2012 IEEE 36th International Conference on Computer Software and Applications

Domain Independent Event Analysis for Log Data Reduction

Theodoros Kalamatianos and Kostas Kontogiannis
National Technical University of Athens
Dept. of Electrical and Computer Engineering
Athens, 15780, Greece
thkala@softlab.ntua.gr, kkontog@softlab.ntua.gr

Peter Matthews
CA Labs
Computer Associates Technologies
Ditton Park, UK
peter.matthews@ca.com

***Abstract*—Analyzing the run time behavior of large software systems is a difficult and challenging task. Log analysis has been proposed as a possible solution. However, such an analysis poses unique challenges, mostly due to the volume and diversity of the logged data that is collected, thus making this analysis often intractable for practical purposes. In this paper, we present a log analysis technique that aims to compute a smaller, compared to the original, collection of events that relate to a given analysis objective. The technique is based on computing a similarity score between the logged events and a collection of significant events that we refer to as beacons. The major novelties of the proposed technique are that it is domain independent and that it does not require the use of a pre-existing training data set. The technique has been evaluated against the DARPA Intrusion Detection Evaluation 1999 and the KDD 1999 data sets with promising results.**

***Keywords*-Software engineering, dynamic analysis, software maintenance, system understanding, log analysis, log reduction**

I. Introduction

Large software systems consist of many interconnected components emitting a wealth of information in the form of event logs. However, even for medium sized systems, these logs contain a high volume of data that makes the analysis intractable when attempting complex tasks, such as root cause analysis, understanding the operation of the system or investigating suspicious behavior. For many applications, it would be useful to reduce, if possible, the size of the logged data to a smaller subset which can then be examined and processed in a more tractable manner.

In this paper, we present an approach that aims to reduce the volume of the logged data that is emitted by the logging infrastructure of a software application. The motivation behind log reduction is that we would like to be able to focus the attention of an operator or an automated process to small parts of the logged data that bear a high likelihood of relating to suspicious or significant system events (i.e. the beacons). Beacon events can be selected by an automated process that looks for outlier or unusual events, by a hypothesis generation process (e.g. Goal or Anti-goal models), or by operators that audit the system logs. In this respect, the log reduction can be considered a first line of attack when one has to examine large volumes of logged data due to a failure or a symptom. The first notable point of the proposed approach is that it is domain independent, in the sense that it does not utilize domain knowledge nor does it treat different types of events or attributes in a preferential manner while computing a similarity score. This makes it readily suitable for collections of logged data that do not conform to the same schema or format. The second notable advantage of the approach is that it does not require a pre-existing training data set, a requirement that limits the use of most current approaches that are based on machine-learning methods.

The proposed log reduction process is composed of four main steps. In the first step, a collection of significant or suspicious log entries is selected, either by the user or by an automated event selection process, as the system operates. This selection may be based on a dictionary of known suspicious attribute-value pairs (e.g. the IP address of a known malicious client that attempts to connect to a server) or on more complex methods. For example, Goal and Anti-goal models have already been used to denote patterns of events to assist on root cause analysis diagnosis, intrusion detection and log filtering [1], [2]. In the second step, logged events and user-selected beacon events are fed to a log modeling pre-processing phase. The logged data is transformed, represented and stored as JSON objects. In the third step of the process, an event correlation algorithm computes similarity values between the beacon events and the logged events. The fourth step of the process focuses on selecting those logged events that present a high similarity to the beacon events.

This paper is organized as follows: Section II presents related work. Section III presents the log filtering process, while Section IV presents implementation considerations. Section V presents experimental results obtained from the DARPA Intrusion Detection Evaluation 1999 [3] data set and the KDD 1999 [4] log data set. Finally, Section VI concludes the paper and discusses future work.

II. Related work

Dynamic program analysis has been extensively used to understand the behavior of software systems. Bruegge et al. [5] proposed a framework to support dynamic analysis by source code instrumentation of systems written in C/C++. K. Koskimies et al. [6] proposed the tool SCED, for modeling dynamic properties of object oriented systems. Both of the above tools assume that access to the source code is available, which might not always be the case. Profiling and debugging is another technique used in dynamic program analysis. This technique utilizes interfaces provided by modern development environments [7] to facilitate runtime data collection. Mancoridis et al. [8] propose an approach, combining dynamic and static analysis, to map use-cases to specific sections of the source code. However this approach could result in limitations such as performance degradation, and it only works with programs executing within the same process space.

Complex Event Processing (CEP) is an event processing concept that deals with techniques for processing multiple events from many diverse sources with the goal of identifying the meaningful events within large data sets of collected events. In [9] and [10] the challenges and the themes of CEP as these are applied in large software systems are presented. More information on this emerging field can be found on the web site of the Complex Event Processing Community [11].

In the area of data set filtering, in [12] two data filtering and noise reduction techniques are discussed. The first is based on multiple-partitioning filtering while the second is based on iterative partitioning filtering. In [13] a technique to identify important event features to be used for dynamic analysis and, in particular, intrusion detection is presented. The reduced feature sets allow for more tractable analysis to be performed. The difference from our work is that the work in [13] is fine tuned for intrusion detection and aims to reduce features as opposed to events. In [14] the Enhanced Support Vector Decision Function technique is used for selecting important features in log entries to support intrusion detection analysis. In [15] a technique that allows for the discovery of processes by analyzing event logs is proposed. Finally, in [16] a log analysis technique has been used to evaluate the evolution of business models by comparing known business model templates to actual models.

III. Event filtering

Current software systems produce massive amounts of log events. While there are several techniques that can extract useful information from the raw logs, these are usually computationally expensive. In this section, we present a log filtering technique that allows for log reduction, by considering a collection of beacon events that are significant or raise interest to the system operator or a monitoring process.

A. Log modeling and preprocessing

Modern monitoring systems generate and store event logs in a variety of formats. Therefore, it becomes necessary to ensure their availability in a form that is suitable for further analysis. More specifically, let $E = \{e_1, e_2, \dots, e_n\}$ be the set of input events, where e_i the i^{th} event and n the number of events, and let $A = \{a_1, a_2, \dots a_{n_a}\}$ be the set of n_a discrete attributes in the event set E. Also let each event e_i be defined as a set of attribute-value pairs: $e_i = \{\langle a_j, ev_{j,i}\rangle \mid ev_{j,i}$ *the value of* a_j *for the event* e_i*, or* null *if unset*$, 1 \leq j \leq n_a\}$, $1 \leq i \leq n$, and let V_j be the set of discrete values for the attribute a_j in the input set E: $V_j = \{v \mid \forall v \exists i : ev_{j,i} = v, \forall i\ ev_{j,i} \in V_j\}$.

The preprocessing phase is composed of three steps: In the first step, each event e_i, $1 \leq i \leq n$ in the data set with m attributes $a_1, \dots a_m$, having corresponding values $v_{1,i}, \dots v_{m,i}$ is represented by a single XML element x_i and its descendants. In the second step, the XML entry is converted to a JSON-based format, where each event is represented by a single JSON object that directly reflects the hierarchy of its components. Finally, in the third step of the preprocessing phase, each JSON object is inserted into MongoDB [17], a schema-agnostic document-oriented database. Such databases are more suited than relational databases when the data model in use is not fully known, allowing for arbitrary JSON objects to be stored and retrieved in a single operation. They also allow the event schema to be updated implicitly when additional attributes appear in a received event.

B. Event similarity evaluation

The core of the system is the algorithm that determines the degree of similarity between each input event e_i and a set of beacon events B. In simple terms, the beacon events are used as a model that is considered to be representative of those log entries that are of interest or raise suspicion to a user or to an auditing process. Each event from the running systems' log data input set is separately compared to each beacon event b_k and a similarity measure is computed. The average of these similarity measures provides a metric that can be used to determine the correlation of the input event with the beacon event set as a whole.

The similarity measure $esm_{i,k}$ between a beacon event b_k and an input event e_i is computed using a layered approach. Each attribute a_j is examined independently and the corresponding values are compared using a combination of string-based similarity functions to produce a value correlation estimator $vsm_{j,i,k}$. In order to increase the selectivity of the system, a coefficient $c_{j,i,k}$ is computed using the frequencies of each attribute value in the beacon events and in the general event population. The purpose of this contribution coefficient is to emphasize the values that are prevalent in the beacon events, while reducing the impact of any values that are common enough in the input set to be of relatively limited use as a distinguishing feature. The product of the estimator $vsm_{j,i,k}$ and the coefficient $c_{j,i,k}$ is entered into a weighted average calculation to generate the $esm_{i,k}$ metric. The overall event similarity evaluation process is composed of five steps that are discussed in detail below.

1) Beacon event selection: The user of the system manually selects a set of beacon events B that are representative of the scenario of interest and will be used as the comparison basis in the reduction process. More specifically, let $B = \{b_1, b_2, \dots b_{n_b}\} = \{b_k : b_k \in E, 1 \leq k \leq n_b\}$, where n_b the number of selected beacon events. The objective is that the reduced event set that will be produced by the filtering process will contain those events that have a relatively high degree of similarity with the beacon event set B.

2) Beacon event attribute weight determination: In order to increase the accuracy of the event similarity measure evaluation, it is desirable to assign increased weights to those attributes and their corresponding values that are dominant within the selected beacon events and may, therefore, be a distinguishing feature for our analysis. The outline of the algorithm is presented below.

Input: $B = \{b_1, b_2, \ldots b_{n_b}\}$
Output: attribute and attribute value weight sets NA_b, NV_b
Process:

i. Let $b_k = \{\langle a_j, ev_{j,k}\rangle \mid a_j \text{ the attribute}, \ ev_{j,k} \text{ the value}, 1{\le}j{\le}n_a\}$, $1{\le}k{\le}n_b$, where n_b the number of beacon events.
ii. Let NA_b be the key-value pair set where each attribute a_j is paired with the number of its appearances $na_{j,b}$ in B: $NA_b = \{\langle a_j, na_{j,b}\rangle \mid 1{\le}j{\le}n_a, \ na_{j,b} = \sum_j(0 \textit{ if } ev_{j,k} == \text{ null}, \textit{1 otherwise})\}$.
iii. Let VB_j be the set of discrete values $v_{j,b,q}$ of the attribute a_j in the beacon event set B: $VB_j = \{v_{j,b,q} : \forall v_{j,b,q} \exists k : ev_{j,k} = v_{j,b,q}, \forall k \ ev_{j,k} {\in} V_j, 1{\le}k{\le}n_b\}$
iv. Let $NV_{b,j}$ be the set of tuples where each attribute value $v_{j,b,q}$ of the attribute a_j is associated with its number of appearances in the beacon event set B: $NV_{b,j} = \{\langle v_{j,b,q}, nv_{j,b,q}\rangle : 1{\le}q{\le}|VB_j|, nv_{j,b,q} = \sum_q(1 \textit{ if } ev_{j,k} = v_{j,b,q}, 0 \textit{ otherwise})\}$
v. Create a map hierarchy that associates each discrete attribute value with the number of beacon events in which it is present: $NV_b = \{\langle a_j, NV_{b,j}\rangle : 1{\le}j{\le}n_a\}$
vi. Return NA_b and NV_b.

3) Attribute weight determination for non-beacon events: While the attribute weights computed in the previous step are representative of the beacon events, a high weight value does not necessarily indicate a distinguishing feature. The weight of each attribute value should be adjusted in relation to its frequency in the input data. There are two extreme cases where this necessity is more apparent:

- Attributes that have the same value throughout the input data set cannot be used to distinguish desired events.
- Attributes that always have unique values are usually identifiers with limited use outside the scope of the event that contains each instance.

The input event set attribute weights are essentially used as a counter-balance to the weights produced in the previous step. The outline of the algorithm is presented below:

Input: $E' = E \setminus B$
Output: attribute and attribute value weight sets NA_e, NV_e
Process:

i. Let NA_e be the key-value pair set where each attribute a_j is paired with the number of its appearances in E': $NA_e = \{\langle a_j, na_{j,e}\rangle : 1{\le}j{\le}n_a, na_{j,e} = \sum_j(0 \textit{ if } ev_{j,i} = \text{ null }, 1 \textit{ otherwise})\}$
ii. Let VE_j be the set of discrete values $v_{j,q}$ of the attribute a_j in the input event set E': $VE_j = \{v_{j,e,q} : \forall v_{j,e,q} \exists k : ev_{j,i} = v_{j,e,q}, \forall k \ ev_{j,i} {\in} V_j, 1{\le}i{\le}n_e\}$
iii. Let $NV_{e,j}$ be the set of tuples where each attribute value $v_{j,e,q}$ of the attribute a_j is associated with its number of appearances in the beacon event set B: $NV_{e,j} = \{\langle v_{j,e,q}, nv_{j,e,q}\rangle : 1{\le}q{\le}|VE_j|, nv_{j,e,q} = \sum(1 \textit{ if } ev_{j,i} = v_{j,e,q}, 0 \textit{ otherwise})\}$
iv. Create a map hierarchy that associates each discrete attribute value with the number of input events in which it is present: $NV_e = \{\langle a_j, NV_{e,j}\rangle : 1{\le}j{\le}n_a\}$
v. Return NA_e, NV_e.

4) Beacon set/Input similarity determination: For each input event e_i in E, determine a similarity measure sm_i between the beacon event set and itself. The similarity measure is computed as a straight average of the individual similarity measures of the event e_i with each of the beacon events:

Input: E, B
Output: similarity measure set SM for all input events
Process:

i. Let $SM = \{\langle e_i, sm_i\rangle \forall e_i {\in} E\}$.
ii. Let $sm_i = (\sum_k esm_{i,k} \forall k : b_k {\in} B)/n_b$, where $esm_{i,k}$ is the similarity measure between events e_i and b_k.
iii. Return SM.

5) Beacon set/Input event similarity determination: The similarity measure $esm_{i,k}$ is computed as a weighted average of the attribute value similarity measure $vsm_{j,i,k}$ for all attributes.

Input: $e_i = \{\langle a_j, ev_{j,i}\rangle | 1{\le}j{\le}n_a\}$
$b_k = \{\langle a_j, ev_{j,k}\rangle | 1{\le}j{\le}n_a\}$
Output: event similarity measure $esm_{i,k}$
Process:

i. Let $w_{j,i,k}$ be the weight of $vsm_{j,i,k}$.
ii. Let $c_{j,i,k}$ be the contribution coefficient of $vsm_{j,i,k}$.
iii. $esm_{i,k} = \sum_j(w_{j,i,k} \cdot c_{j,i,k} \cdot vsm_{j,i,k}) \ / \ \sum(w_{j,i,k})$, $1{\le}j{\le}n_a$.
iv. Return $\{esm_{i,k} | 1{\le}i{\le}n, 1{\le}k{\le}n_b\}$, where n the number of events and n_b the number of beacon events.

The computation of the weight $w_{j,i,k}$, the contribution coefficient $c_{j,i,k}$ and the similarity value $vsm_{j,i,k}$ are presented in detail below.

a) Weight coefficients: The weight $w_{j,i,k}$ and the contribution coefficient $c_{j,i,k}$ is computed taking into account the NA_b, NA_e, NV_b and NV_e sets with the intent to:

- Increase the impact of attributes and corresponding values that occur frequently in the beacon events.
- Reduce the impact of attributes and values that appear with extreme frequency in the general event population, such as domain-specific constants.
- Reduce the impact of unique or almost unique attribute values in the general event population, such as event counters, identifiers and hash values that have limited importance outside the scope of their particular event.

Input: NA_b, NA_e, NV_b, NV_e
Output: weight $w_{j,i,k}$, contribution coefficient $c_{j,i,k}$
Process:

i. Let $bw_{j,i}$ be the beacon attribute value coefficient: $bw_{j,i} = (nv_{j,b,i}/n_b)^2$.

ii. Let aw_j be the event attribute coefficient: $aw_j = (n_e - n_b - na_{j,e})/n_e$.
iii. Let $vw_{j,i}$ be the event attribute value coefficient: $vw_{j,i} = (na_{j,e} - nv_{j,e,i})/na_{j,e}$.
iv. Let $cw_{j,i}$ be the composite attribute value coefficient: $cw_{j,i} = (aw_j)^2 + (1-(aw_j)^2) \cdot (vw_{j,i})^2$.
v. $c_{j,i,k} = bw_{j,i} \cdot bw_{j,k} \cdot cw_{j,i} \cdot cw_{j,k}$
vi. $w_{j,i,k} = (na_{j,b})^2 \cdot (na_{j,e})^2$
vii. Return $\{w_{j,i,k},\ c_{j,i,k} \mid 1 \le j \le n_a,\ 1 \le i \le n,\ 1 \le k \le n_b\}$, where n_a the number of beacon events.

b) Value similarity: The last component in the similarity determination is the calculation of the string-based similarity measure $vsm_{j,i,k}$ for all attributes. This value is generated as a hierarchy of weighted averages, where each component is the result of a comparison function. A missing attribute value (*null*) in either event results in the similarity function having an output of zero. This hierarchy is produced by several interconnected instances of a composite similarity function loosely based on the Voting Experts [18] method. Each composite similarity function contains references to one or more such functions, each of which is called with the objects under comparison as arguments. The top level of the hierarchy is thus defined by the following algorithm:

Input: $SF = \{\langle sf_{j,l}(v_1, v_2), sfw_{j,l}(v_1, v_2)\rangle\},\ ev_{j,i},\ ev_{j,k}$
Output: similarity measure $vsm_{j,i,k}$ between event i and event k for attribute j
Process:

i. Let $sf_{j,l}(v_1, v_2)$ be the l^{th} string-based similarity function of the attribute a_j.
ii. Let $sfw_{j,l}(v_1, v_2)$ be the weight determination function for the l^{th} string-based similarity function of the attribute a_j.
iii. Let SF be the set of tuples that associates each similarity function with its corresponding weight function.
iv. $vsm_{j,i,k} = \sum_l (sf_{j,l}(ev_{j,i}, ev_{j,k}) \cdot sfw_{j,l}(ev_{j,i}, ev_{j,k})) / \sum_l (sfw_{j,l}(ev_{j,i}, ev_{j,k})),\ 1 \le l \le |SF|$
v. Return $\{vsm_{j,i,k} \mid 1 \le j \le n_a,\ 1 \le i \le n,\ 1 \le k \le n_b\}$

The string based similarity functions $sf_{j,l}(v_1, v_2)$ can be either simple or composite and return a floating point number which indicates the degree of similarity between the two arguments, rather than a simple boolean value.

Composite similarity functions incorporate several simple ones and return a similarity value that can be defined as the average of all collected values from the simple ones. The simple functions that perform the actual value comparisons reside at the bottom of the hierarchy. Most such functions are string based, although it is possible to introduce type-specific functions that are triggered when type information is available. The available comparison functions include case-sensitive and case-insensitive string equality, Damerau-Levenshtein [19] string distance metrics, network address proximity and numeric proximity for textual representations of numbers.

The system is extensible, allowing for more complex similarity functions to be included, such as metrics based on lexicographical and linguistic similarities, domain-agnostic fuzzy association rule systems, or even domain-specific features.

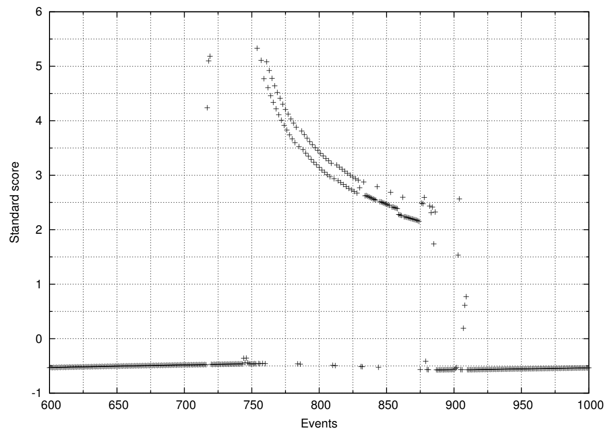

Fig. 1. Standard score values along a part of the input

C. Filtering

The similarity score sm_i between each input event and the beacon event set cannot be used on its own for the determination of the reduced data set, since such similarity score values will vary depending on the input event set and the selected beacons. Therefore a comparative analysis of all scores is necessary for this final selection step.

In the proposed system, the similarity measure values sm_i are fed into a statistics engine, which computes the straight average μ and standard deviation σ for these values. Furthermore, for each event e_i the system computes the standard score z_i (otherwise known as z-score) of the similarity measure sm_i: $z_i = (sm_i - \mu)/s$. The standard score is a dimensionless quantity that measures the deviation of a variable value from its mean, using the standard deviation as a unit. The standard score is then used to determine which events have a statistically high similarity score compared to the beacon set. Standard scores over 1 are a typical selection, although the threshold can be lowered or increased to widen or tighten the selection respectively [20]. Naturally, there is a trade-off between the recall and the precision of the system: lowering the standard score threshold will improve the recall of the filtering stage by increasing the size of the output. This, however, results in worse precision, as well as in a reduction in the performance of any subsequent processing stages. Fig.1 illustrates the use of the standard score for event selection; while most events have a negative standard score, events with high correlation values that should be included in the reduced data set rise over the similarity score baseline, forming easily detectable clusters. This is evident in Fig.1 for the range between the 750^{th} and the 800^{th} event.

IV. Usage considerations

A. Off-line usage

The filtering algorithm as described is directly suitable for off-line event log analysis. This mode of operation is simpler, more efficient and potentially more accurate, as it generally implies a known finite number of events that are available beforehand in randomly accessible storage. Therefore, it is possible to generate the attribute value weight sets NA_e and NV_e for the whole data set a priori. These sets will have to be adjusted before each use, by removing the contribution of the selected beacon event set, but this operation is significantly less intensive than re-creating the NA_e and NV_e sets.

This can improve the quality of the filtering stage results by taking into account the whole event population, while also increasing the overall system performance for multiple queries by computing these weight sets only once for each input event set and reusing them for different beacon event selections.

B. Real-time usage

A more interesting usage scenario is when using the filtering algorithm for processing real-time log data feeds. This scenario, however, brings forth a number of issues to resolve.

First, the input event set is actually a potentially infinite stream and it is not known beforehand – each event can only be processed once it is received. Therefore the weight sets NA_e and NV_e cannot be computed in a way that takes the whole input set into consideration.

Second, the algorithm cannot directly produce results for the initial segment of the input stream, since it relies on comparative statistical analysis of the similarity measures for a number of events, which cannot generate reliable results when the size of the sample is relatively small. In addition, the attribute value weight sets are not meaningful when they reflect only a limited sample of the input.

Third, while the assumption that the set A of discrete attributes is bounded may reasonably hold, the same is not true of the attribute value weight set NV_e. The theoretical design requires a counter for each discrete value of an attribute. In the general case, we can assume that each incoming event will contain an average nv_n of as-of-yet unseen – and potentially unique – values for some of its attributes. Therefore after i events, the NV_e set will contain $i \cdot nv_n$ items. Since the event counter i is not bounded, the NV_e set will grow infinitely over time with a roughly linear space requirement, which makes the use of the original unmodified algorithm technically impossible.

Limiting the size of the NV_e set (i.e. to consider a finite upper bound on the number of events used for the calculation of weight sets), represents a major change in the operation of the system. With the size of the NV_e set being limited by the available storage space, the system must balance the need to retain older entries with high occurrence counts with the need to hold onto recent entries that feature an unknown frequency, but may represent a future trend. Within the scope of the proposed system, the use of a caching, or object replacement, algorithm is an effective method for restricting the size of attribute value weight sets, although the semantics of caching are slightly different. In the process of investigating their use, we implemented and experimented with the following caching algorithms:

Least Recently Used (LRU): When the storage limit is encountered, this algorithm removes the entry that was used least recently. It is simple to implement and performs very well speed-wise, but it exclusively favors recency over frequency. As a result, LRU will behave pathologically in situations where a large enough batch of discrete values appear in sequence, thus replacing all existing entries in the cache with values that may be unlikely to reappear.

TABLE I
CACHING ALGORITHM PERFORMANCE METRICS

	ARC		LRU/LFU		No Replacement	
	Memory (MB)	Time (s)	Memory (MB)	Time (s)	Memory (MB)	Time (s)
SMTP	916	340	648	356	1,552	359
HTTP	817	426	784	455	1,553	492
TELNET	714	341	583	362	1,421	376

Least Frequently Used (LFU): At the opposite extreme of LRU, LFU always replaces the entry with the lowest access count. It favors frequency to the point that a perfect LFU implementation will barely be able to accept new entries once the cache is full. The use of this algorithm minimises the adaptability of the system, while also having a higher performance overhead.

LRU/LFU: Using a combination of both LRU and LFU allows their weak points to be covered to a degree. The values are entered in a c_{LRU}-sized LRU cache and are moved to a c_{LFU}-sized LFU cache once the LRU stage discards them. A sequential value scan on the input only affects the LRU stage, while new entries are able to enter the LFU stage with already-increased access counts. However, this alternative presents a number of issues that became visible during our experiments. First, the implementation is more complex and accessing two separate caches affects the performance of the system. Second, the c_{LRU} and c_{LFU} sizes need to be determined beforehand, which cannot happen optimally for arbitrary input event sets. Finally, the LFU stage has no concept of aging. Values with an inordinately high occurrence count will be retained indefinitely, despite not being encountered recently, leading to saturation with old values in long-running systems. Despite these issues, the LRU/LFU combination proved effective when processing moderately-sized input sets.

Adaptive Replacement Cache (ARC): ARC [21] achieved slightly better runtime performance than LRU, while being self-tuning over time and resistant to pathological behaviors. This became the algorithm of choice for the majority of the experiments we have conducted.

Table I presents a performance comparison between the main entry management algorithms implemented in our prototype system, for three test cases of $200,000$ events each. The no-replacement (NR) algorithm had the worst runtime performance and significantly higher memory usage. In fact, while the ARC and LRU/LFU variations reached a stable point in their memory consumption within a short time, the heap space of the NR process continued to grow indefinitely. In our test system, $200,000$ events was the absolute maximum that the NR implementation was able to handle without an out-of-memory exception.

Of the other two algorithms, ARC had a higher memory usage than LRU/LFU, which was expected due to additional space overhead imposed by its design. However, ARC proved to be consistently faster and was able to provide better results when tested with very large data sets in real-time mode, due to its inherent adaptability to changes in the input.

C. *Parallel processing*

The proposed filtering algorithm is highly parallelizable, using two orthogonal approaches:

First, the computation of the similarity measure of an input event with the beacon events can be parallelized, with each thread handling the comparison with a different beacon event.

Moreover, in most cases we can reasonably assume that the input event set is macroscopically uniform when separated into segments with a sufficiently small segment size. This allows the system to scale by splitting the input event set into small chunks that are then distributed into separately processed stripes. Each computing node can have its own attribute value weight sets and statistics engine, reducing the need for shared data and extensive thread synchronization.

In our prototype system, we have implemented the first approach, thus making full use of all processor cores in a multi-processor system.

V. Case studies

In order to evaluate the proposed algorithm, we used network captures from a subset of the DARPA Intrusion Detection Evaluation 1999 [3] and the KDD 1999 [4] data sets. The objective of the case study is to assess the proposed technique by identifying whether intrusion sessions are contained in the reduced logs computed using a collection of suspicious beacon events. The DARPA data set contains raw network traffic events while the KDD data set is a post-processed view of the DARPA set illustrating event segments pertaining to network intrusion attempts. The Wireshark [22] network analysis tool was used to provide accurate reference event sets to be used as a golden standard for evaluating precision and recall values. More specifically, we have conducted a number of case studies to evaluate and assess a) the run-time performance of the approach by measuring the log reduction processing time as a function of the size of the log data set; b) the quality of the approach by evaluating precision and recall values for cases drawn from both the DARPA and the KDD data sets; c) the run-time memory usage of the approach as a function of the caching algorithm used and; d) the stability of the approach by varying the beacon set and; e) the compositionality of the results by examining whether beacon sets selected from several different sessions will also attract events pertaining to these sessions.

A. *Runtime performance*

For measuring the runtime performance of the system we have conducted a series of runs with variable size beacon sets, applied to both the DARPA and the KDD data sets. The obtained results highlight two important characteristics of the proposed system that ensure its scalability. First, since each input log event is treated independently, the overall time complexity is linear with regard to the number of events. In fact, the event throughput of the system remains macroscopically stable and depends only on the selection of beacon events and the allocated computing resources. Second, the system is highly parallelizable, with the prototype implementation making full use of all four allocated processors. Fig.2 demonstrates both the linear complexity of the algorithm, as well as its scalability over multiple processors. The prototype implementation running on a four-core 2.83 GHz processor reduces a 200K event set in about 400 seconds, providing a recall typically over 73%.

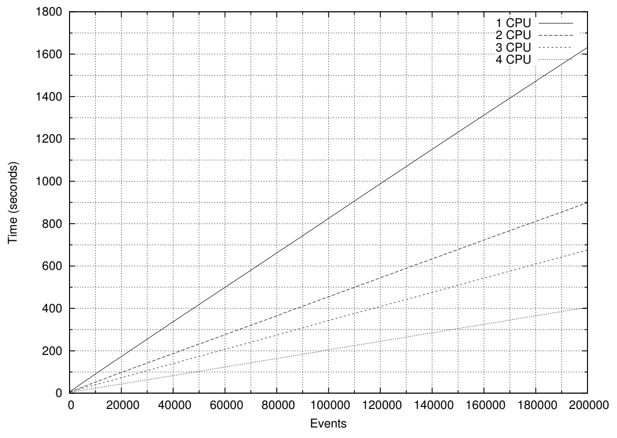

Fig. 2. Processing time in relation to input size

B. *Memory usage*

Memory usage is a crucial issue in log analysis, due to the limitations in keeping whole streams of logged data in memory and the lack of generalized methods for piece-wise processing. Such methods, often sliding-window variations, are difficult to use without a significant risk of losing important data. In this approach we have experimented with caching techniques borrowed from the area of operating systems, namely with the ARC and the LRU/LFU algorithms. Our results indicate that the use of such algorithms allows the system to keep important events in the cache, even when these occurred at a point in time that a sliding window type of approach would have missed. The experimental results in memory and time usage obtained by utilizing these algorithms are presented in Table I of Section IV-B. These results indicate that the ARC algorithm performs slightly better than the LRU/LFU algorithms, while both caching techniques outperform the no-caching alternative, which is not practically usable in most cases anyway.

C. *Quality of results*

1) Precision and recall: The event selection algorithm of the proposed system is tunable via the threshold value

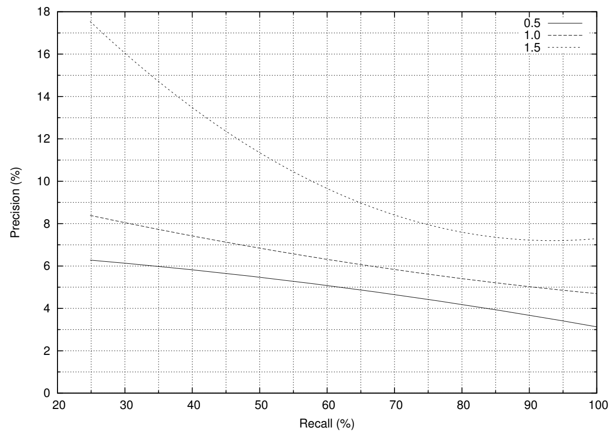

Fig. 3. Recall/precision curve for the DARPA data set using three different threshold values

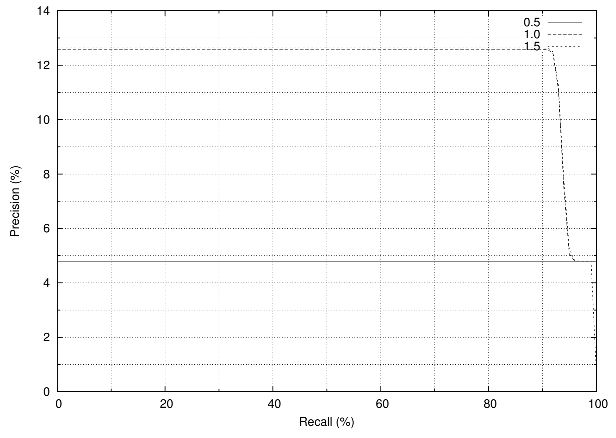

Fig. 4. Recall/precision curve for the KDD data set using three different threshold values

of the similarity measure standard score. Fig.3 depicts the recall/precision relations obtained by applying 18 different queries for different scenarios (SMTP, TELNET, HTTP+SSH) in the DARPA data set. The graph is obtained by using the interpolated precision method for fixed recall levels presented in [23]. The results indicate that for a threshold of 0.5 (a more relaxed cut-off point for similarity values), obtaining a recall level over 90% has an expected precision of 3%. It is noted that because the reduction is at the level of 95% and above, the precision of 3% is acceptable for 90% recall. The number of beacons in these experiments varied from 5 to 20 for cases comprising of 57 to 1960 events. In this respect, the results indicate that even with a small number of beacons we were able to have a high rate of reduction and a high level of recall. As expected, the results in Fig.3 indicate that for a threshold value of 1.5, the precision increases as we become stricter on the similarity cut-off value and a higher similarity value requirement is imposed for considering two events related.

Similar to the above, we conducted a corresponding set of experiments on the KDD data set. The objective is to assess

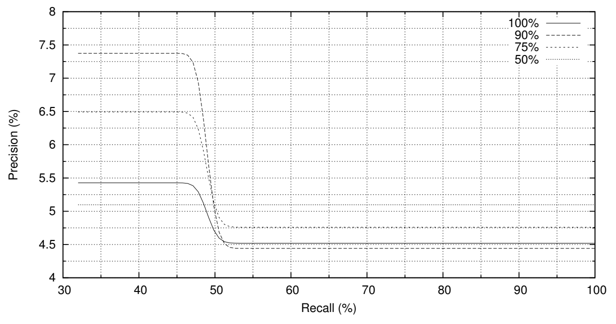

Fig. 5. Stability analysis for four different levels of beacon set correctness

whether we are able to capture in the resulting reduced logs whole collections of intrusion sessions by just providing the system with a small set of suspicious beacon events belonging in the intrusion set. In other words, the recall value indicates whether we are able to assemble the whole intrusion session by looking at a small subset of it, while the precision aims to assess the noise in the obtained results. The results are illustrated in Fig.4. It is noted that the reduction rate in this set of experiments is also at a level of 95% and above. In this type of evaluation we were still able to obtain a high recall and a high reduction rate, indicating a consistent behavior of the approach when applied to this data set as well.

2) Stability evaluation: The stability evaluation case study was conducted to assess the obtained results when the beacon set is varied by considering a percentage of the beacon events that is not contained in the set of the events that should be retrieved. The results are illustrated in Fig.5 and indicate that random noise in the beacon event set does not influence the quality of the obtained results, since noisy events do not contribute to the calculation of the weights and similarity values in a consistent manner, contrary to the more coherent beacon events that belong to a use case. More specifically, we have experimented with 100% of the selected beacons belonging to the case and then comparing with the results of the same beacon sets after introducing 10%, 25% and 50% of random noisy beacons. Fig.5 illustrates that the precision and recall levels are not significantly influenced by the noisy beacons, indicating that the proposed approach exhibits an acceptable level of stability with respect to the correct selection of beacon events.

3) Compositionality of cases: For this collection of use cases we have selected beacons spanning across different cases, with the objective to identify whether we would be able to attract events from both cases and recover the sessions the beacons were referring to. Table II illustrates precision and recall levels for combinations of HTTP, SMTP and TELNET sessions when the beacons were selected across these sessions. The results indicate that we have high recall levels (e.g.

TABLE II
RESULT METRICS FOR DIFFERENT COMPOSITE QUERIES

Scenario	**Average Recall** (%)	**Average Precision** (%)	**Average Reduction** (%)
FTP+HTTP	76.667	0.101	88.596
FTP+SMTP	93.333	0.104	86.583
FTP+TELNET	96.667	0.063	76.884
HTTP+SMTP	83.333	0.120	89.569
HTTP+SSH	83.333	0.120	89.577
SSH+TELNET	96.667	0.064	77.178

76.66% for FTP and HTTP combinations) while maintaining a high reduction rate.

D. Threats to validity

The threats to validity can be summarized in three points. The first point deals with the nature of the data. More specifically, when events differ in only a limited number of attributes, then a high similarity score may be computed while this small difference warrants a low score in a specific domain. Another type of such pathological behavior is when the distinguishing factor for a low or high similarity value is the differences in combinations of two or more attributes. The second issue relates to the beacon event selection process. More specifically, when the beacon events vary significantly on their values then noise may be introduced in the obtained reduced data set. Furthermore, the stability of the process may be affected if the majority of non-random beacon evens do not belong in the session that pertains to the problem at hand. Goal models and other specification models can be possibly used to ensure that the selected beacons are relevant to the case at hand. Finally, the third type of threat to validity problem relates to the similarity threshold selection process. More specifically, lowering the similarity threshold allows more noise in the results, while increasing the recall. Conversely, increasing the threshold improves the precision, while reducing the recall at the same time. There is no fixed algorithm to determine a single threshold value. However, the cumulative distribution function of the similarity scores often indicates a range of threshold values by which high recall and reduction levels can be obtained, for acceptable levels of precision.

VI. CONCLUSION

In this paper, we proposed a domain independent technique that allows for filtering large log data sets, based on a collection of beacon events that are selected by the operator or an automated auditing process as the system operates. The technique computes similarity values between logged events and the beacon events. The objective is for these beacon events to "attract" other events to form cohesive groups with a high likelihood of corresponding to a complete use case.

Future work can expand towards several different directions. First, the incorporation of feature reduction techniques, such as those in [13] and [14] would allow the removal of select attributes, thus improving the runtime performance and potentially the quality of its results. Second, the parallelization of the system using a distributed computing framework would prove the scalability of the algorithm to large monitoring infrastructures. Third, the combination of the system with a stateful second-stage algorithm that would take inter-event relationships into account, such as clustering or pattern matching could possibly improve the overall precision. This work is supported by CALabs of CA Technologies UK.

REFERENCES

[1] H. Zawawy, K. Kontogiannis, and J. Mylopoulos, "Log filtering and interpretation for root cause analysis," in *Software Maintenance, IEEE International Conference on*, 2010, pp. 1–5.

[2] H. Zawawy, K. Kontogiannis, J. Mylopoulos, and S. Mankovskii, "Towards a requirements-driven framework for detecting malicious behavior against software systems," in *Proceedings of the 2011 Conference of the Center for Advanced Studies on Collaborative Research*. IBM Corp., 2011, pp. 15–29.

[3] "DARPA intrusion detection evaluation." [Online]. Available: http://www.ll.mit.edu/mission/communications/ist/corpora/ideval/

[4] "The Knowledge Discovery & Data Mining Cup 1999 Data." [Online]. Available: http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html

[5] B. Bruegge, T. Gottschalk, and B. Luo, "A framework for dynamic program analyzers," in *SIGPLAN Notices*, vol. 28, no. 10. ACM, 1993, pp. 65–82.

[6] K. Koskimies, T. Männistö, T. Systä, and J. Tuomi, "SCED: a tool for dynamic modelling of object systems," *University of Tampere, Dept. of Computer Science, Report A-1996*, vol. 4, p. 199, 1996.

[7] "Java$^{\text{TM}}$ Virtual Machine Tool Interface (JVM TI)." [Online]. Available: http://docs.oracle.com/javase/7/docs/technotes/guides/jvmti/

[8] M. Salah, S. Mancoridis, G. Antoniol, and M. Di Penta, "Scenario-driven dynamic analysis for comprehending large software systems," in *Software Maintenance and Reengineering, Proceedings of the 10th European Conference on*, 2006, pp. 10–80.

[9] J. Kobielus, "Complex event processing: still on the launch pad," *Network World*, vol. 8, p. 16, 2007.

[10] D. Luckham, *The Power of Events*. Addison-Wesley, Boston, 2002.

[11] "CEP community." [Online]. Available: http://forum.complexevents.com

[12] T. M. Khoshgoftaar and P. Rebours, "Improving software quality prediction by noise filtering techniques," *Journal of Computer Science and Technology*, vol. 22, no. 3, pp. 387–396, 2007.

[13] G. Zargar and P. Kabiri, "Identification of effective network features for probing attack detection," in *Networked Digital Technologies, First International Conference on*, 2009, pp. 392–397.

[14] S. Zaman and F. Karray, "Features selection for intrusion detection systems based on support vector machines," in *6th Consumer Communications and Networking Conference*. IEEE, 2009, pp. 1–8.

[15] S. Goedertier, J. De Weerdt, D. Martens, J. Vanthienen, and B. Baesens, "Process discovery in event logs: An application in the telecom industry," *Applied Soft Computing*, vol. 11, no. 2, pp. 1697–1710, 2011.

[16] R. Agrawal, D. Gunopulos, and F. Leymann, "Mining process models from workflow logs," *Advances in Database Technology – EDBT'98*, pp. 467–483, 1998.

[17] "MongoDB." [Online]. Available: http://www.mongodb.org/

[18] P. Cohen and N. Adams, "An algorithm for segmenting categorical time series into meaningful episodes," *Advances in Intelligent Data Analysis*, pp. 198–207, 2001.

[19] F. Damerau, "A technique for computer detection and correction of spelling errors," *Communications of the ACM*, vol. 7, no. 3, pp. 171–176, 1964.

[20] R. Larsen and M. Marx, *An introduction to mathematical statistics and its applications*. Pearson College Div, 2001, vol. 1.

[21] N. Megiddo and D. Modha, "ARC: a self-tuning, low overhead replacement cache," in *Proceedings of the 2nd USENIX Conference on File and Storage Technologies*, 2003, pp. 115–130.

[22] "Wireshark." [Online]. Available: http://www.wireshark.org/

[23] C. Manning, P. Raghavan, and H. Schutze, *Introduction to information retrieval*. Cambridge University Press, 2008.