Syslog Processing for Switch Failure Diagnosis and Prediction in Datacenter Networks

Shenglin Zhang † ‡ § Weibin Meng † ‡ § Jiahao Bu † ‡ § Sen Yang ¶ Ying Liu † § Dan Pei * ‡ §
Jun (Jim) Xu ¶ Yu Chen ‖ Hui Dong ‖ Xianping Qu ‖ Lei Song ‖
† Institute for Network Sciences and Cyberspace, Tsinghua University, Beijing, China
‡ Department of Computer Science and Technology, Tsinghua University, Beijing, China
§ Tsinghua National Laboratory for Information Science and Technology (TNList), Beijing, China
¶ School of Computer Science, Georgia Institute of Technology, Atlanta, GA, USA
‖ Baidu, Inc, Beijing, China
Email: {zhangsl12, mwb16, bjh16}@mails.tsinghua.edu.cn, sen.yang@gatech.edu, liuying@cernet.edu.cn,
peidan@tsinghua.edu.cn, jx@cc.gatech.edu, {chengyu07, donghui02, quxianping, song_lei}@baidu.com,

***Abstract*—Syslogs on switches are a rich source of information for both post-mortem diagnosis and proactive prediction of switch failures in a datacenter network. However, such information can be effectively extracted only through proper processing of syslogs, *e.g.,* using suitable machine learning techniques. A common approach to syslog processing is to extract (*i.e.,* build) templates from historical syslog messages and then match syslog messages to these templates. However, existing template extraction techniques either have low accuracies in learning the "correct" set of templates, or does not support incremental learning in the sense the entire set of templates has to be rebuilt (from processing all historical syslog messages again) when a new template is to be added, which is prohibitively expensive computationally if used for a large datacenter network. To address these two problems, we propose a frequent template tree (FT-tree) model in which frequent combinations of (syslog) words are identified and then used as message templates. FT-tree empirically extracts message templates more accurately than existing approaches, and naturally supports incremental learning. To compare the performance of FT-tree and three other template learning techniques, we experimented them on two-years' worth of failure tickets and syslogs collected from switches deployed across 10+ datacenters of a tier-1 cloud service provider. The experiments demonstrated that FT-tree improved the estimation/prediction accuracy (as measured by F1) by 155% to 188%, and the computational efficiency by 117 to 730 times.**

I. Introduction

As nowadays cloud service providers employ a large number of switches in their datacenter networks, switch failures have become a fact of life to be dealt with. For example, tens of thousands of switches are deployed in Microsoft's datacenter network [1], in which 400+ switch failures occur each year [2]. As a switch failure often has a significant impact on the performance of a datacenter network, much research efforts have been devoted to the diagnosis or proactive detection of switch failures, resulting a host of diagnosis or proactive failure detection techniques in recent years [3]–[6].

Syslog messages generated by these switches (and routers) have long been recognized as a rich source of information for (switch) failure diagnosis, detection, or prediction [6]–[9]. However, since syslogs are usually unstructured texts, they have to be properly processed (*e.g.,* using machine learning techniques) before they can be effectively used for such purposes [10], [11]. In this paper, we focus on *syslog processing* techniques that can lead to more accurate and efficient switch failure diagnosis, detection, or prediction. The current approach to switch syslog processing is to extract message templates from syslog messages and then match the syslog messages to the templates [6]–[8]. For example, in the syslog message **Interface ae0, changed state to down**, **ae0** is a parameter (word) that varies from one message to another and is not a part of a template, whereas the rest, *i.e.,* **Interface ..., changed state to up** sketches out the event and hence is a template that summarizes this and other similar syslog messages.

Despite the crucial role that syslog processing plays in switch failure diagnosis, detection, or prediction, existing syslog processing techniques have low accuracies in learning the "correct" set of templates (*e.g.,* Statistical Template Extraction (STE) [8] and LogSimilarity [6]), or do not support incremental learning in the sense the entire set of templates has to be rebuilt (from processing all historical syslog messages again) when a new template is to be added (*e.g.,* the signature tree based method [1] [7] and STE [8]), or both (*e.g.,* STE [8]). For example (concerning the "low accuracies" case), with STE, some syslog messages cannot match to any template, and with LogSimilarity, two syslog messages could be matched to the same template even when they represent very different events and should be matched to different templates.

Syslog-based failure diagnosis or prediction usually employs machine learning techniques to discover patterns from historical failures [6], [8], [12], [13]. The underlying diagnosis or prediction model, consisting of the set of templates and the associated statistics, needs to be kept up-to-date, through retraining, every once in a while. It is highly desirable for the set of templates to be incrementally retrainable for the following reason. Network operators frequently conduct software or firmware upgrades on switches to introduce new features,

* Dan Pei is the correspondence author.

[1]Hereafter, we collectively refer to this method as Signature Tree

or fix bugs in the previous version [14], [15]. These updates can generate new *subtypes* of syslog messages that cannot match to any existing template, thus requiring new templates to be extracted from these new messages and added to the set. If the template extraction model (*i.e.,* the set of templates) is incrementally retrainable, only the syslog messages that arrive after the previous (retraining) update need to be matched to the new set of templates (possibly with new templates added as a result of software/firmware upgrades); otherwise, all historical syslog messages have to be reprocessed (*i.e.,* retrained) according to the new set of templates. The latter case (not incrementally retrainable), which STE and Signature Tree fall into, is prohibitively expensive computationally if used for a large datacenter network that employs tens of thousands of switches and "learns" from historical syslogs and failure tickets accumulated over a long period of time (say two years).

We propose a novel frequent template tree (FT-tree) technique that both has high accuracy in identifying "correct" templates, and is incrementally retrainable. Based on the observation that a "correct" message template is usually a combination of words that occur frequently in syslog messages, FT-tree dynamically maintains a tree structure of such frequent words that implicitly defines the set of message templates. Since this implicitly-defined set of message templates dynamically and "gradually" (*i.e.,* incrementally) evolves with the arrival of new syslog messages, which may be of the new message *subtypes* (due to the aforementioned software/firmware upgrades), FT-tree is naturally incrementally retrainable. Intuitively, FT-tree also empirically guarantees that, with high probability, the template extracted from a given syslog message accurately characterizes the event that the message describes.

To compare the performance of FT-tree to those of Signature Tree, STE, and LogSimilarity, we applied failure tickets and syslogs collected from $2,223$ switches deployed across $10+$ datacenters owned by *a tier-1 cloud service provider* over a two-year period. We evaluated the accuracies of the four techniques based not only on manual classifications of syslog messages, but also on the *failure prediction* results. FT-tree and Signature Tree achieve the same accuracy, and they both improve the prediction accuracy by 155% to 188% compared to STE and LogSimilarity. However, Signature Tree and STE respectively consume 730 times and 117 times more computational resources than FT-tree, as they are not incrementally retainable. The evaluation results clearly demonstrate the benefits of FT-tree: highly accurate and incrementally retrainable.

The rest of the paper is organized as follows. We provide an introduction to switch syslog, switch failure diagnosis and prediction, and the intuition behind extracting templates in Section II. The design of FT-tree is described in Section III. The evaluation of FT-tree is presented in Section IV. The related works, including Signature Tree, STE and LogSimilarity, are discussed in Section V. Finally, we conclude our paper in Section VI.

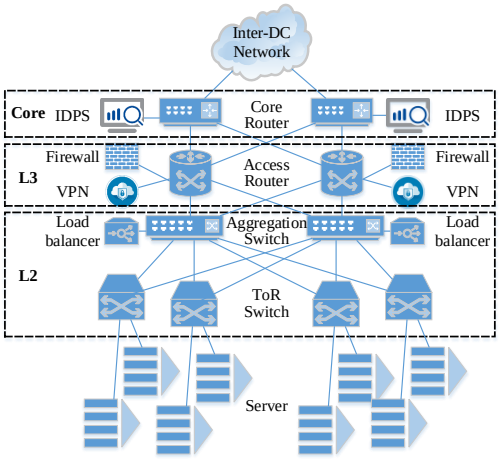

Fig. 1: Typical datacenter network architecture

II. Background

In this section, we first describe switch syslogs in details in Section II-A. Then in Sections II-B and II-C respectively, we introduce syslog-based failure diagnosis and prediction, and describe the intuitions behind extracting templates for such purposes.

A. Switch Syslogs

Each switch reports, from time to time, the observed hardware/software condition or (anomalous) event, in a syslog message. Examples of such conditions or events include state changes of interfaces, links, or neighbors (*e.g.*, the state of an interface changes from up to down), operational maintenance (*e.g.,* operators log in/out), environmental condition alerts (*e.g.,* high temperature), *etc.* Although syslog messages are designed mainly for debugging software and hardware problems of the switches, they can also be used for root cause analysis of a network incident. Hence, usually dedicated servers are deployed in a datacenter network to collect syslog messages from all its switches.

As can be seen from several example syslog messages shown in Table I, a syslog message usually has a simple structure consisting of several fields, including a timestamp recording when the switch generated the syslog message, a switch ID identifying the switch that generated the message, a message type that describes the rough characteristics of the message, and a message body depicting the details of the event. The syntax and semantics of the *message type* field and the *detailed message* field vary with switch vendors and models.

B. Failure Diagnosis and Prediction

A switch failure occurs when the service provided by a switch (traffic forwarding) deviates from the norm [16], [17]. Switch failures are known to significantly impact the performance of datacenter networks. For example, the switch failures in the datacenters of Colo4 [18], and Hosting.com [19] all brought outages to their datacenters. Although switches in datacenter networks have built-in failover technologies, such

TABLE I: Examples of switch syslog messages

Vendor	Time stamp	Switch ID	Message type	Detailed message
Vendor 1	Jan 23 14:24:41 2016	Switch 1	SIF	Interface te-1/1/8, changed state to up
Vendor 1	Mar 19 15:04:11 2016	Switch 4	OSPF	A single neighbour should be configured
Vendor 1	Apr 16 08:07:19 2016	Switch 4	lacp	Attempt to send lacpdu on port(38) from lag failed,Transport failed
Vendor 2	Apr 21 14:53:05 2016	Switch 11	10DEVM/2/POWER_FAILED	Power PowerSupply1 failed
Vendor 2	Sep 23 00:10:39 2015	Switch 13	10IFNET/3/LINK_UPDOWN	GigabitEthernet1/0/18 link status is DOWN
Vendor 2	Nov 8 07:29:06 2015	Switch 17	10CFM/5/CFM_SAVE CONFIG_SUCCESSFULLY	Configuration is saved successfully

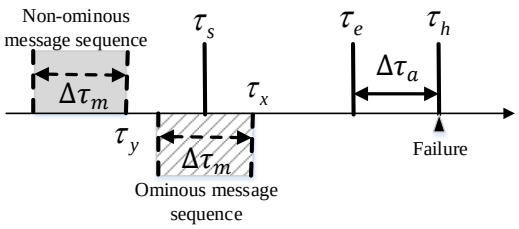

Fig. 2: The model of switch failure prediction. For a given switch failure occurred at τ_h, our mission is to predict the failure during $[\tau_s, \tau_e]$. τ_e is $\Delta\tau_a$ before τ_h for network operators need at most $\Delta\tau_a$ to react to a positive failure prediction. For any τ_x in $[\tau_s, \tau_e]$, the syslog sequence in $[\tau_x - \Delta\tau_m, \tau_x]$ is an ominous message sequence, while the syslog sequence in $[\tau_y - \Delta\tau_m, \tau_y]$ is a non-ominous message sequence when $\tau_y \notin [\tau_s, \tau_h]$.

as replicating data for services and deploying a backup switch for redundancy, they do not "mask" all switch failure scenarios [16] (see Figure 1). Therefore, it is important to accurately and quickly diagnose switch failures so that measures can be taken to effectively and promptly mitigate the service degradation brought by the failures. Furthermore, if we can accurately predict switch failures, carefully reroute the traffic, and replace the switches before failures actually happen, we can keep the impact of switch failures on datacenter network performance to a minimum.

In this paper, we focus on diagnosis or prediction of a single switch failure, and do not consider the relationship between two spatially and temporally related switch failures. In addition, we do not explore the detailed network topology of data center network in this work.

While syslog-based network device failure diagnosis has been a well formulated problem [6], [7], [20], syslog-based failure prediction has not been investigated before. Hence, we derive the switch failure prediction problem formulation from that of the failure prediction for computer systems [17]. More specifically, the objective of switch failure prediction is to determine *in real time* whether a switch failure will happen in the *near future*, based on the *the switch and network status information contained in the syslog messages*.

Fig. 2 illustrates the task of switch failure prediction. For this task, the (continuous) time is discretized into small time bins of fixed duration (say 15 min) and numbered as integers. Suppose a failure occurs at τ_h. The mission of switch failure prediction is to predict the failure at any time bin τ_x within $[\tau_s, \tau_e]$. Because network operators need at most $\Delta\tau_a$ time to react to a positive failure prediction, such as to reroute the traffic and to replace the failure-looming switch, τ_e is ahead of τ_h by $\Delta\tau_a$. While making the duration $[\tau_s, \tau_e]$ longer always leads to a higher precision of failure prediction (*e.g.,* if we set this duration to ∞, a positive failure prediction guarantees to be correct), if the duration of $[\tau_s, \tau_e]$ is too long, the (positive failure) prediction is of little use as it does not provide a clear guidance to operators as to whether the switch at issue needs to be replaced and when. Hence, in this work, we try to make this duration meaningfully small.

The failure prediction at any time τ_x in Fig. 2 is based on the statistical analysis (*i.e.,* machine learning) of the switch syslog messages within the interval $[\tau_x - \Delta\tau_m, \tau_x]$ leading up to τ_x. Hence, we refer to the syslog message sequence within the interval $[\tau_x - \Delta\tau_m, \tau_x]$ simply as τ_x's corresponding message sequence. For any time τ_x, we call $\tau_x's$ corresponding message sequence or the time τ_x *ominous* if $\tau_x \in [\tau_s, \tau_e]$ – because in this case τ_x is close enough to the failure time τ_h and its corresponding message sequence should give some cues about the imminent failure – and the number of messages in the sequence is large enough (say at least θ) to allow for meaningful statistical analysis. Similarly, for any $\tau_y \notin [\tau_s, \tau_h]$, $\tau_y's$ corresponding message sequence or the time τ_y is called non-ominous – because it is not close enough to the failure time τ_h – as long as there are at least θ syslog messages in the sequence. The time bins described later all contain enough syslog messages (to exceed this threshold θ) so that it is either ominous or non-ominous. Then the failure prediction problem becomes that of classifying each time bin as either ominous or non-ominous.

C. Syslog Processing for Failure Diagnosis and Prediction by Extracting Templates

Since switch syslog messages are unstructured texts in many different forms, and a certain pair of neighboring IP addresses or interface IDs that appear in a syslog message may never appear again, it is nearly infeasible to extract patterns from raw syslog messages for failure diagnosis or prediction. Although there is a message type field (see Table I) that describes the schematic characteristics of the event in each syslog message, each message type can include multiple *subtypes*. For example, although there are 12 syslog messages in Table II that

TABLE II: Example of syslog sequence before a switch failure

Time stamp	Message type	Detailed message
9:21:10	SIF	Interface ae3, changed state to down
9:22:23	SIF	Vlan-interface vlan22, changed state to down
9:23:45	SIF	Interface ae3, changed state to up
9:25:28	SIF	Vlan-interface vlan22, changed state to up
9:27:20	OSPF	Neighbour(rid:10.231.0.42, addr:10.231.38.85) on vlan22, changed state from Full to Down
9:30:31	SIF	Interface ae1, changed state to down
9:32:34	SIF	Vlan-interface vlan20, changed state to down
9:35:25	SIF	Interface ae1, changed state to up
9:39:21	OSPF	Neighbour(rid:10.231.0.40, addr:10.231.36.85) on vlan20, changed state from Full to Down
9:41:29	SIF	Vlan-interface vlan20, changed state to up
9:41:52	OSPF	A single neighbour should be configured
9:42:50	SIF	Interface ae1, changed state to down
9:44:38	SIF	Vlan-interface vlan20, changed state to down
9:45:15	SIF	Interface ae1, changed state to up
9:47:58	SIF	Vlan-interface vlan20, changed state to up
9:49:25	OSPF	A single neighbour should be configured

TABLE III: Syslog message *subtypes* of SIF

Subtype No.	*Subtype* structure
$N1$	Interface *, changed state to down
$N2$	Interface *, changed state to up
$N3$	Vlan-interface *, changed state to down
$N4$	Vlan-interface *, changed state to up

belong to the message type "SIF" (system interconnect fabric) and describe the switch status changes collected using SIF technology, the detailed messages can be quite different. The detailed message field of syslog messages is essentially free-form text, and a switch's OS usually generates this field using the "printf" function with detailed information as variables, such as the location (line card/port/interface), package loss ratio, IP address, *etc*. For instance, the detailed message field of the syslog message in the first line of Table II, *i.e.,* **Interface ae3, changed state to down**, means that the event impacted on interface **ae3**, and, as a result, the state of the interface changed to down. In other words, the **ae3** part is the detailed information variable, whereas the rest parts, *i.e.,* **Interface ..., changed state to up**, are predefined outputs by the switch's OS and can be used as a *subtype* for the syslog messages that belong to this message type, *i.e.,* **SIF**. When we mask the variable for the detailed message field of the 12 syslog messages, *i.e.,* interface number or vlan-interface number using the same symbol, *e.g.*, an asterisk as shown in Table IV, there are only four different syslog message structures, or *subtypes*. However, manually obtaining all *subtypes* without domain knowledge is almost impossible because not every part that should be masked in the detailed message can be as easily characterizable as the interface number. In addition, although *part* of the syslog *subtypes* can be obtained with support from vendors, these *subtypes* may *change* due to software upgrades. Therefore, our objective is to automatically obtain *message templates* in which the need-to-be-masked parts are removed and the message *subtypes* are retained without relying on any domain knowledge.

III. Frequent Template Tree

Our objective in syslog processing is to automatically extract template and *subtype* information from syslog messages without relying on any domain knowledge. Although three techniques, namely Signature Tree, STE, and LogSimilarity, were proposed for this purpose, they are not well suited for our application scenario due either to their low accuracies (in template or *subtype* extraction) or to their inability to be incrementally trained. Inspired by the Frequent Pattern Tree (FP-tree) [21], we propose FT-tree, an incrementally trainable technique with high accuracy for template and *subtype* extraction. FT-tree is an extended prefix-tree structure for encoding message templates. The basic idea behind FT-tree is that, a syslog message *subtype* is usually the longest combination of words with high frequencies, and hence extracting a template is equivalent to identifying such longest combination of frequent words from syslog messages.

In the following, we first introduce the design and construction of FT-tree in Section III-A, and then demonstrate how FT-tree facilitates incremental template learning in Section III-B.

A. Design and Construction

Let $I = a_1, a_2, ..., a_m$ be the set of distinct words that occur in a message set $DM = \langle M_1, M_2, ..., M_n \rangle$, where each M_i is a syslog message. The **support** (*i.e.*, the frequency of occurrence) of a word combination (*i.e.*, a set of words) A, is the number of distinct messages containing A in DM. A is considered a template if A is occurring frequently (*i.e.*, with a large support).

The second column of Table IV shows an example syslog message set $DM = \langle M_1, M_2, ..., M_8 \rangle$, in which every message belongs to the message type "SIF" (see in the second column of Table II). The FT-tree for this DM is constructed, using the algorithm shown in Algorithm I, as follows.

First, our FT-tree construction algorithm scans DM once (line 1 in Algorithm I), and derives a *list* L of words in the descending order of their frequencies of occurrences (the number after each ":"). Clearly, $L = \langle$("changed":8), ("state":8),

TABLE IV: Different messages belonging to message type "SIF" in Table II, and the words ordered according to L

Message No.	Detailed Message	Words ordered according to L
M_1	Interface ae3, changed state to down	"changed", "state", "to", "Interface", "down", "ae3"
M_2	Vlan-interface vlan22, changed state to down	"changed", "state", "to", "Vlan-interface", "down", "vlan22"
M_3	Interface ae3, changed state to up	"changed", "state", "to", "Interface", "up", "ae3"
M_4	Vlan-interface vlan22, changed state to up	"changed", "state", "to", "Vlan-interface", "up", "vlan22"
M_5	Interface ae1, changed state to down	"changed", "state", "to", "Interface", "down", "ae1"
M_6	Vlan-interface vlan20, changed state to down	"changed", "state", "to", "Vlan-interface", "down", "vlan20"
M_7	Interface ae1, changed state to up	"changed", "state", "to", "Interface", "up", "ae1"
M_8	Vlan-interface vlan20, changed state to up	"changed", "state", "to", "Vlan-interface", "up", "vlan20"

("to":8), ("Interface":4), ("Vlan-interface":4), ("down":4), ("up":4), ("ae3":2), ("ae1":2), ("vlan22":2), ("vlan20":2)⟩.

Then, we create the root of a tree which is labeled with the message type, which in this case is "SIF". Our FT-tree construction algorithm scans DM for a second time (lines 5 through 9 in Algorithm I). The processing of M_1 leads to the construction of the first path/branch of the tree: ⟨"changed", "state", "to", "Interface", "down", "ae3"⟩ (a word in an FT-tree is called the $word\text{-}name$ of the node containing the word). Note these words are ordered according to the order of the words in L. When M_2 is processed, since its ordered word list ⟨"changed", "state", "to", "Vlan-interface", "down", "vlan22"⟩ shares a common prefix ⟨"changed", "state", "to"⟩ with the existing path/branch ⟨"changed", "state", "to", "Interface", "down", "ae3"⟩, a new branch ⟨"Vlan-interface", "down", "vlan22"⟩ is created as a subtree of node ⟨"to"⟩. The rest six messages in Table IV are processed similarly, resulting in the final FT-tree shown in Fig. 3 (the rightmost tree).

Finally, we prune the tree until it satisfies the following node degree constraint (lines 10 through 14 in Algorithm I). Intuitively, there should be only a small number of *subtypes* for each message type, and, for each *subtype*, there should be many different messages that match to it. Therefore, if a node has too many children (say exceeding a threshold k), all its children (or subtrees) are deleted from the tree and the node becomes leaf itself. In the pruned FT-tree, each root-to-leaf path is a message template (*i.e.,* type + *subtype*). For example, ⟨"SIF", "changed", "state", "to", "Interface", "down"⟩ is a message template, as shown in Figure 3.

Definition 1: As illustrated by the above example, given a specific message type, its FT-tree is defined as follows:

1) Its root is labeled by the message type, and each root-to-leaf path corresponds to a message template.
2) Each non-root node in the FT-tree has only one attribute/field, namely $word\text{-}name$, which registers which word this node represents.

It remains to describe the function $insert\_tree([p|P], T)$ in Algorithm 1. It works as follows. If $\nexists N$, $N.word\text{-}name = p.word\text{-}name$, and N is a child of T, then create a new node N, and make it a child of T. If $P \neq \Phi$, call $insert\_tree(P, N)$ recursively.

B. Incremental Template Learning

As mentioned earlier, for a given message type, new *subtypes* of messages can emerge due to OS or firmware upgrades, and new templates need to be generated for these messages to match to. As shown in Fig. 3 and Algorithm 1, this is accomplished via inserting new nodes/branches to the FT-tree. It is also clear from Algorithm 1 that such insertions can be done incrementally, by scanning only the *recently arrived* syslog messages (after the OS or firmware upgrade) twice. More specifically, when a new message M_{new} that *does not match any template* arrives, a new branch is inserted into the FT-tree by calling $insert\_tree()$ (line 8) in Algorithm 1. Please note that, if one or more words in M_{new} do not exist in L, we will add these words to the tail of L. When a switch starts to generate new *subtypes* of messages, it is highly likely that, *in one day*, these messages will contain enough number (more than the pruning threshold k) of distinct parameter words (such as "ae3", "vlan22" in Figure 3) as the children of a node, so that this node becomes a leaf itself after the pruning.

We illustrate this incremental learning process by an example shown in Fig. 3. Suppose that a new message M_{new} = "Interface ae1 changed state to RETURN" arrives, and that right before this arrival, the FT-tree is the rightmost tree shown in Figure 3, but with all leaf nodes ("ae3", "ae1", "vlan22",

Algorithm 1 FT-tree construction

Input: A message set DM that contains all the different messages of a specific message type, and a threshold k.
Output: A FT-tree, T

1: Scan the message set DM once.
2: Calculate the support for each word in I.
3: Let L be the list of words in the descending order of their supports.
4: Create the root of T and label it as the message type.
5: **for** each message in DM **do**
6: Sort its words according to their order in L
7: Let the sorted word list be $[p|P]$, where p is the first element and P is the remaining list
8: Call $insert\_tree([p|P], T)$
9: **end for**
10: **for** Child C in T **do**
11: **if** C has more than k children **then**
12: Eliminate all the children of C
13: **end if**
14: **end for**
15: **return** T

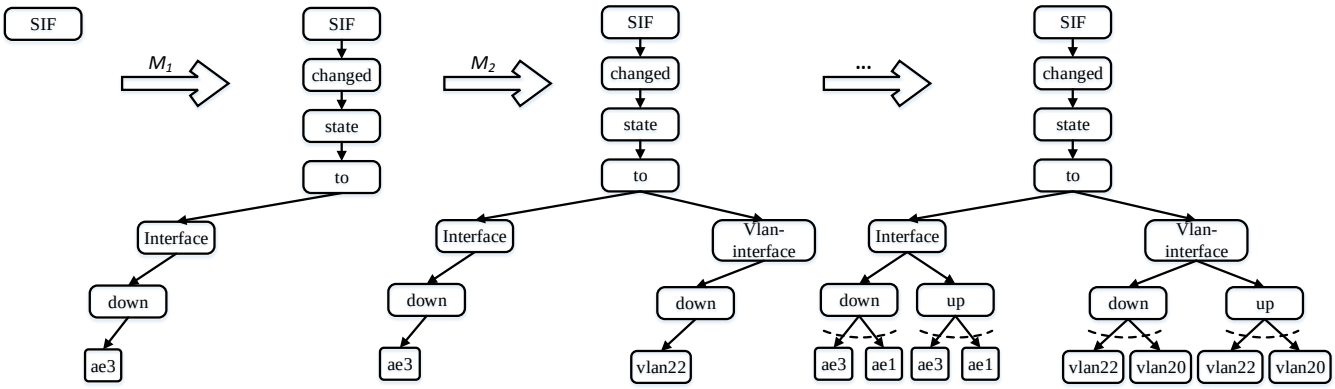

Fig. 3: Example of constructing an FT-tree

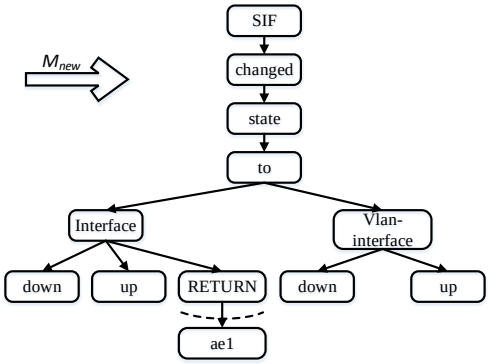

Fig. 4: FT-tree after M_{new} is added

etc.) pruned. Then the new message arrival M_{new} results in the insertion of the branch ("RETURN"→"ae1") to the FT-tree, and the new leaf "ae1" will eventually be pruned as explained earlier.

After learning the templates from syslog messages, we can match a historical syslog message or a recent syslog message to a specific template, which is encoded as a numerical value. More specifically, for a syslog sequence $(s_1, s_2, ..., s_n)$, if s_i is matched to template t_{si}, we refer to $(t_{s1}, t_{s2}, ..., t_{sn})$ as the template sequence of $(s_1, s_2, ..., s_n)$.

IV. Evaluation

In this section, we evaluate and compare the performance of FT-tree to those of Signature Tree, STE and LogSimilarity using syslogs and failure tickets collected from real-world sources. We evaluated the accuracies of these four techniques in both template learning (Section IV-B) and failure prediction (Section IV-C), and compared their efficiency in Section IV-D.

A. Data Sets

In cooperation with a *a tier-1 cloud service provider*, we collected all syslog messages over a 2-year period from all switches of a specific (switch) model across *more than 10* datacenters owned by this cloud service provider. Syslog-based failure diagnosis and prediction was then performed over this syslog data set and compared against the switch failure data, which we will explain shortly, can, to a certain extent, be considered the ground truth. We focus on this homogenous subset of switches (of the same model) because they share the same failure pattern (i.e., observable behaviors before, during, and after a failure), and hence intuitively share the same underlying statistical model for the machine-learning-based failure diagnosis or prediction.

We also collected switch hardware failure data over the same time period across the same data centers, which consist of the following three types.

Failure tickets. When a service anomaly event is detected (e.g., the average response time of the search engine deteriorates significantly for more than a minute or two) and a switch failure is suspected as the root cause (e.g., the total amount of traffic to all servers connected to a switch sees a significant drop during the same time period), a failure ticket is generated (by the data center performance monitoring system) and sent to the *network operators*. The *network operators* will then look into this ticket and determine whether the service deterioration was indeed caused by the suspected switch failure, and record this event in a *failure ticket*.

Proactive switch failure detection via SNMP polling. In the tier-1 cloud service provider we are working with, the real-time status of a switch, such as its CPU and memory utilizations and the traffic rate at each interface, is constantly monitored by the *network operators* via SNMP polling. If a problem in packet forwarding occurs at one or more interfaces, network operators will research the root cause of the failure and record this event accordingly.

Proactive switch failure detection via syslogs. Vendors usually provide operational instructions for their switch products, which include a list of syslog keywords that may indicate a switch failure. A regular expression match, by a syslog message, with any of these keywords will alert network operators of a possible switch failure. When alerted, network operators will verify whether the event is indeed a switch failure and identify the root cause of the failure if the answer is yes. However, such regular expression matching technique can only detect a small portion of switch failures (i.e., high false negative rates) in practice, and can also have false

TABLE V: Detailed information for switches

# failures	# failed switches	# switches in total	# Ominous time bins	# Non-ominous time bins
228	131	2223	1273	5,516,435

positives [16]. Therefore, in this work we employ a machine learning technique based on HSMM (Hidden Semi-Markov Model) – which uses a sequence of syslog messages rather than a single syslog message – to predict switch failures.

Since every such switch hardware failure event (of any of the above three types) was *all manually verified* by the network operators, this failure data set can serve as the ground truth for our evaluations.

Based on their experience, network operators believe that the syslogs within 24 hours before a switch failure have high predicative power; once a failure is so predicted, they need no more than 30 minutes to react to a positive failure prediction (e.g., by rerouting the traffic and replacing the switch). However, constantly maintaining and processing syslogs within a sliding window of 24 hours for the online failure prediction is too computationally expensive. Fortunately, after analyzing the syslogs right before dozens of know switch failures, we found that in most cases a 2-hour sliding window of syslog messages within 24 hours before a given failure can capture the ominous pattern. We also found that, if there are less than five syslog messages in a time bin's corresponding message sequence, the extracted template sequence would be too short to capture the ominous pattern and hence cannot be used for failure prediction. Therefore, in our evaluation experiments, we set $\Delta\tau_m = 2\ h$, $\Delta\tau_a = 30\ min$, $[\tau_s, \tau_e] = 24\ h$, $\xi = 15\ min$, and $\theta = 5$. Table V illustrates the number of hardware failures, the number of failed switches, the number of switches in total, the number of ominous time bins, and the number of non-ominous time bins.

B. Evaluation of Template Learning Accuracy

As mentioned earlier, three techniques were proposed in previous works for parsing syslog messages and learning syslog templates for network devices, *i.e.,* Signature Tree [7], STE [8], and LogSimilarity [6]. In addition, we propose a novel template extraction technique, *i.e.,* FT-tree, for accurately and incrementally learning templates from syslog messages. After analyzing the four techniques, we believe that FT-tree and Signature Tree are more accurate in template extraction than STE and LogSimilarity in our scenario (see Section V for more details). To demonstrate our analysis, we here compare the performance of the four techniques using real-world switch syslogs.

Learning templates from syslog messages is equivalent to classifying syslog messages based on the events they describe. Since the network operators analyze switch syslogs every day, they are really familiar with the event that a given syslog message represents. Therefore, we can use the network operators' manual classification results of syslog messages as the ground truth. As described in Section IV-A, we picked one switch model, and collected *all* the switches that belong to this switch model, and analyzed *all* the syslogs of the switches. That is, billions of syslog messages were analyzed for the above switches and, thus, manually classifying *all* the syslogs was prohibitive. Thereby, we randomly collected a sample of syslog messages for the evaluation as follows. We first *randomly* picked four *message types* (see Table I for the definitions) from *all* switches. For each message type, we *randomly* collected 500 syslog messages. The network operators then manually classified the syslog messages based on the event each message represents. Considering the large number of the syslog messages that should be manually classified (2000), this is a large amount of work. After that, we ran FT-tree, Signature Tree, STE, and LogSimilarity to learn the templates for the above syslog messages, respectively.

We applied the $Rand\ index$ [22] technique to quantitatively compare the accuracy of the four techniques. $Rand\ index$ is a popular technique for evaluating the similarity between two data clustering techniques. We could evaluate the accuracy for each technique by calculating the $Rand\ index$ between the manual classification results and the templates learned by the technique. Specifically, among the template learning results of a specific technique for a given message type, we randomly selected two messages, *i.e.,* x and y, and defined a, b, c, d as follows:

- a: x and y are manually classified into the same cluster and they have the same template;
- b: x and y are manually classified into different clusters and they have different templates;
- c: x and y are manually classified into different clusters and they have the same template;
- d: x and y are manually classified into the same cluster and they have different templates;

Moreover, the $Rand\ index$ can be calculated using the above terms as follows:

$$Rand\ index = \frac{a+b}{a+b+c+d} \tag{1}$$

Fig. 5 shows the $Rand\ indexes$ for FT-tree, Signature Tree, STE, and LogSimilarity among the four message types. FT-tree and Signature Tree achieved averaged close-to-1 $Rand\ indexes$ and performed excellent across all the four message types. In contrast, STE and LogSimilarity achieved relatively low averaged $Rand\ indexes$, *i.e.*, 62.10% and 59.31%, respectively. This is because STE cannot match some specific syslog messages to any template, and LogSimilarity may match some specific syslog messages belonging to different *subtypes* to the same template (see Section V for more details). FT-tree and Signature Tree are both word frequency based techniques, *i.e.,* both FT-tree and Signature Tree are constructed based on the frequency of words in messages. As a consequence, the templates extracted by FT-tree and Signature Tree are almost identical. However, FT-tree can construct a tree and learn templates incrementally, whereas Signature Tree cannot.

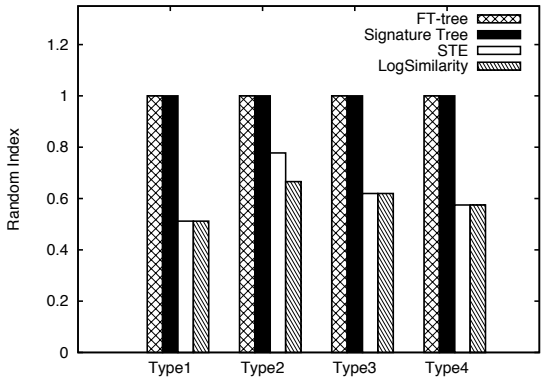

Fig. 5: Comparison of $rand\ indexes$ for FT-tree, Signature Tree, STE and LogSimilarity among four message types

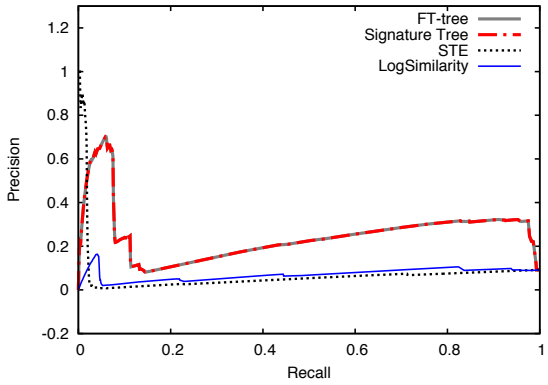

Fig. 6: Comparison of PRCs among FT-tree, Signature Tree, STE, and LogSimilarity

C. Evaluation of Failure Prediction Accuracy

Since FT-tree, Signature Tree, STE and LogSimilarity are all template extraction techniques for failure diagnosis or prediction, we evaluated the four techniques directly on the basis of not only the template learning results, but also the failure prediction results following [10], [11]. HSMM (Hidden Semi-Markov Model) is a popular failure prediction technique used for predicting failures based on logs [12]. It was demonstrated with high accuracy using data collected from commercial cellular networks. Therefore, we here applied HSMM as the failure prediction technique to demonstrate the template learning performance of FT-tree, Signature Tree, STE, and LogSimilarity.

A system's capability to predict failure is usually assessed by the following three intuitive metrics: $Precision, Recall$ and $F1\ measure$ [12], [17]. Hence we used these metrics to evaluate the performance of each technique. For a time bin, according to the ground truth provided by the network operators, we knew the outcome as either an ominous time bin or a non-ominous one. For each technique, we labeled its outcome as a true positive (TP), true negative (TN), false positive (FP), and false negative (FN). True positives were ominous time bins that were accurately determined as such by the technique, and true negatives were time bins that were accurately determined as non-ominous. If the technique determined that a time bin was an ominous one when, in fact, it was actually non-ominous, we then labelled the outcome as a false positive. False negatives were ominous time bins that were incorrectly missed by the technique. We calculated the Precision, Recall and $F1\ mesure$ as follows: $Precision = \frac{TP}{TP+FP}$, $Recall = \frac{TP}{TP+FN}$, $F1\ mesure = \frac{2*Precision*Recall}{Precision+Recall}$.

We used the *10-fold* cross validation model to evaluate the four techniques [23]. *10-fold* cross validation is a model validation technique that provides an insight on how a prediction model will generalize to an independent dataset [24]. The benefit of *10-fold* cross-validation is that all time bins are used for both training and validation, and each time bin is used for validation exactly once.

TABLE VI: $Precision, Recall$ and $F1\ measure$ of FT-tree, Signature Tree, STE and LogSimilarity

technique	$Precision$	$Recall$	$F1\ measure$
FT-tree	32.27%	95.3%	48.21%
Signature Tree	32.27%	95.3%	48.21%
STE	9.14%	99.6%	16.75%
LogSimilarity	10.67%	83.5%	18.93%

Figure 6 shows the comparison of the precision recall curves (PRCs) of the failure prediction results of FT-tree, Signature Tree, STE, and LogSimilarity. Please note that, in a PRC, to the upper right means a good accuracy, whereas to the left bottom means a bad one. As mentioned earlier, the templates extracted by FT-tree and Signature Tree are almost identical and, thus, they had the same PRCs. Through the PRCs, we can see that applying FT-tree or Signature Tree for learning templates achieved a much better failure prediction accuracy than applying STE and LogSimilarity.

To intuitively compare the best accuracy, we show in Table VI the $Precision, Recall$ and $F1\ measure$ when the prediction system achieved the best $F1\ measure$. Whereas the four techniques achieved an approximate $Recall$, FT-tree and Signature Tree improved the $Precision$ for the failure prediction system by 202% to 253% and the $F1\ measure$ by 155% to 188%.

D. Evaluation on Computational Efficiency

As mentioned earlier, network operators may *frequently* conduct firmware or software upgrades on switches to introduce new features or fix bugs in the previous versions [14], [15]. New *subtypes* of syslog messages can be generated because of the said upgrades and, thus, new templates should be extracted. Otherwise, these new *subtypes* of syslog messages cannot be matched to any template. Since both FT-tree and

TABLE VII: Comparison of template matching time per day for FT-tree, Signature Tree, STE and LogSimilarity

FT-tree	Signature Tree	STE	LogSimilarity
51 min	628 h	100 h	80 min

LogSimilarity learn templates in an incremental manner, when new *subtypes* of syslog messages occur it is not necessary that all the templates have to be retrained and all the historical syslogs have to be matched to new templates. However, if new *subtypes* of syslog messages occur, we have to retrain all the templates and rematch all the historical syslog messages to templates when Signature Tree or STE is applied for template extraction. This is because neither Signature Tree nor STE can learn templates incrementally.

Suppose that the failure prediction or diagnosis technique (such as HSMM) is retrained every day. We here compare the computational cost for FT-tree, Signature Tree, STE, and LogSimilarity. Since it is often the case that new *subtypes* of syslog messages can be generated everyday, all the historical syslogs should be rematched to templates for Signature Tree and STE, but not for FT-tree and LogSimilarity. We implemented FT-tree, Signature Tree, STE and LogSimilarity in C++, and deployed them on the same server (CPU information: 12 Intel(R) Xeon(R) CPU E5645 @ 2.40GHz) with a single thread. The CPU utilization remained 100% while the process of each technique was running so that we could use the total time to evaluate the complexity [25].

Table VII shows the time consumed for matching templates per day for FT-tree, Signature Tree, STE, and LogSimilarity. FT-tree and LogSimilarity can incrementally learn templates and, thus, they just had to match 1-day's worth of syslog messages to the templates. However, STE and Signature Tree had to match all the historical syslogs, *i.e.,* 2-year's worth of syslog messages to the templates. Hence FT-tree and LogSimilarity consume much less time compared to STE and Signature Tree. More specifically, FT-tree improved the computational efficiency by 730 times and 117 times compared to Signature Tree and STE, respectively. Although LogSimilarity is also capable of incremental template extraction and hence is also more computationally efficient than Signature Tree and STE, it generates very different templates and is less computationally efficient than FT-tree.

V. Related Works

Using log files for failure diagnosis and prediction has been widely applied in ISP networks [10], [12], computers [11], [13], [26], [27], and online ad services [28]. Liang *et al.* investigated the RAS event logs and developed three simple failure prediction techniques based on the characteristics of failure events, as well as on the correlation between failure events and non-failure events [26]. Realizing the the importance of the sequential feature of log files to failure prediction, Fronza *et al.* used random indexing (RI) to represent the sequence of operations extracted from logs, and then applied weighted support vector machine to associate sequences to a class of failures or non-failures [27]. Salfner *et al.* applied HSMM to recognize the patterns of logs that indicate an imminent failure directly [12].

Syslog parsing techniques for *network devices including routers and switches* have been well studied in [6]–[8]. Specifically, inspired by the signature abstraction applied in spam detection, Qiu *et al.* proposed a template extraction technique [7]. The idea behind this technique is that a syslog message *subtype* is usually a combination of words with high frequency. Therefore, for syslog messages that belong to a given message type, the technique constructs a signature tree whose root node is the message type and the child nodes are arranged on the basis of the frequency of the *combinations* of words in syslog messages. Using the frequency of the *combinations* of words rather than that of words themselves leads to that Signature Tree is not incrementally retrainable. That is, we have to retrain the signature tree, learn the templates, and match *all* the historical syslog messages when new *subtypes* of syslog messages occur (probably due to upgrades). Considering the large number of syslog messages generated every day (tens of millions) and the long period of historical syslog messages (two years), matching *all* the historical syslog messages to templates does consume a huge amount of computational resources. Therefore, Signature Tree is not suitable for learning templates in our scenario.

In addition, Kimura *et al.* presented an STE approach that extracts log message templates using a statistical clustering algorithm [8]. The high level idea is that template words appear more frequently than parameter words, and that syslog messages that belong to the same *subtype* usually have similar structures with the positions of words. Specifically, for a word w that appears in the x-th position of a syslog message that contains L words, the *word score* for w is $Score(w, x, L) = Probability(w|x, L)$. Then using clustering techniques, STE classifies words with high *word scores* into template words. However, STE can miss some templates and, thus, some *subtypes* of syslog messages cannot be matched to any templates. For example, suppose that the syslog messages that belong to *subtypes* $U_0, U_1, ..., U_n$ have the same number of words. If the syslog messages that belong to U_0 occur much less often than those belonging to $U_1, U_2, ..., U_n$, each of the template words in U_0 will have a relatively small *word score* and, thus, will be classified into parameter words. That is, syslog messages belonging to U_0 cannot be matched into any template. Therefore, as the evaluation experiments show in Sections IV-B and IV-C, STE has a relatively low accuracy. In addition, STE is not incremental either.

To learn templates in an incremental manner, Kimura *et al.* [6] developed an online message template extraction technique, named LogSimilarity. In this technique, they first classified words into five classes on the basis of the tendency to constitute a log template: only symbols, only letters, only symbols and letters, only numbers and letters, and only numbers or numbers and symbols. When a new syslog message arrives, according to the number of words in different classes in the message, the technique will assign this message to an existing template cluster or create a new template cluster from this message. In this technique, message templates are learned on the basis of the *classes of words* rather than of the *words themselves* and, thus, syslog messages that belong to different *subtypes* can be easily assigned to one template cluster. For example, the syslog messages of message type

"SIF" in Table II can be assigned to one or two template clusters in this technique, rather than to four clusters as shown in Table IV.

VI. Conclusion

In this paper, we proposed a novel template extraction technique, *i.e.,* FT-tree, for accurately and incrementally learning templates. We evaluated and compared the performance of FT-tree to those of Signature Tree, STE and LogSimilarity using real-world switch failure tickets and syslogs collected from more than 10 datacenters over a 2-year period. Both FT-tree and Signature Tree achieved much higher accuracy than STE and LogSimilarity, not only on the template learning, but also on the failure prediction. Our experiments also showed that Signature Tree and STE consumed much more computational resources than FT-tree and LogSimilarity, as they are not incrementally retainable. In summary, the evaluation results clearly demonstrate the benefits of FT-tree: highly accurate, low computational cost and incrementally retrainable.

VII. Acknowledgment

The work was supported by National Natural Science Foundation of China (NSFC) under grant No.61402257, No. 61472214 and No. 61472210, US NSF under grant No. CNS-1423182, the State Key Program of National Science of China under grant No. 61233007, the National Key Basic Research Program of China (973 program) under grant No. 2013CB329105, the Global Talent Recruitment (Youth) Program, and the Cross-disciplinary Collaborative Teams Program for Science, Technology and Innovation, of Chinese Academy of Sciences-Network and system technologies for security monitoring and information interaction in smart grid.

References

[1] C. Guo, L. Yuan, D. Xiang, Y. Dang, R. Huang, D. Maltz, Z. Liu, V. Wang, B. Pang, H. Chen, Z.-W. Lin, and V. Kurien, "Pingmesh: A large-scale system for data center network latency measurement and analysis," in *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, ser. SIGCOMM '15, 2015, pp. 139–152.

[2] P. Gill, N. Jain, and N. Nagappan, "Understanding network failures in data centers: Measurement, analysis, and implications," in *SIGCOMM*, 2011.

[3] N. Handigol, B. Heller, V. Jeyakumar, D. Mazières, and N. McKeown, "I know what your packet did last hop: Using packet histories to troubleshoot networks," in *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*, 2014, pp. 71–85.

[4] X. Wu, D. Turner, C.-C. Chen, D. A. Maltz, X. Yang, L. Yuan, and M. Zhang, "Netpilot: automating datacenter network failure mitigation," in *Proceedings of the 2012 ACM Conference on Special Interest Group on Data Communication*, ser. SIGCOMM '12, 2012, pp. 419–430.

[5] Y. Wu, M. Zhao, A. Haeberlen, W. Zhou, and B. T. Loo, "Diagnosing missing events in distributed systems with negative provenance," in *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 4. ACM, 2014, pp. 383–394.

[6] T. Kimura, A. Watanabe, T. Toyono, and K. Ishibashi, "Proactive failure detection learning generation patterns of large-scale network logs," in *Network and Service Management (CNSM), 2015 11th International Conference on*. IEEE, 2015, pp. 8–14.

[7] T. Qiu, Z. Ge, D. Pei, J. Wang, and J. Xu, "What happened in my network: Mining network events from router syslogs," in *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*, ser. IMC '10, 2010, pp. 472–484.

[8] T. Kimura, K. Ishibashi, T. Mori, H. Sawada, T. Toyono, K. Nishimatsu, A. Watanabe, A. Shimoda, and K. Shiomoto, "Spatio-temporal factorization of log data for understanding network events," in *INFOCOM, 2014 Proceedings IEEE*. IEEE, 2014, pp. 610–618.

[9] K. Yamanishi and Y. Maruyama, "Dynamic syslog mining for network failure monitoring," in *Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery in Data Mining*, ser. KDD '05, 2005, pp. 499–508.

[10] F. Salfner and S. Tschirpke, "Error log processing for accurate failure prediction," in *Proceedings of the First USENIX Conference on Analysis of System Logs*, ser. WASL'08, 2008.

[11] Z. Zheng, Z. Lan, B. H. Park, and A. Geist, "System log pre-processing to improve failure prediction," in *Dependable Systems Networks, 2009. DSN '09. IEEE/IFIP International Conference on*, June 2009, pp. 572–577.

[12] F. Salfner and M. Malek, "Using hidden semi-markov models for effective online failure prediction," in *Reliable Distributed Systems, 2007. SRDS 2007. 26th IEEE International Symposium on*. IEEE, 2007, pp. 161–174.

[13] E. W. Fulp, G. A. Fink, and J. N. Haack, "Predicting computer system failures using support vector machines," *WASL*, vol. 8, pp. 5–5, 2008.

[14] A. A. Mahimkar, H. H. Song, Z. Ge, A. Shaikh, J. Wang, J. Yates, Y. Zhang, and J. Emmons, "Detecting the performance impact of upgrades in large operational networks," in *SIGCOMM*, New Delhi, India, August 2010.

[15] A. Mahimkar, Z. Ge, J. Wang, J. Yates, Y. Zhang, J. Emmons, B. Huntley, and M. Stockert, "Rapid detection of maintenance induced changes in service performance," in *CONEXT*, Tokyo, Japan, December 2011.

[16] P. Gill, N. Jain, and N. Nagappan, "Understanding network failures in data centers: Measurement, analysis, and implications," in *Proceedings of the ACM SIGCOMM 2011 Conference*, ser. SIGCOMM '11, 2011, pp. 350–361.

[17] F. Salfner, M. Lenk, and M. Malek, "A survey of online failure prediction methods," *ACM Computing Surveys (CSUR)*, vol. 42, no. 3, p. 10, 2010.

[18] "Transfer switch failure causes outage at colo4 data center," http://www.datacenterdynamics.com/content-tracks/power-cooling/transfer-switch-failure-causes-outage-at-colo4-data-center/32548.fullarticle.

[19] "Switch failure causes outages at hosting.com data center," http://www.datacenterdynamics.com/content-tracks/servers-storage/switch-failure-causes-outages-at-hostingcom-data-center/32344.fullarticle.

[20] Y. Zhang, M. Roughan, W. Willinger, and L. Qiu, "Spatio-temporal compressive sensing and internet traffic matrices," in *SIGCOMM*, Barcelona, Spain, 2009.

[21] J. Han, J. Pei, and Y. Yin, "Mining frequent patterns without candidate generation," in *ACM Sigmod Record*, vol. 29, no. 2. ACM, 2000, pp. 1–12.

[22] W. M. Rand, "Objective criteria for the evaluation of clustering methods," *Journal of the American Statistical association*, vol. 66, no. 336, pp. 846–850, 1971.

[23] G. McLachlan, K.-A. Do, and C. Ambroise, *Analyzing microarray gene expression data*. John Wiley & Sons, 2005, vol. 422.

[24] R. Kohavi, "A study of cross-validation and bootstrap for accuracy estimation and model selection," in *Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 2*, ser. IJCAI'95, 1995, pp. 1137–1143.

[25] M. Yamada, A. Kimura, F. Naya, and H. Sawada, "Change-point detection with feature selection in high-dimensional time-series data," in *Proceedings of the Twenty-Third international joint conference on Artificial Intelligence*. AAAI Press, 2013, pp. 1827–1833.

[26] Y. Liang, Y. Zhang, M. Jette, A. Sivasubramaniam, and R. Sahoo, "Bluegene/l failure analysis and prediction models," in *Dependable Systems and Networks, 2006. DSN 2006. International Conference on*. IEEE, 2006, pp. 425–434.

[27] I. Fronza, A. Sillitti, G. Succi, M. Terho, and J. Vlasenko, "Failure prediction based on log files using random indexing and support vector machines," *Journal of Systems and Software*, vol. 86, no. 1, pp. 2–11, 2013.

[28] M. Shatnawi and M. Hefeeda, "Real-time failure prediction in online services," in *2015 IEEE Conference on Computer Communications (INFOCOM)*. IEEE, 2015, pp. 1391–1399.