Online Anomaly Detection over Big Data Streams

Laura Rettig*†, Mourad Khayati†, Philippe Cudré-Mauroux† and Michał Piórkowski*

*Big Data and Business Intelligence Competence Center at Swisscom, Bern—Switzerland
{firstname.lastname}@swisscom.com

†eXascale Infolab, University of Fribourg—Switzerland
{firstname.lastname}@unifr.ch

***Abstract*—Data quality is a challenging problem in many real world application domains. While a lot of attention has been given to detect anomalies for data at rest, detecting anomalies for streaming applications still largely remains an open problem. For applications involving several data streams, the challenge of detecting anomalies has become harder over time, as data can dynamically evolve in subtle ways following changes in the underlying infrastructure. In this paper, we describe and empirically evaluate an online anomaly detection pipeline that satisfies two key conditions: generality and scalability. Our technique works on numerical data as well as on categorical data and makes no assumption on the underlying data distributions. We implement two metrics, relative entropy and Pearson correlation, to dynamically detect anomalies. The two metrics we use provide an efficient and effective detection of anomalies over high velocity streams of events.**

In the following, we describe the design and implementation of our approach in a Big Data scenario using state-of-the-art streaming components. Specifically, we build on Kafka queues and Spark Streaming for realizing our approach while satisfying the generality and scalability requirements given above. We show how a combination of the two metrics we put forward can be applied to detect several types of anomalies—like infrastructure failures, hardware misconfiguration or user-driven anomalies—in large-scale telecommunication networks. We also discuss the merits and limitations of the resulting architecture and empirically evaluate its scalability on a real deployment over live streams capturing events from millions of mobile devices.

I. Introduction

Data quality is a challenging problem in many domains such as medicine, environmental monitoring, or IT infrastructures. Assessing the quality of the data requires the deployment of a number of fundamental data services including anomaly detection. Data anomalies can manifest themselves in many different ways—for instance via missing values or outliers—and can be caused by erroneous procedures, system failures or unexpected events. A number of methods have already been proposed in the literature to detect and classify anomalies (cf. Section II). The applicability of those methods heavily depends on data dynamics. Specifically, detecting anomalies is relatively common for data *at rest*. For applications where data is *on the move*, for example for applications involving several data streams, the challenge of detecting anomalies has become harder over time, as data can dynamically evolve in subtle ways following changes in the underlying infrastructure. In such cases, being able to detect anomalies in real-time becomes a crucial feature.

For telecommunication companies, it has become essential to deploy dedicated quality assurance systems in order to guarantee that the telecommunication services are provided with the highest possible quality. The complexity of such monitoring systems grows together with the complexity of the communication services being monitored and with the demand for providing nontrivial monitoring insight in real-time. In this work, we focus on detecting anomalies on the signaling traffic of a mobile cellular network, where any mobile terminal attached to the cellular network produces signaling messages, which are subsequently captured by the network infrastructure for the sake of quality assurance. The characteristics of the signaling traffic we consider falls into the class of Big Data streams. Specifically, (i) the cumulative daily volume we consider is in the order of TBs, (ii) the signaling data is multidimensional, while (iii) the velocity of the data, measured in number of events per time unit, is in the order of hundreds of millions per second.

The very nature of modern cellular network infrastructures, supporting a number of elaborate telecommunication protocols, implies a high complexity for the signaling traffic. For instance, the infrastructure of many modern mobile cellular networks consists of a radio access network (RAN) and a core network (CN). The latter is split into circuit switched (CS - voice calls) and packet switched (PS - data traffic) domains. Mobile devices can attach to CS or PS, or both at the same time. The radio communication takes place between a mobile device and a base station within RAN, serving one or more radio cells, which then carries the voice and data traffic via fixed networks to/from CN. Radio cells are the smallest spatial entities in the cellular network. Depending on the radio bearer, they can be classified as 2G (GSM/EDGE), 3G (UMTS/HSPA) or 4G (LTE). For the purpose of quality assurance, a passive monitoring system collects signaling events from the links between the RAN and CN parts of the network, covering all 2G, 3G and 4G—specifically on the A, Gb, IuPS, IuCs and S1-MME interfaces. Swisscom built a Big Data streaming infrastructure providing such data as real-time streams that can be dynamically fed through dedicated message buses. We call this new infrastructure the *Firehose*. It provides inter alia with multiplexing, parsing, serialization and encryption to ensure that the signaling data is properly preprocessed (e.g., anonymized) and ready for further processing.

The state-of-the-art time-series database system we used so far for quality assurance is InfluxDB. The latter provides only a relatively basic set of aggregate functions over unidimensional time series. Although handful, this set of functions is clearly insufficient to detect a plethora of anomalies in a dynamic and robust manner.

In the following, we introduce the new anomaly detection method that we have developed for our needs. Our solution satisfies two key conditions [1]: generality and scalability. The method we propose is applicable to both multidimensional as well as categorical data. In addition, our solution does not take any assumptions about the data distribution. Moreover, the method is Big Data *friendly* as it can cope with large volumes of data *at rest* as well as *on the move* through streaming interfaces. In summary, the main contributions of this paper are:

- a new system implemented over Apache Spark [2] combining two well-known metrics, relative entropy and Pearson correlation, to detect anomalies over both high-velocity streams and/or large volumes of data at rest;
- an empirical evaluation of our system showing the effectiveness of the two metrics we leverage for detecting anomalies. The results of our experiments show that the entropy metric is well-suited for detecting gradual changes in data streams, while the correlation metric is more appropriate for detecting abrupt changes caused for example by hardware failures;
- an empirical evaluation of our system demonstrating its graceful scalability when both the number of nodes and the amount of inspected data increase.

II. Related Work

A number of techniques have been proposed to detect anomalies in multidimensional data streams or for multidimensional time-series data.

Zhang et al. [3] propose a solution that detects outliers in multidimensional data. This approach performs anomaly detection by measuring the distance of a data point in various subspaces. The authors show that for multi-dimensional data, changes may be observable on one dimension, over a subset of dimensions, or overall. We leverage this property to detect different types of anomalies in the following. However, the proposed technique based on indexing and subspace pruning is not applicable to real-time scenarios due to the high number of iterations over the data.

Dasu et al. [1] present an approach to detect sudden changes in multidimensional data streams. In their approach, multidimensional stream instances are represented as *kdq-trees* (a combination of *kd-trees* and *quadtrees*), while relative entropy is used as a similarity measure. To detect changes on unknown distributions, the method resamples the data using a bootstrap technique in order to obtain the expected distribution of the data. The relative entropy between the distributions gives a bound for the relative entropy (under the assumption that the samples originate from the same distribution), allowing for a statistically sound detection of significant changes. The authors introduce two different window comparison models. The first model compares adjacent windows, which is well suited for detecting abrupt changes. The second model compares a sliding window to a previous window, which is convenient to detect more gradual changes. We use similar techniques to measure changes between successive time windows over multi-dimensional data streams. However, we do not rely on complex and multidimensional data structures that are difficult to distribute and efficiently update on clusters of machines.

Li and Han [4] also address the problem of detecting anomalies in subspaces of multidimensional data by introducing the *time-series data cube* as a new data structure capable of handling the multidimensional space. Using this data structure, they are able to identify the subspaces that are most likely to be anomalous. To this end, they measure the entropy for each attribute and consider attributes with low entropy, i.e., attributes with mostly homogeneous values, as suitable for subspaces in which anomalies are easily detectable. By selecting likely anomalous subspaces, they elude the curse of dimensionality and avoid having to search for anomalies in every possible subspace. The authors apply their technique to detect anomalies in synthetic data with anomalous time series, defining four different kinds of anomalies: trend, magnitude, phase, and miscellaneous. However, their solution works only for larger fluctuations and is not suitable in case of more subtle differences that emerge from most real-world applications.

Young et al. [5] detect and classify emergency and non-emergency events using annotated network data. Similarly to our work, they compare normal and anomalous days to detect deviations from a baseline representing some average behavior. They use autoregressive hidden Markov models to detect the precise onset of an event. Unlike our solution, they use a metric that considers only the closest cell tower to the known event. Furthermore, the applied matrix factorization is computed on data at rest and not in real-time, unlike the high-velocity streams we consider in the following.

Anomaly detection techniques have also been proposed for strictly temporal data. Gupta et al. [6] present an overview of anomaly detection on various kinds of temporal data. They define anomalies as outliers and present detection methods for both the discrete and the continuous cases. While the overview presents a wide array of different techniques, it also mentions that there is a great deal of different problems that can be addressed by detecting outliers on time series data and that solutions need to be adapted to meet the needs of specific problems. For stream data, the use of models that update and decay over time is suggested.

Wu and Shao [7] apply an autoregressive process to detect sudden changes between adjacent windows of network traffic data. Their use of moving windows allows real-time anomaly detection. However, their model is limited to detecting major and sudden changes, such as in denial-of-service attacks to a network and is not useful for detecting finer variations.

Statistical metrics (e.g., probabilistic data structures) that represent sliding windows in streams have also been proposed for detecting anomalies. Datar et al. [8] introduce approximate stream summary statistics for sliding windows. Since regularities in streams may evolve over time, the issue of data decay is handled by giving more weight to recent objects and aggregating, then eventually discarding older data. The authors store information using *exponential histograms*. This data structure uses timestamps as the bins and the count of an item in the stream as the value for each temporal range. While their work is suitable for computing approximate statistics with bounded errors to summarize aspects of the content of a stream, they do not address the issue of detecting changes.

Papapetrou et al. [9] introduce the *ECM-sketch*, a technique that is suitable for summarizing streams and for answering complex queries over data streams. Their *ECM-sketch* combines the capabilities of stream summary through key-based counting in a large key space (similar to the count-min sketch [10]) but extends them with exponential histograms for synopses of sliding windows. Unlike our work, *ECM-sketch* cannot use deterministic data structures, e.g., hashmaps, due to the potentially very large number of distinct items (keys) for which a counter (values) has to be maintained.

Cormode and Muthukrishnan [11] define *deltoids* as items where the computed metrics indicate that significant change took place in monitoring network traffic data. These deltoids are probabilistic metrics designed to use little space, to require short update times, and to produce accurate results based on fixed thresholds. The authors distinguish between different variations in data streams—absolute, relative, and variational—and maintain deltoids for each type. Their approach requires a preconfiguration of the parameters that are used to detect deltoids leveraging training data. Our proposed solution, on the other hand, aims at being more general-purpose and does not require any preprocessing or training.

III. Background

We describe in this section the two measures, i.e., Relative Entropy and Pearson Correlation, and the Big Data framework (Spark) we leverage in our system.

A. Relative Entropy

The relative entropy, or Kullback-Leibler Divergence, $D(P||Q)$ [12] is a non-symmetric measure of information loss. It is defined on two probability distributions P and Q as follows:

$$D(P\|Q) = \sum_{i \in A} P(i) \log \frac{P(i)}{Q(i)} \tag{1}$$

where $P(i)$ and $Q(i)$ are the probability of item i in the respective probability distribution, given by

$$P(i) = \frac{m_i}{\sum_{a \in A} m_a} \tag{2}$$

where A is the set of all possible items i in the probability distributions and m_i and m_a are the numbers of items i and a in the current distribution P.

Relative entropy is used to measure the difference between two probability distributions P and Q. This is a generic method that applies to multidimensional data without requiring any prior domain knowledge about the underlying distributions P and Q. In our context, $D(P\|Q)$ is used to measure changes between successive time windows over multi-dimensional data streams, as introduced in [1].

D then represents how different two probability distributions are. The values of $P(i)$ and $Q(i)$ are defined over $[0, 1]$. A low relative entropy indicates regularity in the sense that the two distributions are similar. Anomalies are detected in cases where the value of D increases.

B. Pearson Correlation

The Pearson correlation coefficient is a statistical value measuring the linear dependence between two variables X and Y. It is defined as

$$r(X,Y) = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}} \tag{3}$$

where $\bar{x}$ and $\bar{y}$ stand for the mean of X and Y respectively.

The coefficient $r(X,Y)$ ranges between 1 and -1. Positive values from $(0, 1]$ indicate positive correlation between X and Y, while negative values from $[-1, 0)$ indicate negative correlation. When the Pearson correlation coefficient is 0, there is no correlation between X and Y.

C. Spark

Apache Spark is a general-purpose engine for large-scale data processing. It offers several advantages over MapReduce, such as faster in-memory execution, especially for cases where multiple passes are made over the same data (such as when multiple stages of transforming, mapping and reducing are applied over the data). It also offers a higher-level Scala API, greatly facilitating the expression of complex processing pipelines.

Spark's main abstraction are resilient distributed datasets (RDDs) [13]. In batch processing mode, RDDs are created by loading data, for example from HDFS, or by transforming other RDDs. RDDs are immutable abstractions of distributed data that can be organized in a Directed Acyclic Graph (DAG) to represent transformations on top of the data. Since RDDs are evaluated lazily, the transformations are only applied when materialization becomes necessary. These transformations are then applied in a manner that minimizes data shuffling between the executors.

Spark includes a streaming library called Spark Streaming. Spark Streaming is based on micro-batch computations and introduces another core abstraction, discretized streams (DStreams) [14]. DStreams are continuous sequences of RDDs, with one RDD containing all the data belonging to one micro-batch. Many of the functions available for RDDs are also available for DStreams, abstracting away the individual processing of RDDs during streaming, such that transformations can be directly applied on DStreams. The underlying execution engine, the Spark engine, is the same for both streaming and batch modes. The execution engine obtains one RDD from the DStream per micro-batch time interval and applies the transformations directly to the RDD. Since both RDDs and DStreams are immutable, the output of applying a transformation to a DStream is a new DStream representing a continuous sequence of transformed RDDs. As part of the code can be reused between the batch and streaming modes, Apache Spark is well-suited for cases where both batch and streaming data should be jointly processed.

The MLlib library from Spark provides an efficient implementation for computing the Pearson correlation between two vectors. However, this solution assumes a specific format of the input data. The relative entropy is not available under Spark.

IV. System Overview

An overview of the system we developed for anomaly detection is shown in Figure 1. The figure depicts the entire pipeline starting from cell towers on the left-hand side to the anomaly detection results on the right-hand side. Signaling traffic is received from network probes, staged in real-time, and then written to a dedicated queue for real-time consumption as well as to HDFS for longer-term storage and processing. The anomaly detection component consumes the data—either in real-time (queue) or in batch (HDFS) mode—, processes it, and outputs both metrics and alerts. The various components of the architecture are described in more detail in the following.

A. Data Collection

Signaling traffic carrying various network events is captured by probes connected to links between RAN and CN. Those captured events contain non-personal monitoring information from the mobile terminals. They do not carry any

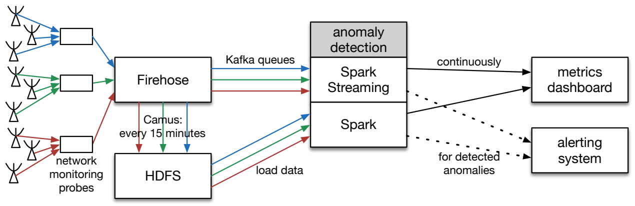


Figure 1. Integration of the anomaly detection component into the Big Data receiving and processing pipeline.

information about the content transmitted over the mobile network. There exist different types of probes and different types of data for the various network components. Incoming data is processed separately depending on the probe type. Specifically, for each generation and each network type (e.g. 2G and A or 3G and IuPS), separate probes tap into the network, producing separate data streams. Low-level, binary events are first fed to the Firehose, which parses and then serializes the events using the Avro [15] format. The events are then written into a series of Kafka queues [16], one per probe type. Periodically every 15 minutes, events are pipelined from the queue into HDFS.Both the queues and the storage consider the same data format and can be used as input to Spark Streaming and standard Spark, respectively. As each event is timestamped, it can be treated in the same way as real-time data and can also be used to simulate streams. In that sense, our system emulates the so-called lambda architecture [17] with analogous pipelines for batch and real-time processing on the same data source.

B. Stream Processing

1) Streaming System: Our system for anomaly detection on network monitoring events is built using Apache Spark on top of YARN.

The duration of the micro-batches in Spark Streaming is chosen experimentally, as it depends on the volume of data and as we consider several interfaces (e.g., 2G and 3G). Longer micro-batches require more storage, since more data needs to be cached in-memory. This follows from the fact that all data from a micro-batch are computed jointly and hence the processing can only start once all data for one micro-batch have been received. On the other hand, shorter micro-batches require faster online algorithms. Since micro-batches are computed in order, the time required to process one micro-batch cannot be longer than the duration of the micro-batch. If it is not the case, micro-batches rapidly accumulate and lead to data losses.

Streaming receivers, which represent the input interface of the Spark Streaming component, connect as consumers to Kafka located in Firehose (see Figure 1). In order to load-balance the system, multiple receivers can consume events in parallel. For our detection system, nine parallel receivers are used and their data streams are then combined for joint-processing. The consumers are configured to receive every event only once, beginning at the time when the application is first started. Output operations have to be defined prior to starting the stream. Typically, such outputs consist of transforming the DStream and then applying operations to each RDD such as returning a value to a network connection or a file for visualization purposes. In our anomaly detection system, metrics are written continuously, whereas alerts are triggered upon detection of an anomalous event.

In order to perform anomaly detection, two metrics are continuously maintained over the streams: relative entropy on individual data streams and Pearson correlation across multiple streams. These metrics form a constant baseline over non-anomalous data points, from which anomalous instances can be detected.

2) Relative Entropy Pipeline: Relative entropy is computed separately on each interface by comparing the empirical distributions of event types at their respective topological level. The data pipeline for computing relative entropy is shown in Figure 2. Each batch in the incoming DStream is mapped into a new DStream of *((location, type), 1)* key-value pairs, where the identifier for the location and the type of the event form a composite key. We consider three topology levels (i) *cellsite*, where the location key is a 4-character identifier, (ii) *LAC* (location area code), which is an aggregate of multiple co-located cells for larger regions within the country with a unique ID and (iii) *globally* for the whole country, discarding the location key. While looking at lower levels in the topology facilitates the detection of local anomalies, a more global model is faster to compute due to the smaller key space. By summing up the values per key in a *reduce* operation, the number of events per location and event type get counted. By grouping per cell, we obtain a new RDD containing the event histograms, i.e., the counts per event type and per cell.

These histograms are interpreted as aggregates of anonymized user-triggered actions, since they capture various aspects of human activity (making a phone call, moving across the network, etc.). The probability $P(i)$ of each event type i in the current distribution P, as in Equation (1), is given by dividing the count for this event, m_i, by the sum of the counts m_a in the current histogram for all possible event types in A. Hence, $P(i)$ represents the relative frequency of event type i in the current distribution. Finally, for each location indicator, the relative entropy $D(P_t \| Q_{t-\Delta t})$ at time t is computed by summing up the comparison of each possible event type i. In streaming mode, the probability distribution from the previous RDD is stored for comparing adjacent windows, yielding a distance measure between the two time periods per location. We do not only compare adjacent windows but also compare the histograms of all hours, both daily and weekly. Hence, we can detect both abrupt changes and gradual changes over time. While adjacent window computation with Δt set to one hour is performed on both streaming and batch data, the current solution for comparing with larger values of Δt, such as one week, is done purely on data that is loaded from HDFS. As noted above, Spark's programming model facilitates code reuse between both real-time and batch processing and guarantees comparable results.

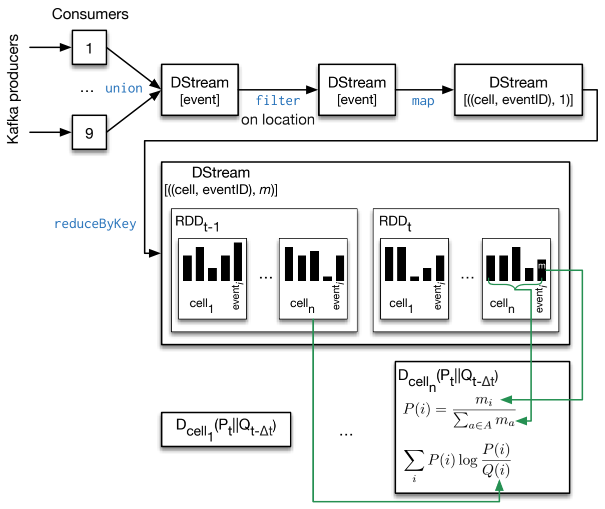

Consumers
Kafka producers
1
9
...
union
DStream [event]
filter on location
DStream [event]
map
DStream [((cell, eventID), 1)]
DStream [((cell, eventID), m)]
reduceByKey
RDD_{t-1}
RDD_t
cell_1
cell_n
event_i
$D_{cell_n}(P_t \| Q_{t-\Delta t})$
$P(i) = \frac{m_i}{\sum_{a \in A} m_a}$
$\sum_i P(i) \log \frac{P(i)}{Q(i)}$
$D_{cell_1}(P_t \| Q_{t-\Delta t})$

Figure 2. Relative entropy computation pipeline.

Example 1: We consider streams of messages of the form $[(t_1, c_1, Eid_1), \ldots, (t_n, c_n, Eid_n)]$, with c_i is coming from the set of cells $\{B, C\}$ and Eid_i is coming from the set of possible event type IDs $A = \{1, 2\}$. A micro-batch $[(t_0, C, 1), (t_1, C, 2), (t_2, C, 1), (t_3, B, 2), (t_4, C, 1), (t_5, C, 1)]$ is obtained at a time t with timestamps t_i ranging between $t - \Delta t$, the previous micro-batch computation time, and t. The partition of the stream at time t is mapped into $[((c_i, Eid_k), 1), ((c_j, Eid_l), 1), \ldots]$. We apply a reduce operation on the composite key consisting of the cell and the event type to transform this stream into tuples containing the key and the corresponding count of events in the current stream as follows: $[((C, 1), 4), ((C, 2), 1), ((B, 2), 1)]$. Since we compute the relative entropy for each cell individually, we illustrate the computation for cell C only (similar computations are applied to all the other cells). At time t, the histogram's counts of cell C in our example are respectively 4 for event type 1 and 1 for event type 2. Using Equation (2) the probabilities in P_t are respectively $P(1) = \frac{4}{5}$ and $P(2) = \frac{1}{5}$. We compare the distribution P_t to the probability distribution from a previous micro-batch $Q_{t-\Delta t}$ from cell C with $Q(1) = \frac{2}{3}$ and $Q(2) = \frac{1}{3}$. By applying Equation (1), we get for our example $D(P \| Q) = \frac{4}{5} \log \frac{4/5}{2/3} + \frac{1}{5} \log \frac{1/5}{1/3} = 0.044$.

3) Pearson Correlation Pipeline: In order to compute the Pearson correlation coefficient $r(S_X, S_Y)$ of two streams S_X and S_Y, we consider sets of receivers consuming events from at least two separate interfaces. Both streams are treated separately, mapping each stream onto a DStream containing anonymized user IDs and then counting the number of distinct IDs per micro-batch, such that we obtain one count per RDD. A graphical representation of the data transformation process for the Pearson correlation pipeline is shown in Figure 3. By windowing over the counts of micro-batches over longer durations, we obtain RDDs containing multiple counts—essentially the vectors X and Y introduced in Equation (3). At this point, two previously separate DStreams are combined as a DStream of pairs of RDDs, one from each DStream, with corresponding timestamps. Using the pairs of RDDs, containing the unique input counts $x_1, \ldots, x_n$ and $y_1, \ldots, y_n$ respectively, a correlation coefficient for the particular time period considered is computed according to Equation (3).

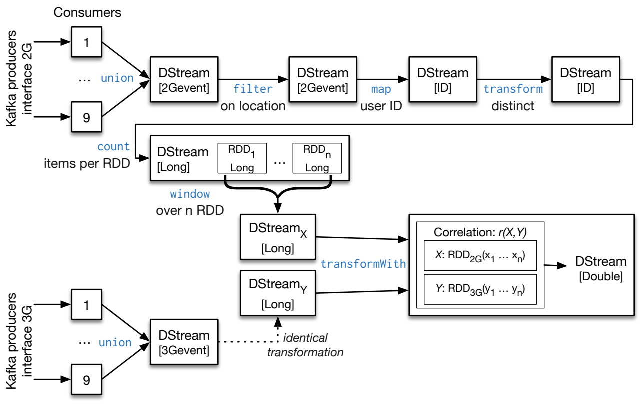


Figure 3. Pearson correlation computation pipeline.

Example 2: Given two distinct streams of messages mapped onto a stream of anonymized user identifiers Uid_i of the following form $[Uid_1, \ldots, Uid_n]$, we collect the user IDs during a short period of time (e.g., 10 seconds), then count the number of distinct IDs during this period of time and write the counts to a new stream. Let us assume stream S_X contains $[A, B, B, A, B]$ and stream S_Y contains $[C, B, C, C, D, C, D, E, F, A]$ in this short window. By applying a *distinct* operation on each stream (for S_X, yielding $[A, B]$) and then retrieving the length of this window, we obtain the count of distinct users per timeframe. These two counts, respectively 2 and 6, are then written to the respective output (result) streams RS_X and RS_Y. After 40 seconds, the previous output streams contain multiple counts: RS_X, $[2, 1, 1, 3]$ and RS_Y, $[6, 5, 4, 6]$. By windowing over these counts (e.g., every 40 seconds) over both streams, we obtain two vectors X and Y from the same timespan, each containing, in this case, 4 counts. Grouping these vectors into pairs of (x_i, y_i) gives $[(2, 6), (1, 5), (1, 4), (3, 7)]$. Plugging the values into Equation (3) yields a Pearson correlation score of $\frac{9}{\sqrt{55}} = 0.94$. Consider the case where the network monitoring probe producing events on S_Y fails, such that we no longer receive events from one area. Then, by reducing the events on S_Y and the count of distinct users on RS_Y after a time range, e.g., 20 seconds, an increase in x_i meets a decrease in y_i. Thus, the stream of grouped pairs is as follows: $[(2, 6), (1, 5), (1, 2), (3, 5)]$ so that the correlation $r(X, Y)$ for this pair of vectors decreases to $\frac{5}{3\sqrt{11}} = 0.5$.

V. Empirical Evaluation

To evaluate the efficiency and the effectiveness of our anomaly detection pipeline, we conducted experiments over real-world Big Data streams regrouping events from millions of mobile devices running on the Swisscom network. The data we focus on for our experiments is captured at the A and the IuCS interfaces by the probes monitoring 2G voice and 3G voice links, respectively, which report network events on Swisscom's telecommunication network.

In the following, we evaluate the accuracy of our anomaly detection for two different types of real-world events. The first events we focus on are events where anomalies originate from human behavior and that are caused by sports events, popular concerts, exhibitions or traffic disruptions, etc. The second events we focus on result from failing IT infrastructure components, for which we simulate one data loss scenario. In order to evaluate our metrics, we suppose that hardware failures are reported to the responsible monitoring teams, while large-scale human events are publicly known.

A. Anomaly Detection Accuracy

We start by evaluating the accuracy of our anomaly detection system below.

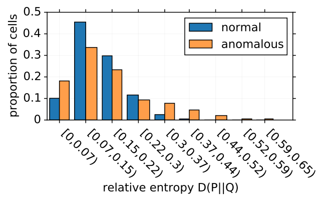
1) Relative Entropy Accuracy: As explained in Section IV-B1, the event histograms can be interpreted as an aggregate of mobile phone users' activity within the network. In the experiment of Figure 4, each histogram is constructed over the counts of events per event type. Under non-anomalous circumstances, human behavior is mostly regular, i.e., there is no major change in the relative proportion of the counts for each event type in different histograms for a sufficiently large population of mobile devices. However, anomalous events, such as sudden differences in movement patterns, lead to a change in the distribution of the events. For example, the proportion of events of the type "phone call" may increase in a situation where vehicular traffic is blocked and the proportion of events of the type "moved to another cell" may decrease. By measuring the difference between usual and current histograms, it is possible to detect change in the habitual patterns using the relative entropy.

As a real-world example of an anomaly relating to a human event, we consider the flood event that took place in Geneva, Switzerland, on May 2, 2015. This event caused a significant change in the movement patterns of our users as several bridges had to be closed and as users had to pick new routes to get to their usual destinations. The change implies a higher relative entropy, both when comparing adjacent and fixed windows. When considering the distribution of the relative entropy, the anomaly could potentially be detected through the unusual proportion of higher relative entropy values in the anomalous scenario (on May 2) compared to the baseline scenario (on all other days with no known major movement disruption).

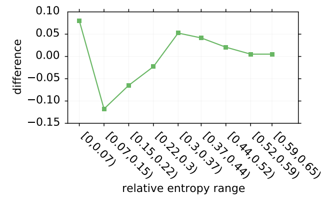
In the experiment of Figure 4(a), the relative entropy is computed between histograms of adjacent windows considering one hour per cell. We filtered out the incoming streams to events originating from cells within the city of Geneva. Figure 4(a) shows the distribution of the mean relative entropies per cell for one day. For known "normal" days, the mean entropy $\bar{D}$ over all cells is computed and the ranges for the bins on the x-axis are taken as multiples of $\bar{D}$, where $\bar{D} \approx 0.15$. The y-axis shows the relative proportion of cells with mean relative entropy falling into the range given on the x-axis. For comparison, we are differentiating between the data for the known anomalous day and a baseline average obtained from computing the relative entropy on multiple days in April and May. The results show that the majority of cells display low relative entropies for both normal and anomalous days, although normal days' entropy values are closer to $\bar{D}$. On the other hand, Figure 4(b) shows that the proportion of cells where the mean daily relative entropy exceeds $2\bar{D}$, i.e., bins with ranges greater than 0.3, is clearly higher on May 2 than on average on known normal days. The high proportion of very low relative entropy values for the anomalous day in Figure 4(b) is linked to inactivity at some cells. This example, which was confirmed in several similar cases, indicates that high relative entropy values can be used to detect human anomalies such as the one that occurred in Geneva on May 2.

In order to detect anomalies automatically, we leverage domain knowledge about the shift of the distribution to set thresholds. We consider a multiple of the baseline mean $k\bar{D}$ as a threshold and count the number of cells where the threshold is exceeded. In our deployment, setting k to 2 gives the best experimental results. By comparing known normal and known anomalous days, thresholds for anomaly alerts can be set for the number of cells with a relative entropy above a certain threshold within a geographical area. For example, the city of Geneva exceeds the number of cells with relative entropy above a threshold on a normal day by $k\sigma$, where σ is the standard deviation to the count of cells with relative entropy exceeding the threshold on normal days. In a series of experiments on this scenario, setting k to 1.5 yielded an accurate detection of the anomaly events as shown in Figure 4(a).

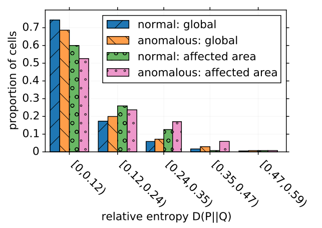
In the experiment of Figure 4(c), we consider another event, where a fire in the Lausanne train station interrupted the entire rail traffic in Western Switzerland during one hour on June 22, 2015. Compared to the previous event, which affected only the Geneva area, the region which was affected



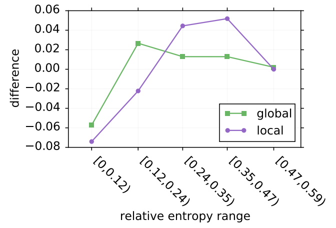
(a) Adjacent windows.



(b) Difference between normal and anomalous proportions of cells per range.



(c) Same hours but different days. Global vs local scale.



(d) Difference between normal and anomalous proportions of cells per range, on a global and on a local scale.

Figure 4. Distribution of the cells' mean relative entropy and difference between normal and anomalous days.

by this interruption is much larger. We use the data from the entire day, which is available from HDFS, and compute the relative entropy between the hourly event histograms from June 22 and from a non-anomalous day, as well as between non-anomalous days as a baseline. For comparing two days d_1 and d_2, we compute the relative entropy between the histogram summarizing hour h_i in cell c_j on d_1 and the histogram summarizing the same hour h_i in the same cell c_j on d_2. This allows us to take into account periodical behavioral changes throughout the day and compute the relative entropy between two histograms that are expected to be similar. We performed such computations on two geographical scales. First, the relative entropy is computed per cell globally for all data available in Switzerland.

Small-scale events affecting few cells (such as the previous scenario) are typically not observable on a global scale but only on a local scale. Second, cells from the city of Lausanne were filtered.

Similarly to the experiment of Figure 4(a), the results are plotted on a histogram showing the distribution of the cells' mean relative entropy (averaged over all hourly relative entropy values for each cell). In the global relative entropy distribution, we can observe a higher proportion of cells with higher mean relative entropy in the anomalous case than in the normal case. As Figure 4(d) shows, the differences between the normal and the anomalous metrics are more pronounced when considering only the area of Lausanne as affected area but are nevertheless visible on a global scale. The results of Figure 4(c) show that large-scale events affecting many cells are detectable on a global level. When using thresholds for anomaly detection, the parameters need to be defined individually for different geographical scales, since the global impact will commonly be lower than the regional impact.

In order to detect local anomalies, one might consider computing the relative entropy between two geographical units, e.g., cells. Relative entropy is not suited for this purpose, since each cell or area has its own distinctive distribution of event types related to its geographical location and thus, relative entropy between different locations would be high in any scenario.

2) Pearson Correlation Accuracy: The physical network monitoring probes from which we obtain the data are located close to Swisscom's core network. There is not one physical monitoring probe per base station (cell), as each physical device is responsible for an aggregate of cells. Probe failures are hence detectable by looking at large-scale changes, which are obtained by maintaining global Pearson correlation coefficients. Since none of the probes failed during the period of this study we resort to a simulation that aims to imitate a realistic failure scenario of a network monitoring probe. We simulate the failure of a monitoring device by filtering out the events in its area, that is, by removing all events that originate from a certain area for a given period of time. Due to the high computational cost of simultaneously processing several streams, this simulation was executed on events coming from only one LAC. This is a downscaled failure scenario of the case where the monitoring component belonging to one specific LAC on one interface ceases to transmit data.

In the non-anomalous scenario, the data streams coming from different telecommunication interfaces (2G and 3G, specifically) are highly correlated in the counts of users on the interfaces during a period in time. This domain knowledge helps to maintain correlation coefficients in order to detect changes that affect components belonging to one of the two interfaces. Infrastructure failures typically yield abrupt changes. In our case, since no users are counted for the area in which the probe has failed, fewer events get transmitted from the respective monitoring probe and thus lead to lower user (i.e., input) counts. As a result, we expect a sudden drop in the Pearson correlation coefficient $r(X,Y)$. Due to a mostly uniform loss of events, subsequent vectors again have higher correlation, hence we have detect the anomaly based on the abrupt drop of events. If the failure happens precisely between two windows, no decrease in $r(X,Y)$ can be observed. We therefore use overlapping windows to capture every possible failure.

Figure 5 displays the results of computing the Pearson correlation between the counts of distinct users on the 2G voice (A) and the 3G voice (IuCS) streams during one hour with one count every 10 seconds and one correlation score every 90 seconds. After 30 minutes, we filter out one third of the IuCS stream counts. The results show that both before and after the failure, $r(X,Y)$ between the

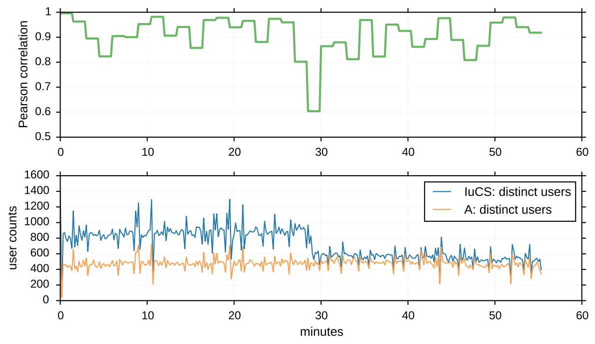


Figure 5. Impact of cessation of data transmission from one probe on the correlation between two streams.

counts over the two streams is consistently high (ranging between 0.8 and 1). Before the failure, high correlation of the streams is our baseline for detecting outliers. At failure time, there is a momentary decrease of the correlation to 0.6 during one 90 second window after which it stabilizes again to the previous range. This momentary decrease of 0.3 is significant considering the baseline average of 0.91 having a standard deviation σ of 0.06. We detected such cases by identifying correlation coefficients that deviate from the average by $k\sigma$; in our deployment, picking $k = 4$ gives overall an accurate detection of infrastructure failures in time.

Pearson correlation performs well for detecting abrupt changes. However, it is not suitable to detect gradual changes such as in events caused by human behavior. The anomaly detection accuracy could be further improved by computing the correlation between more streams simultaneously, though such computations are resource-intensive.

B. System Scalability

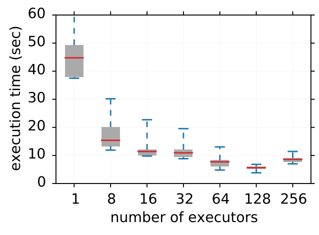
We now turn to evaluating the scalability of our anomaly detection system. As our system must be able to scale-out and to efficiently cope with large amounts of data, we vary both the number of executors (nodes) and the volume of the input data in the following.

1) Scale-Out Performance: Parallel processing is a key feature of Big Data infrastructures. In order to evaluate the scalability of our implementation, we conduct an experiment with a varying number of processing executors (nodes). Since we are running on top of YARN, each executor corresponds to an isolated memory partition. In our case, we consider memory partitions of 4GB. All the following experiments were executed during business hours, where we can observe a relatively constant data rate. Each stream is consumed in parallel by nine receivers (see Section IV). These receivers do not run any task other than receiving data. Executors used for receiving data do not process the data any further and directly distribute the data to other executors. In order to isolate the performance of the processing from the data receiving, we do not consider the stream consumers but only the executors performing the transformation and the subsequent computations. Thus, we add nine executors for each stream that is being consumed to the total number of executors but only consider the additional executors for the scale-up experiments.

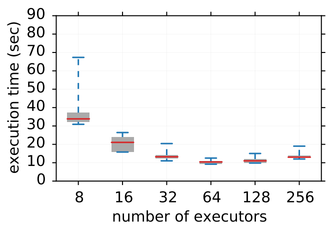
Since we are interested in the ability of the system to process the data streams in real-time, all experiments have been conducted over a period of 20 minutes. Micro-batches should on average be computed within very shorts periods of time. If the execution of the micro-batch task exceeds a predefined micro-batch time upper bound, then we switch to a different configuration.

a) Relative Entropy Scalability: In order to evaluate the scalability of the algorithm for computing the relative entropy $D(P_t \| Q_{t-\Delta t})$, we choose a Δt of 60 seconds, i.e., we compute the relative entropy between two adjacent windows of 60 seconds duration.

The experiment in Figure 6(a) shows the processing time with an increasing number of executors. This experiment shows that while using only one executor for processing the data, the algorithm terminates on average below the duration of a micro-batch, such that it is able to compute the relative entropy on the stream in real-time. The execution time and the variance decrease by adding more executors. A low variance helps to guarantee an upper bound for the execution time. A low execution time (that is as far as possible from the maximum execution time given by the micro-batch duration) is important in the case of failure. In fact, Spark's built-in recovery mechanisms guarantee that all data will be processed, though it may require recomputing some partitions and hence imposing some overhead computations. While lower executor counts fulfill the requirements, the optimum is reached at 128 executors. Increasing the parallelism beyond 128 executors does not speed up the processing any further. In our deployment, this is caused by the overhead of the Spark master when managing a large number of executors.



(a) Relative entropy algorithm on the A stream.



(b) Pearson correlation algorithm between A and IuCS streams.

Figure 6. Micro-batch processing times per number of executors.

b) Pearson Correlation Scalability: Given the size of the 3G data and the complexity of receiving and processing two streams at the same time, we consider only events from one LAC for the following correlation experiment. The experiment of Figure 6(b) displays the micro-batch processing time as a function of the number of executors for computing the Pearson correlation $r(X, Y)$. We limit this experiment to the respective batch duration considered. In the case of Pearson correlation, execution with less than 8 executors was consistently unable to terminate within the given 90 second batch duration. Therefore, these configurations are not shown in Figure 6(b). The experiment of Figure 6(b) shows that the average execution time decreases by adding more executors, until reaching a point where the executor management overhead decreases the performance. For the Pearson correlation algorithm, this point is reached at 64 executors, after which execution speed does not increase any further. The range of suitable configurations for computing the Pearson correlation between two streams is smaller than for computing relative entropy on one stream.

Despite the difference between the algorithms of the two metrics, we observe that the execution times with the optimal number of executors are similar, although slightly higher for Pearson correlation than for relative entropy. This results from the larger quantities of data needed for the computation of the Pearson correlation coefficient. This indicates that the algorithms as well as the Spark processing are efficient, both in terms of computing the algorithms in real-time and in scaling horizontally, and that a significant fraction of the processing time is caused by data shuffling and executor management.

2) Performance over Increased Data Load: For experimenting with increased data loads, we use the newest generation of mobile telecommunication protocols, currently 4G. 4G data has a higher throughput leading to larger amounts of event data to be processed than previous generations of mobile technology.

In order to evaluate the scalability of the algorithms over large volumes of data (measured in hours), we compare different workloads. Unlike the previous experiments, where we have considered the relative entropy and Pearson correlation separately, they are in the following experiment computed together. The number of executors was fixed to 80 and each executor was configured to 4GB of memory.

The computation times for varying data quantities are shown in Figure 7. We observe that the algorithms scale linearly with the quantity of the data. Taking around 15 minutes for one day of data, the system is able to process the data within a reasonable timeframe. With limited computational resources, we cannot afford computationally complex processes to handle the data. However, the usage of the shared resources on which anomaly detection is performed vary throughout the day. In cases where periodical batch processing is required, the batches can be processed during idle resource times, for which 15 minutes time windows sound reasonable in our case.

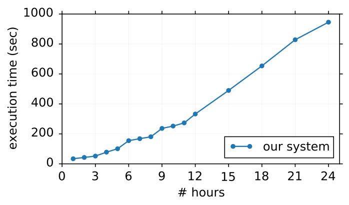


Figure 7. Joint computation of relative entropy and Pearson correlation: processing time vs. number of hours of collected data.

VI. Conclusion

In this paper, we introduced a new system leveraging both relative entropy and Pearson correlation for online anomaly detection over Big Data streams. Our solution builds on Spark and can be used for data streams as well as for data at rest. The results of the experiments we ran on real world data show that relative entropy is well-suited for detecting gradual changes caused by large-scale human activity, whereas Pearson correlation is best-suited for detecting abrupt changes, caused for example by network failures. Our proposed anomaly detection system works at different scales, both geographically (network topology) and temporally (length of windows, batch vs. stream processing). We also ran experiments showing that our system gracefully scales with the number of nodes and with the data volume.

In future work, we plan to integrate additional ground-truth data to automatically determine the correct thresholds when reporting the anomalies. Another promising direction is to extend our approach to anomaly clustering. Clustering would help us to determine the general types of the anomalies as well as to better evaluate the consistency and the quality of the data.

Acknowledgment

This work was partially supported by the Swiss National Science Foundation under grant number CRSII2_147609. The authors would like to thank Swisscom for providing the data and the infrastructure we used for the experiments. We would also like to give special thanks to the anonymous reviewers for their insightful comments.

References

[1] T. Dasu, S. Krishnan, S. Venkatasubramanian, and K. Yi, "An information-theoretic approach to detecting changes in multi-dimensional data streams," in *In Proc. Symp. on the Interface of Statistics, Computing Science, and Applications*, 2006.

[2] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster Computing with Working Sets," in *Proceedings of the 2Nd USENIX Conference on Hot Topics in Cloud Computing*, 2010, pp. 10–10.

[3] J. Zhang, M. Lou, T. W. Ling, and H. Wang, "HOS-Miner: a system for detecting outlying subspaces of high-dimensional data," in *Proceedings of the 30th International Conference on Very Large Databases*, Toronto, Canada, 2004, pp. 1265–1268.

[4] X. Li and J. Han, "Mining approximate top-k subspace anomalies in multi-dimensional time-series data," in *Proceedings of the 33rd International Conference on Very Large Data Bases, Vienna, Austria, September 23-27, 2007*, 2007, pp. 447–458.

[5] W. C. Young, J. E. Blumenstock, E. B. Fox, and T. H. Mccormick, "Detecting and classifying anomalous behavior in spatiotemporal network data," in *The 20th ACM Conference on Knowledge Discovery and Mining (KDD '14), Workshop on Data Science for Social Good*, New York, NY, 2014.

[6] M. Gupta, J. Gao, C. C. Aggarwal, and J. Han, "Outlier detection for temporal data: A survey," *IEEE Transactions on Knowledge and Data Engineering*, vol. 25, no. 1, pp. 1–1, 2014.

[7] Q. Wu and Z. Shao, "Network Anomaly Detection Using Time Series Analysis," in *Joint International Conference on Autonomic and Autonomous Systems and International Conference on Networking and Services - (icas-isns'05)*, 2005, p. 42.

[8] M. Datar, A. Gionis, P. Indyk, and R. Motwani, "Maintaining stream statistics over sliding windows," *SIAM Journal on Computing*, vol. 31, pp. 1794–1813, 2002.

[9] O. Papapetrou, M. Garofalakis, and A. Deligiannakis, "Sketch-based querying of distributed sliding-window data streams," in *Proceedings of the VLDB Endowment*, vol. 5, 2012, pp. 992–1003.

[10] G. Cormode and S. Muthukrishnan, "An improved data stream summary: The count-min sketch and its applications," *Journal of Algorithms*, vol. 55, no. 1, pp. 58–75, 2005.

[11] G. Cormode and S. Muthukrishnan, "What's new: Finding significant differences in network data streams," *INFOCOM 2004. Twenty-third Annual Joint Conference of the IEEE Computer and Communications Societies*, vol. 3, pp. 1534–1545, 2004.

[12] S. Kullback and R. A. Leibler, "On information and sufficiency," *Ann. Math. Statist.*, vol. 22, no. 1, pp. 79–86, 03 1951.

[13] M. Zaharia, M. Chowdhury, T. Das, A. Dave, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica, "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing," in *NSDI'12 Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, 2012, pp. 2–2.

[14] M. Zaharia, T. Das, H. Li, T. Hunter, S. Shenker, and I. Stoica, "Discretized streams: fault-tolerant streaming computation at scale," in *SOSP '13 Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, 2013, pp. 423–438.

[15] The Apache Software Foundation, "Apache Avro," 2015. [Online]. Available: https://avro.apache.org/

[16] The Apache Software Foundation, "Apache Kafka," 2015. [Online]. Available: https://kafka.apache.org/

[17] N. Marz and J. Warren, *Big Data: Principles and best practices of scalable realtime data systems*. Greenwich, CT: Manning Publications Co., 2013.