Tools and techniques for event log analysis

Risto Vaarandi

Faculty of Information Technology
Department of Computer Engineering
Chair of System Programming
TALLINN UNIVERSITY OF TECHNOLOGY

Supervisor:

Prof. Ahto Kalja
Department of Computer Engineering, Tallinn University of Technology

Opponents:

Dr. Gabriel Jakobson
Altusys Corporation, USA

Dr. Jaak Vilo
Institute of Computer Science, University of Tartu, Estonia

Dr. Aleksander Reitsakas
Cell Network, Estonia

Declaration

Hereby I declare that this doctoral thesis, my original investigation and achievement, submitted for the doctoral degree at Tallinn University of Technology has not been submitted before for any degree or examination at any other university.

Table of contents

1. Risto Vaarandi. 2002. Platform Independent Tool for Local Event Correlation. Acta Cybernetica 15(4), pp. 705-723.
2. Risto Vaarandi. 2002. Platform Independent Event Correlation Tool for Network Management. Proceedings of the 8th IEEE/IFIP Network Operations and Management Symposium, pp. 907-910.
3. Risto Vaarandi. 2002. SEC – a Lightweight Event Correlation Tool. Proceedings of the 2002 IEEE Workshop on IP Operations and Management, pp. 111-115.
4. Risto Vaarandi. 2003. A Data Clustering Algorithm for Mining Patterns From Event Logs. Proceedings of the 2003 IEEE Workshop on IP Operations and Management, pp. 119-126.
5. Risto Vaarandi. 2004. A Breadth-First Algorithm for Mining Frequent Patterns from Event Logs. Proceedings of the 2004 IFIP International Conference on Intelligence in Communication Systems, LNCS Vol. 3283, pp. 293-308.

Abstract

This thesis discusses the problems of event correlation and data mining in the context of event log analysis, and presents novel tools and techniques for addressing these problems. Event logs play an important role in modern IT systems, since they are an excellent source of information for monitoring the system in real-time and for conducting retrospective event analysis.

Event correlation is one of the most prominent event processing techniques today. It has received a lot of attention in the context of network fault management over the past decade, and is becoming increasingly important in other domains as well, including event log analysis. A number of approaches have been proposed for event correlation, and a number of event correlation products are available on the market. Unfortunately, existing products are mostly expensive, platform-dependent, and heavyweight solutions that have complicated design, being therefore difficult to deploy and maintain, and requiring extensive user training. For these reasons, they are often unsuitable for employment in smaller IT systems and on network nodes with limited computing resources.

Data mining techniques are a common choice for knowledge discovery from event logs, and the mining of patterns from event logs has been identified as an important system and network management task. Recently proposed mining approaches for accomplishing this task have often been based on some well-known algorithm for mining frequent itemsets, and they have focused on detecting frequent event type patterns. However, existing approaches have several shortcomings. Firstly, many of the proposed algorithms are variants of the Apriori algorithm which is inefficient for mining longer patterns. Secondly, recent research has concentrated on detecting frequent patterns, but the discovery of infrequent patterns is equally important, since this might reveal anomalous events that represent unexpected behavior of the system. Unfortunately, data clustering methods that can tackle this problem have seldom been applied for mining patterns from event logs. Thirdly, existing algorithms mostly focus on finding event type patterns, ignoring patterns of other sorts. In particular, the mining of line patterns provides the user a valuable insight into event logs, but this issue has received very little attention so far.

In order to address the problems described above, this thesis proposes novel tools and techniques for event log analysis. The main contributions of this thesis are the following:

- the development of Simple Event Correlator (SEC) that demonstrates the efficiency of a lightweight, platform independent, and open-source event correlator for monitoring event logs and processing event streams,
- the proposal of a novel data clustering algorithm for mining patterns from event logs,
- the proposal of a novel frequent itemset mining algorithm for mining frequent patterns from event logs.

Kokkuvõte

Käesolev väitekiri käsitleb sündmuste logide analüüsiga seotud sündmuste korrelatsiooni ja andmekaevandamise probleeme ning tutvustab uudseid vahendeid ja tehnikaid nende probleemide lahendamiseks. Sündmuste logid mängivad tänapäeva infosüsteemides tähtsat rolli, sest neis leiduv info on äärmiselt oluline süsteemi monitooringuks reaalajas ning juba toimunud sündmuste hilisemaks põhjalikumaks analüüsiks.

Sündmuste korrelatsioon on üks tähtsamaid sündmuste töötlemise tehnikaid, mida on viimase kümnekonna aasta jooksul võrguhalduse kontekstis põhjalikult uuritud ning mis on muutumas järjest olulisemaks ka teistes valdkondades, kaasa arvatud logide analüüs. Sündmuste korrelatsiooniks on välja pakutud mitmeid lähenemisi ning on loodud terve hulk tarkvaratooteid. Kahjuks on olemasolevad tooted kallid, platvormist sõltuvad ning keeruka ülesehitusega lahendused, mille tõttu nende installeerimine ja hooldus pole lihtsad ülesanded ning nende kasutamine nõuab mahukat koolitust. Seetõttu on nad sageli ebasobivad rakendamiseks väiksemates infosüsteemides ning piiratud arvutusvõimsusega võrgusõlmedes.

Teadmiste otsimiseks sündmuste logidest rakendatakse tihti andmekaevandamise tehnikaid ning logidest mustrite otsimine on oluline süsteemi- ja võrguhalduse ülesanne. Senised lähenemised selle ülesande lahendamiseks on enamasti põhinenud mõnel populaarsel sagedaste elemendihulkade otsimise algoritmil ning peamiselt on kaevandatud sündmuste tüüpidest koosnevaid mustreid. Kahjuks on senini kasutatud meetoditel mõned olulised puudused. Esiteks, paljud neist põhinevad Apriori algoritmil, mis aga ei sobi pikemate mustrite kaevandamiseks. Teiseks, olemasolevad meetodid on keskendunud sageliesinevate mustrite otsimisele, kuid harvaesinevate mustrite avastamine on sama oluline, sest see aitab kaasa anomaalsete sündmuste leidmisele. Kuigi andmete klasterdamise algoritmid võimaldavad seda probleemi lahendada, on neid harva rakendatud logidest mustrite kaevandamiseks. Kolmandaks, senini kasutatud meetodid on loodud peamiselt sündmuste tüüpidest koosnevate mustrite avastamiseks. Samas on eriti reamustrite kaevandamine oluline ülesanne, mis võimaldab kasutajal paremini mõista logis leiduva info iseloomu.

Lahendamaks ülal loetletud probleeme, tutvustab käesolev väitekiri uudseid vahendeid ning tehnikaid sündmuste logide analüüsiks. Väitekirja teaduslik panus on järgmine:

- Simple Event Correlator'i (SEC) väljatöötamine, mis demonstreerib kergekaalulise, platvormist sõltumatu ja avatud lähtekoodiga sündmuste korrelaatori sobivust ning efektiivsust sündmuste logide monitooringuks ja sündmuste voogude töötlemiseks,
- uudse klasterdamisalgoritmi väljatöötamine sündmuste logidest mustrite kaevandamiseks,
- uudse sagedaste elemendihulkade otsimise algoritmi väljatöötamine sündmuste logidest mustrite kaevandamiseks.

Acknowledgements

First of all, I would like to thank my supervisor Professor Ahto Kalja for accepting me as his PhD student. Unfortunately, event correlation and data mining for event logs are rather narrow fields of research and finding a good supervisor is not an easy task. I had already received several refusals from candidates from Estonia and other European countries when I was finally fortunate enough to meet Ahto Kalja. I am also thankful to Ahto for his patience, since on several occasions his student acted in a rather stubborn manner.

During the studies, a PhD student almost always faces funding problems and related paperwork (finding appropriate funding sources, writing applications for scholarships, etc.), and I was not an exception. Here I would like to express my gratitude to Dr. Margus Kruus, the head of the Department of Computer Engineering, who as a skillful manager helped me with these matters and spent many hours of his valuable time.

During the years of my PhD studies, I worked as a network and system management engineer at SEB Eesti Ühispank, and many of my research experiments were conducted in the bank's IT environment. I am grateful to my employer for the most generous financial support and for excellent in-house research opportunities.

Two of my colleagues from the bank deserve a special recognition. CTO Dr. Paul Leis actually suggested in late 1999 that I should pursue a PhD degree, and backed my academic efforts throughout my studies. DSO Kaido Raiend, my direct superior, accepted my activities at the university and always gave his full support to my research projects. It is not an exaggeration to say that without Paul and Kaido this thesis would have never been completed.

In addition to the financial support received from SEB Eesti Ühispank, this work was also supported by the Archimedes Foundation, the Estonian Science Foundation (grants 4067 and 5766), and the Estonian Information Technology Foundation (EITSA).

A number of people helped me to develop SEC with their ideas and thoughts, and I would especially like to thank (in alphabetic order) Al Sorrell, James Brown, John P. Rouillard, Jon Frazier, Mark D. Nagel, Rick Casey, and William Gertz. I deeply appreciate the help of all institutions that provided a detailed feedback about the application of SEC in their environment. I also wish to express my gratitude to Jon Stearley for his comments and suggestions about SLCT.

Last but not least, I would like to thank my family for their support and care – without it I would have never even started my PhD research.

Part I - Overview

1. Introduction

Event logging and event logs play an important role in modern IT systems. Today, many applications, operating systems, network devices, and other system components are able to log their events to a local or remote log server. For this reason, event logs are an excellent source for determining the health status of the system, and a number of tools have been developed over the past 10-15 years for monitoring event logs in real-time. However, majority of these tools can accomplish simple tasks only, e.g., raise an alarm immediately after a fault message has been appended to a log file. On the other hand, quite many essential event processing tasks involve *event correlation* – a conceptual interpretation procedure where new meaning is assigned to a set of events that happen within a predefined time interval [Jakobson and Weissman, 1995].

Event correlation is one of the most prominent real-time event processing techniques today. It has received a lot of attention in the context of network fault management over the past decade, and is becoming increasingly important in other domains as well, including event log monitoring. A number of approaches have been proposed for event correlation, and a number of event correlation products are available on the market. Unfortunately, existing products are mostly expensive, platform-dependent, and heavyweight solutions that have complicated design, being therefore difficult to deploy and maintain, and requiring extensive user training. For these reasons, they are often unsuitable for employment in smaller IT systems and on network nodes with limited computing resources.

So far, the rule-based approach has been frequently used for monitoring event logs – event processing tasks are specified by the human analyst as a set of rules, where each rule has the form *IF condition THEN action*. For example, the analyst could define a number of message patterns in the regular expression language, and configure the monitoring tool to send an SMS notification when a message that matches one of the patterns is appended to the event log. Despite its popularity, the rule-based approach has nevertheless some weaknesses – since the analyst specifies rules by hand using his/her past experience, it is impossible to develop rules for the cases that are not yet known to the analyst; also, finding an analyst with a solid amount of knowledge about the system is usually a difficult task. In order to overcome these weaknesses, various knowledge discovery techniques have been employed for event logs, with data mining methods being a common choice. It should be noted that while event log monitoring tools conduct on-line (real-time) analysis of event log data, data mining methods are designed for off-line analysis – an existing event log data set is processed for discovering new knowledge, with the data set remaining constant throughout the discovery process.

Recently proposed mining approaches for event logs have often been based on some well-known algorithm for *mining frequent itemsets*, and they have focused on detecting frequent event type patterns. However, existing approaches have several shortcomings. Firstly, many of the proposed algorithms are variants of the Apriori algorithm [Agrawal and Srikant, 1994] which is inefficient for mining longer patterns. Secondly, recent research has concentrated on detecting frequent patterns, but the discovery of infrequent patterns is equally important, since this might reveal anomalous events that represent unexpected behavior of the system. Unfortunately, *data clustering* methods that can tackle this problem have seldom been applied for mining patterns from event logs. Thirdly, existing algorithms mostly focus on finding event type patterns, ignoring patterns of other sorts. In particular, the mining of line patterns provides the user a valuable insight into event logs, but this issue has received very little attention so far.

The main contributions of this thesis are the following:

- the development of Simple Event Correlator (SEC) that demonstrates the efficiency of a lightweight, platform independent, and open-source event correlator for monitoring event logs and processing event streams,
- the proposal of a novel data clustering algorithm for mining patterns from event logs,
- the proposal of a novel frequent itemset mining algorithm for mining frequent patterns from event logs.

The remainder of this thesis overview paper is organized as follows – section 2 discusses event logging and event log monitoring issues; section 3 provides an overview of related work on event correlation, and describes SEC and its application experience; section 4 discusses common approaches for mining patterns from event logs, related work on data clustering and frequent itemset mining, and presents novel algorithms for mining patterns from event logs; section 5 concludes the paper.

2. Event logging and event log monitoring

Before discussing event logging and event log monitoring issues in detail, this section presents some introductory definitions. *Event* is a change in the system state, with some events corresponding to system faults (e.g., a disk failure) and some reflecting normal system activity (e.g., a successful user login). When a system component encounters an event, the component could emit an *event message* that describes the event. For example, when a disk of a server becomes full, the server could generate a timestamped "Disk full" message for appending to a local log file or for sending over the network as an SNMP trap. *Event logging* is a procedure of storing event messages to the event log, where *event log* is a regular file that is modified by appending event messages. (Although sometimes databases of events messages are also called event logs, this thesis has focused on flat-file event logs.) *Log client* is the system component that emits event messages for event logging. In this thesis, the term event has often been used for denoting event message when it is clear from the context.

In modern IT systems, event logs play an important role:

- since in most cases event messages are appended to event logs in real-time as they are emitted by system components, event logs are an excellent source of information for monitoring the system,

- information that is stored to the event log can be useful for analysis at a later time, e.g., for audit procedures or for retrospective incident analysis.

Event logging can take place in various ways. In the simplest case the log client keeps the event log on a local disk and modifies it when an event occurs. Unfortunately, event logs will be scattered across the system with this logging strategy, each log possibly requiring separate monitoring or other analysis. Furthermore, the strategy assumes the presence of a local disk which is not the case for many network nodes (e.g., switches and routers).

In order to address these problems, a flexible logging protocol called *syslog* was implemented for the BSD UNIX in the middle of 1980s. Over the past two decades, the BSD *syslog* protocol has become a widely accepted standard [Lonvick 2001] that is supported on many operating systems and is implemented in a wide range of devices like routers, switches, laser printers, etc. The *syslog* event message normally contains a message string, program name, level, and facility. Program name identifies the name of the sending application or process (e.g., *ftpd* or *sendmail*), level describes the severity of the event (e.g., *warning* or *emerg*), while facility describes the event category (e.g., *mail* or *auth*). In order to log an event, the log client must create a valid *syslog* event message and send it to a local or remote *syslog* server.

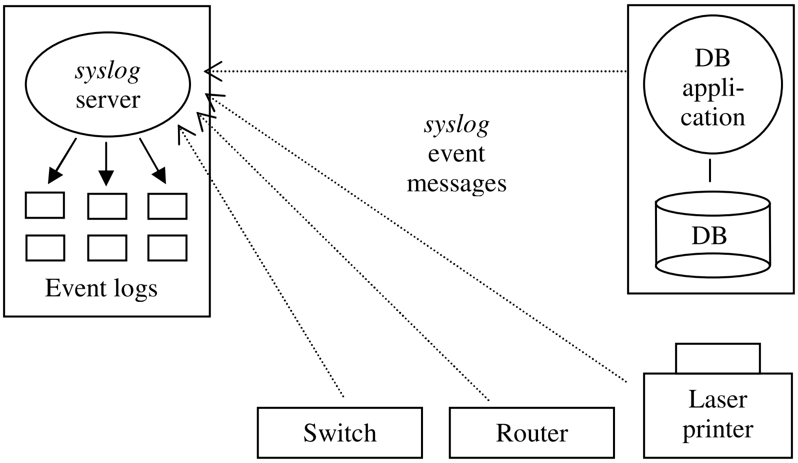

Figure 2.1 A sample centralized logging infrastructure

The communication between the client and the server takes place over the UDP transport protocol (usually, local clients can also contact the server via a file system socket). Unlike TCP, UDP does not provide guaranteed packet delivery, but it is much faster and consumes less network bandwidth than TCP. This makes the *syslog* protocol suitable for use in larger IT systems with many network nodes (the well-known SNMP protocol is popular for the same reason).

Also, since many applications, operating systems, and network devices support the *syslog* protocol, it is a perfect choice for building a centralized logging infrastructure. A centralized logging infrastructure comprises central log server(s) with applications, servers, routers, switches, and other system components acting as log clients (see Figure 2.1). Since event logs are now stored on a few central servers (rather than being scattered across the system), the event log monitoring and other analysis becomes much easier.

A central log server is usually a UNIX host which runs the *syslogd* daemon as *syslog* server for receiving events from remote and local clients. The very first implementation of *syslogd* was included in the 4.3BSD UNIX, and since then a number of *syslogd* variants have been created for other platforms. After receiving an event from a log client, the *syslogd* daemon classifies it using the event parameters (usually level and facility), and processes the event according to its class. Processing the event could mean appending it to a local event log, relaying it to another server, forwarding it to an external program, or simply discarding it.

Despite being widely used, the *syslog* protocol and many of its current implementations have some drawbacks – the protocol runs on top of UDP only and the log client can't choose a reliable transport protocol for event logging, event messages are not encrypted before they are sent over the network, and the *syslogd* daemon classifies events only using their level and facility. In order to overcome these difficulties, a standard for reliable *syslog* transmission has been proposed [New and Rose, 2001], and some *syslog* implementations support event logging over TCP which provides guaranteed packet delivery and allows one to use tunneling software for message encryption [Campi 2005]. As for the event classification, the *syslog-ng* software suite contains a *syslog* server that is able to distinguish between events by matching the sender host name, program name, and message string with user-defined regular expressions.

Apart from the *syslog*-style event logging described above, event logs can also be produced with the help of some other event management protocol that was not specifically designed for logging. When a certain protocol is used for sending event messages from system components to a central event management server, the central event management software could write received events to an external event log. For example, many network management platforms and applications that rely on SNMP protocol (e.g., HP OpenView NNM, Tivoli NetView, and *snmptrapd* daemon from the Net-SNMP software package) are capable of producing external event log files.

Because of the importance of event logs as the source of system health information, many tools have been developed over the past 10-15 years for monitoring event logs in real-time. Swatch [Hansen and Atkins, 1993] was the first such tool and is still used by many sites. Swatch monitors log files by reading every event message line that is appended to the log file, and compares it with rules where the conditional part of each rule is a regular expression (rules are stored in a textual configuration file). If the regular expression of a certain rule matches the event message line, Swatch executes the action part of the rule. Actions include sending a mail, executing an external program, writing a notification to the system console, etc. Swatch has also an option for ignoring repeated event messages for a given time interval.

Another popular tool for event log monitoring is Logsurfer [Ley and Ellerman, 1996]. Like Swatch, Logsurfer uses rule-based approach for event processing, employs regular expressions for recognizing input events, and monitors log files by comparing appended message lines with its rules. Apart from executing actions immediately when certain event messages are observed, Logsurfer also supports contexts and dynamic rules. Context is a memory-based buffer for storing event messages, and Logsurfer can report the content of a context through an external program. Dynamic rule is a rule that has been created from another rule with a special action.

In addition to commonly used Swatch and Logsurfer, a number of other tools exist for monitoring event logs in real-time, and the interested reader is referred to the Loganalysis website (*http://www.loganalysis.org*) for more information. Apart from standalone monitoring tools, some system and network management platforms like HP OpenView Operations (formerly called ITO) and Tivoli Risk Manager have also capabilities for monitoring event logs. Nevertheless, in order to use these capabilities, the whole platform must be deployed which is a complex and time-consuming task.

3. Simple Event Correlator (SEC)

3.1. Introduction

According to the widely accepted definition by Jakobson and Weissman, *event correlation* is a conceptual interpretation procedure where new meaning is assigned to a set of events that happen within a predefined time interval [Jakobson and Weissman, 1995]. During this procedure, original events might be removed and new events might be created. For example, consecutive events "Device internal temperature too high" and "Device unreachable" could be replaced by an event "Device stopped working due to overheating". A software application that implements event correlation is called *event correlator*.

Jakobson and Weissman have considered the following common event correlation operations in their work:

- Compression – reduce multiple occurrences of identical events into a single event,
- Filtering – suppress an event if one of its parameters has a certain value,
- Suppression – suppress an event if a certain operational context is present,
- Counting – counting and thresholding the number of repeated arrivals of identical events,
- Escalation – in the presence of a certain operational context, assign a higher value to a certain event parameter (e.g., severity),
- Generalization – replace an event with an event from its superclass,
- Specialization – replace an event with an event from its subclass,
- Temporal relationship – correlate events depending on the order and time

of their arrival,

- Clustering – employ a complex correlation pattern where pattern components are previously defined correlation operations, primary network events, or external tests.

The use of event correlation allows one to reduce the number of event messages sent to the system technicians, to identify root causes of the problems, to derive new and more meaningful events from existing ones, etc. Without the employment of event correlation techniques, event messages can easily overwhelm the human, since in modern IT systems it is not rare that hundreds of event messages are emitted by system components every second.

As an example of the importance of event correlation, consider a common scenario involving *linkdown* and *linkup* event messages emitted by many network devices. On one hand, network link faults are usually critical events that require human intervention, but on the other hand, very short occasional link outages (*linkdown* is immediately followed by *linkup* with no further failures for the link) are frequent in today's networks and normally don't deserve any attention. Therefore, it is highly impractical to forward *linkdown* event messages to network technicians without previously correlating them. As a solution for this problem, many sites use the following event correlation scheme – if the *linkdown* event is not followed by *linkup* in *t* seconds, forward the *linkdown* event message to network technicians; otherwise generate the *linkbounce* event, and if more than *n* such events have been observed within the last *t'* seconds, generate the *linkqualitylow* event and forward it to network technicians. In this way, technicians can focus on relevant network link issues and are not overwhelmed by vast amounts of meaningless event messages.

One of the contributions of this thesis is the development of a lightweight, platform independent, and open-source tool for event correlation called Simple Event Correlator (SEC). SEC receives its input from regular files, named pipes, and standard input, and can thus be employed as an event log monitoring solution and be easily integrated with other applications. The rest of this section is organized as follows – section 3.2 provides an overview of related work on event correlation and discusses the motivation for developing SEC, section 3.3 gives an overview of SEC, and section 3.4 describes its application experience.

3.2. Related work on event correlation and the motivation for developing SEC

Over the past decade, event correlation has received a lot of attention in the context of network fault management. A number of approaches have been proposed for event correlation, including rule-based [Froehlich et al., 2002], codebook based [Yemini et al., 1996], Bayes network based [Meira 1997; Steinder and Sethi, 2002], neural network based [Wietgrefe et al., 1997; Wietgrefe 2002], and graph based [Gruschke 1998] methods. There are also a number of event correlator products available on the market, like HP ECS, SMARTS, NetCool, NerveCenter, LOGEC, and RuleCore.

Rule-based approach is a common approach for event correlation and has been employed in several products like HP ECS and RuleCore. Some rule-based correlators closely resemble rule-based artificial intelligence (AI) systems where rules represent the knowledge about the system (the Rete algorithm [Forgy 1982] can be employed in such systems for finding matching rules efficiently), while in the case of other rule-based correlators rules are used to define the event correlation algorithm. One of the main advantages of the rule-based event correlation is the fact that humans find it usually natural to express their knowledge in terms of rules. For example, it is easy to describe temporal relations between events with rules, while it could be cumbersome with other methods. Furthermore, unlike some other event correlation methods (e.g., neural network based correlation), the rule-based event correlation is clear and transparent to the end user. As argued in [Rich and Knight, 1991], if end users do not understand why and how the application reached its output, they tend to ignore the results computed by that application.

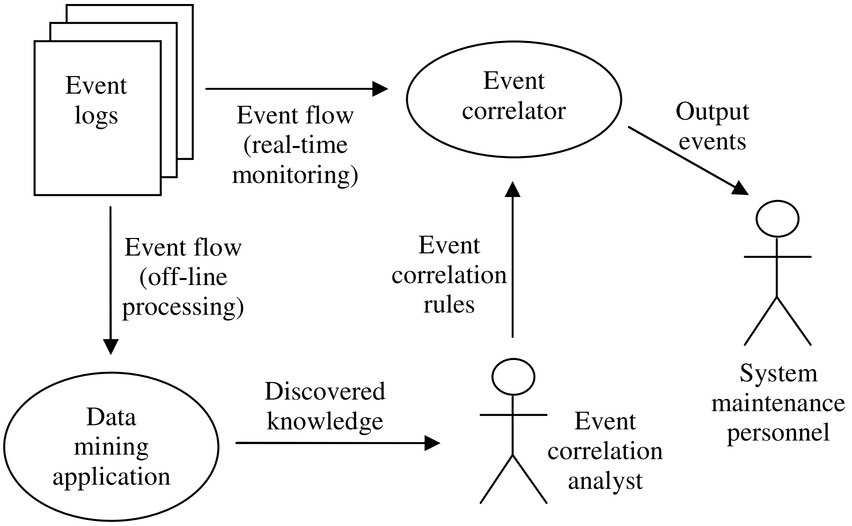
Event logs
Event flow (real-time monitoring)
Event correlator
Output events
Event flow (off-line processing)
Event correlation rules
System maintenance personnel
Data mining application
Discovered knowledge
Event correlation analyst

Figure 3.1 The synergy between rule-based event correlation and data mining

It should be noted that the rule-based event correlation does not include the learning process – past experience is not used for deriving new knowledge, and all event correlation rules are specified by the human analyst. However, as discussed in section 1, it is impossible to develop rules for the cases that are not yet known to the analyst, and also, finding an analyst with a solid amount of knowledge about the system is usually a difficult task. In order to address these problems, various data mining methods have been proposed for knowledge discovery from event logs. The TASA system developed at the University of Helsinki [Klemettinen 1999] mines frequent event type patterns and derives rules from detected patterns that describe temporal relations between event types. The analyst then selects interesting rules and converts them to event correlation rules. Burns et al. have developed the EventBrowser system [Burns et al., 2001] that uses data mining and visualization techniques for finding event patterns and for creating event correlation rules. The tools and algorithms described in section 4 can also assist the analyst in discovering event correlation knowledge. Figure 3.1 depicts the synergy between rule-based event correlation and data mining for event logs.

Although event correlation has been long applied primarily for network fault management, there is a clear trend to extend its use to other application domains as well [Jakobson et al., 2000], most notably to security management and intrusion detection. Staniford et al. have implemented Spice event correlation engine for detecting stealthy portscans [Staniford et al., 2002], Julisch has proposed an alarm clustering method for root cause analysis [Julisch 2003], Morin and Debar have suggested the chronicles formalism for correlating intrusion symptoms [Morin and Debar, 2003], Snort IDS [Roesch 1999] is able to count and threshold events, etc.

Although event correlation systems that are currently available on the market have been highly successful and are used worldwide by many larger companies, they suffer from a number of drawbacks.

Firstly, existing systems are often heavyweight solutions that have complicated design and user interface. This means that their deployment and maintenance is time-consuming, and they require extensive user training. Also, their complexity and resource requirements make them often unsuitable for employment in smaller IT systems and for event correlation on nodes with limited computing resources, e.g., for distributed event correlation in ad hoc and sensor networks (nodes in such networks have limited hardware capabilities).

Secondly, since existing systems are mostly commercial, they are platform-dependent – customers are supplied with program binaries that run on a limited number of operating systems. Furthermore, several commercial systems have been designed for one particular network management platform only. Some systems also suffer from the fact that they have been designed specifically for network fault management, and their application in other domains (including event log monitoring) is cumbersome.

Thirdly, existing systems tend to be quite expensive. Therefore, many academic institutions and smaller companies with more limited budget are unable to use them for daily network and system management tasks or for research experiments. Since a lot of research has been done in the field of event correlation recently, some experimental correlation engine prototypes have been created, but most such prototypes are not publicly available on the Internet. Currently, there is no open-source event correlation engine available that would be actively developed and mature enough for use in a production environment (although RuleCore was initially an open-source project, it has become a commercial product). One interesting event correlation related open-source project is CLIPS, which is an environment for creation of rule-based expert systems. Although CLIPS itself is not an event correlation tool, it has been successfully used for constructing event correlation systems [Jakobson and Weissman 1995; Jakobson et al., 2000].

It should be noted that some event correlation operations are supported by popular event log monitoring tools, e.g., Swatch supports event compression and Logsurfer supports temporal relationship operations. Nevertheless, the event correlation capabilities of current log monitoring tools are quite limited.

For the reasons above, quite many sites are using homegrown event correlation solutions which often comprise a few application-specific shell scripts. Each time a new application is set up, a new solution has to be developed, which is rather impractical and time-consuming.

One of the main contributions of this thesis was the development of an open-source platform independent tool for rule-based event correlation called Simple Event Correlator (SEC) which addresses the problems described in this section. The primary design goal of SEC was to fill the gap between homegrown and commercial solutions, and to create a lightweight and easily customizable tool that could be used for a wide variety of event correlation tasks, either standalone or integrated with other applications.

3.3. Description of SEC

SEC is an open-source event correlation tool that uses rule-based approach for processing events. This approach was chosen because of its naturalness of knowledge representation and transparency of the event correlation process. The main design objectives for SEC were platform independence, lightweight build and ease of configuration, applicability for a wide variety of event correlation tasks, and low consumption of system resources.

In order to achieve independence from operating system platforms, the author decided to write SEC in Perl. Since Perl runs on almost every operating system flavour and has become a standard part of many OS distributons, Perl applications are able to run on a wide range of operating systems. In addition, well-written Perl programs are fast and memory-efficient.

SEC does not need much disk space and is very easy to install, since its current size is only about 250KB, and its configuration is stored in regular text files (the size of each file is typically a few kilobytes). Also, since SEC is written entirely in Perl and does not depend on other software packages, it can be used instantly after its source distribution has been unpacked, without any additional preparations (such as compiling and linking the source or installing other software).

SEC receives its input events from file streams (before the 2.2.0 version only one input stream was supported), and can produce output events by executing user-specified shell commands, by writing messages to files or named pipes, by calling precompiled Perl subroutines, etc. (note that output events can be sent over the network to another instance of SEC, allowing one to configure distributed event correlation schemes). Regular files, named pipes, and standard input are currently supported as input, allowing one to use SEC as an event log monitoring solution and to integrate it with any application that is able to write its output events to a file stream. Applications that have an event management API can also be integrated through simple plugins that employ API calls to read the application’s event stream, and copy it to the standard output or file (a sample plugin for HP OpenView Operations is a part of the SEC package).

SEC configuration is stored in text files which can be created and modified with any text editor. Each configuration file contains one or more rules, and rulesets from different files are applied virtually in parallel. The 1.X versions of SEC used a configuration file syntax where a rule definition was a list of values separated by the bar symbol (|). Starting from the 2.0 version, a keyword-value-like syntax is employed which is more readable and flexible.

An important part of the SEC rule is the event matching pattern. SEC supports regular expressions, substrings, Perl subroutines, and truth values as patterns. Support for regular expressions eases the configuration of SEC, since many UNIX tools (like *grep*, *sed*, *find*, etc.) rely on regular expressions, and therefore most system and network administrators are already familiar with the regular expression language. Also, since majority of event log monitoring tools use regular expression language for matching events, SEC can be deployed as a log monitoring solution without any extra integration work. Starting from the 2.3.0 version, events can be passed to precompiled Perl subroutines for recognition which allows the user to configure custom event matching schemes.

In addition to event matching pattern, most rule definitions specify a list of *actions*, and optionally a Boolean expression of *contexts* (starting from the 2.1.7 version, Perl expressions can also be used as operands). The SEC contexts are logical entities created during the event correlation process, with each context having a certain lifetime (either finite or infinite). Contexts can be used for activating and deactivating rules dynamically at runtime, e.g., if a rule definition has (*X* OR *Y*) specified for its context expression and neither the context *X* nor the context *Y* exist at a given moment, the rule will not be applied. Another important function of the SEC contexts is to act as event stores – events of interest can be associated with a context, and all the collected events supplied for an external processing at a later time (this idea was borrowed from Logsurfer).

Currently, SEC supports nine rule types that implement a number of common event correlation operations. SEC actions were not only designed for generating output events, but also for making rules to interact, for managing contexts and storing events, for connecting external event analysis modules to SEC, for executing Perl miniprograms and subroutines without forking a separate process, etc. By combining several rules with appropriate action lists and context expressions, more complex event correlation schemes can be defined (see the research papers of this thesis [Vaarandi 2002a; Vaarandi 2002c] and the SEC online documentation for detailed examples). In order to learn more about SEC performance, please see the case studies in the research papers of this thesis [Vaarandi 2002a; Vaarandi 2002c].

This subsection contained only a brief overview of SEC and its capabilities. For a thorough description, the interested reader is referred to the SEC online documentation. There are also several other sources of information available about SEC. “Working with SEC – the Simple Event Correlator” [Brown 2003] is an online tutorial that not only provides a good introduction to SEC but also covers a number of advanced issues like integrating SEC with MySQL. Chapter 5 of “Hardening Linux” [Turnbull 2005] discusses how to employ SEC for monitoring *syslog* log files. Also, recently a paper with a useful ruleset library has been published that describes the application of SEC at the University of Massachusetts at Boston [Rouillard 2004].

3.4. SEC application experience

The 1.0 version of SEC was released in March 2001. The first versions of SEC were mainly used for accomplishing network management tasks, e.g., for augmenting network management agents with event correlation capabilities [Vaarandi 2002a] and for central event correlation at the HP OpenView NNM management server [Vaarandi 2002b]. At the time of writing this thesis overview paper (April 2005), SEC is applied for a wide variety of event correlation tasks in the domains of network fault and performance management, intrusion detection, log file analysis, event warehousing, etc. A number of research papers have been published that describe the use of SEC in various domains [Casey 2001; Dillis 2003; Gorton 2003; Meehan 2005; Rouillard 2004; Sawall 2004].

Applications and database platforms that SEC has been integrated with include HP OpenView Network Node Manager, HP OpenView Operations (both management server and agents), CiscoWorks, BMC Patrol, Nagios, SNMPTT, Snort IDS, Oracle, and MySQL. SEC has been used on a variety of OS platforms, like Linux, Solaris, HP-UX, AIX, FreeBSD, Tru64 UNIX, Mac OS X, and Windows2000.

SEC has been successfully adopted by institutions with various sizes, from companies with relatively small IT systems to large corporations with global networks. The author has received a detailed feedback from more than 20 institutions that use SEC (see Appendix A). The data in Appendix A reveal that the major advantages of SEC over other solutions are its open-source nature and free download status, ease of configuration, flexibility, applicability for a wide range of event correlation tasks, and ability to run on multiple platforms.

4. Pattern mining techniques for event logs

4.1. Introduction

Since event logs play a significant role in modern IT systems, the mining of patterns from event logs has been identified as an important system and network management task [Klemettinen 1999; Ma and Hellerstein 2000; Mannila et al., 1997; Pei et al., 2000; Srivastava et al., 2000; Zheng et al., 2002]. Recently proposed mining algorithms have often been variants of the Apriori algorithm for mining frequent itemsets [Klemettinen 1999; Ma and Hellerstein 2000; Mannila et al., 1997; Zheng et al., 2002], and they have been mainly designed for detecting frequent event type patterns [Klemettinen 1999; Ma and Hellerstein 2000; Mannila et al., 1997; Pei et al. 2000; Zheng et al., 2002]. The algorithms assume that each event from the event log has two attributes – time of event occurrence and event type – and the event log is considered as a sequence $E_1,...,E_n$, where $E_i = (t_i, e_i)$ is an event, e_i is the type of E_i, t_i is the occurrence time of E_i, and $t_i \leq t_j$ when $i < j$.

Detected frequent patterns reveal what event types are more closely related, e.g., the *OutOfMemory* event is always immediately followed by the *Reboot* event, or events *GET article193.html* and *GET article426.html* often occur together. The knowledge discovered during the mining can be used for various purposes, like building rules for event correlation systems (see section 3.2 for a detailed discussion), improving designs of web sites [Pei et al., 2000; Srivastava et al., 2000], etc. In some cases, it might be useful to derive association rules from detected patterns, since this representation of knowledge could be more convenient to the end user. Each association rule has the following form – if a certain combination of event types occurs within a certain time window, then another combination occurs within another time window with a certain probability. As an example, consider the following rule – if events *A* and *B* occur within 20 seconds then event *C* also occurs within 60 seconds with the probability of 95%. However, since the generation of association rules from frequent patterns is a well-studied problem, this issue has not been investigated in this thesis.

There are several ways for defining the frequent event type pattern, with two definitions being most common. In the case of the first definition (e.g., see [Klemettinen 1999]), the algorithm views the event log as a set of overlapping windows, where each window starts from a time moment *T* and contains events from a time frame of *W* seconds: $\{E_i \mid T \leq t_i < T + W\}$, $W > 0$, $T \leq t_n, T + W > t_1$ (the window size *W* is given by the user). A certain combination of event types is considered a frequent pattern if this combination is present at least in *s* windows, where the threshold *s* is specified by the user.

Original event log:
(1, login), (2, automatic backup done), (3, logout), (13, automatic backup done), (22, login), (23, automatic backup done), (24, logout)

Event log after it has been divided into slices:
(1, login), (3, logout) – issued by host A
(2, automatic backup done) – issued by host B
(13, automatic backup done) – issued by host C
(22,login), (24, logout) – issued by host D
(23, automatic backup done) – issued by host E

Figure 4.1 A sample event log

In the case of the second definition (e.g., see [Pei et al., 2000]), the algorithm assumes that the event log has been divided into non-overlapping slices according to some criteria (e.g., events from the same slice were all issued by the same host). A certain combination of event types is considered a frequent pattern if this combination is present at least in *s* slices (the threshold *s* is given by the user). Although the use of this definition requires more elaborate preprocessing of the event log, it also eliminates the noise that could appear when events from different slices are mixed.

As an example, consider the event log in Figure 4.1. If the threshold *s* is 2 and the window size *W* is 3, then event types *login*, *automatic backup done*, and *logout* would form a frequent pattern according to the first definition (this combination of event types is present in windows starting from time moments 1 and 22). On the other hand, it is obvious to the human observer that the presence of the *automatic backup done* event near *login* and *logout* is purely coincidental, since automatic backups don't depend on user sessions at other nodes. Fortunately, the second definition would not associate the *automatic backup done* event with *login* and *logout.* As another example, if the goal is to detect patterns in intrusion attempts from the same IP address, the first definition is likely to yield many irrelevant patterns, while the second definition produces relevant patterns only (if slices are formed by the source IP address of intrusion attempts). In other words, in many cases the appropriate division of the event log into slices helps the mining algorithm to focus more closely on its task, and thus to save computing resources and to increase its output quality. Also, event logs often contain data that help to arrange more closely related events into slices (e.g., the program name field of *syslog* messages). For these reasons, the second approach for defining the frequent event type pattern has been employed in this thesis.

The order of events in windows or slices (occurrence time ascending order) is often taken into account during the mining, since this could reveal causal relations between event types – e.g., instead of an unordered set *{DeviceDown, FanFailure}* the algorithm outputs a sequence *FanFailure* → *DeviceDown*. However, as pointed out in [Klemettinen 1999], the mining of unordered frequent event type sets is equally important. Due to network latencies, events from remote nodes might arrive and be written to the log in the order that differs from their actual occurrence order. Even if events are timestamped by the sender, system clocks of network nodes are not always synchronized, making it impossible to restore the original order of events. Also, in many cases events *A* and *B* might occur in many windows or slices together, but their occurrence order could vary (e.g., since they are not causally related). Therefore, the order of events in a slice has not been considered important in this thesis.

Existing mining algorithms for event logs have several shortcomings. Firstly, many of the algorithms are variants of Apriori which is inefficient for mining longer patterns (see section 4.3 for a detailed discussion). Secondly, recent research has focused on detecting frequent patterns, but as pointed out in [Burns et al., 2001], the discovery of infrequent patterns is equally important, since this might reveal anomalous events that represent unexpected behavior of the system. For example, fault events are usually very infrequent in a well-maintained system, but are nevertheless highly interesting. Unfortunately, frequent itemset mining algorithms like Apriori don't address this problem, while data clustering methods that are able to tackle the problem have seldom been applied for mining patterns from event logs.

Thirdly, existing algorithms mostly focus on finding event type patterns, ignoring patterns of other sorts. However, since many event logs are textual and contain single line messages (e.g., this is the case for *syslog* log files), the mining of line patterns provides the user a valuable insight into event logs. Because event log messages rarely contain explicit event type codes (e.g., *syslog* messages do not have the event type parameter), it is difficult to mine frequent event type patterns from a raw event log. Fortunately, it is possible to derive event types from event log lines, since very often the events of the same type correspond to a certain line pattern. For example, the lines

Router myrouter1 interface 192.168.13.1 down
Router myrouter2 interface 10.10.10.12 down
Router myrouter5 interface 192.168.22.5 down

represent the event type "router interface down", and correspond to the line pattern *Router * interface * down*.

Note that the mining of line patterns is not merely a preprocessing step, but can be very useful for other purposes as well. For example, frequent line patterns could help the human analyst to construct the event log model that describes the normal system activity (because event messages that reflect the normal system activity are usually frequent). The model can be employed for event log monitoring – if an event message is appended to the log that does not fit the model, it can be regarded anomalous and an alarm can be raised. On the other hand, the detection of infrequent line patterns could help the analyst to find previously unknown fault messages.

In order to address the problems discussed above, this thesis analyses the suitability of existing prominent frequent itemset mining algorithms for event log data, and proposes a novel algorithm for mining frequent event type and line patterns from event logs.

Another contribution of this thesis is the study of data clustering algorithms and the proposal of a new clustering algorithm for mining line patterns from event logs. Clustering algorithms aim at dividing the set of objects into groups (or *clusters*), where objects in each cluster are similar to each other (and as dissimilar as possible to objects from other clusters). Objects that do not fit well to any of the clusters detected by the algorithm are considered to form a special cluster of *outliers*. When event log lines are viewed as objects, clustering algorithms are a natural choice, because line patterns form natural clusters – lines that match a certain pattern are all similar to each other, and generally dissimilar to lines that match other patterns. Also, the cluster of outliers would contain infrequent lines that could represent previously unknown fault conditions, or other unexpected behavior of the system that deserves closer investigation.

The rest of this section is organized as follows – sections 4.2 and 4.3 discuss related work on data clustering and frequent itemset mining, section 4.4 describes the properties of event log data and the motivation for the development of new algorithms for mining patterns from event logs, and sections 4.5 and 4.6 present an overview of novel data clustering and frequent itemset mining algorithms for event logs.

4.2. Related work on data clustering

Clustering methods have been researched extensively over the past decades, and many algorithms have been developed [Berkhin 2002; Hand et al., 2001; Jain et al., 1999]. The clustering problem is often defined as follows: given a set of points with *n* attributes in the data space $\Re^n$, find a partition of points into clusters so that points within each cluster are close (similar) to each other. In order to determine, how close (similar) two points *x* and *y* are to each other, a distance function *d(x, y)* is employed. Many algorithms use a certain variant of L_p norm (p = 1, 2, ...) for the distance function:

$$d_p(x, y) = \sqrt[p]{\sum_{i=1}^{n} \left| x_i - y_i \right|^p}$$

As a clustering example, consider the *k*-medoids method which divides the set of points into *k* clusters, where the value of *k* is given by the user. Each cluster is represented by a certain point (medoid) from the cluster, and each point belongs to the cluster represented by the closest medoid. The method starts with an arbitrary selection of *k* medoids, and continues its work by replacing a medoid with a non-medoid at each step, until the best clustering is achieved (after each step, the quality of clustering is measured with a special function). Variants of this method are used by a number of algorithms like PAM, CLARA, and CLARANS [Ng and Han, 1994]. Another popular methods include the *k*-means method (like *k*-medoids, it divides the set of points into *k* clusters, but instead of medoids employs means for representing clusters), the divisive method (it starts with a single cluster containing all points and splits clusters recursively), the agglomerative method (it starts with single point clusters and joins clusters recursively), etc. [Berkhin 2002; Jain et al., 1999]. It should be noted that while some methods expect the user to specify the number of clusters, other methods don’t have that restriction.

Today, there are two major challenges for traditional clustering methods that were originally designed for clustering numerical data in low-dimensional spaces (where usually *n* is well below 10).

Firstly, quite many data sets consist of points with *categorical* attributes, where the domain of an attribute is a finite and unordered set of values [Ganti et al. 1999, Guha et al., 2000]. As an example, consider a categorical data set with attributes *car-manufacturer*, *model*, *type*, and *color*, and data points ('Honda', 'Civic', 'hatchback', 'green') and ('Ford', 'Focus', 'sedan', 'red'). Also, it is quite common for categorical data that different points can have different number of attributes. Therefore, it is not obvious how to measure the distance between data points. Though several distance functions for categorical data have been proposed, e.g., the Jaccard coefficient

$$d(X,Y) = \frac{|X \cap Y|}{|X \cup Y|}$$

the choice of the right function is often not an easy task [Guha et al., 2000].

Note that event log lines can be viewed as points from a categorical data set, since each line can be divided into words, with the n-th word serving as a value for the n-th attribute. For example, the log file line *Connection from 192.168.1.1* could be represented by the data point ('Connection', 'from', '192.168.1.1'). This representation of event log data has also been used in this thesis.

Secondly, quite many data sets today are high-dimensional, where data points can easily have tens of attributes. Unfortunately, traditional clustering methods have been found not to work well when they are applied to high-dimensional data. As the number of dimensions n increases, it is often the case that for every pair of points there exist dimensions where these points are far apart from each other, which makes the detection of any clusters almost impossible (according to some sources, this problem starts to be severe when $n \geq 15$) [Aggarwal et al., 1999; Agrawal et al., 1998; Berkhin 2002; Hinneburg and Keim, 1999]. Furthermore, traditional clustering methods are often unable to detect natural clusters that exist in subspaces of the original high-dimensional space [Aggarwal et al., 1999; Agrawal et al., 1998]. For instance, data points (1333, 1, 1, 99, 25, 2033, 1044), (12, 1, 1, 724, 667, 36, 2307), and (501, 1, 1, 1822, 1749, 808, 9838) are not seen as a cluster by many traditional methods, since in the original data space they are not very close to each other. On the other hand, they form a very dense cluster in the second and third dimension of the space.

The dimensionality problems described above are also relevant to the clustering of event log data, since event log data is typically high-dimensional (i.e., there are usually more than just 3-4 words on every line), and most of the line patterns correspond to clusters in subspaces. For example, the lines

log: connection from 192.168.1.1
log: RSA key generation complete
log: Password authentication for john accepted.

form a natural cluster in the first dimension of the data space, and correspond to the line pattern *log: **.

During past few years, several algorithms have been developed for clustering high-dimensional data, like CLIQUE, MAFIA, CACTUS, and PROCLUS. The CLIQUE [Agrawal et al., 1998] and MAFIA [Goil et al., 1999] algorithms closely remind the Apriori algorithm for mining frequent itemsets [Agrawal and Srikant, 1994]: they start with identifying all clusters in 1-dimensional subspaces, and after they have identified clusters $C_1,...,C_m$ in k-1-dimensional subspaces, they form cluster candidates for k-dimensional subspaces from $C_1,...,C_m$, and then check which of those candidates are actual clusters. Those algorithms are effective in discovering clusters in subspaces, because they do not attempt to measure distance between individual points, which is often meaningless in a high-dimensional data space. Instead, their approach is *density based*, where a clustering algorithm tries to identify *dense regions* in the data space, and forms clusters from those regions. Unfortunately, the CLIQUE and MAFIA algorithms suffer from the fact that Apriori-like candidate generation and testing for higher-dimensional subspaces involves high runtime overhead (see section 4.3 for a detailed discussion). The CACTUS algorithm [Ganti et al., 1999] first makes a pass over the data and builds a data summary, then generates cluster candidates during the second pass using the data summary, and finally determines the set of actual clusters. Although CACTUS makes only two passes over the data and is therefore fast, it is susceptible to the phenomenon of chaining (long strings of points are assigned to the same cluster) [Hand et al., 2001], which is undesirable if one wants to discover line patterns from event logs. The PROCLUS algorithm [Aggarwal et al., 1999] uses the *k*-medoids method for detecting *k* clusters in subspaces of the original space. However, in the case of event log data the number of clusters can rarely be predicted accurately, and therefore it is not obvious what is the right value for *k*.

4.3. Related work on frequent itemset mining

Recently proposed mining approaches for event logs have often been based on some well-known algorithm for mining frequent itemsets (like Apriori or FP-growth) [Klemettinen 1999; Ma and Hellerstein 2000; Mannila et al., 1997; Pei et al. 2000; Zheng et al., 2002]. In this subsection we will discuss the frequent itemset mining problem and prominent algorithms for addressing this problem.

Let $I = \{i_1,...,i_n\}$ be a set of items. If $X \subseteq I$, X is called an *itemset*, and if $|X| = k$, X is also called a *k-itemset*. A *transaction* is a tuple $T = (tid, X)$ where *tid* is a transaction identifier and X is an itemset. A *transaction database D* is a set of transactions, and the *cover* of an itemset X is the set of identifiers of transactions that contain X: $cover(X) = \{tid \mid (tid, Y) \in D, X \subseteq Y\}$. The *support* of an itemset X is defined as the number of elements in its cover: $supp(X) = |cover(X)|$. The task of *mining frequent itemsets* is formulated as follows – given the transaction database D and the *support threshold s*, find itemsets $\{X \mid supp(X) \geq s\}$ and their supports (each such set is called a *frequent itemset*).

The frequent itemset mining problem has received a lot of attention during the past decade, and a number of mining algorithms have been developed. For the sake of efficient implementation, most algorithms order the items according to certain criteria, and use this ordering for representing itemsets. In the rest of this section, we assume that if $X = \{x_1,...,x_k\}$ is an itemset, then $x_1 < ... < x_k$.

The first algorithm developed for mining frequent itemsets was Apriori [Agrawal and Srikant, 1994] which works in a breadth-first manner – discovered frequent *k*-itemsets are used to form candidate *k*+1-itemsets, and frequent *k*+1-itemsets are found from the set of candidates.

Recently, an efficient *trie* (prefix tree) data structure has been proposed for the candidate support counting [Bodon 2003; Borgelt 2003]. Each edge in the *itemset trie* is labeled with the name of a certain item, and when the Apriori algorithm terminates, non-root nodes of the trie represent all frequent itemsets. If the path from the root node to a non-root node N is $x_1,...,x_k$, N identifies the frequent itemset $X = \{x_1,...,x_k\}$ and contains a counter that equals to $supp(X)$. In the remainder of this section, we will use notations $node(x_1,...,x_k)$ and $node(X)$ for N, and also, we will always use the term *path* to denote a path that starts from the root node. Figure 4.2 depicts a sample transaction database and an itemset trie (the support threshold is 2 and items are ordered in lexicographic order $a < b < c < d < e$).

Transaction ID	Itemset
1	*abcde*
2	*abc*
3	*bcd*
4	*abc*
5	*ab*

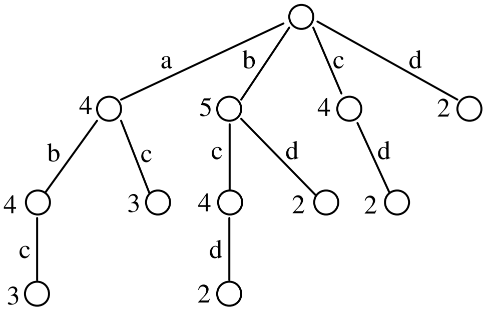

Figure 4.2 A sample Apriori itemset trie

As its first step, the Apriori algorithm detects frequent 1-itemsets and creates nodes for them. The nodes for candidate k+1-itemsets are generated as follows – for each node $node(x_1,...,x_k)$ at depth k all its siblings (nodes with the same parent) will be inspected. If $x_k < y_k$ for the sibling $node(x_1,...,x_{k-1},y_k)$, then the candidate node $node(x_1,...,x_k,y_k)$ will be inserted into the trie with its counter set to zero. Since every subset of a frequent itemset must also be frequent, this candidate generation procedure guarantees that all frequent k+1-itemsets are present in the set of candidates. In order to find frequent k+1-itemsets, the algorithm traverses the itemset trie for each transaction $(tid, Y) \in D$, and increments the counter in $node(X)$ if $X \subseteq Y, |X| = k + 1$. After the database pass, the algorithm removes nodes for infrequent candidate itemsets.

Although the Apriori algorithm works well when frequent itemsets contain relatively few items (e.g., 4-5), its performance starts to deteriorate when the size of frequent itemsets increases [Bayardo 1998; Han et al., 2000; Zaki 2000]. In order to produce a frequent itemset $\{x_1,...,x_k\}$, the algorithm must first produce its 2^k-2 subsets that are also frequent, and when the database contains frequent k-itemsets for larger values of k (e.g., 30-40), the number of nodes in the itemset trie could be very large. As a result, the runtime cost of the repeated traversal of the trie will be prohibitive, and the trie will consume large amounts of memory.

In recent past, several algorithms have been proposed that explore the search space in a depth-first manner, and that are reportedly by an order of a magnitude faster than Apriori. The most prominent depth-first algorithms for mining frequent itemsets are Eclat [Zaki 2000] and FP-growth [Han et al., 2000]. An important assumption made by Eclat and FP-growth is that the transaction database fits into main memory. At each step of the depth-first search, the algorithms are looking for frequent k-itemsets $\{p_1,...,p_{k-1},x\}$, where the prefix $P = \{p_1,...,p_{k-1}\}$ is a previously detected frequent k-1-itemset. When looking for these itemsets, the algorithms extract from the database the data describing transactions that contain the itemset P, and search only this part of the database. If frequent k-itemsets were found, one such itemset is chosen for the prefix of the next step, otherwise the new prefix is found by backtracking. Since the database is kept in main memory using data structures that facilitate the fast extraction of data, Eclat and FP-growth can explore the search space faster than Apriori.

The main difference between the Eclat and FP-growth algorithm is how the transaction database is stored in memory. Eclat keeps item covers in memory (this representation is also called the *vertical* database layout) which allows the algorithm to calculate itemset supports with fast intersection operations, e.g., $|cover(P) \cap cover(\{x\})|$ equals to the support of $P \cup \{x\}$. FP-growth saves all transactions into *FP-tree* which is a tree-like data structure [Han et al., 2000]. Each non-root node of the FP-tree contains a counter and is labeled with the name of a certain frequent item (frequent 1-itemset). In order to build the FP-tree, the FP-growth algorithm first detects frequent items and orders them in support ascending order. Frequent items of each transaction are then saved into FP-tree in *reverse* order as a path, by incrementing counters in existing nodes of the path and creating missing nodes with counters set to 1 (infrequent items are ignored, since they can’t belong to any frequent itemset). In that way, nodes closer to the root node correspond to more frequent items, and are more likely to be shared by many transactions, yielding a smaller FP-tree. In addition, nodes corresponding to the same item are linked into a chain with *node-links*, and the *item header table* holds a pointer to the first node of each such chain. This allows the FP-growth algorithm to quickly locate all nodes for a certain item. Figure 4.3 depicts a sample FP-tree data structure (the support threshold is 2 and frequent items are ordered in support ascending order $d < a < c < b$).

Transaction ID	Itemset
1	*abcde*
2	*abc*
3	*bcd*
4	*abc*
5	*ab*

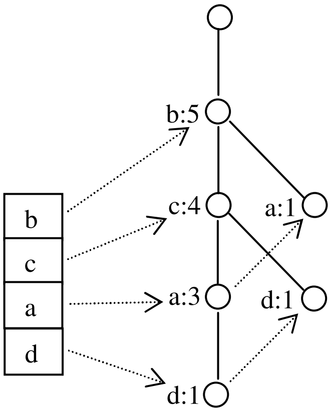

Figure 4.3 A sample FP-tree data structure

Note that the nature of transaction data determines whether Eclat or FP-growth is more efficient in terms of memory consumption. In some cases (e.g., see [Goethals 2004]) Eclat could consume less memory, while the results presented in this thesis suggest that the memory requirements of FP-growth are more modest for event log data. Unfortunately, both Eclat and FP-growth can't be employed for larger transaction databases which don't fit into main memory.

Although some techniques have been proposed for addressing this problem (e.g., the partitioning of the database), these techniques are often infeasible [Goethals 2004]. In the next section we will show that this problem is also relevant for event log data sets.

4.4. The nature of event log data and the motivation for developing new pattern mining algorithms

In order to cluster event log lines, event logs are viewed as categorical data sets in this thesis, and each event log line is considered to be a data point with words of the line serving as attribute values. In order to apply frequent itemset mining algorithms for event logs, event logs are viewed as transaction databases in this thesis, and the task of mining frequent event type patterns or frequent line patterns is formulated as the task of mining frequent itemsets. In the case of event type patterns, event types act as items, and in the case of line patterns, items are words from event log lines (the word positions are taken into account during the mining).

The nature of input data plays an important role when designing an efficient knowledge discovery algorithm. The experiments described in the research papers of this thesis [Vaarandi 2003; Vaarandi 2004] have revealed the following important properties of event log data:

- the number of items (or attribute values) in the data set can be quite large, especially when line patterns are mined from raw event logs; however, only few items (attribute values) are relatively frequent, and also, most items (attribute values) appear only few times in the data set,
- frequent itemsets may contain many items, which means that Apriori is not always adequate for processing event log data,
- there are often strong correlations between frequent items (attribute values), i.e., items (attribute values) occur together in the same event log slice or line many times in the data set.

In order to assess how well the Apriori, Eclat, and FP-growth algorithms are suited for mining frequent patterns from event logs, a number of experiments were conducted [Vaarandi 2004]. The experiment results indicate that all tested algorithms are not entirely suitable for discovering patterns from event logs – depth-first algorithms could face difficulties when they attempt to load the transaction database into main memory, Apriori has a poor performance, and for data sets containing larger frequent itemsets all algorithms are too slow. As discussed in section 4.2, existing data clustering algorithms are also inconvenient for processing event log data.

The following subsections will present an overview of efficient mining algorithms that address the shortcomings of existing algorithms.

4.5. A clustering algorithm for mining line patterns

This thesis proposes an algorithm for clustering event log lines which makes only a few passes over the data and is thus fast, and which detects clusters that are present in subspaces of the original data space. The algorithm relies on the special properties of event log data discussed in section 4.4, and uses the density based approach for clustering.

The data space is assumed to contain data points with categorical attributes, where each point represents a line from an event log data set, and the attributes of each point are the words from the corresponding line. The data space has n dimensions, where n is the maximum number of words per line in the data set. A *region S* is a subset of the data space, where certain attributes $i_1,...,i_k$ ($1 \leq k \leq n$) of all points that belong to S have identical values $v_1,...,v_k$: $x \in S$, $xi_1 = v_1$, ..., $xi_k = v_k$. We call the set $\{(i_1,v_1),...,(i_k,v_k)\}$ the set of *fixed attributes* of region S. If $k = 1$ (i.e., there is just one fixed attribute), the region is called *1-region*. A *dense region* is a region that contains at least N points, where N is the *support threshold value* given by the user.

The algorithm consists of three steps like the CACTUS algorithm [Ganti et al., 1999] – it first makes a pass over the data and builds a data summary, and then makes another pass to build cluster candidates, using the summary information collected before. As a final step, clusters are selected from the set of candidates.

During the first step of the algorithm (data summarization), the algorithm identifies all dense 1-regions. Since this step could require large amounts of memory (due to the large number of attribute values), the summary vector technique is employed for reducing its memory cost [Vaarandi 2003].

After dense 1-regions have been identified, the algorithm builds all cluster candidates during one pass. The data set is processed line by line, and when a line is found to belong to m dense 1-regions that have fixed attributes $(i_1,v_1),...,(i_m,v_m)$, then a region with the set of fixed attributes $\{(i_1,v_1),...,(i_m,v_m)\}$ becomes a cluster candidate. If the cluster candidate is not present in the candidate table, it will be inserted into the table with the support value 1, otherwise its support value will be incremented. In both cases, the line is assigned to the cluster candidate.

During the final step of the algorithm, the candidate table is inspected, and all regions with support values equal or greater than the support threshold value (i.e., regions that are guaranteed to be dense) are reported by the algorithm as clusters. Because of the definition of a region, each cluster corresponds to a certain line pattern, e.g., the cluster with the set of fixed attributes *{(1, 'Password'), (2, 'authentication'), (3, 'for'), (5, 'accepted')}* corresponds to the line pattern *Password authentication for * accepted*. Thus, the algorithm can report clusters in a concise way by just printing out line patterns, without reporting individual lines that belong to each cluster (the CLIQUE algorithm reports clusters in a similar manner [Agrawal et al., 1998]).

The algorithm described above has been implemented in a tool called Simple Logfile Clustering Tool (SLCT). Apart from identifying clusters, the tool can also report outlier lines, refine wildcard parts of cluster descriptions, filter and convert input lines with the help of regular expressions, etc. Further information about SLCT and the underlying algorithm can be found in SLCT online documentation and in one of the research papers of this thesis [Vaarandi 2003]. The paper also discusses the performance of SLCT and provides examples of detected patterns and anomalous event log lines.

Another recent paper [Stearley 2004] describes the use of SLCT for analyzing *syslog* log files – SLCT has been incorporated into the Sisyphus log analysis toolkit (developed at Sandia National Laboratories) and is employed for automated message typing. The paper also compares the performance and output of SLCT and Teiresias for several event log data sets (Teiresias is a well-known pattern discovery algorithm originally developed for bioinformatics [Rigoutsos and Floratos, 1998]), and concludes that SLCT is an efficient tool for detecting message types.

4.6. A frequent itemset mining algorithm for mining event type and line patterns

This thesis also proposes an efficient frequent itemset mining algorithm for event logs that combines some of the features of previously discussed algorithms and takes into account the properties of event log data. Since depth-first Eclat and FP-growth are inherently dependent on the amount of main memory, the proposed algorithm works in a breadth-first manner and employs the itemset trie data structure (see section 4.3). In order to avoid the weaknesses of Apriori, the algorithm uses several techniques for speeding up its work and reducing its memory consumption.

The mining of frequent items is the first step of any breadth-first algorithm which creates a base for further mining. Unfortunately, because the number of items can be very large, the memory cost of this step is often quite high. In order to overcome this problem, the algorithm employs the summary vector technique [Vaarandi 2004].

Eclat and FP-growth are fast not only because of their depth-first search strategy, but also because they load the transaction database from disk (or other secondary storage device) into main memory. In order to speed up its work in a similar way, the algorithm loads most frequently used transaction data into the memory-based cache. Note that unlike Eclat and FP-growth, the algorithm does not depend on the amount of main memory, since the amount of data stored to the cache is controlled by the user [Vaarandi 2004].

As discussed in section 4.3, the Apriori algorithm is not well suited for processing data sets which contain frequent *k*-itemsets for larger values of *k*, since the itemset trie could become very large, making the runtime and memory cost of the algorithm prohibitive. However, when there are many strong correlations between frequent items in transactions, many parts of the Apriori itemset trie are likely to contain information that is already present in other parts. The algorithm proposed in this thesis employs a special technique for reducing the size of the itemset trie, so that the trie would still represent all frequent itemsets.

Let $F = \{f_1,...,f_n\}$ be the set of all frequent items. We call the set $dep(f_i) = \{f_j \mid f_i \neq f_j, cover(\{f_i\}) \subseteq cover(\{f_j\})\}$ the *dependency set* of f_i, and say that an item *f_i has m dependencies* if $|dep(f_i)| = m$. A *dependency prefix of the item f_i* is the set $pr(f_i) = \{f_j \mid f_j \in dep(f_i), f_j < f_i\}$. A *dependency prefix of the itemset* $\{f_{i_1},\dots,f_{i_k}\}$ is the set $pr(\{f_{i_1},\dots,f_{i_k}\}) = \cup_{j=1}^{k} pr(f_{i_j})$.

The technique for reducing the size of the itemset trie can be summarized as follows – *if the itemset does not contain its dependency prefix, don't create a node in the trie for that itemset* (please see [Vaarandi 2004] for a detailed discussion how the itemset trie is built). Although the resulting trie is often much smaller than the Apriori itemset trie, all frequent itemsets can be easily derived from its nodes (see Appendix B for formal proofs).

The algorithm can be further optimized – if the trie reduction technique was not applied at node *N* for reducing the number of its child nodes, and node *M* is a child of *N*, then the siblings of *M* contain all necessary nodes for the creation of candidate child nodes for *M* in Apriori fashion. It is easy to see that with this optimization the algorithm is a generalization of Apriori – if at node *N* the algorithm discovers that the trie reduction technique is no longer effective, it switches to Apriori for the subtrie that starts from *N*, and if there are no frequent items that have dependencies, the algorithm switches to Apriori at the root node, i.e., it behaves like Apriori from the start.

The experiment results indicate that the trie reduction technique is efficient for event log data sets, and often significantly smaller itemset trie is produced than in the case of Apriori. The results also indicate that the algorithm performs quite well when compared to FP-growth, and outperforms it in several cases (please see [Vaarandi 2004] for a detailed discussion).

In order to implement the algorithm presented in this section, a tool called LogHound has been developed. The tool can be employed for mining frequent line patterns from raw event logs, but also for mining frequent event type patterns. For further information about LogHound and examples of detected patterns, the reader is referred to the research paper of this thesis [Vaarandi 2004] and the LogHound online documentation.

5. Conclusion

This thesis discusses the problems of event correlation and data mining in the context of event log analysis, and presents novel tools and techniques for addressing these problems. The thesis also provides an overview of related work on event logging, event correlation, and data mining for event logs. The main contributions of this thesis are the following:

- the development of Simple Event Correlator (SEC) that demonstrates the efficiency of a lightweight, platform independent, and open-source event correlator for monitoring event logs and processing event streams,
- the proposal of a novel data clustering algorithm for mining patterns from event logs,
- the proposal of a novel frequent itemset mining algorithm for mining frequent patterns from event logs.

Event correlation is one of the most prominent real-time event processing techniques today. It has received a lot of attention in the context of network fault management over the past decade, and is becoming increasingly important in other domains as well, including event log monitoring. A number of approaches have been proposed for event correlation, and a number of event correlation products are available on the market. Unfortunately, existing products are mostly expensive, platform-dependent, and heavyweight solutions that have complicated design, being therefore difficult to deploy and maintain, and requiring extensive user training. For these reasons, they are often unsuitable for employment in smaller IT systems and on network nodes with limited computing resources.

The SEC event correlator presented in this thesis demonstrates that a lightweight and platform independent event correlator with an open-source status can be an efficient tool for monitoring event logs and processing event streams. Furthermore, it can also be a serious alternative to heavyweight and platform-dependent proprietary solutions. SEC has been adopted by many institutions over the past few years, ranging from companies with relatively small IT systems to large corporations with global networks. It has been applied for a wide variety of event correlation tasks in the domains of network fault and performance management, intrusion detection, log file analysis, event warehousing, etc. SEC has also been successfully employed with many applications and OS platforms. A number of research papers, online tutorials, and other documents have been published that describe the use of SEC for solving various event correlation problems. The user feedback data presented in this thesis reveal that the major advantages of SEC over other solutions are its open-source nature and free download status, ease of configuration, flexibility, applicability for a wide range of event correlation tasks, and ability to run on multiple platforms.

Since event logs play a significant role in modern IT systems, the mining of patterns from event logs has been identified as an important system and network management task. Recently proposed mining approaches for accomplishing this task have often been based on some well-known algorithm for mining frequent itemsets, and they have focused on detecting frequent event type patterns. However, existing approaches have several shortcomings. Firstly, many of the proposed algorithms are variants of the Apriori algorithm which is inefficient for mining longer patterns. Secondly, recent research has concentrated on detecting frequent patterns, but the discovery of infrequent patterns is equally important, since this might reveal anomalous events that represent unexpected behavior of the system. Unfortunately, data clustering methods that can tackle this problem have seldom been applied for mining patterns from event logs. Thirdly, existing algorithms mostly focus on finding event type patterns, ignoring patterns of other sorts. In particular, the mining of line patterns provides the user a valuable insight into event logs, but this issue has received very little attention so far.

In order to address the problems described above, this thesis proposes novel data clustering and frequent itemset mining algorithms for mining patterns from event logs. During their work, the algorithms take into account the special properties of event log data that have been discussed in this thesis.

The data clustering algorithm proposed in this thesis has been designed for mining line patterns. It views event log lines as data points and clusters them, so that each regular cluster corresponds to a certain frequently occurring line pattern and the cluster of outliers contains infrequent lines that could represent previously unknown fault conditions, or other unexpected behavior of the system that deserves closer investigation. The algorithm has been implemented in a tool called SLCT, and the experiment results indicate that the algorithm works fast, consumes little memory, and is able to detect many interesting patterns. SLCT has also been incorporated into the Sisyphus log analysis toolkit developed at Sandia National Laboratories.

The frequent itemset mining algorithm proposed in this thesis has been designed for mining both event type and line patterns. The experiment results presented in this thesis suggest that none of the prominent Apriori, Eclat, and FP-growth frequent itemset mining algorithms is well-suited for processing event log data. In order to avoid dependency on the amount of main memory (the main weakness of Eclat and FP-growth), the proposed algorithm employs the breadth-first approach and the itemset trie data structure like Apriori, but uses special techniques for avoiding inherent weaknesses of Apriori. The algorithm has been implemented in a tool called LogHound, and the experiment results indicate that in many cases the algorithm works faster and consumes less memory than other algorithms.

There are several interesting research problems that were not investigated in this thesis. One such issue is the generation of regular expressions or SEC rules from SLCT and LogHound, so that the discovered knowledge can be made available for SEC (or other event log monitoring solution) with minimal overhead. Another open problem is the creation of open-source event correlation tools employing non-rule-based correlation approaches. Techniques for making depth-first frequent itemset mining algorithms less dependent on the amount of main memory also deserve closer investigation, since such techniques make depth-first algorithms much more convenient for processing event log data.

References

Charu C. Aggarwal, Cecilia Procopiuc, Joel L. Wolf, Philip S. Yu, and Jong Soo Park. 1999. Fast Algorithms for Projected Clustering. Proceedings of the ACM SIGMOD International Conference on Management of Data, pp. 61-72.

Rakesh Agrawal, Johannes Gehrke, Dimitrios Gunopulos, and Prabhakar Raghavan. 1998. Automatic Subspace Clustering of High Dimensional Data for Data Mining Applications. Proceedings of the ACM SIGMOD International Conference on Management of Data, pp. 94-105.

Rakesh Agrawal and Ramakrishnan Srikant. 1994. Fast Algorithms for Mining Association Rules. Proceedings of the 20th International Conference on Very Large Data Bases, pp. 478-499.

Roberto J. Bayardo Jr. 1998. Efficiently Mining Long Patterns from Databases. Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data, pp. 85-93.

Pavel Berkhin. 2002. Survey of Clustering Data Mining Techniques. http://citeseer.nj.nec.com/berkhin02survey.html.

Ferenc Bodon. 2003. A fast APRIORI implementation. Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations.

Christian Borgelt. 2003. Efficient Implementations of Apriori and Eclat. Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations.

Jim Brown. 2003. Working with SEC – the Simple Event Correlator. http://sixshooter.v6.thrupoint.net/SEC-examples/article.html

L. Burns, J. L. Hellerstein, S. Ma, C. S. Perng, D. A. Rabenhorst, and D. Taylor. 2001. A Systematic Approach to Discovering Correlation Rules For Event Management. Proceedings of the 7th IFIP/IEEE International Symposium on Integrated Network Management, pp. 345-359.

Nathan Campi. 2005. Central Loghost Mini-HOWTO. http://www.campin.net/newlogcheck.html

Hugh R. Casey. 2001. The Simple Event Monitor, A Tool for Network Management. MSc thesis, University of Colorado.

Christopher Dillis. 2003. IDS Event Correlation with SEC – The Simple Event Correlator. http://www.giac.org/practical/GCIA/Christopher_Dillis_GCIA.pdf, Technical Report, SANS Institute.

Charles L. Forgy. 1982. Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem. Artificial Intelligence 19(1982), pp. 17-37.

P. Froehlich, W. Nejdl, M. Schroeder, C. V. Damasio, L. M. Pereira. 2002. Using Extended Logic Programming for Alarm-Correlation in Cellular Phone Networks. Applied Intelligence 17(2), pp. 187-202.

Venkatesh Ganti, Johannes Gehrke, and Raghu Ramakrishnan. 1999. CACTUS – Clustering Categorical Data Using Summaries. Proceedings of the 5th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 73-83.

Bart Goethals. 2004. Memory issues in frequent itemset mining. Proceedings of the 2004 ACM Symposium on Applied Computing, pp. 530-534.

Sanjay Goil, Harsha Nagesh, and Alok Choudhary. 1999. MAFIA: Efficient and Scalable Subspace Clustering for Very Large Data Sets. Technical Report No. CPDC-TR-9906-010, Northwestern University.

Dan Gorton. 2003. Extending Intrusion Detection with Alert Correlation and Intrusion Tolerance. Licentiate thesis, Chalmers University of Technology.

Boris Gruschke. 1998. Integrated Event Management: Event Correlation using Dependency Graphs. Proceedings of the 9th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management, pp. 130-141.

Sudipto Guha, Rajeev Rastogi, and Kyuseok Shim. 2000. ROCK: A Robust Clustering Algorithm for Categorical Attributes. Information Systems 25(5), pp. 345-366.

Jiawei Han, Jian Pei, and Yiwen Yin. 2000. Mining Frequent Patterns without Candidate Generation. Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, pp. 1-12.

David Hand, Heikki Mannila, and Padhraic Smyth. 2001. Principles of Data Mining. The MIT Press, ISBN: 0-262-08290-X.

Stephen E. Hansen and E. Todd Atkins. 1993. Automated System Monitoring and Notification With Swatch. Proceedings of USENIX 7th System Administration Conference, pp. 145-152.

Alexander Hinneburg and Daniel A. Keim. 1999. Optimal Grid-Clustering: Towards Breaking the Curse of Dimensionality in High-Dimensional Clustering. Proceedings of the 25th International Conference on Very Large Data Bases, pp. 506-517.

A. K. Jain, M. N. Murty, and P. J. Flynn. 1999. Data Clustering: a Review. ACM Computing Surveys 31(3), pp. 264-323.

G. Jakobson and M. Weissman. 1995. Real-time telecommunication network management: Extending event correlation with temporal constraints. Proceedings of the 4th International Symposium on Integrated Network Management, pp. 290-301.

G. Jakobson, M. Weissman, L. Brenner, C. Lafond, C. Matheus. 2000. GRACE: Building Next Generation Event Correlation Services. Proceedings of the 7th IEEE/IFIP Network Operations and Management Symposium, pp. 701-714.

Klaus Julisch. 2003. Clustering intrusion detection alarms to support root cause analysis. ACM Transactions on Information and System Security 6(4), pp. 443-471.

Mika Klemettinen. 1999. A Knowledge Discovery Methodology for Telecommunication Network Alarm Databases. PhD thesis, University of Helsinki.

Wolfgang Ley and Uwe Ellerman. 1996. logsurfer(1) and logsurfer.conf(4) manual pages. http://www.cert.dfn.de/eng/logsurf/

C. Lonvick. 2001. The BSD syslog protocol. RFC3164.

Sheng Ma and Joseph L. Hellerstein. 2000. Mining Partially Periodic Event Patterns with Unknown Periods. Proceedings of the 16th International Conference on Data Engineering, pp. 205-214.

H. Mannila, H. Toivonen, and A. I. Verkamo. 1997. Discovery of frequent episodes in event sequences. Data Mining and Knowledge Discovery 1(3), pp. 259-289.

Francois Meehan. 2005. SNMP Trap Handling with Nagios. Sys Admin 14(3), pp. 41-44.

Dilmar Malheiros Meira. 1997. A Model For Alarm Correlation in Telecommunication Networks. PhD thesis, Federal Unversity of Minas Gerais.

Benjamin Morin and Herve Debar. 2003. Correlation of Intrusion Symptoms: an Application of Chronicles. Proceedings of the 6th Symposium on Recent Advances in Intrusion Detection, pp. 94-112.

D. New and M. Rose. 2001. Reliable Delivery for syslog. RFC3195.

Raymond T. Ng and Jiawei Han. 1994. Efficient and Effective Clustering Methods for Spatial Data Mining. Proceedings of the 20th International Conference on Very Large Data Bases, pp. 144-155.

Jian Pei, Jiawei Han, Behzad Mortazavi-asl, and Hua Zhu. 2000. Mining Access Patterns Efficiently from Web Logs. Proceedings of the 4th Pacific-Asia Conference on Knowledge Discovery and Data Mining, pp. 396-407.

Elaine Rich and Kevin Knight. 1991. Artificial Intelligence, 2nd edition. McGraw-Hill, ISBN 0-07-052263-4.

I. Rigoutsos and A. Floratos. 1998. Combinatorial pattern discovery in biological sequences: the TEIRESIAS algorithm. Bioinformatics 14(1), pp. 55-67.

Martin Roesch. 1999. Snort – Lightweight Intrusion Detection for Networks. Proceedings of USENIX 13th System Administration Conference, pp. 229-238.

John P. Rouillard. 2004. Real-time Logfile Analysis Using the Simple Event Correlator (SEC). Proceedings of USENIX 18th System Administration Conference, pp. 133-149.

Chris Sawall. 2004. Auditing a Syslog Server Running Fedora Core 1. http://www.giac.org/practical/GSNA/Chris_Sawall_GSNA.pdf, Technical Report, SANS Institute.

Jaideep Srivastava, Robert Cooley, Mukund Deshpande, and Pang-Ning Tan. 2000. Web Usage Mining: Discovery and Applications of Usage Patterns from Web Data. ACM SIGKDD Explorations 1(2), pp. 12-23.

Stuart Staniford, James A. Hoagland, Joseph A. McAlerney. 2002. Practical Automated Detection of Stealthy Portscans. Journal of Computer Security 10 (1-2), pp. 105-136.

Jon Stearley. 2004. Towards Informatic Analysis of Syslogs. Proceedings of the 2004 IEEE International Conference on Cluster Computing.

M. Steinder and A. S. Sethi. 2002. End-to-end Service Failure Diagnosis Using Belief Networks. Proceedings of the 8th IEEE/IFIP Network Operations and Management Symposium, pp. 375-390.

James Turnbull. 2005. Hardening Linux. Apress, ISBN: 1-59059-444-4.

Risto Vaarandi. 2002. Platform Independent Tool for Local Event Correlation. Acta Cybernetica 15(4), pp. 705-723.

Risto Vaarandi. 2002. Platform Independent Event Correlation Tool for Network Management. Proceedings of the 8th IEEE/IFIP Network Operations and Management Symposium, pp. 907-910.

Risto Vaarandi. 2002. SEC – a Lightweight Event Correlation Tool. Proceedings of the 2002 IEEE Workshop on IP Operations and Management, pp. 111-115.

Risto Vaarandi. 2003. A Data Clustering Algorithm for Mining Patterns From Event Logs. Proceedings of the 2003 IEEE Workshop on IP Operations and Management, pp. 119-126.

Risto Vaarandi. 2004. A Breadth-First Algorithm for Mining Frequent Patterns from Event Logs. Proceedings of the 2004 IFIP International Conference on Intelligence in Communication Systems, LNCS Vol. 3283, pp. 293-308.

Hermann Wietgrefe. 2002. Investigation and Practical Assessment of Alarm Correlation Methods for the Use in GSM Access Networks. Proceedings of the 8th IEEE/IFIP Network Operations and Management Symposium, pp. 391-404.

Hermann Wietgrefe, Klaus-Dieter Tuchs, Klaus Jobmann, Guido Carls, Peter Froehlich, Wolfgang Nejdl, Sebastian Steinfeld. 1997. Using Neural Networks for Alarm Correlation in Cellular Phone Networks. Proceedings of the International Workshop on Applications of Neural Networks in Telecommunications, pp. 248-255.

S. A. Yemini, S. Kliger, E. Mozes, Y. Yemini, and D. Ohsie. 1996. High speed and robust event correlation. IEEE Communications Magazine 34(5), pp. 82-90.

Mohammed J. Zaki. 2000. Scalable Algorithms for Association Mining. IEEE Transactions on Knowledge and Data Engineering 12(3), pp. 372-390.

Qingguo Zheng, Ke Xu, Weifeng Lv, and Shilong Ma. 2002. Intelligent Search of Correlated Alarms from Database Containing Noise Data. Proceedings of the 8th IEEE/IFIP Network Operations and Management Symposium, pp. 405-419.

Part II - Research papers

Appendix A: SEC application experience

Type of the company	**Location**	**Description of the managed system**	**How SEC is applied**	**Advantages of SEC over other event correlation systems**
Banking card authorization center	Europe	30 servers, routers, and firewalls	Event correlation engine for NMS and IDS, log file monitoring and system monitoring. An important application of SEC is fraud detection.	Straightforward, easy, and transparent configuration and rule definition system.
Technology based marketing agency	US	600 nodes across US and UK	Gather and correlate service issues from Cisco CSS content switches.	Power and control in the amount you choose.
Financial institution	US	6000 workstations, 400 servers, 350 switches, 250 routers (distributed over US plus 5 other countries)	Used as a central event correlation engine for HP OpenView NNM. Also used for central monitoring of *syslog* messages from Cisco devices.	More flexible and customizable than other event correlation systems.
Retail sales of consumer electronics	US	8000 managed nodes; the company WAN covers continental US, Alaska, Hawaii, and US territories	Network management with HP OpenView, log file monitoring, dynamic webpage generation, etc.	SEC provides a low cost and efficient method to plug in event correlation and event management into HP OpenView.

Type of the company	**Location**	**Description of the managed system**	**How SEC is applied**	**Advantages of SEC over other event correlation systems**
Telecommunications Carrier/ Provider	US	One of the largest international networks in the world	Log file monitoring (collect and interpret alarms at call centers, and send a notification to a national support team).	Good level of control over monitoring triggers.
Network consulting	Global (more than 30 offices in US, Europe, and Asia)	SEC is used in the US network of a major European car manufacturer (100 routers, 300 switches)	Used as a correlation engine for Cisco DFM platform and for Snort IDS.	Provides event correlation without significant programming resources, runs on multiple platforms, integrates well with external scripting languages.
Software development, IT consulting and services	Global (offices in Europe, US, Asia, Australia, South-America)	Global network, spread worldwide across the globe	Used as a prototype for event correlation experiments.	Free download status.
Top wireless telephone company	US	Several thousand UNIX servers	Log file monitoring located on each server feeding a central Nagios monitoring system.	Free and open source, Perl-based and thus cross-platform, capable of complex correlations.
Managed security and hosting provider	Pennsylvania, US	About 200 servers	SEC is used on about 150 servers running Bastille Linux for real-time log file monitoring and application against *iptables* software firewall.	Support from the author and user community is available and prompt, ease of expansion – creating new rules for new situations is relatively easy.

Type of the company	Location	Description of the managed system	How SEC is applied	Advantages of SEC over other event correlation systems
ISP	Germany	500 routers and switches, 300 servers, 500 other active components, 5000 customer routers which are monitored (the company operates in most parts of Germany)	HP OpenView and SNMPTT are used for organizing incoming traps, and SEC will be used in both environments for correlating the traps. SEC will also be used for *syslog* log file and Snort IDS monitoring. (The company is still evaluating SEC.)	Configuration files are ASCII-like and can therefore be modified without any special software. SEC is also written in Perl and is extremely flexible - you can create all correlation combinations you need.
Government organization (Patents and Trade Marks)	Australia	1500 end nodes, 150 servers, 10 routers, 60 switches, 6 Frame Relay PVCs	Event correlation for Cisco ISDN call accounting (from messages sent by routers to *syslog* servers), SNMP trap compression, passive service check result generation for Nagios.	SEC is well documented, is actively developed, has a productive mailing list, is more reliable than Swatch. It is easy to integrate SEC with other applications.
Internet media company	NYC, US	About 100 servers with a dozen supporting routers, switches, and other devices	The sending of e-mail pages on critical firewall events.	SEC is GPL licensed, flexible, highly configurable, platform independent, and the documentation is sufficiently detailed.
Manu-facturer of medical equipment and software	US	Global network	SEC is used in company products for event correlation (tracking system availability).	Support for regular expressions, flexibility of maintaining variables and reporting in different ways.

Type of the company	**Location**	**Description of the managed system**	**How SEC is applied**	**Advantages of SEC over other event correlation systems**
IT security manage-ment company	Luxembourg	3 servers and 1 firewall (SEC is used for company clients who have larger infrastructures of 100+ servers)	SEC is used for event correlation on central log servers integrated with company's security solutions.	Flexibility.
E-mail service provider	Colorado, US	300 computers, 15 managed switches, 5 routers, 4 UPS's, 4 air conditioners	Monitoring over 300 devices. SEC is used to monitor AIDE, Snort IDS, Cisco, and HP OpenView ITO logs.	Strong user support group, SEC is written in Perl which allows portability and extensibility, SEC's feature set is rich and mature.
A fire department (24 fire vehicles, 90 square miles of rugged terrain, and 5 of Colorado's largest wildfires in the past 8 years)	Colorado, US	8 computers, 2 routers, 1 weather station	SEC correlates events from all weather inputs and alerts firefighters and support personnel well ahead of impending storms which cause lightning-strike fires, flooding, and wildfire/RED-FLAG conditions.	Strong user support group, SEC is written in Perl which allows portability and extensibility, SEC's feature set is rich and mature.
Financial institution	US	More than 500 routers, 750 UNIX servers, and 500 Windows servers	Log monitoring for security operations.	SEC is cost-efficient, flexible, self-contained, and provides event correlation functionality not otherwise found for security operations.

Type of the company	**Location**	**Description of the managed system**	**How SEC is applied**	**Advantages of SEC over other event correlation systems**
Telecom company	North America	Large national network carrying both internal traffic as well as bandwidth resold for external customer traffic under a subsidiary company	The company has employed SEC for the past 2.5 years and the number of implementations has grown steadily. SEC is used as an event correlation engine for HP OpenView NNM, as a log file monitoring solution, and as an event warehousing solution in an Oracle database.	The company originally considered several commercial event correlation systems, but SEC was found to be more robust and cost-efficient. SEC is easy to maintain and configure, and it is able to handle a variety of events - video traffic events, cable modem traffic events, and layer 2 events (ATM, SONET, Ethernet).
IT systems integrator and managed service provider	Germany	40 managed nodes at one customer site	Primary function of SEC is the correlation of security events (Cisco PIX and Checkpoint firewalls, UNIX, *syslog* events received from Windows servers). SEC is also used for monitoring purposes (event correlator for Nagios; monitoring routers, switches and servers).	The way that contexts are implemented in SEC, the ease of integration with existing solutions, the ease of understanding how the tool works through excellent testing and debugging options.

Type of the company	**Location**	**Description of the managed system**	**How SEC is applied**	**Advantages of SEC over other event correlation systems**
Lottery company	Germany	About 25 servers and an Oracle database	Event correlation for an alarming system.	Simple, cost-efficient, and easily customizable.
Financial institution	Slovakia	About 300 Cisco routers and switches, and about 300 servers	Log file monitoring for the central *syslog-ng* server, monitoring for web proxy error logs.	SEC is free, easy to install and configure, and flexible (can be used in a variety of ways for a variety of applications). It is also fast and supports the Perl dialect of the regular expression language.
ICT services and solutions provider	Canada	About 850 nodes	Event correlation for HP OpenView NNM.	Varied capabilities of SEC and good documentation.

Appendix B: Formal proofs

Lemma 1. If $pr(\{f_{i_1},...,f_{i_k}\}) \subseteq \{f_{i_1},...,f_{i_k}\}$, then $pr(\{f_{i_1},...,f_{i_{k-1}}\}) \subseteq \{f_{i_1},...,f_{i_{k-1}}\}$.
Proof: Follows from $pr(\{f_{i_1},...,f_{i_{k-1}}\}) \subseteq pr(\{f_{i_1},...,f_{i_k}\})$ and $\forall f \in pr(\{f_{i_1},..,f_{i_{k-1}}\})$, $f < f_{i_{k-1}} < f_{i_k}$.

Lemma 2. If $a, b, c \in F$, $a \in pr(b)$, $b \in pr(c)$, then $a \in pr(c)$.
Proof: Directly follows from the definition of the item dependency prefix.

Lemma 3. $pr(X \setminus pr(X)) = pr(X)$.
Proof: Note that since $(X \setminus pr(X)) \subseteq X$, then also $pr(X \setminus pr(X)) \subseteq pr(X)$. Let $a \in pr(X)$. According to Lemma 2, $\exists b \in (X \setminus pr(X))$ so that $a \in pr(b)$, i.e., $pr(X) \subseteq pr(X \setminus pr(X))$. This means that $pr(X \setminus pr(X)) = pr(X)$.

Lemma 4. $supp(X \setminus pr(X)) = supp(X)$.
Proof: Note that according to the definition of the itemset dependency prefix $supp(Y) = supp(Y \cup pr(Y))$, and for any sets $(A \setminus B) \cup B = A \cup B$. Then according to Lemma 3 $supp(X \setminus pr(X)) = supp((X \setminus pr(X)) \cup pr(X \setminus pr(X))) = supp((X \setminus pr(X)) \cup pr(X)) = supp(X \cup pr(X)) = supp(X)$.

Lemma 5. The non-root nodes of the itemset trie constructed by the algorithm from section 4.6 correspond to all frequent itemsets that contain their dependency prefixes.
Proof: By its definition, the algorithm does not create trie nodes for frequent itemsets that do not contain their dependency prefixes, i.e., all non-root nodes of the trie represent frequent itemsets that contain their dependency prefixes. Also, if $\{f_{i_1},...,f_{i_k}\}$ is a frequent itemset and $pr(\{f_{i_1},...,f_{i_k}\}) \subseteq \{f_{i_1},...,f_{i_k}\}$, then according to Lemma 1 $pr(\{f_{i_1},...,f_{i_{k-1}}\}) \subseteq \{f_{i_1},...,f_{i_{k-1}}\}$, $pr(\{f_{i_1},...,f_{i_{k-2}}\}) \subseteq \{f_{i_1},...,f_{i_{k-2}}\}$, ..., $pr(\{f_{i_1}\}) \subseteq \{f_{i_1}\}$ (i.e., $pr(\{f_{i_1}\}) = \varnothing$). By its definition, the algorithm inserts nodes $node(\{f_{i_1}\})$, $node(\{f_{i_1}, f_{i_2}\})$, ..., $node(\{f_{i_1},...,f_{i_k}\})$ into the itemset trie, i.e., the trie contains a node for any frequent itemset that contains its dependency prefix.

Lemma 6. The itemset trie constructed by the algorithm from section 4.6 represents all frequent itemsets, and there is a unique node for deriving each frequent itemset.
Proof: Let X be a frequent itemset, and let $Y = X \cup pr(X)$. According to Lemma 5, $node(Y)$ is present in the itemset trie. Since $(Y \setminus pr(Y)) \subseteq X \subseteq Y$, X can be derived from $node(Y)$, and according to Lemma 4, $supp(Y \setminus pr(Y)) = supp(X) = supp(Y)$. The $node(Y)$ is unique, since if $(Z \setminus pr(Z)) \subseteq X \subseteq Z$ and $pr(Z) \subseteq Z$, then according to Lemma 3 $pr(Z \setminus pr(Z)) = pr(X) = pr(Z)$. On the other hand, from $(Z \setminus pr(Z)) \subseteq X \subseteq Z$ it follows that $((Z \setminus pr(Z)) \cup pr(X)) \subseteq (X \cup pr(X)) \subseteq (Z \cup pr(X))$. However, since $pr(X) = pr(Z)$, then $((Z \setminus pr(Z)) \cup pr(Z)) \subseteq (X \cup pr(X)) \subseteq (Z \cup pr(Z))$, and since $pr(Z) \subseteq Z$, then $Z \subseteq (X \cup pr(X)) \subseteq Z$. In other words, $Y = Z$.

Appendix C: Product web pages

BMC Patrol – http://www.bmc.com/

CiscoWorks – http://www.cisco.com/

CLIPS – http://www.ghg.net/clips/CLIPS.html

HP ECS – http://www.managementsoftware.hp.com/products/ecs/index.html

HP OpenView – http://www.openview.hp.com/

LOGEC – http://www.logec.com/

LogHound – http://kodu.neti.ee/~risto/loghound/

Logsurfer – http://www.cert.dfn.de/eng/logsurf/

MySQL – http://www.mysql.com/

Nagios – http://www.nagios.org/

NerveCenter – http://www.open.com/products/nervecenter.jsp

Net-SNMP – http://www.net-snmp.org/

NetCool – http://www.micromuse.com/

Oracle – http://www.oracle.com/

RuleCore – http://www.rulecore.com/

SEC – http://simple-evcorr.sourceforge.net/

Sisyphus – http://www.cs.sandia.gov/sisyphus/

SLCT – http://kodu.neti.ee/~risto/slct/

SMARTS – http://www.smarts.com/

SNMPTT – http://snmptt.sourceforge.net/

Snort – http://www.snort.org/

Swatch – http://swatch.sourceforge.net/

Syslog-ng – http://www.balabit.com/products/syslog_ng/

Tivoli – http://www.tivoli.com/

Curriculum Vitae

1. Personal data

Name: Risto Vaarandi
Date and place of birth: September 18, 1972, Tallinn, Estonia
Citizenship: Estonian
Marital status: married
Children: a daughter

2. Contact data

Address: Tornimäe 2, Tallinn 15010, Estonia
Telephone: +372 6655451
E-mail: risto dot vaarandi at seb dot ee

3. Education

University of Tartu, 1997, Master of Computer Science
Tallinn High School No. 21, 1990, high school diploma

4. Languages

Estonian – first language
English – fluent
Russian – reading
Finnish – reading

5. Complementary training

-

6. Career

Since April 1998: SEB Eesti Ühispank, IT development engineer
March 1997 – April 1998: AS Estpak Data, network administrator
August 1994 – March 1997: Tartu County Government, IT specialist

7. Previously defended theses

“Distributed File Systems”, MSc thesis, University of Tartu, 1997

8. Awards

-

9. Research interests

Event correlation in monitoring and management systems
Data mining algorithms for analyzing event logs
Network management and intrusion detection
System and application monitoring techniques

10. Other projects

-

Elulookirjeldus

1. Isikuandmed

Nimi: Risto Vaarandi
Sünniaeg ja -koht: 18. september 1972. a., Tallinn, Eesti
Kodakondsus: Eesti
Perekonnaseis: abielus
Lapsed: tütar

2. Kontaktandmed

Aadress: Tornimäe 2, Tallinn 15010, Eesti
Telefon: 6655451
E-mail: risto dot vaarandi at seb dot ee

3. Hariduskäik

Tartu Ülikool, 1997, arvutiteaduse magister
Tallinna 21. Keskkool, 1990, keskharidus

4. Keelteoskus

eesti – emakeel
inglise – kõrgtase
vene – algtase
soome – algtase

5. Täiendõpe

-

6. Teenistuskäik

Alates aprillist 1998: SEB Eesti Ühispank, IT arendusinsener
Märts 1997 – aprill 1998: AS Estpak Data, võrguadministraator
August 1994 – märts 1997: Tartu Maavalitsus, IT spetsialist

7. Kaitstud lõputööd

“Hajusad failisüsteemid”, magistritöö, Tartu Ülikool, 1997

8. Auhinnad

-

9. Teadustöö põhisuunad

Sündmuste korrelatsioon süsteemide monitooringuks ja halduseks
Andmekaevandamisalgoritmid logide analüüsimiseks
Võrguhaldus ja ründetuvastus
Süsteemide ja rakenduste monitoorimise tehnikad

10. Teised uurimisprojektid

-

Selected Publications

1. Risto Vaarandi. 2002. Platform Independent Tool for Local Event Correlation. Acta Cybernetica 15(4), pp. 705-723.
2. Risto Vaarandi. 2002. Platform Independent Event Correlation Tool for Network Management. Proceedings of the 8th IEEE/IFIP Network Operations and Management Symposium, pp. 907-910.
3. Risto Vaarandi. 2002. SEC – a Lightweight Event Correlation Tool. Proceedings of the 2002 IEEE Workshop on IP Operations and Management, pp. 111-115.
4. Risto Vaarandi. 2003. A Data Clustering Algorithm for Mining Patterns From Event Logs. Proceedings of the 2003 IEEE Workshop on IP Operations and Management, pp. 119-126.
5. Risto Vaarandi. 2004. A Breadth-First Algorithm for Mining Frequent Patterns from Event Logs. Proceedings of the 2004 IFIP International Conference on Intelligence in Communication Systems, LNCS Vol. 3283, pp. 293-308.