaly detection methods. In the following subsections, these patterns are used to detect anomalies and consequently to detect and predict failures.

4.3.1 Statistical Analysis (frequency)

Statistical analysis requires categorization of event entries by system experts. Such categorizations are time-consuming, only partially feasible, and subject to drastic changes after each major maintenance of the HPC system. Furthermore, due to the heterogeneity of modern HPC systems, every subset of the computing system may project a different behavior thus, a single threshold may not hold true for the entire system. To address these challenges an unsupervised method of anomaly detection based on statistical analysis is proposed. This method does not require manual categorization of entries and automatically adapts its patterns to the current status of the system. The pattern of each vicinity is separately extracted and regularly updated. Therefore, comparisons are performed locally within each vicinity, hence, increasing the overall accuracy of the anomaly detection. Figure 4.12a and 4.12b show two snapshots of the comparisons among 11 nodes in the vicinity[7] of each other.

The common behavior of the majority of nodes within a node vicinity is considered as the "normal" behavior in that vicinity (majority voting). Behavior of a node is monitored using the syslog generation frequency of that node (hereafter SG). The SG parameter is dynamically calculated based on the number of syslog entries received from each computing node during a sliding time window prior to the current (observation) moment. A selection mechanism filters out the unnecessary entries thus, maximizing the fluctuation of SG parameter[8]. The SG parameter of each node is compared against the SG of other nodes in the same vicinity. Based on these comparisons, the normal value of the SG parameter for certain node vicinity at a given moment of time is calculated. Once the deviation of a node's SG parameter from the normal value exceeds a certain threshold, the node's behavior is considered *abnormal*.

The deviation threshold is dynamically calculated within each vicinity[9]. To calculate the deviation threshold, all nodes within a vicinity (i.e. one row of Figure 4.12a) are partitioned into two clusters based on their SG parameter via a clustering method such as K-Means. The deviation threshold is the relative density of resulting clusters which is calculated as the sum of squared distances of samples to their closest cluster center[10].

Figure 4.12a illustrates the behavior of node 1110 and 8 other neighboring nodes (physical location vicinity) prior to 11 points in time that node 1110 experienced failure in the year 2017. Figure 4.12b on the other hand, illustrates the behavior of the same nodes prior to 11 random points of time in which node 1110 was functioning normally. In Figure 4.12a and

[7]Physical location vicinity

[8]For example, all periodic entries can be suppressed to prevent smoothening of the irregular entries.

[9]A sample code written in python to demonstrate the calculation of dynamic thresholds via k-means is available: ghiasvand.net/u/param

[10]Also known as: *within cluster sum*

	Node ID									
Time of Failure on node 1110	1106	1107	1108	1109	**1110**	1111	1112	1113	1114	**Norm**
10/03/2017 13:55:51	11	11	11	206	214	15	15	15	15	13
10/03/2017 14:07:24	17	16	20	306	1681	21	21	21	21	76
14/03/2017 16:45:56	69	71	67	77	76	59	68	76	55	71
22/03/2017 13:31:13	60	52	28	30	107	21	30	30	30	38
04/05/2017 01:39:16	7	7	7	7	7	7	7	7	7	7
31/07/2017 12:30:49	10	10	10	0	14	10	10	10	10	10
31/07/2017 15:20:17	23	23	23	0	36	23	23	23	23	23
02/08/2017 16:40:15	5	5	7	0	20	5	5	5	5	6
08/08/2017 12:27:44	10	10	10	341	212	11	11	11	11	10
08/08/2017 13:30:34	20	20	10	157	156	11	11	11	11	14
08/08/2017 13:55:20	12	6	6	1581	1593	9	9	9	9	9

(a) Detection of failures (shown in orange) of node 1110 via the proposed failure detection mechanism. Cells colored in light blue indicate non-responsive nodes.

	Node ID									
Time of random events, node 1110	1106	1107	1108	1109	**1110**	1111	1112	1113	1114	**Norm**
04/03/2017 15:07:52	18	17	17	0	19	19	19	17	17	17
08/04/2017 15:43:38	12	11	12	11	12	12	11	11	11	11
19/04/2017 07:30:36	42	41	40	41	41	41	41	41	41	41
01/05/2017 11:02:41	28	37	29	29	17	17	36	34	30	31
22/05/2017 14:29:44	19	19	19	19	19	19	19	19	19	19
03/06/2017 13:48:12	7	6	7	6	7	7	6	6	6	6
14/06/2017 16:45:55	19	18	19	18	19	19	18	18	18	18
02/07/2017 12:47:53	6	6	6	0	9	6	6	6	9	6
19/08/2017 13:23:33	11	11	11	12	12	11	11	11	11	11
20/09/2017 17:16:35	12	12	12	12	12	12	12	12	12	12
12/10/2017 05:16:35	12	12	12	12	12	12	12	12	12	12

(b) Application of the proposed failure detection mechanism during normal behavior of node 1110. Detected failures are shown in orange. Cells colored in light blue indicate non-responsive nodes.

Figure 4.12: Anomaly detection in physical location vicinity using majority voting.

Figure 4.12b, the timestamp at the beginning of each row represents the failure observation moment. The value of each cell represents the SG parameter of the respective node within a time interval of 30 minutes before the observation moment. Cells with abnormal behavior are shown in orange. The cell coloring in each row is relative to the value of other cells in that particular row (vicinity). According to Figure 4.12a, node 1110 experienced 11 failures in 2017. For 7 out of the 11 failures illustrated in Figure 4.12a, the deviation of the SG parameter correctly identifies the abnormal behavior of node 1110.

The strength of this method stems from the majority voting inside node vicinities. This is an unsupervised method, therefore, no additional information, except the node vicinities, is required. Fine tuning of the sliding time window interval, as well as adjusting the data filtering rate improves the accuracy of anomaly detection. A simplified workflow of the behavioral analysis method is described below[11]:

1. The hardware architecture and physical location vicinities are identified [12].
2. Taurus syslog entries are streamed from computing nodes into the syslog collector.
3. The structured part of each syslog entry (metadata) is parsed and reduced to four fields, namely `timestamp`, `source`, `facility` and `severity`.
4. PαRS processes, de-identifies, and encodes the unstructured part of each syslog entry (message) into a fixed-size[13] hash key (event pattern) using a collision resistant hashing algorithm[14].
5. The timing errors are compensated via binning syslog entries using buckets of 10 minutes.
6. Syslog entries are grouped based on the values of their `facility`, `severity` and `message` fields.
7. The normal behavioral pattern in each vicinity is defined as the behavior of the majority in that vicinity.
8. The frequency of syslog entries (the SG parameter) within a sliding time window of 30 minutes is calculated.
9. Derivation of the node's SG parameter from the normal behavioral pattern in each vicinity (majority voting among nodes of that vicinity) is considered as sign of an upcoming failure.

Furthermore, the `confidence` value of failure prediction increases closer to the point of failure. Simultaneous anomalous behavior of multiple monitoring parameters (`facility`, `severity` and `message`) increases the `confidence` of failure prediction. Since failures are more probable to occur on nodes with a history of failures (refer to Figure 5.9b), the `confidence` of failure prediction for such nodes is higher. Larger derivation of the node's SG parameter from the normal behavior further increases the `confidence` of failure prediction.

4.3.2 Pattern Detection (order)

The pattern detection based approaches can be applied on a single node as well as multiple nodes. Applying the pattern detection approach on a group of nodes in the vicinity of each other significantly increases the accuracy. Various pattern detection based methods shown acceptable results in detecting anomalies using HPC systems monitoring data. Among them, suffix arrays and sequence analysis provides higher accuracy and less false positives. Pattern detection methods in contrast to statistical analysis mainly utilize the order of events for anomaly detection.

[11]The workflow of behavioral analysis method including all major building blocks is shown in Figure 3.1
[12]This is the only manual step of the entire workflow.
[13]The default size is 8 characters.
[14]SHAKE 128 is the default hashing function used in this work.

Suffix trees and arrays

Suffix trees and suffix arrays are powerful data structures to detect recurring patterns in long sequences. Recurring patterns play an important role in describing the normal behavior of HPC systems. Numerous Cron jobs[15] generate periodic syslog entries that always follow a certain order. Furthermore, multi-step procedures, such as authentication, generate blocks of syslog entries with predefined orders. Therefore, the goal is to detect the recurring sequences of events in syslog entries. Sudden changes in the pattern of recurring events might indicate anomalies.

Suffix arrays are constructed via performing a depth-first traversal of a suffix tree. Both suffix trees and suffix arrays solve the same problem with similar time complexity. However, suffix arrays require less space and provide better cache locality. Similar to previous method, various granularities of monitoring data can be used by suffix trees and suffix arrays. Furthermore, an independent suffix tree is built for every computing node.

A sequence of 7 syslog entries is shown in Figure 4.13a. Each entry is encoded into its event pattern (hash key) via PαRS. For better readability, every event pattern in this example is further encoded into a single character (symbol). The collection of all symbols forms the alphabet (i.e., alphabet={J,A,M,E}). To extract the patterns of recurring sequences of events, a simpler form of the classic problem of the longest repeated sub-string (LRS) should be solved. Using suffix trees, the LRS can be found in $O(n)$. The suffix tree of syslog entries from Figure 4.13a is shown in Figure 4.13b. The longest path[16] from the root to non-leaf vertexes identifies the LRS. In this example, among the non-leaf vertexes of 5, 7 and 9 the path from root to vertex 5 is the longest path. Therefore, the longest repeated sequence is `J` » `A` » `M` or `23666bbc` » `760c5208` » `85f5c18b`.

Any repeated sequence of events (two and more events) represents a pattern and is significant. Thus, instead of extracting only the longest repeated sub-string, all repeated sub-strings (RS) will be identified. In a suffix tree, repeated sub-strings are represented by paths from root to the parent of each leaf. Among the overlapping paths with similar repetition rate, the largest path is chosen. Paths should have a minimum length of two symbols. Therefore, according to the suffix tree shown in Figure 4.13b, although both paths from root to 5 and 7 represent a repeated sub-string, due to the overlaps only the pat h to 5 is considered.

The detected RS among system logs identifies the block of syslog entries which are expected to appear in a certain order. Any derivation from these patterns might be a sign of anomalous behavior. An example of applying pattern detection on Taurus syslog entries is shown in Figure 4.14.

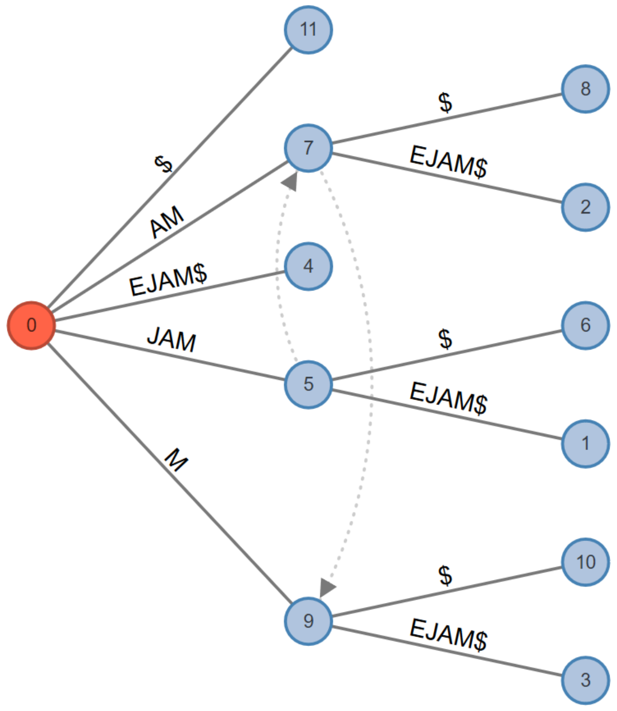
Sequence Analysis

The structure of syslog entries has resemblance to Human's DNA. Both are long sequences of unites (log entries / base pairs) with numerous repeated sub-sequences [257].

[15]Jobs which are executed using Cron, the time-based job scheduler of Unix-like operating systems.
[16]The path with the most number of symbols.

```
 TIMESTAMP  EVENT

1488424393  23666bbc  ->  J

1488424398  760c5208  ->  A

1488424414  85f5c18b  ->  M

1488424459  bba3d47c  ->  E

1488424570  23666bbc  ->  J

1488424596  760c5208  ->  A

1488424611  85f5c18b  ->  M
```

(a) Encoding event patterns into symbols



(b) The suffix tree of encoded event patterns. Each path from root to a non-leaf vertex is a repetitive sub-sequence.

Figure 4.13: Creation of suffix tree for sample syslog entries

#	Message	Event
1	(siavash) CMD (/usr/bin/check >/dev/null 2>&1)	66dc2742
2	(parya) CMD (/usr/lib32/lm/lm1 1 1)	66dc2742
3	(siavash) CMD (run-parts /etc/cron.hourly)	66dc2742
4	starting 0anacron	dd740712
5	Anacron started on 2018-01-30	e5a59462
6	Jobs will be executed sequentially	f1e7eac3
7	Normal exit (0 jobs run)	eac7924f
8	finished 0anacron	a5803a8a
9	(siavash) CMD (/usr/lib32/lm/lm1 1 1)	66dc2742
10	(root) CMD (/usr/lib32/cl/cl2 1 1)	66dc2742
11	(root) CMD (/usr/lib64/lm/lm1 1 1)	66dc2742
12	(siavash) CMD (/usr/bin/check >/dev/null 2>&1)	66dc2742
13	(parya) CMD (/usr/bin/run >/dev/null 2>&1)	66dc2742
14	(siavash) CMD (/usr/bin/exec >/dev/null 2>&1)	66dc2742
15	(siavash) CMD (run-parts /etc/cron.hourly)	66dc2742
16	starting 0anacron	dd740712
17	Anacron started on 2018-01-31	e5a59462
18	Jobs will be executed sequentially	f1e7eac3
19	Normal exit (4 jobs run)	eac7924f
20	finished 0anacron	a5803a8a

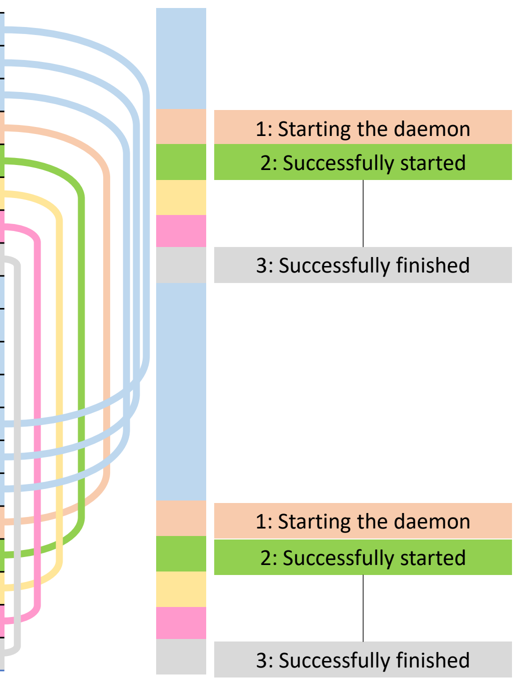


Figure 4.14: Recurring blocks of syslog entries. Similar colors indicate similar events. The events are automatically derived from syslog messages using PαRS.

Throughout the years various techniques and tools have been developed to analyze the sequence of base pairs in DNA. The goal of this section is to employ DNA sequence analyzing methods for syslog analysis based on the similarities between DNA structures and syslog entries (e.g., long range correlations [258]).

Sequence alignment map (SAM) is a text-based format suitable for storing DNA sequences aligned to a reference sequence [259]. Since the SAM format is widely used by various DNA sequence analyzing tools, in order to apply the DNA sequence analyzing methods, Taurus syslog entries were transformed into the SAM format.

Sequence alignment is used to detect anomalies with a reduced false positive rate. Using majority voting among the nodes in the vicinity of each other, the reference sequence of events on HPC system is extracted. The system logs are aligned to the reference sequence in their SAM format. Derivations from the reference sequence can be detected via sequence alignment methods. All existing derivations are not erroneous, in fact a certain degree of change is expected due to the dynamic nature of the HPC systems and the users behavior.

Point mutation is a change in the bases of DNA. This concept is used to mitigate the noises in system logs and reduce the false positives. Single mismatches during the comparison of current sequence and the reference sequence is interpreted as noise (point mutation).

Sequence motifs are significant sequence patterns in DNA. For system logs the sequence motifs are defined according to their length, frequency, and accuracy of repetition. Longer sequence patterns that are more frequent and have precise recurring time intervals are considered as sequence motifs. Derivations of sequence motifs from the reference sequence are interpreted as signs of anomalies.

The main advantage of using tools and methods that are originally built for analyzing DNA sequences is their capability in processing radically large sequences. However, to be able to use those tools and methods, the monitoring data (e.g., system logs) must be transformed into compatible formats. Furthermore, the important features of the monitoring data must be preserved. In this work the transformation of Taurus system logs into sequence alignment map is performed using PαRS.

PαRS encodes syslog entries into anonymized event patterns (*anonymized list*). Frequency of each event pattern in a fixed interval (e.g., 24 hours) is calculated. A reverse ordered (descending) list of all event patterns based on their frequency is generated (*frequency list*). The frequency list is divided into subsections, each with 4 entries (*frequency sub-list*). The anonymized list is divided into subsections according to the event patterns in each frequency sub-list (*anonymized sub-list*). A unique symbol from a 4-letter predefined alphabet (e.g., {A,T,C,G}) is assigned to each event pattern of a frequency sub-list. The event patterns of each anonymized sub-list are substituted by the relevant symbols listed in the corresponding frequency sub-list (*sequence sub-list*). Each sequence sub-list is converted into a SAM file and is ready for further analysis. Table 4.2 provides an example of applying the proposed workflow on a collection of syslog entries.

Table 4.2: Encoding syslog entries into DNA-like sequences

Anonymized list		**Frequency sub-list**			**Sequence sub-list**	
Timestamp	Event pattern	Event pattern	Frequency	Symbol	timestamp	Symbol
1490997601	34b25731	1c6f9d5e	7	A	1490997601	C
1490997601	1c6f9d5e	1808e388	3	T	1490997601	A
1490997601	1808e388	34b25731	2	C	1490997601	T
1490997602	90a389bc	fa144f05	1	G	1490997661	G
1490997661	2e307f37	e92704ab	1	A	1490997901	A
1490997661	fa144f05	ddab3d0a	1	T	1490998201	A
1490997661	6234343a	90a389bc	1	C	1490998201	T
1490997661	ddab3d0a	8c41e908	1	G	1490998501	A
1490997661	8c41e908	8372c3dc	1	A	1490998801	T
1490997661	e92704ab	6234343a	1	T	1490998801	A
1490997661	8372c3dc	35d87b50	1	C	1490999101	A
1490997661	35d87b50	2e307f37	1	G	1490999401	A
1490997901	1c6f9d5e				1490999401	C
1490998201	1c6f9d5e				1490997661	A
1490998201	1808e388				1490997661	T
1490998501	1c6f9d5e				1490997661	C
1490998801	1808e388				1490997661	G
1490998801	1c6f9d5e				1490997661	A
1490999101	1c6f9d5e				1490997661	T
1490999401	1c6f9d5e				1490997602	C
1490999401	34b25731				1490997661	G

The size of alphabet can be increased to any arbitrary number of symbols. However, large alphabets may negatively impact the detection mechanisms. According to Taurus syslog analysis, high frequency entries deliver less significant information in comparison to low frequency entries. Therefore, grouping the *anonymized list* on syslog entries into smaller sub-lists according to their frequency improves the detectability of low frequency (but significant) entries via comparing events only within groups of events with similar frequency. It is important to note that in sequence analysis, similar to other methods proposed in this work, the normal behavioral pattern is inferred via majority voting within nodes vicinity.

Figure 4.15 shows a 24-hour sample of system logs (in SAM format) collected from 100 nodes visualized using Tablet (Table B.1). Each row represents a single node (100 rows) and each column represents a one-minute time window ($1,440$ columns). Several potential anomalies in form of horizontal red zones are visible in the middle and bottom of the figure.

4.3.3 Machine Learning

The anomaly detection workflow, described in Section 4.3.1 on page 72, automatically analyzes the monitoring data and detects the anomalous node behaviors. Since the patterns of normal behavior are directly, individually and continuously extracted for each node vicinity, the anomaly detector automatically adapts itself to the system changes. The only exceptions are changing the node's processor architecture or its physical location. In the

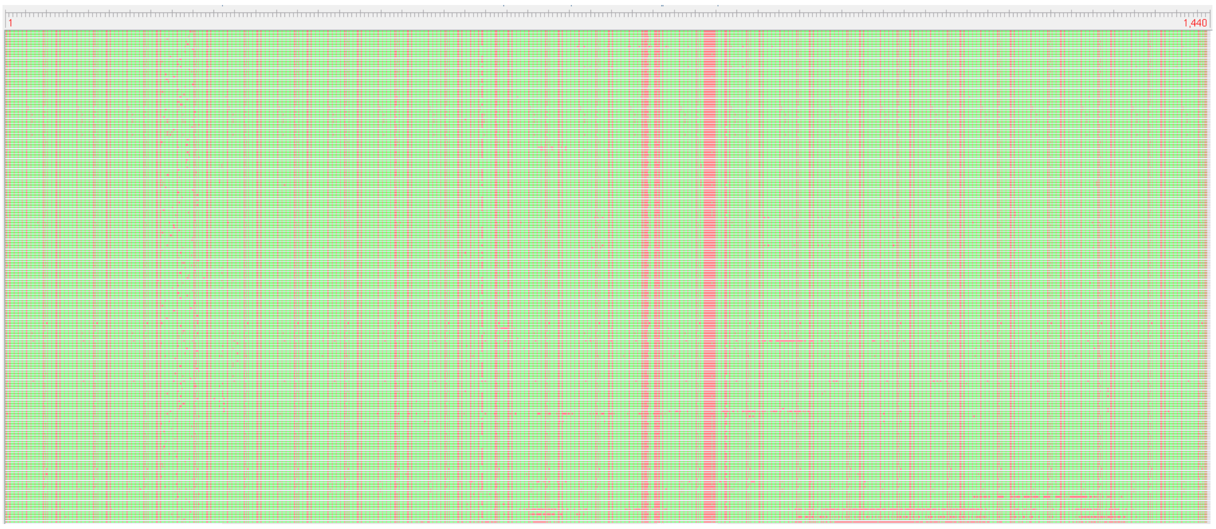


Figure 4.15: A 24-hour sample of system logs collected from 100 nodes visualized using Tablet. Each row represents a node. Each column represents a one-minute time window. Potential anomalies are visible in form of horizontal red zones in the middle and bottom of the figure.

later scenarios nodes must be re-grouped according to the new vicinities (step 1 in the workflow). However, the rest of the workflow (steps 2-9) remains intact.

In addition to the main behavioral analysis method, two alternative approaches were proposed using neural and hierarchical temporal memory (HTM) networks. The neural network approach is defined to extend the known set of correlations among nodes and failures. The HTM approach on the other hand is intended to detect anomalous behaviors of individual nodes without (explicitly) considering node vicinities. Preliminary results indicate high potentials of machine learning techniques for automatic detection of abnormal behaviors in HPC systems using anonymized system logs. Figure 4.16 illustrates the workflow of the proposed approach for detecting anomalies and predicting systems behavior using system logs.

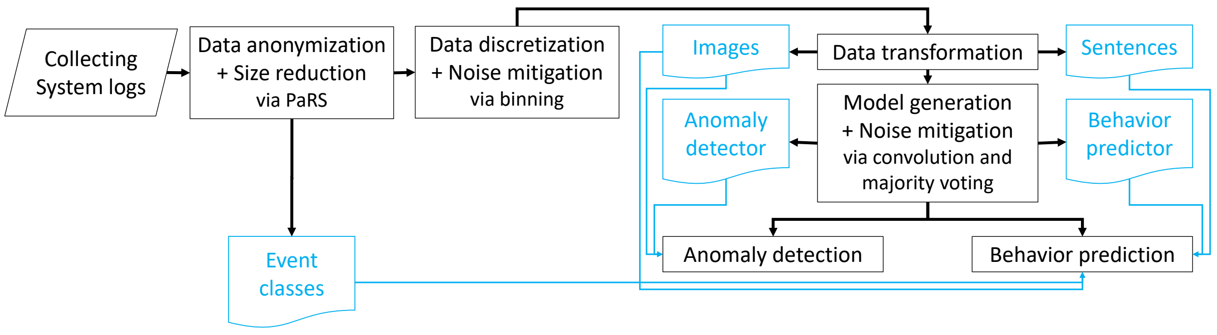

Collecting System logs
Data anonymization + Size reduction via PaRS
Data discretization + Noise mitigation via binning
Images
Data transformation
Sentences
Anomaly detector
Model generation + Noise mitigation via convolution and majority voting
Behavior predictor
Event classes
Anomaly detection
Behavior prediction

Figure 4.16: The workflow of anomaly detection via syslog analysis with the focus on machine learning.

Insufficient amount of failure samples was one of the main motivations to use an unsupervised approach for anomaly detection in this work. On Taurus, except certain failures which are caused by distributed file system, rest of the failures are not frequent[17]. Due to insufficient amount of failure samples, automatic extraction of abnormal patterns, which are leading to non-frequent failures is not feasible. Therefore, rather than extracting the pattern of abnormal behaviors and using them to identify similar anomalies, this work considers the common system behavior[18] as the norm and evaluates the divergence of system behavior from this norm.

Three classical neural network (NN) models and one hierarchical temporal memory (HTM) model are used. For the first two NN models, syslog entries are transformed into images and processed via image processing techniques, while the third NN model uses a text auto-completion technique to predict the upcoming events [260]. The HTM model, which is based on a biologically constrained model of intelligence, uses a sparse distributed representation of system logs. Although the classical NN models show promising potentials, the HTM model outperforms them. Thus, this work focuses on the HTM model. Appendix A provides detailed information about the proposed NN models.

Hierarchical Temporal Memory

The anomaly detection approach in this work relies on the node vicinity and majority voting. However, a large system-wide failure may alter the behavior of the majority of nodes. In such cases, majority voting within the node's vicinity may fail to detect the abnormal behavior. Furthermore, as HPC systems are becoming more heterogeneous, identifying the homogeneous sections of the system becomes challenging. Therefore, the HTM model is used to complement the vicinity-based anomaly detection method and improve the overall prediction accuracy.

HTM is a biologically constrained model of intelligence [261]. The main ability of HTM networks is anomaly detection using small amount of input data [262]. In the HTM model used in this work, the input data (syslog) is semantically encoded (using PαRS) as a *sparse distributed representation* (SDR). SDR is an array of 0's and 1's, that represent neurons. An SDR at each stage has less than 2% of its elements 1's (active neurons). This encoded sparse array is further normalized via *spatial pooling* into a sparse output vector with fixed sparsity. The process of spatial pooling uses the *temporal memory* algorithm to retain the context of the input data. The temporal memory learns the transitions of patterns as they occur and recalls the sequences of previous patterns. Therefore, as the input changes, the HTM model updates itself (online learning). Using these steps, the HTM builds a predictive model that is capable of providing multiple predictions simultaneously and evaluate their likelihood online.

HTM models, in contrast to the other contextual-aware recurrent neural networks such as LSTMs, do not employ back-propagation. Instead, HTM works based on (unsupervised) Hebbian theory [19].

[17]Less than 10 occurrences per year

[18]Within each node vicinity

[19]A theory in neuroscience that claims "an increase in synaptic efficacy arises from a presynaptic cell's repeated and persistent stimulation of a postsynaptic cell." In another word, "Cells that fire together wire

The HTM model, used in this work, is designed to consider the behavior of an individual node and compare its current behavior with its previous behavior. Sudden changes in the current behavior are assumed as abnormal behavior. The current implementation of the HTM network in this work is based on `NuPIC` python library and heavily borrows from the `NuPIC`'s documentation and sample codes.

The task of anomaly detection in HPC systems using syslog entries is highly compatible with the characteristics of HTM networks. Therefore, this approach has been also applied on Taurus monitoring data. Designing the *spatial pooler* is the most challenging part of building an HTM model. The spatial pooler must (1) maintain a fixed sparsity regardless of the input size (normalizing the input) and (2) maintain the overlap properties such that two similar input generate two similar output. Such spatial pooler provides a correct conversion of syslog entries into an equivalent sparse representation that accurately delivers the semantics and correlations among syslog entries. Users behavioral pattern significantly changes in respect to time and date. Since users behavior affects the behavior of computing system, rather than using the temporal information of system logs only in its chronological form, the `timestamp` is interpreted into three repetitive values, namely `time of the day`, `day of the week` and `weekday or weekend`. Considering the fully anonymized syslog entry shown in Example 4.1, the four remaining data fields are `node ID`, `severity`, `facility`, and `message` respectively. Therefore, the resulting SDR consists of seven sections[20].

Example 4.1: Encoded syslog entry prepared for HTM model

```
Timestamp             | Node ID | Severity | Facility | Message
2017-12-12 01:56:00     5314      6          10         f9cfa0b9
```

The `time of the day`, `day of the week` and `weekday or weekend` are encoded into 24, 7 and 2 bit SDRs respectively. In current implementation the `node ID` is skipped, since every node has its own HTM network. However, for technical reasons a dummy 2-bit zero-filled SDR is reserved for the `node ID` which can be extended. The remaining three fields, namely `facility`, `severity` and `message` are encoded into 168, 504 and 500 bit SDRs respectively. Concatenating these seven SDRs forms the final SDR with 1207 bits. The higher weight in the current setup is given to the `facility`, `severity` and `message` values of the syslog entries.

The results of analyzing 2000 syslog entries of node taurusi5314 using the implemented HTM network is shown in Figure 4.17. Vertical red lines mark the anomalies within the stream of syslog entries.

The HTM model predicts the future trends of syslog entries, based on the previously seen entries. In current implementation, more than 50% derivation between the actual trends of incoming system logs and the predicted trends based on previously seen system logs is considered as an anomaly. In Figure 4.17 the first anomaly (901) was detected

together" [263].

[20] `time of the day`, `day of the week`, `weekday or weekend`, `node ID`, `severity`, `facility`, and `message`.

at 14:19:13 based on abnormal `slurm` daemon behavior. The second anomaly (1262) was reported at 17:17:46 as kernel issued unexpected log entries. The complete node outage happened at 21:35:01 and the node was rebooted at 21:38:19. The third and fourth anomalies (1511 and 1541) reported by the HTM model are caused by system's activities during the unstable phase after the node reboot. In practice, *jam–e jam* utilizes both majority voting

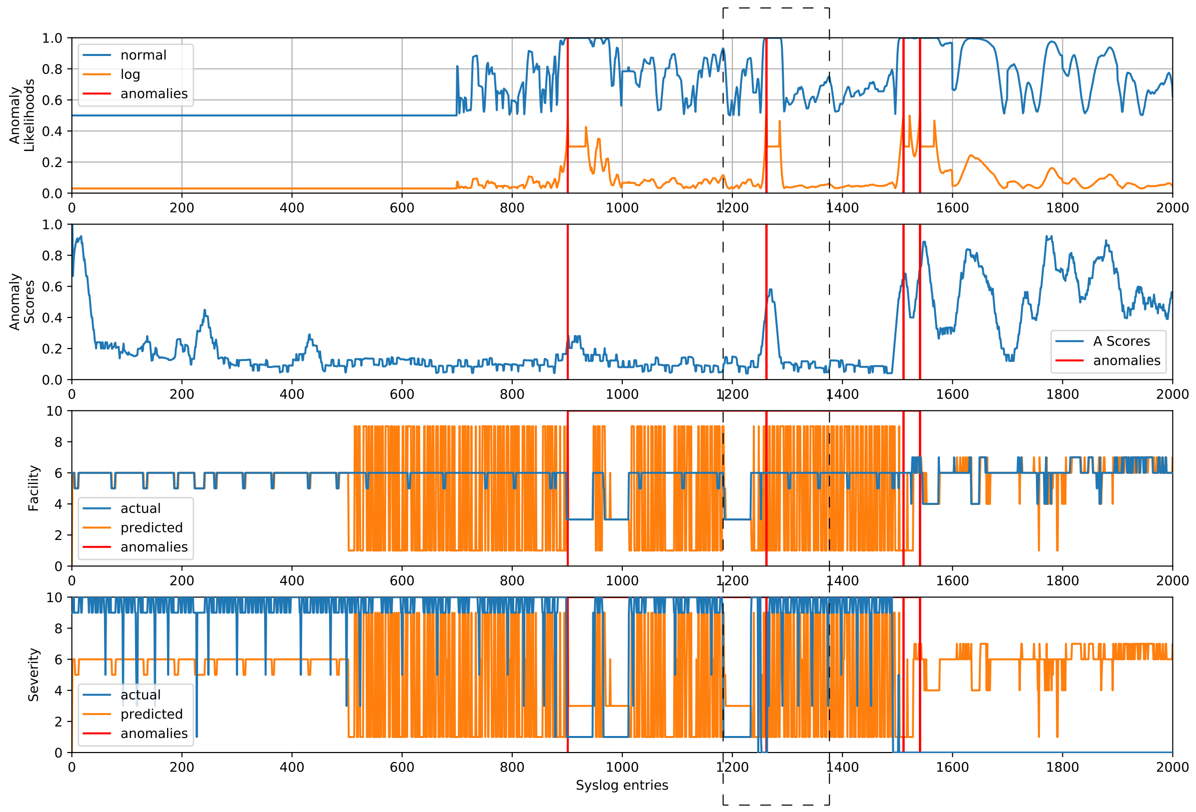


Figure 4.17: Unsupervised detection of anomalies among 2000 syslog entries of node `taurusi5314` using the HTM network. Anomalies are marked with vertical red lines.

and HTM methods simultaneously. The input stream of the system logs is sent to both detectors, and both anticipate the future state of the system. If both methods predict similarly, their prediction will be announced as the final prediction. However, if the outcome of HTM and majority voting oppose each other, according to the confidence level of each prediction (relative distance from the threshold), the final output will be a weighted average of both predictions. Furthermore, as time passes, each round that a new prediction confirms the outcome of previous prediction, the confidence value of that prediction increases. The confidence value of each failure prediction is reported together with the prediction.

In summary, the HTM-based model monitors the behavior of each node individually. The HTM is proposed as a complementary method for anomaly detection in heterogeneous HPC systems and during major system-wide failures. The model is highly noise-resistant with a short training phase, and continuously adapts itself to the new behavioral patterns. The combination of statistical analysis (majority voting) and HTM-based model significantly reduces the false positive predictions.

4.4 Adaptive resilience

Different approaches in HPC systems have been introduced to prevent failures (e.g., redundancy) or at least to minimize their impacts (e.g., via checkpoint-restart). In most cases, when these approaches are employed to increase the resilience of certain parts of a system, performance significantly degrades, and/or energy consumption rapidly increases.

Since there is no eternal hardware, in theory, failures can not be truly avoided. However, it is possible to significantly decrease the probability of their occurrence, or in some cases postpone them. In this work, avoidable failures are defined as failures that can be hidden from a specific system layer. The contrary cases are unavoidable failures. The origins of failures in an HPC system can be analogized to a tree, which has its roots in the lowest system layer and its leaves in the highest layers. Figure 3.3 on page 28 illustrates this analogy. Traversing the system from top to bottom, the diversity of failures decreases while their impacts increase. During propagation across system layers, failures may retain their original characteristics, or they may morph into other types of failures. Therefore, each system layer requires its own protection to prevent the propagation of specific failures to the upper layers.

While protection layers are added between system layers to identify, address, and prevent failures from propagating upwardly, certain overheads are imposed on the system. As long as the failure protection layers are in place, they impose overheads, regardless of the presence or absence of failures. In certain cases, adding overheads might not be worthwhile to provide fault tolerance.

The proposed approach in this work adapts the level of resiliency to the system condition via on-demand activation of available failure protection mechanisms according to probability of failure occurrence [264]. In the case that predicted failures cannot be mitigated using a predefined method, e.g., when no surrogate resource is available, the system administrators will be notified in view of performing further investigations and reactions. This approach provides adaptive resilience, progress in computation, and saves energy.

5 Results

This work proposes multiple data collection, preparation and validation approaches (Figure 3.1), as well as several methods for anomaly and failure detection (Figure 4.1). However, some proposed methods have operational overlap, e.g., noise mitigation via data binning and via majority voting. Others, such as DNA sequencing or topological analysis, are designed to provide offline information about the HPC system and its requirements, thus those are not directly utilized for online anomaly detection and failure prediction. There are also several methods which are proposed to fulfill identical tasks of failure prediction via various approaches and in different scenarios, e.g., LSTM and HTM. In the end, although all proposed methods are providing competitive outcomes, *Jam–e Jam* utilizes only those methods that provide the most appropriate result according to the conditions of the underlying HPC system, i.e., Taurus. In Figure 5.1, orange blocks are those building blocks of *Jam–e Jam* that are used to achieve the results provided in this chapter.

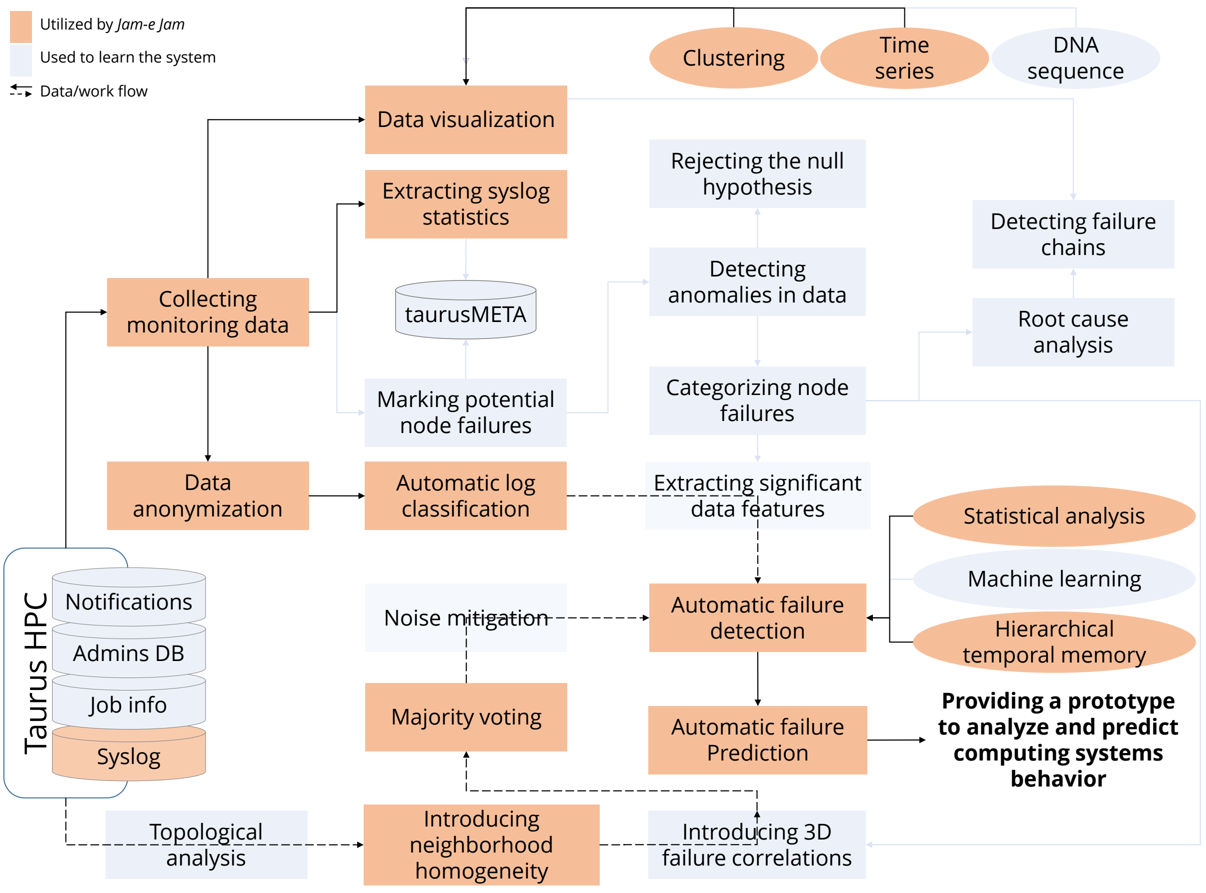

Utilized by Jam-e Jam
Used to learn the system
Data/work flow
Clustering
Time series
DNA sequence
Data visualization
Extracting syslog statistics
taurusMETA
Marking potential node failures
Rejecting the null hypothesis
Detecting anomalies in data
Categorizing node failures
Extracting significant data features
Detecting failure chains
Root cause analysis
Collecting monitoring data
Data anonymization
Automatic log classification
Statistical analysis
Machine learning
Hierarchical temporal memory
Taurus HPC
Notifications
Admins DB
Job info
Syslog
Noise mitigation
Automatic failure detection
Majority voting
Automatic failure Prediction
Providing a prototype to analyze and predict computing systems behavior
Topological analysis
Introducing neighborhood homogeneity
Introducing 3D failure correlations

Figure 5.1: Building blocks and the workflow of *Jam–e Jam* in operation mode. Building blocks shown in orange are used to achieve the results provided in Chapter 5.

Taurus has been used as the use case in this work. Various monitoring data of Taurus such as syslog entries, power consumption and job reports were collected. To provide a general approach that is applicable on other TOP500 HPC systems, the syslog entries were chosen as the main source of monitoring data in this work and the rest were only used for verification purposes. This chapter summarizes the results of proposed methods.

5.1 Taurus System Logs

During the 365 days of the year 2017, in total more than 3.2 billion syslog entries with a total size of 344 GiB were collected. Detailed statistics regarding the number of Taurus syslog entries, divided by their originating island, is shown in Table 3.6 on page 35.

The column `Node/Day` shows the average number of entries generated by a single node per day in its respective island. Figure 3.15 illustrates the distribution of syslog entries generated by each node in the year 2017. Although several outliers are visible, the majority of nodes in each island generated similar number of syslog entries. Figure 5.2 shows a side by side comparison of the average number of syslog entries generated by each node in the year 2017.

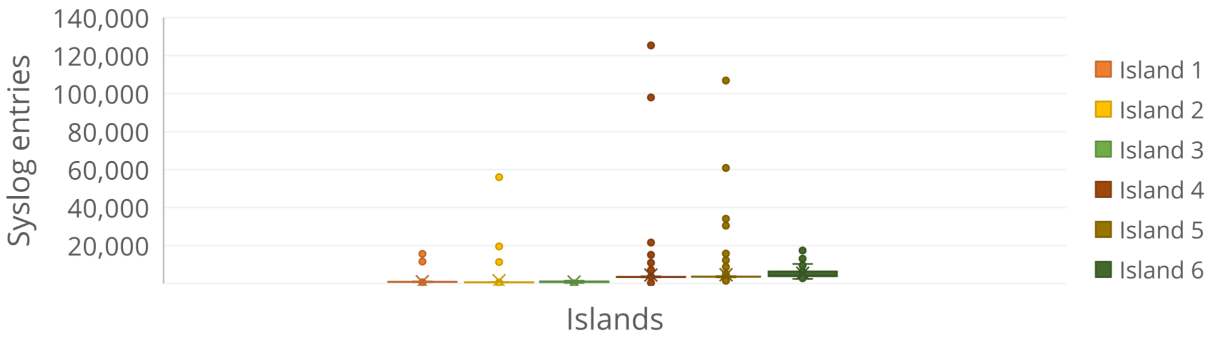


Figure 5.2: Comparison of the average number of syslog entries generated by each node in the year 2017.

The timestamp of Taurus system logs has the accuracy of one second. Therefore, the chronological order of syslog entries generated by a computing node within a second (hereafter: simultaneous entries) are not known. However, based on the analysis of Taurus monitoring data, it can be concluded that the order of simultaneous entries is a consequence of internal system characteristics. Therefore, recurring simultaneous entries either reappear in the same order as previous occurrences or the change of the order can be safely ignored[1].

Furthermore, a sudden increase in the number of simultaneous entries of each computing node can be a sign of behavioral anomalies. As shown in Figure 5.3, receiving up to 10 simultaneous entries per node is a common behavior. Between 10 and 40 simultaneous entries is common but requires further investigations. Receiving more than 40 simultaneous entries in a second is most likely a sign of behavioral anomaly.

[1]In this work the simultaneous events are always sorted to avoid the ordering problem.

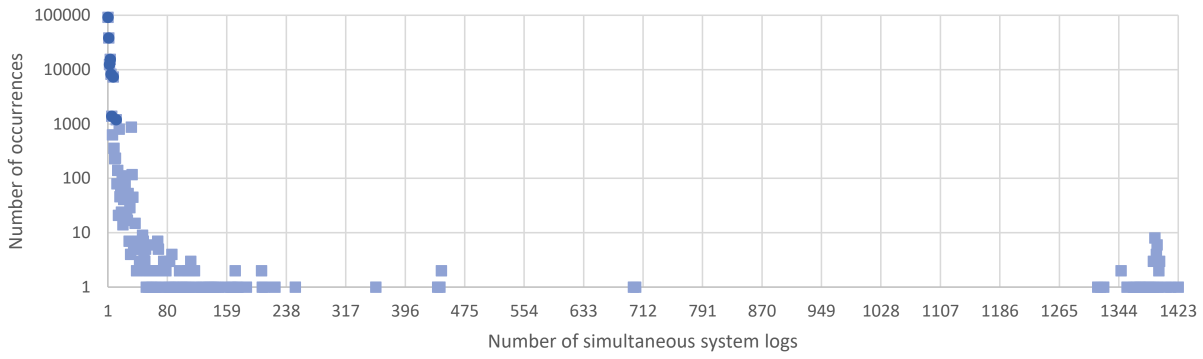


Figure 5.3: Average frequency of simultaneous entries in Taurus. Number of simultaneous entries shown in light blue has low frequency, thus might be sign of anomalies.

The `#Event patterns` column in Table 3.6 on page 35 indicates the number of extracted unique event patterns per island. Event patterns carry the essential semantics of syslog entries. The number of unique event patterns that exclusively exists in a certain island is shown in the right most column of Table 3.6. According to the values shown in column `Exclusive`, Island 2 has 357 exclusive event patterns which is the highest number of exclusive patterns among all 6 islands of Taurus. Island 2 also has the highest ratio of exclusive event patterns with as high as 26% of its total event patterns. Furthermore, Island 2 hosts the least number of computing nodes (108 nodes) in comparison to other islands of Taurus (refer to Figure 3.2). This behavior is caused by the special hardware characteristics of Island 2, as the only GPU enabled island of Taurus. Observing these phenomena further confirms the importance of performing syslog analysis within the Hardware Architecture Vicinity.

Data Filtering

Production HPC clusters such as Taurus are under constant maintenance. Furthermore, various research projects demand frequent changing of hardware and software. Syslog is an invaluable facility for monitoring and debugging such changes. Therefore, it is common to observe temporary appearance of unknown syslog entries within the stream of system logs. In this work data filtering is used to address this challenge as well as improving the accuracy of anomaly detection mechanism.

In summary, all syslog entries with the `severity` level of `debug` are removed. Project specific log entries[2] are kept in Taurus syslog collection, however, they are not included in syslog analysis. To preserve the highest level of accuracy, no further filtering has been performed in this work.

Syslog Event Patterns

In total 4027 unique event patterns were extracted from Taurus system logs in the year 2017. Figure 5.4 shows the distribution of extracted event patterns on 6 islands of Taurus.

[2]Certain research projects on Taurus are producing their own customized log entries which are directed to

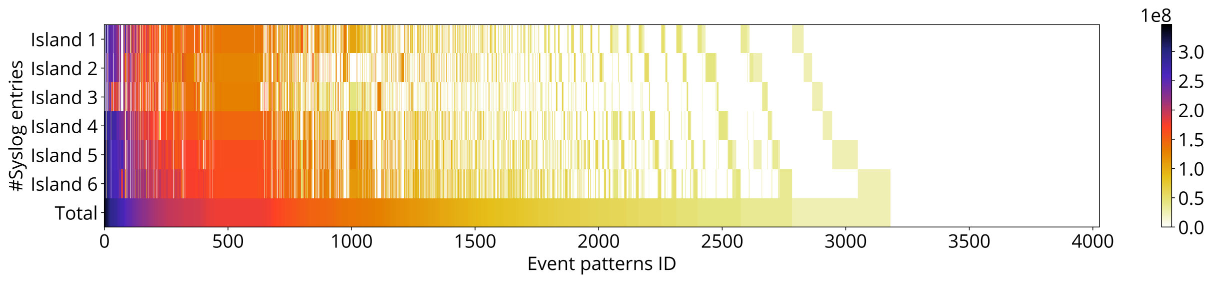


Figure 5.4: Distribution of 4027 extracted event patterns in each of the 6 Taurus islands

Out of the 4027 event patterns shown in Figure 5.4, 660 common event patterns exist on all Taurus islands. Although few exceptions are observable, the general ratio of common event patterns frequency in all islands are identical.

Surprisingly, the generation of syslog entries on Taurus does not follow the Pareto principle[3]. On Taurus, more than 80% of the common system logs are generated based on as few as 25 event patterns[4]. Therefore, more than 80% of syslog entries are generated based on less than 4% of common event patterns. These highly frequent syslog entries are periodic and reappear after fixed time intervals. The deterministic pattern of highly frequent syslog entries is used as a baseline to analyze the occurrence of event-driven syslog entries. The absence or sudden changes in the occurrence pattern of highly frequent syslog entries can be a sign of potential anomalies.

Storage Size Reduction

Applying PαRS on Taurus syslog collection from the year 2017 and preserving all metadata, reduces the total size of syslog collection from $416\ GiB$ to $258\ GiB$ which is 38% reduction. Applying PαRS and preserving only the 4 necessary metadata (timestamp, node ID, severity, and facility) reduces the size of syslog collection to $140\ GiB$ which is equal to 66% reduction.

The above calculations are made assuming that the final hash key has a length of 8 bytes. However, the total number of event patterns in Taurus is only 4026. Therefore, the entire hashing space can be further squeezed into 3 bytes hash keys. Thus, a reduction ratio of 70% can be achieved. For long-term storage of the syslog collection, additional lossless compression mechanisms can be employed. Different levels of size reduction using PαRS are shown in Figure 5.5.

As shown in Figure 5.6, the `message` field of Taurus syslog entries consist of words with various lengths from 1 to 34 letters. About 50% of Taurus syslog entries consist of 4-letter words and on average, each syslog entry consists of 10 words. Therefore, choosing an 8-letter hash key reduces the length of Taurus syslog entries' `message` field to 20% of its original length. However, as shown in Figure 5.5, due to the importance of metadata for syslog analysis this information must be kept. Thus, in practice encoding Taurus syslog

and captured by syslog facility.

[3]Also known as principle of factor sparsity, 80/20 rule and the law of the vital few.

[4]In some studies the Pareto principle holds true [5, 26].

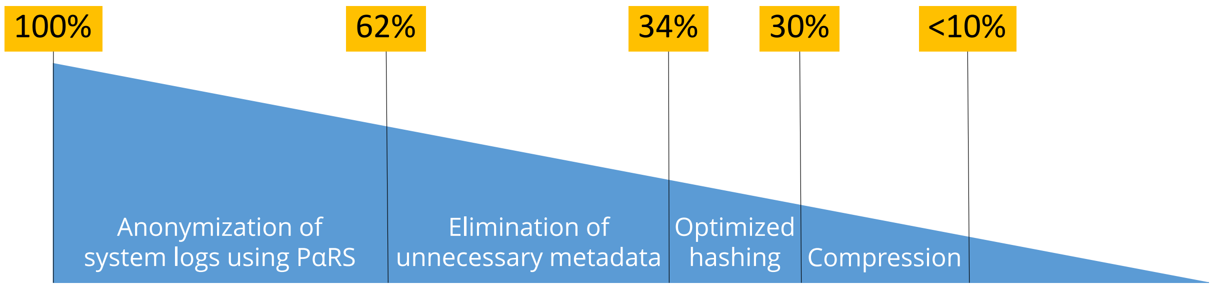

100%
62%
34%
30%
<10%
Anonymization of system logs using PαRS
Elimination of unnecessary metadata
Optimized hashing
Compression

Figure 5.5: Different levels of size reduction using PαRS.

entries using PαRS for the purpose of this work reduces the length of entries up to 44% of their original length[5]. The encoded data are ready to be directly used for anomaly detection without further decoding.

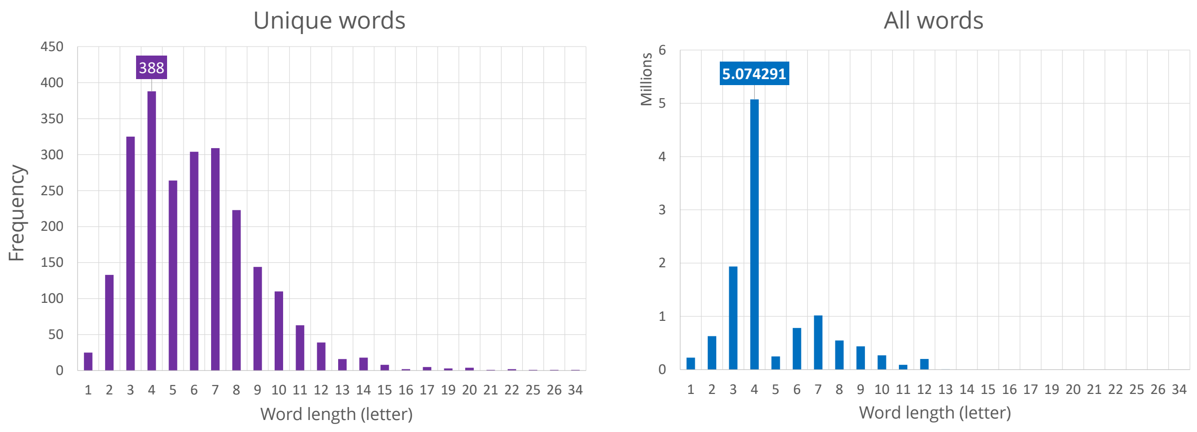


Figure 5.6: Distribution of words in syslog entries based on their size and frequency

Noise Mitigation

The proposed noise mitigation mechanism works based on majority voting among the members of the chosen node vicinity. Therefore, the accuracy of noise mitigation is directly influenced by the correct selection of node vicinity. The recommended node vicinities for anomaly detection are hardware architecture and physical location vicinities. In Taurus, computing nodes of each rack (18 nodes) are physically collocated and are equipped with similar processor architecture. Thus, these computing nodes are member of both hardware architecture and physical location vicinities.

The syslog generation pattern of 8 nodes within a single rack is shown in Figure 5.7. The color of each cell represents the number of syslog entries generated during 24 hours. Darker colors indicate higher number of syslog entries. On each day, the syslog generation pattern of the majority of the nodes indicates the normal pattern for that day. Furthermore, the sudden changes in this pattern in comparison to previous days may indicate anomalous behavior.

[5]An ascii character (letter) can be stored in a single byte.

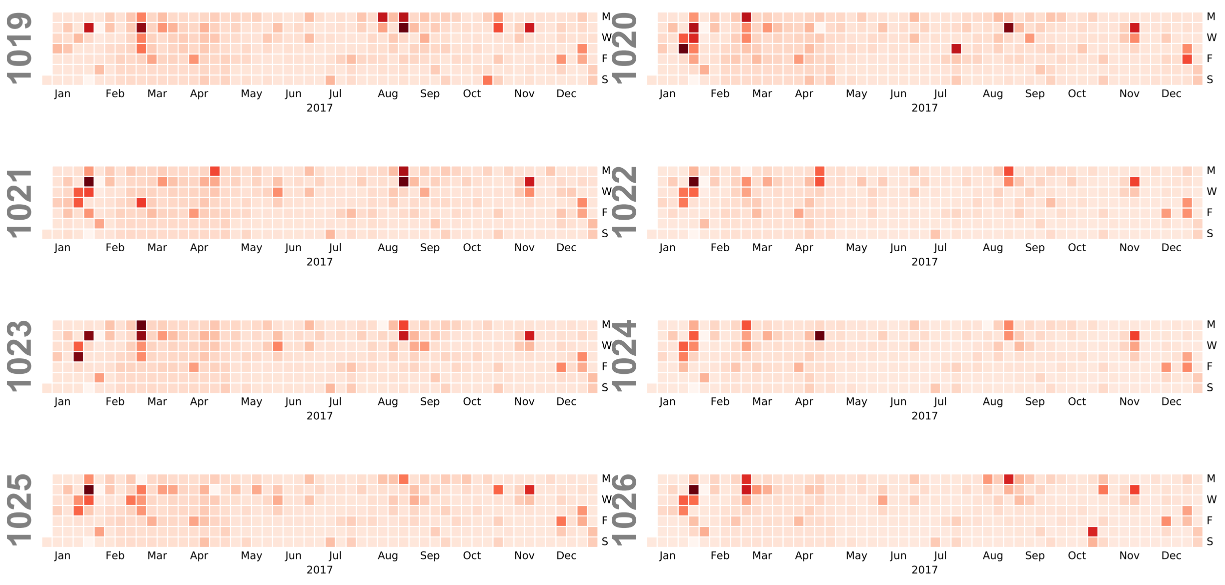


Figure 5.7: Similar patterns of syslog generation by various nodes within the rack 2 of Island 1 in Taurus. Each cell shows the number of entries generated per day during the year 2017. Darker colors indicate higher number of syslog entries.

In summary, most noises caused during the collection of system logs can be mitigated via majority voting within node vicinities. The majority voting must be performed within a sliding time windows of 20 seconds[6] in order to suppress network congestion side effects (refer to Figure 3.19). Using majority voting on Taurus, on average noises up to 20% of the input data could be accurately mitigated[7].

5.2 System-wide Failure Patterns

Existence of system-wide temporal and spatial patterns among failures greatly contributes to identifying root causes of failures and predicting upcoming failures. Therefore, the time and location of all potential failures of Taurus were analyzed. The temporal pattern of potential Taurus failures in the year 2017 is shown in Figure 5.8.

In Figure 5.8 the bottom most diagram illustrates the timeline of all potential node failures on Taurus during 2017. The red and purple diagrams show the multiple node failures and single node failures respectively. As shown in Figure 5.8, no system-wide temporal patterns exist among the node failures. However, according to Figure 3.5 on page 29 the number of failures significantly increases during the working hours (Monday to Friday from $08:00$ to $17:00$).

Figure 5.9a shows the location of failures occurred in the year 2017. The horizontal axis denotes the node ID of each failure as its location. Node IDs are ordered based on their

[6]This should not being mistaken by the 30-minute time window for calculating the SG parameter in Section 4.3.1. Majority voting in the 20-second time window only mitigates the network congestion.

[7]In theory it is possible to mitigate noises up to 49% of the input data however, due to node failures and irregular activities on computing nodes it is reduced to 20%.

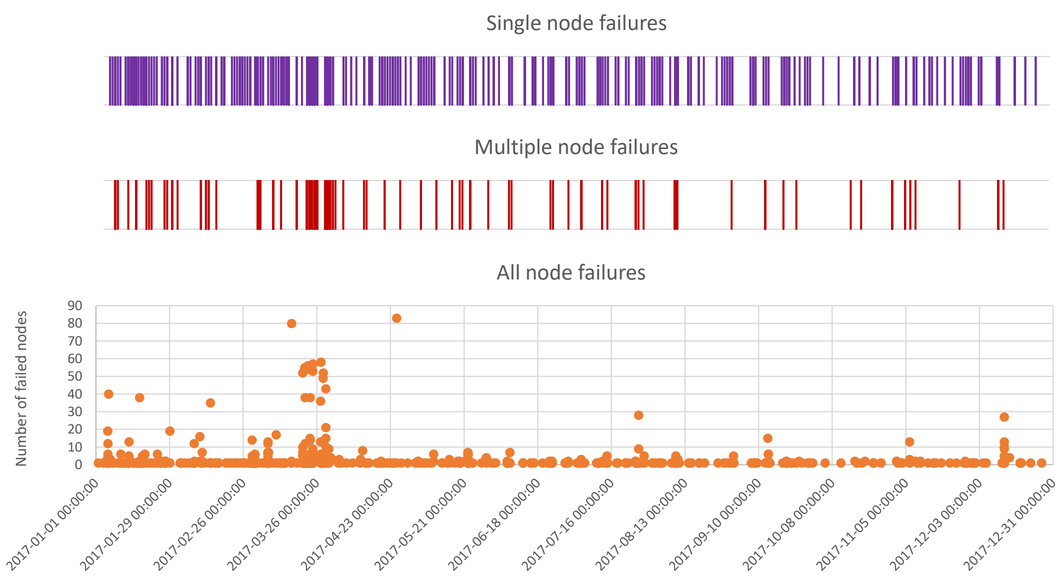
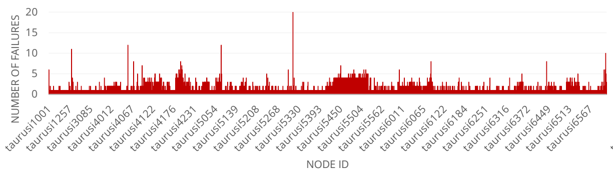
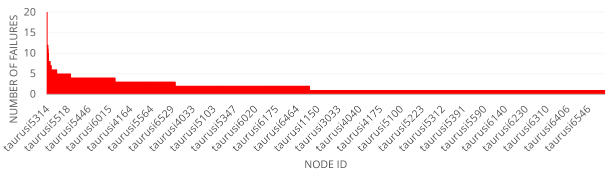


Figure 5.8: Timeline of Taurus potential failures occurrence in the year 2017



(a) Node IDs sorted by physical location



(b) Node IDs sorted by number of failures

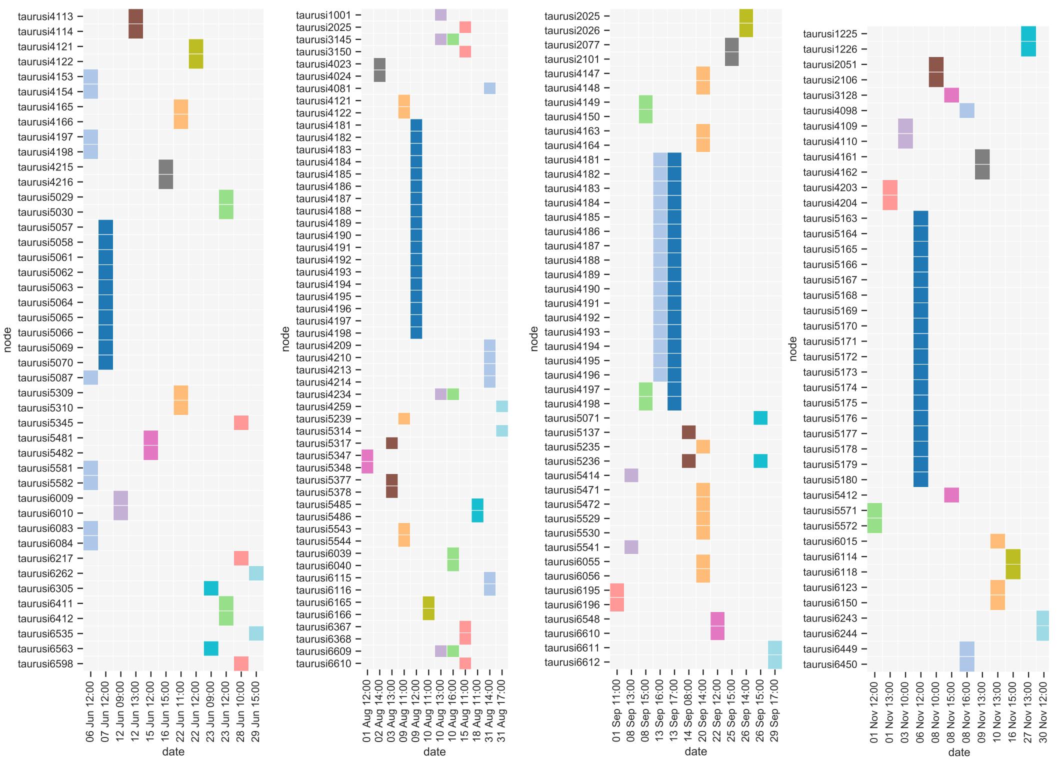
Figure 5.9: Spatial distribution of potential failures occurred in Taurus in the year 2017

physical location in each island. According to Figure 5.9a there are no system-wide spatial patterns among the potential node failures on Taurus.

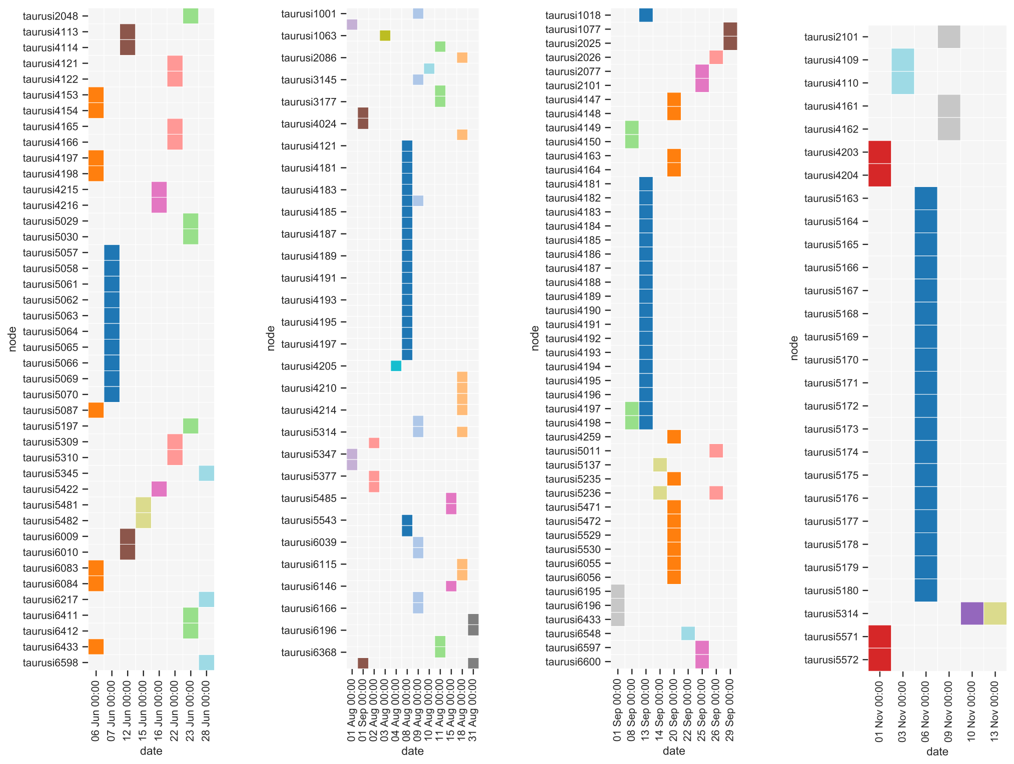
However, sorting the node IDs based on the number of failures on each node, reveals an exponential trend. The node IDs on the horizontal axis of Figure 5.9b are rearranged according to the number of failures on each node. According to the exponential trend of node failures in Figure 5.9b it is concluded that failures are more likely to happen on nodes that already experienced more failures. Therefore, the probability of failure occurrence on a node is in proportion to the number of previous failures on that node.

5.3 Failure Correlations

Failure correlation in contrast to system-wide failure patterns (Section 5.2) refers to correlations among failures in smaller and not necessarily collocated subsets of computing nodes in the HPC system. As proposed in Section 4.2.1, the probability of observing such correlations in node vicinities is significantly higher than other subsets of computing nodes. Therefore, Taurus node and job failures inside the recommended node vicinites, namely hardware architecture vicinity and physical location vicinity were analyzed. The most important temporal/spatial failure correlations detected on Taurus in June, August, September and October 2017 are shown in Figure 5.10a and 5.10b.



(a) Node failures



(b) Job failures

Figure 5.10: Temporal/spatial correlations of Taurus failures. Each column indicates a failure. In each subplot correlated failures are marked with identical colors.

Three important observations were made via analyzing temporal/spatial correlations of Taurus node/job failures. These observations are used to perform root cause analysis on node failures.

Observation 1: On Taurus, if a temporal correlation is being detected only between two compute nodes, even if both failures occurred in the exact same moment, most likely their simultaneous occurrence is just a coincidence. A strong temporal correlation between failures occurring on three or more compute nodes, in many cases, implies an identical failure reason.

Observation 2: Complementary to *Observation 1*, it is learned that on Taurus, when the number of correlated failures is relatively high, the spatial correlation dominates. When inferring correlations and analyzing reasons of failures, the highest priority should be given to spatial correlations followed by temporal correlations. The chances of finding the same reason for spatially correlated failures are higher than in the case of temporally correlated failures.

Observation 3: On Taurus, the combination of temporal and spatial correlations is highly revealing. In situations in which both strong temporal and spatial correlations are observable, the reason behind the failure is identical. This lesson can reveal the logical correlation of failures in situations which the logical correlation is not independently detectable.

5.4 Taurus Failures Statistics

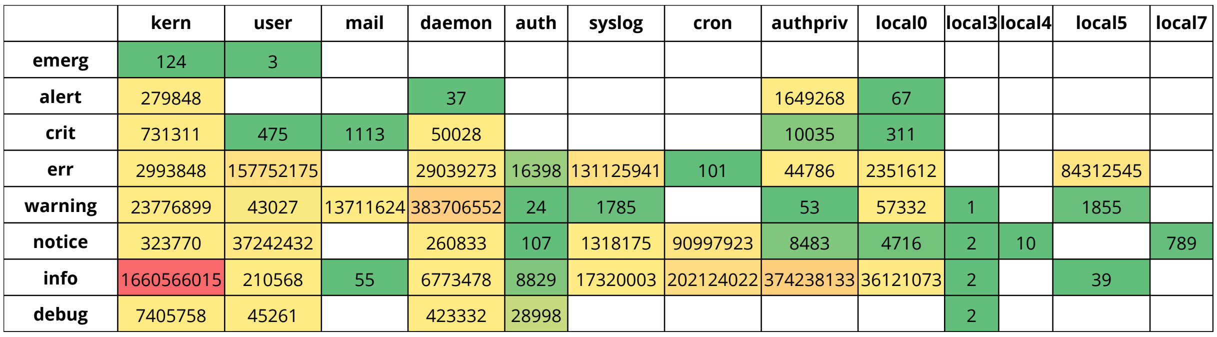
During the year 2017 in total $2,535$ regular node failures on Taurus were recorded. These $2,535$ node failures occurred at 878 unique timestamps[8]. The entire collection of syslog entries was analyzed using the proposed anomaly detection method. The root cause of false negative and false positive cases, in comparison to the known ground truth, were manually analyzed. Comparing the outcome of manual root cause analysis and the known ground truth, failures were divided into two groups of unpredictable and predictable. The syslog metadata fields of `facility` and `severity` were chosen as the main parameters for anomaly detection. Significant changes in the frequency of syslog messages generated by a certain `facility` at a certain `severity` level are considered as signs of failures. Figure 5.11 shows the distribution of syslog entries according to different combinations of `facility` and `severity` values[9]. The `facility` columns which did not have any relevant syslog entry were omitted from Figure 5.11 to improve the readability[10].

In Figure 5.11 from left to right and from top to bottom, the significance of syslog entries decreases. Therefore, the most significant syslog entries are entries generated by `kern` (kernel) facility with the severity level of `emerg` (emergency).

[8]The precision of timestamps is 1 second.

[9]In syslog standard, there are 24 facilities and 8 severity levels. Therefore, there are 192 possible combinations. However, in practice syslog entries of each HPC system only contain a subset of these combinations.

[10]In addition, 65 syslog entries were broken, thus removed from the rest of analysis.



	kern	user	mail	daemon	auth	syslog	cron	authpriv	local0	local3	local4	local5	local7
emerg	124	3											
alert	279848			37				1649268	67				
crit	731311	475	1113	50028				10035	311				
err	2993848	157752175		29039273	16398	131125941	101	44786	2351612			84312545	
warning	23776899	43027	13711624	383706552	24	1785		53	57332	1		1855	
notice	323770	37242432		260833	107	1318175	90997923	8483	4716	2	10		789
info	1660566015	210568	55	6773478	8829	17320003	202124022	374238133	36121073	2		39	
debug	7405758	45261		423332	28998					2			

Figure 5.11: Number of syslog messages collected on Taurus based on their facility (columns) and severity (rows) level.

Table 5.1: Distribution of node failures in Taurus islands

Island	Node failures	Percentage	Nodes in island	Average failure per node
1	63	2.49%	270	0.23
2	27	1.07%	108	0.25
3	79	3.12%	180	0.43
4	511	20.16%	264	1.93
5	1017	40.12%	612	1.66
6	838	33.06%	612	1.36

The distribution of $2,535$ Taurus node failures among 6 islands of Taurus is shown in Table 5.1. The highest number of node failures have occurred in Island 5 and 6 which is due to the higher density[11] of nodes in these 2 islands. However, on average, nodes in Island 4 have the highest failure probability. From another perspective, nodes in islands with higher density are more prone to failures.

Node failures are side effects of the interplay of multiple parameters. Islands with higher number of nodes have higher complexity. Furthermore, larger islands attract larger parallel jobs. User activities directly influence the HPC systems behavior. Therefore, the combination of serving more users and executing larger parallel jobs in more complex computation environment explains the higher probability of failures observed on larger Taurus islands.

The anomaly detection method identifies abnormal behaviors and failures via tracing anomalous trends of the monitoring parameter among syslog entries. Therefore, sudden failures such as power outage or throttling via overheating protection mechanism that do not leave any footprints in syslog entries cannot be individually detected. The distribution of Taurus node failures as well as unpredictable and predictable failures are shown in Figure 5.12.

[11]Number of nodes in each island.

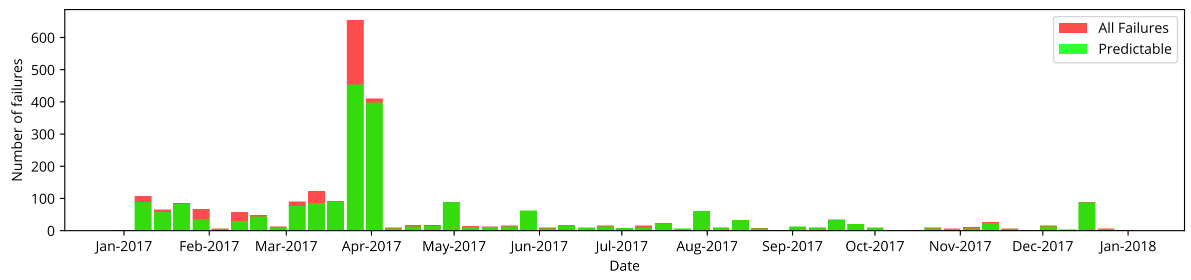


Figure 5.12: Distribution of predictable and unpredictable Taurus node failures over time

Unpredictable Failures

Out of the total number of $2,535$ node failures on Taurus in the year 2017, syslog analysis revealed that 428 node failures were (individually[12]) unpredictable. In another word, the normal behavior of these 428 failures according to the chosen monitoring parameters[13] did not change prior to the failure occurrence. These 428 unpredictable failures occurred on 373 different nodes. The distribution of unpredictable node failures among 6 islands of Taurus is shown in Table 5.2. According to Table 5.2 it is concluded that 16.88% of all node failures on Taurus are unpredictable. The right most column in Table 5.2 shows the percentage of unpredictable failures within each island. According to this measure, more than one fourth of node failures on Island 4 are unpredictable.

Table 5.2: Distribution of unpredictable node failures in Taurus islands

Island	Unpredictable failures	Percentage	Percentage of unpredictable failures in the island
1	9	2.10%	14.28%
2	3	0.70%	11.11%
3	5	1.17%	6.32%
4	135	31.54%	26.41%
5	171	39.95%	16.81%
6	105	24.53%	12.52%

In summary, Island 4 beside having the highest ratio of node failures, has the highest percentage (26.41%) of unpredictable failures. Although Islands 5 and 6 are identical in the number of nodes and processor architecture, the behavior of nodes in Island 6 are significantly more deterministic than Island 5. In current conditions, the highest possible recall rate of failure prediction on Taurus (system-wide) is 83.12%.

Predictable Failures

The remaining $2,107$ node failures on Taurus are predictable. Each of these failures, prior of their occurrence, leave certain footprints in system logs which differs from the normal syslog trends. Therefore, upon detection of these anomalous behavior via syslog analysis, the related upcoming failures are predictable.

[12]Certain failures such as defect in cooling system can be predicted via monitoring other nodes of the vicinity.
[13]`facility`, `severity` and `message` of syslog entries.

To perform the anomaly detection, the combination of two syslog metadata namely: `facility` and `severity` levels were used as the monitoring parameter. Analysis revealed that up to 8 combinations of syslog `facility`-`severity` levels may simultaneously project abnormal trends before a failure. Higher number of simultaneous abnormal pairs significantly increases the accuracy of failure prediction. Table 5.3 shows all failures in the year 2017, that before their occurrence exactly 7 `facility`-`severity` pairs projected abnormal behavior. These 9 failures could have been detected via any of the 7 pairs shown in the third column. The right most column shows the pair that projected most abnormal trend, thus proposing the best pair to be used for anomaly detection.

Table 5.3: Observation of abnormal trends in `facility`-`severity` pairs. These 9 failures could have been detected via any of the 7 pairs shown in the third column. The right most column shows the pair that projected most abnormal trend, thus the best pair to be used for anomaly detection.

Failure time	Node	Pairs showing abnormal trends	Most abnormal trend
2017-01-13 14:01:01	3165	kern_err, kern_warning, kern_notice, kern_info, kern_debug, user_err, user_notice	user_err
2017-03-16 10:50:01	5451	kern_err, kern_warning, kern_info, kern_debug, user_err, user_notice, user_info	user_err
2017-03-16 10:50:01	5468	kern_crit, kern_err, kern_warning, kern_info, kern_debug, user_err, user_notice	user_err
2017-03-16 10:50:01	5486	kern_crit, kern_err, kern_warning, kern_info, kern_debug, user_err, user_notice	user_notice
2017-03-16 10:50:01	5505	kern_err, kern_warning, kern_info, kern_debug, user_err, user_notice, user_info	user_notice
2017-03-24 07:29:41	4056	kern_notice, kern_info, kern_debug, user_err, user_notice, user_info, user_debug	user_debug
2017-06-23 08:30:01	6563	kern_warning, kern_notice, kern_info, kern_debug, user_notice, user_info, user_debug	kern_info
2017-08-31 15:10:40	6196	kern_warning, kern_notice, kern_info, kern_debug, user_err, user_info, user_debug	kern_info
2017-10-26 14:14:03	6450	kern_err, kern_warning, kern_notice, kern_info, kern_debug, user_err, user_notice	kern_warning

Table 5.4 shows a summary of `facility-severity` pairs that project abnormal trends before a failure. The *abnormal values* shown in the header of each sub-table indicates the number of `facility-severity` pairs that could have been used to detect the upcoming failure. The `facility-severity` pairs shown in each sub-table are those combinations which had the highest amount of derivation from the normal behavior. As shown in Table 5.4, prior to the appearance of the majority of node failures, at least 2 parameters simultaneously projected anomalous behavior. Figure 5.13 illustrates the dominant parameters shown in Table 5.4.

Table 5.4: List of detected anomalies in Taurus system logs

Abnormal values	**1**	Abnormal values	**2**	Abnormal values	**3**	Abnormal values	**4**
kern-crit	1	kern-crit	27	kern-crit	6	kern-crit	9
kern-err	1	kern-err	6	kern-err	6	kern-err	5
kern-info	105	kern-info	60	kern-info	24	kern-info	25
kern-notice	0	kern-notice	0	kern-notice	0	kern-notice	0
kern-warning	6	kern-warning	17	kern-warning	31	kern-warning	22
user-debug	0	user-debug	0	user-debug	0	user-debug	0
user-err	59	user-err	211	user-err	131	user-err	75
user-notice	143	user-notice	497	user-notice	178	user-notice	73
Total	**315**	Total	**818**	Total	**376**	Total	**209**
Abnormal values	**5**	Abnormal values	**6**	Abnormal values	**7**	Abnormal values	**8**
kern-crit	5	kern-crit	0	kern-crit	0	kern-crit	0
kern-err	5	kern-err	1	kern-err	0	kern-err	0
kern-info	36	kern-info	1	kern-info	2	kern-info	1
kern-notice	2	kern-notice	0	kern-notice	0	kern-notice	0
kern-warning	23	kern-warning	1	kern-warning	1	kern-warning	0
user-debug	0	user-debug	0	user-debug	1	user-debug	0
user-err	139	user-err	45	user-err	3	user-err	0
user-notice	75	user-notice	46	user-notice	2	user-notice	0
Total	**285**	Total	**94**	Total	**9**	Total	**1**

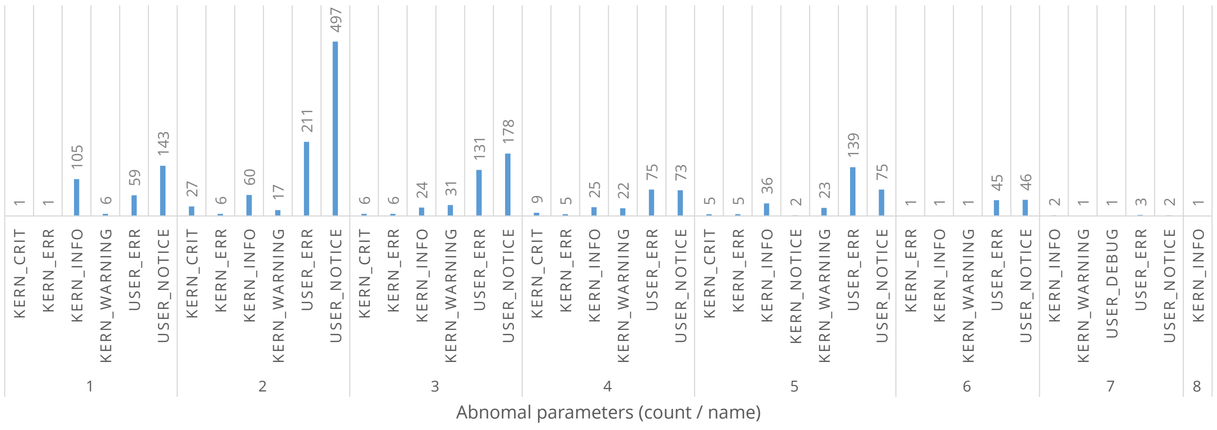


Figure 5.13: Detected anomalies in Taurus system logs

Prediction Lead Time

Employing each of the `facility-severity` combinations of system logs metadata provides a different prediction lead time. Longer lead times provide better opportunities for preventing failures propagation and activating protection and recovery mechanisms. However, predictions with shorter lead time provide higher accuracy, thus results in less false positives[14]. Figure 5.14 shows the ranges of prediction lead time according to each of the 8 significant `facility-severity` combinations.

In summary, the most accurate failure predictions with useful lead time are achieved via analyzing system logs generated by `kernel` facility with the `severity` level of `critical`. The longest prediction lead time is achieved via analyzing system logs generated by `kernel` facility with the `severity` level of `error`. Although in few cases system logs generated by

[14]Mistakenly evaluate the normal behavior as abnormal.

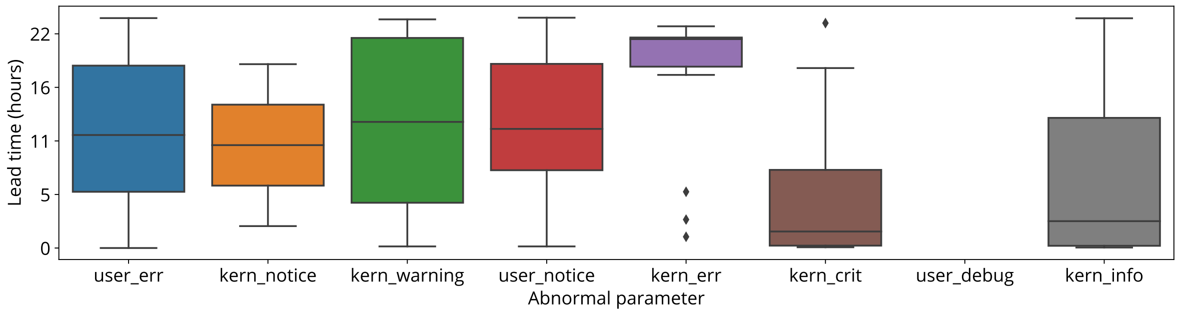


Figure 5.14: Prediction lead time based on various combinations of syslog parameters

`kernel` facility with the `severity` level of `notice` and the system logs generated by `user` facility with the `severity` level of `debug`[15] could be used to predict upcoming failures, in general they are not recommended due to their rareness and insignificant semantics.

5.5 *Jam-e Jam* Prototype

The current implementation of *jam–e jam*'s prototype contains all main building blocks that are shown in Figure 3.1. The prototype is implemented in Python[16] using various libraries[17] for visualization (e.g., seaborn, Matplotlib, Plotly), statistical analysis (e.g., NumPy, SciPy, Pandas) and machine learning (e.g., TensorFlow, Keras, NLTK, NuPIC). The entire implementation will be accessible on Logalyst[18] web page.

The prototype was implemented solely as a proof of concept and for the purpose of this work. Therefore, advanced programming techniques, complex structures and all complexities that could reduce the readability of codes were intentionally avoided. Consequently, the algorithms behind each script can be better understood.

Visualization

To increase the self-descriptively of algorithms and proposed methods, various visualization techniques were applied. The input/output of multiple intermediate steps were also visualized to assist the better understanding of proposed methods. Example 5.1 shows a sample of *jam–e jam*'s final output. Each row consists of four values, namely `Node ID`, `Probability`, `Confidence` and `Updated`. Each row predicts the `Probability` of failure occurrence on a certain `Node ID` with a certain `Confidence`. The `Updated` field shows the timestamp of last assessment.

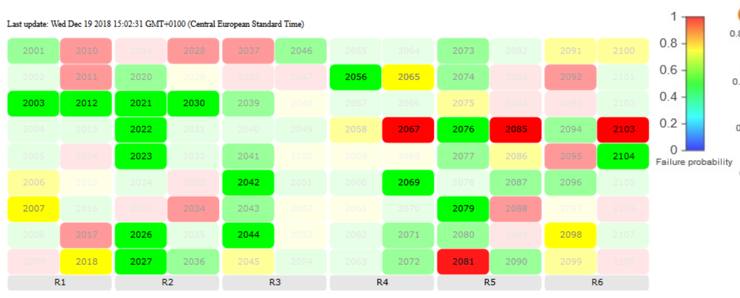
Various visualization techniques were tested to demonstrate the output of *jam–e jam* in an effective and intuitive form. Out of which, two demonstrations were chosen as the most intuitive forms for visualizing the output of *jam–e jam*. Both visualizations use color codes

[15]The `debug` messages are not used in production mode.
[16]https://www.python.org/
[17]https://wiki.python.org/moin/NumericAndScientific
[18]https://logalyst.github.io

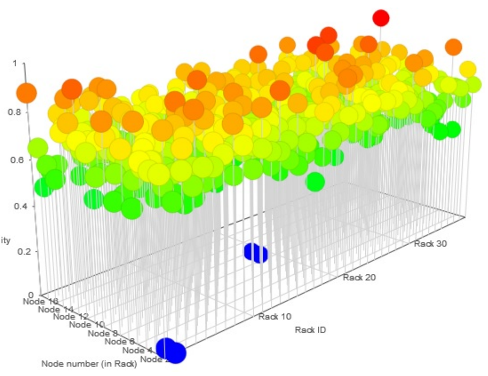
to indicate the probability of failures and different levels of transparency to indicate the confidence level. The first visualization approach shown in Figure 5.15a groups nodes into racks and provides additional information in each cell. The second visualization approach, shown in Figure 5.15b, provides a single image of the entire HPC system. *Jam–e Jam* uses the single image visualization approach as the default output and switches to the rack-based visualization for monitoring smaller regions of the HPC cluster.

Example 5.1: Sample of *Jam–e Jam* final output

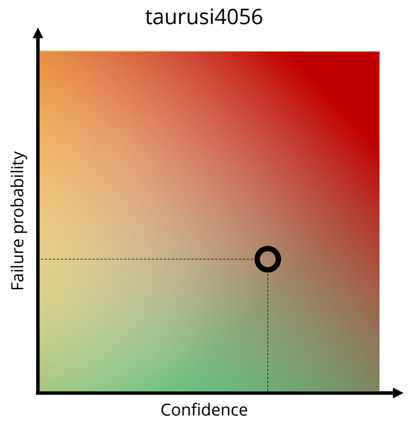
```
Node ID         Probability    Confidence    Updated
-----------     -----------    ----------    -----------
taurusi3001     0.3            0.4           14:23:45
taurusi3002     0.2            0.6           14:23:53
taurusi3003     0.7            0.3           14:22:31
taurusi3004     0.3            0.4           14:23:45
taurusi3005     0.5            0.6           14:24:10
taurusi3006     0.8            0.7           14:13:04
taurusi3007     0.4            0.5           14:21:27
```



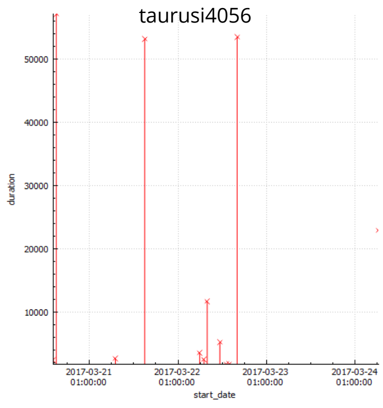
(a) Island view



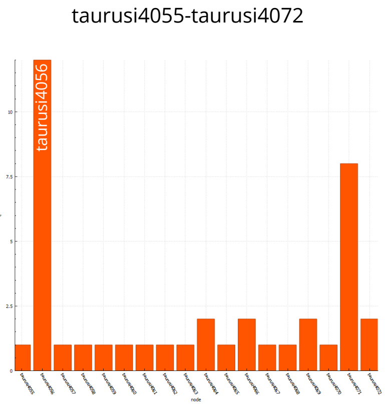
(b) Cluster view



(c) Probability and confidence



(d) Failures history of node



(e) Failures history in vicinity

Figure 5.15: Visualization of failure prediction in *Jam–e Jam*

Jam–e Jam provides additional information that are intended to help administrators in taking adequate decisions during doubtful conditions. Figure 5.15c shows a single node view, illustrating the relation of failure probability and the confidence of a failure prediction. Figure 5.15d provides the history of previous failures on a particular node including the duration of each failure. Figure 5.15e provides the number of previous failures in the relevant node vicinity. Using this comparative information and the history of node failures, administrators can assess the system stability even during uncertain conditions.

Precision and Recall

The current implementation of *ȷam–e ȷam* only using the majority voting module predicts node failures with a precision of 62%. It is important to note that the majority voting approach predicts the upcoming node failures using an exceptionally short history interval of 30 minutes. Increasing the history interval to 24 hours and coupling the majority voting with the HTM model increases the precision of failure prediction to 85%. Table 5.5 shows the detailed outcome of predictions made using majority voting and HTM model on Taurus syslog entries collected in the year 2017. Table 5.6 summarizes the precision, recall, and F_1Score of both approaches.

Table 5.5: Predictions made using majority voting and HTM model

			# number of events
Normal events	Correctly predicted as normal event (TN)		>3.2 billion
	Wrongly predicted as failure (FP)		351
Failure events	Correctly predicted as failure (TP)		2034
	Wrongly predicted as normal event (FN)	Unpredictables	428
		Undetected	73

Table 5.6: Results of applying the proposed failure prediction methods on Taurus syslog entries.

Prediction method	Input data	Precision	Recall	F_1Score
Majority voting	Fully anonymized Syslog entries	62.3%	83.1%	71.2%
Majority voting $+$ HTM	Fully anonymized Syslog entries	85.2%	80.2%	82.6%

5.6 Summary and Discussion

The results indicate that, in general, certain failures in HPC systems are predictable. The most important findings and limitations of this work are summarized and discussed in this section.

System Logs as Main Monitoring Data

The choice of monitoring data is a fundamental decision. This work uses system logs as the main source of monitoring data. However, other existing monitoring data can be also used for similar purposes. The main advantage of system logs over other monitoring data is their availability and generality. Almost every computing system generates certain form of system logs. The standardization of syslog entries format via RFC5424 and the widespread use of Linux kernel in HPC systems made syslog the only generic monitoring facility that uniformly exists on all major HPC systems regardless of their underlying hardware and software stack. Furthermore, other monitoring data can be simply redirected to be collected and stored by syslog facility. Therefore, an analyzer which is able to process syslog entries is also able to handle other monitoring data and can be universally applicable to other computing systems.

Noises in Monitoring Data

As long as failures occur in HPC systems, noises are integral part of monitoring data regardless of the chosen source of monitoring data. Noises are introduced by errors in various components of the data collection mechanism from the producers to the storage. The proposed approach in this work is noise-tolerant and in various stages of the workflow noises are mitigated.

Statistical Analysis

During the early stages of syslog analysis in this work, various additional methods such as natural language processing were tested. Although, these methods could provide additional information in certain cases, the exposed overhead caused by complexity of natural language processing algorithms was not justifiable. Furthermore, through text analysis it has been revealed that despite sudden changes of syslog entries after each software update and the unstructured free-form message field of syslog entries, there are only a few hundred syslog patterns. These patterns can be detected and learned, thus imposing excessive overheads through the use of complex methods is avoided.

Extraction of syslog patterns itself is a challenge. Several previous studies extracted the syslog patterns from the software source code. Although, this seems to be the easiest and most accurate approach, many parts of the software stack are closed-source, or they change frequently. Thus, the syslog patterns can not be directly extracted from the software source code. To address this challenge and extract the syslog patterns (event patterns) PαRS has been designed.

Comparison of the information gained via analyzing event patterns and analyzing original syslog entries revealed that except for certain hardware related messages (e.g., CPU temperature) there are no significant differences. Therefore, it was concluded that only knowing the type of syslog entries (regardless of their variable values) is sufficient to track nodes behavior. In another word, the required information to track nodes behavior is the relation between events (e.g., first `logged in` then `logged out`) rather than the details of each event (e.g., first `siavash logged in to 192.168.0.1` then `siavash logged out from 192.168.0.1`).

Furthermore, one of the goals of this study was to propose a general approach which can be also applied to future Exascale HPC systems. Therefore, automation and scalability of the approach was one of the main priorities. Knowing the insignificance of event details and the demand for automation and scalability, the statistical analysis has been chosen as the main analysis method.

Data Anonymization

Following the implementation of $GDPR$ since May 2018 the processing and storage of syslog entries are allowed only for a short amount of time and after proper de-identification of the personal identifiers. Anonymization of syslog entries according to the privacy policy of computing centers and the general data protection regulations significantly reduces the intelligible semantics of syslog entries. Majority of syslog entries lose their entire semantics. To guarantee the data anonymization, an additional encoding (hashing) step was added to PαRS, transforming it to a comprehensive classifier/anonymizer.

PαRS converts each syslog entry into a fixed size hash key. The hash key is an irreversible encoded form of the event pattern. The temporal relation among events will be kept also among the hash keys. The encoding step decreases the size of syslog messages up to the 30% of their original size. Furthermore, the final encoding step prevents any potential leakage of information into the anonymized output.

Due to the existence of highly frequent entries in syslog collections, the same level of size reduction can be achieved via applying lossless compression algorithms. However, the main advantage of size reduction using PαRS is the irreversible compression which still preserves the relation of events and eliminates the need for decoding entries. Therefore, the anonymized and encoded syslog entries can be archived and processed at any time without requiring further data preparation. The PαRS mechanism is specially useful in scenarios that data must leave the secure zone of HPC system for long-term storage, further analysis, or publications.

Correlations and Node Vicinities

About one third of failure incidents on Taurus are temporally correlated. This ratio rises to about 80% by considering the number of individual node failures rather than the number of failure incidents (Figure 3.25 on page 58). Therefore, this is a necessity to consider failure correlations in behavioral analysis in order to prevent large scale failures.

The main anomaly detection method proposed in this study works solely based on statistical analysis. Its accuracy with only a 30-minute stream of syslog entries is already acceptable. This outstanding performance is rooted in the characteristics of HPC systems. Preliminary results revealed strong correlations among certain node failures. The correlations were further investigated in 3 dimensions namely time, space and logic. Although, many correlations could be explained using these 3 dimensions, certain correlations were observed however, could not be justified (Figure 3.20). Therefore, new dimensions (node vicinities) were defined. The application of *ȷam–e ȷam* in various node vicinities was tested. The best performance was achieved inside hardware architecture and physical location vicinities.

According to the results of this work, similar computing nodes intend to project similar behaviors. Users prefer homogeneous set of nodes to bypass potential incompatibilities and technical challenges. Most batch schedulers are configured to allocate tasks of a job to nearby computing nodes since communication between neighboring nodes is faster, shared resources are better accessible, network congestion is less and many other reasons. Therefore, the placement of computing nodes inside HPC systems and their connections to each other are not random phenomena, rather a thoroughly thought decision. In current HPC systems, nodes with similar characteristics are intentionally placed next to each other to avoid various technical difficulties and improve performance. Because of these considerations the hardware architecture and physical location vicinities in Taurus have large overlaps. Similar conditions apply to other HPC systems. Therefore, it is enough to apply *ȷam–e ȷam* on a rack-based setup in almost all current HPC systems (without knowing the detail of system's topology) to utilize the benefits of hardware architecture and physical location vicinities. However, going toward heterogeneous computing systems the importance of performing behavioral analysis inside hardware architecture and physical location vicinities significantly increases.

Failures Propagate

Due to the shared resources in HPC systems and the tightly coupling of computing nodes, failures may propagate through the system from one node to the other. Most propagated failures are caused by errors in distributed file system and I/O mechanisms. Stabilizing the distributed file system eliminates a significant fraction of node failures in HPC systems.

The Solution for Exascale

Unstructured format of syslog messages, continuous updates of software stack, frequent system maintenance, dynamic algorithms, adaptive load balancing, performance/energy optimization mechanisms, unpredictable users behavior, nondeterministic protection mechanisms and interactive computational environment are some factors that frequently affect the behavior of HPC systems. Therefore, a static normal behavior cannot be defined. Instead, the normal behavior should be dynamically adjusted to the current status of the computing system. Given the size and complexity of future Exascale computing systems unsupervised approaches are the only feasible solution. The unsupervised behavioral analysis approaches based on statistical methods provide more scalability and transparency and do not require the complete semantics of the syslog entries. The later is required to perform behavioral analysis using anonymized system logs.

Heterogeneity and Homogeneity of Nodes

Differences among available memory, disk space and network capacity of computing nodes have minimum impact on nodes' behavior in Taurus. On the other hand, CPU architecture significantly influences nodes' behavior. Therefore, the heterogeneity and homogeneity of computing nodes in this work are defined according to the nodes CPU architecture. The failure detection approach proposed in this work, performs more accurately on homogeneous HPC systems. Thus, this study recommends dividing heterogeneous HPC systems into smaller homogeneous sections before applying the proposed failure detection approach.

It is important to mention that this work is focused on the resiliency of large HPC clusters. Given the limited number of available different CPU architectures, it is a valid assumption to consider any large heterogeneous computing system as a set of smaller homogeneous sections, thus the proposed method is applicable. In the unlikely situation of lack of homogeneous sections in heterogeneous computing systems, the HTM model should be used without majority voting.

Precision and Recall

ɟam–e ɟam predicts Taurus node failures with outstanding F_1Score value of 82%. However, as it is shown in Table 2.7, there are other failure prediction methods that achieved even higher F_1Score values. The main difference lies in the definition of failure and the quality of monitoring data. In contrast to this work, system logs are not the only monitoring data used by the methods of Table 2.7. Some of those methods, beside the syslog entries, had access to additional information that are required to predict irregular failures such as failure of cooling system, CPU burn out, and disk failure. Furthermore, most studies did not provide sufficient information about the data curation and preparation process, which have significant influence on the coverage and precision of final predictions. User interactions also significantly increase the nondeterministic behaviors of HPC systems, thus reducing the predictability of upcoming failures. In addition, certain failures are not predictable due to the lack of any footprints prior to the point of failure e.g., failures caused by sudden reaction of the overheating protection mechanism, or power outage.

The prediction lead time can not be estimated. Although, in several cases the lead time of failure prediction is surprisingly large, its duration itself cannot be accurately estimated. Rather than statistically estimating the expected failure time, *ɟam–e ɟam* increases the value of prediction `confidence` as time passes (Section 4.3.3).

Detection of system-wide failure patterns is not practical. Different subsets of computing nodes project different behaviors, thus different failure patterns. The dynamic nature of the modern HPC systems and in particular the interactive computational environment (users) is causing these differences. Therefore, detecting system-wide failure patterns is not practical nor useful. A system-wide failure pattern will fail to accurately model the failure pattern of subsections of the HPC system. Therefore, the behavior pattern of each subsection should be individually extracted.

Combining the majority voting and the HTM model, the proposed approach reached the outstanding precision of 85% for node failure prediction (Table 5.6). A more reliable data collection mechanism is expected to improve the precision of failure prediction.

To Predict or Not To Predict: On-demand Adaptive Resilience

In total more than $4,000$ incidents were recorded on Taurus during one year. More than $15,000$ node outages occurred during these incidents. Many of these node outages were caused by planned maintenance or urgent security updates (e.g., Spectre and Meltdown). Another group of node outages were caused by hardware failures (e.g., failure of cooling system). The third group of outages were consecutive outages that occurred during the instability periods immediately after recovering from the previous outage (Figure 1.1). The last group of node outages occurred during the normal operation of the HPC system and are caused by internal factors such as software and hardware errors or racing conditions. Only the last group of node outages (regular failures) can potentially be predicted.

During the year 2017 $2,535$ regular failures were recorded on Taurus. Only considering the blackout interval of those $2,262$ nodes that recovered in less than 5 hours, a total amount of $6,175$ node-hours are lost[19]. This amount of lost hours is comparable to shutting down the entire Taurus cluster for more than 3 days[20]. In larger HPC systems the failure ratio may increase even further due to higher system complexities.

It is important to mention that existing failure protection mechanisms (e.g., checkpoint/restart, redundancy) themselves introduce failures in HPC systems. Furthermore, always-active layers of protection impose unnecessary overhead and decrease the HPC systems performance. Switching from an always-active resilience to an on-demand resilience decreases the unnecessary overhead of always-active protection layers and reduces the probability of failures caused by interference of protection mechanisms.

The decision among always-active and on-demand resilience heavily depends on the requirements and demands of the HPC system. There are unpredictable failures and the exact length of prediction lead time is usually unknown. Therefore, on-demand protection mechanisms will fail to protect the system against certain failures. For HPC systems with short jobs the on-demand resilience can be beneficial. On the other hand, always-active resilience is the only solution to compensate irregular failures which may occur on HPC system with long execution time or time-critical jobs.

However, regardless of the HPC system requirements and demands, the on-demand adaptive resilience approach is the solution for protecting future HPC systems. The on-demand adaptive resilience approach provides light-weight always-active protection layers that are strengthened via on-demand activation of additional protection mechanisms in case of detecting abnormal behaviors or predicting instabilities.

[19] Each node has multiple CPUs.
[20] Taurus has 2046 computing nodes.

6 Conclusion and Future Works

Following the expansion of HPC systems in size and complexity, and the advent of Exascale computing systems, the existence of failures became a norm rather than an exception. In the presence of failures, even the most de facto mechanisms such as checkpoint/restart, redundancy, and migration may fail to support the continuous operation of ever growing large scale HPC systems. Predicting upcoming failures reinforces existing mechanisms and enables timely protection via providing adequate lead time. The goal is to provide adaptive resilience for HPC systems. In an ideal scenario all failure protection mechanisms remain inactive during the normal conditions. Consequently, unnecessary overheads and performance penalties are avoided, the energy consumption is reduced, and a potential source of failure is removed. Upon prediction or detection of failures, proportional to the systems condition, adequate protection mechanisms will be activated to prevent failure propagation and compensate potential side effects. The adequate action could be an on-demand check-pointing of the unstable node, an on-demand cloning of the unstable processes or an on-demand migration of tasks to other stable nodes (surrogates).

This work showed that the majority of failures in HPC systems are predictable. To predict the regular failures, *ȷam–e ȷam* as a general and unsupervised behavioral analyzer was proposed and can be applied to other HPC systems without any modifications. It has been shown that system logs have invaluable information about various levels of computing system and is a useful source of monitoring data that exists on virtually all HPC systems. To comply with data protection regulations and to address the users privacy concerns, a robust data anonymization method was introduced. PαRS turned privacy constraints into analysis advantage that further facilitates the storage and processing of anonymized monitoring data.

Furthermore, the concepts of node vicinity and majority voting inside node vicinities were introduced. The proposed behavior analysis method based on majority voting tolerates noise by design, and accurately works with as short as 30 minutes of logging history. The statistical-based anomaly detection inside relevant node vicinities significantly improved the accuracy of failure prediction. The proposed machine learning based methods complemented the decisions of the statistical method to further improve the accuracy of failure predictions. *ȷam–e ȷam* with its outstanding failure prediction precision of 85%, achieved using a fully automatic and unsupervised approach, proved its functionality for the future extreme scale HPC systems.

The goal of this work was designing a prototype to analyze and predict system behavior as a main step toward resilience in high performance computing systems. Due to the size and complexity of the problem, certain assumptions were made to generalize the outcomes. The most important assumptions are explained on page 25. Although, these assumptions were necessary to conduct this study, for future studies several assumptions can be loosened according to the outcomes of the current work.

To provide a general analysis approach applicable to all HPC systems, a general source of data must have been chosen. Currently, system logs are the only widely available monitoring data which exist virtually on all HPC systems. However, it seems that computing centers and system producers are acknowledging the importance of behavioral analysis via providing a broader range of monitoring data sources. The new widespread sources of monitoring data should be added to *ȷam–e ȷam* in order to improve the accuracy of its analysis.

The failure correlation detector in this work automatically detects correlation among failures in two dimensions of time and space. Failure correlations in other dimensions are mainly perceived indirectly via analyzing the temporal and spacial correlations. Therefore, the automatic detection of failure correlation requires further improvements. Detecting correlations among node vicinities is only the first step toward fully automatic detection of system-wide failure correlations.

Selecting the node vicinities is the only manual part of the *ȷam–e ȷam* mechanism and requires the expert knowledge of system topology. Automatic selection of node vicinities eliminates the only manual step, and turns *ȷam–e ȷam* into a fully automatic mechanism. However, as it has been discussed before, in the current HPC systems the automatic selection of node vicinities is not a high priority. Due to the fact that system providers intentionally group homogeneous computing nodes together, in current HPC systems a rack can be considered as a default node vicinity. However, with the arrival of heterogeneous systems the automatic selection of node vicinities remains as an important challenge.

Despite the promising preliminary results of Text Auto-completion model, the technical challenges of detecting very long recurring sequences of symbols in text strings restricts the accuracy of Text Auto-completion model. Therefore, improvement of Text Auto-completion model is postponed as part of the future work. Furthermore, the fine-tuning of HTM model is planned.

As discussed in Section 5.6, the precision of the proposed failure prediction approach in this work cannot be directly compared to previous failure predictors. However, a comparison study is planned to be conducted as soon as the requirements of the study are fulfilled. Application of the proposed failure prediction approach on Taurus as a live service is also planned. At the time of writing, the first prerequisite step (a reliable stream of syslog entries) is already accomplished and further steps are in progress.

Bibliography

[1] G. Moore, “Cramming More Components Onto Integrated Circuits,” *Proceedings of the IEEE*, vol. 86, pp. 82–85, Jan. 1998.

[2] J. Dongarra, P. Beckman, T. Moore, P. Aerts, G. Aloisio, J.-C. Andre, D. Barkai, J.-Y. Berthou, T. Boku, B. Braunschweig, F. Cappello, B. Chapman, Xuebin Chi, A. Choudhary, S. Dosanjh, T. Dunning, S. Fiore, A. Geist, B. Gropp, R. Harrison, M. Hereld, M. Heroux, A. Hoisie, K. Hotta, Zhong Jin, Y. Ishikawa, F. Johnson, S. Kale, R. Kenway, D. Keyes, B. Kramer, J. Labarta, A. Lichnewsky, T. Lippert, B. Lucas, B. Maccabe, S. Matsuoka, P. Messina, P. Michielse, B. Mohr, M. S. Mueller, W. E. Nagel, H. Nakashima, M. E. Papka, D. Reed, M. Sato, E. Seidel, J. Shalf, D. Skinner, M. Snir, T. Sterling, R. Stevens, F. Streitz, B. Sugar, S. Sumimoto, W. Tang, J. Taylor, R. Thakur, A. Trefethen, M. Valero, A. van der Steen, J. Vetter, P. Williams, R. Wisniewski, and K. Yelick, “The International Exascale Software Project Roadmap,” *The International Journal of High Performance Computing Applications*, vol. 25, pp. 3–60, Feb. 2011.

[3] M. Snir, R. W. Wisniewski, J. A. Abraham, S. V. Adve, S. Bagchi, P. Balaji, J. Belak, P. Bose, F. Cappello, B. Carlson, A. A. Chien, P. Coteus, N. A. DeBardeleben, P. C. Diniz, C. Engelmann, M. Erez, S. Fazzari, A. Geist, R. Gupta, F. Johnson, S. Krishnamoorthy, S. Leyffer, D. Liberty, S. Mitra, T. Munson, R. Schreiber, J. Stearley, and E. V. Hensbergen, “Addressing Failures in Exascale Computing,” *The International Journal of High Performance Computing Applications*, vol. 28, pp. 129–173, May 2014.

[4] M. Platini, T. Ropars, B. Pelletier, and N. De Palma, “CPU Overheating Characterization in HPC Systems: A Case Study,” in *2018 IEEE/ACM 8th Workshop on Fault Tolerance for HPC at eXtreme Scale (FTXS)*, pp. 59–68, Nov. 2018.

[5] S. Di, H. Guo, E. Pershey, M. Snir, and F. Cappello, “Characterizing and Understanding HPC Job Failures Over the 2k-Day Life of IBM BlueGene/Q System,” in *2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pp. 473–484, June 2019.

[6] C. Pachajoa, C. Pacher, and W. N. Gansterer, “Node-Failure-Resistant Preconditioned Conjugate Gradient Method without Replacement Nodes,” in *2019 IEEE/ACM 9th Workshop on Fault Tolerance for HPC at eXtreme Scale (FTXS)*, pp. 31–40, Nov. 2019.

[7] H. Casanova, Y. Robert, and U. Schwiegelshohn, “Algorithms and Scheduling Techniques for Exascale Systems (Dagstuhl Seminar 13381),” *Dagstuhl Reports*, p. 122, 2014.

[8] F. Cappello, A. Geist, W. Gropp, S. Kale, B. Kramer, and M. Snir, “Toward Exascale

Resilience: 2014 update," *Supercomputing Frontiers and Innovations*, vol. 1, pp. 5–28, June 2014.

[9] Z. Hussain, X. Cui, T. Znati, and R. Melhem, "CoLoR: Co-Located Rescuers for Fault Tolerance in HPC Systems," in *2018 IEEE 24th International Conference on Parallel and Distributed Systems (ICPADS)*, pp. 569–576, Dec. 2018.

[10] D. Dauwe, S. Pasricha, A. A. Maciejewski, and H. J. Siegel, "Resilience-Aware Resource Management for Exascale Computing Systems," *IEEE Transactions on Sustainable Computing*, vol. 3, pp. 332–345, Oct. 2018.

[11] B. Fang, J. Chen, k. Pattabiraman, M. Ripeanu, and S. Krishnamoorthy, "Towards Predicting the Impact of Roll-Forward Failure Recovery for HPC Applications," in *2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks – Supplemental Volume (DSN-S)*, pp. 13–14, June 2019.

[12] R. A. Ashraf, S. Hukerikar, and C. Engelmann, "Shrink or Substitute: Handling Process Failures in HPC Systems Using In-Situ Recovery," in *2018 26th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*, pp. 178–185, Mar. 2018.

[13] M. A. Amrizal, P. Li, M. Agung, R. Egawa, and H. Takizawa, "A Failure Prediction-Based Adaptive Checkpointing Method with Less Reliance on Temperature Monitoring for HPC Applications," in *2018 IEEE International Conference on Cluster Computing (CLUSTER)*, pp. 515–523, Sept. 2018.

[14] J. C. Duenas, J. M. Navarro, H. A. Parada G., J. Andion, and F. Cuadrado, "Applying Event Stream Processing to Network Online Failure Prediction," *IEEE Communications Magazine*, vol. 56, pp. 166–170, Jan. 2018.

[15] A. Frank, D. Yang, A. Brinkmann, M. Schulz, and T. Süss, "Reducing False Node Failure Predictions in HPC," in *2019 IEEE 26th International Conference on High Performance Computing, Data, and Analytics (HiPC)*, pp. 323–332, Dec. 2019.

[16] A. Gainaru, F. Cappello, M. Snir, and W. Kramer, "Failure Prediction for HPC Systems and Applications: Current Situation and Open Issues," *The International Journal of High Performance Computing Applications*, vol. 27, pp. 273–282, Aug. 2013.

[17] M. Ball and F. Hardie, "Effects and Detection of Intermittent Failures in Digital Systelns," *Proceedings of the AFIPS conference*, vol. 35, pp. 329–335, 1969.

[18] S. Gupta, T. Patel, C. Engelmann, and D. Tiwari, "Failures in Large Scale Systems: Long-Term Measurement, Analysis, and Implications," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis on - SC '17*, (Denver, Colorado), pp. 1–12, ACM Press, 2017.

[19] V. Chandra and R. Aitken, “Impact of Technology and Voltage Scaling on the Soft Error Susceptibility in Nanoscale CMOS,” in *2008 IEEE International Symposium on Defect and Fault Tolerance of VLSI Systems*, pp. 114–122, Oct. 2008.

[20] I. P. Egwutuoha, D. Levy, B. Selic, and S. Chen, “A Survey of Fault Tolerance Mechanisms and Checkpoint/Restart Implementations for High Performance Computing Systems,” *The Journal of Supercomputing*, vol. 65, pp. 1302–1326, Sept. 2013.

[21] M. Prieur, “The Return Rates of the Components,” Tech. Rep. 9, hardware.fr, Oct. 2013.

[22] M. Prieur, “The Return Rates of the Components,” Tech. Rep. 12, hardware.fr, May 2015.

[23] M. Prieur, “The Return Rates of the Components,” Tech. Rep. 15, hardware.fr, Dec. 2016.

[24] J. Lienig and H. Bruemmer, “Reliability Analysis,” in *Fundamentals of Electronic Systems Design* (J. Lienig and H. Bruemmer, eds.), pp. 45–73, Cham: Springer International Publishing, 2017.

[25] W. M. Jones, J. T. Daly, and N. DeBardeleben, “Application Monitoring and Checkpointing in HPC: Looking Towards Exascale Systems,” in *Proceedings of the 50th Annual Southeast Regional Conference*, ACM-SE ’12, (Tuscaloosa, Alabama), pp. 262–267, ACM, 2012.

[26] S. Di, H. Guo, R. Gupta, E. R. Pershey, M. Snir, and F. Cappello, “Exploring Properties and Correlations of Fatal Events in a Large-Scale HPC System,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, pp. 361–374, Feb. 2019.

[27] J. Elliott, K. Kharbas, D. Fiala, F. Mueller, K. Ferreira, and C. Engelmann, “Combining Partial Redundancy and Checkpointing for HPC,” in *2012 IEEE 32nd International Conference on Distributed Computing Systems*, (Macau, China), pp. 615–626, IEEE, June 2012.

[28] J. Daly, B. Harrod, T. Hoang, L. Nowell, B. Adolf, S. Borkar, N. DeBardeleben, M. Elnozahy, M. Heroux, and D. Rogers, “Inter-Agency Workshop on HPC Resilience at Extreme Scale,” in *National Security Agency Advanced Computing Systems*, Feb. 2012.

[29] B. Schroeder and G. A. Gibson, “Understanding Failures in Petascale Computers,” *Journal of Physics: Conference Series*, vol. 78, p. 012022, July 2007.

[30] H. Härtig, S. Matsuoka, F. Mueller, and A. Reinefeld, “Resilience in Exascale Computing,” in *Dagstuhl Reports* (M. Herbstritt, ed.), vol. 4, (Dagstuhl, Germany), pp. 124–139, Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2015.

[31] C. Engelm and F. Lauer, “Facilitating Co-Design for Extreme-Scale Systems Through Lightweight Simulation,” in *2010 IEEE International Conference On Cluster Computing Workshops and Posters*, (HERAKLION, Greece), pp. 1–8, IEEE, Sept. 2010.

[32] C. Engelmann and T. Naughton, “Toward a Performance/Resilience Tool for Hardware/Software Co-design of High-Performance Computing Systems,” in *2013 42nd International Conference on Parallel Processing*, (Lyon, France), pp. 960–969, IEEE, Oct. 2013.

[33] P. Gill, N. Jain, and N. Nagappan, “Understanding Network Failures in Data Centers: Measurement, Analysis, and Implications,” *Proceedings of the ACM SIGCOMM*, vol. 41, pp. 350–361, Aug. 2011.

[34] R. K. Iyer, D. J. Rossetti, and M. C. Hsueh, “Measurement and Modeling of Computer Reliability as Affected by System Activity,” *ACM Transactions on Computer Systems*, vol. 4, pp. 214–237, Aug. 1986.

[35] J. Gray, “Why Do Computers Stop and What Can Be Done About It?,” Technical Report 85.7 PN87614, Tandem Computers, June 1985.

[36] J. Gray, “A Census of Tandem System Availability Between 1985 and 1990,” *IEEE Transactions on Reliability*, vol. 39, no. 4, p. 10, 1990.

[37] D. Tang, R. Iyer, and S. Subramani, “Failure Analysis and Modeling of a VAXcluster System,” in *Digest of Papers. Fault-Tolerant Computing: 20th International Symposium(FTCS)*, (Newcastle Upon Tyne, UK), pp. 244–251, IEEE, June 1990.

[38] T.-Y. Lin and D. P. Siewiorek, “Error Log Analysis: Statistical Modeling and Heuristic Trend Analysis,” *IEEE Transactions on Reliability*, vol. 39, pp. 419–432, Oct. 1990.

[39] R. Iyer, Z. Kalbarczyk, and M. Kalyanakrishnam, “Failure Data Analysis of a LAN of Windows NT Based Computers,” in *Reliable Distributed Systems, IEEE Symposium on(SRDS)*, p. 178, Oct. 1999.

[40] Jun Xu, Z. Kalbarczyk, and R. Iyer, “Networked Windows NT System Field Failure Data Analysis,” in *Proceedings 1999 Pacific Rim International Symposium on Dependable Computing*, (Hong Kong), pp. 178–185, IEEE Comput. Soc, 1999.

[41] D. Oppenheimer, A. Ganapathi, and D. A. Patterson, “Why Do Internet Services Fail, and What Can Be Done About It?,” in *Proceedings of the 4th Usenix Symposium on Internet Technologies and Systems*, p. 15, 2003.

[42] R. K. Sahoo, M. S. Squillante, A. Sivasubramaniam, and Yanyong Zhang, “Failure Data Analysis of a Large-Scale Heterogeneous Server Environment,” in *International Conference on Dependable Systems and Networks, 2004*, pp. 772–781, June 2004.

[43] A. Oliner and J. Stearley, “What Supercomputers Say: A Study of Five System Logs,” in *37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN’07)*, pp. 575–584, June 2007.

[44] A. Moody, G. Bronevetsky, K. Mohror, and B. R. d Supinski, “Design, Modeling, and Evaluation of a Scalable Multi-level Checkpointing System,” in *SC ’10: Proceedings of the*

2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis, pp. 1–11, Nov. 2010.

[45] K. Ferreira, J. Stearley, J. H. Laros, R. Oldfield, K. Pedretti, R. Brightwell, R. Riesen, P. G. Bridges, and D. Arnold, “Evaluating the Viability of Process Replication Reliability for Exascale Systems,” in *SC ’11: Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–12, Nov. 2011.

[46] D. Turner, K. Levchenko, A. C. Snoeren, and S. Savage, “California Fault Lines: Understanding the Causes and Impact of Network Failures,” *Proceedings of the ACM SIGCOMM 2010 conference*, pp. 315–326, 2010.

[47] G. Zheng, X. Ni, and L. V. Kalé, “A Scalable Double in-Memory Checkpoint and Restart Scheme Towards Exascale,” in *IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN 2012)*, pp. 1–6, June 2012.

[48] K. Yamamoto, A. Uno, H. Murai, T. Tsukamoto, F. Shoji, S. Matsui, R. Sekizawa, F. Sueyasu, H. Uchiyama, M. Okamoto, N. Ohgushi, K. Takashina, D. Wakabayashi, Y. Taguchi, and M. Yokokawa, “The K computer Operations: Experiences and Statistics,” *Procedia Computer Science*, vol. 29, pp. 576–585, 2014.

[49] C. D. Martino, Z. Kalbarczyk, R. K. Iyer, F. Baccanico, J. Fullop, and W. Kramer, “Lessons Learned from the Analysis of System Failures at Petascale: The Case of Blue Waters,” in *2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pp. 610–621, June 2014.

[50] X. Ni, *Mitigation of Failures in High Performance Computing via Runtime Techniques*. PhD thesis, University of Illinois at Urbana-Champaign, 2016.

[51] M. Gamell, D. S. Katz, H. Kolla, J. Chen, S. Klasky, and M. Parashar, “Exploring Automatic, Online Failure Recovery for Scientific Applications at Extreme Scales,” in *SC ’14: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 895–906, Nov. 2014.

[52] D. Dauwe, S. Pasricha, A. A. Maciejewski, and H. J. Siegel, “An Analysis of Multilevel Checkpoint Performance Models,” in *2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pp. 783–792, May 2018.

[53] J. Dongarra, T. Herault, and Y. Robert, “Fault Tolerance Techniques for High-Performance Computing,” in *Fault-Tolerance Techniques for High-Performance Computing* (T. Herault and Y. Robert, eds.), Computer Communications and Networks, pp. 3–85, Cham: Springer International Publishing, 2015.

[54] S. Levy, K. B. Ferreira, N. DeBardeleben, T. Siddiqua, V. Sridharan, and E. Baseman, “Lessons Learned from Memory Errors Observed over the Lifetime of Cielo,” in *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis*, SC ’18, (Piscataway, NJ, USA), pp. 43:1–43:12, IEEE Press, 2018.

[55] V. Sridharan, N. DeBardeleben, S. Blanchard, K. B. Ferreira, J. Stearley, J. Shalf, and S. Gurumurthi, "Memory Errors in Modern Systems: The Good, The Bad, and The Ugly," in *Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '15, (Istanbul, Turkey), pp. 297–310, Association for Computing Machinery, Mar. 2015.

[56] J. Meza, T. Xu, K. Veeraraghavan, and O. Mutlu, "A Large Scale Study of Data Center Network Reliability," *Proceedings of the Internet Measurement Conference*, pp. 393–407, Nov. 2018.

[57] V.-P. Ranganath and D. Andresen, "Why do Users Kill HPC Jobs?," in *2018 IEEE 25th International Conference on High Performance Computing (HiPC)*, pp. 276–283, Dec. 2018.

[58] S. Di, Y. Robert, F. Vivien, and F. Cappello, "Toward an Optimal Online Checkpoint Solution under a Two-Level HPC Checkpoint Model," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, pp. 244–259, Jan. 2017.

[59] E. Rojas, E. Meneses, T. Jones, and D. Maxwell, "Analyzing a Five-Year Failure Record of a Leadership-Class Supercomputer," in *2019 31st International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, pp. 196–203, Oct. 2019.

[60] B. H. Park, Y. Hui, S. Boehm, R. A. Ashraf, C. Layton, and C. Engelmann, "A Big Data Analytics Framework for HPC Log Data: Three Case Studies Using the Titan Supercomputer Log," in *2018 IEEE International Conference on Cluster Computing (CLUSTER)*, pp. 571–579, Sept. 2018.

[61] Q. Chen, K. Chen, Z.-N. Chen, W. Xue, X. Ji, and B. Yang, "Lessons Learned from Optimizing the Sunway Storage System for Higher Application I/O Performance," *Journal of Computer Science and Technology*, vol. 35, pp. 47–60, Jan. 2020.

[62] Y. Zhu, Y. Liu, and G. Zhang, "FT-PBLAS: PBLAS-Based Fault-Tolerant Linear Algebra Computation on High-performance Computing Systems," *IEEE Access*, vol. 8, pp. 42674–42688, 2020.

[63] C. Lonvick, "The BSD Syslog Protocol." `https://tools.ietf.org/html/rfc3164`, Aug. 2001. [accessed 13-03-2020].

[64] R. Gerhards, "The Syslog Protocol." `https://tools.ietf.org/html/rfc5424`, Mar. 2009. [accessed 13-03-2020].

[65] W. Xu, L. Huang, A. Fox, D. Patterson, and M. I. Jordan, "Detecting Large-Scale System Problems by Mining Console Logs," in *Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles - SOSP '09*, (Big Sky, Montana, USA), p. 117, 2009.

[66] S. Chhajed, *Learning ELK Stack*. Packt Publishing Ltd, Nov. 2015.

[67] X. Yu, P. Joshi, J. Xu, G. Jin, H. Zhang, and G. Jiang, “CloudSeer: Workflow Monitoring of Cloud Infrastructures via Interleaved Logs,” in *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems - ASPLOS ’16*, (Atlanta, Georgia, USA), pp. 489–502, ACM Press, 2016.

[68] B. H. Park, S. Hukerikar, R. Adamson, and C. Engelmann, “Big Data Meets HPC Log Analytics: Scalable Approach to Understanding Systems at Extreme Scale,” in *2017 IEEE International Conference on Cluster Computing (CLUSTER)*, pp. 758–765, Sept. 2017.

[69] R. Vaarandi and M. Pihelgas, “Logcluster - A Data Clustering and Pattern Mining Algorithm for Event Logs,” in *2015 11th International Conference on Network and Service Management (CNSM)*, pp. 1–7, Nov. 2015.

[70] M. Nagappan and M. A. Vouk, “Abstracting Log Lines to Log Event Types for Mining Software System Logs,” in *2010 7th IEEE Working Conference on Mining Software Repositories (MSR 2010)*, pp. 114–117, May 2010.

[71] Z. M. Jiang, A. E. Hassan, P. Flora, and G. Hamann, “Abstracting Execution Logs to Execution Events for Enterprise Applications (Short Paper),” in *2008 The Eighth International Conference on Quality Software*, pp. 181–186, Aug. 2008.

[72] Q. Fu, J.-G. Lou, Y. Wang, and J. Li, “Execution Anomaly Detection in Distributed Systems through Unstructured Log Analysis,” in *2009 Ninth IEEE International Conference on Data Mining*, (Miami Beach, FL, USA), pp. 149–158, IEEE, Dec. 2009.

[73] A. Makanju, A. N. Zincir-Heywood, and E. E. Milios, “A Lightweight Algorithm for Message Type Extraction in System Application Logs,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 24, pp. 1921–1936, Nov. 2012.

[74] L. Tang, T. Li, and C.-S. Perng, “LogSig: Generating System Events from Raw Textual Logs,” in *Proceedings of the 20th ACM International Conference on Information and Knowledge Management - CIKM ’11*, (Glasgow, Scotland, UK), p. 785, ACM Press, 2011.

[75] M. Mizutani, “Incremental Mining of System Log Format,” in *2013 IEEE International Conference on Services Computing*, pp. 595–602, June 2013.

[76] K. Shima, “Length Matters: Clustering System Log Messages using Length of Words,” *arXiv:1611.03213 [cs]*, Nov. 2016.

[77] H. Hamooni, B. Debnath, J. Xu, H. Zhang, G. Jiang, and A. Mueen, “LogMine: Fast Pattern Recognition for Log Analytics,” in *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management - CIKM ’16*, (Indianapolis, Indiana, USA), pp. 1573–1582, ACM Press, 2016.

[78] M. Du and F. Li, “Spell: Streaming Parsing of System Event Logs,” in *2016 IEEE 16th International Conference on Data Mining (ICDM)*, pp. 859–864, Dec. 2016.

[79] P. He, J. Zhu, Z. Zheng, and M. R. Lyu, "Drain: An Online Log Parsing Approach with Fixed Depth Tree," in *2017 IEEE International Conference on Web Services (ICWS)*, (Honolulu, HI, USA), pp. 33–40, IEEE, June 2017.

[80] S. Messaoudi, A. Panichella, D. Bianculli, L. Briand, and R. Sasnauskas, "A Search-Based Approach for Accurate Identification of Log Message Formats," in *Proceedings of the 26th Conference on Program Comprehension - ICPC '18*, (Gothenburg, Sweden), pp. 167–177, ACM Press, 2018.

[81] P. He, J. Zhu, S. He, J. Li, and M. R. Lyu, "An Evaluation Study on Log Parsing and Its Use in Log Mining," in *2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pp. 654–661, June 2016.

[82] J. Zhu, S. He, J. Liu, P. He, Q. Xie, Z. Zheng, and M. R. Lyu, "Tools and benchmarks for automated log parsing," in *Proceedings of the 41st International Conference on Software Engineering: Software Engineering in Practice*, ICSE-SEIP '19, (Montreal, Quebec, Canada), pp. 121–130, IEEE Press, May 2019.

[83] X. Fu, R. Ren, J. Zhan, W. Zhou, Z. Jia, and G. Lu, "LogMaster: Mining Event Correlations in Logs of Large-Scale Cluster Systems," in *2012 IEEE 31st Symposium on Reliable Distributed Systems*, pp. 71–80, Oct. 2012.

[84] "The Alert Logic agent for Linux." https://docs.alertlogic.com/prepare/alert-logic-agent-linux.htm. [accessed 13-03-2020].

[85] "Cloudlytics." https://cloudlytics.com/overview#event-section. [accessed 13-03-2020].

[86] "EventSentry." https://www.eventsentry.com/documentation/overview/html/index.html. [accessed 13-03-2020].

[87] "Log Management Solution for Compliance & Security - EventTracker." https://eventtracker.com/solutions/log-manager/. [accessed 13-03-2020].

[88] "Syslog Parser Plugin: Fluentd." https://docs.fluentd.org/v1.0/articles/parser_syslog. [accessed 13-03-2020].

[89] "Apache Flume 1.9.0 User Guide." https://flume.apache.org/releases/content/1.9.0/FlumeUserGuide.html#syslog-sources. [accessed 13-03-2020].

[90] "Extractors — Graylog 3.0.0 documentation." http://docs.graylog.org/en/3.0/pages/extractors.html#the-problem-explained. [accessed 13-03-2020].

[91] "InTrust: Windows Event Log Management and Analysis Tool." https://www.quest.com/products/intrust/. [accessed 13-03-2020].

[92] "MOVEit Automation Web Admin Help." https://docs.ipswitch.com/MOVEit/Automation2019/Help/Admin/en/index.htm#48174.htm. [accessed 13-03-2020].

[93] "Log Formats — lnav 0.8.4 documentation." https://lnav.readthedocs.io/en/latest/formats.html. [accessed 13-03-2020].

[94] "LOGalyze - Log analysis." http://www.logalyze.com/solutions/log-analysis. [accessed 13-03-2020].

[95] "LogDNA Quick Start Guide." https://docs.logdna.com/docs/. [accessed 13-03-2020].

[96] Logentries, "Alerting and Reporting." https://logentries.com/product/alerting-and-reporting/. [accessed 13-03-2020].

[97] "Scrub Sensitive Data in Rsyslog." https://www.loggly.com/docs/scrub-data-rsyslog/. [accessed 13-03-2020].

[98] "Logmatic." https://doc.logmatic.io/. [accessed 13-03-2020].

[99] "Log Management & Log Analysis | LogRhythm." https://logrhythm.com/solutions/security/log-management/. [accessed 13-03-2020].

[100] "Logsign: Intelligent Next-Gen SIEM Solution." https://www.logsign.com/index.php. [accessed 13-03-2020].

[101] J. Turnbull, *The Logstash Book*. Amazon, Mar. 2013.

[102] Logstrom, "Log Management Solutions." https://www.blackstratus.com/log-storm/. [accessed 11-Mar-2020].

[103] J. E. Prewett, "Analyzing Cluster Log FIles Using Logsurfer," in *Proceedings of the 4th Annual Conference on Linux Clusters*, June 2003.

[104] logzio, "Alerts." https://docs.logz.io/user-guide/alerts/. [accessed 13-03-2020].

[105] "Loom's Solution vs. Log Management Solutions." http://support.loomsystems.com/comparisons/looms-solution-vs-log-management-solutions. [accessed 13-03-2020].

[106] motadata, "Log Management Tools | Event Log Analyzer | Log Monitoring Software."

[107] "Nagios Log Server - Full Architecture Overview." https://support.nagios.com/kb/article/nagios-log-server-full-architecture-overview-98.html. [accessed 13-03-2020].

[108] "NXLog User Guide." https://nxlog.co/documentation/nxlog-user-guide. [accessed 13-03-2020].

[109] N. Taerat, J. Brandt, A. Gentile, M. Wong, and C. Leangsuksun, “Baler: Deterministic, Lossless Log Message Clustering Tool,” *Computer Science - Research and Development*, vol. 26, p. 285, Apr. 2011.

[110] J. Brandt, F. Chen, A. Gentile, C. B. Leangsuksun, J. Mayo, P. Pebay, D. Roe, N. Taerat, D. Thompson, and M. Wong, “Framework for Enabling System Understanding,” in *Euro-Par 2011: Parallel Processing Workshops* (M. Alexander, P. D’Ambra, A. Belloum, G. Bosilca, M. Cannataro, M. Danelutto, B. Di Martino, M. Gerndt, E. Jeannot, R. Namyst, J. Roman, S. L. Scott, J. L. Traff, G. Vallée, and J. Weidendorfer, eds.), Lecture Notes in Computer Science, (Berlin, Heidelberg), pp. 231–240, Springer, 2012.

[111] “Scalyr Under The Hood: Log Management That’s Fast, At Scale.” `https://www.scalyr.com/product`. [accessed 13-03-2020].

[112] splunk, “AI for IT: Preventing Outages With Predictive Analytics.” `https://www.splunk.com/en_us/form/ai-for-it-preventing-outages-with-predictive-analytics.html`. [accessed 13-03-2020].

[113] “Machine Learning Powered Analytics.” `https://www.sumologic.com/solutions/machine-learning-powered-analytics/`. [accessed 13-03-2020].

[114] S. E. Hansen and E. T. Atkins, “Automated System Monitoring and Notification With Swatch,” in *Proceedings of the 7th USENIX Conference on System Administration*, pp. 145–152, USENIX Association, May 1993.

[115] “XpoLog Center Documentation.” `http://wiki.xplg.com/display/XPOL/XpoLog+Analytics`. [accessed 13-03-2020].

[116] J.-G. Lou, Q. Fu, S. Yang, Y. Xu, and J. Li, “Mining Invariants from Console Logs for System Problem Detection,” *USENIX Annual Technical Conference (ATC)*, pp. 231–244, 2010.

[117] K. Zhang, J. Xu, M. R. Min, G. Jiang, K. Pelechrinis, and H. Zhang, “Automated IT system failure prediction: A deep learning approach,” in *2016 IEEE International Conference on Big Data (Big Data)*, pp. 1291–1300, Dec. 2016.

[118] M. Du, F. Li, G. Zheng, and V. Srikumar, “DeepLog: Anomaly Detection and Diagnosis from System Logs through Deep Learning,” in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security - CCS ’17*, (Dallas, Texas, USA), pp. 1285–1298, ACM Press, 2017.

[119] A. Netti, Z. Kiziltan, O. Babaoglu, A. Sîrbu, A. Bartolini, and A. Borghesi, “FINJ: A Fault Injection Tool for HPC Systems,” in *Euro-Par 2018: Parallel Processing Workshops*, vol. 11339, pp. 800–812, Cham: Springer International Publishing, 2019.

[120] T. E. PARLIAMENT, “Regulation (EU) 2018/1725 of the European Parliament and of the Council of 23 October 2018 on the Protection of Natural Persons with Regard to the

Processing of Personal Data by the Union Institutions, Bodies, Offices and Agencies and on the Free Movement of Such Data, and Repealing Regulation (EC) No 45/2001 and Decision No 1247/2002/ECText with EEA Relevance.," *Official Journal of the European Union*, Oct. 2018.

[121] L. Sweeney, "Simple Demographics Often Identify People Uniquely," . *Pittsburgh*, p. 34, 2000.

[122] A. Machanavajjhala, J. Gehrke, D. Kifer, and M. Venkitasubramaniam, "L-Diversity: Privacy Beyond k-Anonymity," in *22nd International Conference on Data Engineering (ICDE'06)*, pp. 24–24, Apr. 2006.

[123] N. Li, T. Li, and S. Venkatasubramanian, "T-Closeness: Privacy Beyond k-Anonymity and l-Diversity," in *2007 IEEE 23rd International Conference on Data Engineering*, pp. 106–115, Apr. 2007.

[124] P. M. V. Kumar, "T-Closeness Integrated L-Diversity Slicing for Privacy Preserving Data Publishing," *Journal of Computational and Theoretical Nanoscience*, vol. 15, pp. 106–110, Jan. 2018.

[125] C. Dwork, "Differential Privacy," in *Automata, Languages and Programming* (M. Bugliesi, B. Preneel, V. Sassone, and I. Wegener, eds.), Lecture Notes in Computer Science, pp. 1–12, Springer Berlin Heidelberg, 2006.

[126] T. E. PARLIAMENT, "Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the Protection of Natural Persons with Regard to the Processing of Personal Data and on the Free Movement of Such Data, and Repealing Directive 95/46/EC (General Data Protection Regulation) (Text with EEA Relevance)," May 2016.

[127] R. Dahlberg and T. Pulls, *Standardized Syslog Processing : Revisiting Secure Reliable Data Transfer and Message Compression*. Karlstads universitet, 2016.

[128] R. Gerhards, "RSyslog Documentation." `https://www.rsyslog.com/doc/`. [accessed 13-03-2020].

[129] J. Sissel, "Logstash, Centralize, Transform and Stash Your Data." `https://www.elastic.co/products/logstash`. [accessed 13-03-2020].

[130] S. Sanjappa and M. Ahmed, "Analysis of Logs by Using Logstash," in *Proceedings of the 5th International Conference on Frontiers in Intelligent Computing: Theory and Applications*, Advances in Intelligent Systems and Computing, pp. 579–585, Springer Singapore, 2017.

[131] "Loggy, Log management." `http://www.loggly.com/`. [accessed 11-Mar-2020].

[132] A. Gholami, E. Laure, P. Somogyi, O. Spjuth, S. Niazi, and J. Dowling, "Privacy-Preservation for Publishing Sample Availability Data with Personal Identifiers," *Journal of Medical and Bioengineering*, vol. 4, no. 2, pp. 117–125, 2015.

[133] M. Templ, B. Meindl, and A. Kowarik, “sdcMicro: Statistical Disclosure Control Methods for Anonymization of Microdata and Risk Estimation,” May 2018.

[134] C. Dai, G. Ghinita, E. Bertino, J.-W. Byun, and N. Li, “TIAMAT: A Tool for Interactive Analysis of Microdata Anonymization Techniques,” *Proceedings of the VLDB Endowment*, vol. 2, pp. 1618–1621, Aug. 2009.

[135] M. Ciglic, J. Eder, and C. Koncilia, “Anonymization of Data Sets with NULL Values,” in *Transactions on Large-Scale Data- and Knowledge-Centered Systems XXIV: Special Issue on Database- and Expert-Systems Applications*, Lecture Notes in Computer Science, pp. 193–220, Berlin, Heidelberg: Springer Berlin Heidelberg, 2016.

[136] “UTD Anonymization ToolBox.” `http://cs.utdallas.edu/dspl/cgi-bin/toolbox/`. [accessed 13-03-2020].

[137] X. Xiao, G. Wang, and J. Gehrke, “Interactive Anonymization of Sensitive Data,” in *Proceedings of the 35th SIGMOD International Conference on Management of Data - SIGMOD ’09*, (Providence, Rhode Island, USA), p. 1051, ACM Press, 2009.

[138] A. Meyerson and R. Williams, “On the Complexity of Optimal k-Anonymity,” in *Proceedings of the Twenty-Third ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems - PODS ’04*, (Paris, France), p. 223, ACM Press, 2004.

[139] R. J. Bayardo and R. Agrawal, “Data Privacy Through Optimal k-Anonymization,” in *21st International Conference on Data Engineering (ICDE’05)*, pp. 217–228, Apr. 2005.

[140] A. Gionis and T. Tassa, “K-Anonymization with Minimal Loss of Information,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 21, pp. 206–219, Feb. 2009.

[141] K. Murakami and T. Uno, “Optimization Algorithm for k-Anonymization of Datasets with Low Information Loss,” *International Journal of Information Security*, vol. 17, pp. 631–644, Nov. 2018.

[142] J. Xu, W. Wang, J. Pei, X. Wang, B. Shi, and A. W.-C. Fu, “Utility-Based Anonymization for Privacy Preservation with Less Information Loss,” *ACM SIGKDD Explorations Newsletter*, vol. 8, pp. 21–30, Dec. 2006.

[143] M. Terrovitis, N. Mamoulis, and P. Kalnis, “Local and Global Recoding Methods for Anonymizing Set-Valued Data,” *The VLDB Journal*, vol. 20, pp. 83–106, Feb. 2011.

[144] M. E. Nergiz, M. Z. Gök, and U. Özkanlı, “Preservation of Utility through Hybrid k-Anonymization,” in *Trust, Privacy, and Security in Digital Business*, Lecture Notes in Computer Science, pp. 97–111, Springer Berlin Heidelberg, 2013.

[145] Z.-H. WANG, J. XU, W. WANG, and B.-L. SHI, “Clustering-Based Approach for Data Anonymization,” *Journal of Software*, 2014.

[146] J. Soria-Comas and J. Domingo-Ferrer, “Self-enforcing Collaborative Anonymization Via Co-utility,” in *Co-Utility: Theory and Applications*, Studies in Systems, Decision and Control, pp. 139–151, Cham: Springer International Publishing, 2018.

[147] R. C.-W. Wong, A. W.-C. Fu, K. Wang, P. S. Yu, and J. Pei, “Can the Utility of Anonymized Data be Used for Privacy Breaches?,” *ACM Transactions on Knowledge Discovery from Data*, vol. 5, pp. 1–24, Aug. 2011.

[148] M. Templ, “Data Utility and Information Loss,” in *Statistical Disclosure Control for Microdata: Methods and Applications in R* (M. Templ, ed.), pp. 133–156, Cham: Springer International Publishing, 2017.

[149] G. Loukides and J. Shao, “Capturing Data Usefulness and Privacy Protection in k-Anonymisation,” in *Proceedings of the 2007 ACM Symposium on Applied Computing - SAC ’07*, (Seoul, Korea), p. 370, ACM Press, 2007.

[150] F. Prasser and F. Kohlmayer, “Putting Statistical Disclosure Control into Practice: The ARX Data Anonymization Tool,” in *Medical Data Privacy Handbook* (A. Gkoulalas-Divanis and G. Loukides, eds.), pp. 111–148, Cham: Springer International Publishing, 2015.

[151] J. Gardner and L. Xiong, “An Integrated Framework for De-Identifying Unstructured Medical Data,” *Data & Knowledge Engineering*, vol. 68, pp. 1441–1451, Dec. 2009.

[152] C. Rath, “A Privacy-Aware Logging Framework Extension for Logback.: Nobecutan/Privacy-Aware-Logging,” June 2016.

[153] C. Rath, “Usable Privacy-Aware Logging for Unstructured Log Entries,” in *2016 11th International Conference on Availability, Reliability and Security (ARES)*, pp. 272–277, Aug. 2016.

[154] B. Ozcelik and C. Yilmaz, “Seer: A Lightweight Online Failure Prediction Approach,” in *2017 IEEE 41st Annual Computer Software and Applications Conference (COMPSAC)*, vol. 1, pp. 624–625, July 2017.

[155] F. Salfner, M. Lenk, and M. Malek, “A Survey of Online Failure Prediction Methods,” *ACM Comput. Surv.*, vol. 42, pp. 10:1–10:42, Mar. 2010.

[156] T. Herault and Y. Robert, eds., *Fault-Tolerance Techniques for High-Performance Computing*. Computer Communications and Networks, Cham: Springer International Publishing, 2015.

[157] S. He, J. Zhu, P. He, and M. R. Lyu, “Experience Report: System Log Analysis for Anomaly Detection,” in *2016 IEEE 27th International Symposium on Software Reliability Engineering (ISSRE)*, pp. 207–218, Oct. 2016.

[158] D. Jauk, D. Yang, and M. Schulz, “Predicting faults in high performance computing systems: An in-depth survey of the state-of-the-practice,” in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC ’19, (Denver, Colorado), pp. 1–13, Association for Computing Machinery, Nov. 2019.

[159] T. Islam and D. Manivannan, “Predicting Application Failure in Cloud: A Machine Learning Approach,” in *2017 IEEE International Conference on Cognitive Computing (ICCC)*, pp. 24–31, June 2017.

[160] G. Aupy, Y. Robert, and F. Vivien, “Assuming Failure Independence: Are We Right to be Wrong?,” in *2017 IEEE International Conference on Cluster Computing (CLUSTER)*, pp. 709–716, Sept. 2017.

[161] C. Liang, T. Benson, P. Kanuparthy, and Y. He, “Finding Needles in the Haystack: Harnessing Syslogs for Data Center Management,” *arXiv:1605.06150 [cs]*, May 2016.

[162] D. Zou, H. Qin, H. Jin, W. Qiang, Z. Han, and X. Chen, “Improving Log-Based Fault Diagnosis by Log Classification,” in *Advanced Information Systems Engineering*, vol. 7908, pp. 446–458, Berlin, Heidelberg: Springer Berlin Heidelberg, 2014.

[163] N. El-Sayed and B. Schroeder, “Reading Between the Lines of Failure Logs: Understanding How HPC Systems Fail,” in *2013 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pp. 1–12, June 2013.

[164] N. Yigitbasi, M. Gallet, D. Kondo, A. Iosup, and D. Epema, “Analysis and Modeling of Time-Correlated Failures in Large-Scale Distributed Systems,” in *2010 11th IEEE/ACM International Conference on Grid Computing*, pp. 65–72, Oct. 2010.

[165] K. Yamanishi, J.-i. Takeuchi, G. Williams, and P. Milne, “On-Line Unsupervised Outlier Detection Using Finite Mixtures with Discounting Learning Algorithms,” *Data Mining and Knowledge Discovery*, vol. 8, pp. 275–300, May 2004.

[166] R. Vaarandi, B. Blumbergs, and M. Kont, “An Unsupervised Framework for Detecting Anomalous Messages from Syslog Log Files,” in *NOMS 2018 - 2018 IEEE/IFIP Network Operations and Management Symposium*, pp. 1–6, Apr. 2018.

[167] T. Kimura, A. Watanabe, T. Toyono, and K. Ishibashi, “Proactive Failure Detection Learning Generation Patterns of Large-scale Network Logs,” *IEICE Transactions on Communications*, 2018.

[168] W. Xu, L. Huang, A. Fox, D. Patterson, and M. Jordan, “Online System Problem Detection by Mining Patterns of Console Logs,” in *2009 Ninth IEEE International Conference on Data Mining*, (Miami Beach, FL, USA), pp. 588–597, IEEE, Dec. 2009.

[169] J. P. Rouillard, “Real-time Log File Analysis Using the Simple Event Correlator (SEC),” in *18th Large Installation System Administration Conference*, (Atlanta, Georgia, USA), pp. 133–150, USENIX Association, Nov. 2004.

[170] P. He, J. Zhu, P. Xu, Z. Zheng, and M. R. Lyu, “A Directed Acyclic Graph Approach to Online Log Parsing,” *arXiv:1806.04356 [cs]*, June 2018.

[171] R. Baldoni, L. Montanari, and M. Rizzuto, “On-line Failure Prediction in Safety-critical Systems,” *Future Generation Computer Systems*, vol. 45, pp. 123–132, Apr. 2015.

[172] A. W. Williams, S. M. Pertet, and P. Narasimhan, “Tiresias: Black-Box Failure Prediction in Distributed Systems,” in *2007 IEEE International Parallel and Distributed Processing Symposium*, pp. 1–8, Mar. 2007.

[173] H. S. Pannu, J. Liu, and S. Fu, “A Self-Evolving Anomaly Detection Framework for Developing Highly Dependable Utility Clouds,” in *2012 IEEE Global Communications Conference (GLOBECOM)*, Dec. 2012.

[174] Y. Tan, X. Gu, and H. Wang, “Adaptive System Anomaly Prediction for Large-scale Hosting Infrastructures,” in *Proceedings of the 29th ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, PODC ’10, (Zurich, Switzerland), pp. 173–182, ACM, 2010.

[175] F. Rasheed, M. Alshalalfa, and R. Alhajj, “Efficient Periodicity Mining in Time Series Databases Using Suffix Trees,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 23, pp. 79–94, Jan. 2011.

[176] K. F. Xylogiannopoulos, P. Karampelas, and R. Alhajj, “Periodicity Data Mining in Time Series Using Suffix Arrays,” in *2012 6th IEEE International Conference Intelligent Systems*, pp. 172–181, Sept. 2012.

[177] E. Ukkonen, “On-Line Construction of Suffix Trees,” *Algorithmica*, vol. 14, pp. 249–260, Sept. 1995.

[178] K. F. Xylogiannopoulos, P. Karampelas, and R. Alhajj, “Analyzing Very Large Time Series Using Suffix Arrays,” *Applied Intelligence*, vol. 41, pp. 941–955, Oct. 2014.

[179] P. Fournier-Viger, J. C.-W. Lin, B. Vo, T. T. Chi, J. Zhang, and H. B. Le, “A Survey of Itemset Mining,” *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 7, July 2017.

[180] A. Patcha and J.-M. Park, “An Overview of Anomaly Detection Techniques: Existing Solutions and Latest Technological Trends,” *Computer Networks*, 2007.

[181] V. Chandola, A. Banerjee, and V. Kumar, “Anomaly Detection: A Survey,” *ACM Comput. Surv.*, vol. 41, pp. 1–58, July 2009.

[182] T.-F. Yen, A. Oprea, K. Onarlioglu, T. Leetham, W. Robertson, A. Juels, and E. Kirda, “Beehive: Large-scale Log Analysis for Detecting Suspicious Activity in Enterprise Networks,” in *Proceedings of the 29th Annual Computer Security Applications Conference*, ACSAC ’13, (New Orleans, Louisiana, USA), pp. 199–208, ACM, 2013.

[183] A. Oprea, Z. Li, T. Yen, S. H. Chin, and S. Alrwais, "Detection of Early-Stage Enterprise Infection by Mining Large-Scale Log Data," in *45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pp. 45–56, June 2015.

[184] S. Roy, A. C. König, I. Dvorkin, and M. Kumar, "PerfAugur: Robust Diagnostics for Performance Anomalies in Cloud Services," in *2015 IEEE 31st International Conference on Data Engineering*, Apr. 2015.

[185] I. Beschastnikh, Y. Brun, M. D. Ernst, A. Krishnamurthy, and T. E. Anderson, "Mining Temporal Invariants from Partially Ordered Logs," *SIGOPS Oper. Syst. Rev.*, vol. 45, pp. 39–46, Jan. 2012.

[186] I. Jolliffe, "Principal Component Analysis," in *International Encyclopedia of Statistical Science* (M. Lovric, ed.), pp. 1094–1096, Berlin, Heidelberg: Springer, 2011.

[187] Z. Li, M. Davidson, S. Fu, S. Blanchard, and M. Lang, "Converting Unstructured System Logs into Structured Event List for Anomaly Detection," in *Proceedings of the 13th International Conference on Availability, Reliability and Security - ARES 2018*, (Hamburg, Germany), pp. 1–10, ACM Press, 2018.

[188] N. Aussel, Y. Petetin, and S. Chabridon, "Improving Performances of Log Mining for Anomaly Prediction Through NLP-Based Log Parsing," in *2018 IEEE 26th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, pp. 237–243, Sept. 2018.

[189] Y. Liang, Y. Zhang, M. Jette, A. Sivasubramaniam, and R. K. Sahoo, "BlueGene/L Failure Analysis and Prediction Models," in *Dependable Systems and Networks, 2006. DSN 2006. International Conference On*, pp. 425–434, June 2006.

[190] J. Brandt, A. Gentile, J. Mayo, P. Pébay, D. Roe, D. Thompson, and M. Wong, "Methodologies for Advance Warning of Compute Cluster Problems Via Statistical Analysis: A Case Study," in *Proceedings of the 2009 Workshop on Resiliency in High Performance*, (New York, NY, USA), pp. 7–14, ACM, 2009.

[191] A. Gainaru, F. Cappello, and W. Kramer, "Taming of the Shrew: Modeling the Normal and Faulty Behaviour of Large-Scale Hpc Systems," in *26th IEEE Int'l Parallel and Distributed Processing Symp*, pp. 1168–1179, Scholar: IEEE, May 2012.

[192] C. Costa, Y. Park, B. Rosenburg, C. Cher, and K. Ryu, "A System Software Approach to Proactive Memory-Error Avoidance," in *SC14: Int'l Conf. for High Performance Computing, Networking, Storage and Analysis*, pp. 707–718, IEEE, Nov. 2014.

[193] A. Gainaru, F. Cappello, M. Snir, and W. Kramer, "Fault Prediction Under the Microscope: A Closer Look into Hpc Systems," in *2012 International Conference for High Performance Computing, Networking, Storage and Analysis*, (Salt Lake City, UT), pp. 1–11, IEEE, Nov. 2012.

[194] X. Fu, R. Ren, S. Mckee, J. Zhan, and N. Sun, “Digging Deeper into Cluster System Logs for Failure Prediction and Root Cause Diagnosis,” in *Cluster Computing (CLUSTER), 2014 IEEE International Conference On*, pp. 103–112, Sept. 2014.

[195] R. Rajachandrasekar, X. Besseron, and D. K. Panda, “Monitoring and Predicting Hardware Failures in HPC Clusters with FTB-IPMI,” in *2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops PhD Forum*, pp. 1136–1143, May 2012.

[196] Y. Watanabe, H. Otsuka, M. Sonoda, S. Kikuchi, and Y. Matsumoto, “Online Failure Prediction in Cloud Datacenters by Real-Time Message Pattern Learning,” in *4th IEEE Int'l Conf. on Cloud Computing Technology and Science Proceedings*, (Scholar), pp. 504–511, Dec. 2012.

[197] Y. Watanabe, H. Otsuka, and Y. Matsumoto, “Failure Prediction for Cloud Datacenter by Hybrid Message Pattern Learning,” in *IEEE 11th Int'l Conference on Ubiquitous Intelligence / Computing IEEE 14th Int'l Conf on Scalable Computing and Communications and Its Associated Workshops*, pp. 425–432, Dec. 2014.

[198] L. Guo, D. Li, I. Laguna, and M. Schulz, “Understanding Natural Error Resilience in HPC Applications,” in *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis*, (Piscataway, NJ, USA), IEEE Press, 2018.

[199] A. Das, F. Mueller, P. Hargrove, E. Roman, and S. Baden, “Doomsday: Predicting Which Node Will Fail when on Supercomputers,” in *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis*, SC '18, (Piscataway, NJ, USA), pp. 9:1–9:14, IEEE Press, 2018.

[200] 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*, vol. 11, 4, Nov. 2015.

[201] Z. Zheng, Z. Lan, R. Gupta, S. Coghlan, and P. Beckman, “A Practical Failure Prediction with Location and Lead Time for Blue Gene/P,” in *2010 International Conference on Dependable Systems and Networks Workshops (DSN-W)*, pp. 15–22, June 2010.

[202] J. Thompson, D. Dreisigmeyer, T. Jones, M. Kirby, and J. Ladd, “Accurate Fault Prediction of BlueGene/P RAS Logs via Geometric Reduction,” in *Int'l Conf. on Dependable Systems and Networks Workshops (DSN-W*, (Scholar), pp. 8–14, June 2010.

[203] C. Rincón, J. Pâris, R. Vilalta, A. Cheng, and D. Long, “Disk Failure Prediction in Heterogeneous Environments,” in *2017 Int'l. Symp. on Performance Evaluation of Computer and Telecommunication Systems (SPECTS*, pp. 1–7, IEEE, July 2017.

[204] S. Ganguly, A. Consul, A. Khan, B. Bussone, J. Richards, and A. Miguel, “A Practical Approach to Hard Disk Failure Prediction in Cloud Platforms: Big Data Model for Failure Management in Datacenters,” in *IEEE Second Int'l Conf. on Big Data Computing Service and Applications (BigDataService*, pp. 105–116, IEEE, Mar. 2016.

[205] N. Nakka, A. Agrawal, and A. Choudhary, "Predicting Node Failure in High Performance Computing Systems from Failure and Usage Logs," in *2011 IEEE International Symposium on Parallel and Distributed Processing Workshops and Phd Forum*, pp. 1557–1566, May 2011.

[206] J. Klinkenberg, C. Terboven, S. Lankes, and M. S. Müller, "Data Mining-Based Analysis of HPC Center Operations," in *2017 IEEE International Conference on Cluster Computing (CLUSTER)*, pp. 766–773, Sept. 2017.

[207] M. Soualhia, F. Khomh, and S. Tahar, "Predicting Scheduling Failures in the Cloud: A Case Study with Google Clusters and Hadoop on Amazon Emr," in *Proceedings of the 2015 IEEE 17th Int'l Conf. on High Performance Computing and Communications, 7th Int. Symp. on Cyberspace Safety and Security, and 12th Int'l Conf. on Embedded Software and Systems*, (Washington, DC, USA), pp. 58–65, IEEE Computer Society, 2015.

[208] N. El-Sayed, H. Zhu, and B. Schroeder, "Learning from Failure Across Multiple Clusters: A Trace-Driven Approach to Understanding, Predicting, and Mitigating Job Terminations," in *IEEE 37th Int'l Conf. on Distributed Computing Systems (ICDCS*, pp. 1333–1344, Scholar: IEEE, June 2017.

[209] T. Chalermarrewong, T. Achalakul, and S. See, "Failure Prediction of Data Centers Using Time Series and Fault Tree Analysis," in *2012 IEEE 18th Int'l Conf. on Parallel and Distributed Systems*, pp. 794–799, Scholar: IEEE, Dec. 2012.

[210] Q. Guan, Z. Zhang, and S. Fu, "Proactive Failure Management by Integrated Unsupervised and Semi-Supervised Learning for Dependable Cloud Systems," in *2011 Sixth Int'l Conf. on Availability, Reliability and Security*, pp. 83–90, Scholar: IEEE, Aug. 2011.

[211] A. Sîrbu and O. Babaoglu, "Towards Operator-Less Data Centers Through Data-Driven, Predictive, Proactive Autonomics," *Cluster Computing*, vol. 19, 2, June 2016.

[212] Q. Liu, J. Zhou, G. Jin, Q. Sun, and M. Xi, "FABSR: A Method for Cluster Failure Prediction Based on Bayesian Serial Revision and an Application to LANL Cluster," *Quality and Reliability Engineering Int'l*, vol. 27, 4, 2011.

[213] Y. Liang, Y. Zhang, H. Xiong, and R. Sahoo, "Failure Prediction in IBM BlueGene/L Event Logs," in *Seventh IEEE Int'l Conf. on Data Mining (ICDM 2007*, pp. 583–588, Scholar: IEEE, Oct. 2007.

[214] X. Lu, WANG, j. Z. H., R., GE, and B., "Autonomic Failure Prediction Based on Manifold Learning for Large-Scale Distributed Systems," *The Journal of China Universities of Posts and Telecommunications*, vol. 17, 4, pp. 116–124, 2010.

[215] 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 Symp. on Mass Storage Systems and Technologies (MSST*, pp. 1–5, IEEE, May 2013.

[216] A. Pelaez, A. Quiroz, J. Browne, E. Chuah, and M. Parashar, “Online Failure Prediction for HPCResources Using Decentralized Clustering,” in *2014 21st Int’l Conf. on High Performance Computing (HiPC*, pp. 1–9, Scholar: IEEE, Dec. 2014.

[217] B. Agrawal, T. Wiktorski, and C. Rong, “Analyzing and Predicting Failure in Hadoop Clusters Using Distributed Hidden Markov Model,” in *Revised Selected Papers of the Second Int’l Conf. on Cloud Computing and Big Data*, vol. 9106, pp. 232–246, New York, NY, USA: Springer, 2015.

[218] L. Yu, Z. Zheng, Z. Lan, and S. Coghlan, “Practical Online Failure Prediction for Blue Gene/P: Period-Based Vs Event-Driven,” in *2011 IEEE/IFIP 41st Int’l Conf. on Dependable Systems and Networks Workshops (DSN-W*, pp. 259–264, Scholar: IEEE, June 2011.

[219] X. Chen, C. Lu, and K. Pattabiraman, “Failure Prediction of Jobs in Compute Clouds: A Google Cluster Case Study,” in *2014 IEEE Int’l. Symp. on Software Reliability Engineering Workshops*, pp. 341–346, Scholar: IEEE, Nov. 2014.

[220] J. Gu, Z. Zheng, Z. Lan, J. White, E. Hocks, and B. Park, “Dynamic Meta-Learning for Failure Prediction in Large-Scale Systems: A Case Study,” in *2008 37th Int’l Conf. on Parallel Processing*, pp. 157–164, Scholar: IEEE, Sept. 2008.

[221] Z. Lan, J. Gu, Z. Zheng, R. Thakur, and S. Coghlan, “A Study of Dynamic Meta-Learning for Failure Prediction in Large-Scale Systems,” *Journal of Parallel and Distributed Computing*, vol. 70, 6, pp. 630–643, 2010.

[222] A. Gainaru, F. Cappello, S. Trausan-Matu, and B. Kramer, “Event Log Mining Tool for Large Scale HPC Systems,” in *Euro-Par 2011 Parallel Processing* (E. Jeannot, R. Namyst, and J. Roman, eds.), vol. 6852, pp. 52–64, Berlin, Heidelberg: Springer Berlin Heidelberg, 2011.

[223] A. Gainaru, M. S. Bouguerra, F. Cappello, M. Snir, and W. Kramer, “Navigating the Blue Waters: Online Failure Prediction in the Petascale Era,” Thechnical Report ANL/MCS-P5219, Argonne National Laboratory Technical Report, 2014.

[224] T. Pitakrat, D. Okanović, A. van Hoorn, and L. Grunske, “Hora: Architecture-aware online failure prediction,” *Journal of Systems and Software*, vol. 137, pp. 669–685, Mar. 2018.

[225] A. Das, F. Mueller, C. Siegel, and A. Vishnu, “Desh: Deep Learning for System Health Prediction of Lead Times to Failure in HPC,” in *Proceedings of the 27th International Symposium on High-Performance Parallel and Distributed Computing*, HPDC ’18, (New York, NY, USA), pp. 40–51, ACM, 2018.

[226] M. Zasadziński, V. Muntés-Mulero, M. Solé, D. Carrera, and T. Ludwig, “Early Termination of Failed HPC Jobs Through Machine and Deep Learning,” in *Euro-Par 2018: Parallel Processing* (M. Aldinucci, L. Padovani, and M. Torquati, eds.), vol. 11014, pp. 163–177, Cham: Springer International Publishing, 2018.

[227] M. Landauer, M. Wurzenberger, F. Skopik, G. Settanni, and P. Filzmoser, "Dynamic Log File Analysis: An Unsupervised Cluster Evolution Approach for Anomaly Detection," *Computers & Security*, vol. 79, pp. 94–116, Nov. 2018.

[228] A. Borghesi, A. Bartolini, M. Lombardi, M. Milano, and L. Benini, "Anomaly Detection Using Autoencoders in High Performance Computing Systems," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, pp. 9428–9433, July 2019.

[229] A. Rezaei, F. Mueller, P. Hargrove, and E. Roman, "Dino: Divergent Node Cloning for Sustained Redundancy in HPC," *Journal of Parallel and Distributed Computing*, vol. 109, pp. 350–362, Nov. 2017.

[230] C.-K. Chang, S. Lym, N. Kelly, M. B. Sullivan, and M. Erez, "Evaluating and Accelerating High-fidelity Error Injection for HPC," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis*, SC '18, (Piscataway, NJ, USA), pp. 45:1–45:13, IEEE Press, 2018.

[231] E. Horn, D. Fulp, J. Calhoun, and L. Olson, "FaultSight: A Fault Analysis Tool for HPC Researchers," in *2019 IEEE/ACM 9th Workshop on Fault Tolerance for HPC at eXtreme Scale (FTXS)*, pp. 21–30, Nov. 2019.

[232] A. B. Yoo, M. A. Jette, and M. Grondona, "SLURM: Simple Linux Utility for Resource Management," in *Job Scheduling Strategies for Parallel Processing* (D. Feitelson, L. Rudolph, and U. Schwiegelshohn, eds.), Lecture Notes in Computer Science, pp. 44–60, Springer Berlin Heidelberg, 2003.

[233] P. Schwan, "Lustre: Building a File System for 1,000-node Clusters," *In Proceedings of the Ottawa Linux Symposium*, vol. 2003, p. 9, 2003.

[234] "TOP500 Supercomputer Sites." `https://www.top500.org/lists/2019/11/`. [accessed 11-03-2020].

[235] D. L. Quoc, M. Beck, P. Bhatotia, R. Chen, C. Fetzer, and T. Strufe, "PrivApprox: Privacy-preserving stream analytics," in *2017 USENIX Annual Technical Conference (USENIX ATC 17)*, (Santa Clara, CA), pp. 659–672, USENIX Association, July 2017.

[236] S. Ghiasvand and F. M. Ciorba, "Anonymization of System Logs for Preserving Privacy and Reducing Storage," in *Proceedings of the 2018 Future of Information and Communications Conference (FICC)*, vol. 1, (Singapore), pp. 440–447, Springer, Apr. 2018.

[237] S. Ghiasvand, F. M. Ciorba, and W. E. Nagel, "Turning Privacy Constraints into Syslog Analysis Advantage," in *International Conference for High Performance Computing, Networking, Storage and Analysis*, (Salt Lake City, Utah, USA), Nov. 2016.

[238] G. Bertoni, J. Daemen, M. Peeters, and G. V. Assche, "The Making of KECCAK," *Cryptologia*, vol. 38, pp. 26–60, Jan. 2014.

[239] M. J. Dworkin, "SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions," Tech. Rep. NIST FIPS 202, National Institute of Standards and Technology, July 2015.

[240] S. Ghiasvand and F. M. Ciorba, "Assessing Data Usefulness for Failure Analysis in Anonymized System Logs," in *Proceedings of the 17th International Symposium on Parallel and Distributed Computing (ISPDC)*, (Geneva, Switzerland), pp. 164–171, June 2018.

[241] Technical University of Dresden, "Ordnung zur Errichtu ng und zum Betrieb eines Identitätsmanagementsystems an der Technischen Universität Dresden," July 2011.

[242] L. Collin, "A Quick Benchmark: Gzip vs. Bzip2 vs. LZMA." `http://tukaani.org/lzma/benchmarks.html`. [accessed 13-03-2020].

[243] G. Danti, "Linux Compressors Comparison on CentOS 6.5 x86-64: Lzo vs lz4 vs gzip vs bzip2 vs lzma." `https://www.ghiasvand.net/u/compression2`. [accessed 13-03-2020].

[244] J. Alakuijala, E. Kliuchnikov, Z. Szabadka, and L. Vandevenne, "Comparison of Brotli, Deflate, Zopfli, LZMA, LZHAM and Bzip2 Compression Algorithms," tech. rep., Google Inc., 22-Sep-2015.

[245] S. Ghiasvand, W. E. Nagel, and F. M. Ciorba, "Toward Resilience in HPC: A Prototype to Analyze and Predict System Behavior," in *International Supercomputing*, (Frankfurt, Germany), June 2016.

[246] A. Bartoli, A. D. Lorenzo, E. Medvet, and F. Tarlao, "Inference of Regular Expressions for Text Extraction from Examples," *IEEE Transactions on Knowledge and Data Engineering*, vol. 28, pp. 1217–1230, May 2016.

[247] Z. Zhong, J. Guo, W. Yang, T. Xie, J.-G. Lou, T. Liu, and D. Zhang, "Generating Regular Expressions from Natural Language Specifications: Are We There Yet?," in *Workshops at the Thirty-Second AAAI Conference on Artificial Intelligence*, (New Orleans, Louisiana, USA), p. 4, AAAI, Feb. 2018.

[248] S. Ghiasvand and F. M. Ciorba, "Automatic Classification of System Logs," in *International Supercomputing*, (Frankfurt, Germany), June 2018.

[249] V. Levenshtein, "Binary Codes Capable of Correcting Deletions, Insertions, and Reversals," *Dokl. Akad. Nauk SSSR*, vol. 10, pp. 707–710, Feb. 1965.

[250] S. Ghiasvand, F. M. Ciorba, R. Tschuter, and W. E. Nagel, "Lessons Learned from Spatial and Temporal Correlation of Node Failures in High Performance Computers," in *Proceedings of the 24th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP)*, (Heraklion, Crete, Greece), pp. 377–381, IEEE, Feb. 2016.

[251] S. Ghiasvand and F. M. Ciorba, “Event Pattern Identification in Anonymized System Logs,” in *International Supercomputing*, (Frankfurt, Germany), June 2017.

[252] C. D. Martino, W. Kramer, Z. Kalbarczyk, and R. Iyer, “Measuring and Understanding Extreme-Scale Application Resilience: A Field Study of 5,000,000 HPC Application Runs,” in *2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pp. 25–36, June 2015.

[253] S. Ghiasvand, F. M. Ciorba, R. Tschuter, and W. E. Nagel, “Analysis of Node Failures in High Performance Computers Based on System Logs,” in *International Conference for High Performance Computing, Networking, Storage and Analysis*, (Austin, Texas), Nov. 2015.

[254] D. Dai, O. R. Gatla, and M. Zheng, “A Performance Study of Lustre File System Checker: Bottlenecks and Potentials,” in *2019 35th Symposium on Mass Storage Systems and Technologies (MSST)*, pp. 7–13, May 2019.

[255] S. M. Khorandi, S. Ghiasvand, and M. Sharifi, “Reducing Load Imbalance of Virtual Clusters via Reconfiguration and Adaptive Job Scheduling,” in *Proceedings of the 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, (Madrid, Spain), pp. 992–999, IEEE, May 2017.

[256] S. Ghiasvand and F. M. Ciorba, “Anomaly Detection in High Performance Computers: A Vicinity Perspective,” in *2019 18th International Symposium on Parallel and Distributed Computing (ISPDC)*, pp. 112–120, June 2019.

[257] A. P. J. de Koning, W. Gu, T. A. Castoe, M. A. Batzer, and D. D. Pollock, “Repetitive Elements May Comprise Over Two-Thirds of the Human Genome,” *PLOS Genetics*, vol. 7, p. e1002384, Dec. 2011.

[258] C.-K. Peng, S. V. Buldyrev, A. L. Goldberger, S. Havlin, F. Sciortino, M. Simons, and H. E. Stanley, “Long-Range Correlations in Nucleotide Sequences,” *Nature*, vol. 356, p. 168, Mar. 1992.

[259] H. Li, B. Handsaker, A. Wysoker, T. Fennell, J. Ruan, N. Homer, G. Marth, G. Abecasis, and R. Durbin, “The Sequence Alignment/Map Format and SAMtools,” *Bioinformatics*, vol. 25, pp. 2078–2079, Aug. 2009.

[260] S. Ghiasvand, “uPAD: Unsupervised Privacy-Aware Anomaly Detection in High Performance Computing Systems:,” in *Proceedings of the 8th International Conference on Pattern Recognition Applications and Methods*, (Prague, Czech Republic), pp. 852–859, SCITEPRESS - Science and Technology Publications, 2019.

[261] J. Hawkins, S. Ahmad, S. Purdy, and A. Lavin, *Biological and Machine Intelligence (BAMI)*. Numenta, 2016. Initial online release 0.4.

[262] S. Ahmad, A. Lavin, S. Purdy, and Z. Agha, “Unsupervised Real-Time Anomaly Detection for Streaming Data,” *Neurocomputing*, vol. 262, pp. 134–147, Nov. 2017.

[263] S. Lowel and W. Singer, “Selection of Intrinsic Horizontal Connections in the Visual Cortex by Correlated Neuronal Activity,” *Science*, vol. 255, no. 5041, pp. 209–212, 1992.

[264] S. Ghiasvand and F. M. Ciorba, “Towards Adaptive Resilience in High Performance Computing,” in *Proceedings of WiP in 25th EUROMICRO International Conference on Parallel, Distributed and Network-Based Processing* (E. Grosspietsch and K. Kloeckner, eds.), vol. 1, (St. Petersburg, Russia), pp. 5–6, SEA-Publications-Austria, 6.

[265] D. Duce, “Portable Network Graphics Specification.” w3.org, 2018.

List of Figures

List of Tables

Appendix

A Neural Network Models

In order to extend the known set of correlations among nodes and failures, in addition to the main behavioral analysis method and the hierarchical temporal memory (HTM) model, three alternative neural network models are proposed. For the first and second model, syslog entries are transformed into images and processed via image processing techniques, while the third model uses a text auto-completion technique to predict the upcoming events [260].

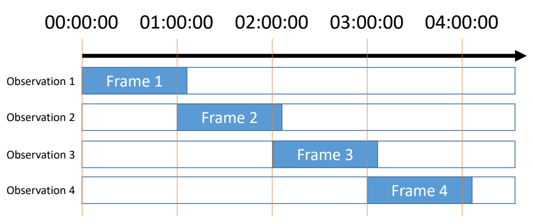
Image Processing

Many periodic events in HPC systems have static time intervals. The longest interval between two consecutive occurrences of a periodic event on Taurus is 60 minutes, thus, every periodic syslog entry appears at least once during an hour. Therefore, the observation window of one hour was chosen to monitor Taurus behavior. To simplify future calculations, the width of observation window is extended to 64 minutes[1]. However, in each observation, the window is shifted forward by 60 minutes such that the observation window always starts exactly on the hour. Hereafter, the data which is captured in an observation window is referred to as a *frame*. Figure A.1a shows the shifting of observation window to capture data frames for a duration of four hours.

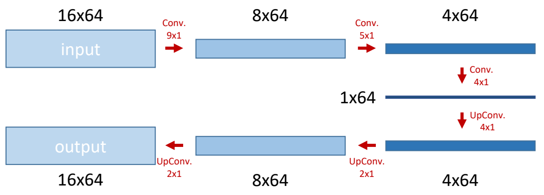
Each frame is represented as a two-dimensional matrix of F_e, with t columns and n rows. t represents the time bins of one-minute and is equal to the width of the observation window (64) and n is the number of nodes which have been observed. The value of each cell (v_{nt}) denotes the re-occurrences of event e for all events of the same `severity` level[2] within the time bin of t on node n. A sample frame is shown in Figure A.1c(a), which represents all events with the `severity` level of emergency that occurred on 18 adjacent computing nodes (a rack) during a 64-minute time window. Two nodes (rows) are randomly chosen to be removed from the frame to simplify the future calculations[3]. To eliminate potential accuracy penalties caused by random node removals, two different copies of each frame are generated. For the second copy, two nodes other than those which were removed from the first copy are randomly chosen to be removed. During the learning phase, networks are trained on both copies.

For each of the eight syslog `severity` levels, a separate frame is captured as shown in Figure A.1c(b). Frames containing *emergency*, *alert*, and *critical* events are merged (accumulated) into a single frame. Similarly, the *error* frame is merged with *warning* frame, and the

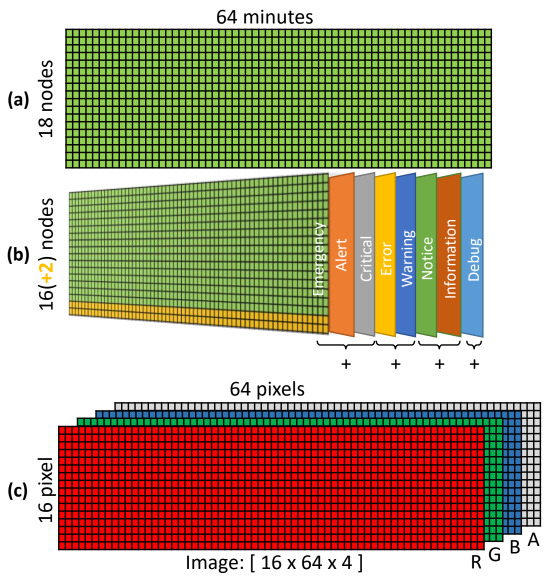
[1] 64 is a power of two (2^6).
[2] Table 2.2 on page 9 provides a complete list of syslog severity levels.
[3] 16 is a power of two ($18 - 2 = 2^4$).



(a) Capturing data frames from monitoring data.



(b) The lightweight convolutional autoencoder for extracting behavioral patterns. Encoding layers perform the majority voting and noise mitigation. While decoding layers are fine tuning the results.



(c) Transforming syslog entries to an RGBA image. Value of each cell indicates the number of event re-occurrences per node and per minute. (a) 2D representation of syslog entries. (b) Removing extra rows and merging events with similar severity levels. (c) RGBA representation of syslog entries.

Figure A.1: Anomaly detection using image processing techniques

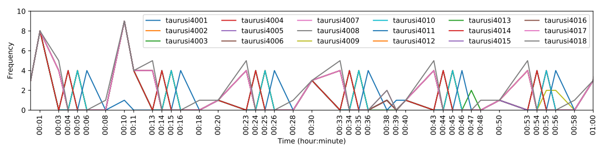
notice frame is merged with *information* frame. The *debug* frame remains unchanged. Furthermore, the values of each cell is normalized to the range of 0 and 255. The four resulting frames are stored as a three-dimensional matrix shown in Figure A.1c(c), representing a 16 by 64 pixels RGBA PNG image [265].

Two different autoencoders are designed to model the normal behavior of Taurus via image data. Both networks are trained via a sequence of 24 RGBA images in size of 16 by 64 pixels. The first approach trains a state-less convolutional autoencoder shown in Figure A.1b. The expected output of the network is the same image (frame) as the input.

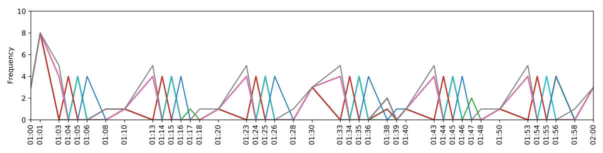
The second approach trains a long short term memory (LSTM) autoencoder. The expected output of the network in this approach is the next image (frame) in the input sequence. In another word, the network should predict the next image of the sequence. A similar network as shown in Figure A.1b is used, with the convolutional layers substituted by convolutional LSTMs.

Using the neural network, the hidden correlations among nodes and failures within node vicinities can be automatically detected and used to improve the failure prediction accuracy. As described in Chapter A on page 137 two models of neural networks were defined. The first model works based on image processing while the second model employs text auto-completion techniques. Both models perform the analysis inside relevant nodes vicinities. The image processing model has the additional advantage of providing intuitive insights about the system behavior instantly, since its input and output are both figures.

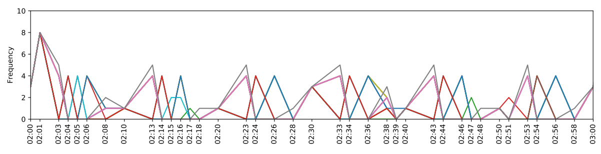
Figure A.2 shows the behavior of 18 nodes from Island 4.



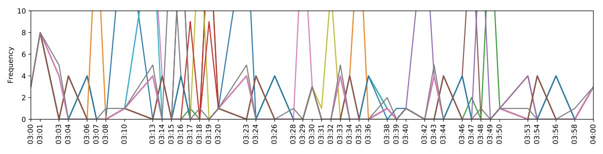
(a) From 00:00:00 to 01:00:00



(b) From 01:00:00 to 02:00:00



(c) From 02:00:00 to 03:00:00



(d) From 03:00:00 to 04:00:00

Figure A.2: Hourly pattern of computing nodes Taurusi4001-Taurusi4018. Between $03:00$ and $04:00$ a.m. some nodes project abnormal behavior.

Each sub-figure shows a time interval of 60 minutes starting from a full hour to the next full hour. Except Figure A.2d which indicates some abnormal behaviors, the hourly patterns of all nodes are identical. Both models of neural networks are designed based on this recurring hourly patterns[4].

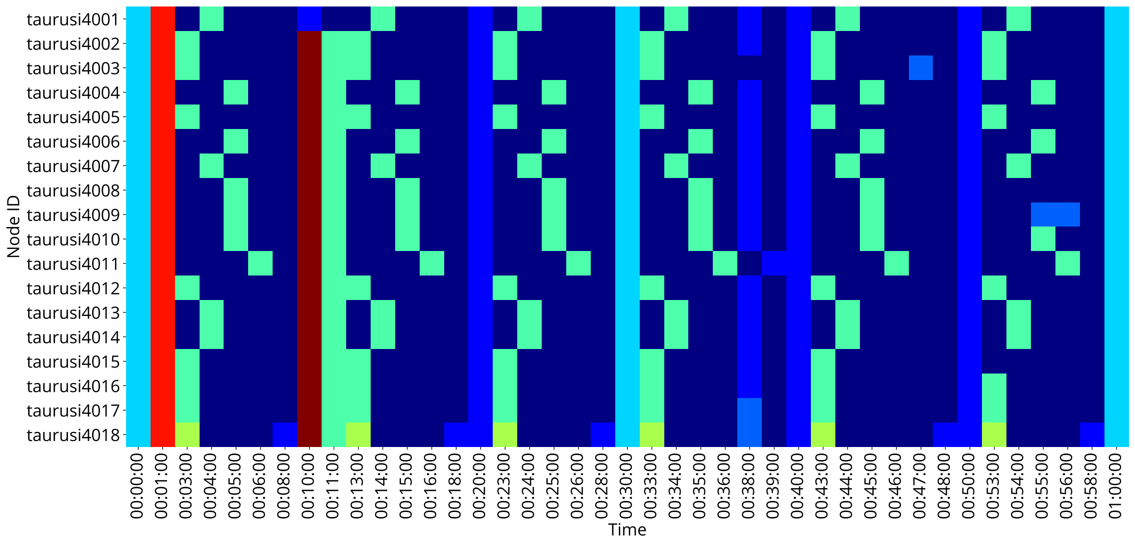
The `Keras` python library is used for the implementation of neural networks in this work. The image processing model was able to automatically extract strong hidden correlations among node behaviors inside their node vicinity. Figure A.3a demonstrates the image representation of Figure A.2a that can be directly processed via image processing model.

The left side of Figure A.3b shows the automatically extracted behavioral pattern for nodes taurusi4001-taurusi4018 from a set of 24 frames[5] similar to the frame shown on the right side of Figure A.3b. Vertical concentrations of similar color as well as shades of blue are identifying the significant regions of the behavioral pattern. This model will be updated continuously after each 60 minutes. Derivation of nodes behavior from the extracted "normal" pattern is concluded as a behavioral anomaly. The performance of Image Processing model is highly dependent on the correct selection of node vicinities. Figure A.3c shows the behavioral pattern extracted from a more homogeneous node vicinity.

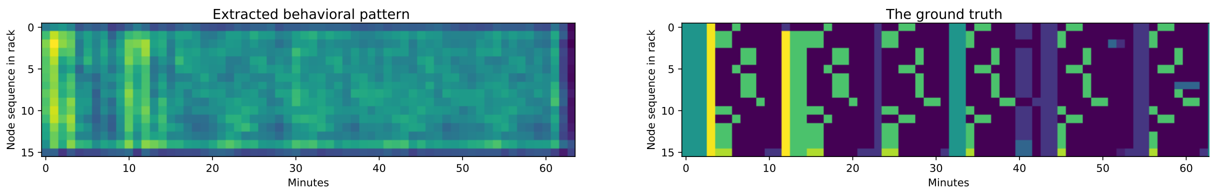
Text Auto-completion

The third NN model uses a text auto-completer. The input of network is a sequence of anonymized syslog entries (event classes) and the expected output is the upcoming entries. In another word, this network predicts the future events and completes the sequence. The event class of each syslog entry is a *word*. A sequence of 60 words (one word per minute) forms a *sentence*, and 24 sentences form a *text* (one day). Multiple occurrences of an identical event within a minute are ignored, and the occurrence of concurrent distinct events are accumulated. Empty bins (minutes) are filled with the event class of the previous

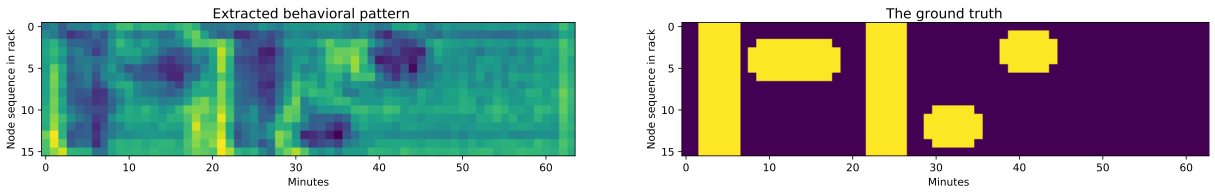
[4]The recurring hourly pattern is a common behavior in all HPC systems.
[5]Each frame covers the interval of one hour therefore, the 24 frames cover an entire day.



(a) Hourly pattern of 18 nodes (taurusi4001-taurusi4018)



(b) Significant hidden correlations among behavioral patterns of nodes taurusi4001-taurusi4018



(c) Improvement of automatic behavioral pattern extraction in more homogeneous node vicinities

Figure A.3: Automatic extraction of behavioral pattern using Image Processing model

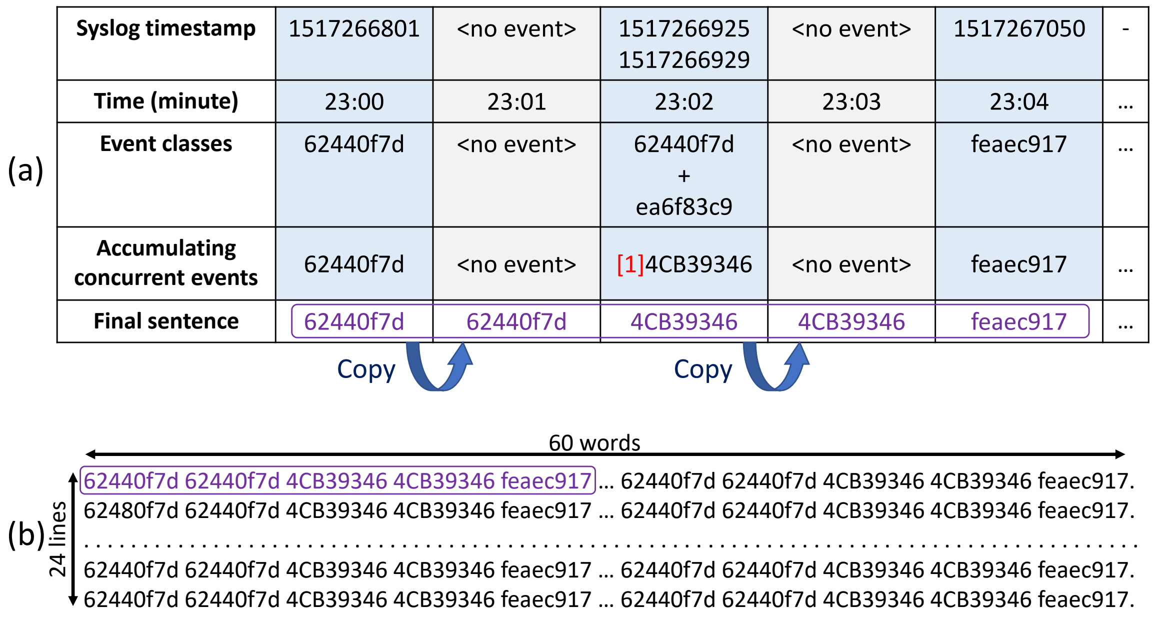
bin. Figure A.4(a) transforms the sample syslog entries shown in Table A.1 into a five-word (incomplete) sentence. Example of a text is shown in Figure A.4(b).

Table A.1: Sample of syslog entries with their respective severity level and event class

Timestamp	Source	Message	Severity	Event class
1517266801	taurusi1020	(siavash) CMD (/usr/bin/check)	Information	62440f7d
1517266925	taurusi1020	(root) CMD (/fast/sbin/start)	Information	62440f7d
1517266929	taurusi1020	Accepted publickey for siavash from 192.68.31.32	Notice	ea6f83c9
1517267050	taurusi1020	pam_unix(sshd:session): session opened for user siavash	Notice	feaec917

The network used in this approach consists of two layers, a dense layer attached to an LSTM. To increase the accuracy of detection mechanism, similar to other methods proposed in this work, only the data collected from nodes in the vicinity of each other are compared.

The preliminary results of Text Auto-completion model were promising. However, the technical challenges of detecting very long recurring sequences of symbols in text restricts the accuracy of Text Auto-completion model. Therefore, improvement of Text Auto-completion



Syslog timestamp	1517266801	<no event>	1517266925 1517266929	<no event>	1517267050	-
Time (minute)	23:00	23:01	23:02	23:03	23:04	...
Event classes	62440f7d	<no event>	62440f7d + ea6f83c9	<no event>	feaec917	...
Accumulating concurrent events	62440f7d	<no event>	[1]4CB39346	<no event>	feaec917	...
Final sentence	62440f7d	62440f7d	4CB39346	4CB39346	feaec917	...

```
62440f7d 62440f7d 4CB39346 4CB39346 feaec917 ... 62440f7d 62440f7d 4CB39346 4CB39346 feaec917.
62480f7d 62440f7d 4CB39346 4CB39346 feaec917 ... 62440f7d 62440f7d 4CB39346 4CB39346 feaec917.
.......................................................................................
62440f7d 62440f7d 4CB39346 4CB39346 feaec917 ... 62440f7d 62440f7d 4CB39346 4CB39346 feaec917.
62440f7d 62440f7d 4CB39346 4CB39346 feaec917 ... 62440f7d 62440f7d 4CB39346 4CB39346 feaec917.
```

Figure A.4: Failure prediction via text auto-completer. (a) Transforming sample syslog entries from Table A.1 into an incomplete sentence. (b) 24 hours of syslog entries represented as 24 sentences with the constant length of 60 words.

model is postponed as part of the future work.

B External Tools

To gain a better understanding of Taurus's behavior, during the early stages of behavioral analysis in this work, multiple external tools have been used. However, none of these tools are required for reproducing the results of this work. Table B.1 provides a list of external tools that were used to analyze Taurus behavior. Majority of tools listed in Table B.1 are data visualizers. Visualization is an outstanding method to achieve a global overview on potential patterns in large datasets such as `taurusCleansed` in this work. A complementary web page[1] (*Logalyst*) provides additional visualizations (e.g., interactive, animated) to assist better understanding of the data used in this work.

Table B.1: External tools used in this work

Tool	**Purpose**
Kibana	Visualizing syslog entries » github.com/elastic/kibana
Elasticsearch	Storing syslog entries (online analysis) » github.com/elastic/elasticsearch
SQLite	Storing syslog entries (offline analysis) » sqlite.org/src
Logstash	Preprocessing and unifying syslog entries » github.com/elastic/logstash
SAM Tools	Manipulating sequence alignment map (SAM) files » github.com/samtools/samtools
UGENE	Visualizing SAM files » github.com/ugeneunipro/ugene
Tablet	Visualizing SAM files » github.com/cropgeeks/tablet
New Genome Browser (NGB)	Visualization of structural variations » github.com/epam/NGB
GenomeView	Interactive visualization of string sequences » genomeview.org
PHIRE	Detecting unusual sequence patterns in string sequences » www.biw.kuleuven.be/logt/PHIRE.htm

The main data analysis tasks were conducted using Python scripts. Simple data handling tasks were performed using GNU tools[2] and bash scripting. In addition, various python libraries[3] were used for visualization (e.g., seaborn, Matplotlib, Plotly), statistical analysis (e.g., NumPy, SciPy, Pandas) and machine learning (e.g., TensorFlow, Keras, NLTK, NuPIC). Majority of scripts are accessible on *Logalyst* web page. These scripts were developed solely as a proof of concept and for the purpose of this study. Therefore, advanced programming techniques, complex structures and all complexities that could reduce the readability of source codes were intentionally avoided. Consequently, the algorithms behind each script can be better understood although the performance can still be improved.

[1]https://logalyst.github.io
[2]https://www.gnu.org/manual/blurbs.html
[3]https://wiki.python.org/moin/NumericAndScientific

C Structure of Failure Metadata Databse

The `taurusMETA` database[1] contains the metadata that are used in this work to analyze Taurus behavior. Figure C.1 shows a schema of the `taurusMETA` database.

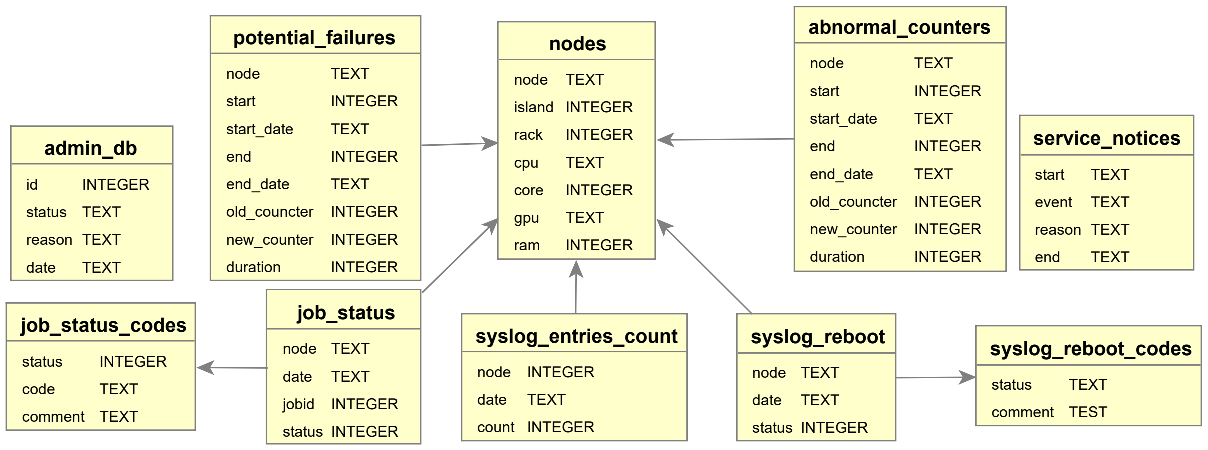


Figure C.1: Schema of `taurusMETA` database

Relations[2] are omitted from this view to improve readability. The extracted ground truth is stored in the `potential_failures` table. Data in `nodes` table provides required information about the Taurus topology. Tables `job_status_codes` and `syslog_reboot_codes` contain the definition of their respective numeric codes. Table `syslog_entries_count` stores the number of syslog entries per node per day. The number of syslog entries per day is used to quickly identify offline nodes (e.g., the nodes under maintenance) as well as extreme outliers. The `status` of each record in `syslog_reboot` table indicates the occurrence of a crash, shutdown or reboot on the respective `node`.

After each failure, system administrators investigate the reason behind it. This information in addition to node `id`, `status` and the relevant `timestamp` are stored in table `admin_db`. Table `job_status` stores the `status` of all submitted jobs. All scheduled maintenance periods and significant outages are reported via an internal mailing list. Table `service_notices` stores all service notification.

As discussed in Section 3.4, syslog-ng internal metrics can be monitored to detect the abnormal behaviors of syslog daemon. The `abnormal_counters` table stores the time intervals in which syslog-ng's internal metrics were abnormal. Although, these intervals do not necessarily indicate a general abnormal behavior, the probability of observing failures in these intervals is significantly higher. Therefore, the interplay of `abnormal_counters` data and other available information such as `syslog_reboot` reveals failure intervals. All potential node failures that occurred in the year 2017 are illustrated in Figure 3.25.

[1]More details in Section 3.4
[2]Foreign keys

D Reproducibility

This appendix provides technical details required to execute *ȷam–e ȷam* and reproduce the results of this work. Regardless of the underlying operating system, the only required software package is $Python$. All parts of *ȷam–e ȷam* except the HTM model are implemented using $Python 3.x$. Since the $Python$ library of `NuPIC` is only compatible with $Python 2.7$, the HTM related sections of *ȷam–e ȷam* are implemented in $Python 2.7$. The main libraries used in this work are shown in Table D.1[1].

Table D.1: Main libraries used for implementation of the *ȷam–e ȷam* prototype

Library	**Version**	**Library**	**Version**	**Library**	**Version**
Python	**2.7**	Python	**3.7**	redis	2.10.6
Cython	0.29.7	calmap	0.0.7	regex	2018.8.17
DBUtils	1.1	csuffixtree	0.3.3	scikit-learn	0.19.2
matplotlib	2.2.4	Flask	1.0.2	scipy	1.1.0
memory-profiler	0.55.0	graphviz	0.10.1	seaborn	0.9.0
numpy	1.12.1	html5lib	1.0.1	simplejson	3.16.0
nupic	1.0.5	Keras	2.2.2	sqlite-bro	0.8.11
nupic.bindings	1.0.6	matplotlib	2.2.3	suffix-trees	0.2.4.4
nupic-studio	1.1.3	memory-profiler	0.55.0	tensorboard	1.9.0
pysha3	1.0.2	nltk	3.3	tensorflow	1.9.0
redis	2.10.5	numpy	1.14.5+mkl		

Example D.1: Expected format of syslog entries for PαRS

```
EPOCH DATE TIME NODE FACILITY SEVERITY [PID] DAEMON <>  MESSAGE
```

The current implementation of PαRS[2] accepts a stream of syslog entries as its input. To use *ȷam–e ȷam* right out of the box, the syslog entries must have the structure shown in Example D.1. However, only the `EPOCH, NODE, FACILITY, SEVERITY` and `MESSAGE` are considered and the rest are ignored. Therefore, the non-required fields can be filled with dummy data. The main use of $<>$ symbol beside facilitating the correct division of syslog entry into its structured (metadata) and unstructured (message) parts is to verify the consistency and integrity of syslog entry. A valid Syslog entry must have 8 space-separated fields before the $<>$ symbol and 1 non-empty string after the $<>$ symbol. System logs with other combinations of fields and separators are considered invalid, thus ignored. Additional technical details are available at $logalyst$ online repository.

[1]These libraries may have dependencies. The $Python$ installer (pip) automatically resolves and installs the required dependencies.

[2]The anonymizer module of *ȷam–e ȷam*

E Publicly Available HPC Monitoring Datasets

Name of Dataset	Origin	Duration of Monitoring	Data	URL to Access
LANL	22 HPC cluster systems	Dec. 1996 - Nov. 2005	Records of cluster node outages, workload logs and error logs	usrc.lanl.gov/data/failure-data.php
HPC4	Thunderbird, Spirit, Liberty, BlueGene/L	Various duration in 2004 - 2006	Event and system logs	usenix.org/cfdr-data#hpc4
OpenCloud	Hadoop Cluster - 64 nodes	May 2010 - Dec. 2011	Job traces	ftp.pdl.cmu.edu/pub/datasets/hla
ATLAS	Various HPC Clusters	Various duration	Job traces	ftp.pdl.cmu.edu/pub/datasets/ATLAS/
PNNL[1]	A HPC cluster - 980 nodes	Nov. 2003 - Sep. 2007	Log of hardware failures	usenix.org/system/files/failuredata.zip
Cray	One or more Cray XT series machines	3 Month	Event logs, syslog, console logs	usenix.org/cfdr-data#cray
Blue Gene/P	Blue Gene/P	Jan. 09 - Aug. 09	Reliability, Availability, and Serviceability log (RAS)	usenix.org/cfdr-data#bgp
ASK.com	A server farm - 212 nodes	Dec. 06 - Feb. 07	Memory error data	cs.rochester.edu/research/os/memerror/
NERSC	Various HPC clusters	2001 - 2006	Database with I/O specific failures	fta.scem.westernsydney.edu.au/
HPC2	A HPC cluster - 256 nodes	Jan. 2004 - Jul. 2006	Hardware replacement log	fta.scem.westernsydney.edu.au/
Mogon	A HPC cluster - 512 nodes	Jul. 2018 - Jan. 2019	Failure statistics and sensor values	mediatum.ub.tum.de/1506655
D.A.V.I.D.E.	45 OpenPower8 nodes	Mar. 2018 - May 2018	Physical sensors to performance counters	zenodo.org/record/3251873
Antarex[2]	Two Intel Xeon E5-2630 v3	4-12 days in 2018	System traces (CPU, Memory, Hard disk)	zenodo.org/record/1453949
Google	Various Google cells	Various duration in 2009, 2011, 2019	Job and machine traces	github.com/google/cluster-data
ECMWF[1]	Various systems	Various duration	Mainly storage traces	iotta.snia.org/traces/
OpenStack[2]	Cloud provider - 10 nodes	N/A	Event traces	doi.org/10.5281/zenodo.1144100
GWA	Various grids	Various duration in 1998 - 2001	Workload traces	gwa.ewi.tudelft.nl/datasets/
MAUI	Various clusters	Various duration	Workload/Resource traces	supercluster.org/research/traces/
PWA	Various systems	Various duration in 1993 - 2018	Workload traces	cse.huji.ac.il/labs/parallel/workload/logs.html

[1] Requires registration.
[2] Fault injection techniques are used.

F Glossary

Jam–e Jam	Proper noun; a cup of divination in ancient mythology. Pronounced as Jām-e Jam..
Confidence	An indication of how often the rule has been found to be true. Confidence of a rule $A => B$ is the support of the rule divided by the number of sequences containing the items of A.
error	A triggered fault (e.g., $\frac{x}{y=0}$).
event	A change in the system. Event can emit a message.
failure	The event in which a component fails to perform its expected functionality.
failure chain	A sequence of successive identical failures.
failure correlation	Interpretation of a set of events that happen within a common dimention.
fault	A defect within the systems' components (e.g., $\frac{x}{y}$).
golden interval	The time interval between detection of an anomaly and occurrence of the subsequent failure.
	.
node failure	An event in which a computing node cannot perform any useful function.
PαRS	A system log anonymization method for preserving Privacy and Reducing Storage.
regular failure	An unintential failure caused by internal factors.
Support	The number of sequences in which a certain pattern appears.

Taurus	Taurus HPC cluster; a set of tightly coupled clusters located in Dresden, Germany.
UNIX time	Also known as POSIX time or UNIX Epoch time. It is equal to the number of seconds that have elapsed since 00:00:00 January 1st 1970 in UTC timezone..

G Acronyms

ET	event time
FN	false negative
FP	false positive
HPC	high performance computing
IET	inter-event time
MTBF	mean time between failure
MTTI	mean time to interrupt
NER	named entity recognizer
PET	pre-event time
PoET	post-event time
syslog	system log
TN	true negative
TP	true positive

Disclaimer