Robust and Rapid Adaption for Concept Drift in Software System Anomaly Detection

Minghua Ma [†§], Shenglin Zhang [*‡], Dan Pei [†§], Xin Huang [¶], Hongwei Dai [¶]
[†] *Department of Computer Science and Technology, Tsinghua University,*
[§] *Beijing National Research Center for Information Science and Technology (BNRist),*
[‡] *College of Software, Naikai University* [¶] *Sogou Inc*

***Abstract*—Anomaly detection is critical for web-based software systems. Anecdotal evidence suggests that in these systems, the accuracy of a static anomaly detection method that was previously ensured is bound to degrade over time. It is due to the significant change of data distribution, namely concept drift, which is caused by software change or personal preferences evolving. Even though dozens of anomaly detectors have been proposed over the years in the context of software system, they have not tackled the problem of concept drift. In this paper, we present a framework, StepWise, which can detect concept drift without tuning detection threshold or per-KPI (Key Performance Indicator) model parameters in a large scale KPI streams, take external factors into account to distinguish the concept drift which under operators' expectations, and help any kind of anomaly detection algorithm to handle it rapidly. For the prototype deployed in Sogou, our empirical evaluation shows StepWise improve the average F-score by 206% for many widely-used anomaly detectors over a baseline without any concept drift detection.**

***Keywords*-Anomaly detection; Concept drift; Software service KPI; Web-based software system.**

I. Introduction

With the booming development of web-based software systems like search engine, online shopping and social networks, careful analysis of monitoring data is becoming increasingly important for software reliability. In general, operators of the above systems monitor millions of Key Performance Indicator (KPI, *e.g.* number of Page Views (PV), number of online users, average response time) streams [1]. Diverse types of detectors can be used to detect KPI anomalies, *e.g.*, Moving Average (MA) [2] and Time Series Decomposition (TSD) [3]. In order to accurately detect KPI anomalies, operators carefully tune the parameters of detectors based on their domain knowledge [1], [4]–[6].

Anecdotal evidence indicates that when there is a concept drift, the accuracy of detectors is bound to degrade [7]. In this work, "concept" means KPI stream distribution, and the term "concept drift" means that KPI stream distribution changes significantly [8]. In our scenario, a concept drift is typically a level shift, *e.g.*, a sudden drop in PV as shown in Fig. 1, or a ramp-up/ramp-down, *i.e.*, a deteriorating condition [9]. A concept drift can be either unexpected or expected. An unexpected concept drift is an anomaly situation which is beyond operators' expectations. In order to mitigate the losses imposed by unexpected concept drifts, operators must immediately take actions, *e.g.*, rolling back software changes, stopping traffic shifting. After that, the KPI stream distribution will return to what it was before the concept drift. On the contrary, an expected concept drift yields the KPI distribution change under operators' expectations. In this case, the anomaly detectors designed for the *old concept* before the concept drift are no longer accurate in detecting the KPI anomalies of the *new concept* after the concept drift.

Operators usually conduct software changes, namely software upgrades, on-demand scaling up, migration and configuration changes, in order to deploy new features, fix bugs, or improve system performance [10]. For example, if operators deploy a software service on more servers, the PV recorded by each server will drop significantly because the total PV remains stable. Apparently, this is an expected concept drift: operators expect that the PV on each server will witness a prominent drop in a short time (*e.g.*, within five minutes). Suppose that operators use TSD for anomaly detection [3], and the parameters of TSD are trained based on the old concept (*trend* and the *standard deviation* of noise) before this concept drift. As Fig. 1 shows, since the data distribution has changed dramatically, TSD will generate a lot of false alarms for a long time (the periodicity of KPI stream, say one day) because it uses one period of historical data to predict the upcoming "normal" data. Then, it will gradually catch up with the trend of the new concept, but still cannot adapt to the standard deviation of the noise in the new concept. That is to say, TSD "believes" that most of the KPI data points in the new concept are anomalous, while operators consider these points as normal. In this work, we try to *adapt anomaly detection methods to avoid the long period of accuracy loss after expected concept drifts*.

There are several interesting challenges:

1. High frequency of expected concept drifts. In web-based software systems, software upgrades and configuration changes occur thousands of times every day [9], which leads to a great number of concept drifts in KPI streams. For example, in the studied company Sogou, a top-tier search

* Shenglin Zhang is the corresponding author.

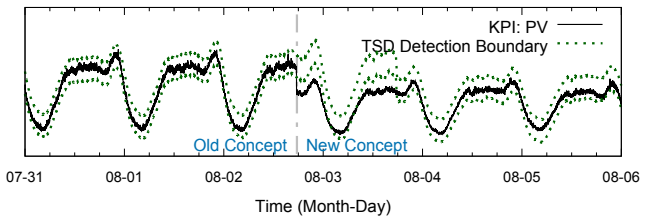


Figure 1: A toy example of concept drift. KPI data beyond the detection boundary will trigger alarm.

engine in China, there are *more than 3000* concept drifts of KPI streams *per day*, *more than 80%* of which are expected ones. Manually tuning the parameters of anomaly detectors to adapt to such a large number of expected concept drifts is infeasible, thus an *automatic* concept drift adaption approach should be provided.

2. Huge amount of KPI data. In large web-based software systems, tens of software systems are deployed on tens of thousands of servers [9]. Degradation of a software service KPI on (recorded by) a specific server may indicate that the service provided by this server is affected. As a result, anomaly detectors and their concept drift adaption (detection) approach must monitor the service KPIs on every server. In addition, typically tens of types of KPIs are monitored for a single software system. That is, the concept drift adaption approach should continuously detect concept drifts on millions of KPI streams. It will take operators a lot of time if the parameters of the concept drift adaption (detection) approach have to be tuned *manually for each KPI stream on each server*.

3. Diverse types of detectors. Since no anomaly detector is perfectly suitable for all types of KPI streams, it is a common practice that each KPI stream is assigned to a specialized anomaly detector or even a specialized combination of multiple detectors [1]. Therefore, the concept drift adaption approach should be generic so that it can accommodate to various types of detectors.

4. Rapid adaption. A burst of false alarms usually appear soon after a concept drift, because the anomaly detector (or the combination of detectors) trained before the concept drift cannot quickly adapt to the new concept. Therefore, the concept drift adaption approach must detect concept drifts and help anomaly detectors adapt it in a rapid manner.

To tackle the above challenges, we propose a concept drift adaption framework for software KPI anomaly detection, called **StepWise**, to robustly and rapidly adapt anomaly detectors in order to accurately detect KPI anomalies after concept drift. Based on the KPI streams from Sogou, we made the following two main observations. First, in general system, KPI streams are strongly seasonal (a periodicity can be by the hour/day/week/month) because it is heavily affected by the periodic user behavior. Second, distributions before and after a concept drift are statistically correlated, namely, having a significant linear correlation.

Based on the above observations, StepWise has three components to adapt anomaly detectors in a robust and rapid manner. (1) **detection**: StepWise adopts an **iSST-EVT** method to detect concept drift for a large number of KPIs, without tuning detection threshold or model parameters per KPI stream. The Extreme Value Theory (EVT) is used for the first time in the literature to set threshold for change score (the output of concept drift detection model) automatically, to the best of our knowledge. All model parameters can use model-wide empirical value which is shown to be robust in practice. (2) **classification**: StepWise takes into account external factors, namely software changes and traffic shifting, to determine expected concept drifts from unexpected ones (with operators' confirmations). (3) **adaption**: taking advantage of the significant statistical correlation between the old concept and the new one, StepWise applies a novel **concept drift adaption algorithm** to automatically adapt anomaly detectors to accurately detect anomalies after concept drift (in terms of both trend and the width of detection boundaries). The only work of operators is to determine whether the concept drift is expected or unexpected.

Our contributions are summarized as follows:

- To the best of our knowledge, there are no related works on concept drift adaption methods for software anomaly detection in the literature. This paper is the first one to identify the problem of robustly and rapidly adapting anomaly detectors to the new concept after concept drift, and its research challenges in terms of scalability, robustness, and adaption delay.
- We implement a prototype of StepWise which addresses the above challenges and integrate it in the operation platform of Sogou. StepWise adopts a concept drift detection method that does not require parameter tuning or threshold for change score, tells apart expected concept drift from unexpected ones using external factors, and employs a novel concept drift adaption method based on the statistical correlation between the old concept and the new one.
- To demonstrate the performance of StepWise, we have conducted extensive evaluation experiments using the KPI data collected from hundreds of servers over six months from Sogou. The results show that StepWise can rapidly and robustly improve the performance of most anomaly detectors by helping them rapidly adapt to new concepts. Our evaluation shows that StepWise improves the average F-score by 206% for many widely-used anomaly detectors over a baseline without any concept drift detection.

II. Background

In this section, we provide a brief introduction to software service KPI anomaly, concept drift, and anomaly detection.

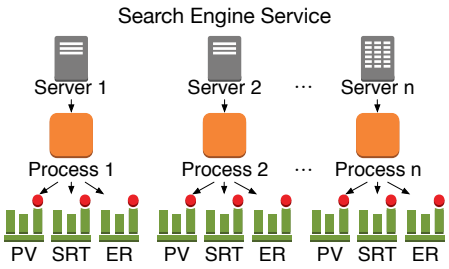


Figure 2: A toy example of how software service KPI measurements are collected.

Software Service KPI: In this work, we collect KPI streams from Sogou. Sogou is the second largest search engine and the fourth largest Internet company of China, with more than 511 million monthly active users and more than 1 billion queries per day (in September 2017). In Sogou, there are tens of thousands of servers providing various types of software systems. Each software service runs on one or more servers with a specific process on each server. Operators deploy an agent on each server to continuously monitor the status of each process and collect the KPI measurements of all processes, including PV, search response time (SRT), error rate (ER), *etc*. For example, immediately after a process serves a user's search request, the PV is incremented and an SRT is recorded. Figure 2 shows a toy example of how a software service KPI is collected. Suppose that one of the search engine services is deployed on server 1 to server n, and each server runs a process (process 1 to process n) of the system. The monitoring agent on each server collects KPI (PV, SRT, ER) measurements of process 1 to process n.

After collecting KPI measurements of processes, the agent on each server delivers the measurements to a distributed Apache Hbase database. The database also provides a subscription tool for other systems [11]–[13], to periodically receive subscribed measurements. A data collection interval is typically one minute. Within one second, the measurements subscribed by StepWise are pushed to StepWise. Typically, the KPI measurements of a process constitute a KPI stream, with a format of $(time, value)$ for each time bin.

In this work, we focus on software service KPIs instead of resource KPIs, *e.g.*, CPU utilization, memory utilization. For a resource KPI, operators usually care about whether it reaches the threshold indicating the upper bound of physical ability. Instead, they do not care about concept drifts in resource KPIs.

KPI Anomaly: An anomaly in the KPI stream is typically a sudden spike, a jitter, or a dip, *etc.*, which indicates an unexpected pattern of KPI. It is usually a short-term or spasmodic change, but it can also be an unexpected level shift or ramp up/down. In general, operators deploy anomaly detectors which are based on their domain knowledge to identify KPI anomalies [1]. Nevertheless, this knowledge is difficult to be precisely defined by some pre-defined rules and needs many rounds of time-consuming iterations [3].

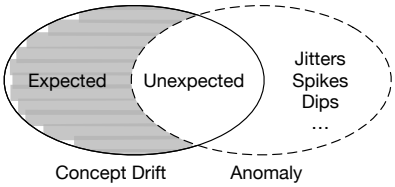


Figure 3: Relationship between concept drift and anomaly.

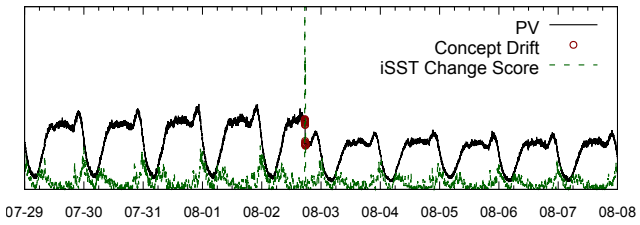
Concept drift: The data distribution can change over time due to dynamically changing and non-stationary environments [8], leading to the phenomenon of concept drift. A concept drift in a KPI stream is typically an immediate level-shift, a ramp up or a ramp down [9], which can be caused by a software upgrade, a configuration change (*e.g.* shifting traffic from one data center to another one), seasonality (*e.g.*, Christmas or Lunar New Year), a network breakdown, a malicious attack, a flash event (*e.g.*, breaking news), or a change in a peer company, *etc.* As Fig. 1 shows, there is a traffic shifting at 17:40 on August 2nd. The the distribution of PV changed significantly within 5 minutes and remained unchanged for over three days.

As shown in the Venn diagram in Fig. 3, there are expected concepts drift and unexpected ones. An unexpected concept drift in a KPI stream is the one beyond operators' expectation. In order to mitigate loss imposed by an unexpected concept drift, operators must immediately take actions, *e.g.*, rolling back the software change, stopping traffic shifting. After that, the distribution of the KPI stream will return to what it was before the concept drift. On the other hand, an expected concept drift is the one under operators' expectation. When it occurs, anomaly detectors usually cannot adapt to the new concept rapidly, generating a lot of false alarms or false negatives (see §V for details). Consequently, it is very necessary to adapt anomaly detectors immediately after a concept drift.

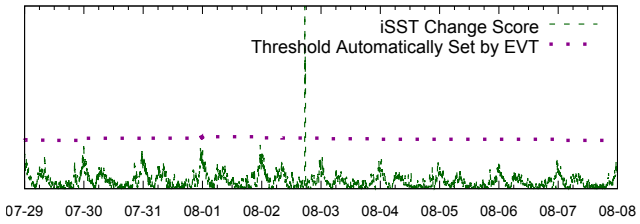
Fig. 3 shows the similarities and differences between concept drift and KPI anomaly. Both concept drift and KPI anomaly include unexpected concept drift. However, an expected concept drift is an expected pattern of KPI stream, while an anomaly is an unexpected one.

Anomaly detection algorithm: Recently, an increasing number of studies have paid attention to anomaly detection algorithms. An anomaly detection algorithm is a single anomaly detector, *e.g.*, Static threshold [14], TSD [3], Diff [1], MA [2], Weighted MA [15], EWMA [15], Holt-Winters [16], or a combination of two or more detectors, *e.g.*, Opprentice [1]. In Sogou, operators have deployed the above seven detectors to detect anomalies in KPI streams.

The basic schema of an anomaly detector works in the following way. When an anomaly detector receives an incoming KPI value, it internally produces a forecast value.



(a) Time (Month-Day)



(b) Time (Month-Day)

Figure 4: A concept drift in original KPI stream (marked by red circles) is successfully detected by iSST-EVT because the corresponding iSST change score is higher than the threshold automatically set by EVT (§IV-A).

If the absolute difference between the incoming value and the forecast one surpass a given threshold, an anomaly alert will be triggered. The threshold is manually set by operators with domain knowledge. For example, the threshold of TSD is set by operators based on the underlying distribution of noises. However, if a concept drift occurs, the distribution of noises will change dramatically. Therefore, the trained TSD is no longer accurate in detecting anomalies after the concept drift. In addition, most detectors are parameterized and have a set of internal parameters. For example, Holt-Winters has three parameters: α, β, γ. These parameters are also set based on the distribution before a concept drift, and thus should be re-trained after it.

III. Intuition behind StepWise

This section presents the domain-specific insights we use to help address the challenges mentioned in §I. We need to detect concept drifts in KPI streams in order to rapidly adapt to the expected concept drifts. The first insight is that concept drift detection can be converted into a spike detection problem in the change score stream. After an expected concept drift, the performance of anomaly detectors degrades. It is difficult to tune parameters and the severity threshold for the anomaly detectors manually. Fortunately, we leverage a second insight that distributions before and after concept drift are statistically correlated. We conclude this section by concluding the two challenges in applying these insights in a practical system.

A. Concept Drift Detection Algorithm

There are many previous works on time series concept drift detection (also known as change point detection), but their models need either parameter tuning or the change score threshold tuning *for each KPI stream*, which are not affordable in our scenario due to the large number and variety of KPI streams. Therefore, *our goal for concept drift detection algorithm is the one that does not require per-KPI parameter tuning or change score threshold tuning*.

Previous studies on time series concept drift detection typically slide two adjacent (but non-overlapping) windows along time dimension on the KPI stream and compute the absolute or relative difference between the behavior of time-series in these two windows [6], [17]–[20]. The output of these detection algorithms is the *change score* (indicating the severity level of the change) on each time interval rather than concept drift directly. Most of these models need either parameter tuning or change score threshold tuning for each KPI stream. On the one hand, the internal parameters in some models (*e.g.*, CUSUM [18]), kernel-based model [20], SST [21], EGADS [6]) need to be tuned on a per-KPI basis based on operators' domain knowledge. On the other hand, the change score detection thresholds for some models (*e.g.*, CUSUM [18], Multiscale Robust Local Subspace (MRLS) [19], improved Singular Spectrum Transform (iSST) [9]) need to be tuned on a per-KPI basis so that the change is severe enough to the operators. However, neither tuning model parameters nor tuning detection thresholds on a per-KPI basis is affordable in our scenario due to the large number and variety of KPIs.

After studying all these methods, our first insight is that the spike of change score stream indicates the time of concept drift. We can summarize the intuition as follows:

Insight 1: ***Concept drift detection can be converted into a spike detection problem in the change score stream.***

For instance, we can use the iSST [9] algorithm to calculate the change score, as Fig. 4 (a) shows. There is a spike in the iSST change score stream on 2 $\sim$ 3 August, at the time of concept drift. iSST [9] is a recent concept drift detection approach which is robust and rapid in that it can use model-wide empirical values for model parameters, thus does not need model parameter tuning on a per-KPI basis. However, for different KPI streams, the threshold of change score may be different, which is one remaining challenge. Inspired by Insight 1, we design a spike detection approach in §IV-A which do not need to tune per-KPI change score threshold.

B. Measurement Study

For the KPI stream from web-based software systems, we discover that concept drift, if happening, almost always corresponds to concept drift caused by software changes. According to our statistics of concept drifts from Sogou, 95% of concept drifts are caused by software changes (62% by traffic shifting and 33% by code update), and the other

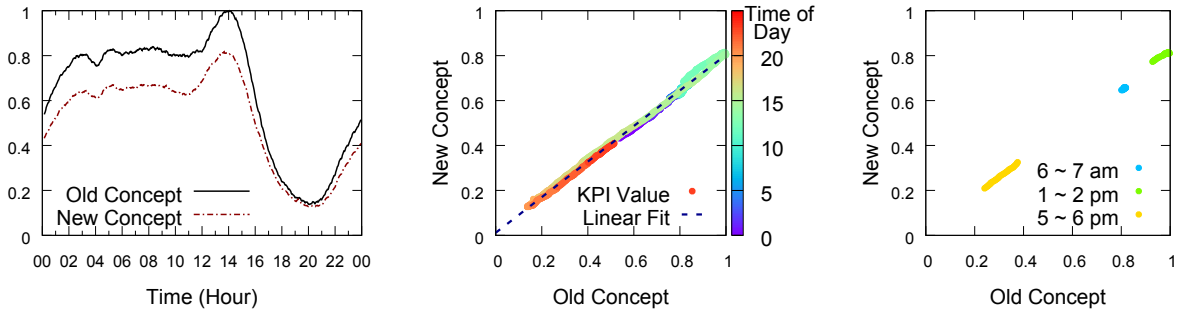


Figure 5: (a) The comparison of median value for each daily time slot in new concept and old concept. (b) Each data point (old-concept, new-concept) is from the two Y-values of the same time of day in (a), and the color of the point indicates the time of day (X-value in (a)). The dotted line is the linear regression. (c) Three one-hour intervals picked from (b).

5% are caused by holiday impact. We believe this is not rare in Web-based software systems.

The data distribution before a concept drift is considered as **old concept**, while the data distribution after is considered as **new concept**. Fig. 4 shows an example where new concept *looks* similar to old concept. Fig. 5 (a) provides a more intuitive look. We take 10 days of old concept and new concept data, and then calculate the median value for each daily time slot (*e.g.*, 20:00, 20:01...). The median value can eliminate impact of the potential anomalous value. The comparison in Fig. 5 (a) shows that in this example the new concept is a "shrunk version" of the old one. Fig. 5 (b) shows a more quantitative comparison, where each data point (old-concept, new-concept) is from two Y-values of the same X-value in Fig. 5 (a), and the color of the point indicates the time of day.

Insight 2: ***The relationship between new concept and old concept can be almost perfectly fitted by linear regression.***

Note that this KPI with concept drift is randomly picked among 288 KPIs, we observe that new concept and old concept consistently have a linear relationship with little variance across all 282 expected concept drifts recorded in our dataset. This suggests that the liner model is a promising direction for concept drift detection in that we can "predict" the new concept with the old concept given the linear relationship.

C. Practical Challenges

There are two issues in the practical system design.

- How to detect spikes in the change score stream? This is challenging because the amplitude of spikes vary both across different KPIs and over time [9], and it is infeasible to manually configure the threshold of change score for each KPI stream.
- How to estimate the parameter of the linear model in real-time? More specifically, we observe that the points of continuous time are clustered in Fig. 5 (b) and this phenomenon is more clearly shown in Fig. 5 (c). If concept drift happens at 6 $\sim$ 7 am (see Fig. 5 (c)), there are too few data points to do the linear fitting, thus we cannot estimate the relationship between the old concept and the new concept in real-time (within one hour). As aforementioned, concept drift is always caused by sporadic software changes, which may happen at any time. This phenomenon brings a great challenge to design the adaption algorithm.

IV. StepWise Detailed Design

We propose a practical framework, StepWise, for robust and rapid adaption for concept drift in software anomaly detection. As Fig. 6 shows, our framework has three main components: **detection**, **classification**, and **adaption**. First, in §IV-A we propose a concept drift detection method that does not require tuning model parameters or tuning change score threshold. Second, in §IV-B we classify concept drifts into *expected* vs. *unexpected* ones. We apply causal impact analysis to make this classification semi-automatic. Third, in §IV-C, we propose a robust and rapid adaption algorithm. Our main contributions focus on the detection and adaption components.

A. Concept Drift Detection without Tuning Model Parameters or Detection Thresholds

Recall that the first practical challenge is to detect spikes in the change score stream without tuning per-KPI threshold. We design an Improved Singular Spectrum Transform without Thresholds (iSST-EVT) algorithm which take advantage of both iSST and Extreme Value Theory.

1) Improved Singular Spectrum Transform without Thresholds (iSST-EVT): As aforementioned in §III-A, we use iSST to calculate the change score on each time

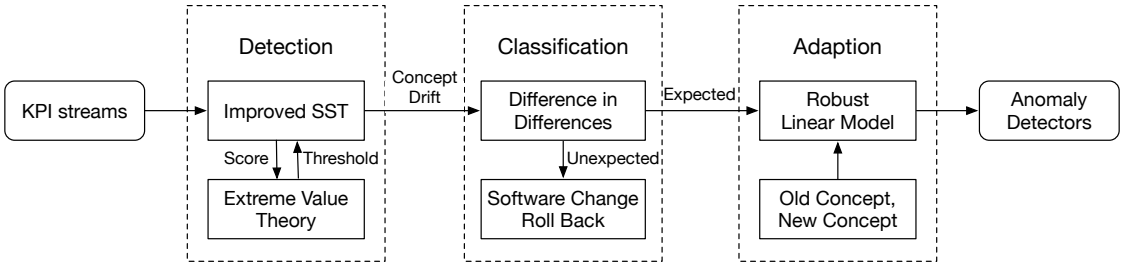


Figure 6: Framework of StepWise.

interval. Our core idea is to apply EVT to the change score stream because EVT is a powerful spike detection method.

We provide a brief description of the theoretical background of EVT. Extreme Value Theory (EVT) [22] can set the threshold of data stream in an automatic fashion *without making any assumption about the data distribution*. The goal of EVT is to find the law of extreme value. These extreme value (spikes) have the same kind of distribution, regardless of the original one. According to experiments from [7], the maximums of KPI streams from several fields (network, physics, finance) have almost the same distribution although the distributions of the KPI streams are not likely to be the same.

For a random variable X, F is cumulative distribution function: $F(x) = P(X \leq x)$. We denote $\bar{F}(x)$ as the "tail" of this distribution. This extreme value distribution has the following form:

$$G_\gamma : x \mapsto exp(-(1+\gamma x)^{-\frac{1}{\gamma}}), \gamma \in \mathbb{R}, 1+\gamma x > 0.$$

However, γ is the extreme value index which is hard to estimate efficiently in this original formula. To make EVT more practical, Pickands-Balkema-De Haan theorem [23], [24] (also called *second theorem* in EVT) is proposed as follows:

Theorem IV.1 (Pickands-Balkema-De Haan)**.** *F converges in distribution G_γ, if and only if a function $\sigma(t)$ exists, for all $x \in \mathbb{R}$ s.t. $1+\gamma x > 0$:*

$$\bar{F}_t(x) = P(X - t > x | X > t) \sim (1 + \frac{\gamma x}{\sigma(t)})^{-\frac{1}{\gamma}}.$$

This theorem shows that the excess over a threshold t, denoted as $X - t$, follows G_γ with parameters σ, γ. t is the initial threshold, whose value is not paramount except that it must be "high" enough.

As Fig. 4 (b) shows, we apply the Pickands-Balkema-De Haan theorem to detect the spikes in the *change score stream output by iSST* (instead of the original KPI stream), an extreme value of which can indicate a concept drift in the original KPI stream. Based on domain experience and [7], we suggests that the model-wide empirical parameter t is set to a high empirical percentile (say 98%). Then σ and γ can be estimated by Maximum Likelihood Estimation, and these model-wide empirical values for EVT parameters are shown to work well in detecting spikes, which are the right time of concept drifts.

In this way, the threshold for concept drift detection is automatically set by EVT. *To the best of our knowledge, this paper is the first work to apply EVT to change score stream to automatically set the detection threshold for base concept drift detection including but not limited to iSST.*

Our proposed new concept drift detection approach, called **iSST-EVT**, has the following desirable properties: *no detection threshold to tune, no per-KPI-stream model parameters to tune, and all model parameters can use model-wide empirical values which are shown to be robust in practice.*

2) Model-wide empirical parameter values in iSST-EVT: Although both EVT and iSTT still have model parameters, these parameters have model-wide empirical values that can be applied to all KPI streams, without having to be tuned per KPI stream. The model parameter t for EVT is set to a high empirical percentile (98%) [7] and then σ and γ can be estimated by the maximum likelihood estimation. For iSST, there is only one parameter to be set. According to [9], for a system that needs quick mitigation on concept drift, window size ω can be set to a small value such as 5. For a system that needs more precise assessment of concept drift, ω can be set to a large value such as 15. Empirically we set the window size of 6 (minutes) in StepWise to declare a change in KPI stream as a concept drift.

B. Concept Drift Classification

After a concept drift is detected, we need to classify whether concept drift is expected or unexpected. For external events (*e.g.*, sales promotion) that cause the expected concept drift, there might not be reliable data feeds, but when we do have reliable event data feeds (*e.g.*, software upgrade, configuration change, Christmas or Lunar New Year holidays), we can apply causal impact analysis [25] to conduct semi-automatic classification. Causal impact analysis [25] is the process of drawing a conclusion about a causal connection based on conditions of the occurrence of an effect. In this paper, we adopt the methodology similar to that used in FUNNEL [9], namely Difference in Differences (DiD) [26], to *automatically* analyze whether a concept drift is caused by a software change (*e.g.*, code deployment, configuration change, traffic shifting) whose event logs are available as

data feeds. If the causal analysis result indicates that a concept drift is indeed caused by the software change, the final classification still has to be done manually by the operators since some software changes are not supposed to cause concept drift but might have done due to bugs or misconfiguration. If unexpected, the software change will be rolled back. Otherwise, the concept drift adaption will be triggered because of the expected concept drift. Given only the final confirmation is done manually, we call this classification semi-automatic. For adaption to concept drift, which we mentioned later, is only the expected one.

C. Concept Drift Adaption

Anomaly detection is vital for managing the service performance of web-based software systems. The best parameters and thresholds of a given anomaly detector often highly depend on actual data in a given KPI, thus they are very time-consuming to be tuned. In practice, operators take a long time to gradually do so. However, after the expected concept drift, the performance of anomaly detectors significantly degrade sharply. It is infeasible for the operators to manually tune the parameters and thresholds again *in a very timely manner*.

From the measurements in Fig. 5, we learn that it is reasonable to model the relationship between the old concept and the new one with linear regression. A simple and effective idea is to use linear transformation to change the magnitude of new concept to be the same as old concept's. This enables anomaly detectors to be still useful in new concept, even though there are not many data points available yet in the new concept *alone*.

Note that we choose not to do it the other way around (*i.e.*, *transforming old concept to new concept*) for the following reasons. Our proposed method changes the input of anomaly detectors, rather than changes any part of anomaly detectors which are introduced in §V-B. Should we choose to transform the old concept instead, the parameters and thresholds of anomaly detectors should be re-trained. As more points become available in the new concept, the distribution becomes more stable. Thus we have to keep doing re-training, which is not only difficult but also time-consuming. In addition, if there are real anomalies in the new concept and they are used as the basis to train the anomaly detector, the transformation becomes inaccurate.

We design a robust and rapid adaption algorithm. The input of the algorithm needs a concept drift time c to separate the KPI stream as K_{old} and K_{new}. Note that K_{new} changes as time goes by with t as the timestamp of the latest data points. Other inputs are an empirical value of iSST window size ω, and a KPI periodicity T which is used as the step-by-step adaption terminate time. The algorithm can be summarized as the following four steps.

(1) The window size of iSST ω gives the first part of data list, denoted by A (in green color). As Fig. 7 shows, n samples within the same time range as A are extracted from the K_{old}. Since the KPI values vary and there are even some anomalies in the past, we use the median value for every time bin of these samples which is a robust estimation [27], denoted as B (in purple color).

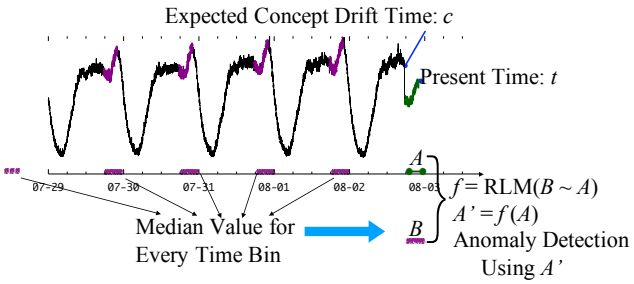


Figure 7: The core idea of the adaption algorithm. X-axis is Time (Month-Day).

(2) A and B are fitted by Robust Linear Model (RLM) [28] with the output of the linear function f. There are many linear regression methods, like LSE (least squares regression). However, LSE is highly sensitive to (not robust against) anomalies. This is because a squared error penalizes deviations quadratically, so points far from the line have more effect on the fitting than the points close to it. We want to get rid of the impact of anomaly data points by using the RLM. The way to achieve robustness against anomalies is to replace the Gaussian distribution for the response variable with a distribution that has heavy tails, which will assign higher likelihood to anomalies, without having to perturb the straight line to "explain" them. More details of the RLM is out of the scope of this paper.

(3) As soon as we get the linear model, we start a linear transformation to the current point $A'[t]$, which is fed into anomaly detectors.

(4) However, the RLM might not be exactly accurate when there are not sufficient data points in the new concept. For example, in Fig. 5 (c), the color in the continuous time slot is constant and clustered, and the slopes of each of the three hours are slightly different from each other and also different from the global slope in Fig. 5 (b). Thus we keep re-fitting the linear relationship (by repeating steps 1, 2) as more new data points in the new concept become available. Fig. 8 shows how the average anomaly detection $F\text{-}score_a$ (defined in §V-B) of StepWise gets higher and higher as more and more data points after the concept drift become available. This re-fitting will continue until the adaption termination time T after concept drift. T is set empirically to be one day since Fig. 5(b) indicates that one day is sufficient to have a very good linear fitting. From Fig. 8 we can see that a good linear fitting can be achieved before one day (*e.g.*, the average $F\text{-}score_a$ gets to 0.60 by 6 hours after the concept drift).

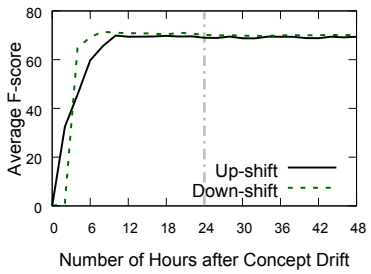


Figure 8: The average $F\text{-}score_a$ of anomaly detectors (for the new concept) as time goes by after concept drifts.

Table I: Summary of datasets.

Datasets		Up-shift	Down-shift
# KPIs		288	
# Unexpected concept drifts		14	38
# Expected concept drifts		107	175
# Total data points		175680	110880
Anomaly data points (ratio to the total)		3316 (1.89%)	3808 (3.43%)
Old concepts	Total data points	87485	55775
	Anomaly data points	1810 (2.07%)	1804 (3.23%)
New concepts	Total points	88195	55105
	Anomaly data points	1506 (1.71%)	2004 (3.64%)

V. Evaluation

StepWise is designed to rapidly and robustly adapt to the expected concept drift for anomaly detection algorithms. We deployed a prototype of StepWise in Sogou, a search engine company. In this section, we evaluate the performance of StepWise. We first evaluate the overall performance of StepWise in terms of how it robustly improves the accuracy of anomaly detectors, as shown in §V-B. Then we compare our concept drift detection method with FUNNEL [29] in §V-C. We show the adaption lag of StepWise at the end of this Section.

A. Datasets

Cooperating with operators, we have deployed a prototype of StepWise in Sogou. To evaluate the performance of StepWise, we have collected a large volume of KPI data. The dataset consists of 288 KPI streams in a six-month period with a one-minute sampling interval, which is randomly picked from the search engine systems by operators, as Table I shows. In addition, we classify concept drifts into up-shift (Fig. 10 (a)) and down-shift (Fig. 10 (b)), for the reason that an anomaly detector performs differently during up-shift and down-shift. We list the number of total data points and anomaly data points labeled by operators. Although KPI anomaly occurs in a very low frequency, the six-month period is long enough to cover almost all types of anomalies, and this is sufficient for evaluating the performance of StepWise.

Note that all anomalies and all concept drifts in our dataset are manually verified by operators, which can be served as the ground truth of our evaluation experiments. We plan to release the dataset to the public. Although the KPI data is collected from the search engine systems, we believe that they are representative of the KPI data of most types of web-based software systems, *e.g.,* social media, online shopping. As for future work, we will conduct more evaluation experiments with data collected from other types of software systems.

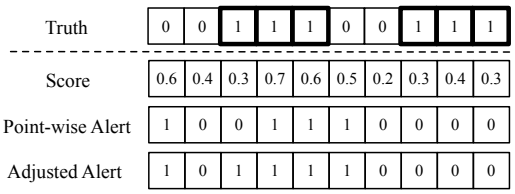


Figure 9: A toy example of how adjusted alerts of KPI anomaly are generated. The first row is the truth with 10 contiguous points and two anomaly segments highlighted in the shaded squares. Detector scores are shown in the second row. The third row shows point-wise detector results with a threshold of 0.5. The fourth row shows the detection results after adjustment. We shall get precision 0.6, and recall 0.5.

Table II: Basic detectors and sampled parameters. Some abbreviations are Diff (Differences), TSD (Time Series Decomposition), MA (Moving Average), EWMA (Exponentially Weighted MA), TSD (Time Series Decomposition).

Detector	Sampled parameters
Static threshold [14]	none
TSD [3]	period $= 1$ day
Diff [1]	period $= 1$ week
MA [2]	window $= 30$ points
Weighted MA [15]	window $= 30$ points
EWMA [15]	$\alpha = 0.1$
Holt-Winters [16]	$\alpha = 0.5, \beta = 0.1, \gamma = 0.3$

B. Evaluation of The Overall Performance

In this section, we try to evaluate how StepWise can robustly improve the accuracy of anomaly detectors without tuning parameters for each detector.

An anomaly detector's capability to detect anomaly is usually assessed by three intuitive metrics, *i.e.*, precision, recall, F-score. Following [30], we apply the improved anomaly detection metrics, because in general operators do not care about the point-wise alerts. Instead, they prefer to trigger an alert for any time interval in a contiguous anomalous segment. Fig. 9 shows a toy example of how adjusted alerts of KPI anomalies are generated. If any point in an anomaly segment in the ground truth can be detected (over a given threshold), we say this segment is detected correctly, and all points in this segment are treated as they can be detected. Comparing the *truth points* (the first row in Fig. 9) and the *adjusted alert points* (the fourth row in Fig. 9), we label an anomaly detector's outcome as

Table III: The $Precision_a$, $Recall_a$, and $F\text{-}score_a$ of anomaly detectors in the **Old** concept, in the **New** concept without StepWise, and in the new concept with **StepWise**. Improvement rate (↑) is calculated by (**StepWise** - **New**) / **New**.

Drift type	Up-shift									Down-shift								
Metrics	$Precision_a$ %			$Recall_a$ %			$F\text{-}score_a$ %			$Precision_a$ %			$Recall_a$ %			$F\text{-}score_a$ %		
Algorithms	Old	New	Step-Wise (↑)	Old	New	Step-Wise (↑)	Old	New	Step-Wise (↑)	Old	New	Step-Wise (↑)	Old	New	Step-Wise (↑)	Old	New	Step-Wise (↑)
Column No.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
Threshold	97	2	38 (1800)	41	100	64 (-36)	57	5	48 (860)	44	0	38 (-)	29	0	67 (-)	35	0	48 (-)
TSD	73	26	75 (188)	40	100	84 (-16)	52	41	79 (93)	75	67	46 (-31)	88	50	98 (96)	81	57	63 (11)
Diff	31	8	24 (200)	52	98	83 (-15)	38	16	37 (131)	71	7	64 (814)	72	31	72 (132)	72	12	68 (467)
MA	49	12	50 (316)	65	100	93 (-7)	56	22	65 (195)	63	90	61 (-32)	83	19	91 (379)	72	31	73 (135)
WMA	91	21	83 (295)	65	100	93 (-7)	76	35	87 (149)	94	92	89 (-3)	81	19	89 (368)	87	31	89 (187)
EWMA	94	24	86 (258)	66	100	93 (-7)	78	38	89 (134)	97	94	92 (-2)	81	19	84 (342)	88	31	88 (184)
Holt-Winters	87	21	63 (200)	90	100	93 (-7)	88	36	75 (108)	74	97	55 (-43)	100	40	97 (143)	85	57	70 (23)

a true positive (TP_a), true negative (TN_a), false positive (FP_a), and false negative (FN_a). We calculated precision, recall, and F-score as follows: $precision_a = \frac{TP_a}{TP_a+FP_a}$, $recall_a = \frac{TP_a}{TP_a+FN_a}$, $F\text{-}Score_a = \frac{2*precision_a*recall_a}{precision_a+recall_a}$.

1) Choices of Detectors and Parameters: In the studied scenario, operators have deployed seven anomaly detectors for the 288 KPI streams, as shown in Table II. All the seven anomaly detectors are widely-used in web-based software systems beyond the search engine [1]. Note that StepWise is not limited to the seven anomaly detectors and instead it can be applied to all anomaly detectors. In this paper, our objective is to adapt to expected concept drifts for *all* anomaly detectors, instead of improving the accuracy of *a specific* anomaly detector by tuning its parameters. We believe the set of the seven anomaly detectors is general enough to represent other kinds of anomaly detectors.

For each KPI stream, operators have previously configured the parameters of its anomaly detector *best for F-score*. Without StepWise, the parameters of anomaly detectors remain unchanged after the concept drift, resulting in a lot of false alarms.

2) Evaluation Results and Analysis: As aforementioned, although the parameters of anomaly detectors are carefully tailored by operators before each concept drift, they are no longer suitable for the new concept after the concept drift, because the data distribution has greatly changed. As Table III shows, without concept drift adaption, the $F\text{-}score_a$ of each anomaly detector degrades dramatically after the concept drift, whether it is an up-shit or a down-shift (see the 7th, 8th, 16th and 17th columns in the table).

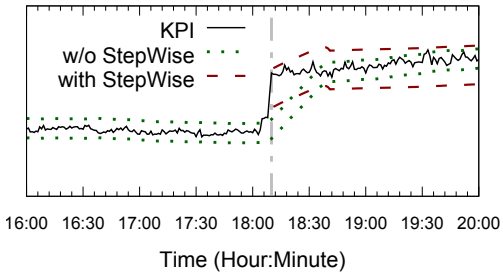
StepWise improves the $F\text{-}score_a$s of all anomaly detectors in the new concept by 206% on average (see the 8th, 9th, 17th and 18th columns in Table III), and makes them comparable with those of anomaly detectors in the old concept. The results are quite satisfactory to operators based on an on-site investigation.

More specifically, with StepWise the precisions of anomaly detectors increase dramatically after up-shifts (see the 2st, 3nd columns in Table III). However, without StepWise the recalls of anomaly detectors witness increases after up-shifts (see the 4th and 5th columns in Table III), which are higher than those of anomaly detectors with StepWise (see the 5th and 6th columns in Table III). On the contrary, with StepWise the recalls of anomaly detectors rise greatly after down-shifts (see the 14th, 15th columns in Table III). In addition, StepWise decreases the precisions of most anomaly detectors (except for Threshold and Diff) after down-shifts (see the 11th and 12th columns in Table III).

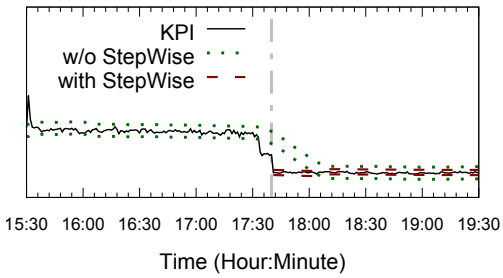
Table IV: The number of false positives ($\#FP_a$) and false negatives ($\#FN_a$) in the **New** concept without StepWise and in the new concept with **StepWise**, respectively.

Drift type	Up-shift				Down-shift			
False type	$\#FP_a$		$\#FN_a$		$\#FP_a$		$\#FN_a$	
	New	Step-Wise	New	Step-Wise	New	Step-Wise	New	Step-Wise
Column No.	1	2	3	4	5	6	7	8
Threshold	52678	1645	0	535	0	2519	2004	661
TSD	4144	418	0	230	414	2294	998	28
Diff	15308	3910	16	245	7847	1338	1371	544
MA	10360	1418	0	104	19	1202	1622	166
WMA	5514	282	0	104	13	212	1622	219
EWMA	4805	233	0	104	9	136	1622	312
Holt-Winters	5507	852	0	104	23	1599	1195	55

To explain the above results, for each anomaly detector (after the concept drift), in Table IV we list the number of false positives and false negatives, without StepWise and with StepWise, respectively. As shown in the 1st column of Table IV, after up-shifts each anomaly detector generates a large number of false positives when without executed StepWise. In this way, most anomalies are recalled. That is to say, after up-shifts the anomaly detectors are so sensitive that they successfully detect almost all anomalies (see the 3th column of Table IV), which results in very high recall ratios, *at the cost of a large number of false alarms*. For example, as shown in Fig. 10 (a), the MA method which is without the adaption of StepWise remains the same sensitivity (in terms of the width of detection boundaries)



(a) MA, Up-shift



(b) MA, Down-shift

Figure 10: The detection boundaries of MA during up-shift and down shift, with and without StepWise, respectively.

after the up-drift. As a result, the method determines a lot of normal KPI data points as anomalies. However, with the adaption of StepWise, MA can quickly catch up with the "correct" detection boundaries (§V-D). Similarly, we can explain why StepWise decreases the precisions of anomaly detectors after down-shifts, with the data listed in the 5th, 6th, 7th and 8th columns of Table IV and the example shown in Fig. 10 (b).

C. Evaluation of Concept Drift Detection

As described in §IV-A, we propose a concept drift detection method, iSST-EVT, which does not require parameter tuning or threshold of change score. To demonstrate the performance of iSST-EVT, we compare it with iSST in FUNNEL [29], which has been deployed and proved to be efficient to detect concept drifts during software upgrades and configuration changes in large web-based software systems but need to tune threshold per-KPI-stream.

Similarly to the evaluation of overall system in §V-B, we also use precision, recall and F-score which are intuitively interpretative to evaluate the performance of iSST-EVT and iSST. Note that concept drift detection is different from anomaly detection, and thus we re-define precision, recall and F-score for concept drift detection. Specifically, a concept drift detection algorithm is a binary classifier, which gives *yes* or *no* decisions. We use true positive (TP_c), true negative (TN_c), false positive (FP_c), and false negative (FN_c) to label the outcome of an algorithm. Based on ground truth provided by the operators, the true positive is the concept drift confirmed by operators and detected by the algorithm. If the concept drift is labeled by operators while the algorithm neglects it, we label the item as a false negative. The false positive is the concept drift detected by the algorithm but it is not a concept drift labeled by operators. In the true negative, both operators and algorithm decide that the item is not a concept drift. We calculate $Precision_c$, $Recall_c$, and $F\text{-}score_c$ in the same way as $Precision_a$, $Recall_a$, and $F\text{-}score_a$ of anomaly detection, because they just have slightly different definitions on counting true positive, true negative, *etc.*.

Table V: Comparison of precision, recall, F-score, and threshold tuning time between iSST-EVT and iSST

Method	iSST-EVT	iSST
$Precision_c$	91.33%	91.09%
$Recall_c$	90.29%	88.46%
$F\text{-}score_c$	90.81%	89.76%
Running time per time window	$540.5\mu s$	$238.6\mu s$
Avg. tuning time per KPI stream	0	$\sim 15\ min$

Table VI: Adaption lag of StepWise, which is the summation of the delay of the concept drift detection and running the RLM algorithm

Adaption lag	Delay of the concept drift detection	Delay of running RLM
$\sim 360.015s$	$\sim 360s$	$0.015s$

Table V shows the comparison results of accuracy between iSST-EVT and iSST, in terms of $Precision_c$, $Recall_c$, and $F\text{-}score_c$. For a fair comparison, the threshold of iSST of every KPI stream is set as the best for accuracy (F-score). We can observe that iSST-EVT and iSST achieve very close precision, recall, and F-score.

As mentioned in §IV-A, the threshold of iSST has to be manually tuned for every KPI stream. For each KPI stream, it approximately takes fifteen minutes for operators to tune its threshold. As millions of KPI streams should be monitored to determine whether they are impacted by the concept drift, tuning threshold for all KPI streams is almost infeasible. Therefore, it is very important that thresholds are automatically tuned for the sake of scalability and deployability. Our proposed iSST-EVT does not need to manually tune any threshold or parameter, and thus it is appropriate in our scenario.

D. Evaluation of Adaption Lag

To see how "rapid" StepWise is, we evaluate the adaption lag of StepWise, which is defined as the period between when a concept drift occurs and when the anomaly detector is adapted. Apparently, it is the summation of the delay of detection and adaption components.

The delay of detecting the concept drift is dominated by the window size of iSST, which is set to 6 in our scenario

according to §IV-A. Since the KPI data is sampled every minute, for StepWise (iSST-EVT) the delay of detecting the concept drift is at most six minutes (running time per time window is 540.5 μs in Table V).

To calculate the average delay of running the RLM algorithm, we implement the algorithm with C++ and run the program on a Dell PowerEdge R420 server with an Intel Xeon E5-2420 CPU and a 24GB memory. For each KPI data point, the average delay of running the RLM algorithm is 0.015s, which is negligible compared to the delay of the concept drift detection. In this way, the adaption lag of StepWise is $\sim 360.015s$, which is quite acceptable by operators based on the on-site investigation.

VI. Related work

Anomaly detection: Time series anomaly detection has been an active research topic [31]. Considering the performance after concept drift, anomaly detection methods can be classified into the following three categories. However, all of them are not satisfied to operators' need of robust and rapid adaption to the expected concept drift for online anomaly detection. First, some of the anomaly detection methods are designed for processing data in batches or providing a historical analysis, *e.g.*, Twitter's S-H-ESD [4], Netflix's RPCA [5]. They are not suitable for the online scenario. Second, based on supervised learning, Opprentice [1], Yahoo's EGADS [6] and Donut [30] train an ensemble detectors or neural network offline and use this model for online detection. However, they can mitigate the impact imposed by concept drift *after retraining the model*. They have high computational overhead and are not rapid for real-time requirements. Third, some studies like [7], [32] proposed methods that keep a short time detection memory in order to quickly adapt to the concept drift, while they cannot record the historical anomaly pattern which is not robust to detect anomalies.

Change detection: Previous studies focused on the unsupervised change detection are always parametric or have high computational cost [17], for example, CUSUM [18], MRLS [19], iSST [9], [33] and Yahoo's EGADS [6]. However, all the above methods give the change score and it is hard to determine above which score is a significant change in order to apply on the large-scale KPI streams automatically.

Concept drift adaption in other fields: The study of concept drift in online classification scenario has attracted considerable attention in recent years. [8] summarizes that an adaptive learning algorithm can typically have four components, *i.e.*, *memory*, *change detection*, *learning*, and *loss estimation*. The intuitive method of adapting concept drift is to forget old information [34]. The *change detection* component is always necessary for explicit drift detection in such a rapid way [35]. The *learning* component means to generalize from examples and update the predictive models from evolving data, *e.g.*, retraining the new data [36] or incremental adaption updates of the model [37]. Supervised adaptive systems rely *loss estimation* based on environment feedback [38]. However, these methods are not suited for the KPI anomaly detection scenario because the learning-based algorithm needs the classification feedback in real-time. Besides, we cannot forget the old concept but rather make good use of it.

VII. Conclusion

In this paper, we present StepWise, a framework for robustly and rapidly detecting concept drift and adapting anomaly detection algorithms to the new concept after concept drift. Our key observation is that most service KPIs are strongly periodical, concept drift detection can be converted into spike detection in the change score stream and the distributions in the old concept and the new one are statistically correlated. Based on that, StepWise successfully addressed interesting challenges in the adaption for concept drift in terms of scalability, robustness and adaption delay through two novel approaches, iSST-EVT and adaption algorithm. Specifically, there is no manual tuning for threshold or parameter in the iSST-EVT. In addition, StepWise rapidly adapts all types of anomaly detectors after concept drift, instead of tuning parameters for a specific anomaly detector. Our evaluation experiments demonstrate StepWise that not only improves the F-score of anomaly detectors by 206% over a baseline without any concept drift detection, but also is rapid with a detection lag of around 6 minutes.

VIII. Acknowledgements

We strongly appreciate Kaixin Sui, Xiaohui Nie and Dapeng Liu for their valuable suggestions. We thank Juexing Liao and Nengwen Zhao for proofreading this paper. Thorough comments and valuable feedbacks from the reviewers also helped us improve the work. This work was partly supported by the National Natural Science Foundation of China (NSFC) under grant 61472214, 61472210, the Tsinghua National Laboratory for Information Science and Technology key projects, the Global Talent Recruitment (Youth) Program.

References

[1] D. Liu, Y. Zhao, H. Xu, Y. Sun, D. Pei, and et.al, "Opprentice: towards practical and automatic anomaly detection through machine learning," in *Proceedings of IMC*. ACM, 2015, pp. 211–224.

[2] D. R. Choffnes, F. E. Bustamante, and Z. Ge, "Crowdsourcing service-level network event monitoring," in *SIGCOMM*, vol. 40, no. 4. ACM, 2010, pp. 387–398.

[3] Y. Chen, R. Mahajan, B. Sridharan, and Z. Zhang, "A provider-side view of web search response time," in *SIGCOMM*, vol. 43, no. 4. ACM, 2013, pp. 243–254.

[4] A. Kejariwal., "Twitter engineering: Introducing practical and robust anomaly detection in a time series," 2015, https://blog.twitter.com/engineering/en_us/a/2015/introducing-practical-and-robust-anomaly-detection-in-a-time-series.html.

[5] W. J., "Netflix surus github, online code repos," 2015, https://github.com/Netflix/Surus.

[6] N. Laptev, S. Amizadeh, and I. Flint, "Generic and scalable framework for automated time-series anomaly detection," in *Proceedings of the 21th SIGKDD*. ACM, 2015, pp. 1939–1947.

[7] A. Siffer, P.-A. Fouque, A. Termier, and C. Largouet, "Anomaly detection in streams with extreme value theory," in *Proceedings of the 23rd SIGKDD*. ACM, 2017, pp. 1067–1075.

[8] J. Gama, I. Žliobaitė, A. Bifet, M. Pechenizkiy, and A. Bouchachia, "A survey on concept drift adaptation," *Computing Surveys (CSUR)*, vol. 46, no. 4, p. 44, 2014.

[9] S. Zhang, Y. Liu, D. Pei, and et.al, "Rapid and robust impact assessment of software changes in large internet-based services," in *Proceedings of the 11th CONEXT*. ACM, 2015, p. 2.

[10] M. Farshchi, J.-G. Schneider, I. Weber, and J. Grundy, "Experience report: Anomaly detection of cloud application operations using log and cloud metric correlation analysis," in *International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 2015, pp. 24–34.

[11] S. Zhang, W. Meng, J. Bu, S. Yang, Y. Liu, D. Pei, J. Xu, Y. Chen, H. Dong, X. Qu *et al.*, "Syslog processing for switch failure diagnosis and prediction in datacenter networks," in *Quality of Service (IWQoS)*. IEEE, 2017, pp. 1–10.

[12] W. Meng, Y. Liu, S. Zhang, D. Pei, H. Dong, L. Song, and X. Luo, "Device-agnostic log anomaly classification with partial labels," in *Quality of Service (IWQoS)*. IEEE, 2018, pp. 1–10.

[13] Y. Sun, Y. Zhao, Y. Su, D. Liu, X. Nie, Y. Meng, S. Cheng, D. Pei, S. Zhang, X. Qu *et al.*, "Hotspot: Anomaly localization for additive kpis with multi-dimensional attributes," *IEEE Access*, vol. 6, pp. 10 909–10 923, 2018.

[14] Amazon, "Cloudwatch alarm," 2015, http://docs.aws.amazon.com/AmazonCloudWatch/latest/DeveloperGuide/ConsoleAlarms.html.

[15] B. Krishnamurthy, S. Sen, Y. Zhang, and Y. Chen, "Sketch-based change detection: methods, evaluation, and applications," in *Proceedings of the 3rd IMC*. ACM, 2003, pp. 234–247.

[16] H. Yan, A. Flavel, Z. Ge, A. Gerber, and et.al, "Argus: End-to-end service anomaly detection and localization from an isp's point of view," in *INFOCOM*. IEEE, 2012, pp. 2756–2760.

[17] S. Aminikhanghahi and D. J. Cook, "A survey of methods for time series change point detection," *Knowledge and information systems*, vol. 51, no. 2, pp. 339–367, 2017.

[18] A. A. Mahimkar, H. H. Song, Z. Ge, A. Shaikh, J. Wang, J. Yates, Y. Zhang, and J. Emmons, "Detecting the performance impact of upgrades in large operational networks," in *SIGCOMM*, vol. 40, no. 4. ACM, 2010, pp. 303–314.

[20] Y. Kawahara, T. Yairi, and K. Machida, "Change-point detection in time-series data based on subspace identification,"

[19] A. Mahimkar, Z. Ge, J. Wang, J. Yates, Y. Zhang, and et.al, "Rapid detection of maintenance induced changes in service performance," in *Proceedings of the 7th CONEXT*. ACM, 2011, p. 13.

in *7th ICDM*. IEEE, 2007, pp. 559–564.

[21] V. Moskvina and A. Zhigljavsky, "An algorithm based on singular spectrum analysis for change-point detection," *Communications in Statistics-Simulation and Computation*, vol. 32, pp. 319–352, 2003.

[22] J. Beirlant, Y. Goegebeur, J. Segers, and J. L. Teugels, *Statistics of extremes: theory and applications*. John Wiley & Sons, 2006.

[23] A. A. Balkema and L. De Haan, "Residual life time at great age," *The Annals of probability*, pp. 792–804, 1974.

[24] J. Pickands III, "Statistical inference using extreme order statistics," *the Annals of Statistics*, pp. 119–131, 1975.

[25] K. H. Brodersen, F. Gallusser, J. Koehler, N. Remy, S. L. Scott *et al.*, "Inferring causal impact using bayesian structural time-series models," *The Annals of Applied Statistics*, vol. 9, no. 1, pp. 247–274, 2015.

[26] O. C. Ashenfelter and D. Card, "Using the longitudinal structure of earnings to estimate the effect of training programs," 1984.

[27] Y. F. Mohammad and T. Nishida, "Robust singular spectrum transform." in *IEA/AIE*. Springer, 2009, pp. 123–132.

[28] R. Maronna, R. D. Martin, and V. Yohai, *Robust statistics*. John Wiley & Sons, Chichester. ISBN, 2006, vol. 1.

[29] S. Zhang, Y. Liu, D. Pei, Y. Chen, X. Qu, S. Tao, Z. Zang, X. Jing, and M. Feng, "Funnel: Assessing software changes in web-based services," *IEEE Transactions on Service Computing*, 2016.

[30] H. Xu, W. Chen, N. Zhao, Z. Li, J. Bu, Z. Li, Y. Liu, Y. Zhao, D. Pei, Y. Feng *et al.*, "Unsupervised anomaly detection via variational auto-encoder for seasonal kpis in web applications," in *Proceedings of the 2018 World Wide Web Conference*. ACM, 2018, pp. 187–196.

[31] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *Computing surveys (CSUR)*, vol. 41, no. 3, p. 15, 2009.

[32] H. Huang and S. P. Kasiviswanathan, "Streaming anomaly detection using randomized matrix sketching," *Proceedings of the VLDB Endowment*, vol. 9, no. 3, pp. 192–203, 2015.

[33] S. Zhang, Y. Liu, W. Meng, Z. Luo, J. Bu, S. Yang, P. Liang, D. Pei, J. Xu, Y. Zhang *et al.*, "Prefix: Switch failure prediction in datacenter networks," *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, vol. 2, no. 1, p. 2, 2018.

[34] A. Bouchachia, "Fuzzy classification in dynamic environments," *Soft Computing*, vol. 15, no. 5, pp. 1009–1022, 2011.

[35] A. Haque, L. Khan, and M. Baron, "Sand: Semi-supervised adaptive novel class detection and classification over data stream." in *AAAI*, 2016, pp. 1652–1658.

[36] J. Gama, P. Medas, G. Castillo, and P. Rodrigues, "Learning with drift detection," in *Brazilian Symposium on Artificial Intelligence*. Springer, 2004, pp. 286–295.

[37] A. Haque, L. Khan, M. Baron, B. Thuraisingham, and C. Aggarwal, "Efficient handling of concept drift and concept evolution over stream data," in *32nd ICDE*. IEEE, 2016, pp. 481–492.

[38] M. Harel, S. Mannor, R. El-Yaniv, and K. Crammer, "Concept drift detection through resampling," in *Proceedings of the 31st ICML*, 2014, pp. 1009–1017.