Data-driven Techniques in Computing System Management

Tao Li, Nanjing University of Posts and Telecommunications & Florida International University
Chunqiu Zeng, Florida International University
Yexi Jiang, Florida International University
Wubai Zhou, Florida International University
Liang Tang, Florida International University
Zheng Liu, Nanjing University of Posts and Telecommunications
Yue Huang, Nanjing University of Posts and Telecommunications

Modern forms of computing systems are becoming progressively more complex, with an increasing number of heterogeneous hardware and software components. As a result, it is quite challenging to manage these complex systems and meet the requirements in manageability, dependability, and performance that are demanded by enterprise customers. The survey presents a variety of data-driven techniques and applications with a focus on computing system management. In particular, the survey introduces the intelligent methods for event generation that can transform diverse log data sources into structured events, reviews different types of event patterns and the corresponding event mining techniques, and summarizes various event summarization methods and data-driven approaches for problem diagnosis in system management. We hope the survey will provide a good overview for data-driven techniques in computing system management.

General Terms: Survey, Design, Algorithms

Additional Key Words and Phrases: Computing System Management, Data Mining, Application

1. DATA-DRIVEN SYSTEM MANAGEMENT

Large and complex systems often have a large number of heterogeneous components and are difficult to monitor, maintain, and manage. Traditionally, the system management methods mainly based on domain experts through a cumbersome, error-prone and labor-intensive process, which called knowledge acquisition process. This process translates domain knowledge into operational rules, policies, and dependency models. Generally, it is an expensive process for managing such complex systems with the dynamically changing environment. For instance, in many companies, the maintenance costs about 30% to 70% of their information technology resources [Research 2003]. As a result, intelligent and efficient approaches are greatly needed for monitoring and managing large and complex systems.

Significant initiatives, such as the IBM Autonomic Computing (AC) initiative, with the aim of building autonomic systems capable of self-managing [Horn 2001; Kephart and Chess 2003], led to awareness of automatic system management in both scientific and industrial communities as well as facilitated to bring in more automated and sophisticated procedures, which improve the productivity and guarantee the overall quality of the delivered service.

To realize the goal of self-management, systems need to automatically monitor, characterize, and understand their behaviors and dynamics, mine events to uncover useful patterns, and acquire valuable knowledge from historical log/event data.

Figure 1 presents an architecture of data-driven system management [Li et al. 2010; Li 2015], whose key components are described below.

Author's addresses: T. Li, School of Computer Science, Nanjing University of Posts and Telecommunication & School of Computing and Information Sciences, Florida International University, email: taoli@cs.fiu.edu. C. Zeng, Y. Jiang, W. Zhou and L. Tang are with School of Computing and Information Sciences, Florida International University. Z. Liu and Y. Huang are with Jiangsu BDSIP Key Lab, School of Computer Science, Nanjing University of Posts and Telecommunications. The work was supported in part by the National Science Foundation under Grant Nos. IIS-1213026, CNS-1126619, CNS-1461926, Chinese National Natural Science Foundation under grant 91646116, Scientific and Technological Support Project (Society) of Jiangsu Province No. BE2016776, Ministry of Education/China Mobile joint research grant under Project No.5-10, and an FIU Dissertation Year Fellowship.

DOI: http://dx.doi.org/10.1145/0000000.0000000

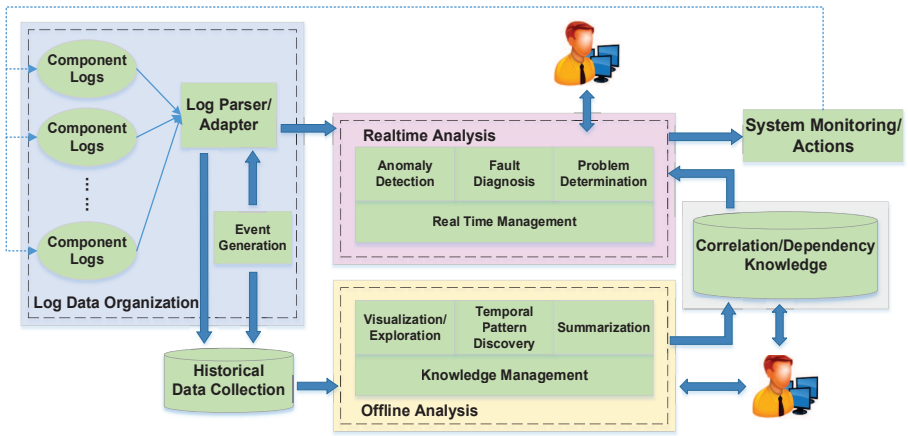

Fig. 1. The architecture of an integrated data-driven system management framework.

— Log Data Organization: The log data are generated and collected by different system components, possibly with instrumented applications. *Log Parser/Adapter* and *Event Generation* enable generic data collection, integration, and transformation from multiple heterogeneous data sources into the historical data collection.
— Real-time Analysis: This module processes and analyzes incoming data in real time. Based on the knowledge obtained from offline analysis module, real-time analysis performs online operations and actions. Some representative real-time analysis tasks include problem diagnosis and determination, anomaly detection, and performance prediction.
— Offline Analysis: This module constructs and derives knowledge repositories (such as correlation rules, dependency patterns, and relationship knowledge) from historical data. Representative offline analysis techniques include temporal pattern discovery, event summarization, and correlation rule construction.

1.1. The Scope of the Survey

In computing system management, the historical data often include the raw log data, performance data (e.g., time series), discrete system events, monitoring events, and incident tickets. The service providers usually keep track of the raw log data, the time series data as well as the historical system events (generated by the production systems), monitoring events (generated by the monitoring system) and incident tickets (edited by humans) to diagnose incoming system issues. The time series data often report the observed system performance; the raw log data, system events and monitoring events describe system internal operations, alerts and faults; and the incident tickets reveal the human judgements on these events in terms of system incidents. Data-driven techniques, including techniques for log data organization, real-time analysis, and offline analysis, can automatically and efficiently extract valuable knowledge from historical log/event data and play an important role in computing system management. The purpose of this survey is to present a variety of data-driven techniques and applications with a focus on computing system management.

There are a few existing surveys on complex event processing [Owens 2007; Fülöp et al. 2010; de Carvalho et al. 2013], data stream processing [Cugola and Margara 2012], and data and event mining [Wrench et al. 2016; Liu et al. 2016]. However, to the best of our knowledge, there is no work that provides a systematic and comprehensive coverage on data-driven techniques in computing system management, covering different real-time and offline tasks (including log parsing, event generation, classification, clustering, pattern mining, summarization, and problem diagnosis) acted on various types of data (including raw logs, time series, historical system events, monitoring events, and incident tickets). Table I summarizes the main content of the related surveys. In particular, this paper compares different event generation methods for transforming diverse log data sources into structured events, reviews different types of event patterns and the corresponding event mining techniques, and

summarizes various event summarization methods and data-driven approaches for problem diagnosis in system management.

Table I. Relations with other related surveys

Related Surveys	Summary	Pros & Cons
[Owens 2007]	. Introduces the brief history, basic key words, phrases and concepts related to event processing, and its several application scenarios . Presents several existing event processing systems and query languages	(+) A complete introduction on basic concepts of event and event processing system (-) Limited coverage on technical details
[Fülöp et al. 2010]	. Offers a review on terminology, research areas/achievements, existing solutions/tools and open issues . Examines the related fields of predictive analytics and those specific methods applicable in telecommunication	(+) An informative review on existing commercial and academic tools in event processing and predictive analytics (+) Good comparison and classification of different methods (-) Mainly focused on predictive methods in telecommunication domain (-) Lack of a systematic approach in organization and presentation and in-depth discussion on technical details
[Cugola and Margara 2012]	. Reviews information flow processing (IFP) systems for event processing from the perspective of data stream management	(+) A good comparison on different data stream management systems for event processing (-) Mainly focused on data stream management
[de Carvalho et al. 2013]	. Discusses the weaknesses and limitations on a set of state-of-the-art event processing systems . mainly focuses on the systems requiring high throughputs over large amount of data	(+) Good reference material for existing commercial event processing systems (-) Limited coverage, and only present some popular event processing systems at a high level (-) Lack of technical details
[Wrench et al. 2016]	. Clarifies the positions of Event Stream Processing (ESP) and Complex Event Processing (CEP) . Outlines the range of Data Mining opportunities within ESP and CEP	(+) A systematic review of the core concepts and difference between ESP and CEP (-) Mainly focused on event stream data processing
This paper	. provides a comprehensive review of data-driven techniques in computing system management . Summarizes a wide range of techniques acted on different data types	(+) A comprehensive and systematic review on data-driven techniques (+) Good coverage on technical details (-) mainly focused on computing system management.

Learning about data-driven techniques in computing system management is challenging, as it is an inter-disciplinary field that requires familiarity with several research areas and the relevant literature is scattered in a variety of publication venues such as ACM International Conference on Knowledge Discovery and Data Mining (ACM SIGKDD), IEEE International Conference in Data Mining (IEEE ICDM), IEEE/IFIP Network Operations and Management Symposium (NOMS), International Conference on Network and Service Management (CNSM), and IFIP/IEEE Symposium on Integrated Network and Service Management (IM). We hope that this survey will make the field easier to approach by providing a good starting point for readers not familiar with the topic as well as a comprehensive reference for those working in the field.

The rest of this survey is organized as follows. Section 2 investigates the methods that can transform the log data in disparate formats and contents into a canonical form, which creates consistency and enables the correlation discovery across multiple logs. Section 3 reviews different types of event patterns and presents the corresponding event mining techniques as well as the application scenarios. Section 4 introduces time lag mining for temporal patterns, and both non-parametric and parametric methods are discussed for discovering time lags between correlated events. Section 5 provides a summary of different event summarization techniques and presents an algorithm independent summarization framework that can efficiently support different summarization techniques in real applications. Section 6 summarizes several data-driven approaches that can help administrators perform a detailed diagnosis for detecting system issues. Finally Section 7 concludes the survey.

2. EVENT GENERATION: FROM LOGS TO EVENTS

The data in the log/trace files indicate the status of each component and are usually collected or reported when some event occurs. Contents of the data may include the running states of the component (e.g., started, interrupted, connected, and stopped), its CPU utilization, and its parameter values. Since most computing systems record the internal operations, status, and errors by logs, it is straightforward to obtain the system events from the system logs. In this section we mainly focus on the methodologies of event generation from the system logs. In system management, many researchers have studied system event mining and proposed many techniques and algorithms for discovering the abnormal system

behaviors and relationships of events/system components [Peng et al. 2007; Xu et al. 2008; Hellerstein et al. 2002a; Li et al. 2005; Gao et al. 2009; Oliner et al. 2008; Wang et al. 2010; Kiernan and Terzi 2009]. In those studies, the data is a collection of discrete items or structured events, rather than textual log messages. Discrete or structured events are much easier to be visualized and explored by human experts than raw textual log messages. Many visualization toolkits were developed to provide a quick overview of system behaviors over a large collection of discrete events. However, most of the computing systems only generate textual logs containing detailed information. Therefore, there is a need to convert the textual logs into discrete or structured events.

2.1. An Example of Converting Textual Logs into Events

Before introducing the details of the approaches, an example is presented to illustrate the benefits of the conversion.

Figure 2 presents an example of messages from a Simple File Transfer Protocol (SFTP) log collected from a FTP software called FileZilla [url 2015a]. Each message in the example describing a certain event. To understand and analyze the behaviors of FTP visits, these messages are often converted into types of events and organized as timelines. The event timeline formed by the example messages is shown in Figure 3. Note that the event timelines can help people understand the event behaviors and enable the discovery of event patterns.

No.	Message
s_1	2010-05-02 00:21:39 Command: put "E:/Tomcat/apps/index.html" "/disk/...
s_2	2010-05-02 00:21:40 Status: File transfer successful, transferred 823 bytes...
s_3	2010-05-02 00:21:41 Command: cd "/disk/storage006/users/lt...
s_4	2010-05-02 00:21:42 Command: cd "/disk/storage006/users/lt...
s_5	2010-05-02 00:21:42 Command: cd "/disk/storage006/users/lt...
s_6	2010-05-02 00:21:42 Command: put "E:/Tomcat/apps/record1.html" "/disk/...
s_7	2010-05-02 00:21:42 Status: Listing directory /disk/storage006/users/lt...
s_8	2010-05-02 00:21:42 Status: File transfer successful, transferred 1,232 bytes...
s_9	2010-05-02 00:21:42 Command: put "E:/Tomcat/apps/record2.html" "/disk/...
s_{10}	2010-05-02 00:21:42 Response: New directory is: "/disk/storage006/users/lt...
s_{11}	2010-05-02 00:21:42 Command: mkdir "libraries"
s_{12}	2010-05-02 00:21:42 Error: Directory /disk/storage006/users/lt...
s_{13}	2010-05-02 00:21:44 Status: Retrieving directory listing...
s_{14}	2010-05-02 00:21:44 Command: ls
s_{15}	2010-05-02 00:21:45 Command: cd "/disk/storage006/users/lt...
...	... ...

Fig. 2. An example of FileZilla's log.

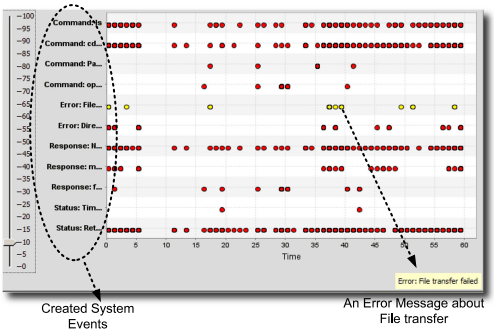

Fig. 3. Event timeline for the FileZilla's log example.

Converting log messages to events provides the capability of canonically describing the semantics of log data and improves the ability of correlating the logs from multiple components. Due to the heterogeneous nature of current systems, the log generating mechanisms result in disparate formats and contents focused on individual components. Each component may generate the data using its own format and content. As a result, analyzing errors and logs collected by different components and products is quite challenging due to the variability in log messages [Topol et al. 2003]. By organizing the messages into a set of common semantic events (also termed "situations" or "categories"), i.e., adding a semantic situation type to a message [Li et al. 2010], the transformed representation creates consistency, enables the correlation discovery across multiple logs, and also provides the initial connections of syntax to semantics.

2.2. Potential Solutions

To convert a collection of textual logs into system events, there are generally three types of solutions: log-parser-based solutions, classification-based solutions, and clustering-based solutions. In this section, we provide an overview for the three types of solutions. The pros and cons of the three types of approaches are briefly summarized in Table II.

The most straightforward solution is the log-parser-based approach, in which a log parser is built for log files in different formats. Since the log users may be very familiar with the meanings of each log message, they can write a simple text parser and accurately extract all the needed system information from the logs. However, for a large and complex system, implementing the log parser is not easy. The user may not understand all possible log message generation mechanisms. It is also not efficient to implement different parsers for different types of system logs. Although there are some common formats of logs, how to adapt a log parser for different types of logs is still a challenging issue in reality.

In many event mining applications, people only need to know the type of an event since many mining algorithms are mainly discovering the unknown relationship between different event types. Consequently, in those applications, there is no need to extract the detailed information of a system event, such as system metrics. Therefore, converting a log message into an event is equivalent to finding the type of the log message. As a result, the conversion problem becomes a text classification problem. Many classification algorithms, such as support vector machines (SVMs), can be applied to solve the classification problem. The main disadvantage of these types of methods (e.g., classification-based methods) is that the classification algorithms require the users to provide the labeled training data in advance. In other words, a set of labeled log messages has to be prepared. For a complicated large system, this can only be done by domain experts, so it is time consuming and costly.

Clustering-based approaches do not require the user to prepare the labeled training data. They can infer the event type from the log message itself. Although the inferred event types may not be as accurate as those obtained using the classification-based or log-parser-based approaches, they are often acceptable for event mining algorithms or human exploration.

Table II. Summary of the three types of event generation approaches

Approach Type	Pros	Cons	Application Scenarios
Log Parser	Very accurate.	Require the user to understand system logs. Hard to adapt to various system logs with different formats. Require human efforts to develop the log parser software.	The application needs accurate generated system events, such as alarm detection systems or monitoring systems.
Classification	Accurate and can be adapted to various system logs.	Require the user to provide the training log data. Labeling the log data by domain experts may be costly and time consuming.	Easy to have labeled training log data.
Clustering	Do not need a lot of human effort and can be adapted to various system logs.	Not very accurate.	Some event mining applications that can be tolerant to some errors or noisy events.

2.3. Log Parser

Implementing a log parser is a straightforward solution to converting textual logs into structural events. However, the approach requires the familiarity with the type and format of each raw message, so that the detailed semantic information can be extracted. Clearly, the approach is not efficient for large complex systems with heterogeneous components having different log-generating mechanisms, and disparate formats and contents.

In software development, there are a lot of log generation libraries, such as log4j[1] and Apache common logging[2]. The log messages generated by these libraries have some standard formats. Hence, many researchers argue that those logs are structural or semi-structural data, rather than pure textual data. However, log messages generated by many other software packages or systems have no such standard formats.

Many researchers have investigated the approaches to building log parsers based on analysis of the source code [Xu et al. 2008]. In particular, using the Common Base Event (CBE) format, the Generic Log Adapter (GLA) provided in the IBM Autonomic Computing toolkit allows for generic data collection from heterogeneous data sources [Grabarnik et al. 2004]. For those approaches, the input is the source code of the software that generates the log data. The output is a log parser or some information to be embedded into a log parser. Many modern computing systems are open-source, so these approaches can be applied to many software packages, such as Apache Tomcat and Hadoop.

The log generation code usually has some fixed patterns. As mentioned in [Xu et al. 2008], the source code can be viewed as the schema of the logs and the structure of the generated logs can be inferred from the schema. Then, the log parser makes use of the schema to determine the event type of each log message. It also can extract the variable values as the attributes of the event.

2.4. Log Message Classification

In many applications, the monitoring and analysis only need to extract the types of events described by the log messages. In these scenarios, the detailed information, e.g., attribute values of every log message, are not necessary. Moreover, the log messages are allowed to be classified in a hierarchal manner.

[1]http://logging.apache.org/log4j/1.2/

[2]http://commons.apache.org/proper/commons-logging/

A straightforward approach to identifying the event types of log messages is the classification method, which categorizes a log message into several pre-defined event types. A simple classification method is to define a regular expression pattern for an event type. Then, when a log message is collected, if it matches a given regular expression, it will then be categorized to the corresponding event type. This type of classification is also called "filter."

Another approach for log message classification is the learning-based method. Users can provide some labeled log messages, where the event type of each log message is assigned. Then, a learning algorithm builds a classification model using the labeled data to classify incoming log messages. The classification model is built based on the joint distribution of the message terms and the corresponding event types. In text mining, a traditional approach for handling the text information is the *bag-of-words* model [Salton and McGill 1984]. In such a model, a log message is split into a collection of terms, where a term represents one feature. For the binary vector space representation, a feature value is 1 or 0 where 1 indicates that the term appears in the log message and 0 indicates that the term does not appear in the log message. Then, any classification algorithms can be applied to train a classification model. However, the learning-based method is not practical in many real-world system management applications. The main reason is that the human cost of labeling log messages may not be less than the cost of implementing a partial log parser or a regular-expression-based filter. The log messages are often short. Given a set of log messages, if you already know how to label them, then it may not be more difficult to define the corresponding parsing logic or write the regular expression.

2.5. Log Message Clustering

Log message clustering is an unsupervised method to categorize the logs to events [Li and Peng 2005]. Since it does not require preparing a set of labeled training data or regular expressions, this approach is more practical and useful.

Recent studies [Aharon et al. 2009; Makanju et al. 2009] apply data clustering techniques to divide log messages into separate groups, where each group corresponds a certain event type. Traditionally, the clustering approaches, based on the *bag-of-words* model, cannot achieve good performance since the log messages often have short message length with a large vocabulary size [Stearley 2004]. Message clustering methods using the structure/format information [Aharon et al. 2009; Makanju et al. 2009] have also been proposed. As a result, these clustering techniques only perform well for structured logs as their performances largely depend on the message formats/structures. In [Aharon et al. 2009], Aharon et al. introduced a similarity function based on the number of matched words between two log messages. If a group of log messages belong to the same event type, they must be generated by the same piece of codes, which is also called "template" in [Aharon et al. 2009]. The only different words in these log messages are the variable terms. Another limitation is that this similarity function treats every word equally (as having equal importance).

In [Makanju et al. 2009], a log message clustering algorithm is presented. The algorithm consists of the following four steps: (1) Partition by the number of words (or tokens); (2) Partition by the word positions; (3) Partition by search for bijection; and (4) Discover the descriptive words from each partition. It should be pointed out that the algorithm tries to identify the template words (non-variable terms) in the clustering algorithm. In the second step, it first discovers some word positions in which the words have a large number of occurrences (say, most frequent words), then partitions the logs using those words' positions. These frequent words actually are very likely to be the template words (non-variable terms). In the third step, the bijection is the relationship between two elements in the most frequent word positions. Once the bijection is found, different token values (or words) of the bijection can be partitioned into different clusters. In other words, the algorithm aims to partition the log messages using the value of those template words. Therefore, the two steps of [Makanju et al. 2009] partition the log messages according to the positions and values of the potential template words. This method can run very fast since the time complexity is linear, so it is applicable for massive log data generated by production systems. However, it still may not work for some types of log data. For example, if the variable terms of one template have a different number of words in different log messages, the first step would fail and the following three steps would not be correct as well. Here the main challenge is the identification of the template words.

Recently Liang and Li [Tang and Li 2010] presented a tree-structure-based clustering algorithm, `LogTree`, which computes the similarity of log messages based on the established tree representation in the clustering process. The structural and format information in log messages are used in the `LogTree`

algorithm and the event can be generated more effectively and efficiently utilizing message segment table. The LogTree algorithm builds tree patterns for log messages, where the root of a tree pattern is the category of the log message (e.g., event types), and the leaves are the field information in messages. LogSig, a message-signature-based clustering algorithm for converting textual logs into system events is proposed in [Tang et al. 2011]. Although log messages have various types and different variables and parameters (e.g., host name and IP address), different log messages often have common subsequences describing the semantic information. These common subsequences are treated as the **signatures** of event types. LogSig extracts the most representative message signatures, and categorizes the textual log messages into different event types based on the extracted signatures.

3. EVENT PATTERN MINING

3.1. Introduction

Temporal data is prevalent across different application domains. Basically, temporal data is often referred to as a collection of data items associated with timestamps, describing the state changes or evolving trends over time. A typical example for temporal data in system management is illustrated in Figure 4. System monitoring, one important component in system management, tracks the states of a system by collecting system information such as the CPU utilization, the memory usage, the number of data bytes written and read on disk, the amount of data received and sent through the network, the sequence of requests and responses processed on an application server, etc. All the system information is collected with a fixed frequency and each data item is recorded with its timestamp.

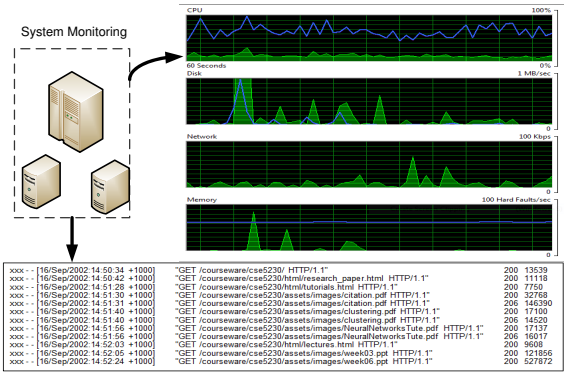

Fig. 4. Temporal data in system management.

In light of the different types of data items, temporal data are generally divided into two categories: event data and time series. Time series are used to describe the temporal data where the value of the data item is continuous. Event data denotes the temporal data with discrete data item values. For example, in the system management shown in Figure 4, the CPU utilization, the memory usage, and the number of data bytes written and read on disk, the amount of data received and sent through the network are represented as time series. The requests and responses processed by an application server in system management and the posts occurring in social media are referred to as event data since their values are categorical. Although mining both types of temporal data has attracted increasing attention in recent years, our main focus in this survey is on the event data.

Event pattern mining aims to find the hidden patterns, latent trends, or other interesting relationships among events. The techniques used in event pattern mining are related in many research areas, like data mining, database, machine learning, and statistics. The event patterns discovered can be used by analysts to make decisions for the behaviors of future events. Diverse requirements in various application domains dictate different types of event patterns for problem solving. In this section, we provide a survey of different types of event patterns and present the corresponding event mining techniques as well as the application scenarios.

3.2. Sequential Pattern

Agrawal and Srikant propose the sequential pattern mining problem at first in [Agrawal and Srikant 1995], with the purpose of discovering frequent subsequences from a sequence database. In comparison

with frequent pattern mining [Agrawal et al. 1993], which is to find frequent item sets within each transaction, sequential pattern mining mainly focuses on the patterns across different transactions by considering their sequential order.

Problem Definition. The problem of sequential pattern mining can be formally described as follows. A sequence database D is defined as a set of sequences $D = \{S_0, S_1, ..., S_i, ..., S_n\}$. Each sequence $S_i \in D$ is a sequence of itemsets, denoted as $S_i =< T_1, ..., T_j, ..., T_m >$ where T_j is a non-empty set of items and all the itemsets in the sequence are organized in temporal order. For a sequence, if itemset T_i happens before itemset T_j, we denote it as $T_i \prec T_j$. A sequence containing k itemsets is also referred to as a k-sequence. A non-empty itemset T_j can be further represented as $T_j = \{I_1, I_2, ..., I_k, ..., I_l\}$, where I_k is an item. In sequential pattern mining, the non-empty itemset T_j is also referred to as an event, so these two concepts are interchangeable in this paper.

A sequence $R =< T_{R1}, T_{R2}..., T_{Rm} >$ is a subsequence of $S =< T_{S1}, T_{S2}, ..., T_{Sn} >$ if existing m itemsets $T_{Si1} \prec T_{Si2} \prec ... \prec T_{Sim}$ in S satisfy $T_{R1} \subseteq T_{Si1}$, $T_{R2} \subseteq T_{Si2}$, ..., $T_{Rm} \subseteq T_{Sim}$. It is also said that S contains R.

Given a sequence database D, we define the support of a sequence S as the fraction of all data sequences that contains S. Sequential patterns are those sequences whose supports are not less than a predefined threshold with respect to the sequence database D.

Mining sequential patterns is a computationally challenging task because there are exponentially many sequences contained in a given data sequence [Tan et al. 2005]. Since the sequential pattern mining problem was introduced in [Agrawal and Srikant 1995], various studies and strategies with respect to this problem have been presented. Typically, the related algorithms in the literature for sequential pattern discovery are classified broadly into two groups: Apriori-based and pattern-growth-based methods.

Apriori-Based Algorithms. The main characteristics of Apriori-based algorithms involve three aspects [Irfan and Anoop 2012]. First, Apriori-based algorithms are regarded as level-wise search algorithms since all the k-sequences are constructed in the k^{th} iteration of the algorithms when traversing the search space. Second, after the candidates are generated, pruning techniques are used for removing those candidates which cannot be sequential patterns. Third, Apriori-based algorithms often require multiple scans of the sequence database with a high I/O cost.

Accompanying the definition of sequential pattern mining in [Agrawal and Srikant 1995], the AprioriAll algorithm is first proposed to address the sequential pattern discovery problem utilizing the Apriori property (which is also referred to as the anti-monotone property). The Apriori property is that, if a sequence is not a sequential pattern, then all of its supersequences cannot be sequential patterns either. On the basis of Apriori and AprioriAll algorithms, a set of improved algorithms utilizing the Apriori property are proposed. The GSP (i.e., **G**eneralized **S**equential **P**attern) algorithm proposed in [Srikant and Agrawal 1996] requires multiple scans of the sequence database. In comparison with AprioriAll, the GSP algorithm uses the number of items as the length of a sequence instead of the number of itemsets. An algorithm named SPIRIT (i.e., **S**equential **P**attern m**I**ning with **R**egular expressIon consTraints) is proposed in [Garofalakis et al. 1999] for sequential pattern mining. This algorithm is capable of discovering sequential patterns with flexible constraints represented in regular expression. SPADE (i.e., **S**equential **PA**ttern **D**iscovery using **E**quivalence classes), a new algorithm for fast mining sequential pattern in a large database, is presented in [Zaki 2001]. The SPADE algorithm transforms the sequential dataset into a vertical ID-List database format. Utilizing this format, each sequence is linked with a list of objects occurring in the sequence and along with the timestamps. All the sequential patterns can be enumerated via simple temporal joins on ID-Lists. For propose of finishing a sequential pattern mining task, the SPADE algorithm requires three passes of database scanning. To reduce the merge cost, SPAM (i.e., **S**equential **PA**ttern **M**ining) is proposed in [Ayres et al. 2002]. The SPAM algorithm represents each ID-List as a vertical bitmap data structure. As a consequence, the space for storing ID-Lists is reduced so that the ID-Lists can be entriely stored in the main memory during sequential pattern mining.

Pattern-Growth-Based Algorithms. Pattern-growth-based algorithms utilize efficient data structures to prune candidate sequences early in the sequential pattern mining process. The search space of sequential patterns is typically represented as a tree data structure. The Apriori property can be used when the tree is traversed for searching sequential patterns in either breadth-first or depth-first order. When the number of candidate sequences is large, for managing memory efficiently, the tree data structure representation allows partitioning of the corresponding search space [Irfan and Anoop

2012]. After the partition is completed, each smaller space can be searched in parallel. Related literatures have been proposed several pattern-growth-based algorithms for handling various application scenarios.

FreeSpan (i.e., **Fre**qu**e**nt pattern-projected **S**equential **Pa**ttern**n**) is one of the initial pattern-growth-based algorithms for sequential pattern mining [Han et al. 2000]. The novel idea of this approach is to integrate the mining of sequential patterns with the mining of frequent patterns. It also uses projected sequence databases to confine the search and the growth of subsequence fragments. Given a sequence $S =< T_1, ..., T_k, ..., T_m >$, the itemset $T = \cup_{k=1}^{m} T_k$ is the projected itemset of S. A useful property is that, if an itemset T is infrequent, any sequence whose projected itemset is a superset of T cannot be a sequential pattern. Based on the property, the efforts of candidate subsequence generations are greatly reduced during the process of sequential pattern mining.

The WAP-Mine (i.e., **W**eb **A**ccess **P**attern **Mine**) algorithm is proposed to mine access patterns from web logs in [Pei et al. 2000]. This algorithm takes advantage of a novel data structure named WAP-tree (**W**eb **A**ccess **P**attern **tree**) for mining access patterns from pieces of logs efficiently. This algorithm takes two passes of scanning the sequence database to build the WAP-tree. Although this algorithm is able to avoid the issue of generating a huge number of candidates like Apriori-based approach, WAP-Mine suffers from the huge memory consumption problem because of recursive reconstruction of numerous intermediate WAP-trees during sequential pattern mining.

PrefixSpan (i.e., **Prefix**-projected **S**equential **pa**ttern mining), a novel sequential pattern mining method, is proposed in [Pei et al. 2001]. The PrefixSpan algorithm takes advantage of prefix projection techniques to substantially reduce the size of projected databases and leads to efficient processing for sequential pattern mining. The related study shows that the performance of PrefixSpan algorithm is better than both the Apriori-based GSP algorithm and the pattern-growth-based FreeSpan algorithm in mining large sequence databases.

3.3. Fully Dependent Pattern

In system management, the system administrators typically have much more interest in the patterns that are capable of predicting undesirable situations such as service disruptions and security intrusions. As a matter of fact, however, such patterns do not happen frequently but have statistically significant dependency, especially in a well-managed systems. Therefore, traditional frequent pattern mining methods are no longer suitable/feasible in the application scenarios of system management.

Several issues cause the pattern discovery to be a challenging task. First, the patterns to be found are typically infrequent but can be statistically dependent events which can provide insights into the system. For example, in a computer network, the knowledge acquired from the dependent temporal event sequences enables the prediction of incoming events. In particular, if the events are related to malfunctions or service disruptions, such knowledge can be used for problem detection and root cause determination. Unfortunately, in order to discover the infrequent patterns, the support thresholds should be set very low. Consequently, this would raise a new issue that a large number of unimportant patterns are mixed in with a few patterns of interest. Second, the event data are collected in a noisy environment. For the applications depending on networks, data may be lost because of the traffic-overloaded communication lines or the overflowing router buffer. In addition, the data may be corrupted because of human errors during data processing. As a result, some valid patterns may be missed due to the presence of noise. Third, the distribution of events is often skewed in the whole data collection. As a consequence, a fixed minimum support threshold is not applicable for mining patterns from such event data.

In [Liang et al. 2002], to address the aforementioned issues, the concept of a fully dependent pattern, known as a d-pattern, is proposed to discover the infrequent, but dependent event patterns. To avoid the issue brought about by the fixed minimum support threshold, the hypothesis test is applied for a dependency test.

Let Ω be the item space and assume that there are k distinct items (i.e., $|\Omega| = k$) in the item space. An event is an itemset $E = \{I_1, ..., I_i, ..., I_m\}$ where I_i is an item from the item space. An event is considered to be a random variable taking values of all possible subsets of the item space. Assume that there exists an unknown distribution P on the 2^k possible states of E. Given an event database D, all events $E \in D$ are assumed to independently and identically follow the distribution P. Obviously, if all

the items in an event E are independent, the probability of event E occurring can be derived as follows.

$$p_E = P(I_1, ..., I_i, ...I_m) = \prod_{i=1}^{m} P(I_i) = \prod_{i=1}^{m} p_i, \tag{1}$$

where p_i is the probability of occurrence of item I_i. On the other hand, the probability of their occurrence should be higher than the one under the independent assumption. Let $p^* = \prod_{i=1}^{m} p_i$; then the hypothesis test for dependency is given as below.

$$\begin{aligned} H_0\ (null\ hypothesis) &: p_E = p^* \\ H_1\ (alternative\ hypothesis) &: p_E > p^* \end{aligned} \tag{2}$$

Since the real values of p_i's are not available, in order to calculate p^*, all p_is are replaced by their estimators $\hat{p_i} = \frac{support(I_i)}{|D|}$, where $support(I_i)$ is the number of events containing item I_i and $|D|$ is the number of events in the event database D. If the $null\ hypothesis$ in Eq. (2) is true, then the random variable $C_E = support(E)$ follows a binomial distribution $B(p^*, n)$, where $support(E)$ is the occurring number of event E in event database D. The $null\ hypothesis$ should be rejected if C_E is bigger than some threshold. The threshold can be determined by a pre-specified significance level α, where α is known as the upper bound for the probability of a false positive.

Accordingly, a new random variable can be derived as follows:

$$Z = \frac{C_E - np^*}{\sqrt{np^*(1-p^*)}}. \tag{3}$$

Typically, the number of events in D is very large, so Z is assumed to follow the standard normal distribution $N(0, 1)$ according to the central limiting theorem [Durrett 2010]. With Eq. (3), the dependency test is equivalent to test whether C_E is greater than $minsup(E)$, which is given in

$$minsup(E) = np^* + z_\alpha \sqrt{np^*(1-p^*)}, \tag{4}$$

where z_α is the corresponding $1-\alpha$ normal quantile which can be easily found in any normal table or calculated. Different events Es should have different $minsup(E)$s since their p^*s have different values.

However, the dependency test is neither upward nor downward closed. Therefore, it is computationally infeasible to discover all events that are dependent. In order to discover all such dependent events efficiently, a stronger dependency condition is given to define such patterns, which are referred to as d-patterns.

Definition 3.1. Given a significant level α, an event $E = \{I_1, ..., I_i, ..., I_m\}$ ($m \geq 2$) is a qualified d-pattern if the two conditions below are satisfied.
(1) $support(E) \geq minsup(E)$.
(2) If $E_S \subseteq E$ and $|E_S| > 1$, then $support(E_S) \geq minsup(E_S)$.

With condition (2) in Definition 3.1, the d-pattern can be proved to be downward closed. It can also be shown that the minimum support $minsup(E)$ increases as the frequency of items increases, when the product $p^* \leq 0.5$. With the help of the downward property, the d-patterns can be efficiently discovered by a level-wise search algorithm similar to the Apriori algorithm.

With the definition of d-pattern, three fundamental issues of traditional association mining are addressed so that it is capable of discovering the patterns which are infrequent from noise and unevenly distributed data. Although the strong dependency test requires not only an event but also all its subsets satisfying the dependency test, a level-wise algorithm can be constructed to discover all d-patterns regardless of their supports.

3.4. Partially Periodic Dependent Pattern

Periodicity is one of the most common phenomena in the real world. The characteristics of periodic patterns help analysts gain great insights into the data. First, periodic patterns indicate the persistent occurrence of events. With the help of this characteristic, periodic patterns can be applied for the anomaly detection and problem diagnosis. Second, periodic patterns provide evidence for the predictability of events. It is helpful for analysts to predict the behavior of coming events and study the evolving trends in the future.

However, the limitations of data collection methods and the inherent complexity of periodic behaviors pose a great challenge for periodic pattern detection. In real practice, several issues need to be considered [Ma and Hellerstein 2001b].

— First, the periodic behaviors are not persistent. Take complex networks as an example. Periodic problem reports are initialized when there occurs an exception such as disconnection of the network and are terminated once the problem is fixed.
— Second, imprecise time information is recorded due to lack of clock synchronization, rounding, and network delays.
— Third, periods are not known in advance. It is computationally infeasible to discover the true period by exhaustively searching for all possible periods.
— Furthermore, a fixed support level has difficulty in capturing the periods for all the patterns, since the numbers of occurrences of periodic patterns vary drastically. For example, a daily periodic pattern results in at most seven occurrences in a week, while an hourly pattern results in 168 occurrences for one week.
— Finally, events may be missing from a periodic pattern, or random events may be introduced into a periodic pattern. As a result, the periodicity may be disrupted due to noise.

To discover periodic patterns considering the above issues, partial patterns (i.e., p-patterns) are discussed in this section.

Problem Description. Note that an event is defined as a tuple $(type, time)$, where $type$ is the event type and $time$ is the occurring timestamp of the event.

Definition 3.2. An **event sequence** is defined as a collection of events ordered by the occurring timestamps, i.e., $S =< e_1, ..., e_i..., e_n >$, where $e_i = (type_i, time_i)$. A **point sequence** is an ordered collection of timestamps with respect to a given event type A, $P_A =< t_1, t_2, ..., t_i, ..., t_n >$, where t_i is a timestamp.

Definition 3.3. A **partially periodic point process** of a given event type A is a point sequence $P_A =< t_1, ..., t_i, ..., t_n >$. Assume p is the period and δ is the time tolerance. If t_i and t_{i+1} are on the same on-segment, then $t_{i+1} - t_i = p \pm \delta$, where $1 \leq i \leq n$.

Based on the definition of a **partially periodic point process**, the definition of a partially periodic temporal association is given as follows.

Definition 3.4. Given an event sequence S, let T_S be the set of all the event types occurring in S, δ be the time tolerance of period length, ω be the length of time window, $minsup$ be the minimum support threshold, and p be the period length. A set of event types $T \subseteq T_S$ is a **partially periodic temporal association**, referred to as a **p-pattern**, if the number of qualified instances of T in S exceeds the minimum support threshold $minsup$. A qualified instance S_1 satisfies the following conditions.
(1) All types of events in T happen in S_1, where there exists a timestamp t such that for all $e_i \in S_1$, $t \leq time(e_i) \leq t + \omega$.
(2) The point sequence for each event type in S_1 is a partially periodic point process with parameters p and δ.

3.4.1. Solution. According to the definition of the p-pattern, it can be verified that p-patterns satisfy the downward closure property. Because of this property, a level-wise search algorithm can be proposed for computational efficiencies. To discover p-patterns, δ, ω, and $minsup$ are supposed to be given. Therefore, the task for p-pattern discovery includes two steps: finding possible periods p and discovering p-patterns with parameters ω,δ, and $minsup$.

One existing method to find the periods of a point process is to use the **F**ast **F**ourier **T**ransform (i.e., FFT). However, some issues of p-pattern discovery make an FFT algorithm infeasible. The random on-segments and off-segments are introduced in the p-pattern discovery; as a result, an FFT algorithm cannot cope with such a situation well. Moreover, the computational efficiency of FFT is $O(NlogN)$. Here N represents the number of time units. Note that the number of time units is typically large. In [Ma and Hellerstein 2001b], an approach based on chi-squared tests is proposed to find periods. When all the possible periods for each event type are discovered, the subsequent task is to find the p-patterns.

In [Ma and Hellerstein 2001b], according to different orders for period determination and p-pattern discovery, two other algorithms (i.e., association-first and hybrid algorithms) are proposed as well.

The Association-first algorithm starts with temporal association mining and then selects only those associations whose event types occur in a periodic point process with the same period p and tolerance δ. The association-first algorithm suffers a computational complexity comparable to that of temporal mining, where a substantial amount of cost would be caused in the case of large patterns and low support levels. However, the association-first algorithm is more robust to noise [Ma and Hellerstein 2001b]. The hybrid algorithm is a trade-off between the period-first and association-first algorithms. The former provides efficiency while the latter provides robustness to noise.

3.5. Mutually Dependent Pattern

In some application domains, including problem detection in computer networks, intrusion detection in computer systems, and fraud detection in financial systems, normal behaviors dominate compared with rare abnormal behaviors such as failures and intrusions [Ma and Hellerstein 2001a]. Therefore, in these applications, it is more interesting to discover such patterns that comprise infrequent, but highly correlated items than the frequent patterns whose occurring frequencies exceed a predefined minimum support threshold. An example from network management is considered. Three events, i.e., cold start trap, network interface card failure, and unreachable destination, are commonly generated from a router. Typically, the occurrence of cold start trap indicates the router has failed and restarted. Thus, given the occurrences of the first two events, the monitoring system can provide an advanced warning that the third event will happen.

It is difficult to mine such infrequent but strongly correlated patterns. One intuitive way is to apply the traditional frequent pattern mining methods with a very low minimum support threshold to get the initial pattern set. Then, significantly correlated patterns can be identified from these initial pattern set. However, it is impractical since a large number of irrelevant patterns dominate the whole initial pattern set. In order to address the issue, mutually dependent patterns are proposed, which are also known as m-patterns.

Assume that an event is a non-empty set of items associated with its timestamp. Let S be a sequence of events, and E_1 and E_2 be two events. Given S, the dependency of E_1 on E_2 is quantified by the empirical conditional probability denoted by $P_S(E_1|E_2)$.

$$P_S(E_1|E_2) = \frac{support(E_1 \cup E_2)}{support(E_2)}, \tag{5}$$

where $E_1 \cup E_2$ represents a new event containing all the items from E_1 and E_2, and $support(E)$ is the number of occurrences of E in sequence S.

Definition 3.5. Given the minimum dependence threshold $0 \leq minp \leq 1$, two events E_1 and E_2 are **significantly mutually dependent** with respect to the sequence S iff (i.e., if and only if) $P_S(E_1|E_2) \geq minp$ and $P_S(E_2|E_1) \geq minp$.

Definition 3.6. Given a sequence S and the minimum dependence threshold $minp$, let E be an event from S. If any two events $E_1 \subseteq E$ and $E_2 \subseteq E$ are significantly mutually dependent with respect to S, then E is referred to as an **m-pattern**.

According to the definition, an m-pattern can be discovered regardless of the frequency of its occurrence. M-patterns are different from the frequent association rules and correlated patterns.

(1) An m-pattern E requires mutual dependence, which is a two-way dependence. An association rule $E_1 \rightarrow E_2$ only requires one-way dependence (i.e., E_2 depends on E_1). A correlated pattern refers to an itemset whose items are not independent according to a statistical test.
(2) An m-pattern does not require minimum support. This property makes it possible to find all infrequent m-patterns. In contrast, association rule mining is not applicable for infrequent pattern discovery since low minimum support often leads to a large number of irrelevant patterns.
(3) E is an m-pattern if any two sub-events of E are dependent on each other. This requirement makes the correlations in m-patterns more significant than those in both association rules and correlated patterns.

The definition of m-patterns offers several nice properties, which can be used to develop efficient algorithms for m-pattern discovery.

LEMMA 3.7. *An event E is an m-pattern iff $P_S(E - \{I\}|\{I\}) \geq minp$, for every item $I \in E$.*

Similar to frequent itemset discovery, the number of all potential m-patterns is huge. An efficient algorithm is required to search for all m-patterns.

LEMMA 3.8. *Let E' and E be two events satisfying $E' \subseteq E$. If E is an m-pattern, then E' is an m-pattern as well.*

The property given in Lemma 3.8 is the downward closure property of an m-pattern. Similar to the Apriori algorithm, a level-wise search algorithm is proposed for efficient m-pattern discovery.

LEMMA 3.9. *If E is an m-pattern with $minp$, then $\frac{support(E-\{I\})}{support(\{I\})} \geq minp$ for any item $I \in E$.*

Provided along with Lemma 3.9, the supports of patterns found at level 1 and $k-1$ can be used to prune the impossible m-pattern at level k. Clearly, the smaller the candidate set is, the faster the m-pattern searching algorithm can perform.

The m-pattern discovery algorithm [Ma and Hellerstein 2001a] is similar to the Apriori algorithm [Agrawal et al. 1993]. The only difference is that more pruning techniques can be incorporated according to Lemma 3.8 and Lemma 3.9.

3.6. T-Pattern

In system management, the analysis of historical event data helps to discover some interesting patterns, which can provide great insights into system behavior. Specifically, a series of symptom events can be triggered by a computer system problem and they often serve as natural signatures for root cause analysis of system problems. As summarized in [Hellerstein et al. 2002b; Houck et al. 1995], "a problem can identify itself as a sequence of events that propagate from the origin and low layers to high software layers through the dependency tree". Therefore, discovering temporal patterns is useful for pinpointing the root causes and taking subsequent actions.

The pairwise temporal dependency among events has been given much attention for several reasons. First, the pairwise temporal dependency can be well visualized and easily interpreted by domain experts. Moreover, complex temporal dependencies can be constructed on the basis of pairwise temporal dependencies. Therefore, the t-pattern is proposed in [Li et al. 2005] as a pairwise temporal dependent pattern.

With respect to a given event sequence S, a **t-pattern** describes a statistical dependency between events, where the temporal dependency is characterized by the timing information indicating that one event is followed by another event within a time lag interval.

Mining frequent episodes from an event sequence typically can be done by predefining a fixed time window size [Mannila et al. 1995]. With the help of window size, items in the same sliding window are viewed as items in a single transaction. Then the idea of mining frequent itemsets from transaction data is applied for discovering frequent episodes. However, this method causes two issues which must be addressed in applications. First, the fixed-time-window scheme cannot investigate and utilize the temporal information within a window, and may not be able to discover temporal relationships longer than the given window size. For example, in real system management applications, the temporal distance between events may rang from 1 second to 1 day. Second, the common frequent pattern mining framework can not discover infrequent but significant patterns. For example, normal operations in many applications are frequent, but service disruptions are usually infrequent. However, obviously, it is significant.

To address the aforementioned issues, in [Li et al. 2005; Li and Ma 2004], a novel algorithm is proposed for discovering temporal patterns without pre-defined time windows. The temporal patterns discovery consists of two sub-tasks: 1) dependence testing and candidate removal using statistical techniques; and 2) identifying the temporal relationships between dependent event types. The core idea is formulating the dependence problem as a problem of comparing two probability distributions and solving it utilizing the statistical methods including the distance methods of the spatial point process and chi-squared tests. The statistical techniques are often robust against noises and also useful for characterizing the event patterns.

Herein, both **event sequence** and **point sequence** are described in Definition 3.2. Let $S =< e_1, ..., e_i, ..., e_n >$, where $e_i = (type_i, time_i)$. A point sequence as to event type A is denoted as $P_A =< a_1, a_2, ..., a_j, ..., a_m >$, where a_j is a timestamp and $a_i < a_{i+1}$. Assume the time range for a

point sequence P_A is $[0, T]$. Given a point z, we define the distance from z to the point sequence P_A as

$$d(z, P_A) = \inf_{x \in P_A \wedge x \geq z} ||x - z||. \quad (6)$$

Intuitively, the distance is defined to be the shortest distance between the point z and its closest neighbor in P_A.

Definition 3.10. Given two point sequences $P_A =< a_1, a_2, ..., a_j, ..., a_m >$ and $P_B =< b_1, b_2, ..., b_i, ..., b_n >$ for event A and event B, respectively, a **t-pattern** defined over P_A and P_B is denoted as $A \rightarrow_{[\tau-\delta,\tau+\delta]} B$, where τ is B's waiting period after the occurrence of A and δ is the time tolerance. It indicates that B is statistically dependent on A, and that most B's waiting periods after the occurrences of A fall into the interval $[\tau - \delta, \tau + \delta]$.

To qualify a t-pattern, a two-stage method is proposed in [Li et al. 2005]. At the first stage, the dependency between events is tested statistically. The task of the second stage is to identify the waiting periods between two possible dependent events. In order to test the dependency in t-pattern $A \rightarrow_{[\tau-\delta,\tau+\delta]} B$, two distributions are defined as follows.

Definition 3.11. The unconditional distribution of event B's waiting time is defined as

$$F_B(r) = P(d(x, P_B) \leq r),$$

where x and r are any real numbers. $F_B(r)$ describes the probability that event B occurs within time r.

Definition 3.12. The conditional distribution of event type B's waiting time w.r.t to event type A is defined as

$$F_{B|A}(r) = P(d(x, P_B) \leq r : x \in P_A),$$

where r is a real number and x is any point in the point sequence P_A. $F_{B|A}(r)$ describes the conditional probability distribution given that there is an event A occurred at time x.

With the help of the two distributions $F_B(r)$ and $F_{B|A}(r)$, the dependency between event A and event B is given in the following definition.

Definition 3.13. Given two point sequences P_A and P_B corresponding to event types A and B, respectively, $A \rightarrow B$, indicating that B is directly dependent on A, can be statistically true if $F_B(r)$ is significantly different from $F_{B|A}(r)$.

After stage one, two sub-tasks are involved in stage two: 1) dependence identification between the candidate pairs, and 2) waiting period discovery between the pair of dependent events. Assume δ to be the time tolerance considering factors such as lack of clock synchronization and phase shifts.

Definition 3.14. Given B depending on A, the waiting period of B after A is τ if the distance sequence $D_{B|A}$ has a period τ with time tolerance δ.

The waiting periods discovery is conducted utilizing the chi-squared test-based approach proposed in [Ma and Hellerstein 2001b]. Given an arbitrary element τ in $D_{B|A}$ and a fixed δ, let C_τ denote the total number of elements of $D_{B|A}$ occurred in $[\tau - \delta, \tau + \delta]$. Intuitively, C_τ should be small if τ is not a period; otherwise C_τ should be large. The main idea of determining whether or not τ is a period is to compare C_τ with the expected number of elements from a random sequence in $[\tau - \delta, \tau + \delta]$.

3.7. Frequent Episode

In event sequences, an episode is referred to as a collection of events occurring close to each other with respect to a given partial order in terms of timestamps [Mannila et al. 1997]. Typically, a window size is given to describe that all the events within one episode are close to each other. For example, as shown in Figure 5, there are six types of events (i.e.,A, B, C, D, E, F). The episode with window size 2 happens several times, where event E is followed by event F (e.g., $< e_1, f_1 >, < e_2, f_2 >, < e_3, f_3 >, < e_5, f_4 >$).

The episodes occurring with high frequencies (i.e., greater than a given threshold) are referred to as frequent episodes. One of the basic problems in event mining is to discover recurrent episodes from event sequences.

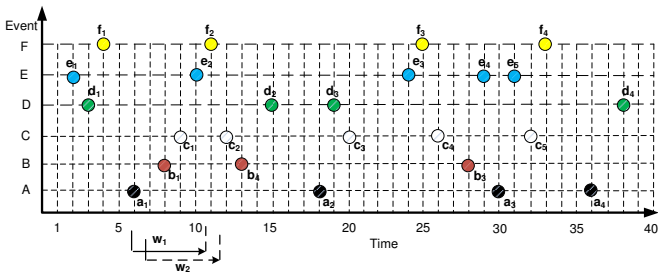

Fig. 5. Episodes of event sequences.

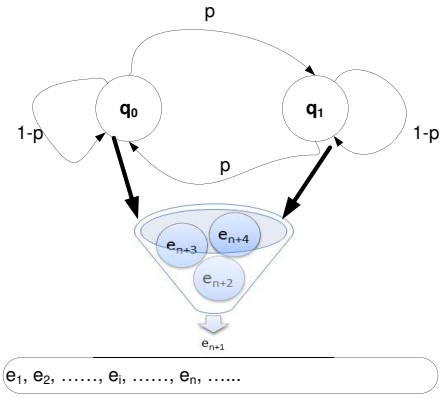

Fig. 6. Two states q_0 and q_1 correspond to the states with a low rate and a high rate, respectively. p denotes the probability of state change. The events generated in both states q_0 and q_1 are mixed into an event sequence.

Let T_S and T_E be the starting time and the ending time of event sequence S. Accordingly, $T_S = 0$ and $T_E = 40$, as shown in Figure 5.

In order to describe the episodes of event sequence S, a window w is defined as a slice of the event sequence S. In particular, w can be denoted as $w = (S_w, w_s, w_e)$, where S_w is the subsequence of S whose timestamps fall into $[w_s, w_e)$, and $w_s \geq T_s$ and $w_e \leq T_e$ represent the starting time and the ending time of the window w, respectively. Let W be the set of all possible windows on event sequence S. For instance, two time windows w_1 and w_2 are marked in Figure 5, where $w_1 = (< a_1, b_1, c_1, e_2 >, 6, 11)$ and $w_2 = (< b_1, c_1, e_2, f_2 >, 7, 12)$.

With the help of the definition of window, episodes can be described as directed acyclic graphs. According to the directed acyclic graphs, episodes are divided into three categories.

— A parallel episode is defined by a set of event types and a given window, where all the event types happen within the time window, without considering the order of them.
— A serial episode is defined by a set of event types and a given window as well. The window contains occurrences of all the event types, and the occurrences of them should keep in a consistent order.
— A composite episode is built recursively from events by serial and parallel composition. A composite episode is defined as: 1) an event, 2) the serial composition of two or more events, and 3) the parallel composition of two or more events.

Definition 3.15. Given the window size, let W be the set of all possible episodes on event sequence S. The frequency of an episode α is defined as $freq(\alpha) = \frac{|\{w \in W:\ \alpha\ occurs\ in\ w\}|}{|W|}$. If $freq(\alpha)$ is no less than the predefined minimum frequency threshold min_freq, then α is referred to as a frequent episode.

Based on the definition of frequent episode [Mannila et al. 1997], a useful lemma is described as follows.

LEMMA 3.16. *If an episode α is frequent in an event sequence S, then all subepisodes β of α are frequent as well.*

According to Lemma 3.16, similar to the Apriori algorithm, a level-wise searching algorithm can be applied to discover all the frequent episodes with respect to the event sequence S [Mannila et al. 1997]. Extended work on the frequent episode discovery in different scenarios can be found in [Laxman et al. 2004] and [Laxman et al. 2007] as well.

3.8. Event Burst

In this section, we focus on how to detect the event burst in an event sequence. For example, the arrival of a single event in a sequence is characterized by the rate at which the relevant events happen. Generally, an event burst is identified based on the high occurrence rate of the events. One intuitive way to model the random arrival time of events is based on the exponential distribution [Kleinberg 2003]. Let x be the inter-arrival time between event e_i and event e_j. Then x follows the exponential distribution with the following density function: $f(x) = \alpha e^{-\alpha x}$, where α^{-1} is the expected gap and α

is referred to as the rate of event arrivals. Accordingly, the event burst can be modeled by a relatively larger α.

To make the model clear, only two states q_0 and q_1 are given in Figure 6, which correspond to the states with a low rate α_0 and a high rate α_1 (i.e., $\alpha_0 \leq \alpha_1$), respectively. Intuitively, periods with a low rate are usually interleaved with periods with a high rate. Thus, let p denote the probability of a state changing from one to the other. $1-p$ is the probability of staying in the same state. The events happen in both states q_0 and q_1 are organized into an event sequence with respect to temporal information.

This model is also referred to as a two-state model in [Kleinberg 2003]. The two-state model can be used to generate the sequence of events. The beginning state is q_0, where events are emitted at a low rate and the inter-arrival gaps follow an exponential distribution according to the density function $f_0(x) = \alpha_0 e^{-\alpha_0 x}$. A state may change to another state with the probability p or stay put with the probability $1-p$. If the current state is q_1, then the inter-arrival gaps between events follow the distribution according to $f_1(x) = \alpha_1 e^{-\alpha_1 x}$.

Suppose that there is a given sequence of $n+1$ events, each of which is associated with its timestamp. A sequence of inter-arrivals $x = (x_1, x_2, ..., , x_n)$ can be determined by the given event sequence. According to the Bayesian theory, the possible state sequence $q = (q_{i_1}, q_{i_2}, ..., q_{i_n})$ can be inferred by maximizing the condition probability of the state sequence given the inter-arrival sequence. The conditional probability is shown as $P(q|x) = \frac{P(q)P(x|q)}{Z}$, where $Z = \sum_q P(q)P(x|q)$ is the normalizing constant. $P(x|q)$ can be computed as $P(x|q) = \prod_{t=1}^{n} f_{i_t}(x_t)$. Let b denote the number of state transitions in sequence q. Then $P(q)$ is

$$P(q) = (\prod_{i_t \neq i_{t+1}} p)(\prod_{i_t \neq i_{t+1}} (1-p)) = (\frac{p}{1-p})^b (1-p)^n. \quad (7)$$

Therefore, $P(q|x)$ is computed as $P(q|x) = \frac{1}{Z}(\frac{p}{1-p})^b (1-p)^n \prod_{t=1}^{n} f_{i_t}(x_t)$. Applying ln on both sides to maximize the likelihood above is equivalent to minimizing the cost function $c(q|x)$ in

$$c(q|x) = bln(\frac{1-p}{p}) + (\sum_{t=1}^{n} -ln(f_{i_t}(x_t))). \quad (8)$$

In order to minimize the cost described in the Eq.(8), the intuitive idea is motivated by its two terms. The first term on the right of Eq.(8) indicates that the sequences with a small number of state changes are preferred, while the second term shows that the sequences should conform well to the inter-arrival sequence. In [Kleinberg 2003], the two-state model is extended to an infinite-state model where there are infinite states and each state has different occurrence rates for events. Based on the infinite-state model, a hierarchical structure from the pattern of bursts can be extracted.

3.9. Rare Event

In system management applications, it is important to predict infrequent but highly correlated events, such as an attack on a computer network. However, there are several challenges in the prediction task. First, since the prediction targets are rare events, only a few subsequences of events are able to contribute to the prediction problem. Second, because of the categorical features of events, the uneven inter-arrival times are considered to be another difficulty. Moreover, because of noise the time recordings can only approximate the true arrival times.

Most prediction methods assume that the data has balanced class distributions. As a consequence, it is difficult to adopt traditional discriminative analysis methods to differentiate the target rare events from other frequent events. In [Vilalta and Ma 2002], a new strategy is proposed to improve the efficiency, accuracy and interpretability of rare event prediction. The main idea is to transform the rare event prediction problem into a search for all frequent event sets preceding target rare events. The unbalanced distribution problem is overcome by searching for patterns on the rare events exclusively. The patterns discovered are then combined into a rule-based model for prediction.

The idea of rare event prediction is illustrated in Figure 7. There are six event types. An event sequence $S =< e_1, b_1, d_1, a_1, f_1, ..., b_3, a_3, f_2 >$ is presented in the figure. Let D_{target} be the subset of event types to be predicted, e.g., $D_{target} = \{F\}$.

Definition 3.17. Given a set of event types Z and a window size w, if each event type in Z can be found in a window, then the window is matched by Z. The support of Z is $s\%$ if $s\%$ of windows with

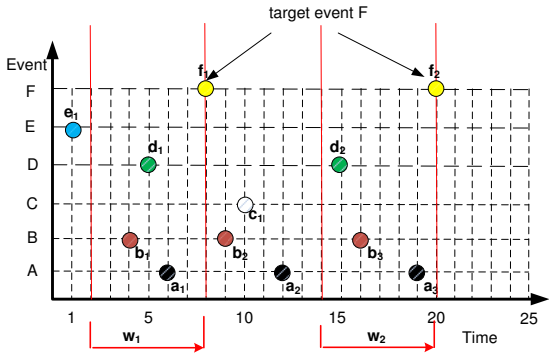

Fig. 7. A fixed window size 6 is given. Predicting rare event F is transformed to searching for the frequent patterns preceding event F. (See color insert.)

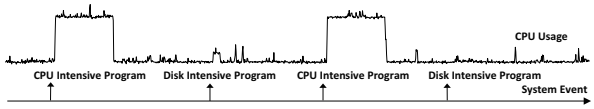

Fig. 8. The CPU usage is continuous time series data, while the system events are identified when starting different types of tasks such as disk intensive task and CPU intensive task.

size w preceding target events are matched by Z. Z is frequent if $s\%$ is above a predefined minimum threshold τ.

Definition 3.18. The subset of event types Z has confidence $c\%$ if $c\%$ of all windows of size w matched by Z preceding the target event. If the confidence of Z is greater than a predefined threshold, then Z is accurate.

It is very straightforward to mine the frequent Z. The general idea is to maintain in memory all events with a window of size w. All the events in a single window are considered to be a transaction. Thus the original problem is transformed to mining frequent patterns from a transaction database, where the *Apriori* [Agrawal et al. 1993] algorithm is applicable.

In order to mine the accurate Z, the basic idea is to count the number of times each of the frequent event sets occurs outside the time windows preceding target events, denoted by x_2. If the support number of Z is x_1, then the confidence is $\frac{x_1}{x_1+x_2}$. With the help of confidence, the accurate Z can be found. The rule-based model is built based on frequent and accurate event type sets. All the sets are ranked properly and the rule set is chosen from the event type sets [Vilalta and Ma 2002].

3.10. Correlated Pattern between Time Series and Event

Despite the importance of correlation analysis in various real applications such as system management and advertisement, limited research efforts have been reported in mining the correlations between two different types of temporal data, that is, the correlation between continuous time series data and temporal event data. Such types of correlation analysis are common, especially in system management. A typical example of system management is given in [Luo et al. 2014] to illustrate the importance of correlation between time series and events in real applications (as shown in Figure 8). There are two types of temporal data in the example. CPU Usage is continuous time series data describing the burden on the CPU. CPU Intensive Program and Disk Intensive Program are temporal events describing the running status of different programs. The CPU usage will increase dramatically when a CPU intensive program starts. However, the CPU usage does not suffer a significant burden due to the start of the disk intensive program. Therefore, it is considered that the CPU intensive program is highly correlated with CPU Usage, while there is no obvious correlation between the disk intensive program and CPU usage.

In [Luo et al. 2014], a novel approach is proposed to identify the correlation between two types of temporal data in three aspects: (a) determining the existence of correlation between the time series and events, (b) finding the time delay of the correlation, (c) identifying the monotonic effect describing whether the correlation is positive or negative. In order to clearly demonstrate the method to identify the correlation between two types of temporal data, three terms are defined, as shown in Figure 9. Given a time window size, the front sub-series is a snippet of time series just before the occurrence of an event and the length of the snippet is fixed with the time window size. Similarly, the rear sub-series is a snippet of time series after the occurrence of the event with the same length as the front sub-series. And a random sub-series is constructed by randomly sampling a snippet of time series with the same window size length. As a consequence, three sets of sub-series can be obtained, including front sub-series set F, rear sub-series set R, and random sub-series set Λ.

The intuitive idea is that, if there exists a correlation between a time series S and an event type E, then a corresponding change of the time series S occurs every time when an event E happens. Therefore, in [Luo et al. 2014] a hypothesis test is applied to verify the correlation statistically.

Given a time series S and an event sequence E, let F be the collection of front sub-series, R be the collection of rear sub-series, and Λ be the set of random sub-series. Several cases with respect to the correlation between E and S are listed as follows:

(1) they are correlated and E often occurs after S changes, denoted as $S \rightarrow E$, if and only if the distribution of sub-series in F is statistically different from the one in Λ.
(2) they are correlated and E often occurs before S changes, denoted as $E \rightarrow S$, if and only if the distribution of sub-series in R is statistically different from the one in Λ.
(3) they are correlated if $S \rightarrow E$ or $E \rightarrow S$.

As described above, the correlation analysis problem can be transformed into a multi-variate hypothesis test problem. A nearest-neighbor-based method is proposed in [Luo et al. 2014] to analyze the correlation between events and time series.

3.11. Pattern Summary

All the event patterns discussed in this chapter are summarized in Table 3.11. Admittedly, there are some other patterns we do not cover, such as spatial-temporal co-location patterns [Celik et al. 2006; Ang et al. 2012; Niebles et al. 2008].

Table III. Summary of mining event patterns

Pattern	Data	Output	Description
Sequential Pattern ([Agrawal and Srikant 1995],[Rao and Sammulal 2013],[Srikant and Agrawal 1996],[Irfan and Anoop 2012],[Tan et al. 2005], [Garofalakis et al. 1999],[Zaki 2001],[Ayres et al. 2002],[Han et al. 2000],[Pei et al. 2000], [Pei et al. 2001] [Mooney and Roddick 2013],[Chang 2011])	Event sequences	Frequent event subsequences, e.g., $< \{A\}, \{B, C\} >$.	All the subsequences with occurrence frequency not less than a given threshold are discovered. Two categories of algorithms are presented, i.e., Apriori-based and pattern-growth-based algorithms.
Fully Dependent Pattern([Liang et al. 2002])	An event database	All the items in a pattern are correlated with each other, e.g., {A,B,C} is a fully dependent pattern iff any of its subsets is a fully dependent pattern.	Hypothesis test is applied for identifying the correlation of items in a pattern.
Partially Periodic Dependent Pattern ([Ma and Hellerstein 2001b])	An event sequence	Periodic pattern with period p and tolerance δ, e.g., $A \rightarrow_{[p-\delta, p+\delta]} A$.	Periodic patterns are discovered from a given event sequence, where the periodic patterns happen on some segments of the sequence, rather than on the whole sequence. The partially periodic dependent pattern is identified by chi-squared hypothesis test.
Mutually Dependent Pattern([Ma and Hellerstein 2001a])	An event sequence	Events in a mutually dependent pattern $\{A, B\}$ depend on each other, i.e., $A \rightarrow B$ and $B \rightarrow A$.	Mutually dependent patterns are identified if the conditional probabilities in both directions are greater than a predefined minimum dependence threshold.
T-Pattern([Li et al. 2005; Li and Ma 2004])	An event sequence	Patterns like $A \rightarrow_{[\tau-\delta, \tau+\delta]} B$ are discovered, where τ is the time interval and δ is the tolerance.	T-Pattern is defined on two events, indicating that an event implies the other one within a time interval.
Frequent Episode ([Mannila et al. 1995; 1997; Achar et al. 2012; Achar et al. 2013; Patnaik et al. 2012])	An event sequence	Given window size p, an episode containing event pattern is frequent if its frequency is not less than a predefined threshold.	Three types of frequent episodes include the serial episode, the parallel episode, and the composite episode.
Event Burst([Kleinberg 2003; Srirangarajan et al. 2013; Yao et al. 2010; Nguyen et al. 2013])	An event sequence	Event burst is defined over a period $[t_1, t_2]$ if the occurrence frequency of a given event is high.	The event burst detection can be used for monitoring the occurrence of a significant event automatically.
Rare Event([Vilalta and Ma 2002])	An event sequence	Given a rare event T, a prediction rule is produced like $\{A, B\} \rightarrow E$.	An anomaly is typically a rare event. The prediction rule can be used to predict the anomaly according to historical events.
Correlation between Time Series and Event ([Luo et al. 2014])	An event sequence and a time series	Given a time series S and an event E, patterns like $S \rightarrow E$ or $E \rightarrow S$ are produced.	Such patterns are useful in practice, for example, the correlation between CPU usage and running a computing job.

4. MINING TIME LAGS

4.1. Introduction

As shown in Table IV, the time lag is one of the key features in many temporal patterns. Specifically, the time lag plays a significant role in identifying the evolving trends of incoming system events as well as predicting the future system behaviors. Time lag can provide the characterization of the

Table IV. Temporal patterns with time lag [Tang et al. 2012]

Temporal Pattern	An Example	Temporal Dependency with Lag Interval
Mutually dependent pattern [Ma and Hellerstein 2001a]	$\{A, B\}$	$A \rightarrow_{[0,\delta]} B, B \rightarrow_{[0,\delta]} A$
Partially periodic pattern [Ma and Hellerstein 2001b]	A with periodic p and a given time tolerance δ	$A \rightarrow_{[p-\delta,p+\delta]} A$
Frequent episode pattern [Mannila et al. 1997]	$A \rightarrow B \rightarrow C$ with a given time window p	$A \rightarrow_{[0,p]} B, B \rightarrow_{[0,p]} C$
Loose temporal pattern [Li and Ma 2004]	B follows by A before time t	$A \rightarrow_{[0,t]} B$
Stringent temporal pattern [Li and Ma 2004]	B follows by A about time t with a given time tolerance δ	$A \rightarrow_{[t-\delta,t+\delta]} B$

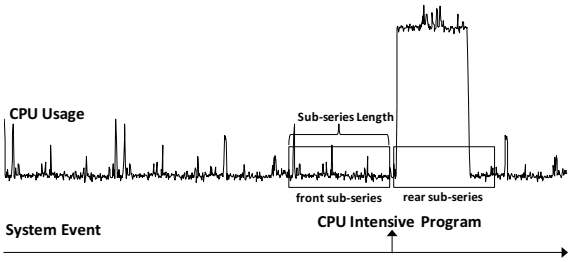

Fig. 9. The front sub-series is a snippet of time series with a fixed time window size before the occurrence of an event, while the rear sub-series is a snippet of time series with the same fixed time window size after the occurrence of the event.

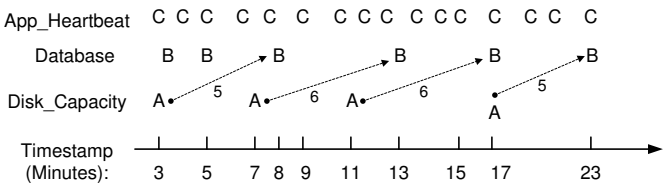

Fig. 10. Lag interval for temporal dependency.

temporal dependencies among events. It also provides temporal information for constructing a fault-error-failure chain and for performing root cause analysis [Avizienis et al. 2001]. In addition, given the appropriate time lag, events triggered by a single issue can be correlated. Correlating and merging those correlated events can help system administrators perform problem diagnosis and incident resolution. Thus, mining time lags of hidden temporal dependencies from sequential data is important in system management [Zeng et al. 2014b].

In real-world systems, the situation becomes complicated because of the inherent system complexity coupled with the limitations of data collection. However, the following assumption is typically used when analyzing the events generated from the monitoring systems: given two correlated events, their time lag is generally constant with small and neglectable fluctuations [Tang et al. 2012]. Although such an assumption is useful and applicable in many application scenarios, fluctuations can cause many problems and should be considered in time lag mining.

It is quite challenging to discover the hidden time lags between correlated and interleaved events if the randomness of the time lag is taken into consideration. First, given a large-scale collection of sequential events with fluctuating interleaved temporal dependencies, it is computationally infeasible to discover the time lag using exhaustive search. Second, the time lags hidden in the sequential data may oscillate with noises and fluctuations due to the synchronization issues and the limitations of data collection.

In summary, the aforementioned issues make the time lag mining problem quite challenging. In this section, both non-parametric methods and parametric methods are presented for discovering the time lag of temporal dependencies.

4.2. Non-Parametric Method

In previous studies on discovering temporal dependencies, interleaved dependencies were not explicitly considered [Li et al. 2005; Bouandas and Osmani 2007; Mannila et al. 1997]. For $A \rightarrow B$ where A and B are events, it is assumed that the dependency only exists between an item A and its first following B. However, it is possible that the dependency may exist between an item A with any following B. As shown in Figure 10, the time lag between two dependent events A and B is from 5 to 6 minutes, whereas the time lag between two adjacent A's about 4 minutes. Note that, for each event of type A', there exists a dependency between it and the second following B (not the first following B). Thus, among these dependent events A and B, the dependencies are interleaved. Given two event types, the numbers of timestamps for both event types and the number of possible time lags are $O(n)$ and $O(n^2)$, respectively. As a result, the number of lag intervals is $O(n^4)$. Therefore, the main challenge is how to find an efficiently way to identify appropriate lag intervals from the $O(n^4)$ candidates.

Let N denote the number of event types and n denote the number of distinct timestamps. An efficient algorithm with time complexity $O(n^2 logn)$ and space complexity $O(N)$ is proposed in [Tang et al. 2012].

4.2.1. Qualified Lag Interval. Let $S = x_1 x_2 ... x_N$ be an event sequence, where $i = 1, 2, ..., N$, x_i is the event type of the i^{th} event, and $t(x_i)$ is the timestamp of x_i. Intuitively, if a temporal dependency $A \rightarrow_{[t_1,t_2]} B$ exists in S, then a lot of A's are followed by some B with a time lag in $[t_1, t_2]$. Here, using $n_{[t_1,t_2]}$ to represent the observed number of A's.

A chi-square test based approach is proposed by Ma and Hellerstein [Ma and Hellerstein 2001b] to determine the minimum required n_r. Note that the independence degree is measured by the chi-square statistic which comparing the expected n_r with the observed n_r under the independence assumption. The chi-squared distribution with 1 degree of freedom approximates the null distribution of the statistic. Let χ_r^2 represent the chi-square statistic for n_r. A high χ_r^2 indicates that the high probability that observed n_r in the given sequence cannot be random. The chi-square statistic is defined as follows:

$$\chi_r^2 = \frac{(n_r - n_A P_r)^2}{n_A P_r (1 - P_r)}, \tag{9}$$

where n_A represents the number of event A in the event sequence S, and P_r represents the probability of a event B occurring in r from a random sequence. Accordingly, $n_A P_r$ represents the expected number of event A that can infer some event B within a time lag r. $n_A P_r(1 - P_r)$ represents the variance. We assume the given sequence S has the same sampling rate for B with the random sequence. The randomness is only for the timestamps of B items. Note that the Poisson process is usually used to simulate the random sequence. In Poisson process, the probability of an item appearing in an interval is proportional to the length of the interval [Ross 1996]. Hence, $P_r = |r| \cdot \frac{n_B}{T}$, where $|r|$ denotes the length of r, $|r| = t_2 - t_1 + w_B$, w_B denotes the minimum time lag of two adjacent B's, $w_B > 0$, and n_B denotes the number of B's in S. The absolute length of the lag interval r is $t_2 - t_1$. w_B is added to $|r|$ because without w_B when $t_1 = t_2$, $|r| = 0$, P_r is always 0 regardless of how large the n_B is. Consequently, χ_r^2 would be overestimated. Actually, the timestamps of items are discrete samples, and w_B is the observed sampling period of B. Therefore, the probability of a B occurring in $t_2 - t_1$ time units is equal to the one in $t_2 - t_1 + w_B$ time units.

A confidence level is used to quantify the value of χ_r^2. For instance, a 95% confidence level corresponds to $\chi_r^2 = 3.84$. According to Eq.(9), the observed n_r should satisfy $n_r > \sqrt{3.84 n_A P_r(1 - P_r)} + n_A P_r$. In our scenario, we only consider positive dependencies, thus $n_r - n_A P_r > 0$. We use ***support*** [Agrawal et al. 1993; Srikant and Agrawal 1996; Ma and Hellerstein 2001b] to guarantee a discovered temporal dependency fits the entire data sequence. Let $supp_A(r)$ (or $supp_B(r)$) be the support of $A \rightarrow_r B$, which represents the number of A's (or B's). The support satisfies $A \rightarrow_r B$ divided by the total number of items N. Let ***minsup*** be the minimum threshold for both $supp_A(r)$ and $supp_B(r)$ predefined by the user [Srikant and Agrawal 1996; Ma and Hellerstein 2001b]. Definition 4.1 gives the definition of the qualified lag interval which we want to discover.

Definition 4.1. Let A and B be the two item types contained in an item sequence S, χ_c^2 and $minsup$ be the two minimum thresholds predefined by the user, a lag interval $r = [t_1, t_2]$ is qualified if and only if $\chi_r^2 > \chi_c^2$, $supp_A(r) > minsup$, and $supp_B(r) > minsup$.

A straightforward algorithm (i.e., a ***brute-force*** algorithm) is developed for discovering all qualified lag intervals first. Then, ***STScan*** and ***STScan**** algorithms, which are much more efficient, are proposed in [Tang et al. 2012]. An algorithm, named STScan Algorithm, has been developed for avoiding scanning the data sequence multiple times. STScan Algorithm based on a sorted table which is a sorted linked list with a collection of sorted integer arrays. Each entry of the linked list is attached to two sorted integer arrays.

Given a sequence S, the sorted table is constructed as follows. At first, inserting each time lag between an A and a B into a red-black tree, which the key of this red-black tree node is the time lag, and the value is the pair of indices of A and B. After building the tree, the linked list of the sorted table is created by traversing the tree in ascending order. Assuming that the numbers of A and B in the sequence S are both $O(N)$, thus the number of $t(x_j) - t(x_i)$ is $O(N^2)$. Creating the red-black tree incurs time cost $O(N^2 \log N)$. The time cost of traversing tree is $O(N^2)$. Therefore, the overall time cost of the sorted table creation is $O(N^2 \log N)$. As mentioned in [Hernandez-Barrera 1996], for two variables X and Y, $O(N^2 \log N)$ is the known lower bound of sorting $X+Y$. Given the linked list with $O(N^2)$ entries

and $O(N)$ elements for each attached integer array, it has been shown that the actual space cost of a sorted table is $O(N^2)$, which is same as the red-black tree.

4.3. Parametric Method

In this subsection, a parametric method is reviewed to model the randomness of time lags which characterize the temporal dependencies between events [Zeng et al. 2014b]. We use an EM-based approach to infer the maximal likelihood model of time lags.

Problem Formulation. Given the event space $\mathbf{\Omega}$ of all possible events, an event sequence $\mathbf{S}$ is defined as ordered finite sequence $\mathbf{S} =< e_1, e_2, ..., e_i, ..., e_k >$. Each element $e_i \in \mathbf{S}$, an instance of an event, is a tuple $e_i = (E_i, t_i)$, where event $E_i \in \mathbf{\Omega}$ and t_i is a timestamp of one event occurrence.

Let A and B denote two types of events contained in the event space $\mathbf{\Omega}$. $\mathbf{S_A} =< (A, a_1), ..., (A, a_m) >$ is referred to as a subsequence from $\mathbf{S}$, where a_i represents the timestamp of i^{th} event A. $\mathbf{S_A}$ can be simplified as a sequence of timestamps, i.e., $\mathbf{S_A} =< a_1, ..., a_m >$ without explicitly giving event type. Similarly, $\mathbf{S_B}$ is denoted as $\mathbf{S_B} =< b_1, ..., b_n >$. The temporal dependency discovery between A and B can be attained by exploring the temporal relation between $\mathbf{S_A}$ and $\mathbf{S_B}$.

Specifically, if the j^{th} instance of event B can be inferred by the i^{th} instance of event A after a time lag $(\mu + \epsilon)$, then we have

$$b_j = a_i + \mu + \epsilon. \tag{10}$$

In Eq.(10), a_i and b_j are the timestamps of two instances of A and B, respectively. The true time lag μ describes the temporal relationship between A and B. The noise during data collection is represented as a random variable ϵ. Because of the existence of noise, the observed time lag between a_i and b_j is various. Accordingly, the lag $L = \mu + \epsilon$ is a random variable.

Definition 4.2. Let $A \rightarrow_L B$ denote the temporal dependency between A and B, where L is a random variable. The temporal dependency indicates the occurrence of B is inferred by the occurrence of A with a time lag L.

The key step of discovering the temporal dependency rule $A \rightarrow_L B$ is to learn the distribution of random variable L. The distribution of L is ruled by the parameter $\mathbf{\Theta}$, which is independent of the occurrence of A. The combination of the time lag L and the occurrence of A can infer the occurrence of an event B. Hence, the problem can be reduced to learning the parameter $\mathbf{\Theta}$ for the distribution of L. Given sequences S_A and S_B, one intuitive way to solve this problem is to learn the maximum likelihood parameter $\mathbf{\Theta}$. It can be formally expressed by

$$\hat{\mathbf{\Theta}} = \underset{\mathbf{\Theta}}{\operatorname{argmax}} P(\mathbf{\Theta} | \mathbf{S_A}, \mathbf{S_B}). \tag{11}$$

The problem can be solved by using iterative expectation maximization (i.e., EM-based method) [Bishop et al. 2006].

5. LOG EVENT SUMMARIZATION

Many systems, from computing systems, physical systems, business systems, to social systems, are only observable through the events they emit. It is well-known that event logs are reliable sources for people to understand the underlying dynamic system. To learn the behavior of a target dynamic system, one popular solution is to leverage the pattern mining technique to uncover the hidden temporal patterns from its logs.

In general, a huge number of events are generated from modern systems. These events describe the running status and activities of each component, including operational changes, security-related operations, and system failures, etc. As the size of event logs grow dramatically and the pattern mining technique tends to return all interesting patterns, the amount of mined patterns would be far beyond the processing capability of the human beings. Due to this fact, people gradually realized that it is wise to have a global overview of the observed system before conducting the detailed system analysis, instead of directly diving into the ocean of the patterns. To meet this need, some research efforts in the area of event mining have been shifted to event summarization in recent years.

5.1. What is Event Summarization?

Event summarization, as its name suggests, is a process to summarize the characteristics (mainly including temporal dynamics) of the events within the given system logs. From a functionality perspective, event summarization is a complementary technique rather than a substitute for event pattern mining. Compared with the patterns mined from off-the-shelf pattern mining techniques, summarized results are easier to be understood by the event analysts. The results of event summarization allow system analysts to obtain an overview of the system running status at a quick glance. According to the description of Kiernan and Terzi [Kiernan and Terzi 2008], a typical event summarization technique should have the following properties:

— *Brevity and accuracy*: The generated summary should be concise compared with the mined patterns obtained from the same piece of log. Moreover, it should also be able to precisely describe the status of the target system.
— *Hierarchical description*: The generated summary should be able to reflect the high level structure of the events. Besides the high level description, it should also be able to reveal information about local patterns.
— *Parameter free*: The parameters of the event summarization algorithm should be as few as possible. An ideal case is that no extra tuning is needed when event analysts are using the summarization solution.

5.2. Event Summarization vs. Event Pattern Mining

An obvious distinction between event summarization and event pattern mining is that event summarization is able to generate concise and summarized results compared with event pattern mining. In fact, besides the aforementioned characteristic, these two types of techniques are different in several other perspectives. Table V briefly summarizes their differences.

Table V. Distinction between event pattern mining and event summarization

	Event Pattern Mining	Event Summarization
Functionality	Detailed analysis	Exploration
Result Representation	Concrete patterns and rules	Concise representation
Result Granularity	High level	Low level

5.2.1. Functionality. From a functionality perspective, event summarization is mainly used for exploration and investigation, while event pattern mining is mainly suitable for detailed data analysis. In practice, the analysis goal is often unclear when the event log is obtained at first. Even experienced analysts don't know how to correctly analyze the events at the beginning. In this case, a global overview of the whole log is helpful and useful for analysts to quickly obtain the main idea of the system status. Event summarization is able to provide a global overview and make suggestions for further analysis. Based on the summarization results, analysts can have a good understanding of the overall status and can set up a good analysis plan. Moreover, as event pattern mining typically involves a lot of parameter tuning, the summarization results can also provide hints and clues on how to set the parameters.

5.2.2. Result Representation. Generally, as shown in Table 3.11, the results of event mining are represented by various types of patterns or rules. For example, frequent episode mining techniques [Mannila et al. 1997; Laxman et al. 2004; Yang et al. 2003] are able to discover the event subsequences that frequently appear in an event sequence. Given an event sequence denoted by $< (E_1, t_1), (E_2, t_2), ... >$, where E_i takes value from a finite event type set $\mathcal{E}$ and t_i denotes the timestamp when E_i occurs, the episode mining technique discovers and exhaustively lists all the frequent patterns in the form of $E_i \rightarrow E_j \rightarrow E_k$. Due to the large number of events and event types in modern computing systems, this technique can easily generate a large number of patterns or rules. Different variations [Das et al. 1998; Guralnik and Srivastava 1999; Höppner 2001] of the basic episodes mining algorithm can generate different sets of patterns or rules, but the basic forms of their patterns or rules are the same.

Different from patterns or rules generated by the event mining techniques, the results generated by event summarization are more concise. Moreover, different event summarization techniques can generate different representations, such as segmentation models [Kiernan and Terzi 2008; 2009; Pham et al. 2009], hidden Markov models [Peng et al. 2005; Wang et al. 2010], graphs [Aharon et al. 2009], and

event relationship networks [Peng et al. 2007; Jiang et al. 2011]. Analysts can freely choose different representations according to their concrete requirements.

5.2.3. Result Granularity. As mentioned in Section 5.2.1, event summarization acts as a complementary solution rather than a substitute for pattern mining. In terms of the result granularity, the results of event summarization are coarser than those provided by pattern mining algorithms. Event summarization algorithms generally only generate results that reflect the high level perspectives about the event relationships. For example, the summarized results often only tell which groups of events are correlated with each other and the change in temporal dynamics. On the other hand, the results of pattern mining usually contain more details, including the list of patterns or rules, the detailed event relationships, and the concrete parameters describing the relationships.

5.3. Event Summarization vs. Frequent Itemset Summarization

5.3.1. Introduction. Frequent itemset summarization is an extension of frequent itemset (pattern) mining. It is proposed to address some of the limitations of frequent itemset mining, including redundancy and interpretability. Traditional frequent itemset mining can generate a large number of frequent patterns and many of the generated patterns could be redundant. Once the number of discovered patterns is greater than hundreds, manual investigation becomes infeasible. More advanced patterns, such as *closed frequent patterns* [Aggarwal and Han 2014], *maximal frequent patterns* [Bifet and Gavaldà 2011; Guns et al. 2013], *top-k patterns* [Han et al. 2002; Salam and Khayal 2012; Wang et al. 2012], and *condensed patterns* [Pei et al. 2002], have been proposed to make the mining results more compact. However, these approaches can only partially solve the redundancy problem since the number of generated patterns can still be very large.

To effectively address the redundancy problem, researchers have developed pattern summarization methods to summarize the frequent patterns with more condensed formats. In general, two kinds of models have been proposed to summarize the frequent patterns: the *pattern profile* [Yan et al. 2005] and the *Markov random field (MRF)* [Wang and Parthasarathy 2006].

For *pattern-profile*-based summarization, the whole set of frequent patterns can be clustered into K groups of patterns and each group is described by a *pattern profile*. A *pattern profile* is essentially a generalization of a *closed frequent pattern*. It can be described as a triple $< \mathbf{p}, \phi, \rho >$, where $\mathbf{p}$ denotes the probability distribution vector learned from the dataset, ϕ denotes the master pattern used to represent a set of similar patterns in a group, and ρ denotes the support. To conduct the clustering, the *K-means* algorithm is used and the similarity between two frequent patterns (each frequent pattern is represented as a special kind of *pattern profile*) is measured based on the *Kullback-Leibler* divergence [Kullback and Leibler 1951] between their distribution vectors.

MRF-based summarization mainly focuses on using *non-derivable* frequent itemsets to construct the MRF as the summary of the whole dataset. The summarization is conducted using a level-wise approach. Generally, all 1-itemsets are used to build an MRF to infer the supports for all 2-itemsets. Then the 2-itemsets whose support cannot be inferred are used to update the MRF. The process continues until all the itemsets are checked. The resulting MRF is a concise summary of the original dataset.

5.3.2. Distinctions. According to the description of frequent itemset summarization, the distinctions between event summarization and frequent itemset summarization can be summarized as follows.

Summarization Data Objects: Although frequent itemset summarization and event summarization have similar tasks, they are working on different types of data objects. Frequent itemset summarization techniques focus on summarizing the transaction type data — the itemsets; while event summarization techniques focus on summarizing temporal datasets. Transaction type data usually do not have time information, or the time information is not critical or important.

Summarization Perspective: Frequent itemset summarization techniques pay more attention to presenting summaries that describe the frequent itemsets, while event summarization techniques pay more attention to generating summaries describing the temporal dynamics of the events. This is because the time information in an event log is a critical piece of information. Besides the pattern frequency, people who analyze the event log also pay close attention to the time information in the summary results.

5.4. Category of Event Summarization

Event summarization is a general type of solution to organize and represent the information extracted from the events. In recent years, several approaches for event summarization approaches have been developed. Although each solution is distinct from the others, they can be categorized into two types: *summarizing with frequency change* and *summarizing with temporal dynamics*.

5.4.1. ***Summarizing with Frequency Change.*** One major direction of event summarization is *to provide a summary of the given event log from the perspective of frequency change*. Generally, these methods leverage the segmentation model to provide a high level overview of the sequence by identifying global intervals on the whole event sequence. In each segment, the events are summarized by a local model, in a way similar to clustering. The local model is able to group the event types into a number of clusters, where the event types in the same cluster have similar frequency and vice versa.

To better illustrate the idea of this approach, Example 5.1 gives an example of the event sequence. In general, the event sequence can be denoted as S that records the occurrences of events during the time range $[1, n]$. Moreover, each event occurrence can be represented in the form (E, t), where E denotes the event type coming from a set $\mathcal{E} = \{E_1, \cdots, E_m\}$, and t denotes the time of the occurrence.

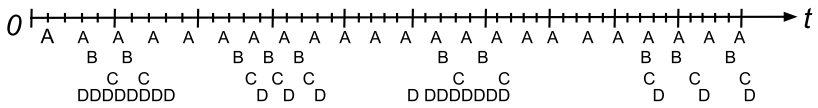

Fig. 11. An example event sequence.

Example 5.1. Figure 11 shows an example event sequence consisting of four event types, i.e., $\mathcal{E} = \{A, B, C, D\}$. The event sequence records the occurrences of all the event types from time 0 to time 40. It can be seen that some patterns might exist in this event sequence. For example, events B, C and D seem to appear sequentially.

One perspective of event summarization is to find an appropriate summary that balances conciseness and accuracy. To find an appropriate summary in such a way, Kiernan and Terzi proposed a method [Kiernan and Terzi 2008] which reduces the event summarization problem to an optimization problem. In particular, the proposed method solved this problem from the information theory perspective: *the best summary of an event sequence is the one with shortest description length quantified by the number of bits*. The *minimum description length (MDL)* principle [Barron et al. 1998; Grünwald et al. 2005; Grünwald 2007] is leveraged to conduct the model (summary) selection by finding a balance between summary coding length and description accuracy.

Concretely, event summarization in [Kiernan and Terzi 2008] is formulated as follows: Suppose there is an event sequence S with time range $[1, n]$, and let $\mathcal{E} = \{E_1, E_2, \cdots, E_m\}$. The goal of finding the best summary is to identify the best segmentation of the event sequence over $[1, n]$, as well as the best grouping of event types within each segment according to appearance frequency. Such a data description model is called the *segmental grouping* model.

Figure 12 illustrates how this method summarizes the event sequence given in Example 5.1. As shown in Figure 12, the whole sequence has been partitioned into four segments. Moreover, as shown in Figure 13, the event types are grouped within each segment according to their local frequency.

Other Solutions. Besides the aforementioned solution, several other algorithms are also proposed to summarize data from the point of frequency change. Wang et al. [Wang et al. 2010] extended the aforementioned work by adding more inter-segment information to reveal more detail about the system dynamics hidden in the event sequence.

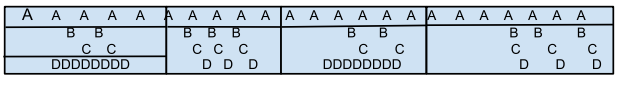

Fig. 12. Event summarization result produced by the solution of [Kiernan and Terzi 2008].

Fig. 13. A high level overview of summary.

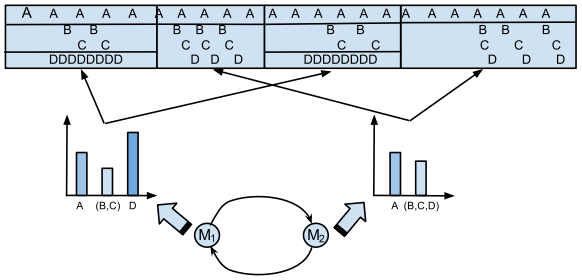

Fig. 14. An example of summarizing events with HMM.

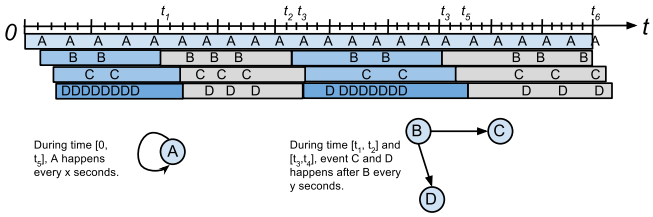

Fig. 15. An example for *natural event summarization* [Jiang et al. 2011].

To summarize the event sequence in this way, the following conjectures are made [Wang et al. 2010]: (1) a system should operate in different states; (2) the system should exhibit stable behavior in each state; and (3) the transitions between states should have certain regularity.

Based on their conjectures, HMM(*hidden Markov model*) is leveraged to model the state transitions of the system given its event sequence. Figure 14 illustrates how this work summarizes the event sequence in Example 5.1. Assume there are two states obtained from the example event sequence:

— State M_1, in which event D occurs the most frequently, events B and C occur the least frequently, and event A lies in the middle. The first and the third segments belong to this state.
— State M_2, in which event A occurs frequently while events B, C, and D occur less frequently. The second and the fourth segments belong to this state.

Clearly, this kind of summarization result is more understandable and more meaningful, as it reveals more interesting insights of the system. Similar to the previous work [Kiernan and Terzi 2008], the problem of finding the best summary that describes the state transition of the system is also formulated as an optimization problem, and the goal is to find the best segmentation as well as the ***HMM*** that describes the given event sequence with the least amount of information. In short, the amount of information to describe the event sequence is quantified as the number of bits used to describe the set of models $\mathcal{M}$ and the set of segments $\mathcal{I}$, i.e.,

$$Q^*(\mathcal{M}, \mathcal{I}) = C_d + C_s, \tag{12}$$

$$Q(\mathcal{M}^*, \mathcal{I}^*) = \underset{\mathcal{M},\mathcal{I}}{\operatorname{argmin}} Q(\mathcal{M}, \mathcal{I}). \tag{13}$$

In Eq.(12), each model in $\mathcal{M}$ consists a set of m probabilities ($M_i = (p_i(E_1), p_i(E_2), \cdots, p_i(E_m))$). Moreover, C_d denotes the number of bits needed to describe the event occurrences within all segments, and C_s denotes the number of bits to describe the segments. For the details of the encoding and the corresponding optimization algorithm, the interested reader can refer to [Wang et al. 2010].

5.4.2. ***Summarizing with Temporal Dynamics***. The state-of-the-art frequency-change-based event summarization solutions are able to reveal the temporal dynamics of the segments, but fail to provide information about the temporal dynamics among events. As the events are more natural components than the generated segments (by the summarization algorithms), it is more intuitive to provide an event-centric description in the summary results.

The frequency-change-based summarization approaches are able to generate a comprehensive summary from the input event sequence. However, this result as well as the frequency-change-based summarization algorithms have several limitations:

(1) The frequency-change-based approaches focus on generating summaries that only demonstrate the frequency changes of event types across adjacent segments. However, they often ignore the temporal information among event types within each segment. Consequently, the temporal dynamics of event patterns cannot be captured by these approaches.
(2) For all event types, the same number of event patterns are generated by these algorithms with the same boundaries. Considering different event types may have various underlying generating mechanisms, the same pattern boundary is unreasonable. Take a distributed system as an ex-

ample. In this distributed system, events may come from a large number of nodes that may be irrelevant to each other. Therefore, requiring a global segmentation would be inappropriate since many real patterns will be broken. Instead, for different event types, allowing the existence of different boundaries could help to identify the event pattern and acquire superior summaries.

(3) For system administrators, the above-generated summary is not easy to understand and take appropriate action. This is because system administrators may not have enough mathematical background to extract useful information from advanced mathematical models.

A new approach *natural event summarization (NES)*, proposed by Jiang et al. in [Jiang et al. 2011], has been used to address the aforementioned limitations. Utilizing inter-arrival histograms, this approach captures the temporal relationships among same-type and different-type events at first, subsequently, a set of disjoint histograms are used to summarize the input event sequence based on *MDL*. Finally, the resulting summary is represented as an *event relationship network*.

There are multiple advantages of this approach. First, different boundaries for different event types are allowed using inter-arrival histograms. Therefore, the summary could be more flexible. Second, the inter-arrival histograms helps to describe two main types of the event patterns: *correlation patterns* and *periodic patterns*. These two patterns are able to capture the majority of the temporal dynamics of event sequences and are useful for the summary generation. Moreover, the generated summaries can be used to derive lots of action rules almost directly. To better describe how *NES* works, Example 5.2 presents an illustrative example.

Example 5.2. Continuing Example 5.1, in which the event sequence contains four event types. Suppose the event types are A "an event created by an antivirus process," B "the firewall asks for the access privilege," C "a port was listed as an exception," and D "the firewall operation mode has been changed." The frequency-change-based event summarization approaches (e.g., [Wang et al. 2010]) segment the sequence into four segments (see Figure 14). Within each segment, this method groups the event types based on their occurrence frequency. Moreover, an *HMM* is leveraged to model the state transition between the segments.

Figure 15 shows the output summary generated by the *NES* method according to the example event sequence in Example 5.1. In this event sequence, the instances of event type C always appear after event type B during the whole time period. We can also observe that during time period $[t_1, t_2]$ and $[t_3, t_6]$, events with type D also appear after B. Hence, two correlation patterns, $B \rightarrow C$ and $B \rightarrow D$ (associated with the corresponding time periods), can be identified. From the example event sequence, we can also observe that the events with type A appear regularly throughout the whole time period. Therefore, all the events with type A can be summarized as only one pattern. Because event type A represents the antivirus monitoring process event, its stable period indicates that this process works normally.

5.5. Facilitating the Summarization Tasks

Besides the researchers who are working on proposing concrete event summarization methods, a lot of other researchers have also been focusing on proposing various other summarization methods [Peng et al. 2007; Schneider et al. 2010; Aharon et al. 2009; Tatti and Vreeken 2012]. Each of these approaches defines its own way of summarizing event sequences. A brief summary of these methods is illustrated in Table 5.5.

Apart from solving the summarization problem from the algorithmic perspective, some efforts [Expression 2015] have also been made toward providing various event summarization representations. In fact, there are many different approaches to conduct event summarization for different users with different purposes. It is inevitable that preprocessing the data and changing the program again and again, if an analyst wants to acquire an event summary from various perspectives. Clearly, the efficiency of this process is low.

Similar to online analytical processing (OLAP), which is an exploration process for transactional data, event summarization is also a trial-and-error process for temporal event data. Because there are repetitive exploration of the events from various perspectives in event summarization, *it is necessary to have an integrated framework to enable users to easily, interactively, and selectively extract, summarize, and analyze the temporal event data.*

Table VI. A brief summary of the event summarization methods

Paper	Category	Description
Peng, Perng & Li, 2007 [Peng et al. 2007]	Temporal Dynamics	Using a correlation graph ERN to summarize the correlation between events.
Kiernan & Terzi, 2008 [Kiernan and Terzi 2008]	Frequency Change	Using segmentation to summarize changes over time and using the event frequency group to summarize events within each time period.
Aharon et al., 2009 [Aharon et al. 2009]	Other	Clustering the events and using the clusters as the summary.
Kiernan & Terzi, 2009 [Kiernan and Terzi 2009]	Frequency Change	Similar to [Kiernan and Terzi 2008], but allowing mismatch among segments.
Wang et al., 2010 [Wang et al. 2010]	Frequency Change	Extension of [Kiernan and Terzi 2008]. Using the Markov model to represent the transition between segments.
Schneider et al., 2010 [Schneider et al. 2010]	Temporal Dynamics	Using a graph to represent the relations of *AlwaysFollowedBy*, *AlwaysPrecededBy*, and *NeverFollowedBy* among events.
Jiang, Perng & Li, 2011 [Jiang et al. 2011]	Temporal Dynamics	A richer form of [Peng et al. 2007]. Summarizing the events from the perspective of periodic patterns and correlation patterns.
Tatti & Vreeken, 2012 [Tatti and Vreeken 2012]	Temporal Dynamics	Summarizing the events using a set of serial episodes under the guidance of MDL.

To meet the above requirements, an extensible and flexible event summarization framework called *META*, proposed in [Jiang et al. 2014], is used to facilitate multi-resolution summarization and its associated tasks. The design principles of *META* include: 1) *META* should be flexible enough to fit various real-world scenarios, and 2) *META* should facilitate summarization task implementation as far as possible.

In general, *META* transforms all the event sequences into *summarization forest*, which is a specifically designed multi-resolution model. The event sequences and the necessary meta-data can be efficiently stored in this model. The *summarization forest* aims to store and represent the event sequence in multi-resolution views with a specified precision. On top of the *summarization forest*, a set of basic operations is proposed to express summarization tasks. Each basic operation can be viewed as an atomic operation that directly operates the data. At a higher level, five commonly used event summarization tasks are presented by using the basic operations. These tasks include ad hoc summarization, event storing, recovering, updating, and merging. By using these event summarization tasks, analysts can quickly conduct event summarization with little extra effort, and their efficiency can be significantly increased.

6. PROBLEM DIAGNOSIS IN SYSTEM MANAGEMENT

Performing a detailed diagnosis for a system issue mainly includes problem identification (i.e., identifying and detecting the problems), determination (i.e., fault diagnosis and root cause analysis), and resolution (i.e., providing resolutions). System diagnosis requires a deep understanding about the target system. In real-world IT infrastructures, many system issues are repeated and the associated resolutions can be found in the relevant events and tickets resolved in the past. In this section, we survey several data-driven applications for system diagnosis.

6.1. Introduction

The typical workflow for the IT service provider prescribed by the ITIL specification [url 2015b] involving problem detection, determination, and resolution is shown in Figure 16. In IT service management, incident management is one of the most important processes. The aim of it is to resolve the incident and efficiently restore the provision of services while based on human intervention or monitoring to detect the malfunction of a component. In terms of problem detection, the monitoring system runs on the servers, which computes metrics for the performances of hardware and software at regular intervals. The system then compares those metrics with acceptable thresholds called *monitoring situations*, and any violation would result in an alert. An event will be emitted by the monitoring if the alert persists beyond a predefined delay. Events coming from an IT environment are consolidated in an enterprise console, which analyzes the monitoring events as well as creates incident tickets in a ticketing system [Tang et al. 2013b]. The system administrators (sysAdmins) usually use the information contained in the tickets for problem determination and resolution. In the case of the provisioning of the services, the efficiency of these resources is crucial [Jiang et al. 2012]. The monitoring should minimize the number of generated *false positive alerts* since they will bring extra manpower costs in resolving *false positive tickets* created from those alerts [Branch et al. 2013; Diao et al. 2014]. Moreover, missed *false negative alerts* might bring severe system crashes so we should optimize the monitoring configuration to decline those alerts. In particular, Tang et al. [Tang et al. 2013b] presents several techniques for optimizing monitoring configuration by eliminating false positive alerts and false negative alerts.

The partial automation is usually used by a lot of IT service providers for incident diagnosis and resolution, with an intertwined operation of the sysAdmins and an automation script. Sometimes the sysAdmin has limited power to perform a known remediation script, but sometimes a complex root cause analysis can be completed by the sysAdmin [Zeng et al. 2014b; Fraenkel et al. 2004; Liu et al. 2016]. Removing the sysAdmin from the process entirely, if it is feasible, is helpful to decrease human error as well as accelerate restoration of service. The change from partially to fully automated problem remediation would promote service delivery to a new qualitative level where automation is an independent and complete process, and where it is not fragmented because of the requirement for adapting to human-driven processes. In this chapter, we mainly focus on reviewing four main types of data-driven applications from these historical tickets to efficiently improve the performance of system diagnosis using data mining techniques.

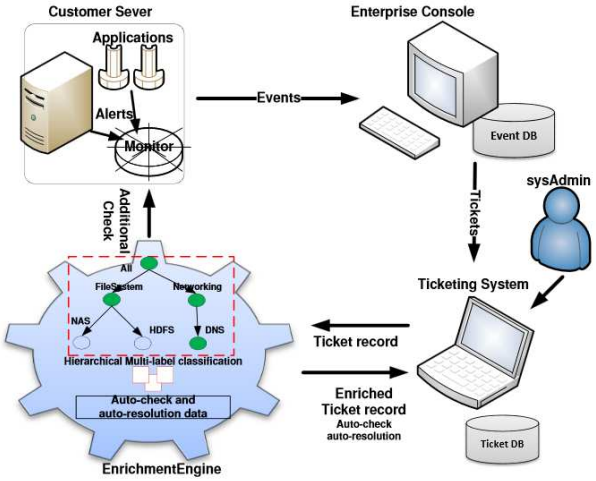

Fig. 16. Problem detection, determination, and resolution.

6.2. System Diagnosis Applications

6.2.1. **Ticket Classification:**. Because problems and incidents can occur at different levels of the hardware and software stack, one major activity to facilitate system diagnosis is to create leading indicators (a.k.a., signature or failure codes), which are used to incidents classification which is helpful for root cause analysis and failure trend monitoring [dos Santos et al. 2011]. Typical example incident categories include disk usage threshold exceeded, system down, application not available, printer not printing, and password reset. The incident categories are usually obtained by analyzing the incident records. The incident records, a.k.a., incident tickets, could store incident details like client name, platform, failure descriptions, severity code, resolution methods, and different timestamps. Many studies have been conducted in classifying IT management tickets. For example, Zeng et al. [Zeng et al. 2017a] proposed a hierarchical approach to classify maintenance request tickets for automated dispatch to the appropriate service delivery personnel. Diao et al. [Diao et al. 2009] proposed a rule-based crowdsourcing approach by combining classification rules with crowdsourcing (i.e., socialize and execute rules using social networking) for ticket classification.

6.2.2. **Ticket Resolution Recommendation**. Automatic techniques of recommending relevant historical tickets with resolutions can significantly improve the efficiency of root cause analysis and incident ticket resolving. Based on the relevant tickets, IT staff can correlate related system problems that happened before and perform a deeper system diagnosis. The solutions described in relevant historical tickets also provide best practices for solving similar issues. Recommendation technique has also been widely studied in e-commerce and online advertising areas. Existing recommendation algorithms can be categorized into two types. The first type is learning-based recommendation, in which the algorithm aims to maximize the rate of user response, such as the user clicks or conversations. The recommendation problem is then naturally formulated as a prediction problem. It utilizes a prediction algorithm to compute the probability of the user response on each item. Then, it recommends the one having the

largest probability. Most prediction algorithms can be utilized in the recommendation, such as naive Bayes, linear regression, logistic regression, matrix factorization, and multi-arm bandit. [Manning and Schuetze 1999; Bishop et al. 2006; Zeng et al. 2016]. The second type of recommendation algorithm focuses on the relevance of items or users, rather than the user response. Lots of algorithms proposed for promoting products to online users [Bell and Koren 2007; Ding and Li 2005; Koren 2009; Liu et al. 2011] belong to this type. They can be categorized as item-based [Sarwar et al. 2000; Karypis 2001; Ning and Karypis 2011] and user-based algorithms [Terveen and Hill 2001; Koren 2009; Bell and Koren 2007; Ding and Li 2005]. Tang et al. [Tang et al. 2013a] and Zhou et al. [Zhou et al. 2016] proposed item-based recommendation algorithms (where every incident ticket is regarded an item) to assist system administrators in resolving the incoming incident tickets that are generated by monitoring systems.

6.2.3. **Predicting System Behaviors**. Incident tickets have been used in many predictive and classification tasks in system diagnosis [Gupta et al. 2009; Di Lucca et al. 2002; Kadar et al. 2011; Bozman and Broderick 2010; Badaloo 2006]. For example, based on simple business rules, it often can be manually decided that when to modernize which elements of the server HW/SW (hardware/software) stack. Bogojeska et al. [Bogojeska et al. 2014; Bogojeska et al. 2013], however, alleviated this problem by supporting the decision process with an automated method. This automated method utilizes the incident tickets and server attributes and executes as follows. First, it identifies and ranks servers with problematic behaviors as candidates for modernization. Second, it uses the random forest classifier to evaluate the impact of multiple modernization actions and discover the most effective ones. Formally, let S represent the p-dimensional space where the p features are extracted from incident ticket data (such as tickets volumes and ticket severities) and server configuration information (such as server family, age, OS family and so on). Then one vector $x \in S$, used as an input for a predictive model M, can represents one server. Once trained on the available set of servers, M associates each x with a probability of being a problematic server, i.e., $M(x) \in [0, 1]$. Thus, the rank of all servers and identification of the problematic ones (e.g., those with $M(x) > 0.5$) can be completed using $M(x)$. Furthermore, the predictive model can evaluate the impact of various server modernization actions and discover the most effective ones. Assume $a : S \times P_a \mapsto S$ is an arbitrary parameterized improvement action. The action is represented as a function which associates an input vector x of a server and an action parameter $p \in P_a$ with a vector of the modified server features $\tilde{x} = a(x, p)$. Note that $\tilde{x}$ is obtained after performing such an improvement action. So $\tilde{x}$ represents x with new features relying on effect of the action. So the improvement of a parameterized action (a, p_a) can be measured by $I(a, p_a) = M(x) - M(\tilde{x})$, which means the difference between the prediction for the server before and after the modification. The actions that yield high improvements would be selected. Many other problems can also be addressed using historical incident tickets. Branch et al. [Branch et al. 2014] utilized ticket properties to predict service delivery efforts. Giurgiu et al. [Giurgiu et al. 2014] presented a comprehensive analysis of labor efforts and their impact factors to solve incident tickets in data centers according to ticket attribute value such as ticket severity and ticket failure code (a.k.a ticket class label).

6.2.4. **Mining Human Knowledge from Ticket**. Two types of tickets are generally exist in service management. They are monitoring ticket automatically generated from monitoring system and incident ticket created manually. Nevertheless, the resolutions attached to them are both written by operator to document the steps while troubleshooting a problem. Several recent studies [Potharaju et al. 2013; Shimpi et al. 2014; Potharaju et al. 2015; Jain and Potharaju 2016] have been carried out on investigating how to mining such valuable human knowledge for service management from tickets, although analyzing those tickets to do problem inference is extremely difficult since fixed fields are often inaccurate or incomplete, and wrote in the format of free-form text in natural language.

Rahul Potharaju et al. [Potharaju et al. 2013] took a practical step towards automatically analyzing natural language text in network tickets to mine the **problems**, troubleshooting **activities** and resolution **actions**. **Problems** denote the entity (e.g., router, database) and its associated state or symptoms (e.g., crash, reboot) as identified by operator; **Activities** indicate the steps conducted during troubleshooting such as clean cable, verify hard disk; **Actions** represent the resolution action(s) performed on the entity to resolve the problem. Rahul Potharaju et al. [Potharaju et al. 2013] first extract important phrases, especially domain-specific, such as “power supply unit” and “load balancer” using statistical NLP techniques. Further filters are applied to those phrases such as Phrase Length/Frequency Filter, Part-Of-Speech (PoS) Filter and Entropy Filter in which total number of phrases left decreased

to a reasonable amount for manual labeling. Second, these domain-specific phrases are then mapped onto an ontology model that formally represents the relationships between entities and stores them in a knowledge base in which each phrase is assigned a predefined class and relationships between those classes are also designated in advance. Given an unstructured text, we can get following output:

> The (load balancer) / **ReplaceableEntity** was (down) / **ProblemCondition**. We (checked) / **MaintenanceAction** the (cables) / **ReplaceableEntity**. This was due to a (faulty) / **ProblemCondition** (power supply unit) / **ReplaceableEntity** which was later (replaced) / **PhysicalAction**.

in which the **bold** words after slash are the predefined classes associated to phrases in brackets. Finally, heuristic class patterns existing in one sentence are given to define concepts **problems**, **activities** and **actions**. For instance, "the load balancer was down" matches the class pattern "[Replaceable Virtual Maintenance] Entity preceded / succeeded by ProblemCondition" which defines the concept **Problem**.

The aforementioned approach is quite effective in helping operators identify valuable knowledge [Potharaju et al. 2013], and can also be extended to build more powerful models such as Fault Tree Analysis [Ruijters and Stoelinga 2015] (FTA) and Event Relationship Network [Thoenen et al. 2001] (ERN). FTA is one of the most prominent techniques used by a wide range of industrial applications, in which Fault Tree (FT) is encoded as a graph that models how failures propagate through the system (i.e., how component failures lead to system failures) [Ruijters and Stoelinga 2015]. An ERN is a directed acyclic graph and describes the correlation between events triggered by different system components, and can be used to support automate problem determination [Thoenen et al. 2001]. In short, the knowledge mined from tickets is able to help effectively construct FTs and ERNs and facilitate efficient and systematic approaches for service management.

7. CONCLUSION

Modern IT infrastructures are constituted by large scale computing systems including various hardware and software components and often administered by IT service providers. Supporting such complex systems requires a huge amount of domain knowledge and experience. The manpower cost is one of the major costs for all IT service providers. Service providers often seek automatic or semi-automatic methodologies of detecting and resolving system issues to improve their service quality and efficiency. This survey provides several data-driven approaches for improving the quality and efficiency of IT service and system management. The improvements focus on several important components of the data-driven framework: event generation, preprocess, system monitoring, off-line analysis (temporal pattern discovery and summarization), and online analysis.

REFERENCES

2015a. FileZilla: An open-source and free FTP/SFTP solution. http://filezilla-project.org. (2015).

2015b. ITIL. http://www.itil-officialsite.com. (2015).

Avinash Achar, A Ibrahim, and PS Sastry. 2013. Pattern-growth based frequent serial episode discovery. *Data & Knowledge Engineering* 87 (2013), 91–108.

Avinash Achar, Srivatsan Laxman, and PS Sastry. 2012. A unified view of the apriori-based algorithms for frequent episode discovery. *Knowledge and Information Systems* 31, 2 (2012), 223–250.

Charu C Aggarwal and Jiawei Han. 2014. *Frequent Pattern Mining*. Springer.

Rakesh Agrawal, Tomasz Imieliński, and Arun Swami. 1993. Mining association rules between sets of items in large databases. In *ACM SIGMOD Record*, Vol. 22. ACM, 207–216.

Rakesh Agrawal and Ramakrishnan Srikant. 1995. Mining sequential patterns. In *Data Engineering, 1995. Proceedings of the Eleventh International Conference on*. IEEE, 3–14.

Michal Aharon, Gilad Barash, Ira Cohen, and Eli Mordechai. 2009. One Graph Is Worth a Thousand Logs: Uncovering Hidden Structures in Massive System Event Logs. In *ECML/PKDD*. 227–243.

Kai Keng Ang, Zheng Yang Chin, Haihong Zhang, and Cuntai Guan. 2012. Mutual information-based selection of optimal spatial–temporal patterns for single-trial EEG-based BCIs. *Pattern Recognition* 45, 6 (2012), 2137–2144.

Algirdas Avizienis, Jean-Claude Laprie, and Brian Randell. 2001. *Fundamental concepts of dependability*. University of Newcastle upon Tyne, Computing Science.

Jay Ayres, Jason Flannick, Johannes Gehrke, and Tomi Yiu. 2002. Sequential pattern mining using a bitmap representation. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 429–435.

Moonish Badaloo. 2006. An examination of server consolidation: trends that can drive efficiencies and help businesses gain a competitive edge. *White paper on IBM Global Services* (2006).

Andrew Barron, Jorma Rissanen, and Bin Yu. 1998. The minimum description length principle in coding and modeling. *Information Theory, IEEE Transactions on* 44, 6 (1998), 2743–2760.

Robert M. Bell and Yehuda Koren. 2007. Scalable Collaborative Filtering with Jointly Derived Neighborhood Interpolation Weights. In *Proceedings of IEEE International Conference on Data Mining (ICDM)*. 43–52.

Albert Bifet and Ricard Gavaldà. 2011. Mining frequent closed trees in evolving data streams. *Intelligent Data Analysis* 15, 1 (2011), 29–48.

Christopher M Bishop and others. 2006. *Pattern Recognition and Machine Learning*. Vol. 1. Springer New York.

Jasmina Bogojeska, Ioana Giurgiu, David Lanyi, George Stark, and Dorothea Wiesmann. 2014. Impact of HW and OS type and currency on server availability derived from problem ticket analysis. In *IEEE NOMS 2014)*, 1–9.

Jasmina Bogojeska, David Lanyi, Ioana Giurgiu, George Stark, and Dorothea Wiesmann. 2013. Classifying server behavior and predicting impact of modernization actions.. In *CNSM 2013*. 59–66.

Khellaf Bouandas and Aomar Osmani. 2007. Mining association rules in temporal sequences. In *Computational Intelligence and Data Mining, 2007. CIDM 2007. IEEE Symposium on*. IEEE, 610–615.

Jean S Bozman and Katherine Broderick. 2010. Server refresh: Meeting the changing needs of enterprise IT with hardware/software optimization. *IDC Whitepaper* (2010).

Joel W Branch, Yixin Diao, Emi K Olsson, Larisa Shwartz, and Li Zhang. 2013. Predicting service delivery costs under business changes. (Aug. 30 2013). US Patent App. 14/015,293.

Joel W. Branch, Yixin Diao, and Larisa Shwartz. 2014. A framework for predicting service delivery efforts using IT infrastructure-to-incident correlation. In *Network Operations and Management Symposium (NOMS)*. IEEE.

Mete Celik, Shashi Shekhar, James P Rogers, James A Shine, and Jin Soung Yoo. 2006. Mixed-Drove Spatio-Temporal Co-occurence Pattern Mining: A Summary of Results.. In *Proceedings of IEEE International Conference on Data Mining (ICDM)*, Vol. 6. 119–128.

Joong Hyuk Chang. 2011. Mining weighted sequential patterns in a sequence database with a time-interval weight. *Knowledge-Based Systems* 24, 1 (2011), 1–9.

Gianpaolo Cugola and Alessandro Margara. 2012. Processing flows of information: From data stream to complex event processing. *ACM Computing Surveys (CSUR)* 44, 3 (2012), 15.

Gautam Das, King-Ip Lin, Heikki Mannila, Gopal Renganathan, and Padhraic Smyth. 1998. Rule Discovery from Time Series.. In *KDD*, Vol. 98. 16–22.

Otávio M de Carvalho, Eduardo Roloff, and Philippe OA Navaux. 2013. A Survey of the State-of-the-art in Event Processing. *WSPPD13* (2013), 16.

Giuseppe A Di Lucca, Massimiliano Di Penta, and Sara Gradara. 2002. An approach to classify software maintenance requests. In *Proceedings of International Conference on Software Maintenance*. IEEE, 93–102.

Yixin Diao, Hani Jamjoom, and David Loewenstern. 2009. Rule-based problem classification in it service management. In *Cloud Computing, 2009. CLOUD'09. IEEE International Conference on*. IEEE, 221–228.

Yixin Diao, Linh Lam, Larisa Shwartz, and David Northcutt. 2014. Predicting service delivery cost for non-standard service level agreements. In *Network Operations and Management Symposium (NOMS), 2014 IEEE*. IEEE, 1–9.

Yi Ding and Xue Li. 2005. Time weight collaborative filtering. In *ACM CIKM*. 485–492.

Ricardo Luis dos Santos, Juliano Araujo Wickboldt, Roben Castagna Lunardi, Bruno Lopes Dalmazo, Lisandro Zambenedetti Granville, Luciano Paschoal Gaspary, Claudio Bartolini, and Marianne Hickey. 2011. A solution for identifying the root cause of problems in IT change management. In *2011 IEEE Symposium on Integrated Network Management (IM)*, 586–593.

Rick Durrett. 2010. *Probability: Theory and Examples*. Cambridge University Press.

Common Event Expression. 2015. Common Event Expression. (2015). http://cee.mitre.org.

Noam A Fraenkel, Guy Goldstein, Ido Sarig, and Refael Haddad. 2004. Root cause analysis of server system performance degradations. (May 18 2004). US Patent 6,738,933.

Lajos Jenő Fülöp, Gabriella Tóth, Róbert Rácz, János Pánczél, Tamás Gergely, Arpád Beszédes, and Lóránt Farkas. 2010. Survey on complex event processing and predictive analytics. In *Proceedings of the Fifth Balkan Conference in Informatics*. 26–31.

Jing Gao, Guofei Jiang, Haifeng Chen, and Jiawei Han. 2009. Modeling Probabilistic Measurement Correlations for Problem Determination in Large-Scale Distributed Systems. In *Proceedings of International Conference on Distributed Computing Systems (ICDCS)*. 623–630.

Minos N Garofalakis, Rajeev Rastogi, and Kyuseok Shim. 1999. SPIRIT: Sequential pattern mining with regular expression constraints. In *VLDB*, Vol. 99. 7–10.

Ioana Giurgiu, Jasmina Bogojeska, Sergii Nikolaiev, George Stark, and Dorothea Wiesmann. 2014. Analysis of Labor Efforts and their Impact Factors to Solve Server Incidents in Datacenters. In *Cluster, Cloud and Grid Computing (CCGrid), 2014 14th IEEE/ACM International Symposium on*. IEEE, 424–433.

Genady Grabarnik, Abdi Salahshour, Balan Subramanian, and Sheng Ma. 2004. Generic adapter logging toolkit. In *Proceedings of First IEEE International Conference on Autonomic Computing(ICAC-04)*. 308–309.

Peter D Grünwald. 2007. *The Minimum Description Length Principle*. The MIT Press.

Peter D Grünwald, In Jae Myung, and Mark A Pitt. 2005. *Advances in Minimum Description Length: Theory and Applications*. MIT press.

Tias Guns, Siegfried Nijssen, and Luc De Raedt. 2013. k-Pattern set mining under constraints. *Knowledge and Data Engineering, IEEE Transactions on* 25, 2 (2013), 402–418.

Rajeev Gupta, K Hima Prasad, Laura Luan, Daniela Rosu, and Christopher Ward. 2009. Multi-dimensional knowledge integration for efficient incident management in a services cloud. In *Services Computing, 2009. SCC'09. IEEE International Conference on*. IEEE, 57–64.

Valery Guralnik and Jaideep Srivastava. 1999. Event detection from time series data. In *Proceedings of the fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '99)*. 33–42.

Jiawei Han, Jian Pei, Behzad Mortazavi-Asl, Qiming Chen, Umeshwar Dayal, and Mei-Chun Hsu. 2000. FreeSpan: frequent pattern-projected sequential pattern mining. In *2000 ACM SIGKDD*. 355–359.

Jiawei Han, Jianyong Wang, Ying Lu, and Petre Tzvetkov. 2002. Mining top-k frequent closed patterns without minimum support. In *Proceedings. 2002 IEEE International Conference on Data Mining*. IEEE, 211–218.

Joseph L. Hellerstein, Sheng Ma, and Chang-Shing Perng. 2002a. Discovering Actionable Patterns in Event Data. *IBM Systems Journal* 43, 3 (2002), 475–493.

Joseph L. Hellerstein, Sheng Ma, and C-S Perng. 2002b. Discovering actionable patterns in event data. *IBM Systems Journal* 41, 3 (2002), 475–493.

Antonio Hernandez-Barrera. 1996. Finding an $o(n^2 \log n)$ Algorithm Is Sometimes Hard. In *Proceedings of the 8th Canadian Conference on Computational Geometry*. 289–294.

Frank Höppner. 2001. Discovery of temporal patterns. In *Principles of Data Mining and Knowledge Discovery*, 192–203.

Paul Horn. 2001. Automatic Computing: IBM's Prospective on the State of Information Technology. http://www.research.ibm.com/autonomic. (2001). IBM Corporation.

K Houck, S Calo, and A Finkel. 1995. Towards a practical alarm correlation system. In *Integrated Network Management IV*. Springer, 226–237.

Khan Irfan and Jain Anoop. 2012. A Comprehensive Survey on Sequential Pattern Mining. *International Journal of Engineering Research and Technology* (2012).

Navendu Jain and Rahul Potharaju. 2016. Problem inference from support tickets. (Jan. 5 2016). US Patent 9,229,800.

Yexi Jiang, Chang-Shing Perng, and Tao Li. 2011. Natural event summarization. In *Proceedings of the 20th ACM International Conference on Information and Knowledge Management (CIKM '11)*. 765–774.

Yexi Jiang, Chang-Shing Perng, and Tao Li. 2014. META: Multi-resolution Framework for Event Summarization. In *SIAM International Conference on Data Mining*.

Yexi Jiang, Chang-Shing Perng, Tao Li, and Rong Chang. 2012. Intelligent cloud capacity management. In *Network Operations and Management Symposium (NOMS), 2012 IEEE*. IEEE, 502–505.

Cristina Kadar, Dorothea Wiesmann, Jose Iria, Dirk Husemann, and Mario Lucic. 2011. Automatic classification of change requests for improved it service quality. In *SRII Global Conference (SRII), 2011 Annual*. IEEE, 430–439.

George Karypis. 2001. Evaluation of Item-Based Top-N Recommendation Algorithms. In *Proceedings of ACM International Conference on Information and Knowledge Management*. 247–254.

Jeffrey O. Kephart and David M. Chess. 2003. The Vision of Autonomic Computing. *Computer* (2003), 41–50.

Jerry Kiernan and Evimaria Terzi. 2008. Constructing Comprehensive Summaries of Large Event Sequences. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '08)*. 417–425.

Jerry Kiernan and Evimaria Terzi. 2009. Constructing comprehensive summaries of large event sequences. *ACM Trans. Knowl. Discov. Data* 3, 4 (Dec. 2009), 21:1–21:31.

Jon Kleinberg. 2003. Bursty and hierarchical structure in streams. *Data Mining and Knowledge Discovery* 7, 4 (2003), 373–397.

Yehuda Koren. 2009. Collaborative filtering with temporal dynamics. In *KDD*. 447–456.

Solomon Kullback and Richard A Leibler. 1951. On information and sufficiency. *The Annals of Mathematical Statistics* (1951), 79–86.

Srivatsan Laxman, PS Sastry, and KP Unnikrishnan. 2007. A fast algorithm for finding frequent episodes in event streams. In *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 410–419.

Srivatsan Laxman, P Shanti Sastry, and KP Unnikrishnan. 2004. Fast algorithms for frequent episode discovery in event sequences. In *Proc. 3rd Workshop on Mining Temporal and Sequential Data*.

Tao Li (Ed.). 2015. *Event Mining: Algorithms and Applications*. CRC Press.

Tao Li, Feng Liang, Sheng Ma, and Wei Peng. 2005. An integrated framework on mining logs files for computing system management. In *ACM SIGKDD International Conference on Knowledge Discovery in Data Mining*. 776–781.

Tao Li and Sheng Ma. 2004. Mining temporal patterns without predefined time windows. In *Data Mining, 2004. ICDM'04. Fourth IEEE International Conference on*. IEEE, 451–454.

Tao Li and Wei Peng. 2005. A Clustering Model Based on Matrix Approximation with Applications to Cluster System Log Files. In *Proceedings of the 16th European Conference on Machine Learning*. 625–632.

Tao Li, Wei Peng, Charles Perng, Sheng Ma, and Haixun Wang. 2010. An Integrated Data-Driven Framework for Computing System Management. *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans* 40, 1 (Jan. 2010), 90–99.

Feng Liang, Sheng Ma, and Joseph L Hellerstein. 2002. Discovering Fully Dependent Patterns.. In *Proceedings of SIAM International Conference on Data Mining (SDM)*. SIAM, 511–527.

Liwei Liu, Nikolay Mehandjiev, and Dong-Ling Xu. 2011. Multi-criteria service recommendation based on user criteria preferences. In *Proceedings of the Fifth ACM Conference on Recommender Systems*. 77–84.

Zheng Liu, Tao Li, and Junchang Wang. 2016. A Survey on Event Mining for ICT Network Infrastructure Management. *ZTE Communications* 14, 2, 47–55.

Chen Luo, Jian-Guang Lou, Qingwei Lin, Qiang Fu, Rui Ding, Dongmei Zhang, and Zhe Wang. 2014. Correlating events with time series for incident diagnosis. In *2014 ACM SIGKDD*, 1583–1592.

Sheng Ma and Joseph L Hellerstein. 2001a. Mining mutually dependent patterns. In *Proceedings IEEE International Conference on Data Mining(ICDM)*. IEEE, 409–416.

Sheng Ma and Joseph L Hellerstein. 2001b. Mining partially periodic event patterns with unknown periods. In *Data Engineering, 2001. Proceedings. 17th International Conference on*. IEEE, 205–214.

Adetokunbo Makanju, A. Nur Zincir-Heywood, and Evangelos E. Milios. 2009. Clustering Event logs Using Iterative Partitioning. In *Proceedings of ACM KDD*. Paris, France, 1255–1264.

Heikki Mannila, Hannu Toivonen, and A Inkeri Verkamo. 1995. Discovering frequent episodes in sequences Extended abstract. In *1st Conference on Knowledge Discovery and Data Mining, Montreal, CA*.

Heikki Mannila, Hannu Toivonen, and A Inkeri Verkamo. 1997. Discovery of frequent episodes in event sequences. *Data Mining and Knowledge Discovery* 1, 3 (1997), 259–289.

Christopher D. Manning and Hinrich Schuetze. 1999. *Foundations of Statistical Natural Language Processing*. MIT Press.

Carl H Mooney and John F Roddick. 2013. Sequential pattern mining–approaches and algorithms. *ACM Computing Surveys (CSUR)* 45, 2 (2013), 19.

Thin Nguyen, Dinh Phung, Brett Adams, and Svetha Venkatesh. 2013. Event extraction using behaviors of sentiment signals and burst structure in social media. *Knowledge and information systems* 37, 2 (2013), 279–304.

Juan Carlos Niebles, Hongcheng Wang, and Li Fei-Fei. 2008. Unsupervised learning of human action categories using spatial-temporal words. *International journal of computer vision* 79, 3 (2008), 299–318.

Xia Ning and George Karypis. 2011. SLIM: Sparse Linear Methods for Top-N Recommender Systems. In *ICDM*. 497–506.

Adam J. Oliner, Alex Aiken, and Jon Stearley. 2008. Alert Detection in System Logs. In *Proceedings of IEEE International Conference on Data Mining(ICDM)*. 959–964.

Thomas J Owens. 2007. *Survey of event processing*. Technical Report. DTIC Document.

Debprakash Patnaik, Srivatsan Laxman, Badrish Chandramouli, and Naren Ramakrishnan. 2012. Efficient Episode Mining of Dynamic Event Streams.. In *Proceedings of IEEE International Conference on Data Mining (ICDM)*. 605–614.

Jian Pei, Guozhu Dong, Wei Zou, and Jiawei Han. 2002. On computing condensed frequent pattern bases. In *Proceedings of 2002 IEEE International Conference on Data Mining*. IEEE, 378–385.

Jian Pei, Jiawei Han, Behzad Mortazavi-Asl, Helen Pinto, Qiming Chen, Umeshwar Dayal, and Mei-Chun Hsu. 2001. Prefixspan: Mining sequential patterns efficiently by prefix-projected pattern growth. In *2013 IEEE 29th International Conference on Data Engineering (ICDE)*. IEEE Computer Society, 0215–0215.

Jian Pei, Jiawei Han, Behzad Mortazavi-Asl, and Hua Zhu. 2000. Mining access patterns efficiently from web logs. In *Knowledge Discovery and Data Mining: Current Issues and New Applications*. Springer, 396–407.

Wei Peng, Tao Li, and Sheng Ma. 2005. Mining logs files for data-driven system management. *SIGKDD Explor. Newsl.* 7, 1 (June 2005), 44–51.

Wei Peng, Charles Perng, Tao Li, and Haixun Wang. 2007. Event summarization for system management. In *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*. 1028–1032.

Quang-Khai Pham, Guillaume Raschia, Noureddine Mouaddib, Regis Saint-Paul, and Boualem Benatallah. 2009. Time sequence summarization to scale up chronology-dependent applications. In *Proceedings of the 18th ACM Conference on Information and Knowledge Management (CIKM '09)*. 1137–1146.

Rahul Potharaju, Joseph Chan, Luhui Hu, Cristina Nita-Rotaru, Mingshi Wang, Liyuan Zhang, and Navendu Jain. 2015. ConfSeer: leveraging customer support knowledge bases for automated misconfiguration detection. *Proceedings of the VLDB Endowment* 8, 12 (2015), 1828–1839.

Rahul Potharaju, Navendu Jain, and Cristina Nita-Rotaru. 2013. Juggling the jigsaw: Towards automated problem inference from network trouble tickets. In *Presented as part of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*. 127–141.

V. Chandra Shekhar Rao and P. Sammulal. 2013. Article: Survey on Sequential Pattern Mining Algorithms. *International Journal of Computer Applications* 76, 12 (August 2013), 24–31. Full text available.

IBM Market Research. 2003. Autonomic Computing Core Technology Study. (2003).

Sheldon M Ross. 1996. *Stochastic Processes*. Vol. 2. John Wiley & Sons New York.

Enno Ruijters and Mariëlle Stoelinga. 2015. Fault tree analysis: A survey of the state-of-the-art in modeling, analysis and tools. *Computer science review* 15 (2015), 29–62.

Abdus Salam and M Sikandar Hayat Khayal. 2012. Mining top-k frequent patterns without minimum support threshold. *Knowledge and Information Systems* 30, 1 (2012), 57–86.

Gerard Salton and Michael McGill. 1984. *Introduction to Modern Information Retrieval*. McGraw-Hill.

Badrul M. Sarwar, George Karypis, Joseph A. Konstan, and John T. Riedl. 2000. Application of Dimensionality Reduction in Recommender System – A Case Study. In *ACM WebKDD Workshop*.

Sigurd Schneider, Ivan Beschastnikh, Slava Chernyak, Michael D Ernst, and Yuriy Brun. 2010. Synoptic: summarizing system logs with refinement. *Workshop Proceedings on Managing Large-Scale Systems via the Analysis of System Logs and the Application of Machine Learning Techniques (SLAML)* (2010).

Vikrant Shimpi, Maitreya Natu, Vaishali Sadaphal, and Vaishali Kulkarni. 2014. Problem identification by mining trouble tickets. In *Proceedings of the 20th International Conference on Management of Data*. Computer Society of India, 76–86.

Ramakrishnan Srikant and Rakesh Agrawal. 1996. Mining quantitative association rules in large relational tables. *ACM SIGMOD Record* 25, 2 (1996), 1–12.

Seshan Srirangarajan, Michael Allen, Ami Preis, Mudasser Iqbal, Hock Beng Lim, and Andrew J Whittle. 2013. Wavelet-based burst event detection and localization in water distribution systems. *Journal of Signal Processing Systems* 72, 1 (2013), 1–16.

John Stearley. 2004. Towards informatic analysis of Syslogs. In *Proceedings of IEEE International Conference on Cluster Computing*. San Diego, California, USA, 309–318.

Pang-Ning Tan, Michael Steinbach, and Vipin Kumar. 2005. *Introduction to Data Mining*. Addison Wesley.

Liang Tang and Tao Li. 2010. LogTree: A Framework for Generating System Events from Raw Textual Logs. In *Proceedings of IEEE International Conference on Data Mining (ICDM)*. 491–500.

Liang Tang, Tao Li, and Chang-Shing Perng. 2011. LogSig: Generating System Events from Raw Textual Logs. In *Proceedings of ACM International Conference on Information and Knowledge Management*. 785–794.

Liang Tang, Tao Li, and Larisa Shwartz. 2012. Discovering lag intervals for temporal dependencies. In *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 633–641.

Liang Tang, Tao Li, Larisa Shwartz, and Genady Grabarnik. 2013a. Recommending Resolutions for Problems Identified by Monitoring. In *Proceedings of IEEE/IFIP International Symposium on Integrated Network Management*. 134–142.

Liang Tang, Tao Li, Larisa Shwartz, Florian Pinel, and Genady Ya Grabarnik. 2013b. An integrated framework for optimizing automatic monitoring systems in large IT infrastructures. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 1249–1257.

Nikolaj Tatti and Jilles Vreeken. 2012. The long and the short of it: summarising event sequences with serial episodes. In *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 462–470.

Loren Terveen and Will Hill. 2001. Beyond Recommender Systems: Helping People Help Each Other. In *HCI in the New Millennium*. 487–509.

David Thoenen, Jim Riosa, and Joseph L Hellerstein. 2001. Event relationship networks: a framework for action oriented analysis in event management. In *Integrated Network Management Proceedings, 2001 IEEE/IFIP International Symposium on*. IEEE, 593–606.

Brad Topol, David Ogle, Donna Pierson, Jim Thoensen, John Sweitzer, Marie Chow, Mary Ann Hoffmann, Pamela Durham, Ric Telford, Sulabha Sheth, and Thomas Studwell. 2003. Automating problem determination: A first step toward self-healing computing systems. IBM White Paper. (October 2003).

Ricardo Vilalta and Sheng Ma. 2002. Predicting rare events in temporal domains. In *Data Mining, 2002. ICDM 2003. Proceedings. 2002 IEEE International Conference on*. IEEE, 474–481.

Chao Wang and Srinivasan Parthasarathy. 2006. Summarizing itemset patterns using probabilistic models. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 730–735.

Peng Wang, Haixun Wang, Majin Liu, and Wei Wang. 2010. An algorithmic approach to event summarization. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data (SIGMOD '10)*. 183–194.

Peilong Wang, Xuan Yang, Yuanyuan Zhang, and Jiadong Ren. 2012. An Algorithm Based on Temporary Table for Mining Top-k Closed Frequent Patterns in Data Streams. *International Journal of Digital Content Technology & its Applications* 6, 20 (2012).

Chris Wrench, Frederic Stahl, Giuseppe Di Fatta, Vidhyalakshmi Karthikeyan, and Detlef D Nauck. 2016. Data Stream Mining of Event and Complex Event Streams: A Survey of Existing and Future. *Enterprise Big Data Engineering, Analytics, and Management* (2016), 24.

Wei Xu, Ling Huang, Armando Fox, David A. Patterson, and Michael I. Jordan. 2008. Mining Console Logs for Large-Scale System Problem Detection. In *SysML*.

Xifeng Yan, Hong Cheng, Jiawei Han, and Dong Xin. 2005. Summarizing itemset patterns: a profile-based approach. In *Proceedings of the eleventh ACM SIGKDD International Conference on Knowledge Discovery in Data Mining*. ACM, 314–323.

Jiong Yang, Wei Wang, and Philip S. Yu. 2003. Mining Asynchronous Periodic Patterns in Time Series Data. *IEEE Trans. on Knowl. and Data Eng.* 15, 3 (March 2003), 613–628.

Junjie Yao, Bin Cui, Yuxin Huang, and Xin Jin. 2010. Temporal and Social Context Based Burst Detection from Folksonomies.. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence (AAAI)*. 21–27.

Mohammed J Zaki. 2001. SPADE: An efficient algorithm for mining frequent sequences. *Machine learning* 42, 1-2 (2001), 31–60.

Chunqiu Zeng, Tao Li, Larisa Shwartz, and Genady Ya Grabarnik. 2014a. Knowledge Guided Hierarchical Multi-Label Classification over Ticket Data. *IEEE Trans. Network and Service Management*, 2017, in press.

Chunqiu Zeng, Liang Tang, Tao Li, Larisa Shwartz, and Genady Ya. Grabarnik. 2017b. An Integrated Framework for Mining Temporal Logs from Fluctuating Events. *IEEE Trans. Services Computing*, 2017, in press.

Chunqiu Zeng, Qing Wang, Shekoofeh Mokhtari, and Tao Li. 2016. Online Context-Aware Recommendation with Time Varying Multi-Armed Bandit. In *2016 ACM SIGKDD*. 2025–2034.

Wubai Zhou, Liang Tang, Chunqiu Zeng, Tao Li, Larisa Shwartz, and Genady Ya. Grabarnik. 2016. Resolution Recommendation for Event Tickets in Service Management. *IEEE Trans. Network and Service Management* 13, 4 (2016), 954–967.