Real time contextual collective anomaly detection over multiple data streams

Yexi Jiang, Chunqiu Zeng
School of Computing and Information Sciences
Florida International University
Miami, FL, USA
{yjian004, czeng001}@cs.fiu.edu

Jian Xu
School of Computer Science Technology and Engineering
Nanjing University of Science and Technology
Nanjing, China
dolphin.xu@njust.edu.cn

Tao Li
School of Computing and Information Sciences
Florida International University
Miami, FL, USA
taoli@cs.fiu.edu

ABSTRACT

Anomaly detection has always been a critical and challenging problem in many application areas such as industry, healthcare, environment and finance. This problem becomes more difficult in the Big Data era as the data scale increases dramatically and the type of anomalies gets more complicated. In time sensitive applications like real time monitoring, data are often fed in streams and anomalies are required to be identified online across multiple streams with a short time delay. The new data characteristics and analysis requirements make existing solutions no longer suitable.

In this paper, we propose a framework to discover a new type of anomaly called contextual collective anomaly over a collection of data streams in real time. A primary advantage of this solution is that it can be seamlessly integrated with real time monitoring systems to timely and accurately identify the anomalies. Also, the proposed framework is designed in a way with a low computational intensity, and is able to handle large scale data streams. To demonstrate the effectiveness and efficiency of our framework, we empirically validate it on two real world applications.

1. INTRODUCTION

Anomaly detection is one of the most important tasks in data-intensive applications such as the healthcare monitoring [22], stock analysis [26], disaster management [32], system anomaly detection [24], and manufacture RFID management [4], etc. In the Big Data era, the aforementioned applications often require real-time processing. However, existing data processing infrastructures are designed based on inherent non-stream programing paradigm such as MapReduce [11], Bulk Synchronous Parallel (BSP) [30], and their variations. To reduce the processing delay, these applications have gradually migrated to stream processing engines [27, 10, 1, 9]. As the infrastructures have been changed, anomalies in these applications are required to be identified online across multiple data streams. The new data characteristics and analysis requirements make existing anomaly detection solutions no longer suitable.

ODD'14, August 24 - 27 2014, New York, NY, USA.
Copyright 2014 ACM 978-1-4503-2998-9/14/08 ...$15.00
http://dx.doi.org/10.1145/2656269.2656271.

1.1 A Motivating Example

Example 1. *Figure 1 illustrates the scenario of monitoring a 6-node computer cluster, where the x-axis denotes the time and the y-axis denotes the CPU utilization. The cluster has been monitored during time* $[0, t_6]$. *At time* t_2, *a computing task has been submitted to the cluster and the cluster finishes this task at time* t_4. *As shown, two nodes (marked in dashed line) behave differently from the majority during some specific time periods. Node ① has a high CPU utilization during* $[t_1, t_2]$ *and a low CPU utilization during* $[t_3, t_4]$ *while node ② has a medium CPU utilization all the time. These two nodes with their associated abnormal periods are regarded as anomalies. Besides these two obvious anomalies, there are a slight delay on node ③ due to the network delay and a transient fluctuation on node ④ due to some random factors. However, they are normal phenomena in distributed systems and are not regarded as anomalies.*

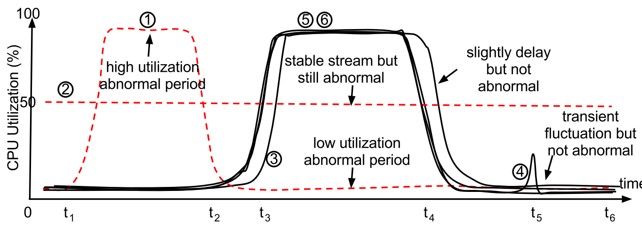

Figure 1: CPU utilization of a computing cluster

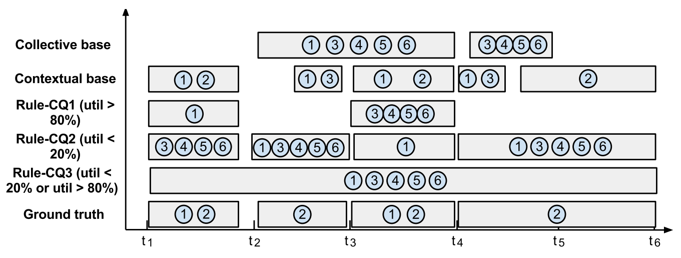

Figure 2: Identified anomalies in Example 1 (The box lists the IDs of abnormal streams during specified time period)

A quick solution for stream based anomaly detection is to leverage the techniques of *complex event processing (CEP)* [23, 2] by expressing the anomalies detection rules with corresponding continuous query statements. This rule-based detection method can be applied to the scenarios where the anomaly can be clearly defined. Besides using CEP, several stream based anomaly detection algorithms have also been proposed. They either focus on identifying contextual anomaly over a collection of stable streams [7] or collective anomaly from one stream [3, 25]. These existing methods are useful in many applications but they still cannot identify certain types of anomalies. A simple example of such scenario is illustrated in Example 1.

Figure 2 plots the ground truth as well as all the anomalies identified by existing methods including the CEP query with three different rules (Rule-CQ1, 2, and 3), the collective based anomaly detection [6], and contextual based anomaly detection [8].

To detect the anomalies via CEP query, the idea is to capture the events when the CPU utilizations of nodes are too high or too low. An example query following the syntax of [2] can be written as follows:

```
PATTERN SEQ(Observation o[])
WHERE avg(o[].cpu) oper threshold
  (AND|OR avg(o[].cpu) oper threshold)*
WITHIN {length of sliding window}
```

where the selection condition in `WHERE` clause is the conjunction of one or more boolean expressions, *oper* is one of {>, <, <>, ==}, and `threshold` can be replaced by any valid expression. However, CEP queries are unable to correctly identify the anomalies in Figure 1 no matter how the selection conditions are specified. For instance, setting the condition as `avg(o[].cpu) > {threshold}` would miss the anomalies during [t_3, t_4] (Rule-CQ1); setting the condition as `avg(o[].cpu) < {threshold}` would miss the anomalies during [t_1, t_2] (Rule-CQ2); and combining the two above expressions with OR still does not work (Rule-CQ3). Besides deciding the selection condition, how to rule out the situations of slight delays and transient fluctuations, and how to set the length of the sliding windows are all difficult problems when writing the continuous queries. The main reason is that the continuous query statement is not suitable to capture the contextual information where the "normal" behaviors are also dynamic (the utilizations of normal nodes also change over time in Figure 1).

Compared with CEP based methods, contextual anomaly detection methods (such as [14, 17]) achieve a better accuracy as they utilize the contextual information of all the streams. However, one limitation of contextual based methods is that they do not leverage the temporal information of streams and are not suitable for anomaly detection in dynamic environments. Therefore, these methods would wrongly identify the slightly delayed and fluctuated nodes as anomalies.

For the given example, collective anomaly detection methods do not work well neither. This is because these methods would identify the anomaly of each stream based on its normal behaviors. Once the current behavior of a stream is different from its normal behaviors (identified based on historical data), it is considered as abnormal. In the example, when the cluster works on the task during $[t_3, t_4]$, all the working nodes would be identified as abnormal due to the sudden burst.

1.2 Contributions

In this paper, we propose an efficient solution to identify this special type of anomaly in Example 1, named *contextual collective anomaly*. Contextual collective anomalies bear the characteristics of both contextual anomalies and collective anomalies. This type of anomaly is common in many applications such as system monitoring, environmental monitoring, and healthcare monitoring, where data come from distributed but homogeneous data sources. We will formally define this type of anomaly in Section 2.

Besides proposing an algorithm to discover the contextual collective anomalies over a collection of data streams, we also consider the scale-out ability of our solution and develop a distributed streaming processing framework for contextual collective anomaly detection. More concretely, our contributions can be described as follows:

- We provide the definition of contextual collective anomaly and propose an incremental algorithm to discover the contextual collective anomalies in real time. The proposed algorithm combines the contextual as well as the historical information to effectively identify the anomalies.
- We propose a flexible three-stage framework to discover such anomalies from multiple data streams. This framework is designed to be distributed and can be used to handle large scale data by scaling out the computing resources. Moreover, each component in the framework is pluggable and can be replaced if a better solution is proposed in the future.
- We empirically demonstrate the effectiveness and efficiency of our solution through the real world scenario experiments.

The rest of the paper is organized as follows. Section 2 gives a definition of contextual collective anomaly and then presents the problem statement. Section 3 provides an overview of our proposed anomaly detection framework. We introduce the three-stage anomaly detection algorithm in detail in Section 4. Section 5 presents the result of experimental evaluation. The related works are discussed in Section 6. Finally, we conclude in Section 7.

2. PROBLEM STATEMENT

In this section, we first give the notations and definitions that are relevant to the anomaly detection problem. Then, we formally define the problem based on the given notations and definitions.

DEFINITION 1. DATA STREAM. *A data stream S_i is an ordered infinite sequence of data instances $\{s_{i1}, s_{i2}, s_{i3}, ...\}$. Each data instance s_{it} is the observation of data stream S_i at timestamp t.*

The data instances s_{it} in S_i can have any number of dimensions, depending on the concrete applications. For the remaining of this paper, the terms "data instance" and "observation" would be used interchangeably. *To make the notation uncluttered, we use s_i in places where the absence of timestamp does not cause the ambiguity.*

Definition 2. Stream Collection. *A stream collection $\mathcal{S} = \{S_1, S_2, ..., S_n\}$ is a collection of data streams. The number of streams $|\mathcal{S}| = n$.*

The input of our anomaly detection framework is a *stream collection.* For instance, in example 1, the input stream collection is $\mathcal{S} = \{①, ②, ③, ④, ⑤, ⑥\}$

Definition 3. Snapshot. *A snapshot is a set of key-value pairs $S^{(t)} = \{S_i : s_{it} | S_i \in \mathcal{S}\}$, denoting the set of the observations $\{s_{1t}, s_{2t}, \cdots, s_{|S|t}\}$ of the data streams in stream collection $\mathcal{S}$ at time t.*

A *snapshot* captures the configuration of each data stream in the stream collection for a certain moment. Taking Figure 1 for example, the snapshot at time t_5 is $S^{(t_5)} = \{① : 0\%, ② : 50\%, ③ : 0\%, ④ : 20\%, ⑤ : 0\%, ⑥ : 0\%\}$. For simplicity, we use $S(i)$ to denote the ith dimension of the observations in a certain snapshot. Note that all the observations in a snapshot have the same dimension.

Definition 4. Contextual Collective Anomaly. *A contextual collective stream anomaly is denoted as a tuple $< S_i, [t_b, t_e], N >$, where S_i denote a data stream from the collection of data streams $\mathcal{S}$, $[t_b, t_e]$ is the associated time period when S_i is observed to constantly deviate from the majority streams in S, and N indicates the severity of the anomaly.*

In Example 1, 3 contextual collective anomalies can be found in total. During time period $[t_1, t_2]$, node ① behaves constantly different from the other nodes, so there is an anomaly $<①, [t_1, t_2], N_1>$. The other two contextual collective anomalies, $<①, [t_3, t_4], N_2>$ and $<②, [0, t_6], N_3>$, can also be found with the same reason.

In Definition 4, the severity of deviation is measured in a given metric space $\mathcal{M}$ with a distance function $f : s_{it} \times s_{ku} \to \mathbb{R}$. For simplicity, we use Euclidean distance as an example throughout this paper.

Problem Definition. The anomaly detection problem in our paper can be described below: Given a stream collection $\mathcal{S} = \{S_1, S_2, ..., S_m\}$, identify the source of the contextual collective anomalies S_i, the associated time period $[t_s, t_e]$, as well as a quantification about the confidence of the detection p. Moreover, the detection has to be conducted on data streams that look-back is not allowed and the anomalies are able to be identified in real time.

3. FRAMEWORK OVERVIEW

In this section, we briefly describe how the aforementioned problem is addressed and then introduce the proposed distributed real time anomaly detection framework from a high level perspective.

As previously mentioned, *this paper focuses on discovering contextual collective anomalies over a collection of data streams obtained from a homogeneous distributed environment.* An example of the homogeneous distributed environment is the system with load balance, which is widely used at the backend by the popular web sites like Google, Facebook, and Amazon, etc.

It is known that, in such a kind of environment, the components should behave similar to each other. Therefore, the snapshots (the current observations) of these streams should be close to each other at any time. Naturally, we need to identify the anomalies by investigating both contextual information (the information of the current snapshot) and collective information (the historical information).

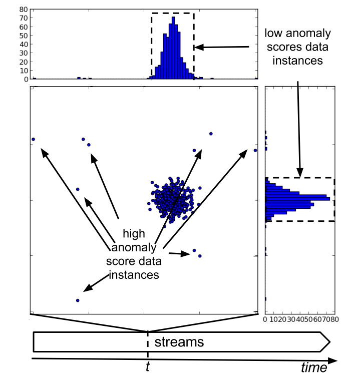

Figure 3: The snapshot at a certain timestamp

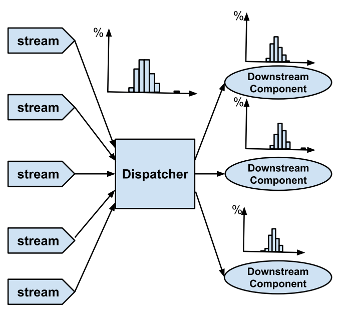

Figure 4: Random partition of data instance

The anomaly detection is conducted in three stages: the *dispatching stage*, the *scoring stage*, and the *alert stage.* The functionality of the three stages are briefly described as follows:

- *Dispatching stage:* This stage uses *dispatchers* to receive the observations from external data sources and then shuffle the observations to different downstream processing components.

- *Scoring stage:* This stage quantifies the candidate anomalies using *snapshot scorer* and then *stream scorer.*

  The *snapshot scorer* leverages contextual information to quantify the confidence of anomaly for each data instance at a given *snapshot.* Taking Figure 3 for example, it shows the data distribution by taking the snapshot of the 2-dimensional data instances of 500 streams at timestamp t. As shown, most of the data instances are close to each other and located in a dense area. These data instances are not likely to be identified as anomalies as their instance anomaly scores are small. On the contrary, a small portion of the data instances (those points that are far away from the dense region) have larger instance anomaly scores and are more likely to be abnormal.

  A data instance with a high anomaly score does not indisputably indicate its corresponding stream to be a real anomaly. This is because the transient fluctuation and phase shift are common in real world distributed environment. To mitigate such effects, the *stream scorer* is designed to handle the problem. In particular, the *stream scorer* combines the information obtained from the *instance scorer* and the historical information of each stream to quantify the anomaly confidence of each stream.

- *Alert stage:* The alert stage contains the *alter trigger.* The alert triggers leverage the unsupervised learning methods to identify and report the outliers.

The advantage of our framework is reflected by the ease of integration, the flexibility, and the algorithm independence.

Firstly, any external data sources can be easily fed to the framework for anomaly detection. Moreover, the components in every stage can be scaled-out to increase the processing capability if necessary. The number of components in each stage can be easily customized according to the data scale of concrete applications. Furthermore, the algorithms in each stage can be replaced and upgraded with better alternatives and the replacement would not interfere with other stages.

4. METHODOLOGY

4.1 Data Receiving and Dispatching

The dispatching stage is an auxiliary stage in our framework. When the data scale (i.e., the number of streams) is too large for a single computing component to process, the dispatcher would shuffle the received observations to downstream computing components in the scoring stage (as shown in Figure 4). By leveraging random shuffling algorithm like Fisher-Yates shuffle [12], dispatching can be conducted in constant time per observation. After dispatching, each downstream component would conduct scoring independently on a sampled stream observations with identical distribution.

Ideally, the observations coming from homogeneous data sources have very similar measurable value (e.g. workload of each server in a load balanced system), but random factors can easily cause the variations of the actual observations. Therefore in fact, an observation $\boldsymbol{s_i} \in \mathbb{R}^d$ is viewed as the ideal case value $\boldsymbol{s_{ideal}}$ with additive Gaussian noise so that $\boldsymbol{s_i} = \boldsymbol{s_{ideal}} + \boldsymbol{\epsilon}$, $\boldsymbol{\epsilon} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$. For those data sources in abnormal conditions, their observations are generated according to a different but unknown distributed. It is not difficult to know that, given enough observations, the mean and covariance can be easily estimated locally via maximum likelihood, i.e. $\hat{\boldsymbol{\mu}}_{ML} = \frac{\sum_i \boldsymbol{s_i}}{n}$ and $\widehat{cov}(S(x), S(y))_{ML} = \frac{1}{n-1}\sum_{i=1}^{n}(s_i(x) - \bar{S}(x))(s_i(y) - \bar{S}(y))$, where $\bar{S}(x)$ and $\bar{S}(y)$ respectively denote the sample mean of dimension x and y.

4.2 Snapshot Anomaly Quantification

Quantifying the anomaly in a snapshot is the first task in the *scoring stage* and we leverage *snapshot scorer* in this step. This score measures the amount of deviation of the specified observation s_{it} to the center of all the observations in a snapshot $S^{(t)}$ at timestamp t.

To quantify the seriousness of snapshot anomaly, we propose a simple yet efficient method. The basic idea is that the anomaly score of an observation is quantified as the amount of uncertainty it brings to the snapshot $S^{(t)}$. As the observations in a snapshot follows the normal distribution, it is suitable to use the increase of entropy to measure the anomaly of an observation. To quantify the anomaly score, two types of variance are needed: the *variance* and the *leave-one-out variance*, where the leave-one-out variance is the variance of the distribution when one specific data instance is not counted.

A naive algorithm to quantify the anomaly scores requires quadratic time ($O(dn + dn^2)$). By reusing the intermediate results, we propose an improved algorithm with time complexity linear to the number of streams. The pseudo code of the proposed algorithm is shown in Algorithm 1. As illustrated, matrix M is used to store the distances between each

Algorithm 1 Snapshot Anomaly Quantification

```
1. INPUT: Snapshot $S^{(t)} = (s_1, \cdots, s_n)$, where $s_i \in \mathbb{R}^d$.
2. OUTPUT: Snapshot anomaly scores $N \in \mathbb{R}^n$.
3. Create a $d \times n$ matrix $M = (s_1, \cdots, s_n)$
4. Conduct 0-1 normalization on rows of $M$.
5. $\boldsymbol{x} \leftarrow \boldsymbol{0}^d$ and $N = \boldsymbol{0}^d$
6. $\boldsymbol{m} \leftarrow (\mathbb{E}(S(1)), \cdots, \mathbb{E}(S(d))^T$
7. $M \leftarrow (s_{1t} - \boldsymbol{m}, s_{2t} - \boldsymbol{m}, \cdots, s_{dt} - \boldsymbol{m})$
8. for $j \leftarrow 0$ to $d$ do
9.    $\boldsymbol{x}_j = ||j$th column of $M||_2^2$
10. end for
11. for all $s_i \in S^{(t)}$ do
12.    Calculate $N_i$ according to Equation (1).
13. end for
14. Conduct 0-1 normalization on $N$.
15. return $N$
```

dimension of the observations to the corresponding mean. Making use of M, the *leave-one-out variance* can be quickly calculated as $\sigma_{ik}^2 = \frac{n\sigma_k^2 - M_{i,k}^2}{n-1}$, where σ_k^2 denotes the variance of dimension k and σ_{ik}^2 denotes the leave-one-out variance of dimension k by excluding s_i. As the entropy of normal distribution is $H = \frac{1}{2}\ln(2\pi e\sigma^2)$, the increase of entropy for observation s_i at dimension k can be calculated as

$$d_k = H_k' - H_k = \ln\frac{\sigma_{ik}^2}{\sigma_k^2} = \ln\frac{(\boldsymbol{x}_k - M_{ik}^2)/(n-1)}{\boldsymbol{x}_j/n}, \quad (1)$$

where H_k' and H_k respectively denote the entropy of the snapshot distribution if s_i is not counted or counted.

Summing up all dimensions, the snapshot anomaly score of s_{it} is $N_{it} = \sum_k d_k$. Note that the computation implicitly ignores the correlation between dimensions. The reason is that if an observation is an outlier, the correlation effect would only deviate it further from other observations.

4.3 Stream Anomaly Quantification

As a stream is continuously evolving and its observations only reflect the transient behavior, snapshot anomaly score alone would result in a lot of false-positives due to the transient fluctuation and slight phase shift. To mitigate such situations, it is critical to quantify the stream anomaly by incorporating the historical information of the stream.

An intuitive way to solve this problem is to calculate the stream anomaly score from the recent historical instances stored in a sliding window. However, this solution has two obvious limitations: (1) It is hard to decide the window length. A long sliding window would miss the real anomaly while a short sliding window cannot rule out the false-positives. (2) It ignores the impact of observations that are not in the sliding window. The observation that is just popped out from the sliding window would immediately and totally lose its impact to the stream.

To well balance the history and the current observation, we use *stream anomaly score* N_i to quantify how significant a stream S_i behaves differently from the majority of the streams. To quantify N_i, we exploit the exponential decay function to control the influence depreciation. Supposing Δt is the time gap between two adjacent observations, *the influence of an observation* s_{it} *at timestamp* $t_{x+k} = t_x + k\Delta t$ can be expressed as $N_{it_x}(t_{x+k}) = N_{it_x}(t_x + k\Delta t) =$

$N_{it_x}e^{-\lambda kt}, (\lambda > 0)$, where λ is a parameter to control the decay speed. In the experiment evaluation, we will discuss how this parameter affects the anomaly detection results.

To make the notation uncluttered, we use t_{-i} to denote the timestamp that is $i\Delta t$ ahead of current timestamp t, i.e. $t_{-i} = t - i\Delta t$. Summing up the influences of all the historical observations, the overall historical influence I_{it} for current timestamp t can be expressed as Equation (2).

$$\begin{aligned} I_{it} &= N_{it_{-1}}(t) + N_{it_{-2}}(t) + N_{it_{-3}}(t) + ... \\ &= N_{it_{-1}}e^{-\lambda} + N_{it_{-2}}e^{-2\lambda} + N_{it_{-3}}e^{-3\lambda} + ... \\ &= e^{-\lambda}(N_{it_{-1}} + e^{-\lambda}(N_{it_{-2}} + e^{-\lambda}(N_{it_{-3}} + ... \\ &= e^{-\lambda}(N_{it_{-1}} + I_{it_{-1}}). \end{aligned} \tag{2}$$

The stream anomaly score of stream S_i is the summation of the data instance anomaly score of current observation N_{it} and the overall historical influence, i.e.,

$$N_i = N_{it} + I_{it}. \tag{3}$$

As shown in Equation (2), the overall historical influence can be incrementally updated with cost $O(1)$ for both time and space complexity. Therefore, *stream anomaly scorer* can be efficiently computed.

4.3.1 Properties of Stream Anomaly Score

The properties of stream anomaly score make our framework insensitive to the transient fluctuation and effective to capture the real anomaly.

Comparing to the transient fluctuation, the real anomaly is more durable. Figure 5 shows the situations of a transient fluctuation (in the left subfigure) and a real anomaly(in the right subfigure). In both situations, the stream behaves normally before timestamp t_x. For the left situation, a transient fluctuation occurs at timestamp t_{x+1}, and then the stream returns to normal at timestamp t_{x+2}. For the right situation, an anomaly begins at timestamp t_{x+1}, lasts for a while till timestamp t_{x+k}, and then the stream returns to normal afterwards. Based on Figure 5, we show two properties about the stream anomaly score.

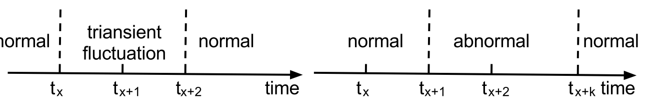

Figure 5: Transient Fluctuation and Anomaly

PROPERTY 1. *The increase of stream anomaly score caused by transient disturbance would decrease over time.*

PROPERTY 2. *The increase of stream anomaly score caused by anomaly would be accumulated over time.*

Similar properties can also be shown for the situation of slight shifts. A slight shift can be treated as two transient fluctuations occur at the beginning and the end of the shift. In the next section, we will leverage these two properties to effectively identify the anomalies in the ALERT STAGE.

4.4 Alert Triggering

Most of the stream anomaly detection solutions [13] identify the anomalies by picking the streams with top-k anomaly scores or the ones whose scores exceed a predefined threshold. However, these two approaches are not practical in real world applications for the following reasons: (1) *Threshold is hard to set.* It requires the users to understand the underlying mechanism of the application to correctly set the parameter. (2) *The number of anomalies are changing all the time.* It is possible that more than k anomaly streams exist at one time, then the top-k approach would miss these real anomalies.

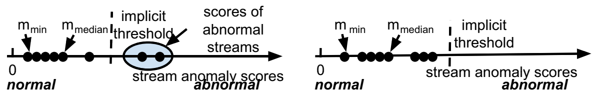

Figure 6: Abnormal Streams Identification

To eliminate the parameters, we propose an unsupervised method to identify and quantify the anomalies by leveraging the distribution of the anomaly scores. The first step is to find the median of the stream anomaly scores (N_{median}). If the distance between a stream anomaly score and the median score is larger than the distance between the median score and the minimal score (N_{min}), the corresponding stream is regarded as abnormal. As shown in Figure 6, this method implicitly defines a dynamic threshold (shown as the dashed line) based on the hypothesis that there is no anomaly. If there is no anomaly, the skewness of the anomaly score distribution should be small and the median score should be close to the mean score. If the hypothesis is true, $N_{median} - N_{min}$ should be close to half of the distance between the minimum score and the maximum score. On the contrary, if a score N_i is larger than $2 \times (N_{median} - N_{min})$, the hypothesis is violated and all the streams with scores at least N_i are abnormal.

Besides the general case, we also need to handle one special case: a transient fluctuation occurs at the current timestamp. According to Property (1) in Section 4.3.1, the effect of transient fluctuation is at most $d_{upper} = N_{it_{x+1}} - N_{jt_{x+1}}$ and it will monotonically decrease. Therefore, even a stream whose anomaly score is larger than $2 \times (N_{median} - N_{min})$, it can still be a normal stream if the difference between its anomaly score and N_{min} is smaller than d_{upper}. To prune the false-positive situations caused by transient fluctuation, the stream is instead identified as abnormal if

$$N_i > max(2(N_{median} - N_{min}), N_{min} + d_{upper}). \tag{4}$$

Another thing needs to be noted is that the stream anomaly scores have a upper bound $\frac{d_{upper}}{1-e^{-\lambda}}$. According to the property of convergent sequence, the stream anomaly scores of all streams would converge to this upper bound. When the values of stream anomaly scores are close to the upper bound, they tend to be close to each other and hard to be distinguished. To handle this problem, we reset all the stream anomaly scores to 0 whenever one of them close to the upper bound.

In terms of the time complexity, the abnormal streams can be found in $O(n)$ time. Algorithm 2 illustrates the algorithm of stream anomalies identification. The median of the scores can be found in $O(n)$ in the worst case using the *BFPRT algorithm [5].* Besides finding the median, this algorithm also partially sorts the list by moving smaller scores before the median and larger scores after the median, making it trivial to identify the abnormal streams by only checking the streams appearing after the median.

5. EXPERIMENTAL EVALUATION

Algorithm 2 Stream Anomaly Identification

```
1. INPUT: $\lambda$, and unordered stream profile list $\mathcal{S} = \{S_1, ..., S_n\}$.
2. $mIdx \leftarrow \lceil \frac{|\mathcal{S}|}{2} \rceil$
3. $N_{median} \leftarrow$ BFPRT($\mathcal{S}$, $mIdx$)
4. $N_{min} \leftarrow \min(S_i.score | 0 \leq i \leq mIdx)$
5. $N_{max} \leftarrow N_{median}$
6. for $i \leftarrow mIdx$ to $|\mathcal{S}|$ do
7.   if Condition (4) is satisfied then
8.     Trigger alert for $S_i$ with score $N_i$ at current time.
9.     if $N_i > N_{max}$ then
10.      $N_{max} \leftarrow N_i$
11.    end if
12.  end if
13. end for
14. if $N_{max}$ is close to the upper bound then
15.   Reset all stream anomaly scores.
16. end if
```

To investigate the effectiveness and efficiency of our framework, we design several sets of experiments with one real world data applications: *anomaly detection over a computing cluster*. Taking these two applications as case studies, we show that our proposed framework can effectively identify the abnormal behavior of streams. It should be pointed out that our proposed framework can also be applied to many other application areas such as PM2.5 environment monitoring, healthcare monitoring, and stock market monitoring.

System anomaly detection is one of the critical tasks in system management. In this set of experiments, we show that our proposed framework can effectively and efficiently discover the abnormal behaviors of the computer nodes with high precision and low latency.

5.1 Experiment Settings

For the experiments, we leverage a distributed system monitoring tool [31] into a 16-node computing cluster. Then we deploy the proposed anomaly detection program on an external computer to analyze the collected trace data in real time. To well evaluate our proposed framework, we terminate all the irrelevant processes running on these nodes. On this node, we intentionally inject various types of anomalies and monitor their running status for 1000 seconds. The source code of injection program is available at `https://github.com/yxjiang/system-noiser`. The details of the injections are listed in Table 1.

Table 1: List of Injections

No.	Time Period	Node	Description
1	[100, 150]	2	Keep CPU utilization above 95.
2	[300, 400]	3	Keep memory usage at 70%.
3	[350, 400]	3	Keep CPU utilization above 95%.
4	[600, 650]	4	Keep memory usage at 70%.
5	[900, 950]	2,5	Keep CPU utilization above 95%.
6	[800, 850]	1-5,7-16	Keep CPU utilization above 95%.

Through these injections, we can answer the following questions about our framework: (1) Whether our framework can identify the anomalies with different types of root causes. (2) Whether our framework can identify multiple anomalies occurring simultaneously.

5.2 Results Analysis

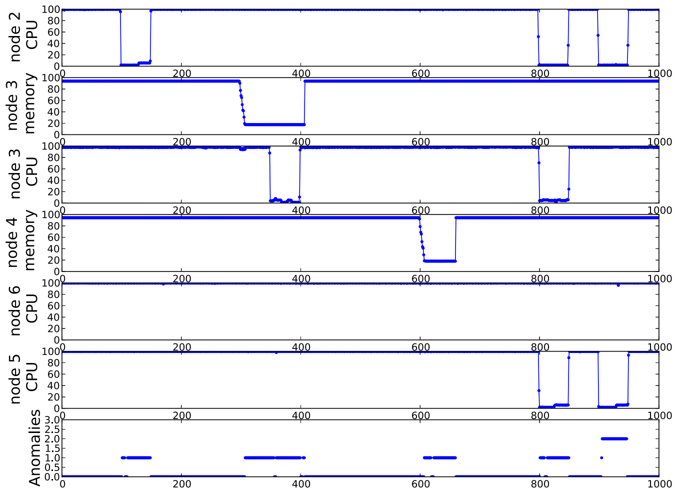

Figure 7: Injections and the captured alerts

Figure 7 illustrates the results of this experiment by plotting the actual injections (top 6 sub-figures) as well as the captured alerts (the bottom subplots), where the x-axis represents the time and y-axis represents the idled CPU utilization, idle memory usage or the number of anomalies in each timestamp. We evaluate the framework from 5 aspects through carefully-designed injections.

1. *Single dimension (e.g. idle CPU utilization or idle memory usage) of a single stream behaves abnormally.* This is the simplest type of anomalies. It is generated by injections No.1 and No.4 in Table 1. As shown in Figure 7, our framework effectively identifies these anomalies with the correct time periods.

2. *Multiple dimensions (e.g. CPU utilization and memory usage) of a single stream behaves abnormally at the same time.* This type of anomalies is generated by injections No.2 and No.3 in Table 1, and our framework correctly captures such anomalies during the time period [300, 400]. One thing should be noted is that the stream anomaly score of node 3 increases faster during the time period [350, 400] than the time period [300, 350]. This is because two types of anomalies (CPU utilization and memory usage) appear simultaneously during the time period [350, 400].

3. *Multiple streams behave abnormally simultaneously.* This type of anomalies is generated by injection No.5. During the injection time period, our framework correctly identifies both anomalies (on node 2 and node 5).

4. *Stable but abnormal streams.* This kind of anomaly is indirectly generated by injection No.6 in Table 1. This injection emulates the scenario that all the nodes but one (i.e., node 6) in a cluster received the command of executing a task. As is shown, although the CPU utilization of node 6 behaves stable all the time, it is still considered to be abnormal during the time period [800, 850]. This is because it remains idle when all the other nodes are busy.

5. *Transient fluctuation and slight delay would not cause false-positive.* As this experiment is conducted in a distributed environment, delays exist and vary for different nodes when executing the injections. Despite

this intervention, our framework still does not report transient fluctuations and slight delays as anomalies.

Based on the evaluation results, we find that our solution is able to correctly identify all the anomalies in all these 5 different cases.

5.3 Results Comparison

To demonstrate the superiority of our framework, we also conduct experiments to identify the anomalies with the same injection settings using the alternative methods including *contextual anomaly detection (CAD)* and *rule-based continuous query (Rule-CQ)*. The contextual anomaly detection is equivalent to the snapshot scoring in our framework. For the rule-based continuous query, we define three rules to capture three types of anomalies, including high CPU utilization (rule 1), low CPU utilization (rule 2), and high memory usage anomalies (rule 3), respectively. Different combinations of the three rules are used in the experiments.

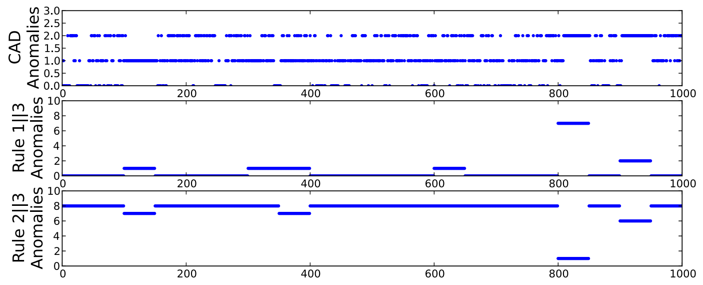

Figure 8: Generated alerts by CAD and Rule-CQ

The generated alerts of these methods are shown in Figure 8, where the x-axis denotes the time and y-axis denotes the number of anomalies. As illustrated, the contextual anomaly detection method generates a lot of false alerts. This is because this method is sensitive to the transient fluctuation. Once an observation deviates from the others at a timestamp, an alert would be triggered. For Rule-CQ method, we experiment all the combinations and report the results of the two best combinations: C1 (rule 1 or rule 2) and C2 (rule 2 or rule 3). Similarly, the Rule-CQ method also generates many false alerts since it is difficult to use rules to cover all the anomaly situations. Table 2 quantitatively shows the precision, recall, and F-measure of the three methods as well as the results of our method. The low-precision and high-recall results of CAD and Rule-CQ indicate that all these method are too sensitive to fluctuations.

Table 2: Measures of different methods performed on the trace data in distributed environment

Method / Measure	precision	recall	F-measure
CAD	0.4207	1.0000	0.5922
C1: Rule 1||3	0.5381	1.0000	0.6997
C2: Rule 2||3	0.0469	1.0000	0.0897
Our method (worst case)	0.9832	0.8400	0.9060

5.4 A real system problem detected

We have identified a real system problem when deployed our framework on two computing clusters in our department. In one of the clusters, we continuously receive alerts. Logging into the cluster, we find the CPU utilization is high even no tasks are running. We further identify that the high CPU utilization is caused by several processes named *hfsd*. We reported the anomaly to IT support staffs and they confirmed that there exist some problems in this cluster. The high CPU utilization is caused by continuous attempts to connect to a failure node in the network file system. After fixing this problem, these out-of-expectation but real alerts disappear.

6. RELATED WORKS

Although anomaly detection has been studied for years [15, 8]. To the best of our knowledge, our method is the first one that focused on mining contextual collective anomalies among multiple streams in real time. In this section, we briefly review two closely related areas: mining outliers from data streams and mining outliers from trajectories.

With the emerging requirements of mining data streams, several techniques have been proposed to handle the data incrementally [33, 20, 19, 28, 18]. Pokrajac et al. [25] modified the static Local Outlier Factor (LOF) [6] method as an incremental algorithm, and then applied it to find data instance anomalies from the data stream. Takeuchi and Yamanishi [16] trained a probabilistic model with an online discounting learning algorithm, and then use the training model to identify the data instance anomalies. Angiulli and Fassetti [3] proposed a distance-based outlier detection algorithm to find the data instance anomalies over the data stream. However, all the aforementioned works focused on the anomaly detection of a single stream, while our work is designed to discover the contextual collective anomalies over multiple data streams.

A lot of works have been conducted on trajectory outlier detection. One of the representative work on trajectory outlier detection is conducted by Lee et al. [21]. They proposed a partition-and-detection framework to identify the anomaly sub-trajectory via the distance-based measurement. Liang et al. [29] improved the efficiency of Lee's work by only computing the distances among the sub-trajectories in the same grid. As the aforementioned two algorithms require to access the entire dataset, they cannot be adapted to trajectory streams. To address the limitation, Bu et al. [7] proposed a novel framework to detect anomalies over continuous trajectory streams. They built local clusters for trajectories and leveraged efficient pruning strategies as well as indexing to reduce the computational cost. However, their approach identified anomalies based on the local-continuity property of the trajectory, while our method does not make such an assumption. Our approach is close to the work of Ge et al. [13], where they proposed an incremental approach to maintain the top-K evolving trajectories for traffic monitoring. However, their approach mainly focused on the geospatial data instances and ignored the temporal correlations, while our approach explicitly considers the temporal information of the data instances.

7. CONCLUSION

In this paper, we propose a real time anomaly detection framework to identify the contextual collective anomalies from a collection of streams. Our proposed method firstly quantifies the snapshot level anomaly of each stream based on the contextual information. Then the contextual information and the historical information are used in combination to quantify the anomaly severity of each stream. Based on the distribution of the stream anomaly scores, an implicit threshold is dynamically calculated and the alerts are triggered accordingly. To demonstrate the usefulness of the proposed framework, several sets of experiments are conducted to demonstrate its effectiveness and efficiency.

8. ACKNOWLEDGEMENT

The work was supported in part by the National Science Foundation under grants DBI-0850203, HRD-0833093, CNS-1126619, and IIS-1213026, the U.S. Department of Homeland Security under grant Award Number 2010-ST-06200039, Army Research Office under grant number W911NF-10-1-0366 and W911NF-12-1-0431, National Natural Science Foundation of China under grant number 61300053, and an FIU Dissertation Year Fellowship.

9. REFERENCES

[1] A.Arasu, B.Babcock, S.Babu, M.Datar, K.Ito, I.Nizhizawa, J.Rosenstein, and J.Widom. Stream: The stanford stream data manager. In *SIGMOD*, 2003.

[2] J. Agrawal, Y. Diao, D. Gyllstrom, and N. Immerman. Efficient pattern matching over event streams. In *SIGMOD*, 2008.

[3] F. Anguilli and F. Fassetti. Detecting distance-based outliers in streams of data. In *CIKM*, 2007.

[4] Y. Bai, F. Wang, P. Liu, C. Zaniolo, and S. Liu. Rfid data processing with a data stream query language. In *ICDE*, 2007.

[5] M. Blum, R. Floyd, V.Pratt, R.Rivest, and R. Tarjan. Time bounds for selection. *Journal of Computer System Science*, 1973.

[6] M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander. Lof: Identifying density-based local outliers. In *SIGMOD*, 2000.

[7] Y. Bu, L. Chen, A. W.-C. Fu, and D. Liu. Efficient anomaly monitoring over moving object trajectory streams. In *KDD*, 2009.

[8] V. Chandola, A. Banerjee, and V. Kumar. Anomaly detection: A survey. *ACM Computing Surveys*, 2009.

[9] S. Chandrasekaran, O.Cooper, A.Deshpande, M.J.Franklin, J.M.Hellerstein, W.Hong, S.Krishnamurthy, S.R.Madden, V.Raman, F.Reiss, and M.A.Shah. Telegraphcq: Continous dataflow processing for an uncertain world. In *CIDR*, 2003.

[10] D.Carney, U.Cetintemel, M.Cherniack, C. Convey, S.Lee, G.Seidman, M.Stonebraker, N.Tatbul, and S.Zdonik. Monitoring streams: A new class of data management applications. In *VLDB*, 2002.

[11] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. In *OSDI*, 2004.

[12] R. A. Fisher, F. Yates, et al. Statistical tables for biological, agricultural and medical research. *Statistical tables for biological, agricultural and medical research.*, (Ed. 3.), 1949.

[13] Y. Ge, H. Xiong, Z.-H. Zhou, H. Ozdemir, J. Yu, and K. C. Lee. Top-eye: Top-k evolving trajectory outlier detection. In *CIKM*, 2010.

[14] M. Gupta, A. B. Sharma, H. Chen, and G. Jiang. Context-aware time series anomaly detection for complex systems. In *WORKSHOP NOTES*, page 14, 2013.

[15] V. Hodge and J. Austin. A survey of outlier detection methodologies. *Artificial Intelligence Review*, 2004.

[16] J. ichi Takeuchi and K. Yamanishi. A unifying framework for detecting outliers and change points from time series. *IEEE Transactions on Knowledge and Data Engineering*, 2006.

[17] G. Jiang, H. Chen, and K. Yoshihira. Modeling and tracking of transaction flow dynamics for fault detection in complex systems. *Dependable and Secure Computing, IEEE Transactions on*, 3(4):312–326, 2006.

[18] Y. Jiang, C.-S. Perng, T. Li, and R. Chang. Asap: A self-adaptive prediction system for instant cloud resource demand provisioning. In *ICDM*, 2011.

[19] Y. Jiang, C.-S. Perng, T. Li, and R. Chang. Intelligent cloud capacity management. In *NOMS*, 2012.

[20] Y. Jiang, C.-S. Perng, T. Li, and R. Chang. Self-adaptive cloud capacity planning. In *SCC*, 2012.

[21] J.-G. Lee, J. Han, and X. Li. Trajectory outlier detection: A partition-and-detect framework. In *ICDE*, 2008.

[22] B. Lo, S. Thiemjarus, R. king, and G. Yang. Body sensor network-a wireless sensor platform for pervasive healthcare monitoring. In *PERVASIVE*, 2005.

[23] B. Mozafari, K. Zeng, and C. Zaniolo. High-performance complex event processing over xml streams. In *SIGMOD*, 2012.

[24] A. Oliner, A. Ganapathi, and W. Xu. Advances and challenges in log analysis. *Comm. of ACM*, 2012.

[25] D. Pokrajac, A. Lazarevic, and L. J. Latecki. Incremental local outlier detection for data streams. In *CIDM*, 2007.

[26] Y. Song and L. Cao. Graph-based coupled behavior analysis: a case study on detecing collaborative manipulations in stock markets. In *IEEE world congress on computational intelligence*, 2012.

[27] storm. http://storm-project.net.

[28] L. Tang, C. Tang, L. Duan, Y. Jiang, C. Zeng, and J. Zhu. Movstream: An efficient algorithm for monitoring clusters evolving in data streams. In *Grc*, 2008.

[29] L. Tang, C. Tang, Y. Jiang, C. Li, L. Duan, C. Zeng, and K. Xu. Troadgrid: An efficient trajectory outlier detection algorithm with grid-based space division. In *NDBC*, 2008.

[30] L. G. Valiant. A bridging model for parallel computation. *Communications of the ACM*, 33(8):103–111, 1990.

[31] C. Zeng, Y. Jiang, L. Zheng, J. Li, L. Li, H. Li, C. Shen, W. Zhou, T. Li, B. Duan, et al. Fiu-miner: a fast, integrated, and user-friendly system for data mining in distributed environment. In *KDD*, pages 1506–1509. ACM, 2013.

[32] L. Zheng, C. Shen, L. Tang, T. Li, S. Luis, S.-C. Chen, and V. Hristidis. Using data mining techniques to address critical information exchange needs in disaster affected public-private networks. In *KDD*, 2010.

[33] Y. Zhu and D. Shasha. Statstream: Statistical monitoring of thousands of data streams in real time. In *VLDB*, 2002.