

An Integrated Data-Driven Framework for Computing System Management

Tao Li, Wei Peng, Charles Perng, Sheng Ma, and Haixun Wang

Abstract—With advancement in science and technology, computing systems are becoming increasingly more complex with a growing number of heterogeneous software and hardware components. They are thus becoming more difficult to monitor, manage, and maintain. Traditional approaches to system management have been largely based on domain experts through a knowledge acquisition solution that translates domain knowledge into operating rules and policies. This process has been well known as cumbersome, labor intensive, and error prone. In addition, traditional approaches for system management are difficult to keep up with the rapidly changing environments. There is a pressing need for automatic and efficient approaches to monitor and manage complex computing systems. In this paper, we propose an integrated data-driven framework for computing system management by acquiring the needed knowledge automatically from a large amount of historical log data. Specifically, we apply text mining techniques to automatically categorize the log messages into a set of canonical categories, incorporate temporal information to improve categorization performance, develop temporal mining techniques to discover the relationships between different events, and take a novel approach called event summarization to provide a concise interpretation of the temporal patterns.

Index Terms—Event mining, mining log data, summarization, system management.

I. INTRODUCTION

A. Computing System Management

MODERN computing systems are difficult to monitor, manage, and maintain as they are becoming more complex with a growing number of heterogeneous software and hardware components. The increasing complexity also worsens system dependability as the failure rate for a system is much higher than before. It is not a trivial task to provide the high performance, dependability, scalability, and manageability that are demanded by enterprise customers. Traditional approaches to system management mainly rely on domain experts to translate domain knowledge into operational rules, policies, and de-

pendence models via a knowledge acquisition process [1]–[12]. The process is extremely expensive, if not impossible, to keep up with the rapidly changing environments [13], [14]. It has been estimated that, in medium and large companies, anywhere from 30% to 70% of their information technology resources are used as administrative(maintenance) cost [15]. There is thus a pressing need for automatic and efficient approaches to monitor and manage complex computing systems [16]. For example, the IBM Autonomic Computing initiative aims to build autonomic systems that are able to manage themselves with minimum human intervention [17].

To realize the goal of autonomic systems, the underlying assumption is the ability to define and maintain a knowledge base and to adapt it with the ever-changing environment. Modern computing systems generate huge amounts of log/trace data. To enable self-management capabilities, a system needs to automatically characterize its behavior and acquire the needed knowledge from historical log data.

B. Examples of System Log Data

The data in the log files indicate the status of each component and record system operational changes, such as the starting and stopping of services, detection of network applications, software configuration modifications, and software execution errors. To enable self-management capabilities, a system needs to automatically characterize its behavior and acquire the needed knowledge from historical log data. In this section, we describe some examples of the system log data. The data can be collected from the distributed computing components including the network.

- 1) *Application-level log*: Application-level log records the application behaviors as well as the generated messages. Examples include Windows NT system and application log, WebSphere application log, Oracle activity log, and Linux system log.
- 2) *Failure data*: Failure data contain the system and application crash dumps as well as the error messages.
- 3) *Performance data*: Performance data report the performance observations of a component at some time intervals (e.g., CPU utilization of a component every 5 min). Typical performance metrics include CPU utilization, memory utilization, swap utilization, workload average, and response time. In particular, performance data are usually numeric.
- 4) *Reports from operators*: They are also known as trouble ticket data and contain the problems and reports described by human operators. In addition, they may also contain possible causes and symptoms for failures.

Manuscript received July 18, 2008. First published November 6, 2009; current version published December 16, 2009. This work was supported in part by IBM Faculty Research Awards (2005, 2007, and 2008), by IBM Shared University Research Award (2005), and by National Science Foundation CAREER Award IIS-0546280. This paper was recommended by Associate Editor M. Mora.

T. Li is with the School of Computer Science, Florida International University, Miami, FL 33199 USA (e-mail: taoli@cs.fiu.edu).

W. Peng is with Xerox Innovation Group, Xerox Corporation, Rochester, NY 14580 USA (e-mail: wei.peng@xerox.com).

C. Perng, S. Ma, and H. Wang are with IBM T. J. Watson Research Center, Yorktown Heights, NY 10598 USA (e-mail: perng@us.ibm.com; shengma@us.ibm.com; haixun@us.ibm.com).

Color versions of one or more of the figures in this paper are available online at <http://ieeexplore.ieee.org>.

Digital Object Identifier 10.1109/TSMCA.2009.2030161

- 5) *Request data*: Request data reports the request (such as time, machine, user ID, and application) executed in the system. Examples include Apache and Microsoft Internet Information Services logs.
- 6) *Other data*: Other examples of log data include network traffic data, network-based alert logs, program traces, probes, etc.

C. Challenges in Data-Driven System Management

Generally, system management includes root cause analysis (RCA), anomaly and intrusion detection, failure prediction, and diagnosis. We identify several challenges for data-driven computing systems (e.g., loosely coupled network of computer workstations in a department, large-scale clusters, and grid computing environments).

- 1) The heterogeneous nature of the computing system makes the management task complex and complicated. A typical computing system contains different devices (e.g., routers, processors, and adapters) with different software components (e.g., operating systems, middlewares, and user applications), possibly from different providers (e.g., Cisco, IBM, and Microsoft). The heterogeneity increases the likelihood of poorly understood interactions and unexpected interactions/dependences. This challenge has often been discussed in the Metadata Integration studies [18], [19].
- 2) Large and complex computing systems usually show unexpected behavior during failures, system perturbations, and even normal operations. The scale and complexity of these systems greatly surpass what can be understood by human operators. It is difficult for any one person to understand the system at the level of details necessary for management.
- 3) Current computing systems are dynamic and rapidly changing with a growing number of software and hardware components. The fast rate of change worsens system dependability and exacerbates the difficulty of understanding system behaviors.
- 4) Correctly understanding and interpreting patterns discovered from the log data is a big challenge. In system management applications, many log data are generated and collected in the form of temporal events. Data mining approaches for analyzing temporal events generally focus on discovering frequent or interesting patterns in the data—albeit their occurrences may only account for a small fraction of the entire data. However, the extracted temporal patterns tend to be massive, without emphasis, intricate, and uninterpretable. Most of them are meaningless or useless to nonexperts, even analysts. It is of critical importance to enable temporal relationships between events for monitoring and managing complex systems.

D. Content of This Paper

In this paper, we describe our research efforts to address the aforementioned challenges in data-driven system management. In particular, we propose an integrated data-driven framework

for computing system management by acquiring the needed knowledge automatically from a large amount of historical log data, possibly from different types of information sources such as system errors, resource performance metrics, and trouble ticket text records. Specifically, we apply text mining techniques to automatically categorize the log messages into a set of canonical categories, incorporate temporal information to improve categorization performance, develop temporal mining techniques to discover the relationships between different events, and take a novel approach called event summarization to provide a concise interpretation of the temporal patterns [20], [21]. Parts of this paper have been published in [22] and [23].

The rest of this paper is organized as follows: Section II describes the architecture overview of the integrated data-driven framework; Section III extends Naive Bayes classifier for categorizing log messages into a set of common categories by incorporating temporal information; Section IV presents our techniques for discovering the relationships between different events; Section V proposes event summarization toward the understanding of the seemingly chaotic temporal data; Section VI shows a case study on mining log data for system management; and finally Section VII provides conclusions and discussions.

II. ARCHITECTURE OF THE DATA-DRIVEN FRAMEWORK

The architecture of the proposed data-driven framework is shown in Fig. 1. The key components of the framework are listed as follows.

- 1) *Log Data Categorization*: Network and host sensors, along with instrumented applications provide the ability to collect log data in the computing system. *Log Adapter* enables generic data collection from multiple heterogeneous data sources by converting individual records and events into the common base event (CBE) format and *Log Data Categorization* standardizes the semantics of heterogeneous log data.
- 2) *Mining Temporal Dependences*: After transforming the log messages into common categories, it is then possible to analyze the historical event data across multiple components to discover interesting patterns embedded in the data. Each message in log files usually has a timestamp, and we try to mine the temporal dependence relationships.
- 3) *Event Summarization*: Event summarization aims at providing a concise interpretation of the seemingly chaotic data, so that domain experts may take actions upon the summarized models. Event summarization decomposes the temporal information into many independent subsets and finds well-fitted models to describe each subset.

Note that planning/action includes administrator notification, active data collection, sensor/actuator deployment or deactivation, etc. Rather than removing the human from the loop, the proposed data-driven integrated framework exploits the synergy of human expertise and automatic intelligent techniques and establishes the pioneering and practical solutions for system management.

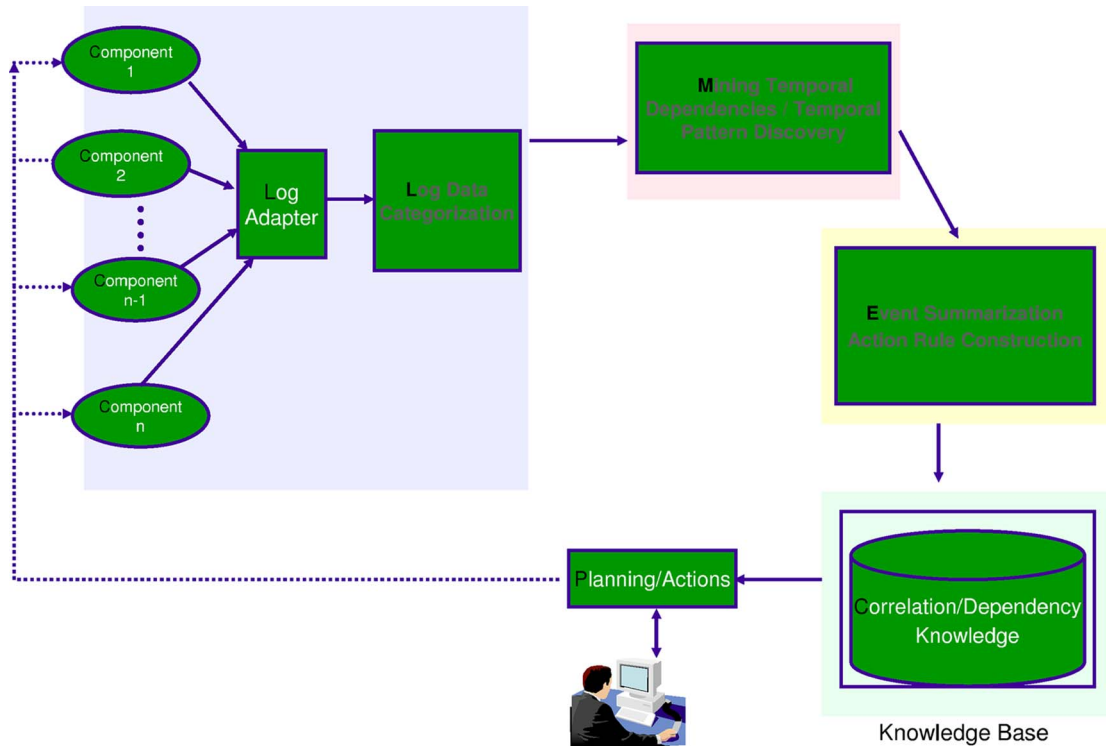


Fig. 1. Architecture of the integrated data-driven framework.

III. LOG DATA CATEGORIZATION

A. Common Categories

The disparate logging mechanisms impede problem investigation because there is no standard approach for classifying them [24]. Two components experiencing the same problem may use different formats to report the same information. For instance, when component A starts running, it reports “A has started” in log file. However, when component B starts running, it may report “B has begun execution.” Although both messages have the same content meaning, they are reported differently. This makes it difficult to correlate logs from across different components when problems occur.

In order to create consistency across similar fields and improves the ability to correlate across multiple logs, it is necessary to transform the messages in the log files into a common base [25] (i.e., a set of common categories). In this paper, we first manually determine a set of categories as the basis for transformation. The set of categories is based on the CBE format established by IBM initiative [24]. CBE provides a finite set of canonical situations after analyzing thousands of log entries across multiple IBM and non-IBM products. Although we use CBE as an internal presentation here, the problem and the proposed approach are generic and can be extended for other data formats as well.

Specifically, the set of categories includes *start*, *stop*, *dependence*, *create*, *connection*, *report*, *request*, *configuration*, and *other*. The category *start* deals with the start-up process of components. The category *stop* deals with the exit process. The category *dependence* handles messages which report components cannot find some features or other components. The category *create* handles the creation problems of components. The category *connection* is used to identify aspects about a

connection to another component. The category *report* deals with performance, task-relevant reporting messages. The category *request* identifies the completion status of a request. The category *other* identifies the blank messages or messages having vague meanings.

B. Log Message Categorization

Given the set of common categories, we can then categorize the messages reported by different components into the prescribed categories. This can be viewed as a text categorization problem where the goal is to assign predefined category labels to unlabeled documents based on the likelihood inferred from the training set of labeled documents [26], [27].

In this paper, we use naive Bayes as our classification approach since it is among the most successful algorithms for learning in text categorization [28]. System log files usually have a large vocabulary and most of the system log messages contain a free-format 1024-B ASCII description of the event [29]. Naive Bayes is a simple practical generative classification algorithm based on Bayes Theorem with the assumption of class conditional independence. Basically, it assumes that the text data is generated by a parametric model, and uses training data to calculate Bayes-optimal estimates of the model parameters [30]. Then, it classifies test data using the generative model by calculating the posterior probability that a class would have generated the test data in question. The most probable class is then assigned to the test data [31].

1) *Naive Bayes Classifier:* Formally, suppose there are L categories, denoted by C_1, C_2, \dots, C_L , and each category is generated from some probability distribution. Each document d_i is created by first selecting a category according to the prior $P(C_j)$ followed by the generation according to the distribution

of the chosen category $P(d_i|C_j)$. We can characterize the likelihood of a document with the sum of probability over all the categories

$$P(d_i) = \sum_{j=1}^L P(C_j)P(d_i|C_j).$$

Given a set of training samples S , the naive Bayes classifier uses S to estimate $P(d_i|C_j)$ and $P(C_j)$. The estimated probabilities are typically calculated in a straightforward way from training data. To classify a new sample, it uses Bayes rule to compute class posterior

$$P(C_j|d_i) = \frac{P(C_j)P(d_i|C_j)}{\sum_{j=1}^L P(C_j)P(d_i|C_j)}.$$

The predicted class for the document d_i is then just $\arg \max_j P(C_j|d_i)$. In this paper, we use multinomial probability model for document generation of each category. More details on Naive Bayes can be found in [31].

C. Incorporating the Temporal Information

In many scenarios, log messages generated in the log files usually contain timestamps. The temporal characteristics provide additional context information of the messages and can be used to facilitate data analysis. As an example, knowing the current status of a network component would help forecast the possible types of messages it will generate. These structural constraints can be naturally represented using naive Bayes algorithm. If a sequence of log messages are considered, the accuracy of categorization for each message can be improved as the structure relationships among the messages can be used for analysis. For example, the components usually first start-up a process, then stop the process at a later time, i.e., the *stop* message usually occurs after the *start* message.

In order to take the relationships between adjacent messages into account, we make some modifications to the naive Bayes algorithm. Suppose we are given a sequence of adjacent messages $D = (d_1, d_2, \dots, d_T)$. Let Q_i be the category labels for message d_i (i.e., Q_i is one of the C_1, C_2, \dots, C_L). Now, we want to classify d_{i+1} . Note that

$$\begin{aligned} P(Q_{i+1}|d_{i+1}, Q_i) &= \frac{P(Q_{i+1}, d_{i+1}|Q_i)}{P(d_{i+1}|Q_i)} \\ &= \frac{P(Q_{i+1}|Q_i)P(d_{i+1}|Q_{i+1}, Q_i)}{P(d_{i+1}|Q_i)}. \end{aligned}$$

Assuming that d_{i+1} is conditionally independent of Q_i given Q_{i+1} , $P(d_{i+1}|Q_{i+1}, Q_i) = P(d_{i+1}|Q_{i+1})$. In addition, once Q_i is determined, $P(d_{i+1}|Q_i)$ is then fixed. Hence, maximizing $P(Q_{i+1}|d_{i+1}, Q_i)$ is equivalent to maximizing $P(Q_{i+1}|Q_i)P(d_{i+1}|Q_{i+1})$. Observe that $P(d_{i+1}|Q_{i+1}) = (P(Q_{i+1}|d_{i+1})P(d_{i+1})/P(Q_{i+1}))$. If we assume a uniform prior on $P(Q_{i+1})$, in order to maximize $P(d_{i+1}|Q_{i+1})$, it is enough to maximize $P(Q_{i+1}|d_{i+1})$. Therefore, in summary, we assign d_{i+1} to the category Q_{i+1} which is

$$\arg \max_j (P(C_j|d_{i+1}) \times P(C_j|Q_i)).$$

In other words, we aim at maximizing the multiplication of text classification probability $P(C_j|d_{i+1})$ and state transition probability $P(C_j|Q_i)$. The transition probability $P(C_j|Q_i)$ can be estimated from training log files.

IV. MINING TEMPORAL DEPENDENCES

A. Introduction

Once the messages in the log file are transformed into common categories, it is then possible to analyze the historical event data across multiple components to discover interesting patterns embedded in the data. Each message in log files usually has a timestamp, and we will try to find the temporal patterns. Temporal patterns of interest appear naturally in the system management applications [32]. Specifically, a computer system problem may trigger a series of symptom events/activities. Such a sequence of symptom events provides a natural signature for identifying the root cause. For example, when component A reports "A is stopped," how should we respond to this *stop* message? Is this a failure in the component A or just a natural exit after successful completion of tasks? In fact, the event "A is stopped" does not necessarily mean that an error or a failure happened. We noticed that in our log data files, the *stop* message often occurs after the component reports *dependence* message (i.e., it cannot find certain features or components). In this case, the *stop* message signifies the failure of a task execution. However, if there is no *dependence* message before the *stop* message, the *stop* message usually indicates the completion of a task. Thus, to decide the proper actions in response to the *stop* messages, we need to check what happens right before them. As summarized in [32] and [33]: A problem manifests itself as a sequence of events propagating from origin and low layer to high software layer through the *dependence* tree. Thus, knowing the temporal patterns can help us pinpoint the root cause and take proper actions [34].

B. Events and Patterns

An event has two components: the event type and the time stamp (occurring time or arrival time). Throughout this paper, we assume a finite set E of event types. We use the symbol e , possibly with subscripts to denote event types. An *event* is then a pair (e, t) where $e \in E$ is an event type and t is the timestamp of e . Our data, called an *event sequence*, are then a collection of all events occurred within some time range (for example, between zero and T) $D = \{(e_1, t_1), (e_2, t_2), \dots, (e_n, t_n)\}$, where $e_i \in E$, $t_i \in [0, T]$, and $t_i \leq t_{i+1}$.

In general, the dependence relationships are essential factors/patterns to be discovered: They assert that an event usually happens after another event. Such patterns of interest appear naturally in the system management applications [35]. In this paper, we start with pairwise temporal dependences.

C. Temporal Dependence

1) *Overview*: Let T_a and T_b be the sequence of timestamps for event a and b , respectively. Let the distance from a point z (or, a timestamp), to an event sequence T_a to be $d(z, T_a) = \min_{x \in T_a, x \geq z} (x - z)$, i.e., the distance from the point z to its nearest neighbor in T_a which occurs after z . In order to

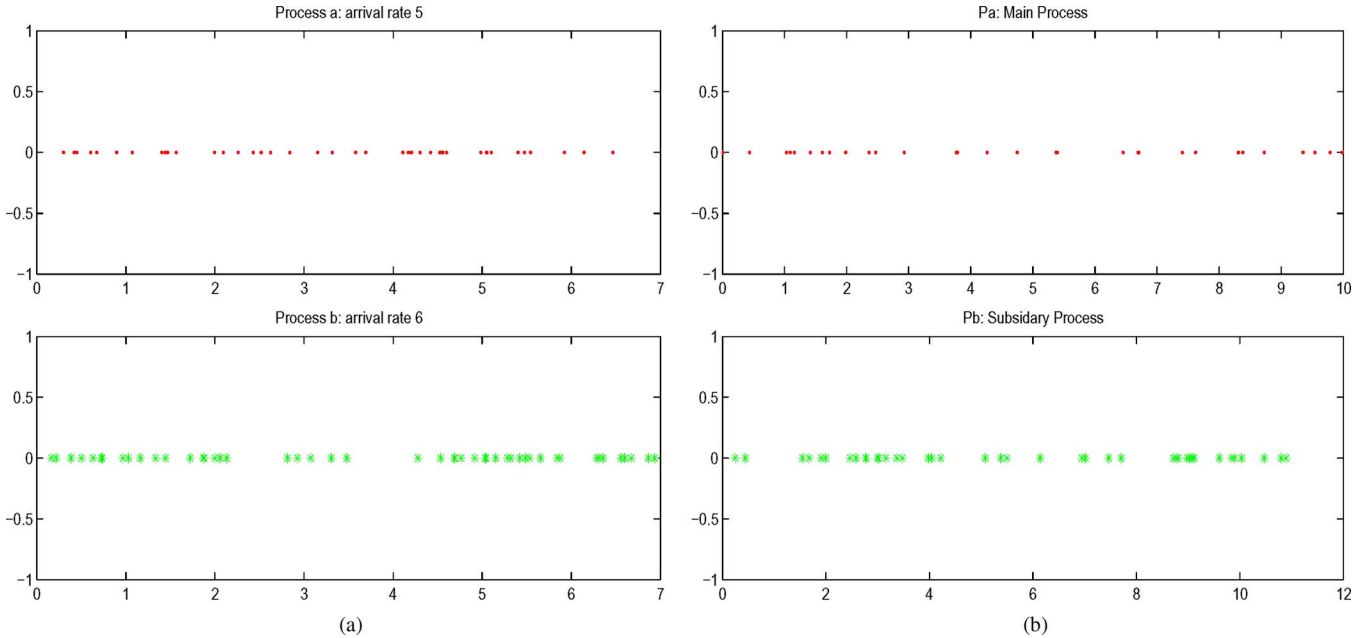


Fig. 2. Arrivals of two events. (a) Arrivals of two independent events. (b) Arrivals of two dependent events.

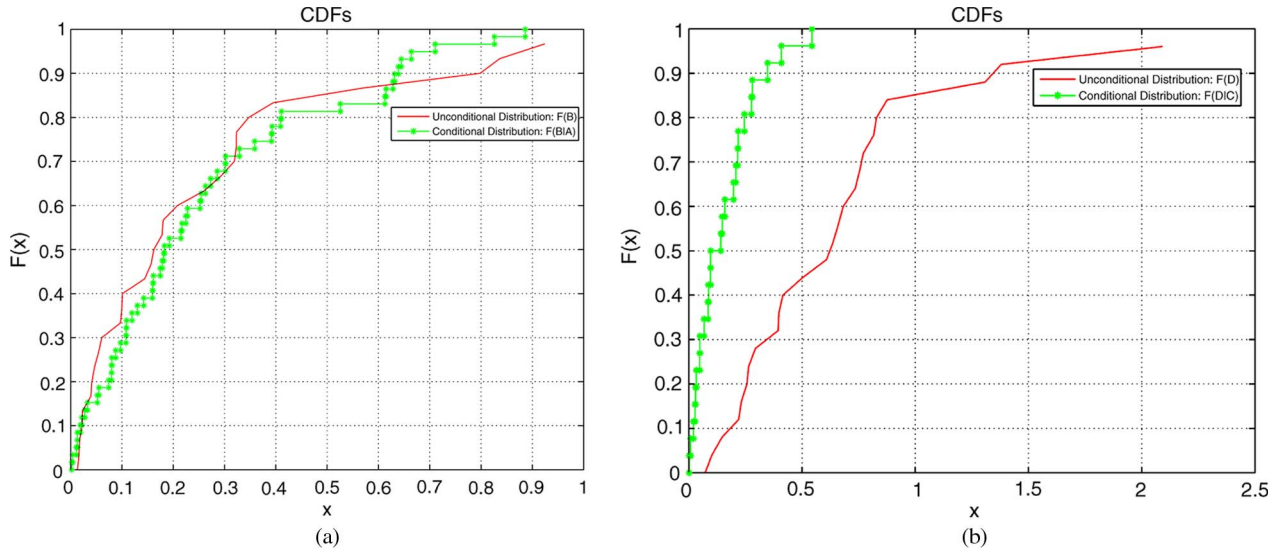


Fig. 3. CDFs. (a) CDFs of independent events. (b) CDFs of two dependent events. (Solid line) Unconditional distribution: $F(B)$. (Dash-dotted line) Conditional distribution: $F(B|A)$.

determine whether b is dependent on a , we compare the typical distance of random points z to T_b to the typical distance of points $z \in T_a$ to T_b . This technique is motivated by the following idea: *if the occurrence of b is predictable by the occurrence of a , then the conditional distribution which models the waiting time of event type b given event type a 's presence would be different from the unconditional one* [22]. In particular, we compare the following two interarrival distributions: 1) the unconditional distribution of the waiting time of event b : $F_b(r) = \mathcal{P}\{d(x, T_b) \leq r\}$ and 2) the conditional distribution for the waiting time of event b with respect to event a : $F_b^a(r) = \mathcal{P}(d(x, T_b) \leq r | x \in T_a)$. The conditional distribution can be regarded as the conditional probability distribution given there is an event of a at time x . If the two distributions $F_b(r)$ and $F_b^a(r)$ are different, then we say that b is directly dependent on a . This technique is extended from our earlier work [36] and

can be used to discover temporal dependences without using predefined time windows.

After temporal dependences have been discovered, the time intervals between the two events can be discovered by applying existing lag detection algorithms (e.g., peak finding [37], clustering, and density estimation [37]) on the collection of time distances between them. These time intervals can be easily incorporated into the event relationship network (ERN) described in Section V-B.

2) *Discovering Temporal Dependences*: To illustrate the procedure for discovering pairwise temporal dependences, Fig. 2(a) shows the arrivals for two sample independent events A and B in our data set, and Fig. 2(b) shows the arrivals for two dependent events C and D. Fig. 3(a) shows the cumulative distribution functions (CDFs) of unconditional F_B and conditional F_B^A , and Fig. 3(b) shows the two CDFs for C and D.

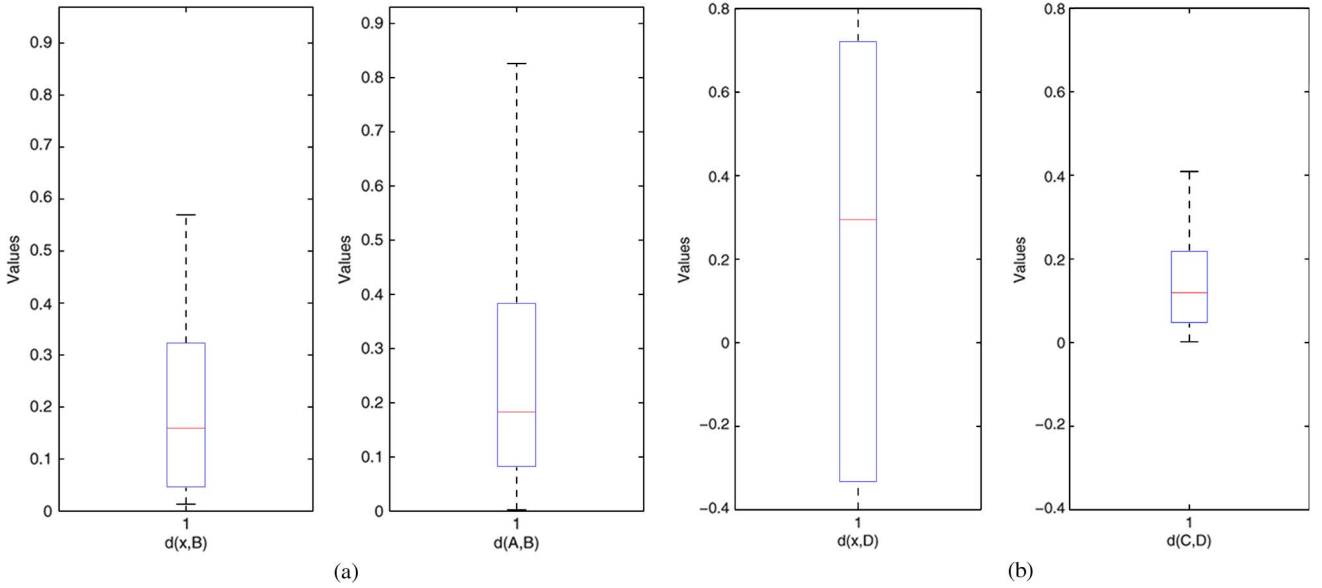


Fig. 4. Boxplots. (a) Boxplots of $d(x, B)$ and $d(A, B)$ of Independent Events A and B. (b) Boxplots of $d(x, D)$ and $d(C, D)$ of dependent Events C and D.

The difference between the two CDFs clearly indicates the dependences between these two events. In order to compare the differences between the independent relationship and dependent relationship, Fig. 4(a) shows the boxplots of $d(x, B)$ where x is a random event and $d(A, B)$. Fig. 4(b) shows the boxplots of $d(x, D)$ and $d(C, D)$. The lines in the approximate middle of boxes are medians. By using a robust order statistics method which compares the ranges of confidence intervals, we can also conclude that event B is independent of event A while event D is dependent of event C.

D. Ranking Dependences

In order to summarize temporal data sets, we use *Entropy* to rank the event dependence relationship. Entropy indicates the amount of information contained in the event patterns and measures the uncertainty of conditional probability distributions on a particular event. We use $FCP(e_j|e_i)$ to denote the forward conditional probability of event e_i followed by e_j relative to all event pairs starting from e_i . $BCP(e_i|e_j)$ is used to denote the backward conditional probability of e_j preceded by e_i relative to all event pairs ending at e_j .

The forward entropy of an event Q can be defined as follows [38]:

$$FE(Q) = - \sum_{e \in E} FCP(e|Q) \times \log_N (FCP(e|Q)) \quad (1)$$

where E is the set of all event types, N is the number of events in E , and $\sum_{e \in E} FCP(e|Q) = 1$. For an instance, if event A is always only followed by B , the forward conditional probability of AB , $FCP(B|A) = 1.0$, and the forward conditional probabilities of the sequences that A is followed by the other events are 0.0. The higher the entropy, the less deterministic the conditional probabilities are distributed. This means that if an event has a lower forward entropy, it has more confidence

or liability to select certain events to proceed. Similarly, the backward entropy can be calculated as follows:

$$BE(Q) = - \sum_{e \in E} BCP(e|Q) \times \log_N (BCP(e|Q)) \quad (2)$$

where $\sum_{e \in E} BCP(e|Q) = 1$. If an event, or an event sequence has a lower backward entropy, it has more deterministic precedents. In summary, entropy aims to capture the temporal dependence relationships among events.

E. Concurrent Behaviors

Concurrency indicates that multiple events are reported or performed in parallel rather than sequentially. Suppose an event e_1 has a forward conditional probability distribution of $(0.9, 0.05, 0.05, 0.0)$ on four events; it is obvious this event has a more deterministic successor. Suppose another event e_2 has a distribution of $(0.45, 0.55, 0.0, 0.0)$. Obviously, e_2 has a two-peak distribution, i.e., there might be concurrent behaviors after e_2 . Hence, to detect concurrent behaviors, for each event, we compare its conditional probability distribution with the prototype distributions $Peak_1 = (1, 0, 0, \dots, 0)$, $Peak_2 = (0.5, 0.5, 0, \dots, 0)$, \dots , $Peak_N = (1/N, 1/N, \dots, 1/N)$, where N is the number of event types, allowing all possible permutations. For example, $(1, 0, 0, \dots, 0)$ is equivalent to $(0, 1, 0, \dots, 0)$. In practice, we first sort the row such that the components decrease from the left to the right, and then assign it to the closest prototype.

V. EVENT SUMMARIZATION

In this section, we discuss how to summarize an event set to an easy-to-understand plot based on the pairwise event dependent relationships. We also discuss some typical ways to reduce summarized patterns into action rules.

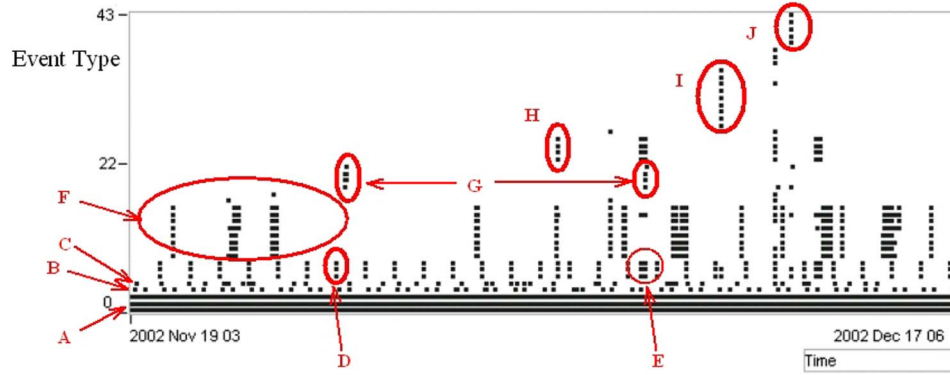


Fig. 5. System management events.

A. Motivating Example

To illustrate the problem, we present a real-world example. Fig. 5 shows a set of system management events taken from a business-to-business gateway facility. After preprocessing, each event contains two attributes: timestamp and event type which are mapped to *X*- and *Y*-axis. By visually analyzing the events, it is not difficult to conclude the following.

- 1) The three event types, labeled by **A** with the lowest event IDs are densely populated events.
- 2) The event type labeled by **B** forms another biperiodic pattern with period 8 and 16 h.
- 3) The event type labeled by **C** appears to be randomly distributed with no apparent period.
- 4) The three event types indicated by **D** almost always occur at the same time. There is a dominating periodic pattern that governs the event group. However, there are some occurrences such as those indicated by **E** that appear randomly.
- 5) The eight events labeled by **F** tend to appear together in a relatively random pace. Sometimes there are another two event types on top that appear along with the group.
- 6) The event groups labeled by **G** and **H** appear a few times randomly.
- 7) The event groups labeled by **I** and **J** only occur once.

The aforementioned observations are usually useful for decision making. For system management experts, the aforementioned summary provides a very good way to itemize patterns and to devise action plans. For example, the **A** section has strong biperiodicity, so it would make better sense to monitor any violation of such biperiodicity instead of the normal occurrences; moreover, seven of the eight event types in **F** section can be turned off without losing any information.

B. ERN

ERNs [39], [40] is a graphical representation of temporal relationship among events. ERN is widely used in IT system management and business services management area as a product-agnostic format for bridging the knowledge discovery process with the system implementation process. Event summarization discussed in this paper is aiming to render output in a form of ERN that is richer and more expressive than what the

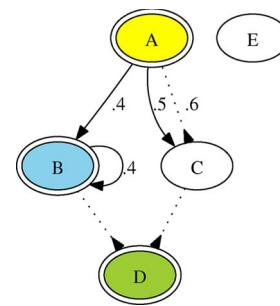


Fig. 6. ERN.

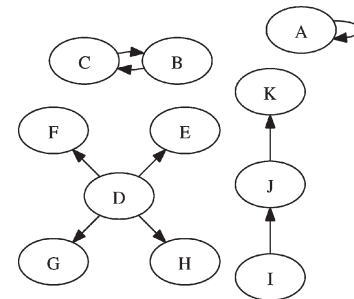


Fig. 7. Usual event patterns.

industry is using. Here, we will explain both the representation and the construction process through an example.

Aside from computing the statistics discussed in the previous section, an additional required input is a threshold tuple for distinguishing whether a particular statistics is significant enough. Threshold tuples are in the form of $\langle \Theta_{FE}, \Theta_{BE}, \Theta_{FCP}, \Theta_{BCP} \rangle$. Note that Θ_{FE} and Θ_{BE} are upper bounds while Θ_{FCP} and Θ_{BCP} are lower bounds. The intended output, ERN, use nodes to denote events with $BE \leq \Theta_{FE}$ or $FE \leq \Theta_{BE}$. These are the events with either more certain leading events or more certain following events. ERNs also need to show significant event relationship as links. The ERN construction process includes the following steps.

- 1) Plot all the events with $FE \leq \Theta_{FE}$ or $BE \leq \Theta_{BE}$ as nodes in color yellow with two peripheries, such as event *A* in Fig. 6.
- 2) Plot all the events with $FCP \geq \Theta_{FCP}$ or $BCP \geq \Theta_{BCP}$ as nodes in color blue and green, respectively, with two peripheries, such as event *B* and *D* in Fig. 6.

TABLE I
EXAMPLE FRACTION OF A LOG FILE WITH THE MANUAL LABEL OF EACH MESSAGE

Time	EventSource	msg	State
1093359583	UPHClean	The following handles in user profile hive FIU-SCS have been closed because they were preventing the profile from unloading successfully wmiprivse.exe (2432)HKCU (0x228)	report
1093359583	UPHClean	User profile hive cleanup service version 1.5.5.21 started successfully.	start
1093359603	AutoEnrollment	Automatic certificate enrollment for local system failed to contact the active directory (0x8007054b). The specified domain either does not exist or could not be contacted.Enrollment will not be performed.	dependency
1093359605	Inventory Scanner	Unable to open output file C:\invdelta.tmp.	dependency
1093359606	Inventory Scanner	LDISCN32: Can't create temporary file.	create
1093359641	Inventory Scanner	Unable to open output file C:\invdelta.tmp.	dependency
1093359918	Inventory Scanner	LDISCN32: Can't create temporary file.	create
1093359583	Userenv		other
1093359584	MsiInstaller	Product: WebFldrs XP – Configuration completed successfully.	dependency

- 3) For each event pair X and Y with $FCP(Y|X) \geq \Theta_{FCP}$, plot a forward link as the one from A to B .
- 4) For each event pair X and Y with $BCP(Y|X) \geq \Theta_{BCP}$, plot a backward link as the one from A to C . The result is an ERN that looks like Fig. 6.

In Fig. 7, we show a few common event patterns.

- 1) *Periodic patterns*: An event with self-loop like event A with near-constant interval is a periodic event pattern. Typical examples are events resulting from periodic probing and system heartbeat monitoring.
- 2) *Circular patterns*: A set of events forming a loop is a circular pattern, such as B, C in Fig. 7.
- 3) *Simultaneous patterns*: A set of events always occur together, such as F in Fig. 5, is a simultaneous pattern. If the intervals among D, E, F, G , and H in Fig. 7 are close to zero, then they form a simultaneous pattern.
- 4) *Event chain*: Event I, J , and K form an event chain. Such patterns are frequently observed from events that represent different stages of a problem and the problem progresses in a predictable sequence.

C. Derive Action Rules From Event Summary

As mentioned, the ERN is a product-agnostic and vendor-neutral representation of event dynamics. Consider action rules in the form of

if **condition** is true, take **action**

ERNs contain the information required by the **condition** part of the rule. The **action** part is usually invocation of self-healing scripts or notification for system administrators. Some popular

types of rules can be derived from ERNs almost directly. Here, are some examples.

- 1) *Event reduction rules*: A self-loop with a nearly constant arrival rate means an event keeps occurring in a constant pace. In system management, such a periodic event pattern is usually a series of probing reports. For those reporting normal, a typical action rule is to invert it—only report the missing of it. For those reporting abnormal, each consecutive “episode” is converted to a start and an end of an anomaly, those in between can be suppressed.
- 2) *Event correlation rules*: An event is usually a symptom of a problem. Each problem may have many symptoms. For system administrators to correctly take remedial actions, the monitoring system provides RCA capabilities. One way to achieve RCA is to use ERNs. An ERN built from a system management event set is usually a set of disconnected subgraphs. Each subgraph contains a set of correlated events and hence represents a problem.
- 3) *Problem avoidance rules*: Some events indicate a problem as reached a severe stage and immediate actions should be taken to prevent them from occurring. ERNs provides a natural way to predict these severe states.

VI. CASE STUDY FOR SYSTEM MANAGEMENT

A. System Log Data Description

We collect log files from several Windows machines in the Graduate Laboratory of the Computer Science School of Florida International University. An example fraction of the system log files extracted from our data collection is shown in Table I. In the example, four columns, i.e., *Time*, *EventSource*,

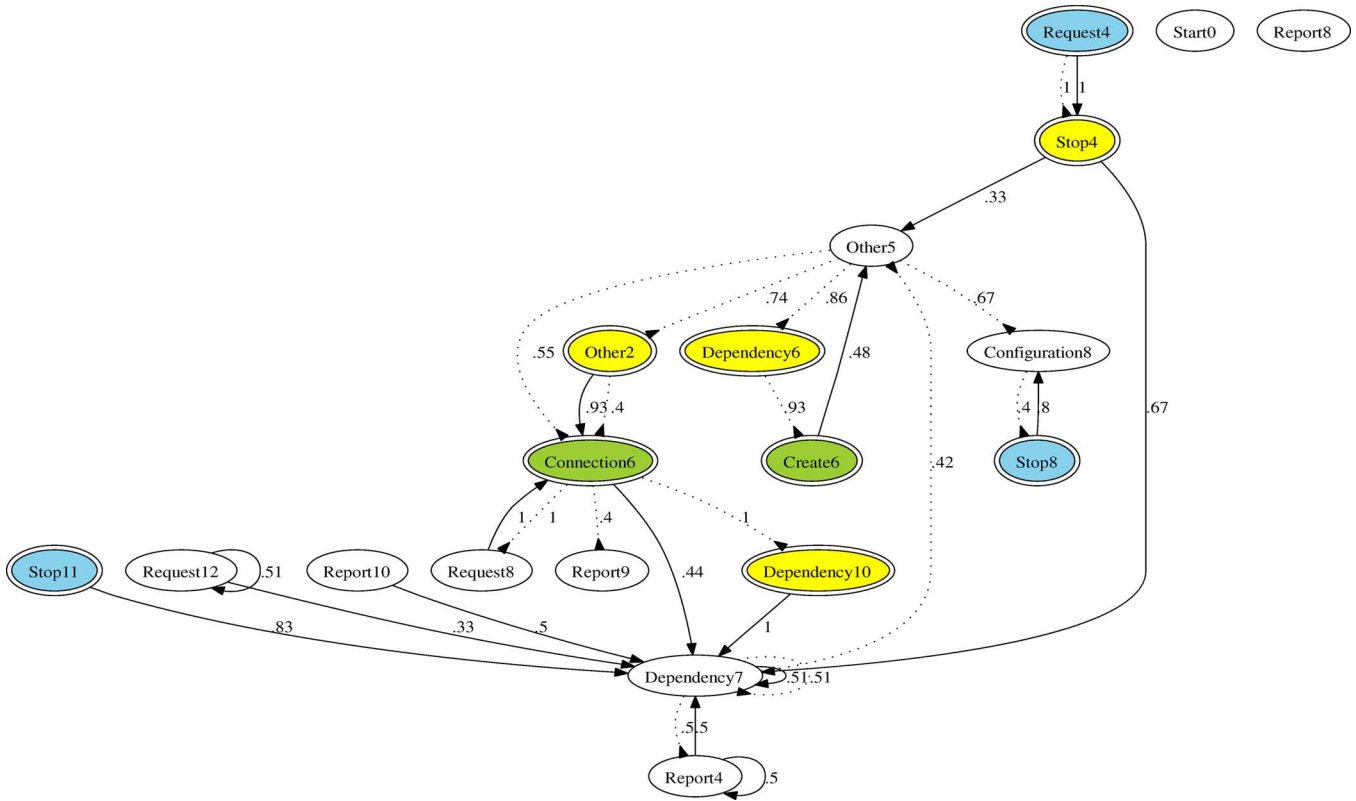


Fig. 8. ERN model summarizing the extracted patterns.

Msg, and *State* are listed.¹ *Time* represents the generated timestamp of log messages, *EventSource* identifies the component which reports the message, *Msg* lists the content of text message, and *State* labels the message category obtained using our modified Naive Bayes classifier.

B. Event Summarization

After log data categorization, each message is represented as a three-tuple $\langle s, c, t \rangle$, where s is the state type, c is the source component of this log message, and t is the timestamp. In order to discover event relationships among components, we map each distinct pair of $\langle s, c \rangle$ into a unique event type e . This representation enable us to analyze the historical event data across multiple components and to discover interesting patterns embedded in the data. In our data set, we have nine different states including *start*, *stop*, *dependence*, *create*, *connection*, *report*, *request*, *configuration*, and *other*. We have 13 different event sources, denoted by $0, 1, \dots, 12$.

In the case study, after the mapping, our data set contains the following event types: *start0*, *dependence7*, *other5*, *other2*, *dependence6*, *create6*, *report9*, *configuration8*, *connection6*, *report0*, *report8*, *stop8*, *request4*, *stop4*, *request8*, *dependence10*, *stop11*, *request12*, and *report4*, where the prefix words in indicate the event states and the last number is the component ID (i.e., event sources) that reports the events. The ERN summarization model constructed from the extracted temporal patterns is shown in Fig. 8.

¹State represents the event type.

The event summarization approach is obviously useful for system management experts to make decisions. First, the dependence patterns are extracted based on the rank. The divide-and-conquer process provides a systematic way to decompose the data set. Second, the ERN model provides a good guideline for identifying patterns and to devise action plans. For example, the temporal pattern event 5 (i.e., dependence6: event source 6 in dependence situation) followed by event 6 (i.e., create6: event source 6 in create situation) implies that the creation requests in component 6 are generally resulted by the lack of resources. Therefore, to reduce the creation requests, we need to allocate more resources to the component. As an another example, the temporal pattern event 12 (i.e., stop8: event source 8 in stop situation) followed by event 8 (i.e., configuration8: event source 8 in configuration situation) implies that the stop process in component 8 often leads to the configuration procedure. Therefore, to speed up the exit process of component 8, we need to either eliminate the configuration procedure when exiting or improve the configuration procedure.

VII. CONCLUSION

In this paper, we present our research efforts of establishing an integrated data-driven framework on mining log files for computing system management by exploring the synergy of text mining, temporal data mining, and event summarization. Event summarization aims at providing a concise interpretation of the seemingly chaotic data, so that domain experts may take actions upon the summarized models. A divide-and-conquer process is employed to extract temporal patterns based on entropy ranking, and an ERN is constructed to provide concise

representations. We present a case study using the real-world system log data set.

There are several directions for future research. First, for log message categorization, instead of manually determining the set of common categories, we could develop techniques to automatically infer them from historical data. For instance, we could use clustering techniques to identify the structures from large amount of log data. Second, in practice, the number of different common categories for system management can be significantly large due to the complex nature of the system. To resolve this issue, it is useful to utilize the dependence relationships among different categories [41]. Third, another natural direction is to use levelwise approach to extend the pairwise temporal dependent patterns to the patterns of length greater than two. Fourth, it is also necessary to develop efficient techniques for generating actionable rules from the ERNs.

REFERENCES

- [1] A. Bouloutas, S. Calo, and A. Finkel, "Alarm correlation and fault identification in communication networks," *IEEE Trans. Commun.*, vol. 42, no. 2–4, pp. 523–533, Feb.–Apr. 1994.
- [2] A. Brown, G. Kar, and A. Keller, "An active approach to characterizing dynamic dependencies for problem determination," in *Proc. 7th IFIP/IEEE Int. Symp. Integr. Netw. Manage.*, 2001, pp. 377–390.
- [3] F. Feather, D. Siewiorek, and R. Maxion, "Fault detection in an Ethernet network using anomaly signature matching," in *Proc. Conf. Commun. Archit. Protocols, Appl. SIGCOMM*, 1993, pp. 279–288.
- [4] B. Gruschke, "A new approach for event correlation based on dependency graphs," in *Proc. 5th Workshop OVUA*, 1998.
- [5] C. S. Hood and C. Ji, "Proactive network fault detection," in *Proc. 16th Annu. Joint Conf. IEEE Comput. Commun. Soc. INFOCOM*, 1997, pp. 1147–1155.
- [6] S. Kliger, S. Yemini, Y. Yemini, D. Ohsie, and S. Stolfo, "A coding approach to event correlation," in *Proc. Intell. Netw. Manage. (IM)*, 1997, pp. 266–277.
- [7] L. Lewis, "A case-based reasoning approach to the resolution of faults in communications networks," in *Proc. Intell. Netw. Manage. (IM)*, 1993, pp. 671–682.
- [8] W. Pedrycz, A. V. Vasilakos, and S. Karnouskos, "Guest editorial special issue on computational intelligence in telecommunications networks and Internet services—Part III," *IEEE Trans. Syst., Man, Cybern. C, Appl. Rev.*, vol. 34, no. 1, pp. 1–3, Feb. 2004.
- [9] I. Rouvellou and G. W. Hart, "Automatic alarm correlation for fault identification," in *Proc. 14th Annu. Joint Conf. IEEE Comput. Commun. Soc. INFOCOM*, Washington, DC, 1995, vol. 2, pp. 553–561.
- [10] S. A. Yemini, S. Kliger, E. Mozes, Y. Yemini, and D. Ohsie, "High speed and robust event correlation," *IEEE Commun. Mag.*, vol. 34, no. 5, pp. 82–90, May 1996.
- [11] H. Arora, B. K. Mishra, and T. S. Raghu, "Autonomic-computing approach to secure knowledge management: A game-theoretic analysis," *IEEE Trans. Syst., Man, Cybern. A, Syst., Humans*, vol. 36, no. 3, pp. 487–497, May 2006.
- [12] Q. Yang, Y. Wang, and Z. Zhang, "SANeT: A service-agent network for call-center scheduling," *IEEE Trans. Syst., Man, Cybern. A, Syst., Humans*, vol. 33, no. 3, pp. 396–406, May 2003.
- [13] J. M. Smith and S. Nettles, "Active networking: One view of the past, present, and future," *IEEE Trans. Syst., Man, Cybern. C, Appl. Rev.*, vol. 34, no. 1, pp. 4–18, Feb. 2004.
- [14] M. K. Agarwal, M. Gupta, V. Mann, N. Sachindran, N. Anerousis, and L. Mummert, "Problem determination in enterprise middleware systems using change point correlation of time series data," in *Proc. IEEE NOMS*, 2006, pp. 471–482.
- [15] IBM Market Research, *Autonomic Computing Core Technology Study*, 2003.
- [16] P. Horn, *Autonomic Computing: IBM's Perspective on the State of Information Technology*. New York: IBM Corp., 2001. [Online]. Available: <http://www.research.ibm.com/autonomic>
- [17] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 41–50, Jan. 2003.
- [18] P. A. Bernstein and L. M. Haas, "Information integration in the enterprise," *Commun. ACM*, vol. 51, no. 9, pp. 72–79, Sep. 2008.
- [19] A. Y. Halevy, N. Ashish, D. Bitton, M. Carey, D. Draper, J. Pollock, A. Rosenthal, and V. Sikka, "Enterprise information integration: Successes, challenges and controversies," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2005, pp. 778–787.
- [20] X. Yan, H. Cheng, J. Han, and D. Xin, "Summarizing itemset patterns: A profile-based approach," in *Proc. 11th ACM SIGKDD Int. Conf. KDD*, 2005, pp. 314–323.
- [21] C. Wang and S. Parthasarathy, "Summarizing itemset patterns using probabilistic models," in *Proc. 12th ACM SIGKDD Int. Conf. KDD*, 2006, pp. 730–735.
- [22] T. Li, F. Liang, S. Ma, and W. Peng, "An integrated framework on mining logs files for computing system management," in *Proc. 11th ACM SIGKDD Int. Conf. KDD*, 2005, pp. 776–781.
- [23] W. Peng, C. Perng, T. Li, and H. Wang, "Event summarization for system management," in *Proc. 13th ACM SIGKDD Int. Conf. KDD*, 2007, pp. 1028–1032.
- [24] B. Topol, D. Ogle, D. Pierson, J. Thoensen, J. Sweitzer, M. Chow, M. A. Hoffmann, P. Durham, R. Telford, S. Sheth, and T. Studwell, "Automating problem determination: A first step toward self-healing computing systems," *IBM White Paper*, Oct. 2003. [Online]. Available: <http://www-106.ibm.com/developerworks/autonomic/library/ac-summary/ac-prob.html>
- [25] M. Chessell, *Specification: Common Base Event*, 2003. [Online]. Available: <http://www-106.ibm.com/developerworks/webservices/library/ws-cbe/>
- [26] T. Joachims, "Text categorization with support vector machines: Learning with many relevant features," in *Proc. 10th ECML*, C. Nédellec, C. Rouveiro, Eds., Berlin, Germany, 1998, vol. 1398, pp. 137–142.
- [27] F. Sebastiani, "Machine learning in automated text categorization," *ACM Comput. Surv.*, vol. 34, no. 1, pp. 1–47, Mar. 2002.
- [28] I. Rish, "An empirical study of the naive Bayes classifier," in *Proc. IJCAI Workshop Empirical Methods AI*, 2001, pp. 41–46.
- [29] J. Stearley, "Towards informatic analysis of syslogs," in *Proc. IEEE Int. Conf. Cluster Comput.*, Sep. 2004, pp. 309–318.
- [30] A. McCallum and K. Nigam, "A comparison of event models for naive Bayes text classification," in *Proc. AAAI Workshop Learn. Text Categorization*, 1998.
- [31] T. M. Mitchell, *Machine Learning*. New York: McGraw-Hill, 1997.
- [32] J. L. Hellerstein, S. Ma, and C. shing Perng, "Discover actionable patterns in event data," *IBM Syst. J.*, vol. 41, no. 3, pp. 475–493, 2002.
- [33] K. Houk, S. B. Calo, and A. J. Finkel, "Towards a practical alarm correlation system," in *Proc. Integr. Netw. Manage. IV*, 1995, pp. 226–237.
- [34] S. Fu and C.-Z. Xu, "Exploring event correlation for failure prediction in coalitions of clusters," in *Proc. SC*, 2007, pp. 1–12.
- [35] J. L. Hellerstein, S. Ma, and C.-S. Perng, "Discovering actionable patterns in event data," *IBM Syst. J.*, vol. 41, no. 3, pp. 475–493, 2002.
- [36] T. Li and S. Ma, "Mining temporal patterns without predefined time windows," in *Proc. 4th IEEE ICDM*, 2004, pp. 451–454.
- [37] A. Webb, *Statistical Pattern Recognition*. New York: Wiley, 2002.
- [38] J. Cook, Z. Du, C. Liu, and A. Wolf, "Discovering models of behavior for concurrent workflows," *Comput. Ind.*, vol. 53, no. 3, pp. 297–319, Apr. 2004.
- [39] C.-S. Perng, D. Thoenen, G. Grabarnik, S. Ma, and J. Hellerstein, "Data-driven validation, completion and construction of event relationship networks," in *Proc. 9th ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2003, pp. 729–734.
- [40] D. Thoenen, J. Riosa, and J. L. Hellerstein, "Event relationship networks: A framework for action oriented analysis for event management," in *Proc. Int. Symp. Integr. Netw. Manage.*, 2001, pp. 593–606.
- [41] N. Cesa-Bianchi, C. Gentile, and L. Zaniboni, "Hierarchical classification: Combining Bayes with SVM," in *Proc. 23rd ICML*, 2006, pp. 177–184.

Photographs and biographies of all authors not available at the time of publication.