The Journal of Systems and Software 129 (2017) 1–25

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

Distributed analysis and filtering of application event streams

Theodoros Kalamatianos [a,*], Kostas Kontogiannis [b]

[a] National Technical University of Athens Dept. of Electrical and Computer Engineering, Athens, 15780, Greece
[b] Western University Computer Science Dept., London ON., N6A 5B7, Canada

ARTICLE INFO

Article history:
Received 31 May 2016
Revised 13 March 2017
Accepted 30 March 2017
Available online 1 April 2017

Keywords:
Software engineering
System understanding
Dynamic analysis
Event stream filtering
Information theory

ABSTRACT

Large information systems comprise different interconnected hardware and software components, that collectively generate large volumes of data. Furthermore, the run-time analysis of such data involves computationally expensive algorithms, and is pivotal to a number of software engineering activities such as, system understanding, diagnostics, and root cause analysis. In a quest to increase the performance of run-time analysis for large sets of logged data, we present an approach that allows for the real time reduction of one or more event streams by utilizing a set of filtering criteria. More specifically, the approach employs a similarity measure that is based on information theory principles, and is applied between the features of the incoming events, and the features of a set of retrieved or constructed events, that we refer to as beacons. The proposed approach is domain and event schema agnostic, can handle infinite data streams using a caching algorithm, and can be parallelized in order to tractably process high volume, high frequency, and high variability data. Experimental results obtained using the KDD'99 and CTU-13 labeled data sets, indicate that the approach is scalable, and can yield highly reduced sets with high recall values with respect to a use case.

1. Introduction

Large information systems consist of many interconnected components, in terms of both physical equipment and software applications. These components perform complex tasks and often run on virtualized and dynamically provisioned environments in order to meet a multitude of requirements.

In such a system, infrastructure management tasks can take the form of rolling out new software versions, migrating between database schemas, instantiating new virtual platforms, invoking on-demand software applications, migrating processes to different virtual machines as well as, transferring, splitting, or distributing data and computation logic to different servers. Due to the scale involved, such tasks are generally undertaken by automated tooling that results in rapidly changing component topologies, often accompanied by shifts in the structure of the interchanged data. Moreover, at any time there can be several such operations in flight, creating complex transition scenarios with rapidly shifting trends.

The operation of such systems is monitored by a correspondingly large network of software and hardware sensors that emit a wealth of information in the form of event traces. Traditionally, event log analysis is applied off-line, using a corpus of events that are persisted in storage, and can be retrieved for analysis purposes. However, in rapidly evolving situations it is desirable to be able to reason about the system behavior while it is in operation. Such a capability is pivotal to important software- and systems-engineering tasks such as fault root-cause analysis, run-time requirements verification and automatic re-configuration.

In this context, event processing should be applied on-line, without requiring access to the whole event stream, and while being able to accommodate high volume, high frequency, and high variability data. However, several common analyses involve computationally expensive techniques, often using algorithms with superlinear time and space complexity, thus making their use for large volumes of data intractable. Therefore, an important objective is to devise methods that allow for the selective reduction of the logged data according to specific filtering criteria, so that any subsequent analysis stages can operate on a significantly smaller volume of data that is, however, still representative of the case being examined.

In this paper, we propose an on-line adaptive filtering approach with constant space and linear time complexity. Given a set of "interesting" events we refer to as *beacons*, it utilizes information theory principles to significantly reduce the volume of incoming event streams, without requiring a training data set. The concept of *beacons* is commonly used in the domain of program comprehension

* Corresponding author.
E-mail addresses: thkala@softlab.ntua.gr (T. Kalamatianos), kostas@csd.uwo.ca (K. Kontogiannis).

http://dx.doi.org/10.1016/j.jss.2017.03.057
0164-1212/

(Brooks, 1983) and refers to code features that serve as distinctive indicators of specific operations or data structures. Within the context of our work, beacon *events* denote identifying information that is of particular interest for a case, and may be defined either by an administrator or by a third-party automated monitoring or data mining application. For example, such a beacon event may represent an unsuccessful login attempt to a particular server, a suspect network packet flow, a warning for a failed transaction, service requests arriving at a high frequency from a group of servers, or a performance degradation alert.

Within our approach, event similarity values are computed between the beacon events and the events in the incoming stream, by comparing attribute values that are then weighted by a set of coefficients computed using information theory concepts. For example, if an attribute value is constant across all input events, then it cannot be used to distinguish specific events. Therefore the applied weights, which are based on its information content, should reduce its participation when computing the overall similarity between an input stream event and a beacon event.

As the filtering system operates and events are observed, an adaptive threshold process is applied to determine events that exhibit an aggregated similarity value with the beacon set that is above a computed threshold value. Such events are then selected as being significant with regard to the specific use case.

The task of filtering and reducing the volume of event traces which is to be considered for further analysis, must comply with three fundamental requirements. First, the filtering process should achieve high recall levels, high reduction rates, and acceptably high precision levels. Second, it should be able to cope with infinite data streams, while avoiding, if possible, windowing techniques, so that no important events will be inadvertently excluded. Finally, the process should be adaptive and tractable so that it can be applied on-line as the system operates or is being re-configured. Thus, it should be able to cope with new data streams that are potentially introduced by dynamically provisioned nodes, and for which there may be no prior knowledge on their event schema or other features.

The approach proposed in this paper satisfies these requirements and, in addition, has three notable advantages over existing dynamic log analysis approaches. First, it does not require any *a priori* knowledge of the monitored infrastructure, such as information on the event schema or the distribution of the monitored values. As a result, it is readily applicable for collections of logged data that are emitted by different monitoring components and do not conform to the same schema, or format. This reduces the need for event schema merging or mapping, a process that is often too computationally expensive to be performed for on-line analysis, and often requires human intervention. It is also important in IT environments where services may be provided by one or more third-party entities, in which case there may be limited visibility into the proprietary details of the used data formats.

Second, the proposed approach does not depend on the existence of a training data set, which is a requirement for a large number of approaches that are based on machine-learning techniques and is frequently very hard to procure. Rather, our work is based on the on-line adaptive re-calculation of an information content-based coefficient for each attribute value, allowing thus the similarity score to be automatically adapted as the system evolves or its operational profile changes.

Finally, it has been designed for distributed operation on a computer *cluster*, with limited shared state across nodes, which allows it to scale for high event rates and for multiple input streams, if needed with the dynamic addition of more computing nodes. This is in contrast to most log analysis techniques that are intended to analyze data in an off-line manner, while often being limited by the capacity of a single host computer. Our prototype implementation was evaluated against the KDD'99 (Knowledge Discovery & Data Mining Cup Data, 1999) and CTU-13 (Garcia et al., 2014) data sets, allowing experiments in two different domains with dissimilar schemas. In addition, several tests were performed both on a single node and in a cluster built upon the Apache Storm framework.

This paper is organized as follows: Section 2 presents related work. Section 3 presents an overview of the proposed process, while 4 describes the used event model. Section 5 discusses the event similarity measure, while Section 6 relates the concepts that enable the proposed system to adapt to input changes and 7 describes the data structures used in our system. Section 8 presents the distributed-mode version of the proposed similarity algorithm and 9 describes the adaptive output event selection step. Finally, Section 10 presents implementation details, 11 presents experimental results, while sections 12 and 13 provide related discussion and conclude the paper.

2. Related work

Dynamic analysis of system traces and event streams has been extensively used, both to understand the behavior of information systems and as a critical component in automated processes. It is a complex problem with facets that touch upon several different areas.

Krempl et al. (2014) enumerate several challenges in the broader field of data stream analysis, with high data volume, velocity, latency and volatile data patterns being among them. Similarly, Tran et al. (2014) identify changing patterns as a major factor to be addressed in event stream processing systems.

In the field of anomaly detection, Lane and Brodley (1999) describe a method using instance-based learning, a data reduction algorithm and a probabilistic threshold determination process to detect sudden changes in computer account usage patterns, in a partially domain-independent manner. In particular, three different object replacement algorithms were tested, in order to restrict the number of recorded instances and thus the time and space complexity of the proposed method. Jiang et al. (2011) propose an information-theoretic method that first clusters related metrics from a monitored system and then uses changes in the entropy of the metric clusters to detect and diagnose faults. Chandola et al. (2012) survey several anomaly detection methods for the task of analyzing temporal sequences, including real-time event streams, in order to identify appropriate methods for specific problems. Pham et al. (2014) propose a technique based on the concept of Compressed Sensing (Donoho, 2006) to process data streams from large scale sensor networks. Rettig et al. (2015) describe a distributed pipeline for detecting anomalies in high-volume event streams using a combination of sliding windows and stream segmentation with either Pearson's correlation or a metric based upon relative entropy. In comparison, our work does not focus or rely on the distinction between normal and anomalous behavior, does not require any *a priori* assumptions about the distribution of the target events or the applicability of any windowing techniques and it is completely domain-agnostic.

In the area of data set filtering, Khoshgoftaar and Rebours (2007) discuss two different data filtering and noise reduction techniques, based on multiple-partitioning and iterative partitioning filtering and targeting a *static* pre-specified reduction rate, for the purpose of reducing the size of machine learner training data sets. Goedertier et al. (2011) examine several data mining techniques for the discovery of processes from event logs, while Agrawal et al. (1998) use a graph-based algorithm to

extract workflows from execution logs. Although the methods from both Goedertier et al. and Agrawal et al. are able to effectively extract event sequence models, an amount of domain knowledge is required in order to establish dependencies between events and to validate certain assumptions, while both approaches are applied off-line. In addition, Zheng et al. (2009) discuss a log preprocessing method that uses temporal and causal relations between events emitted within a window of time to remove redundant events from streams originating in large computer systems. Finally, Vaarandi and Podiņš (2010) use a classification engine and a frequent itemset mining algorithm to cluster redundant alerts from an IDS in order to reduce the size of the output. In contrast to our work, both of the aforementioned methods essentially perform unguided compression, rather than targeted, while the one by Zheng et al. requires the pre-definition of a hierarchy of domain-specific categories, where the events are classified with the use of regular expressions. In our prior work (Kalamatianos and Kontogiannis, 2014) a domain independent approach is proposed for log reduction using information-theoretic metrics. The approach proposed in this paper is distinguished by several differences, intended to improve the overall accuracy and performance of the system for a larger variety of use cases. More specifically, the use of normalization functions during the computation of the weight coefficients extends the applicability of the presented method to a wider range of input streams. Moreover the introduction of a distributed Map/Reduce-like processing framework allows for increased throughput and scalability to multiple computing nodes. Finally, we introduce the concept of the *iTree* structure, which is specifically designed to address precision issues that emerge when performing distributed computation of information-theoretical metrics on streams.

In the field of log analysis by event feature reduction, Tan et al. (2014) propose an adaptive feature scaling (AFS) technique that iteratively solves multiple sub-problems in order to produce a reduced set of features from ultra-dimensional big data, using a set of intermediate caching methods to improve the training efficiency. Similarly, Wang et al. (2014) present a family of algorithms for online feature selection that reduce the training cost by requesting a limited dynamic set of attributes from the training data set and not relying on all features being immediately available. Zawawy et al. (2010) discuss a technique using annotated goal models and Latent Semantic Indexing for log reduction and filtering so that root cause analysis can be tractable for systems emitting large volumes of log data. Zaman and Karray (2009) use the Enhanced Support Vector Decision Function technique along with a limited volume of training data for selecting features that can be efficiently used in a subsequent classification stage. Zargar and Kabiri (2009) use a Principal Component Analysis approach to identify event features that are important for dynamic analysis and, in particular, intrusion detection. Finally, Chou et al. (2008) propose a two phase approach for network intrusion detection that is based on feature reduction and reasoning using fuzzy clustering and the Dempster-Shafer theory. In comparison to these publications, our work does not depend on annotations or training data that are often not available for existing infrastructures, and focuses on reducing the volume of the input stream, rather than its dimensionality. Additionally, it does not rely on any inherent assumptions about the *nature* of the targeted events, as opposed to methods that are fine tuned for intrusion detection.

With regard to information-theoretic attribute metrics, Fleuret (2004) proposes a fast technique based on the maximization of the conditional mutual information and minimization of the dependencies between attributes for the selection of important features in data collections. Last et al. (2001) present an information theoretic approach that selects an optimal set of attributes by removing irrelevant and redundant features, while building a single classification model for the input data. Similarly, Sebban and Nock (2002) present a hybrid approach for feature selection based on information theory as well as filter/wrapper models. In general, these approaches attempt to detect general invariants within the input stream, in order to reduce its dimensionality. In contrast, the system proposed in this paper assigns attribute weights in a *targeted* manner by using the beacon event set $\mathbb{B}$ as a *reference*, so that irrelevant events may be removed.

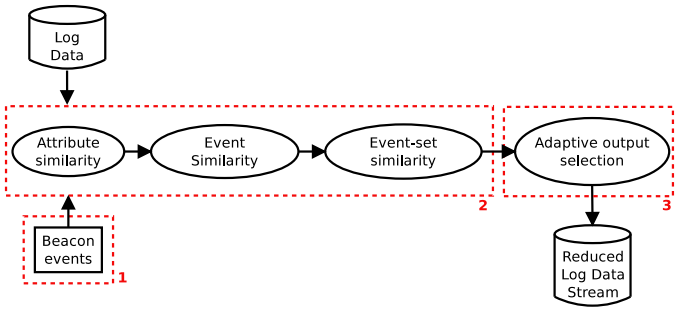

Fig. 1. Process outline.

With regard to distributed event processing, Papapetrou et al. (2012) describe a distributed event stream sketching and sketch aggregation technique for sliding-window applications, while Mutschler and Philippsen (2013) present an adaptive middleware for low-latency reordering of out-of-order events in a distributed sensing environment. Hirzel (2012) contributes a partitioning Complex Event Processing operator for use in distributed environments, along with an automated code generator that implements the resulting aggregations in parallel. Anceaume and Busnel (2014) describe an efficient distributed algorithm for estimating the information divergence between an observed and an expected event stream, with the intention of detecting streams that have been tampered with by an adversary. In comparison, our approach uses the information contributed by each value to target events directly within the observed event stream, while avoiding the artificial constraints imposed by the sliding-window paradigm.

In the field of similarity metrics, Pirró and Euzenat (2010) present an information-theoretic approach to *semantic* similarity suitable for objects referenced within an ontology, while Seco et al. (2004) suggest a metric based on Information Content to assess the correlation between concepts in a taxonomy. In comparison, although some concepts are shared, our work does not require domain-specific information, such as labeled corpora of data or an explicit classification of concepts, and is intended for use on real-time event streams, rather than static object collections.

Overall, the main differences of our work with the related work presented in this section are that our approach is domain and schema independent and it does not require training data sets. In addition, it is applicable for large-volume real-time event stream processing systems, with support for distributed operation on a computer cluster for scalability.

3. Process outline

The event selection process is composed of three main steps, as depicted in Fig. 1. The initial step (1) involves the compilation of a beacon event set $\mathbb{B}$ that will serve as a search sample. In the second step (2), the system iterates over the input stream. Each input event is compared to the beacon set and a similarity value is determined. In general, the desired events receive higher similarity values than those not related to the beacon set, allowing them to be grouped. Fig. 1 depicts the simplified form of the process where all analysis and similarity metrics are computed on a single

processing node. As it will be further discussed in Section 8, step (2) can be distributed in many different processing nodes, where each processing node is handling a part or a fraction of the event stream. The final step (3) is the actual selection of the output event stream, which uses the similarity values computed in the previous step. There are several alternative techniques that could be used at this stage, allowing for an extensive input stream to be reduced to more manageable levels.

3.1. Beacon event set compilation

The first step of the proposed process involves the compilation of a cohesive beacon event set $\mathbb{B} = \{b_1, b_2, \ldots, b_{|\mathbb{B}|}\}$ that serves as the search pattern for the analysis process. The size of this set is normally in the range of ten to twenty events and may be proportional to the number of distinct event types that are expected in a typical case as the one being suspected (e.g. an intrusion case).

The beacon event set can be selected directly from the input stream, but it can also be composed of *pseudo-events* with the same internal structure. Pseudo-events are synthesized artificially by the operator, by selecting and combining attributes and values that are considered important or of interest, for a given case. A beacon set built using such constructed events may result in reduced output noise and increased focus towards specific features.

Since the system requires no information on the actual selection process, it is feasible to use opaque third-party utilities to compile the beacon set. In addition, depending on the nature of the input stream, it may also be possible to leverage certain intrinsic states of the monitored system, e.g. by using metrics captured at error conditions as beacons to filter through a stream composed by metrics sampled periodically during normal operation. Finally, the limited size of $\mathbb{B}$ also makes its manual compilation using expert knowledge a tractable process.

3.2. Similarity determination

The second step of the process consists of the calculation of a similarity metric $\mathcal{S}$ between each event e_i of the input stream $\mathbb{E}$ and the beacon event set $\mathbb{B}$. More specifically, the calculation of $\mathcal{S}$ encompasses the computation of three intermediate values, namely the attribute-level similarity $\mathcal{AS}$, the event-level similarity $\mathcal{ES}$ and the event-set/beacon-set similarity $\mathcal{SS}$. The attribute level similarity $\mathcal{AS}$ leverages a low-level metric $\mathcal{VS}$ (e.g. a string distance metric), in order to compute a proximity measure between the value of a specific attribute in an input event e_i and the value of the corresponding attribute within a beacon event b_k. String distance is one metric that can be utilized to compare individual values at the attribute level. Other distance metrics are also possible to consider without affecting the overall concept of the proposed approach. A detailed discussion on the low-level metric $\mathcal{VS}$ is provided in Section 5.1.

The event-level function $\mathcal{ES}$ provides a correlation estimate between an input event e_i and a beacon event b_k, taking all attributes into account. The event-set/beacon-set similarity function $\mathcal{SS}$ considers the event-level results for all beacon events to produce the final similarity metric $\mathcal{S}$.

The $\mathcal{S}$ similarity metric is stateless, in the sense that no relations between different events, such as event sequences or value transitions, are explicitly considered. The only information that is persisted from the input stream relates to the *distribution* of values for each attribute and is stored in a per-attribute repository. This reduces the processing latency and allows for higher degrees of parallelization. It also simplifies the tuning of the system by reducing the need for detailed knowledge on the monitored infrastructure. Fully stateful systems, on the other hand, generally make several domain-specific assumptions, such as the typical duration of a session or which attributes may serve as a unique event sequence identifier.

```
{
    "service" : "http",
    "flag" : "RSTR",
    "src_bytes" : "54540",
    "dst_bytes" : "8314",
    "is_host_login" : "0",
    "srv_serror_rate" : "0.0",
    ...
}
```

Fig. 2. Example of an event from the KDD'99 data set in JSON notation.

3.3. Adaptive output event stream selection

The third step of the process focuses on the selection of an output event stream $\mathbb{O}$, which is only a subset of the input stream $\mathbb{E}$ and contains those events for which similarity values with $\mathbb{B}$ exceed a certain threshold. In this context, there are different approaches for determining a proper threshold value so that an output set can be computed. These approaches include static selection of the highest values via sorting of the output set, dynamic percentile extraction (Cormode et al., 2006) and various methods of threshold-based selection (Lane and Brodley, 1999). For our approach we consider an adaptive threshold selection method that is discussed in more detail in Section 9.

4. Event modelling

4.1. Static event model

In this section, we present the event model for the proposed system. The primary requirements for such an event model were modeling simplicity and the facilitation of processing performance. Each event e_i of the input stream $\mathbb{E} = \{e_1, e_2, \ldots, e_{|\mathbb{E}|}\}$ is defined as an unordered set of pair tuples, that consist of an attribute $a_j \in \mathbb{A}$ and its associated value $v_{j,i}$. More specifically, each event e_i in the input stream is defined as:

$$e_i = \{< a_j, v_{j,i} > : 1 \le i \le |\mathbb{E}|, \; 1 \le j \le |\mathbb{A}|, \; v_{j,i} \in \mathbb{V}_j\} \quad (1)$$

where $\mathbb{A} = \{a_1, a_2, \ldots, a_{|\mathbb{A}|}\}$ is the set of all attributes a_j appearing in the events of the input stream, and $\mathbb{V}_j$ is the set of all discrete atomic values $v_{j,i}$ encountered for attribute a_j. Similarly each beacon event b_k is defined as:

$$b_k = \{< a_j, v_{j,k} > : 1 \le k \le |\mathbb{B}|, \; 1 \le j \le |\mathbb{A}|, \; v_{j,k} \in \mathbb{V}_j\} \quad (2)$$

Both the input stream events and the beacon events have the same basic structure and lie within the same domain, therefore sharing the attribute set $\mathbb{A}$. Fig. 2 illustrates part of an event from the KDD'99 data set, with different attributes of string, numeric and boolean values.

4.2. Stream model

Due to the implicit insertion of *null* values, it is possible to assume that each input event contains a value for all attributes in the attribute set $\mathbb{A} = \{a_1, a_2, \ldots, a_{|\mathbb{A}|}\}$. This allows each attribute a_j to be defined as a stream S_{a_j} of its values $v_{j,i}$: $S_{a_j} = \{v_{j,1}, v_{j,2}, \ldots v_{j,|\mathbb{E}|}\}$. Therefore, the event stream as a whole can be in turn considered as a set of time series of values that evolve in

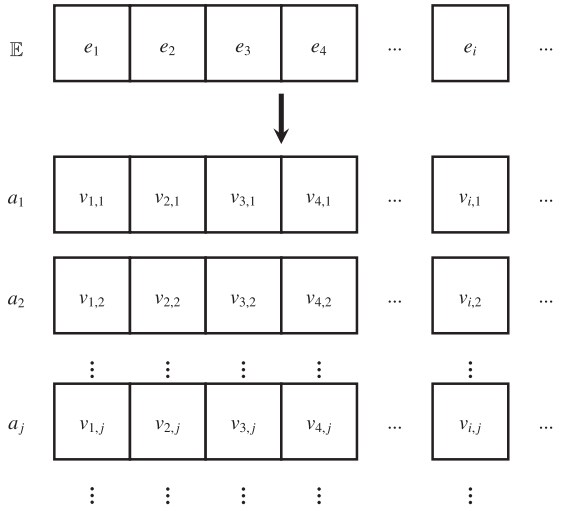

Fig. 3. Transition from the input event stream $\mathbb{E}$ to an equivalent set of attribute value time series.

parallel, with one time series for each discrete attribute. For example, in a system with three event streams, each having four distinct attributes in each event, there will be twelve parallel time series. Fig. 3 illustrates the transition from a stream of events $\mathbb{E}$ to an equivalent set of attribute value time series.

The proposed approach is most effective when each attribute is an independent variable with no correlation to any of the other attributes. Redundant information in the input stream may skew the results of the process by incorrectly emphasizing certain values, while reducing the perceived importance of others. The problem of normalizing redundant information streams has been studied extensively. For example, several feature selection techniques have been proposed (e.g. Fleuret (2004), Yu and Liu (2003), Zargar and Kabiri (2009)) and can be used as a pre-processing stage to remove redundant features.

5. Event similarity metric

In order to be able to select specific events from the input stream, we propose a multi-layer similarity metric $\mathcal{S}$ that computes a similarity value between a single event e_i from the input stream $\mathbb{E}$ and the beacon event $\mathbb{B}$ set as a whole. The proposed approach attempts to determine which attributes and attribute values offer the best event *selectivity* with regard to a specific beacon set. For this purpose, a statistical similarity measure based on information theory is introduced and will be discussed in the following sections.

For its computation, $\mathcal{S}$ combines the low-level value correlation metric $\mathcal{VS}$ with a set of *weight* coefficients to generate an attribute-level metric $\mathcal{AS}$. The $\mathcal{AS}$ value is then used to compute an event-level metric $\mathcal{ES}$, which in turn is used to evaluate the final event-set/beacon-set similarity value $\mathcal{SS}$.

The aforementioned weight coefficient set is dynamically determined and adjusted as events appear in the stream, and provides an indicator of the importance that an attribute or a value carries in the evaluation of a final overall similarity measure. For example, an attribute for which its value is constant throughout the input stream does not carry any significant *information content* for the computation, while attribute values that vary may carry higher *information content*. This allows a low-level similarity metric for primitive values (e.g. strings, or other literal values), to be used as the basis of an event-level similarity measure.

In a domain-specific system, such weight sets are generally provided using expert knowledge. This requires that the event domain is extensively modeled and that the intricacies of the monitored system are well understood and have been a priori compiled to usable rules. In real-world applications, however, the infrastructure models are often too abstract, outdated due to extensive maintenance, or even non-existent altogether.

Alternatively, a training data set may be used to train a machine learning algorithm, such as a neural network. However, the compilation of a training data set is generally not a trivial process. While segmenting a subset of the input for this purpose is generally feasible, providing the *feedback* that is required for the training process requires specific domain knowledge that may not be readily available. Compiling such a data set manually is usually non tractable, while training data created in simulated environments is limited by the accuracy of the simulation with regard to the corresponding live system.

The approach presented in this paper is able to determine these coefficients in a domain and event schema independent manner, without any prior knowledge on the monitored infrastructure or the structure of the emitted events. While this reduces its effectiveness when compared to domain-specific tools, it also increases its applicability. It can be successfully deployed in cases where no domain-specific tool is available, while it is able to adapt to different infrastructure configurations without operator intervention.

In order to demonstrate the calculation of the event similarity metric S in some common cases, as well as the operation of the system as a whole, we use a simple example that is based on a constructed event stream $\mathbb{EX}$ with four attributes $a_1, \ldots a_4$. a_1 and a_2 are essentially identical and contain a single relatively rare value, along with a few more frequent values. a_3 is almost always constant, while a_4 is highly variable. $\mathbb{EX}$ consists of 1,000,000 events, with the following relative frequencies per distinct attribute value:

a_j	$f(v_{j,1})$	$f(v_{j,2})$	$f(v_{j,i})$
a_1	0.001	0.099	0.1, $\forall i \in [3, 11]$
a_2	0.001	0.099	0.1, $\forall i \in [3, 11]$
a_3	0.99	0.001	0.001, $\forall i \in [3, 11]$
a_4	0.0001	0.0001	0.0001, $\forall i \in [3, 10000]$

We then consider a subset $e_1, \cdots e_4$ of $\mathbb{EX}$. Targeting events e_1 and e_3, which have a value of $v_{1,0}$ and $v_{2,0}$ for a_1 and a_2 respectively, we construct a beacon set $\mathbb{BX}$ with two events:

Event / Beacon	a_1	a_2	a_3	a_4
e_1	$v_{1,0}$	$v_{2,0}$	$v_{3,0}$	$v_{4,0}$
e_2	$v_{1,1}$	$v_{2,1}$	$v_{3,0}$	$v_{4,1}$
e_3	$v_{1,0}$	$v_{2,0}$	$v_{3,1}$	$v_{4,2}$
e_4	$v_{1,1}$	$v_{2,1}$	$v_{3,0}$	$v_{4,3}$
b_1	$v_{1,0}$	$v_{2,0}$	$v_{3,0}$	$v_{4,5}$
b_2	$v_{1,0}$	$v_{2,1}$	$v_{3,0}$	$v_{4,6}$

The beacon values for a_2 intentionally introduce a form of noise in $\mathbb{BX}$, while the ones for a_3 and a_4 continue the patterns of a mostly constant and a highly variable attribute respectively.

5.1. Low-level value correlation ($\mathcal{VS}$)

As streams can be dynamically added or removed, an important requirement for the calculation of the overall similarity metric $\mathcal{S}$ is the use of a stateless correlation function $\mathcal{VS}$ that is used to provide a measure of resemblance between the value $v_{j,i}$ of attribute a_j within the input event e_i and the corresponding value $v_{j,k}$ of beacon event b_k for the same attribute. This function $\mathcal{VS}$ is defined as:

$$\mathcal{VS}(j) : \mathbb{V}_j \times \mathbb{V}_j \rightarrow \mathbb{R} \tag{3}$$

In probability theory terms, $\mathcal{VS}$ provides an estimate for the probability $P(v_{j,i} \equiv v_{j,k})$ of the two compared values being equivalent.

Depending on the type of the values, there is a wide range of usable correlation functions, with a simple equality condition being the simplest alternative. The most generic data type is the *string*, which allows the use of a number of string-based distance metrics, both general-purpose and application-specific. Although various approaches have been introduced, such methods are often based on the lexicographic positions of the compared terms using a certain placement mechanism or on the transformation cost of one term to the other. Typical examples include traditional *edit distances*, such as the Damerau–Levenshtein (Damerau, 1964) and Hamming (1950) distances, as well as approximate or fuzzy metrics, such as the ones presented by Wagner and Fischer (1974) and Sellers (1980).

The prototype implementation makes use of a composite metric based on the structure of the Voting Experts (Cohen et al., 2007) algorithm, where several different string distance metrics are entered in a weighted average to produce the overall similarity between two attribute values. A heuristic type and value range detection algorithm is used to partially detect some common data types, such as strings, integers and floating point numbers. If type information is available for a particular attribute, then additional type-specific methods can be introduced, such as heuristic distance metrics for numbers, subnet distances for IP addresses, geometric distance metrics for geographic coordinate values, or even textual semantic similarity metrics (e.g. Mersch and Lang (2015), Mihalcea and Wiebe (2014)) for human-readable text. Although it is not necessary, this allows the accuracy of the system to improve if a degree of domain-specific knowledge can be inferred or is available beforehand.

The low-level value correlation function used in our example with $\mathbb{EX}$ and $\mathbb{BX}$ is based on a simple equality test:

$$\mathcal{VS}(x,y) = \begin{array}{l} 1 \;if\; x == y \\ 0 \;if\; x \neq y \end{array} \tag{4}$$

5.2. Attribute-level similarity metric ($\mathcal{AS}$)

The attribute-level similarity metric $\mathcal{AS}$ applies an appropriate weight coefficient to the raw result from $\mathcal{VS}$ in order to modulate the computed correlation by taking into account the importance of each value within its specific attribute and its contribution to the overall selectivity of the system.

According to information theory, the entropy H of an independent variable X is a measure of the average information content of each sample of X. Likewise, the information content I is a metric of the *importance* of a variable as a whole. If X is a discrete time series of N samples with an alphabet of n symbols and a probability mass function of $p(X)$, we have:

$$H(X) = E(-log_r p(X)) = -\sum_{i=1}^{n} p(x_i) log_r p(x_i) \tag{5}$$

$$\begin{aligned} I(X) &= -\sum_{i=1}^{n} n_i \cdot log_r p(x_i) \\ &= -N \cdot \sum_{i=1}^{n} p(x_i) \cdot log_r p(x_i) \\ &= N \cdot H \end{aligned} \tag{6}$$

where the probability $p(x_i)$ of a symbol x_i is equal to its relative frequency within the time series.

Within the proposed system each attribute a_j is considered an independent discrete random variable with an alphabet $\mathbb{V}_j$ of $|\mathbb{V}_j|$ possible symbols (values). Specializing formula 6 using the individual attribute value frequencies, it is therefore possible to compute the information content of a specific attribute value $v_{j,i} \in \mathbb{V}_j$, as well as that of a discrete attribute a_j as a whole:

$$I(v_{j,i}) = -n_{j,i} \cdot log_r \frac{n_{j,i}}{|\mathbb{E}|} \tag{7}$$

$$I(a_j) = -\sum_{i=1}^{|\mathbb{V}_j|} I(v_{j,i}) = -\sum_{i=1}^{|\mathbb{V}_j|} (n_{j,i} \cdot log_r \frac{n_{j,i}}{|\mathbb{E}|}) \tag{8}$$

The selectivity offered by each event attribute is in direct relation to the information content of the equivalent time series. Conceptually, highly repetitive attributes, such as domain-specific constants, have a limited use as *distinguishing features* between events, but they also exhibit a relatively low entropy that can be used to reduce their participation in the event comparison process. On the other hand, attributes with extreme diversity, e.g. unique identifiers, tend to skew the similarity metric, since they have a high information content despite being of limited value for determining similarity. To offset this issue, the information content $I(v_{j,i})$ contributed by each specific *attribute value* is taken into account in relation to the information content $I(a_j)$ of that particular attribute as a whole. This leads to the following definition of the *information content fractions* $IF(a_j)$ and $IF(v_{j,i})$ for an attribute and an attribute value respectively:

$$IF(a_j) = \frac{I(a_j)}{\sum_j^{|\mathbb{A}|} I(a_j)} \tag{9}$$

$$IF(v_{j,i}) = \frac{I(v_{j,i})}{I(a_j)} \tag{10}$$

While the information content fraction $IF(v_{j,i})$ provides an estimate for the relative *prevalence* of a value, its *selectivity* within an attribute is more closely related to the importance of the *remaining* values, which can be derived by the complement operator:

$$\overline{x} = 1 - x \tag{11}$$

Furthermore, the information content fraction IF and its complement $\overline{IF}$ are by definition dimensionless quantities in the [0, 1] range. However, in most conditions these measures will only cover a miniscule part of this range, thus *compressing* the range of any metric they are incorporated into. Therefore, we define a normalization operator $\mathcal{N}$ that uses an estimate x_{max} for the maximum possible value of a variable x to decompress the input values into the full [0, 1] range:

$$\mathcal{N}(x) = \frac{x}{x_{max}} \tag{12}$$

Based on these definitions, the corresponding normalized information fraction complement $\overline{NIF(v_{j,i})}$ is defined as follows:

$$\overline{NIF(v_{j,i})} = 1 - \mathcal{N}(IF(v_{j,i})) \tag{13}$$

The term $\overline{NIF(v_{j,i})}$ provides a measure of the *selectivity* contributed by the value $v_{j,i}$, allowing its use as a coefficient. Fig. 4 illustrates the behavior of this coefficient for a value $v_{j,i}$ as an intensity map, where lighter areas indicate higher values. $\overline{NIF(v_{j,i})}$ has been calculated for a scenario where all other values share the same frequency, making it directly dependent on their number. It is evident that the highest coefficients are achieved by either a value $v_{j,i}$ that dominates a stream composed of relatively few values (lower right lobe), or by a rare value in a stream where all other values are of moderate frequency (lighter area of the left lobe). Conversely, a rare value can be overshadowed by an extremely frequent one, while the effect of a prevalent value can be reduced by the presence of a large number of rare occurrences in the stream.

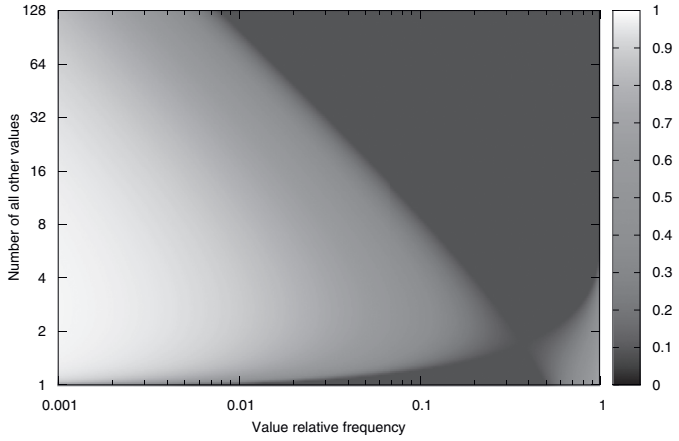

Fig. 4. Variation of the $\overline{NIF(v_{j,i})}$ coefficient depending on the relative frequency of a value $v_{j,i}$ and the number of all other values.

A slightly different approach needs to be considered for the events in $\mathbb{B}$. Since these events essentially form the *search pattern* for the proposed system, a zero or very low entropy indicates the presence of a constant or near-constant attribute value. Such values are often a *defining feature* of the selected beacon event set, especially if they do not frequently exist in the input stream as a whole. On the other hand, attributes with high diversity within the $\mathbb{B}$ set reduce its coherency and indirectly introduce noise within the output. Therefore, in this context, the impact of attributes with relatively high entropy $H_{\mathbb{B},a_j}$ within the beacon event set should be attenuated, while that of values with a high frequency should be reinforced.

Using the aforementioned principles and taking into account the frequency of each beacon event value, the similarity metric $\mathcal{AS}_{j,i,k}$ can be defined for the attribute a_j values of input event e_i and beacon event b_k in terms of the low-level similarity $\mathcal{VS}(j,i,k)$ and its associated weight $W(j,i,k)$ as follows:

$$\begin{aligned}\mathcal{AS}(j,i,k) &= W(j,i,k)\cdot\mathcal{VS}(j,i,k)\\ &= [\overline{NIF(v_{j,k})}\cdot\overline{NIF(v_{j,i})}\cdot\mathcal{N}(f_{\mathbb{B},v_{j,k}})]\cdot\\ &\quad \mathcal{VS}(j,i,k)\end{aligned} \tag{14}$$

where $f_{\mathbb{B},v_{j,k}}$ is the relative frequency of the value $v_{j,k}$ within $\mathbb{B}$ for the attribute a_j. The normalization operator $\mathcal{N}$ maps $f_{\mathbb{B},v_{j,k}}$ into the [0, 1] range, thus providing a measure of the importance of $v_{j,k}$ within $\mathbb{B}$.

For simplicity, in our example with $\mathbb{EX}$ we use as the estimate x_{max} for the normalization function $\mathcal{N}(x)$ the currently known maximum value of the input variable x, increased by 10%. We also do not take into account value frequency changes caused by the $e_1, \cdots e_5$ events. Therefore, for attribute a_1 of events e_1 and b_1, from $\mathbb{EX}$ and $\mathbb{BX}$ respectively, we have:

$$\begin{aligned}I(v_{1,1}) &= -1000 * log_2 0.001 \approx 9965.786\\ I(a_1) &\approx 3330008.851\\ IF(v_{1,1}) &\approx 9965.786/3330008.851 \approx 0.003\\ max(IF(v_{1,*})) &\approx 0.1\\ \overline{NIF(v_{1,1})} &= 1-\mathcal{N}(IF(v_{1,1})) \approx 0.973\\ \mathcal{AS}(1,1,1) &= W(1,1,1)\cdot\mathcal{VS}(v_{1,1},v_{1,1})\\ &= [\overline{NIF(v_{1,1})}^2\cdot\mathcal{N}(f_{\mathbb{B},v_{1,1}})\ ]\cdot 1.0\\ &= \overline{NIF(v_{1,1})}^2\cdot\mathcal{N}(1.0)\\ &\approx 0.860\end{aligned}$$

Using the same method we calculate all $\mathcal{AS}$ values. It is evident that in our example the $\mathcal{VS}$ function controls the final result, but the intermediate weights $W_{j,i,k}$ are visibly variable, with the less selective attributes a_3 and a_4 receiving significantly lower coefficients:

e_i	b_k	a_1		a_2		a_3		a_4	
		$W_{1,i,k}$	$\mathcal{AS}_{1,i,k}$	$W_{2,i,k}$	$\mathcal{AS}_{2,i,k}$	$W_{3,i,k}$	$\mathcal{AS}_{3,i,k}$	$W_{4,i,k}$	$\mathcal{AS}_{4,i,k}$
e_1	b_1	0.860	0.860	0.860	0.860	0.008	0.008	0.008	0.0
	b_2	0.860	0.860	0.085	0.0	0.008	0.008	0.008	0.0
e_2	b_1	0.085	0.0	0.085	0.0	0.008	0.008	0.008	0.0
	b_2	0.085	0.0	0.008	0.008	0.008	0.008	0.008	0.0
e_3	b_1	0.860	0.860	0.860	0.860	0.030	0.0	0.008	0.0
	b_2	0.860	0.860	0.085	0.0	0.030	0.0	0.008	0.0
e_4	b_1	0.085	0.0	0.085	0.0	0.008	0.008	0.008	0.0
	b_2	0.085	0.0	0.008	0.008	0.008	0.008	0.008	0.0

5.3. Event-level similarity metric ($\mathcal{ES}$)

For the event-level similarity $\mathcal{ES}$ we compute a weight W_{a_j} for each attribute a_j in the input stream $\mathbb{E}$:

$$W_{a_j} = \overline{\sum_{b_k\in\mathbb{B}} f_{\mathbb{E},v_{j,k}}}\cdot\frac{1}{1+H_{\mathbb{B},a_j}} \tag{15}$$

The first term of the product is the collective relative frequency of all values in a_j within the input stream $\mathbb{E}$ that are *not* encountered in $\mathbb{B}$, and provides an estimate of the selectivity contributed by this specific attribute. Additionally, it has the indirect effect of reducing the impact of attributes with an extremely limited value set, as in the general case the beacon event set will also contain a significant part of those values.

The second term, on the other hand, is a measure of the homogeneity of the beacon set values for a_j. Values with high variability incur a high beacon set entropy $H_{\mathbb{B},a_j}$ and therefore reduce the impact of a_j in the overall similarity. Moreover, attributes with high diversity will often also present this behavior in $\mathbb{B}$, which reduces their overall impact.

To obtain the metric $\mathcal{ES}$ that compares two *events*, a simple weighted mean is used to leverage the per-attribute similarity values to the event level:

$$\mathcal{ES}(i,k) = \frac{\sum_{j=1}^{|\mathbb{A}|}(W_{a_j}\cdot\mathcal{AS}(j,i,k))}{\sum_{j=1}^{|\mathbb{A}|}W_{a_j}} \tag{16}$$

Lin (1998) offers a formal definition of the concept of similarity, based on three basic intuitive tenets: (a) the more commonality two objects share, the more similar they are; (b) the more differences two objects have, the less similar they are and; (c) two identical objects should always reach the maximum similarity. Using a weighted mean to aggregate the per-attribute similarity values from the previous stages, to an event-level metric $\mathcal{ES}_{i,k}$, satisfies all three basic conditions. Additionally, it allows the attribute-level weights which are not bounded to form a bounded metric with the same range as the primitive correlation metrics.

Applying formula 15 on $\mathbb{EX}$ and $\mathbb{BX}$, we note that the beacon set noise for a_2 has reduced its weight compared to a_1, while the impact of a_3 is also greatly attenuated:

	a_1	a_2	a_3	a_4
$\overline{\sum_{b_k\in\mathbb{BX}} f_{\mathbb{EX},v_{j,k}}}$	999,000	900,000	10,000	999,800
$H_{\mathbb{B},a_j}$	0	1	0	1
W_{a_j}	0.999	0.45	0.01	0.4999

Using formula 16 with the values of $\mathcal{AS}$ from Section 5.2, we can compute the similarity $\mathcal{ES}$ for each event e_i and beacon b_k. As expected, the event-level similarity $\mathcal{ES}$ is significantly higher for the targeted events e_1 and e_3:

	e_1	e_2	e_3	e_4
b_1	0.636	3.84E-5	0.636	3.84E-5
b_2	0.439	0.002	0.439	0.002

5.4. Event-set/beacon-set similarity metric ($\mathcal{SS}$)

The final computation involves the determination of an overall similarity $\mathcal{SS}(i)$ of an input event with the beacon set as a whole. To compute a similarity metric between a single event and an *event set* using a metric defined between single events, it becomes necessary to reexamine the three basic principles mentioned in Section 5.3.

More specifically, the requirement for a maximum similarity result on identical inputs is no longer satisfiable since sets and single items are not directly comparable. It is, of course, possible to establish a *set-contains* condition, which in algorithmic terms would mean returning the maximum similarity achieved between the input event and each of the beacon events. This approach, however, would increase significantly the output noise of the system and make it especially unstable in case of a less than perfect beacon event selection.

Therefore, for the final layer of the proposed similarity metric, it is necessary to forgo compliance with the requirement on maximum similarity. The prototype implementation uses a simple arithmetic mean, averaging the similarities calculated for the input event with each beacon event:

$$\mathcal{S} \equiv \mathcal{SS}(i) = E(\mathcal{ES}(i,k)) = \frac{\sum_{k=1} \mathcal{ES}(i,k)}{|\mathbb{B}|} \tag{17}$$

For the events in $\mathbb{EX}$, the values of $\mathcal{ES}$ from Section 5.3 are aggregated into the final similarity metric $\mathcal{S}$, clearly separating the initially targeted events e_1 and e_3, and allowing the subsequent use of various algorithms for the final event *selection*:

	e_1	e_2	e_3	e_4
$\mathcal{S}$	0.538	0.001	0.537	0.001

6. Adaptivity of the proposed approach

For a real-time event processing system to be adaptive there are two orthogonal issues that must be taken into account. First, the system should be able to handle *new attributes* as they appear, something that may happen when a new part of the monitored infrastructure comes on-line or an architectural alteration occurs. This requirement is increasingly important for a domain-agnostic system, where there is no knowledge of the schema and no assumption regarding the number, type or prevalence of attributes can be made beforehand. Second, the monitoring system should be able to adjust to new operational trends, where certain events may become more frequent, while other may disappear altogether.

6.1. Adaptivity to schema changes

The proposed approach is completely domain agnostic, requiring no *a priory* knowledge of the domain or the event schema. As such, it operates on the inherent assumption that only part of the actual schema has been considered at any given time. An incoming event may contain a number of previously unseen attributes that expand the existing view of the schema of the monitored domain. Each new attribute is essentially treated as an independent time series that only contained *null* values until its time of emergence.

However, for most event domains and monitoring systems it is reasonable to assume a *finite set of attributes*. In other cases it is possible to use a domain-specific pre-processing stage, to map, for example, dynamically generated attributes into a more restricted schema. Therefore, the set of attributes that are known to the system at runtime will gradually converge towards a constant set, with few or no new attributes being added after a sufficient amount of time.

6.2. Information content aging

Typical statistical algorithms are generally targeted at data *sets* and not well suited for streaming input. Therefore, their use is associated with several caveats that require handling:

1. Common set-based metrics exhibit a form of *inertia* as the number of recorded data points increases. New data have a decreasing effect as time passes, creating an asymmetry between data received at the early and later stages of execution. This creates both a conceptual and a practical problem by essentially nullifying the effect of recent events on the system state, reducing to zero its adaptability to new system configurations.
2. Cumulative metrics, such as the mean and standard deviation, suffer from numerical range and precision issues when implemented trivially on a computer. Using software libraries for arbitrary-precision arithmetic alleviates this problem to some degree, generally at the cost of increased code complexity, reduced portability and a significant loss of performance.
3. Most information content metrics tend to rise in a roughly linear manner as the number of processed input events increases. This creates significant numeric stability issues, as the precision of several mathematical operations decreases when the operands exceed the representation capabilities of the underlying computer architecture.

To avoid these issues, the proposed system makes use of object replacement algorithms that *remove* extremely old and irrelevant values from its knowledge base, as mentioned in Section 7.1. In addition, it utilizes methods that *devalue* or *age* older data points with the passage of time. Daneshgaran and Mondin (1997) provide an information theoretic definition of aging, while Cormode et al. (2009) describe a generic exponential decay model for aggregate metrics.

One of the concepts that is unavoidably connected with aging is *time*. Real, or wall-clock, time is not always easy to handle algorithmically. In some cases even its definition is not straightforward from a systemic point of view. For example, basic deterministic automata are designed based on state change sequences and do not handle the concept of time at all. Similarly, our prototype implementation synchronizes using input events as a reference and ignores any other perception of time. As a result, the decay model used for the aging process, is a simple step-wise exponential decay model. For each *cycle*, the information content currently recorded for all attributes and attribute values is multiplied by a positive *decay* coefficient no greater than 1:

$$I_{aged,t+1} = decay \cdot I_t + (I_{t+1} - I_t) \tag{18}$$

The decay coefficient should be selected with regard to the monitored system, most notably its event rate and other temporal characteristics. Typical values for the *decay* coefficient during experimentation were in the [0.95, 1) range - a common value of 0.9999 results in a 63% decay over a period of 10,000 events.

7. Data structures and memory management

7.1. Space complexity and object replacement algorithms

Since large systems have an infinite, or at the very least *unknown*, limit for the length of their emitted log streams, it is important for tractability purposes to keep the space complexity, i.e.

the volume of the events being considered at any given time, constant or at most sub-linear, with respect to the overall size of the log streams. This requirement, however, cannot be satisfied without modifications to the theoretical constructs presented in Section 5. The proposed similarity metric requires the ability to provide the information content contributed by each discrete value for all attributes, which implies at the very least, the storage of an occurrence counter for each attribute value.

Using the limited attribute set assumption mentioned in Section 6.1, the time and space complexity of the system becomes linear with regard to the number of attributes, which can be considered a characteristic of the monitored infrastructure and thus constant in time. The same assumption cannot be made for the attribute *values*. In the general case, where attributes may even correspond to floating point or string variables, there is no limitation for the number of discrete values that may appear for each attribute. With a macroscopically stable rate of new values, the need for a separate counter and other metadata per value imposes a linear space requirement with respect to the total size of the input.

This makes the use of the basic similarity metric non tractable for event stream processing systems. For this reason, it is necessary to *approximate* the operation of the similarity determination algorithm in a space-efficient manner. From the mathematical formulas, it is clear that the values with the least impact are those with the lowest contribution of information content. Therefore, a potential approximation involves eliminating the entries that correspond to low-impact values in order to form a *sketch* of the value distribution for each attribute, instead of maintaining an accurate count for each value.

While it is theoretically possible to use the information content itself as a basis for this task, this implies re-calculating the contribution of all retained values with each input event, thus increasing the time complexity of the system with respect to the volume of retained data.

Taking the effects of the aging process into account, one can intuitively identify the low-impact values as those that (a) are infrequent or (b) have not appeared recently in the input stream. Selecting items based on frequency and recency is essentially the purpose of *object replacement algorithms*, more commonly known as *caching algorithms*. The main intent of a caching subsystem is to use a pre-determined amount of space to store results of past requests, replacing (*evicting*) old or infrequent entries when necessary. Therefore, we can use such replacement algorithms to provide a hard limit for the space usage of the proposed system.

One of the most widely known entry eviction methods is the Least Recently Used (LRU), an algorithm with O(1) complexity for all operations which drops the entries that were last used furthest in the past. It is essentially a finite-size linked list and thus very simple to implement, but it behaves incorrectly when faced with a large block of unique entries on its input, since said entries will replace the whole content of an LRU instance without being actually useful.

Another well-known method is the Least Frequently Used (LFU) algorithm, which drops the objects with the lowest frequency. While LFU seems by definition closer to the ideal approach, its complexity is logarithmic for all operations. In addition, it is not adaptive, since it tends to fill-up to its capacity with old frequently-repeated entries that may be currently irrelevant on a real-time system. There have been several attempts to address these shortcomings by e.g. combining LFU and LRU into algorithms, such as LRFU, that take both recency and frequency into account.

For our prototype implementation, we selected the *Adaptive Replacement Cache (ARC)* algorithm by Megiddo and Modha (2003), due to its simplicity and overall performance. It provides a good balance between recency and frequency, while also being resistant to pathological behaviors and able to adapt to its input.

ARC uses a pair of LRU lists ($T1$, $T2$) to store live objects, as well as another pair of *ghost* LRU lists ($B1$, $B2$) to store recently evicted keys. Within each pair one list tracks keys that have been encountered only once ($T1$, $B1$), while the other contains keys that have been *repeated* at least once ($T2$, $B2$). Hits on the *ghost* lists are used as an indicator for the automatic tuning of the relative sizes of the corresponding live object lists, removing the necessity for any external parameterization that would require domain-specific knowledge. All ARC operations happen in constant (O(1)) time, which is critical for low-latency, high-throughput applications.

7.2. iTree: *A data structure for computing information metrics in streams*

While the information theoretic metrics previously described are well defined for offline use on an input event *set*, their computation proves significantly more difficult when they are used on an event *stream*, due to a number of complicating factors. To illustrate the problem, consider the non-aged information content of an attribute $I(a_j)$:

$$\begin{aligned} I(a_j) &= -\sum_{i=1}^{|\mathbb{V}_j|}(n_{j,i}\cdot log_r\frac{n_{j,i}}{|\mathbb{E}|}) \\ &= -\sum_{i=1}^{|\mathbb{V}_j|}(n_{j,i}\cdot log_r n_{j,i} - n_{j,i}\cdot log_r|\mathbb{E}|) \\ &= |\mathbb{E}|\cdot log_r|\mathbb{E}| - \sum_{i=1}^{|\mathbb{V}_j|}(n_{j,i}\cdot log_r n_{j,i}) \end{aligned} \tag{19}$$

While the first term can be computed reliably in a streaming environment simply by storing the current $|\mathbb{E}|$ value, the second term is significantly more problematic. The direct approach would be to maintain the sum as a variable, each time subtracting the corresponding previous value and adding the new one. Due to the limited precision of floating point operations on current computers, however, this approach is numerically unstable - adding and then subtracting the same number from a sum is not guaranteed to result in the original sum value.

The introduction of the aging coefficients worsens this instability problem, since their use implies that the term subtracted from the sum in question needs to have been already multiplied by an appropriate decay factor. For example, to update a weight sum W when one of its aged component weights w increases after five aging cycles, we will have $W_t = W_{t-1} - decay^5 w_{t-5} + w_t$. Essentially, while the value originally added was w_{t-5}, we now have to subtract its aged equivalent before adding its updated value w. When the intervening time intervals are large or the decay coefficient is relatively low, the value to be subtracted can be several orders of magnitude lower than the original, which causes a measurable loss in precision. The accumulating errors during such a streaming operation soon lead into completely incorrect results. Therefore, in order to ensure adequately the precision of these metrics in a streaming environment, it is preferred to completely recompute each sum from the beginning, rather than update it using the existing value. A naive approach would have all terms existing in a linked list and serially added after each update, with a resulting complexity of $O(n)$ with respect to the number of terms. By using a more complex data structure, though, it is possible to lower that complexity to $O(logn)$.

For the prototype implementation of the proposed system, we present a composite data structure we refer to as *iTree*, in order to address the aforementioned issues. The *iTree* combines elements from several well-known data structures, in an attempt to address

the shortcomings that each one presents on its own for our specific use case:

1) *Hash table* A hash table is used to enable constant average time ($O(1)$) access to each stored entry, mapping each attribute value to a record that contains its observed frequency, the corresponding aging coefficient and other metrics. Apart from a significant performance benefit, using a hash table for the entry retrieval operation removes the need to implement an entry search capability through any other data structure, simplifying their design.
   The implementation uses entry chaining via linked lists to overcome the issue of hash function collisions. The use of linked lists leads to a linear worst-case complexity for entry retrieval operations within *iTree*. Therefore, in pathological situations where the hash function is not effective, it is possible for the performance of the structure to degrade. Such an issue was not observed in our experimental system, although it is theoretically possible. To partially address this liability, it is possible to achieve a logarithmic worst-case complexity for retrievals by adaptively using an entry *tree*, instead of a linked list, in pathological situations of hash function collisions.
2) *Binary tree* An array-backed binary tree is used with the sole purpose of organizing all entries in a binary hierarchical structure. This structure has no sort order and no constraints, apart from the requirement to always form a complete binary tree.
   By arranging all entries in a complete binary tree structure, it becomes possible to update any global sum in logarithmic ($O(logn)$) time. Once the individual metrics of an entry are updated, the tree is traversed upwards from that entry towards the root, updating partial sums that are recorded on each ancestor. The root node then provides the necessary sums for all nodes. Since the number of updated nodes is bounded by the height of a complete binary tree and the retrieval of the starting entry itself occurs in constant time through the hash table, the total complexity of the update operation is $O(logn)$.
   Both the entry insertion and entry removal operations have the same logarithmic complexity as well. Since the tree has no associated constraints, inserting a new entry involves simply adding the entry at the first free array index and then updating all ancestor nodes until the root node is reached. The cost of any array resizing operations, when amortized over all insert operations, is still a constant, leaving the cost of updating the ancestors as the determinant factor. Delete operations are more expensive computationally, since the removed node is essentially replaced by the node that was last inserted, but the overall complexity remains unchanged.
3) *Object replacement algorithm* A modified implementation of the *Adaptive Replacement Cache (ARC)* object replacement algorithm allows for entries to be evicted, restricting the size of the *iTree* structure to a specified limit, as mentioned in Section 7.1. The existing hash table is used for constant-time entry retrieval, while a number of interconnected doubly-linked lists form the internal structures required by the ARC algorithm.
   The ARC implementation in *iTree* incorporates two modifications over the standard version of the algorithm. First, it features the ability to *protect* entries from eviction, which is especially important for the attribute values that are contained in the beacon event set and should, therefore, never be removed. Second, ARC is defined with the existence of a backing main object store in mind - misses during a retrieval operation result in the object being implicitly pulled from the backing store. In contrast, *iTree* is always used in *push* mode, with values being explicitly added when retrieved from the input stream. The ARC implementation in *iTree* has been modified to operate correctly in this context.

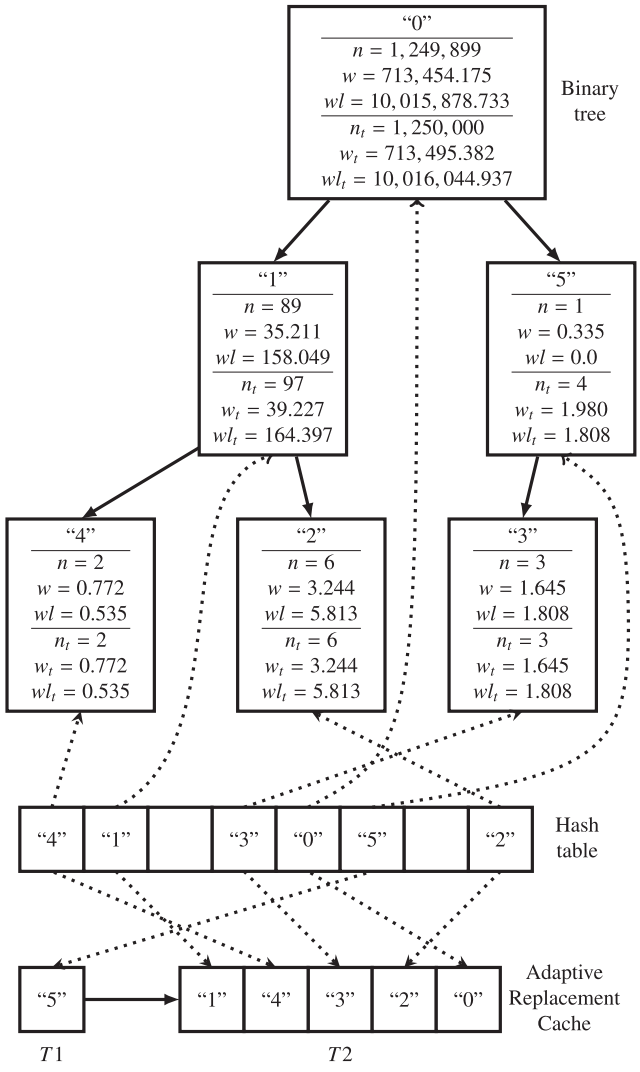

Fig. 5. An example of the *iTree* data structure, illustrating the accumulation of the frequency n and the weight w of each value, along with the product $wl = w \cdot ln(n)$, into the sub-tree measures n_t, w_t and wl_t.

Fig. 5 shows an example *iTree* instance for a single attribute, derived from an actual run of the prototype system. For each attribute value v that is known, e.g. "0" or "2", there is a related binary tree entry that contains the observed frequency n of v in the input event stream and its aged weight w, while the product $wl = w \cdot ln(n)$ can be computed on-demand. In addition, each tree entry contains the sub-tree measures n_t, w_t and wl_t that sum the corresponding fields from the current entry and both of its descendant sub-trees, if any. For reasons of visual clarity, a number of aging-related fields have been taken into account for the shown values and then omitted. It is evident that the n_t, w_t and wl_t fields of the root node provide the overall sums that may be used to calculate any information theoretic metrics that consider all entries, while any update only needs to propagate upwards towards the root, resulting in a logarithmic complexity versus the total number of entries.

To illustrate the use of the *iTree*, let us suppose that value "4" has just been encountered for the second time in the input stream, and the corresponding node has just been updated. Since it is a leaf, its sub-tree fields n_t, w_t and wl_t will be equal to its corresponding node fields n, w and wl. Focusing on the occurrence counters n and n_t, it is therefore the case that $n_t = n = 2$ (since the value "4" has been encountered twice). Visiting its parent node

"1", the sub-tree fields of "1" are reevaluated as the sum of its own node fields and the corresponding sub-tree fields of both its children. For example, $n_{t,\text{"1"}} = n_{\text{"1"}} + n_{t,\text{"4"}} + n_{t,\text{"2"}} = 89 + 2 + 6 = 97$. The root node "0" is then updated in the same manner and we have

$$n_{t,\text{"0"}} = n_{\text{"0"}} + n_{t,\text{"1"}} + n_{t,\text{"5"}} = 1{,}249{,}899 + 97 + 4 = 1{,}250{,}000.$$

Using the same principle as with $n_{t,\text{"0"}}$ the *iTree* data structure can be used to compute $w_{t,\text{"0"}}$, $wl_{t,\text{"0"}}$ and the other tree coefficients. Therefore, using the *iTree* data structure we can efficiently avoid the mathematical precision issues that arise when computing Eq. (19) and other metrics.

8. Scalability through a distributed operation model

Regardless of the efficiency of any event processing algorithm, modern and interconnected large IT infrastructures are capable of producing very high volume and frequency data that is difficult to processes on a single computer system. In order to be able to handle multiple high-speed event streams in a scalable manner, it is necessary to distribute the processing to multiple computers, while also using a reasonable amount of resources in comparison to the monitored systems.

8.1. Workload distribution model

The scalability of the proposed system can be facilitated by distributing the performed work using two orthogonal methods:

1) *Attribute-level parallelization* Each processing node handles a subset of the attributes and their corresponding values from each input event e_i. This method does not alter the semantics of the previously described algorithm, since each attribute is treated as a completely independent time series. However, it is limited in effectiveness, since the average number of attributes within each event e_i may often be less than the number of available processing units when using a sizable computer cluster. In that case some of the nodes will be underutilized, even if the system itself has reached its saturation point.
2) *Event level parallelization* Since the proposed framework analyzes each input event e_i independently, it is feasible to distribute the input stream to a large number of processing nodes using any load-balancing method, such as a round-robin scheduler. However, while this approach allows for a greater degree of parallelization, it also alters the semantics of the similarity algorithm. More specifically, since each processing node handles a proportionate part of the received events, it only has a partial view of the input stream.

In addition to these two cluster-wide distribution approaches, the individual processing nodes themselves may optionally employ *beacon-level parallelization*. In order to produce a composite metric, the algorithm iterates over the corresponding beacon event set values for each attribute, with the intermediate similarity values being computed independently on a separate processor core. This increases the parallelization of the overall architecture, although the potential gains will be limited in most cases due to the small size of the beacon event set.

In order to build a distributed implementation of the proposed approach, a variation of the MapReduce programming model was defined using five node types: an event source node, a mapper node, a processing node, a reducer node and an output node. The main difference from the standard MapReduce model lies in the existence of the processing node between the mapper and reducer nodes. This allows for the computationally intensive processing to be separated from the I/O-intensive operations of the mapper and reducer nodes. The nodes communicate by exchanging tuple objects that may contain any simple or composite object, while separate control channels allow the transfer of critical metadata.

Fig. 6 provides an overview of an event processing network with m mapper nodes, p processing nodes and r reducer nodes, demonstrating the input and output tuples for each node. The control channel tuples are not shown in this figure, as they are to a significant degree implementation-specific.

8.1.1. Source node

The source node, or *S-node*, feeds an input event stream into the topology. Each *S-node* corresponds to a single input event stream *src* than can be either received passively (e.g. a system log feed) or retrieved actively from remote sources. The high-level operation of the node is described in Algorithm 1. Each retrieved

Algorithm 1: Source node operation.

input : event source stream *src*
output: *S-tuple* stream

```
1 i ← 0;
2 while inputAvailable(src) do
3 |   e_i ← get(src);
4 |   emit({src, i, e_i});
5 |   i++;
6 end
7 sleep();
8 emitControl({src, END, i});
```

event e_i is emitted (line 4) as a tuple of the format {*src, i, e_i*} (*S-tuple*) that contains the unique stream identifier *src*, the event counter *i*, as well as the composite event object e_i itself. A control channel is used to signal (line 8) the termination of each stream to the associated output node, allowing for the corresponding output stream to be closed as well.

8.1.2. Mapper node

The first task of the mapper node, or *M-node*, is to *flatten* (Algorithm 2, line 1) each received *S-tuple* {*src, i, e_i*} converting it

Algorithm 2: Mapper node operation.

input : an *S-tuple* $\{src, i, e_i\}$
output: an *M-tuple* for each attribute

```
1 {src, i, v_{1,i}, v_{2,i}, ..., v_{|𝔸|,i}} ← flatten({src, i, e_i});
2 for j ← 1 to |𝔸| do
3 |   emit({src, i, a_j, v_{j,i}});
4 end
```

into an intermediate *I-tuple* of the format $\{src, i, v_{1,i}, v_{2,i}, ..., v_{|\mathbb{A}|,i}\}$. Afterwards, the *I-tuple* is mapped into an equivalent collection of attribute/value tuples of the format {*src, i, a_j, $v_{j,i}$*} (*M-tuple*) and then emitted (line 3) towards the processing nodes.

There are no constraints with regard to the allocation of source tuples to specific mapper nodes, apart from performance and load balancing concerns. Mapper nodes are completely stateless, which makes them equivalent at any point in time.

8.1.3. Processing node

The processing node, or *P-node*, performs the core part of the similarity computation, operating at the event attribute level. Each

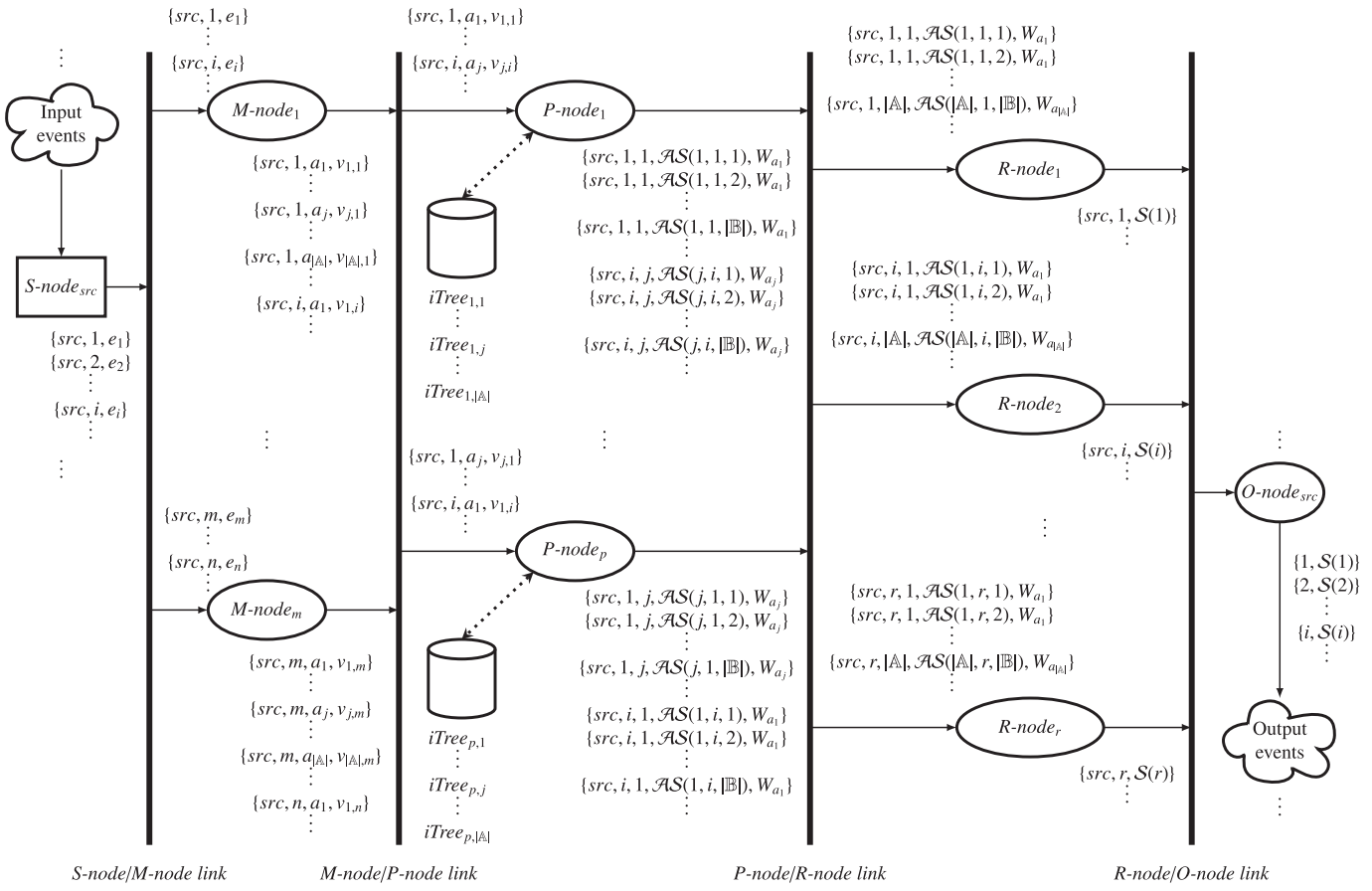

Fig. 6. Distributed operation overview.

M-tuple generated from an *M-node* is randomly distributed to a *P-node* instance and, therefore, each stream attribute may be handled by more than one *P-node* instance, with each one receiving an equal but randomly assigned part of the input.

For each input *M-tuple* $\{src, i, a_j, v_{j,i}\}$ that is received from the *M-node* instances, the contained value for attribute a_j is compared against the corresponding value of each beacon event b_k using the information-theoretic metrics that were described previously. Thus we calculate the attribute-level similarity value $\mathcal{AS}(j, i, k)$ and the weight W_{a_j} as presented in Eqs. (14) and (15). Each partial similarity result and the associated weight are then emitted (Algorithm 3, line 2) towards the reducer nodes as a tuple of the format $\{src, i, j, \mathcal{AS}(j, i, k), W_{a_j}\}$ (*P-tuple*).

Algorithm 3: Processing node operation.

input : an *M-tuple* $\{src, i, a_j, v_{j,i}\}$
output: a *P-tuple* for each beacon event b_k

1 **for** $k \leftarrow 1$ **to** $|\mathbb{B}|$ **do**
2 | `emit`$(\{src, i, j, \mathcal{AS}(j, i, k), W_{a_j}\})$;
3 **end**

The processing nodes are the most computationally-intensive elements of the topology. They incorporate an *iTree* caching value repository for each attribute that they handle. These repositories are updated in real-time, along with the aged information content metrics that are associated with each attribute a_j and then used to generate the coefficients for the similarity calculation.

8.1.4. Reducer node

The reducer node, or *R-node*, collects (Algorithm 4, line 3) the partial results from the processing nodes and aggregates (lines 7, 9) them to produce an overall similarity metric $\mathcal{S}(i)$ between each event e_i and the beacon event set $\mathbb{B}$. The configuration of the *P-node*/*R-node* link ensures that all *P-tuple* objects that relate to a specific event e_i are delivered to the same *R-node* instance. Messages received via the separate control channel from the mapper nodes allow the reducer nodes to obtain critical metadata regarding their operation, such as the number of attribute/value tuples

Algorithm 4: Reducer node operation.

input : all *P-tuple* objects $\{src, i, j, \mathcal{AS}(j, i, k), W_{a_j}\}$ for an event e_i
output: an *R-tuple* for e_i

1 **for** $j \leftarrow 1$ **to** $|\mathbb{A}|$ **do**
2 | **for** $k \leftarrow 1$ **to** $|\mathbb{B}|$ **do**
3 | | $r[j][k] \leftarrow \{src, i, j, \mathcal{AS}(j, i, k), W_{a_j}\}$;
4 | **end**
5 **end**
6 **for** $k \leftarrow 1$ **to** $|\mathbb{B}|$ **do**
7 | $es[k] \leftarrow \mathcal{ES}(i, k)$;
8 **end**
9 $\mathcal{S}(i) \leftarrow E(es[k])$;
10 `emit`$(\{src, i, \mathcal{S}(i)\})$;

generated from each input event e_i. The aggregated results are then emitted as a tuple of the format $\{src, i, \mathcal{S}(i)\}$ (*R-tuple*).

A point of importance is that in general each *R-node* instance only handles a randomly selected part of the output that corresponds to each input stream. The final output stream has to be reconstituted from the *R-node* sub-streams by the output node instances.

8.1.5. Output node

The output node, or *O-node*, generates the final stream that maps each input event e_i to the computed similarity. The communication channels between the reducer and output nodes are configured so that the results from each separate input stream will end up in the same output node, forming a coherent output stream that can be delivered (Algorithm 5, line 3) to a single destination for each input event stream *src*.

Algorithm 5: Output node operation.

input : all *R-tuple* streams for *src*
output: result stream for *src*

```
1 while not receivedControl(src, END, *) do
2 |   {src, i, S(i)} ← receive()
3 |   output(src, {i, S(i)});
4 end
```

Since the *O-node* instances have a complete view of the output stream, they are also able to perform any per-stream post-processing operations, such as reordering the result tuples to match the input order. Depending on the configuration of the node network, the reduced output selection process described in Section 9 may be incorporated in the operation of either the reducer or the output nodes. Using this process it is possible to either *tag* each *R-tuple* appropriately, or even to completely remove low-similarity events from the output.

Using the *R-node* instances allows the selection load to be distributed to multiple physical computers, which may be necessary for high-volume input event streams, but can have a negative impact on the accuracy of the overall process, since each *R-node* has a limited view of the overall output stream. While this loss of accuracy may be acceptable - or even unavoidable - for high-volume streams, such a reduction of the output quality may be undesirable in cases of low-volume input.

The possibility of executing the selection process in the *O-node* instances addresses this issue, since the output tuple stream that corresponds to each input event stream is completely accessible within a single *O-node* instance. Alternatively an independent similarity threshold selection node (*T-node*) may be used, allowing complete flexibility based on the specifics of each installation.

8.2. Distributed operation considerations

8.2.1. Input stream splitting and value repository segmentation

Distributing the operation of the proposed algorithm unavoidably results into changes of its semantics. The most important change is the fact that there are now multiple caching value repositories for each attribute, with the value distribution *sketch* for each attribute being segmented to multiple nodes. Each repository - and its associated processing node - only views a limited, mostly random, part of the input value stream for that specific attribute. Since the contents of the repositories affect the calculation of the weighing coefficients presented in Section 5, this may have a negative impact on the quality of the produced results by increasing the number of events that need to be processed by the system in order to generate results with a certain degree of confidence.

In many cases, however, this issue can be ignored if a certain degree of approximation - and a corresponding error ratio - is acceptable. It is often reasonable to assume that *any sizable subset of the input stream that is evenly distributed in time will be macroscopically representative of the whole input stream*. Therefore, in the long term each node will be able to compute accurate enough metrics using only the event stream subset that it has received. In general, the distributed system after i events is expected to approach the behavior of a single-node system with i/p received events, where p is the number of *P-node* instances.

While it would be possible to ensure the conformance with the single-node version of the algorithm by distributing all values to all nodes or by sharing a single value repository, any such design would mostly defeat the purpose of creating a distributed implementation. As a mitigation method, however, it is possible to reduce the error ratio and avoid extreme cases of divergence, by having the system periodically synchronize the recorded metadata among all nodes, thus setting an upper boundary for the deviation between the different repository segments of each attribute.

8.2.2. Input event reordering and non-determinism

In addition, on a distributed system the interactions of the various networked components are affected by timing factors that are not really controllable. The random distribution of the data to the various nodes, the introduced networking latency and the process scheduling intricacies of the physical nodes themselves all affect the order in which events are processed. As a result, a final reordering of the output (e.g. Mutschler and Philippsen (2013)) may be necessary.

A final point relates to the system evaluation process itself. On a single-stream, single-computer system the output of each testing run is generally deterministic and only depends on the input event stream itself. On the contrary, the equivalent distributed system will produce different results on each run due to the effects discussed above. This may present a degree of difficulty when evaluating the system, especially when targeting input event stream subsets of miniscule size. Performing multiple runs for each evaluation and aggregating the results reduces the error rate to an acceptable level.

9. Adaptive output event stream selection

Conceptually, in order to make use of the similarity algorithm it is necessary to select an output event set with those events for which the highest similarity values were computed. Providing a *threshold* for that selection process, however, is not trivial. Real-time streams are unbounded with regard to space and impose certain latency constraints, while the amount of information that is known a priori is limited. Therefore an *adaptive* threshold selection method should be used for the last phase of the process.

Some common alternative group selection approaches include:

Sorting: Sorting items based on their similarity to the beacon event set is a typical scenario employed by offline grouping engines. Despite its conceptual simplicity, this approach is not tractable for event streams. Real-time sorting algorithms such as various insertion-sort implementations, are available, but they generally exhibit linear space complexity which makes them unusable for input streams with unrestricted length.

Percentile extraction: If the objective is the reduction of the input stream by a specific ratio, several efficient methods (Zhang and Wang (2007), Cormode et al. (2006)) for the approximate determination of percentiles on streams have been proposed. The use of approximate methods allows such algorithms to be effective on large input streams, typically with sublinear space complexity. On

the contrary, sorting-based accurate algorithms are essentially limited to fixed sets. However, the main weakness of this alternative is the determination of the desired output percentile. Since this is a fixed setting, it generally requires domain-specific knowledge so that the target events will not be missed, while also keeping the output noise to a minimum.

Threshold-based selection: While a fixed value cannot be used, there are several possibilities for a more adaptive threshold selection. For example, Lane and Brodley (1999) use a probabilistic method to compute a threshold, based on a desired ratio r of selected events; essentially an indirect approach to percentile extraction, with r being a site-specific setting. Alternatively, a univariate clustering/classification algorithm (e.g. Qiu and Tamhane, 2007) could be used to select those events with the highest similarity values that are adequately close to indicate a coherent part of the input, while also being separated from the rest of the events. It is also possible to use statistical measures for the dynamic determination of a threshold. For example, the standard score, which is defined as the deviation of a specific sample from the mean of a variable measured using the standard deviation as a unit, has been tried with moderate to good results in our prototype system:

$$Z(x) = \frac{x - \bar{x}}{s} \tag{20}$$

A standard score threshold of 1.0 will select those events that have a similarity to the beacon set higher than the average similarity by at least one standard deviation. Unfortunately, simple statistical measures like this one tend to suffer from an inertial effect in long-running streaming systems, as mentioned in Section 6.2. As a result, the quality of the produced results tends to decrease as the input stream evolves in time and new trends are encountered.

In our approach, the resulting event set is adaptively computed using a dynamic threshold, which is determined by detecting significant gaps in the distribution of similarity values. Such gaps near the high end of the similarity value range are often indicative of an underlying distinction between matching and non-matching input events.

More specifically, the system maintains a constant space *sketch* of the probability distribution function (PDF) of the similarity values. The sketch consists of a limited number of aggregate data points, with each data point representing a cluster of similarity results. New values are inserted into the cluster with the closest mean, with a heuristic algorithm being used to determine when two clusters should be combined to allow for the creation of a new one. Since the number of data points is fixed, all operations happen in constant time (O(1)) with regard to the number of inserted values. Consequently, the PDF sketch is updated with each new similarity value in real time, as the input stream is processed. Similar techniques based on sub-linear data sketches have been used for approximate percentile determination on real time streams (Zhang and Wang (2007), Cormode et al. (2006)). In addition, a degree of adaptivity is provided for the threshold determination algorithm by employing an object replacement algorithm that removes old and currently irrelevant aggregate data points.

To determine a selection threshold, an integration method is applied over the PDF sketch to produce the equivalent cumulative distribution function (CDF) of the similarity values. The CDF of the similarity essentially provides the ratio of rejected events for each possible selection threshold. Sharp rises in the CDF indicate the existence of a tight cluster around the same value, while a plateau is caused by the lack of any data points in the corresponding range. Therefore, using a rough computation of the derivative it is possible to detect plateaus in the CDF, which typically separate similarity value clusters. A heuristic algorithm is then used to select an appropriate threshold, by taking into account the distances between the highest valued clusters, their size and their position within the similarity value range.

Fig. 7 depicts a plot for a typical instance of the similarity CDF, captured while processing 1,250,000 input events with a beacon set of 20 events that match the *neptune* attack type from the KDD'99 data set. A gap in the similarity values, indicated by a plateau in the CDF plot for similarities in the approximate range of $17 - 32\%$, separates the matching events from the rest of the input stream. The selected threshold of $\sim 28\%$ preserves roughly 31% of the input stream, which is congruent with the ratio of *neptune*-type events in the same input range.

While the threshold determination process completes in constant time with respect to the input event stream size, it can become relatively expensive as the resolution of the sketch increases. Therefore new threshold values are determined periodically, rather than constantly, in order to reduce the computing overhead.

10. Implementation considerations

The *Apache Storm* framework provides a basic infrastructure for the creation, operation and high-level monitoring of complex distributed networks for real-time stream processing. In contrast to batch processing frameworks, such as *Apache Hadoop* and *Apache Spark, Storm* does not collect events in groups but rather operates on a record-by-record basis. It also does not provide any facilities for permanent or temporary data storage. Instead, it focuses on the fast transfer, routing and manipulation of object tuples. The tuples may contain serializable objects of any complexity, allowing the representation of both the input events and any intermediate data.

Each stream processing network, or *topology*, is defined at a high level as a directed data flow graph of the executed software. It consists of nodes, each representing a specific software unit, that communicate by exchanging object tuples in a well-defined manner. The *Storm* framework is able to execute multiple topologies simultaneously using a single cluster of physical computers. It creates an appropriate number of instances for each software unit, schedules their execution on specific physical computers and sets up the necessary communication channels, according to the topology layout and its parallelization configuration.

Each topology may contain nodes of two kinds:

- Spouts: Spouts are the nodes that act as tuple *sources*, feeding the rest of the topology. They may retrieve tuples from an external data store, remote sensors or even generate them internally.
- Bolts: Bolts are the nodes that perform the actual processing. In general they process each tuple that they receive and emit one or more tuples as a result. They may also act as *sinks*, by not emitting any tuples at all. This is typical of nodes that are charged with delivering the output of the system or with monitoring the internal operation of the topology.

Using the *Storm* framework, the parallelization approach proposed in Section 8 was implemented as follows:

1) *Source node* In our prototype implementation the *S-node* was built as a *Storm* spout. Each instance corresponds to a separate input stream and is able to retrieve events from a MongoDB database, a local file or a remote TCP/IP server. The input system supports various object serialization formats (e.g. JSON, Kryo) and compression algorithms (e.g. gzip, LZ4).
2) *Mapper node* The *M-node* is a *Storm* bolt that emits tuples to both the default channel and a separate control channel. The control channel is used to inform the reducer node instances of the number of attribute tuples that were emitted for each event.

   In order to reduce the overall strain of sending multiple messages on the cluster network, the mapper node may optionally

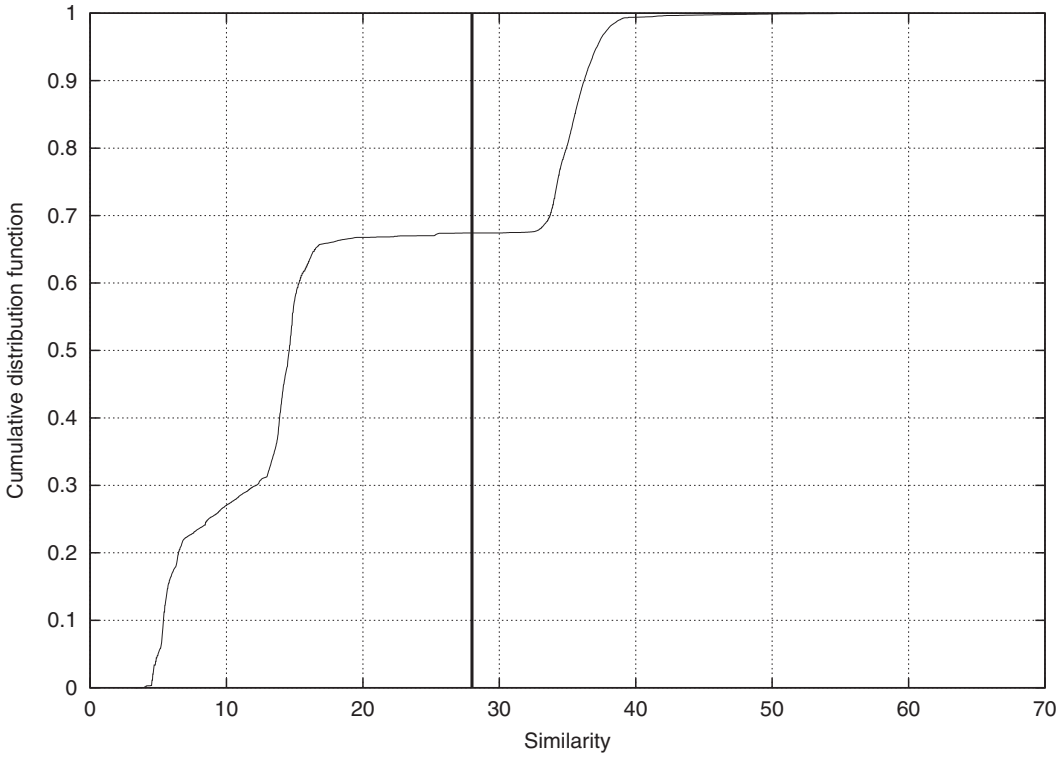

Fig. 7. Heuristic selection of a similarity threshold of $\sim$ 28% for a typical CDF instance.

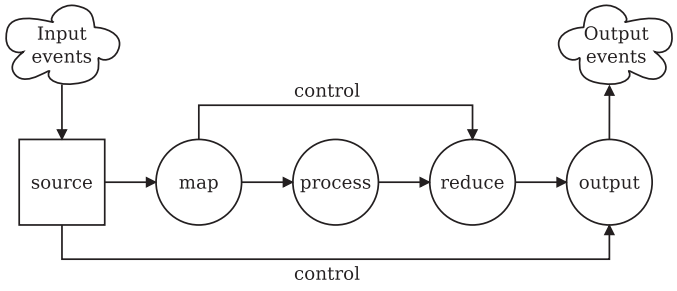

Fig. 8. The Apache Storm topology for the event filtering system.

batch multiple attribute/value pairs in a single tuple of the format {*src, i,* a_j[], $v_{j,i}$[]}. This essentially shares the explicitly-sent event metadata, as well as any implicit overhead, among multiple attributes.

3) *Processing node* The *P-node* is a *Storm* bolt that is typically instantiated in significantly higher numbers than any other node. The *P-node* instances generate multiple tuples for each received event attribute value $v_{j,i}$ - one for each beacon event. In order to reduce the bandwidth requirements imposed by the generated amount of data, and improve the overall performance a separate string registry server was introduced. While not part of the *Storm* topology, the registry server is used by all nodes. It converts each string into a unique numeric identifier that is shared among all nodes, allowing for all attribute names to be reduced into a single integer. The retrieved string identifiers are cached by each node, thus minimizing the resources needed for the operation of the registry server. In addition, the system may emit multiple result tuples in a batched form, similar to the one generated by the *M-node* instances, reducing the number of individual messages sent on the cluster network.
4) *Reducer node* The *R-node* instances make use of metadata received via the control channel to determine the number of result tuples that should be processed for each input event before a final result can be produced. However, the proposed parallelization approach makes no assumptions with regard to the order in which an *R-node* may receive results from the *P-node* instances. As a result, the *R-node* bolt instances in the prototype implementation may receive unordered partial similarity results from multiple input events.

It is therefore necessary for each *R-node* instance to maintain a table of events for which it has received either partial similarity results or control channel metadata. Each such event is recorded upon the reception of the first relevant tuple and removed from the table as soon as the final result is emitted. The memory requirements of the pending event table may be considered constant, as long as the following two assumptions hold:

- No tuples may be lost in transit. A lost *P-node* result tuple would result in the processing of the corresponding event to stall indefinitely. This is not a major issue, since the *Storm* framework provides a mechanism for the reliable transmission of tuples.
- The latency of the *P-node* processing stage is limited. As long as there are pending partial results, the *R-node* instances keep storing the incomplete event records in their local memory. Therefore the time it takes for the *P-node* instances to complete their operation affects the overall memory usage on the *R-node* hosts.

It is possible to mitigate any system-wide effects in cases where these assumptions are not valid by using either a timeout or a memory overflow handler that would remove pending events from the aforementioned table. That would improve the reliability of the processing cluster in high-load situations, at the cost of potentially producing partial results.

5) *Output node* The final *Storm* bolt is the *O-node*. In our prototype implementation the resulting output stream is delivered to a remote TCP/IP server, thus removing the need for the *O-node*

instances to be explicitly forced onto specific physical hosts with appropriate access to storage media.

11. System evaluation

For the evaluation of the proposed algorithm we used two different labeled data sets. The first one was generated within the scope of a competition that was held in conjunction with KDD'99, the Fifth International Conference on Knowledge Discovery and Data Mining. The second one consists of network traffic that was captured in the CTU University, Czech Republic, in 2011.

The KDD'99 data set (Knowledge Discovery & Data Mining Cup Data, 1999) is a derivative of the DARPA Intrusion Detection Evaluation 1999 corpus. As such, its events represent traffic within a computer network, while simulated security breaches are attempted. These attacks, of 38 types in total, can be grouped into four main classes; denial of service (DoS), remote probing, unauthorized remote access and local privilege escalation. The KDD'99 corpus, which has been extensively analyzed by Tavallaee et al. (2009), features about 4.8 million events and 41 distinct attributes, of both continuous and discrete types. The KDD'99 attributes have a minimum, median and maximum cardinality of 1, 87 and 21,490 respectively. They can be classified into three main categories: (a) basic features of an individual connection, such as its protocol, duration and transfer volume (b) extracted features regarding the content of a connection, such as the number of created files or shell commands issued (c) aggregate traffic features over a short time window, such as connection counts and various error rates. The last two categories essentially contain high-level information extracted over time, which means that each entry may actually represent multiple atomic entities.

For the purposes of our testing, we consider KDD'99 to be a stream of individual events. The events involved in each of the aforementioned attack type have relative frequencies within the input that span the range from 0.1% to about 30%, thus covering a wide range of different statistical behaviors. Apart from the attribute fields, KDD'99 contains embedded classification data (labels) for each event. While these labels are, naturally, excluded from any processing performed on the corpus, they can serve as a *reference* for the evaluation of the quality metrics of any selection method. This alleviates the need for domain-specific tools or manual intervention for the creation of the *golden standard* against which our system will be evaluated. This data set was used to evaluate both the quality and the runtime performance characteristics of the proposed approach, in both single-node and distributed operation.

The second corpus used was the CTU-13 data set (Garcia et al., 2014) and was selected in order to evaluate our algorithm against data from a different domain. We opted to use CTU-13, first because it deals with real world cases, second due to its use of the NetFlow event stream format which is an industry standard, and third because it is labeled, thus providing a reference ground truth for precision and recall evaluations. CTU-13 consists of about 20 million entries, which span 13 different scenarios involving the presence of malware in a computer network. The entries represent network traffic that was captured in a partially controlled environment, with real user traffic being mixed with traffic from malware which was intentionally installed for research reasons. For each scenario the data set provides a bidirectional NetFlow text file that aggregates the captured network traffic. NetFlow is a well-known industry file format for network flow data; in its text variation used by the CTU-13 set each entry corresponds to an individual captured network flow and has 14 comma-separated data fields (*attributes*) with a minimum, median and 90th percentile cardinality of 5, 100,900 and 4,237,587 respectively. They contain information such as the IP addresses of the endpoints, the transferred volume and the type of service. An additional field provides the *reference* label for each flow, classifying it into one of 1,400 categories that cover malware, normal and background traffic. Apart from its origin, schema, size and attribute cardinality distribution, the CTU-13 corpus differs from the KDD'99 set in the fact that each entry represents an independent event, with no high-level features. In addition, some attributes contain non-primitive values such as textual timestamps and IP addresses that can be hard to detect and compare effectively.

The prototype implementation was written in the Java programming language and tested extensively in streaming operation, both in single-node and distributed mode. The single-node version has been extensively optimized for use on a single computer with multiple processor cores, while the distributed version leverages the Storm framework to make use of a *cluster* of computers. Both versions incorporate the ability to compare the input stream with *multiple beacon event sets* simultaneously, which results in a significant performance increase by avoiding the repetition of common parts of the computation.

For each experiment, the beacon events used were selected randomly from a specific event type, with the rest of the events that belong to the same category being the expected result. No manual optimizations were attempted, as those would generally imply the use of domain-specific knowledge by the operator.

Compared with our prior work (Kalamatianos and Kontogiannis, 2014), which was also evaluated using the KDD'99 set, the proposed approach generally offers more uniform results over different beacon event sets and increased resiliency to the presence of noise in $\mathbb{B}$. For example, when used with a high ratio of erroneous beacon events, the current technique demonstrates noticeably higher precision for the *back* and *teardrop* event types, while providing significant throughput and scalability improvements.

11.1. Similarity metric evaluation

To evaluate the quality of the similarity metric, it is necessary to examine its selectivity between matching and non-matching events. Intuitively, an acceptable metric would provide results with clear *separation* between potential matches and the rest of the input, allowing the selection of the matching results by means of a simple similarity threshold. Ideally, the separating space would also not contain any noise in the form of infrequent errant values, although their effect on the overall accuracy of the system would be negligible and they could be filtered-out relatively easily using a variety of methods.

At the lowest level of the proposed approach lies the basic similarity function $\mathcal{VS}$. Since $\mathcal{VS}$ only takes into account the attribute values themselves, without considering their frequency or any other features, it is not directly usable as an overall similarity metric. Fig. 9 illustrates the results of the low-level similarity metric for a use case where the expected positives consist less than $1/500th$ of the input, mostly in two main clusters. It is evident that no conclusion can be formed with respect to the size or the location of the matching events with any degree of accuracy, since there is no clear separation between matching and non-matching events.

Applying the value coefficients and attribute weight functions described in Section 5 results in a more usable output (i.e. denser clusters), as depicted in Fig. 10. Both figures were produced using a beacon set of 20 events that match the *back* attack type from the KDD'99 data set. The input set consists of 1.25 million events and contains two event clusters of the same type in this particular range, in the subranges [395615 – 396614] and [498950 – 499951]. The corresponding similarity values rise from a baseline of about

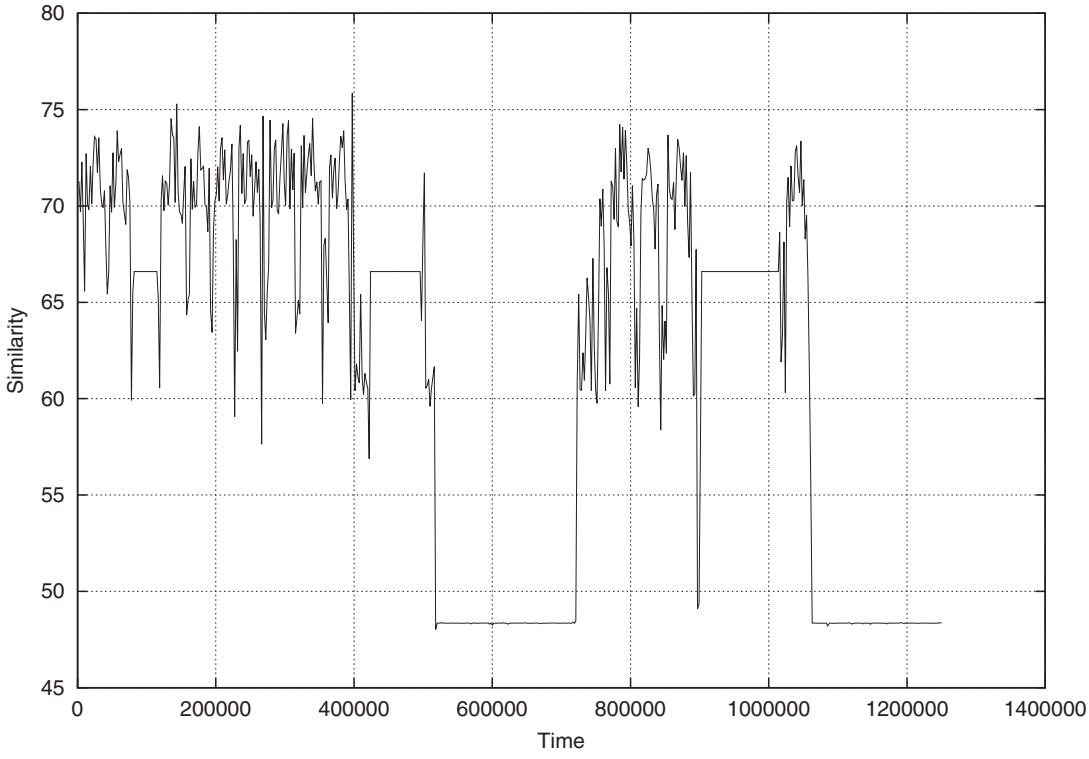

Fig. 9. Low-level similarity values for 1.25 million events.

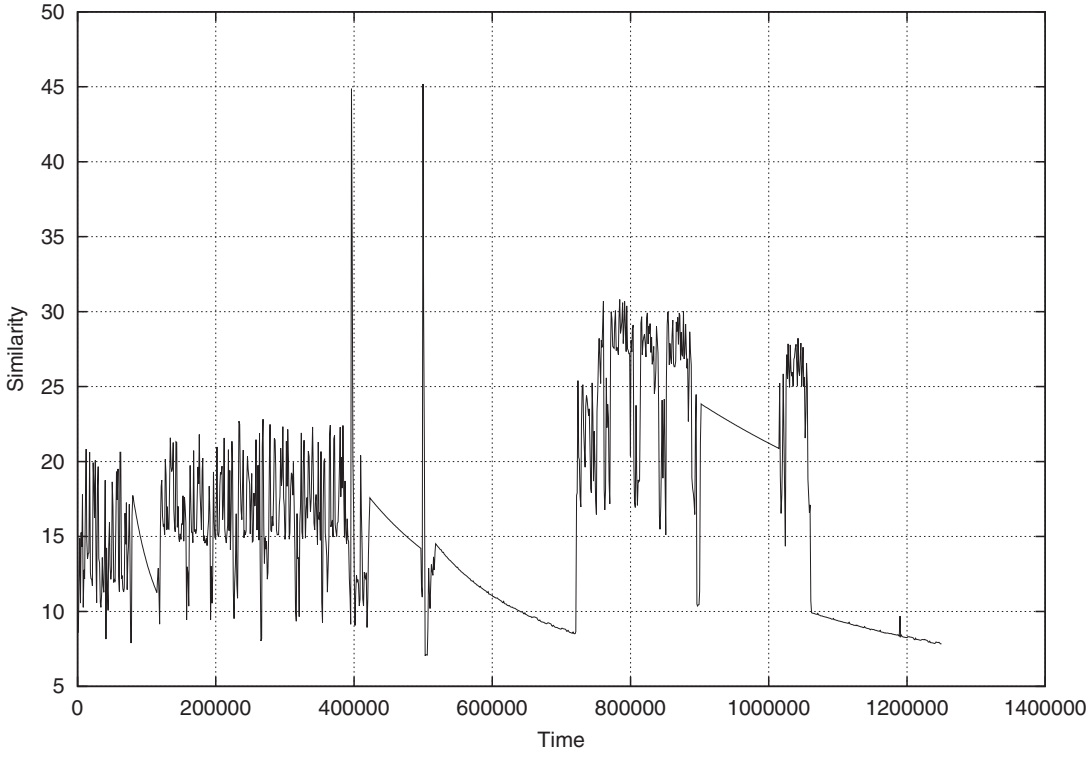

Fig. 10. Final similarity values for 1.25 million events with two visible clusters of potential matches.

5 – 22% to values in the 45 – 60% range, with virtually no noise in the intervening range from approximately 25% to 45%, thus being in complete accordance with the golden standard.

For a more concrete estimate, Buell and Kraft (1981) extend the common metrics of *precision* and *recall* for use in a fuzzy environment, where the output of a retrieval process is a continuous value, rather than a boolean result. This allows our similarity metric to be evaluated directly without interference from the threshold selection stage. In our prototype system, the cited *fuzzy* precision and recall metrics rose from $\sim$ 0.2% and $\sim$ 25% for the low-level similarity to $\sim$ 9% and $\sim$ 60% for the weighted similarity metric.

Table 1
KDD'99: Single-node operation, Beacon set size ($|\mathbb{B}|$) effect.

Test	$\|\mathbb{B}\|$	Positives	True positives	Precision (%)	Recall (%)	Reduction (%)	Accuracy (%)	F_2
back	5	3261	1974	60.534	98.601	99.739	99.895	0.875
back	10	2446	1980	80.948	98.901	99.804	99.961	0.947
back	20	2181	1992	91.334	99.500	99.825	99.984	0.977
imap	5	204,979	0	0.000	0.000	83.601	83.601	-
imap	10	395,032	9	0.002	75.000	68.397	68.398	0.0001
imap	20	638	10	1.567	83.333	99.948	99.950	0.072
ipsweep	5	7660	6243	81.501	82.372	99.387	99.780	0.821
ipsweep	10	7856	6653	84.687	87.782	99.371	99.830	0.871
ipsweep	20	7458	6650	89.166	87.742	99.403	99.861	0.880
neptune	5	317,740	317,311	99.865	80.370	74.580	93.766	0.836
neptune	10	204,621	204,383	99.884	51.767	83.630	84.747	0.572
neptune	20	395,235	394,812	99.893	100.000	68.381	99.966	0.999
nmap	5	416	10	2.404	0.432	99.966	99.783	0.005
nmap	10	8738	1019	11.662	43.998	99.300	99.279	0.283
nmap	20	8285	1014	12.239	43.782	99.337	99.314	0.288
portsweep	5	20,549	2465	11.996	88.605	98.356	98.528	0.389
portsweep	10	13,169	2585	19.629	92.919	98.946	99.138	0.531
portsweep	20	13,911	2506	18.015	90.079	98.887	99.066	0.500
satan	5	16,750	4765	28.448	88.355	98.66	98.991	0.621
satan	10	8776	4906	55.902	90.970	99.297	99.651	0.808
satan	20	8442	4865	57.629	90.210	99.324	99.672	0.810
teardrop	5	1655	199	12.024	100.000	99.867	99.884	0.405
teardrop	10	1685	198	11.751	99.497	99.865	99.881	0.399
teardrop	20	1533	184	12.003	92.462	99.877	99.891	0.395
warezmaster	5	302	16	5.298	80.000	99.975	99.977	0.209
warezmaster	10	270	15	5.556	75.000	99.978	99.979	0.214
warezmaster	20	184	16	8.696	80.000	99.985	99.986	0.303

Consequently, there is now a clear separation of the two matching clusters from the rest of the input stream, which allows the adaptive threshold selection algorithm to identify them correctly without introducing inordinate amounts of noise.

11.2. Quality of results (KDD'99)

An accepted method of assessing an information retrieval process is the measurement of *precision*, *recall* and *accuracy* values, along with the F_β composite measure for various values of β. For our system, recall is more important than precision, since it is critical to ensure that the technique does not discard matching events, especially in a streaming environment where the recovery of such events might not be possible. However, precision still remains an important quality, since it relates to the reduction effected upon the size of the input, which is the final purpose of the proposed system. Therefore, the F_2 measure is more suitable to be used for the overall assessment, since it is a common metric that favors recall without underestimating the importance of precision. It is also particularly useful for comparing the output quality of different approaches for the same input.

$$F_2 = 5 \cdot \frac{precision \cdot recall}{4 \cdot precision + recall} \tag{21}$$

For our evaluation with the KDD'99 data set, the prototype implementation was subjected to a series of experiments, a subset of which is presented in Table 1. Table 1 contains results from experiments performed *in single-node mode* over a range of 1,250,000 events for several different event types and for a variable beacon event set size $|\mathbb{B}|$. The selected scenarios cover a significant event frequency range, with *neptune*-type events comprising about 30% of the input, while the *back* and *teardrop* events correspond to 0.15% and 0.015% respectively. In addition, the expected events themselves present a variety of behaviors with regard to the distribution of their attribute values.

The results indicate that varying the size of the beacon event set, does not generally appear to have a consistently significant effect. In most cases, a larger number of beacon events tends to improve the overall behavior, essentially by allowing the system to more reliably establish which features characterize each event type, as evidenced by the extremely low recall achieved for event types *imap* and *nmap* with a 5-event beacon set. However, a larger size for $\mathbb{B}$ can also lead to a slightly higher amount of noise in the output when the target event set presents relatively limited cohesion and higher diversity among its members. In any case, the system is able to produce acceptable results with as low as 5 beacon events, which makes it usable by human operators without the aid of additional tools.

With 20 beacon events, the proposed approach manages to systematically retrieve over 80% of the requested events, with an accuracy of over 95%. The achieved reduction of the event stream volume is generally better than 95%, except for cases such as the *neptune* event type, where the expected output represents a significant part of the event stream and therefore cannot not be further reduced without losing useful information. For less frequent event types the recall is generally better than 80% and often 90%.

A notable exception is the *nmap* event type, with a maximum achieved recall of about 43%. In the context of the KDD '99 intrusion detection data set, this event type represents an attacker that serially attempts to connect to a number of different services. Therefore each event of this type contains a number of attribute values that are not only frequently encountered during normal operation, but that are also highly diversified within the *nmap* events themselves. As a result, our system is not able to reliably identify events of this type. A series of tests with a more relaxed threshold selection, where the automatically determined threshold was lowered by a fixed percentage, resulted in a recall in the 95% range, but with a significant drop in precision for most other cases. In general, a trade-off between recall and precision is expected, with the precision decreasing as the threshold is lowered. Conversely, a threshold increase may improve the precision, although if done indiscriminately it may also result in no events being selected at all.

The precision of the system is highly variable, since the adaptive threshold selection process has been designed with emphasis on recall and volume reduction. For frequent event types, the system achieves precision metrics better than 99%, avoiding the selection of inordinate amounts of non-matching events. On the other hand, event types that are extremely rare, such as the *warezmaster* type, are significantly more difficult to isolate, although the resulting output volume is still significantly less than that of the original stream. Moreover, since the expected output event count is very low, even a single erroneously selected event can reduce the recorded precision significantly. One such case is the *imap* event type, for which the prototype system was not very effective. Apart from the rarity of the events in *imap*, which numbered a total of 12 occurrences in a 1,25 million event stream, another contributing factor lies in the structure of the events in question. An investigation of the specifics of this event type revealed that each of its member events had a significant number of attribute values in common with numerous events from other types, while at the same time several other attributes displayed extreme variations within the *imap* type.

In order to assess the feasibility and the side-effects of using the proposed approach in a distributed manner, we repeated the exact same set of experiments on a Storm cluster composed of five identical computers, each with two physical CPU cores. Due to the inherent unpredictability of the distributed operation mode, which was mentioned in Section 8.2, successive repetitions of the experiment set resulted in slightly different outputs, although macroscopically there was little variation. The achieved results, which

Table 2
KDD'99: Distributed operation, Beacon set size ($|\mathbb{B}|$) effect.

Test	$\|\mathbb{B}\|$	Positives	True positives	Precision (%)	Recall (%)	Reduction (%)	Accuracy (%)	F_2
back	5	2209	1980	89.633	98.901	99.823	99.980	0.968
back	10	3802	1980	52.078	98.901	99.695	99.852	0.838
back	20	2149	1953	90.879	97.552	99.828	99.980	0.961
imap	5	204,730	0	0.000	0.000	83.621	83.621	-
imap	10	218,206	9	0.004	75.000	82.543	82.544	0.0002
imap	20	686	10	1.458	83.333	99.945	99.946	0.068
ipsweep	5	7715	6296	81.607	83.072	99.382	99.784	0.827
ipsweep	10	7856	6653	84.687	87.782	99.371	99.830	0.871
ipsweep	20	7458	6650	89.166	87.742	99.403	99.861	0.880
neptune	5	311,031	310,511	99.833	78.648	75.117	93.214	0.821
neptune	10	395,079	394,661	99.894	99.962	68.393	99.954	0.999
neptune	20	395,315	394,812	99.873	100.000	68.374	99.960	0.999
nmap	5	416	10	2.404	0.432	99.966	99.783	0.005
nmap	10	8739	1019	11.660	43.998	99.300	99.279	0.283
nmap	20	8499	1014	11.931	43.782	99.320	99.297	0.285
portsweep	5	22,380	2547	11.381	91.553	98.209	98.395	0.380
portsweep	10	17,948	2583	14.392	92.847	98.564	98.755	0.444
portsweep	20	18,056	2522	13.968	90.654	98.555	98.736	0.432
satan	5	15,348	4763	31.033	88.318	98.772	99.103	0.645
satan	10	8588	4909	57.161	91.025	99.312	99.667	0.813
satan	20	8055	4888	60.683	90.636	99.355	99.706	0.824
teardrop	5	1169	199	17.023	100.000	99.906	99.922	0.506
teardrop	10	1400	199	14.214	100.000	99.888	99.904	0.453
teardrop	20	1400	199	14.214	100.000	99.888	99.904	0.453
warezmaster	5	300	16	5.333	80.000	99.976	99.977	0.210
warezmaster	10	278	16	5.755	80.000	99.977	99.979	0.223
warezmaster	20	194	16	8.247	80.000	99.984	99.985	0.291

are presented in Table 2, were generally very similar to those achieved on the single-node version of the prototype system, as confirmed by a comparison of the corresponding F_2 metrics. This can be mostly attributed to the size of the input stream, which enables the assumptions mentioned in Section 8.2. Several tests with a reduced input volume of 50,000 events, however, demonstrated a visible reduction in the quality of results when compared with the single-node version for the same input, with the average precision dropping by almost 25%.

11.3. Quality of results (CTU-13)

To further validate the proposed algorithm, a more extensive series of experiments was conducted with the CTU-13 data set. This series of tests was specifically intended to evaluate our approach in an environment involving concurrent streams from multiple source and with disjoint label sets. Therefore, the NetFlow files from all 13 scenarios of the data set were merged into a single data stream, with all entries ordered using their time-stamp. For each flow category with 100 or more entries, 20 entries were randomly selected as the beacon event set. This resulted in 217 beacon event sets, each corresponding to a distinct category, which were then compared to the entirety of the merged data stream, consisting of 19,976,700 events. While the events of the other 1283 categories still remained in the data stream, they were not used as test targets in order to avoid artifacts related to their extremely low numbers. Table 4 provides statistics about the resulting quality metrics over all 217 cases, while some representative results of this test series are presented in Table 3.

For the CTU-13 data set, the proposed algorithm was able to provide a significant reduction in the volume of the input, with a recall better than 80% in 90% of the cases. However, merging the events from all CTU-13 scenarios into a single stream has caused a reduction in the measured precision in certain cases. While this approach creates a more realistic testing environment, careful inspection revealed that the labeling in the CTU-13 data set has not been fully normalized across different scenarios, causing certain event types to bear multiple labels in the aggregate stream. For example, the *From-Botnet-V50-2-UDP-DNS* and *From-Botnet-V50-4-UDP-DNS* network flow types, which are mentioned in Table 3 with relatively low precision, are essentially subsets of the same category, with no tangible difference between the events they encompass, except for their source scenario. To validate this rationale, we repeated the same testing methodology for the event stream generated by each CTU-13 scenario separately. The result was a significantly improved mean precision of about 68%, with a mean recall and reduction of 89% and 97% respectively.

11.4. Stability assessment

To assess the stability of the system we considered its behavior with respect to the presence of random noise in the beacon event set. This part of the evaluation is particularly important, since in real-life cases it is not always possible to define a beacon event set with absolute precision.

Table 5 presents experimental result metrics from the operation of the single-node version of the system for different ratios of randomly selected noisy events in a beacon event set of 20 events in total, for an input stream of 1,250,000 events from the KDD'99 data set. In general, low to moderate Beacon Noise Ratio (*BNR*) values, e.g. in the $10-30\%$ range, do not have a significant impact on recall, although the precision of the system tends to worsen. Higher amounts of noise can have a more visible impact, with the recall dropping to zero in some cases, while others remain unaffected. Conceptually, event types with limited cohesion, such as the *nmap* type, are far more susceptible to the negative effects of noise, since the beacon events feature a certain amount of noise themselves. For example, the *imap* event type, which was demonstrated to be problematic even for clean beacon event sets of a lesser size, becomes unretrievable even with very low amounts of noise in $\mathbb{B}$.

The same series of experiments were repeated on the aforementioned Storm cluster, with the results presented in Table 6. Once again the overall metrics are comparable to those retrieved from the single-node prototype system. However, there are cases, such as those of the *back* and *neptune* event types, where the quality degradation at high error rates is slightly more pronounced when operating in distributed mode.

It should be noted that these observations are only valid in the case of *random* noise. The effects of *cohesive* erroneous events in the beacon set are more severe, since this case is equivalent to searching for a *composition* of two or more different event types. In these circumstances the system will generally select events that match either the requested or the erroneous event set. While this behavior could be intentionally utilized to retrieve events from multiple types, in the general case it causes a measurable loss in precision.

11.5. Runtime performance

The runtime performance of the prototype implementation was evaluated from several aspects, in order to assess the feasibility of its deployment in production environments. On a mid-range personal computer with four physical and eight virtual processor cores, the throughput of the single-node system was macroscopically stable and typically well in excess of 20,000 events/second, with 30,000 events/second being attainable for smaller beacon event sets. The typical latency for a single beacon event set was in the range of $300-350\mu s$. The use of multiple worker threads provides a notable performance increase, as seen in Fig. 11. The increase is generally linear with the number of worker threads, although contention with the I/O threads and other processes on

Table 3
CTU-13: Single-node operation, Representative results.

Test	Positives	True positives	False positives	False negatives	Precision (%)	Recall (%)	Reduction (%)	Accuracy (%)	F_2
To-Background-Jist	337	337	0	0	100.000%	100.000%	99.998%	100.000%	1.000
From-Botnet-V42-UDP-Attempt-DNS	3951	3057	894	0	77.373%	100.000%	99.980%	99.996%	0.945
Background-google-analytics10	7254	4241	3013	22	58.464%	99.484%	99.964%	99.985%	0.872
Background-google-analytics8	7313	4122	3191	43	56.365%	98.968%	99.963%	99.984%	0.860
Background-google-analytics14	7069	3895	3174	109	55.100%	97.278%	99.965%	99.984%	0.844
From-Botnet-V50-8-TCP-Attempt-SPAM	6433	3435	2998	430	53.397%	88.875%	99.968%	99.983%	0.784
From-Botnet-V43-TCP-Established-HTTP-Ad-34	379	132	247	3	34.828%	97.778%	99.998%	99.999%	0.718
From-Botnet-V54-TCP-Attempt-SPAM	26,154	6581	19,573	36	25.162%	99.456%	99.869%	99.902%	0.625
From-Botnet-V50-6-TCP-WEB-Established	2300	437	1863	0	19.000%	100.000%	99.988%	99.991%	0.540
From-Botnet-V50-1-TCP-Attempt-SPAM	28,004	4729	23,275	222	16.887%	95.516%	99.860%	99.882%	0.495
To-Normal-V49-UDP-NTP-server	1178	178	1000	4	15.110%	97.802%	99.994%	99.995%	0.467
From-Botnet-V51-3-ICMP	72,021	10,000	62,021	363	13.885%	96.497%	99.639%	99.688%	0.441
From-Normal-V42-Grill	48,171	6687	41,484	967	13.882%	87.366%	99.759%	99.787%	0.424
From-Botnet-V42-TCP-CC16-HTTP-Not-Encrypted	1884	221	1663	0	11.730%	100.000%	99.991%	99.992%	0.399
From-Botnet-V43-TCP-Established-HTTP-Ad-41	3002	297	2705	0	9.893%	100.000%	99.985%	99.986%	0.354
From-Botnet-V50-2-TCP-CC16-HTTP-Not-Encrypted	854	81	773	55	9.485%	59.559%	99.996%	99.996%	0.290
From-Botnet-V43-TCP-CC16-HTTP-Not-Encrypted	1884	114	1770	0	6.051%	100.000%	99.991%	99.991%	0.244
From-Normal-V45-Grill	47,308	2474	44,834	0	5.230%	100.000%	99.763%	99.776%	0.216
From-Normal-V42-CVUT-WebServer	5309	241	5068	28	4.539%	89.591%	99.973%	99.974%	0.189
From-Normal-V54-Jist	29,728	948	28,780	0	3.189%	100.000%	99.851%	99.856%	0.141
From-Botnet-V51-2-UDP-Established	11,405	277	11,128	0	2.429%	100.000%	99.943%	99.944%	0.111
From-Botnet-V50-2-UDP-DNS	1,123,495	10,961	1,112,534	0	0.976%	100.000%	94.376%	94.431%	0.047
From-Botnet-V50-4-UDP-DNS	1,577,430	10,707	1,566,723	0	0.679%	100.000%	92.104%	92.157%	0.033
From-Botnet-V51-3-UDP-Attempt	58,245	181	58,064	15	0.311%	92.347%	99.708%	99.709%	0.015
From-Botnet-V50-5-TCP-WEB-Established-SSL	104,743	148	104,595	6	0.141%	96.104%	99.476%	99.476%	0.007

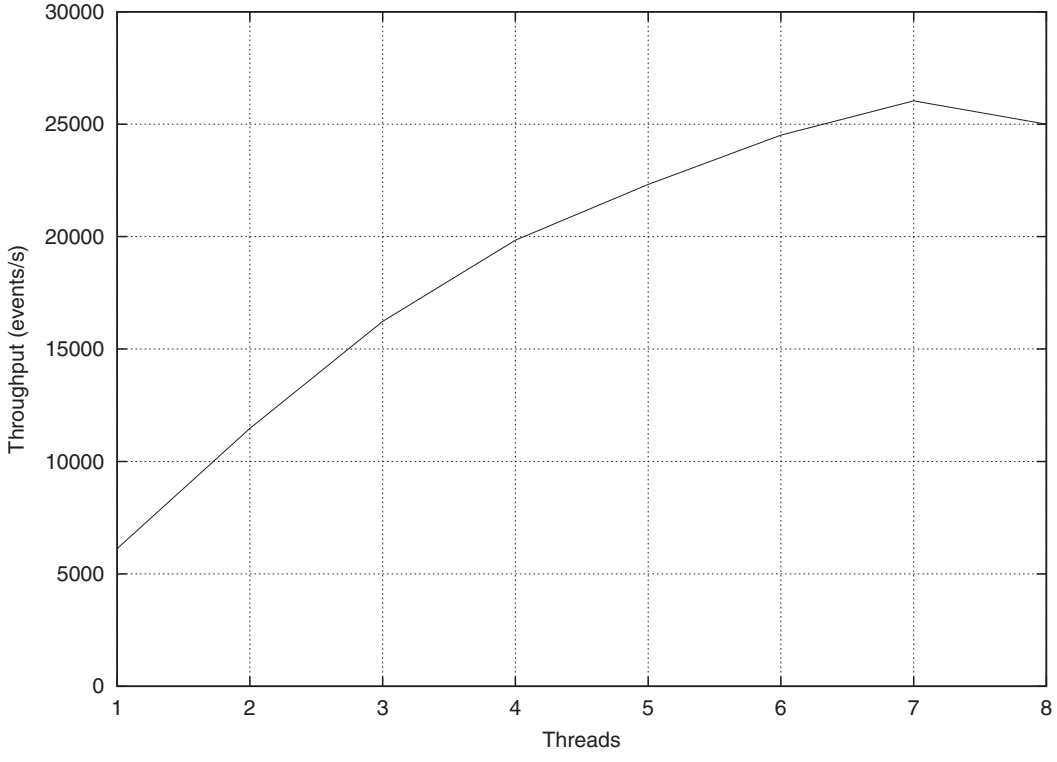

Fig. 11. Single-node event processing throughput for a single beacon event set and a variable number of worker threads.

the host computer results in a slight throughput decrease when all eight cores are involved in the event-processing itself. In addition, processing multiple beacon event sets simultaneously allows for a throughput increase to an aggregated total of 42,000 events/second for all sets on the same mid-range computer.

It is reasonable to expect that replacing code that has been optimized for parallel processing on a single node with a complex distributed computing framework like Storm would have an adverse effect of performance. The main cause lies in the communication costs associated with such a framework; objects that were originally moved around using in-process queues now have to be serialized, transmitted over a network socket and then deserialized. To estimate the overhead imposed by the use of the Storm framework, a one-node *cluster* was formed using the aforementioned personal computer system and a series of experiments was conducted. For a single beacon event set the throughput was about 4,400 events/second, which is about 4 times lower than the single-node version of the system. However, the ability to process

Table 4
CTU-13: Single-node operation, Result metrics.

Metric	Mean (%)	5th	10th	25th	Median (%)	75th	90th	95th
		percentile (%)				percentile (%)		
Precision	25.69	0.38	0.83	4.77	13.88	43.07	70.79	93.97
Recall	93.89	66.23	81.55	95.06	99.58	100	100	100
Reduction	99.23	94.65	98.75	99.75	99.94	99.99	99.99	99.99
Accuracy	99.34	94.78	98.83	99.77	99.97	99.99	99.99	99.99

Table 5
KDD'99: Single-node operation, Beacon set Noise Ratio (BNR) effect.

Test	BNR (%)	Positives	True positives	Precision (%)	Recall (%)	Reduction (%)	Accuracy (%)	F_2
back	10	2725	1980	72.661	98.901	99.782	99.939	0.922
back	30	6414	1986	30.964	99.201	99.486	99.644	0.688
back	50	3734	1503	40.252	75.075	99.701	99.782	0.640
imap	10	14,408	0	0.000	0.000	98.847	98.846	-
imap	30	204,874	0	0.000	0.000	83.610	83.609	-
imap	50	269,378	1	0.000	8.333	78.449	78.449	2.0E-5
ipsweep	10	7074	6653	94.049	87.782	99.434	99.892	0.889
ipsweep	30	8270	6654	80.459	87.795	99.338	99.797	0.862
ipsweep	50	50,236	6156	12.254	81.224	95.981	96.360	0.382
neptune	10	399,027	393,813	98.693	99.747	68.077	99.503	0.995
neptune	30	209,391	203,606	97.237	51.570	83.248	84.241	0.569
neptune	50	397,564	394,135	99.137	99.829	68.194	99.672	0.996
nmap	10	7751	1026	13.237	44.301	99.379	99.359	0.301
nmap	30	7873	1024	13.006	44.214	99.370	99.349	0.298
nmap	50	40,247	65	0.162	2.807	96.780	96.605	0.006
portsweep	10	27,341	2502	9.151	89.935	97.812	97.990	0.325
portsweep	30	12,738	2317	18.190	83.285	98.980	99.129	0.485
portsweep	50	42,654	2350	5.509	84.472	96.587	96.741	0.218
satan	10	44,592	3685	8.264	68.329	96.432	96.591	0.278
satan	30	149,228	4632	3.104	85.889	88.061	88.371	0.135
satan	50	269,537	0	0.000	0.000	78.437	78.006	-
teardrop	10	38,133	199	0.522	100.000	96.949	96.965	0.025
teardrop	30	42,793	198	0.463	99.497	96.576	96.592	0.022
teardrop	50	229,793	198	0.086	99.497	81.616	81.632	0.004
warezmaster	10	415	16	3.855	80.000	99.966	99.968	0.161
warezmaster	30	2811	19	0.676	95.000	99.775	99.777	0.032
warezmaster	50	2158	16	0.741	80.000	99.827	99.828	0.035

multiple beacon event sets simultaneously allows for the overhead to be amortized to a degree, allowing the testing system to reach a maximum of about 23,000 events, versus approximately 42,000 events/second for the single-node version. The introduction of additional inter-process communication queues, as well as the computational cost of serializing and deserializing objects, also has a significant impact on the achieved processing latency, which was in the range of $19 - 21ms$.

To evaluate the *scalability* of the proposed approach in a distributed environment, we performed a series of experiments on the Storm cluster described in Section 11.2. Fig. 12 illustrates the resulting performance for a variable number of nodes in the cluster. Due to the significantly lower computational power of the cluster nodes when compared to the previously described personal computer, a disparity in performance is expected. However, from a scalability point of view, it is very important to note the fact that the increase in event processing throughput is linear with respect to the number of nodes. The throughput of the system rose from 850 events/second for one node to 2,800 events/second for all five nodes participating, with 16,800 events/second being attainable on this cluster when processing multiple beacon event sets simultaneously. Therefore, the capability of the system may be increased well beyond what a single node could handle, simply with the addition of more computers to the cluster. On the other hand, due to the involvement of multiple networked computers and the increased message-passing delays encountered within a cluster, the processing latency was typically in the $30 - 35ms$ range.

Table 6
KDD'99: Distributed operation, Beacon set Noise Ratio (BNR) effect.

Test	BNR (%)	Positives	True positives	Precision (%)	Recall (%)	Reduction (%)	Accuracy (%)	F_2
back	10	2517	1980	78.665	98.901	99.798	99.955	0.940
back	30	5671	1823	32.146	91.059	99.546	99.678	0.666
back	50	8289	1503	18.132	75.075	99.336	99.417	0.461
imap	10	12,826	0	0.000	0.000	98.973	98.973	-
imap	30	194,684	0	0.000	0.000	84.425	84.424	-
imap	50	246,654	1	0.000	8.333	80.267	80.267	2.0E-5
ipsweep	10	7284	6653	91.337	87.782	99.417	99.875	0.884
ipsweep	30	8228	6654	80.870	87.795	99.341	99.800	0.863
ipsweep	50	49,752	6156	12.373	81.224	96.019	96.398	0.384
neptune	10	339,076	333,732	98.424	84.529	72.873	94.686	0.869
neptune	30	210,418	204,488	97.182	51.794	83.166	84.300	0.571
neptune	50	335,280	333,661	99.517	84.511	73.177	94.978	0.871
nmap	10	7751	1026	13.237	44.301	99.379	99.359	0.301
nmap	30	7890	1019	12.915	43.998	99.368	99.347	0.297
nmap	50	39,432	65	0.165	2.807	96.845	96.671	0.006
portsweep	10	26,127	1304	4.991	46.873	97.909	97.896	0.175
portsweep	30	13,068	2491	19.062	89.540	98.954	99.131	0.514
portsweep	50	50,069	2503	4.999	89.971	95.994	96.172	0.204
satan	10	44,316	3833	8.649	71.074	96.454	96.637	0.290
satan	30	156,521	4633	2.960	85.908	87.478	87.788	0.130
satan	50	306,403	1	0.000	0.019	75.487	75.056	1.5E-5
teardrop	10	38,209	199	0.521	100.000	96.943	96.959	0.025
teardrop	30	37,744	198	0.525	99.497	96.980	96.996	0.025
teardrop	50	232,214	198	0.085	99.497	81.422	81.439	0.004
warezmaster	10	440	16	3.636	80.000	99.964	99.966	0.153
warezmaster	30	1977	19	0.961	95.000	99.841	99.843	0.046
warezmaster	50	1483	16	1.079	80.000	99.881	99.882	0.051

Finally, the overall memory requirements of the proposed approach are quite reasonable, with the single-node system being able to process the KDD'99 data set for a single beacon event set with less than 96 MiB of heap memory available to the Java VM, while the distributed system was able to operate on 128 MiB. In both cases, however, the restricted resources led to a measurable performance impact due to the increased frequency of garbage collection by the Java VM, with the throughput of the distributed system dropping from 5,800 events/second to about 3,900 events/second.

12. Discussion

The proposed approach aims to be an efficient, schema-independent event analysis and filtering tool. As such, several assumptions were made which can potentially create situations where the system will misbehave.

The first assumption is that each attribute is considered to be a completely independent time series. Consequently, any redundant information among attributes will skew the similarity results. This is a common problem also encountered in machine learning systems, where any redundancy in the training data set may negatively affect the output. A typical solution involves the use of *feature selection* - otherwise mentioned as *feature reduction* - techniques to reduce the level of redundancy in the input by eliminating superfluous attributes. The selective removal of attributes improves the quality of the results, as well as the runtime performance of the system. It is orthogonal to the operation of the proposed system and can be implemented as a pre-processing filter on the input with no modifications to the existing system. Several methods for feature selection have been proposed including, but not limited to, algorithms based on information theory (Sebban and Nock (2002), Last et al. (2001), Fleuret (2004)), correlation

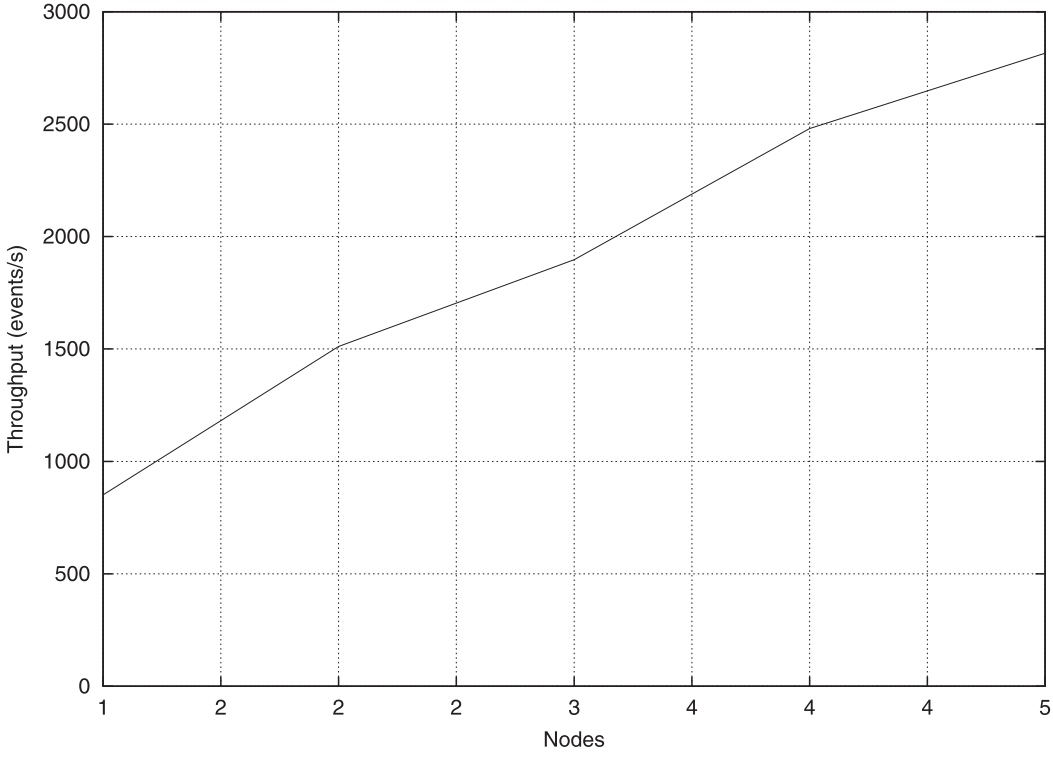

Fig. 12. Distributed event processing throughput for a single beacon event set and a variable number of nodes.

metrics (Yu and Liu, 2003) and machine learning (Zaman and Karray, 2009).

Another point of note is that each attribute is examined in isolation. Currently, the system does not examine the potential usefulness of *combinations* of attribute values as a similarity determinant. For example, it may be that a specific combination of feature values is important, as opposed to individual values, when computing a similarity value between two events. To consider such combinations two issues need to be resolved: (a) a vector similarity metric needs to be defined, although an equality check may be sufficient in most cases; (b) even for input streams with a moderate attribute set, the number of potential attribute combinations can be staggering, especially as the combination size increases, with a corresponding impact on performance. A potential solution could involve statistical analysis of the beacon event set and the use of feature selection methods to significantly reduce the number of combinations at an early stage of operation, thus making attribute combinations more tractable.

In the same vein, the system examines each *event* independently; no temporal relationships are determined between events or attribute values. In general the explicit detection of systematic or schema transitions is considered to be beyond the scope of this system. This simplifies the implementation and increases the potential for parallelism, but does not allow for temporally sensitive *sequences of events* to be effectively used as a beacon event set. In fact, if the beacon events themselves are dissimilar, the expected result is a decrease in the output precision and recall due to the introduction of additional noise. The fact that the proposed approach is essentially stateless, however, can be a significant advantage in certain environments where the monitored infrastructure is partially stateless as well, as is the case with REST-based service providers.

Another important point is that the information content determination algorithm does not take value *proximity* into account. In some cases it makes sense to conceptually *merge* certain attribute values which happen to be equivalent or in close proximity. For this to be possible, however, three prerequisites must be satisfied: (a) a proximity metric for the type of the attribute in question must be defined; (b) a suitable decision threshold should be determined and; (c) the concept of proximity must make sense within the current event domain for this specific attribute and specific merging rules may need to be defined. The prototype implementation system assumes that any such application-specific normalization happens in a separate pre-processing step and handles each discrete value independently.

In general, the proposed approach is best suited for high volume and frequency event streams that feature independent attributes with significant value variability, as a reasonable amount of total entropy must exist for specific features to be distinguishable. In addition, in order to retrieve the events of the requested event type with a reasonable degree of precision a *cohesive* beacon event set $\mathbb{B}$ is required, while the actual number of targeted events should exceed a minimum count that is generally in the same order of magnitude as $\mathbb{B}$ itself, so that the system will be better able to identify them as a group. While these are not *hard* requirements, the output quality of the system may degrade when they are not met.

With regard to the distributed-mode operation of the proposed approach, it should be noted that there is a significant amount of potential optimizations that could improve its performance and, possibly, the quality of its results. For example, improvements in the node allocation and scheduling engine could reduce the required bandwidth for intra-cluster communication. In addition, inferred metrics about the number of attributes and the distribution of their values could be used to improve the overall load balancing, while also reducing the degree of segmentation of each attribute time series to different processing nodes.

A final concern relates to the difficulty of monitoring an infrastructure that is slowly evolving. More specifically, gradual changes in the input stream are incorporated in the *sketch* that the system maintains for the value distribution of each attribute, without any distinction between slowly-evolving failures and benign oper-

ational trend changes. In general, the identification of such large-scale deviations requires domain-specific system-wide evaluation metrics, as well as expert knowledge for defining the desirable behavior and the determination of appropriate thresholds for the aforementioned metrics.

13. Conclusion

In order to perform on-line analysis of the massive amounts of events generated by modern information systems, it is necessary to devise techniques for the selective reduction of the volume of logged data that needs to be finally processed. Moreover, such systems tend to operate in environments where new components and resources are provisioned on-demand, aiming to meet a multitude of requirements, as business processes and operational needs change. As an effect, log data streams have to be added or removed dynamically, as new components and monitors are provisioned.

In this paper, we have described an event stream reduction method that allows for a small number of interesting *beacon* events that correspond to a specific system behavior to act as a pattern for the selection of other similar events, leading thus to a reduced data set. The proposed approach uses an information theoretic metric, that is based on the information content carried by attributes and attribute values in the event stream and the beacon event set $\mathbb{B}$, in order to compute a similarity measure between incoming events and the beacon set as a whole. The approach is schema- and domain- independent and it is able to cope with infinite data streams, without utilizing windowing, by employing object replacement algorithms to discard outdated information from its internal state. For dealing with the load of increasing event stream rates, the approach can be also applied in distributed mode, where the processing of events can be parallelized on computer *clusters*.

The end result of the filtering process, is a highly reduced collection of incoming events that are highly related to a specific system behavior. Results obtained using the KDD'99 and CTU-13 labeled data sets indicate that the approach can generally reduce the input stream to a subset of relevant events, while maintaining high recall and precision levels. Moreover, extensive testing in distributed mode, indicated a linear performance increase with the amount of available nodes, while maintaining moderate to low latency levels. This suggests that the system is capable of handling significantly higher input event rates, as long as more nodes are added to match the growth of the volume of the emitted events.

Possible future work includes the extension of the approach to handle combinations of attributes for computing the similarity between events, and the incorporation of a pre-processing phase to exclude highly correlated attributes from the event streams so that performance and quality can be further increased. Moreover, the ability of the prototype system to compute the similarity of an input event with multiple beacon event sets simultaneously allows for the development of a more expressive event query language involving aggregated results. Another area of interest is to consider optimizations in the distributed version of the prototype system that could potentially yield significant performance gains, as well as better result quality guarantees. Such optimizations can include the design of a node allocation and scheduling engine in order to enhance intra-cluster communication, location-aware scheduling for globally distributed operation, and the design of attribute and attribute value related metrics in order to reduce the segmentation of each attribute time series to different processing nodes. Finally, the incorporation of generic domain transform functions would allow the proposed method to be applicable to input streams that do not map directly to the time-series model. An example can be a beacon set consisting of exception stack traces logged on error conditions, which could then be used to reduce a stream of thread stack dumps sampled periodically over all instances of a service, thus helping to identify error-prone code paths.

This research did not receive any specific grant from funding agencies in the public, commercial, or not-for-profit sectors.

References

Agrawal, R., Gunopulos, D., Leymann, F., 1998. Mining process models from workflow logs. Adv. Database Technol. EDBT'98.
Anceaume, E., Busnel, Y., 2014. A distributed information divergence estimation over data streams. Parallel and Distrib. Syst. IEEE Trans. 25 (2), 478–487.
Brooks, R., 1983. Towards a theory of the comprehension of computer programs. Int. J. Man Mach. Stud. 18 (6), 543–554.
Buell, D.A., Kraft, D.H., 1981. Performance measurement in a fuzzy retrieval environment. In: ACM SIGIR Forum, 16, pp. 56–62.
Chandola, V., Banerjee, A., Kumar, V., 2012. Anomaly detection for discrete sequences: a survey. Knowl. Data Eng. IEEE Trans. 24 (5), 823–839.
Chou, T.S., Yen, K.K., Luo, J., 2008. Network intrusion detection design using feature selection of soft computing paradigms. Int. J. Comput. Intell. 4 (3), 196–208.
Cohen, P., Adams, N., Heeringa, B., 2007. Voting experts: an unsupervised algorithm for segmenting sequences. Intell. Data Anal. 11 (6), 607–625.
Cormode, G., Korn, F., Muthukrishnan, S., Srivastava, D., 2006. Space-and time-efficient deterministic algorithms for biased quantiles over data streams. In: Proceedings of the 25th ACM SIGMOD, pp. 263–272.
Cormode, G., Tirthapura, S., Xu, B., 2009. Time-decayed correlated aggregates over data streams. Stat. Anal. Data Min. 2 (5–6), 294–310.
Damerau, F.J., 1964. A technique for computer detection and correction of spelling errors. Comm. ACM 7 (3), 171–176.
Daneshgaran, F., Mondin, M., 1997. Information aging. In: Information Theory. Proceedings., IEEE International Symposium on, p. 38.
Donoho, D.L., 2006. Compressed sensing. Inf. Theor. IEEE Trans. 52 (4), 1289–1306.
Fleuret, F., 2004. Fast binary feature selection with conditional mutual information. J. Mach. Learn. Res. 5, 1531–1555.
Garcia, S., Grill, M., Stiborek, J., Zunino, A., 2014. An empirical comparison of botnet detection methods. Comput. Secur. 45, 100–123.
Goedertier, S., De Weerdt, J., Martens, D., Vanthienen, J., Baesens, B., 2011. Process discovery in event logs: an application in the telecom industry. Appl. Soft Comput. 11 (2), 1697–1710.
Hamming, R.W., 1950. Error detecting and error correcting codes. Bell Syst. Tech.J. 29 (2), 147–160.
Hirzel, M., 2012. Partition and compose: parallel complex event processing. In: In Proc. 6th International Conference on Distributed Event-Based Systems. ACM, pp. 191–200.
Jiang, M., Munawar, M., Reidemeister, T., Ward, P.A., et al., 2011. Efficient fault detection and diagnosis in complex software systems with information-theoretic monitoring. Dependable Secure Comput. IEEE Trans. 8 (4), 510–522.
Kalamatianos, T., Kontogiannis, K., 2014. Schema independent reduction of streaming log data. In: Advanced Information Systems Engineering. Springer, pp. 394–408.
Khoshgoftaar, T.M., Rebours, P., 2007. Improving software quality prediction by noise filtering techniques. J. Comput. Sci. Technol. 22 (3), 387–396.
Knowledge Discovery & Data Mining Cup Data, 1999, http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html. Accessed: 2016-05-29.
Krempl, G., Žliobaite, I., Brzeziński, D., Hüllermeier, E., Last, M., Lemaire, V., Noack, T., Shaker, A., Sievi, S., Spiliopoulou, M., et al., 2014. Open challenges for data stream mining research. ACM SIGKDD Explorations Newsl. 16 (1), 1–10.
Lane, T., Brodley, C.E., 1999. Temporal sequence learning and data reduction for anomaly detection. ACM Trans. Inf. Syst. Secur. (TISSEC) 2 (3), 295–331.
Last, M., Kandel, A., Maimon, O., 2001. Information-theoretic algorithm for feature selection. Pattern Recognit. Lett. 22, 799.
Lin, D., 1998. An information-theoretic definition of similarity. In: 15th international conference on Machine Learning, 1, p. 296.
Megiddo, N., Modha, D., 2003. ARC: a self-tuning, low overhead replacement cache. In: Proceedings of the 2nd USENIX Conference on File and Storage Technologies, pp. 115–130.
Mersch, J.G., Lang, R.R., 2015. An information-theoretic sentence similarity metric. In: The 28th International FLAIRS Conference.
Mihalcea, R., Wiebe, J., 2014. Simcompass: using deep learning word embeddings to assess cross-level similarity. SemEval '14 560.
Mutschler, C., Philippsen, M., 2013. Distributed low-latency out-of-order event processing for high data rate sensor streams. In: Parallel & Distributed Processing, 2013 IEEE 27th International Symposium on, pp. 1133–1144.
Papapetrou, O., Garofalakis, M., Deligiannakis, A., 2012. Sketch-based querying of distributed sliding-window data streams. Proc. VLDB Endowment 5 (10), 992–1003.
Pham, D.-S., Venkatesh, S., Lazarescu, M., Budhaditya, S., 2014. Anomaly detection in large-scale data stream networks. Data Min. Knowl. Discov. 28 (1), 145–189.
Pirró, G., Euzenat, J., 2010. A feature and information theoretic framework for semantic similarity and relatedness. In: The Semantic Web–ISWC 2010. Springer, pp. 615–630.
Qiu, D., Tamhane, A.C., 2007. A comparative study of the k-means algorithm and the normal mixture model for clustering: univariate case. .J Stat. Plan. Inference 137 (11), 3722–3740.

Rettig, L., Khayati, M., Cudré-Mauroux, P., Piorkowski, M., 2015. Online anomaly detection over big data streams. In: Big Data (Big Data), 2015 IEEE International Conference on. IEEE, pp. 1113–1122.
Sebban, M., Nock, R., 2002. A hybrid filter/wrapper approach of feature selection using information theory. Pattern Recognit. 35 (4), 835–846.
Seco, N., Veale, T., Hayes, J., 2004. An intrinsic information content metric for semantic similarity in WordNet. In: ECAI, 16, p. 1089.
Sellers, P.H., 1980. The theory and computation of evolutionary distances: pattern recognition. J. Algorithms 1 (4), 359.
Tan, M., Tsang, I.W., Wang, L., 2014. Towards ultrahigh dimensional feature selection for big data. J. Mach. Learn. Res. 15 (1), 1371–1429.
Tavallaee, M., Bagheri, E., Lu, W., Ghorbani, A., 2009. A detailed analysis of the kdd cup 99 data set. 2009 IEEE Symposium on Computational Intelligence for Security and Defense Applications.
Tran, D.-H., Gaber, M.M., Sattler, K.-U., 2014. Change detection in streaming data in the era of big data: models and issues. ACM SIGKDD Explorations Newsl. 16 (1), 30–38.
Vaarandi, R., Podiņš, K., 2010. Network IDS alert classification with frequent itemset mining and data clustering. In: Network and Service Management, International Conference on. IEEE, pp. 451–456.
Wagner, R.A., Fischer, M.J., 1974. The string-to-string correction problem. J. ACM (JACM) 21 (1), 168–173.
Wang, J., Zhao, P., Hoi, S.C., Jin, R., 2014. Online feature selection and its applications. Knowl. Data Eng. IEEE Trans. 26 (3), 698–710.
Yu, L., Liu, H., 2003. Feature selection for high-dimensional data: a fast correlation-based filter solution. In: Machine Learning, 20, p. 856.
Zaman, S., Karray, F., 2009. Features selection for intrusion detection systems based on support vector machines. In: 6th Consumer Communications and Networking Conference. IEEE, pp. 1–8.
Zargar, G., Kabiri, P., 2009. Identification of effective network features for probing attack detection. In: Networked Digital Technologies, First International Conference on, pp. 392–397.
Zawawy, H., Kontogiannis, K., Mylopoulos, J., 2010. Log filtering and interpretation for root cause analysis. In: Software Maintenance, IEEE International Conference on, pp. 1–5.
Zhang, Q., Wang, W., 2007. A fast algorithm for approximate quantiles in high speed data streams. In: 19th International Conference on Scientific and Statistical Database Management. IEEE, p. 29.
Zheng, Z., Lan, Z., Park, B.H., Geist, A., 2009. System log pre-processing to improve failure prediction. In: Dependable Systems & Networks, 2009. IEEE/IFIP International Conference on, pp. 572–577.

Theodoros Kalamatianos graduated from the School of Electrical and Computer Engineering at the National Technical University of Athens, Greece, where he is currently a Ph.D candidate. He has significant professional experience in the fields of software engineering and large-scale network design and operations. His research interests include dynamic software analysis, tooling for software comprehension and re-engineering, monitoring of global-scale IT infrastructures, high-volume complex event processing, very large data bases (VLDB) and large-scale high-performance parallel and distributed computing.

Kostas Kontogiannis holds a B.Sc. in Mathematics from the University of Patras, Greece, a M.Sc. in Computer Science from the Katholieke Universiteit Leuven, Belgium and a Ph.D. in Computer Science from McGill University, Canada. He is a Professor at the Dept. of Computer Science at Western University, where he holds a Western Research Chair in Software Engineering for Cyber-Physical Systems. Kostas is a Faculty Fellow at the IBM CAS Research and his current research interests focus on the design and development of tools for continuous software engineering, software evolution, adaptive systems, and services computing.