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

Minghua Ma †§, Shenglin Zhang *‡, Christopher Zheng ¶, Xinhao Jiang ‖, Dan Pei †§,
† Department of Computer Science and Technology, Tsinghua University,
§ Beijing National Research Center for Information Science and Technology (BNRist),
‡ College of Software, Naikai University,
¶ School of Computer Science, McGill University,
‖ Department of Computer Science, University of California (Berkeley)
mmh16@mails.tsinghua.edu.cn, zhangsl@nankai.edu.cn

Abstract—Anomaly detection is critical to 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 a significant change of data distribution, namely a concept drift, which is caused by software changes or evolving user preferences. Even though dozens of anomaly detectors have been proposed over the years in the context of software systems, they have not tackled the problem of concept drifts. In this paper, we present a framework, StepWise, which can detect concept drifts without manually tuning detection threshold or per-KPI (Key Performance Indicator) model parameters in large-scale KPI streams, take external factors into account to distinguish the concept drifts according to operators' expectations, and help any type of anomaly detection algorithms to rapidly handle and adapt them. For the prototype deployed in Sogou, our empirical evaluation shows that StepWise improves the average F-score by 47% for many widely-used anomaly detectors over a baseline without any concept drift detection.

Index Terms—Anomaly detection, concept drift, software service KPI, web-based software system.

1 Introduction

With the booming development of web-based software systems like search engines, online shopping and social networks, thorough analysis of monitoring data is becoming increasingly important to software reliability. In general, operators of the systems above 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, by analogy, the term "concept drift" indicates that KPI stream distribution changes significantly [8]. 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 loss imposed by an unexpected concept drift, 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.

* *Shenglin Zhang is the corresponding author.*

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 [9]. 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 whereas operators consider these points as normal. In this work, we try to *quickly 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 for thousands of times every day [10], and this situation leads to a great number of concept drifts in KPI streams. For example, in the studied company Sogou which operates a top-tier search engine in China, there are *more than 3000* concept

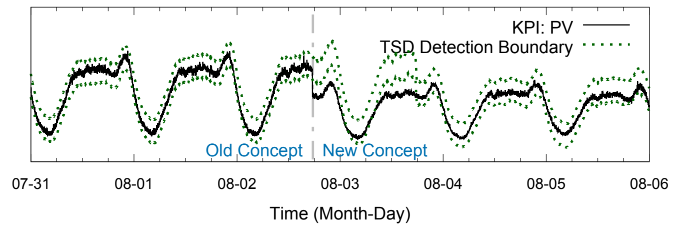


Fig. 1: A toy example of a concept drift. KPI outside the detection boundary triggers an alarm.

drifts of KPI streams *per day*, *more than 80%* of which are expected ones. Besides, concept drifts also have diverse types which render the handling even more challenging. 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 [10]. 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) approaches 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 appears 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. To eradicate false alarms, our concept drift adaption approach must detect concept drifts as well as help anomaly detectors adapt multiple types of concept drifts in a precise and rapid manner.

To tackle the challenges above, 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. StepWise has three components. (1) **detection**: StepWise adopts an **iSST-EVT** method to detect concept drift for a large number of KPIs, without tuning detection thresholds 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 features and behaviors of past timelines, StepWise applies a novel method for different types of concept drifts to automatically adapt anomaly detectors to accurately detect anomalies immediately after concept drifts (in terms of the trend, the width of detection boundaries *etc.*). The only work for operators is to determine which type a concept drift is of.

Our contributions are summarized as follows:

- To the best of our knowledge, there is no related work 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 a concept drift, and its research challenges in terms of scalability, robustness, and adaption delay.
- We implement a prototype of StepWise which addresses the challenges above 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 types of concept drifts and features and behaviors of historical KPI timelines.
- To demonstrate the excellent 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 swiftly adapt to new concepts. Our evaluation shows that StepWise improves the average F-score by 47% for many widely-used anomaly detectors over a baseline without any concept drift detection.

2 Background

In this section, we familiarize readers with selected preliminaries: software service KPI anomaly, concept drift, and anomaly detection.

Software Service KPI: In this work, we collect KPI streams from Sogou. Sogou is China's fourth largest Internet company and operates the second largest search engine. It has 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 hosting 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 deals with 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] 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

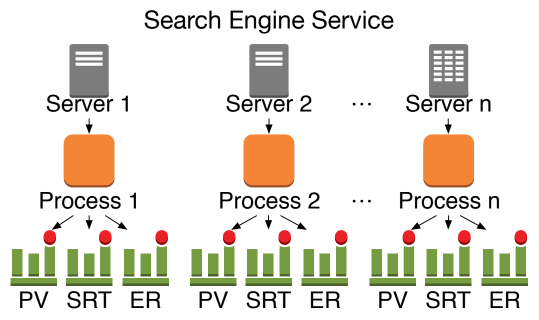


Fig. 2: A toy example of how software service KPI measurements are collected.

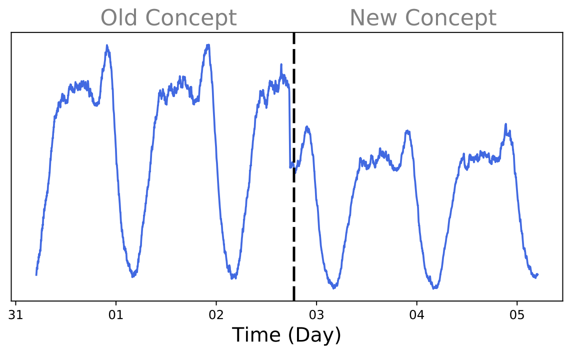
constitute a KPI stream with a format of $(time, value)$ for each time bin.

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, but they do not care about concept drifts in resource KPIs.

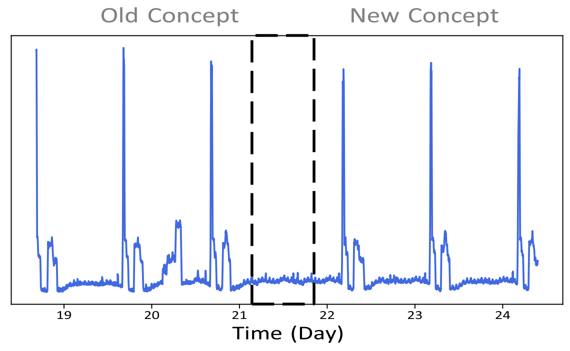
KPI Anomaly: An anomaly in a KPI stream is typically a sudden spike, a jitter, or a dip, *etc.*, and indicates an unexpected pattern of KPI. It is usually a short-term or spasmodic change, but it can also be an unexpected level shift or ramp up/down. In general, operators deploy anomaly detectors based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

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 [10], 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, a traffic shifting occurred at 17:40 on the 2nd, August. The distribution of PV changed significantly within 5 minutes and remained unchanged for over three days.

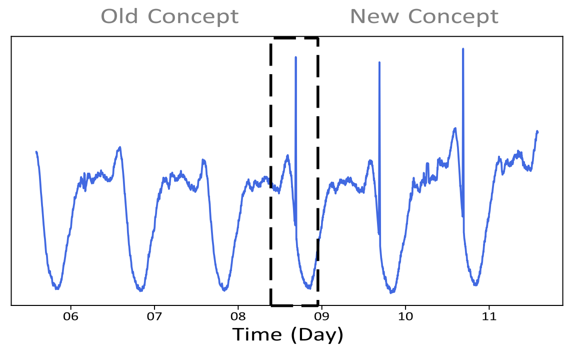
Concept drifts can be divided into four categories. 1) Level-shift concept drift (Fig. 3(a)). It suggests when the level of a KPI timeline noticeably boosts or drops to a new level at certain time and remains there for the immediate following timestamps. Respectively, it is termed a level shift-up or level shift-down. Level shifts are primarily caused by things like parameter changes or workload amount modifications. 2) Phase-shift concept drift (Fig. 3(b)). Unlike a vertical level shift, a phase shift can be interpreted as a horizontal displacement that advances or postpones the next phase repetition by certain magnitude. Phase shifts are usually the side effects of switching working regions or time zones. 3) Overshadowing concept drift (Fig. 3(c)). Given a timestamp, the timeline before it cannot equate to the one after by simple displacement or multiplying any coefficient due to dramatic shape changes *i.e.*, they do not share a linear relationship because there is another pattern intertwining with the original one. This behavior is normally introduced by adding or running massive periodic jobs back-end. The periodicity is often one day. 4) Recurrent concept drift. It encapsulates the recurrence of any



(a) Level-Shift



(b) Phase-Shift



(c) Overshadowing

Fig. 3: Three basic types of concept drifts. The highlighted or enclosed regions by dashes imply when concept drifts occur.

one of the three aforementioned types of concept drifts after its first change done to the timeline. Typically, the timeline shapes before and after are homogeneous because the main reason for recurrent concept drifts is the switching of temporary flows. Of course, the definition does not restrict that the final-phase behavior should always be identical to its original.

As shown in the Venn diagram in Fig. 4, 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 the 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 rapidly adapt to the new concept,

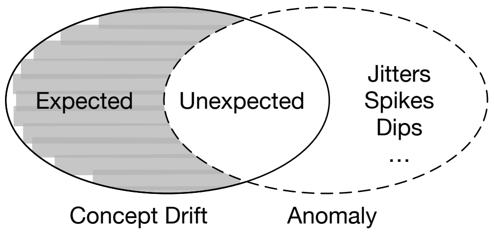


Fig. 4: Relationship between concept drifts and anomalies.

generating a lot of false alarms or false negatives (see §5 for details). Consequently, it is very necessary to adapt anomaly detectors immediately after concept drifts.

Fig. 4 shows the similarities and differences between concept drift and KPI anomaly. Both concept drift and KPI anomaly include unexpected concept drift. An expected concept drift, however, 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.*, TSD [3], Diff [1], SPOT [7] DSPOT [7], or a combination of two or more detectors, *e.g.*, Opprentice [1]. In Sogou, operators have deployed the seven detectors above 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. 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. Nonetheless, 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, DSPOT has two parameters: d, q. These parameters are also configured based on the distribution before a concept drift, and thus should be re-trained after it.

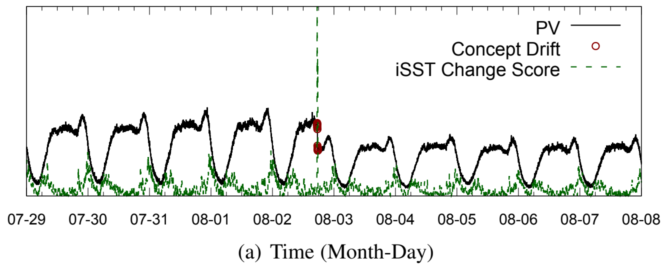
3 Intuition behind StepWise

This section presents the domain-specific insights we use to help address the challenges mentioned in §1. We summarize this section by concluding the two challenges in applying these insights in a practical system.

3.1 Concept Drift Detection

Previous works focus 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 the variety of KPI streams. Therefore, *our goal of concept drift detection algorithm is the one that does not require per-KPI parameter tuning or change score threshold tuning*.

Past 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], [12]–[15]. The output of these detection algorithms



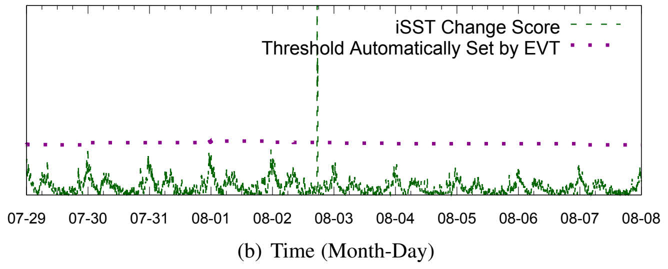


Fig. 5: A concept drift in the original KPI stream (marked by red circles) is successfully detected by iSST-EVT because the corresponding iSST change score is higher than the threshold automatically set by EVT (§4.1).

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 [13]), kernel-based model [15], SST [16], 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 [13], Multiscale Robust Local Subspace (MRLS) [14], improved Singular Spectrum Transform (iSST) [10]) need to be tuned on a per-KPI basis so that the change is severe enough to the operators. Nevertheless, 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 [10] algorithm to calculate the change score, as Fig. 5 (a) shows. There is a spike in the iSST change score stream on 2 $\sim$ 3 August, at the time of concept drift. iSST [10] is a recent concept drift detection approach which is robust and rapid for it can use model-wide empirical values for model parameters, thus does not need model parameter tuning on a per-KPI basis. For different KPI streams, the threshold of change score, however, may be different. This is one remaining challenge. Inspired by Insight 1, we design a spike detection approach in §4.1 which does not need human to tune per-KPI change score threshold.

3.2 Concept Drift Adaption

For the KPI streams from web-based software systems, we discover that a concept drift, if happens, is almost always caused

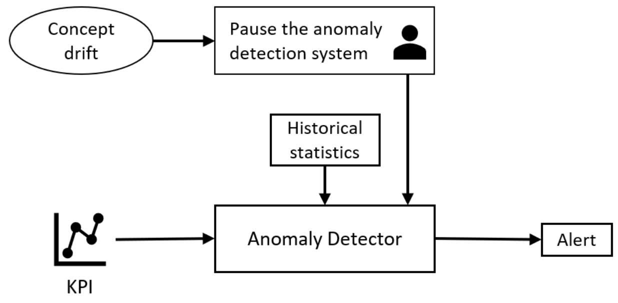


Fig. 6: Traditional way of concept drift adaption

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 5% are caused by holiday impacts. We believe that this situation is not rare in web-based software systems.

A KPI timeline's data distribution before a concept drift is considered as **old concept**, while the distribution after that drift is considered as **new concept**.

Although methods of detecting occurrences of concept drifts are under extensive research, the field of concept drift adaption remains mostly unstudied to the best of our knowledge. No existing methods provide elegant solutions to adapting new concepts to old concepts with little latency. In the industry, when an operator faces a concept drift, he or she basically reboots the anomaly detection system to let it "forget" about the past KPI behaviors and make it gradually adapt the new concept via feeding in new timeline values. This approach is both labor-intensive and time-consuming, and it loses the precious information of past timeline patterns.

Insight 2: ***An efficient method is needed to better utilize KPIs' characteristics and past behaviors for concept drift adaptions.***

Traditional anomaly detection methods, like TSD, analyze a large chunk of historical KPI-related data, receive KPI input, and then decide whether to trigger an alert. These types of anomaly detectors, however, tend to fail when dealing with KPI timelines with one or more concept drifts. As illustrated in Fig. 6, a traditional anomaly detector has to be paused by an operator when he or she is aware of a concept drift. Based on historical data and one day's new data after the concept drift, the operator manually modifies the anomaly detector to a new range that can help monitor the subsequent new concept and resume anomaly detection.

On the contrary, to lift the burden off the operators, we aim for another method of detecting anomalies by feeding something based on but in place of historical data, which requires much less human intervention. Further, this method should be applicable to other anomaly detectors that function by analyzing near-historical data beforehand.

3.3 Practical Challenges

There are two challenges in the practical system design.

- How to detect spikes in the change score stream? This is challenging because the amplitude of spikes varies both across different KPIs and over time [10], and it is infeasible to manually configure the threshold of change score for each KPI stream.
- How to design a method to correctly and swiftly adapt different types of concept drifts? We categorize concept drifts into four types and each of them demonstrates distinct behaviors and properties in KPI timelines. Thus, we can safely assume that we need a versatile and comprehensive method to accurately and rapidly adapt them.

4 StepWise Detailed Design

We propose a practical framework, StepWise, for robust and rapid adaption for concept drift in software anomaly detection. As Fig. 7 shows, our framework has three main components: **detection**, **classification**, and **adaption**. First, in §4.1 we propose a concept drift detection method that does not require tuning model parameters or tuning change score threshold. Second, in §4.2 we classify concept drifts into *expected* vs. *unexpected* ones. We apply causal impact analysis to make this classification semi-automatic. Third, in §4.3, we propose a robust and rapid adaption algorithm. Our main contributions focus on the detection and adaption components.

4.1 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 takes advantage of both iSST and Extreme Value Theory.

4.1.1 Improved Singular Spectrum Transform without Thresholds (iSST-EVT)

As aforementioned in §3.1, we use iSST to calculate the change score on each time 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) [17] 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 values (spikes) have the same kind of distributions, regardless of the original ones. According to the 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 the 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:

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

Nonetheless, 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 [18], [19] (also called *second theorem* in EVT) is proposed as follows:

Theorem 4.1 (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. $1+\gamma x > 0$:*

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

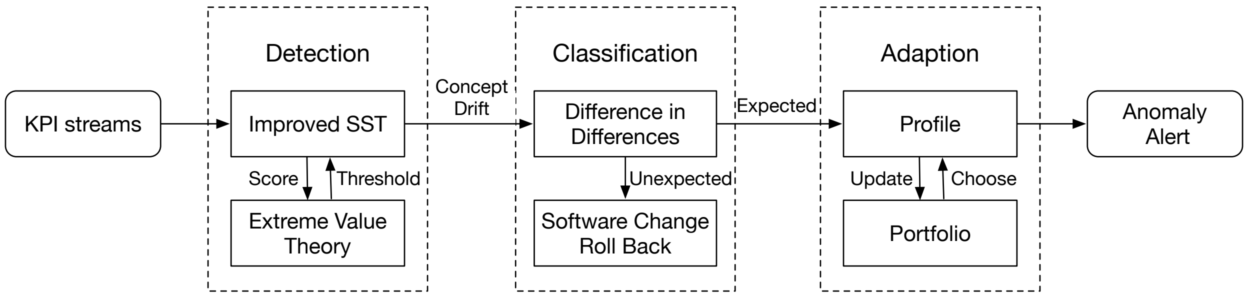


Fig. 7: Framework of StepWise.

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. 5 (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 that can indicate a concept drift in the original KPI stream. Based on the domain experience and knowledge and [7], we suggest 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 that imply the exact occurrence 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.*

4.1.2 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 [10], 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. We empirically set the window size of 6 (minutes) in StepWise to declare a change in KPI stream as a concept drift.

4.2 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 [20] to conduct semi-automatic classification. Causal impact analysis [20] is the process of drawing a conclusion

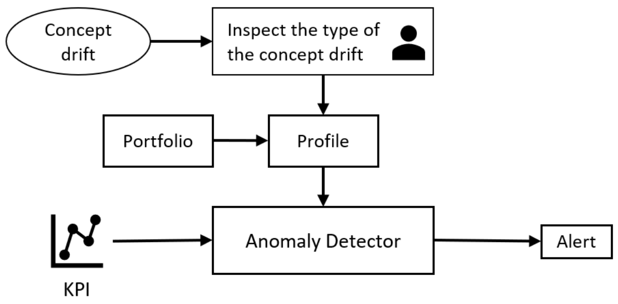


Fig. 8: Our way of semi-auto adaption by updating the profile or directly choosing an existing profile state from its portfolio.

of 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 [10], namely Difference in Differences (DiD) [21], 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 mention later, we talk about only adaptions to expected concept drifts in our work.

4.3 Concept Drift Adaption

Analogous to a human's resume, a **profile** of a KPI stream is the data image for storing valuable historical information of this specific indicator. Stored in a **portfolio**, the profile is concise and suggestive of a KPI's fundamental states and behaviors in the past. Specifically, the profile houses information such as the exact timestamps, baselines, lower bounds, upper bounds *etc.* which are all useful statistics for anomaly detection. After a concept drift happens as shown in Fig. 8, an operator determines the type of it, according to which StepWise updates the KPI's profile so that post-concept-drift anomaly detection can rely on this profile.

Here we introduce the three main procedures related to profile modifications: 1) creating a profile, 2) updating a profile, 3) choosing a profile.

4.3.1 Creating a Profile

As the initialization step, if a KPI is a newcomer or its profile is not existent in our repository, we need to create a new one for it. To ensure a profile is indeed accurate, we accumulate and analyze the KPI's historical statistics and data with a time span that is no less than one day in length. This time length requirement is to make sure that we have a complete set of statistics on a daily basis ready for thorough analyses.

In our scenario, given a timestamp, we assign the median as the baseline of the profile, one standard deviation left and right as the lower and upper value boundaries, respectively. This parameter assignment is default because this combination of the median and standard deviation can potentially offset some impacts of data outliers and fluctuations. Of course, to make the profile construction more versatile and to allow more possibilities, we also accept various types of anomaly detectors' output as building blocks of profiles *i.e.*, we can consider their output as baselines and boundaries to create profiles. In summary, for a complete and valid piece of information stored in one KPI's profile, it must contain a timestamp, a baseline value, an upper value bound, and an lower value bound. Note that, in this case, the timestamp includes only the time in a day but no date because we analyze the timeline behaviors on a daily basis.

4.3.2 Updating a Profile

We need to modify a profile to reflect up-to-date KPI timeline changes accordingly. We assume timeline changes occur after only concept drifts, and the detailed procedures to update a profile are as follows.

After an arbitrary concept drift, a DBA manually labels out the type of this specific concept drift. We do require human intervention only at this step because DBAs are aware of reasons for expected concept drifts, such as scaling up/down systems, switching time zones, running crontab jobs *etc.* After the concept drift's type is determined, StepWise takes actions based on the type. (1) If it is a level-shift concept drift, StepWise utilizes the Robust Linear Model [22], which is robust against anomalies because of the distribution with heavy tails [23], to calculate the coefficient of the timelines before and after, and then updates this KPI's profile by multiplying the coefficient to the original values in the profile and vertically shifts the timelines by a constant also generated by the Robust Linear Model. (2) If it is a phase-shift concept drift, StepWise updates the profile through shifting the timestamps of the information pieces in the profile by the size of the time zone difference specified by the operator. (3) If it is an overshadowing concept drift, StepWise recognizes the pattern of changes made to the timeline before the concept drift and extracts the pattern of this "overshadowing" timeline. From DBAs' domain knowledge, overshadowing concept drifts are generated by running massive jobs (*e.g.*, clearing tasks) at certain time of the day and these jobs tend to render "spikes" overlapped with the original KPI timelines. As an update, StepWise adds this overshadowing pattern to the KPI's profile. On the contrary, if one finishes running crontab jobs and a KPI is back to normal after a concept drift, StepWise deletes the overshadowing pattern from the profile. (4) If it is a recurrent concept drift, StepWise searches in this profile's history and returns this profile back to the state which is the most similar to that of the current timeline.

The usage of historical profile information is the reason why StepWise stores past states of each KPI's profile: they may recur. StepWise creates a set of Top-K profile for a KPI which houses various states of a profile such as workdays, weekends, traffic switches, crontab jobs. Therefore, when one is looking for a KPI profile, StepWise searches in this list of profile states and outputs the best match.

Admittedly, each KPI is different in terms of its baseline and boundaries, but the change of KPIs resembles that of one another in a distributed system. Thus, operators need to label only one concept drift on the KPI and StepWise can replicate this change of model to others by analogy. This transfer can reduce the labeling overheads to a very large extent and improve efficiency.

TABLE 1: Summary of the concept drift detection dataset.

Datasets	Concept Drift
# KPIs	288
# Unexpected concept drifts	52
# Expected concept drifts	282

TABLE 2: Numbers of all data points and of anomaly data points in three types of concept drifts in our dataset.

Concept	# Data Points	Level-shift	Phase-shift	Overshadowing
Old	Total	28415	15840	23040
	Anomaly (Ratio)	115 (0.40%)	431 (2.72%)	306 (1.33%)
New	Total	16225	33834	27360
	Anomaly (Ratio)	531 (3.27%)	180 (0.53%)	523 (1.91%)

4.3.3 Choosing a Profile

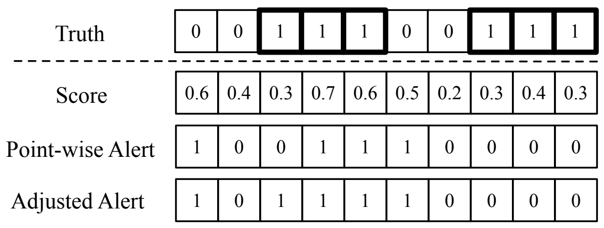
The functionality of choosing profiles is reserved only for recurrent concept drifts. The primary purpose of this functionality is to match the nature of recurrent concept drifts that a new concept corresponds to at least one period of historical old concepts *i.e.*, an old concept of the KPI "recurs". In this case, instead of asking operators to manually decide which past concept state to set back to, we search in all the profiles of the KPI to find and select the most similar one to that of the new concept as the profile for this concept recurrence. Once a new profile is settled, the system can very easily adapt to this recurrent concept drift. The selection policy is to calculate the Mean Squared Error (MSE) of the new concept's profile and all the other profiles, and then determine the one with the lowest MSE score *i.e.*, their profile patterns are the most alike.

5 Evaluation

StepWise is designed to rapidly and robustly adapt to the expected concept drift for anomaly detectors. We deploy 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 §5.2. Then we compare our concept drift detection method with FUNNEL [24] in §5.3. We show the adaption lag of StepWise at the end of this Section.

5.1 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



Truth	0	0	1	1	1	0	0	1	1	1
Score	0.6	0.4	0.3	0.7	0.6	0.5	0.2	0.3	0.4	0.3
Point-wise Alert	1	0	0	1	1	1	0	0	0	0
Adjusted Alert	1	0	1	1	1	1	0	0	0	0

Fig. 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 3: Examined detectors and their parameters. (Abbr: TSD for Time Series Decomposition and Diff for Differences)

Detector	Sampled parameters
TSD [3]	period $= 1$ day
Diff [1]	period $= 1$ day
bidSPOT [2]	d $= 8, 10, 20$; p $= 10^{-4}, 10^{-3}$
dSPOT [25]	d $= 8, 10, 20$; p $= 10^{-4}, 10^{-3}$

search engine systems by operators, as Table 1 shows. We list the numbers of total data points and of anomaly data points labeled by operators in Table 2. 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, and thus 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.

5.2 Evaluation of The Overall Performance

We evaluate the overall performance of StepWise by utilizing five different strategies to deal with various types of concept drifts, and the results are shown in Fig. 10.

The five anomaly detection methods we select are Time Series Decomposition (TSD) [3], Difference [1], bidSPOT [2], dSPOT [2], and our Portfolio, the first four of which are all widely-used anomaly detectors. We compare and contrast their performances in terms of three types of concept drifts (level-shift, phase-shift, overshadowing concept drifts). For level-shift concept drift adaption, all five detectors obtain acceptably high F1-scores before concept drifts, but TSD's, bidSPOT's, and dSPOT's scores after the concept drift plunge to zero or near zero. Although Difference's score after the concept drift is not low, it is led by that of Portfolio by about 15%. Note that Portfolio's score difference (between the before and after scores) is only around 10% which suggests that Portfolio is competent in detecting and adapting to level-shift concept drifts. For phase-shift concept drifts, bidSPOT and dSPOT perform noticeably badly both before and after the concept drift. This is because both of them can not make use of time series seasonality. Thus, they trigger a false alert when they witness a seemingly large value change which is actually present in every phase of this time series. Besides, after the concept, both continuously trigger false alerts because they cannot adapt to new concept after the phase shift. Notice that Portfolio achieves the smallest F1-score difference and best F1-score after the drift. This demonstrates that Portfolio is excellent in phase-shift concept drift detection and adaption. For overshadowing concept drifts, dSPOT and Difference obtain acceptably small score difference value while the score difference of Portfolio is zero and it achieves the highest F1-score both before and after the concept drift. Hence, Portfolio is very effective in rapidly and accurately adapting to overshadowing concept drifts. In summary, our Portfolio is a comprehensive method that is capable to quickly and effectively detect and adapt to all types of concept drifts.

TABLE 4: Comparison of precision, recall, F-score, and threshold-tuning time of iSST-EVT and iSST.

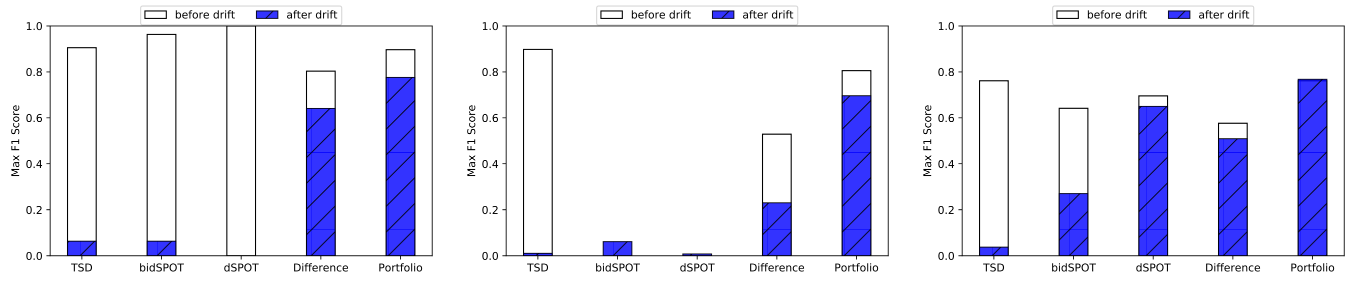
Method	iSST-EVT	iSST
$Precision_c$	91.33%	91.09%
$Recall_c$	90.29%	88.46%
$F\text{-}score_c$	90.81%	89.76%
Running time per time window	$540.5\mu s$	$238.6\mu s$
Avg. tuning time per KPI stream	0	$\sim 15\ min$

5.3 Evaluation of Concept Drift Detection

As described in §4.1, 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 [24], 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.

Similar to the evaluation of overall system in §5.2, 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 4 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 observe



(a) Level-Shift (b) Phase-Shift (c) Overshadowing

Fig. 10: Best F1-score performance of TSD, bidSPOT, dSPOT, Difference, and our Portfolio handling three basic types of concept drifts.

TABLE 5: The adaption lag per item of five anomaly detectors. Adaption lags for both bidSPOT and dSPOT are unavailable.

	TSD	bidSPOT	dSPOT	Difference	Portfolio
Time/adaption	0.07ms	N/A	N/A	1.70ms	1.59ms

that iSST-EVT and iSST achieve very close precision, recall, and F-score.

As mentioned in §4.1, 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 human to manually tune any threshold or parameter, and thus it is appropriate in our scenario.

5.4 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 accordingly 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 §4.1. 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 4). Besides, the result of the adaption components is displayed in Table 5. Although TSD is very fast in undertaking one concept drift adaption (0.07ms/adaption), its poor performance shown in Fig. 10 renders it noncompetitive. Adaption lags for both bidSPOT and dSPOT are unavailable because they do not have fixed time periods for adaption and their adaption performances are not promising either. Portfolio outperforms Difference with adapting one concept drift in 1.59 ms. This result along with the conclusion we draw from Fig. 10 demonstrate that our Portfolio is both accurate and rapid. In this way, the average adaption lag of StepWise is $360.00159s$, which is quite acceptable by operators based on our on-site investigation.

6 Related Work

Anomaly detection: Time series anomaly detection has been extensively studied [26]. Considering the performance after concept drifts, anomaly detection methods can be classified into three categories. First, some of the anomaly detection methods are designed for processing data in batches or providing historical analyses, *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 [27] train ensemble detectors or neural networks offline and use these models for online detection. Admittedly, they can mitigate the impact imposed by concept drifts *after retraining the model*, but they have high computational overheads and are not rapid enough to be real-time. Third, some studies like [7], [28] 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 patterns. Therefore, their propositions are not robust to detect anomalies. All of them are not satisfactory enough for operators' need of robust and rapid adaption to the expected concept drifts for online anomaly detection.

Change detection: Previous studies that focused on the unsupervised change detection are always parametric or have high computational costs [12], *e.g.*, CUSUM [13], MRLS [14], iSST [10], [29] and Yahoo's EGADS [6]. Although all the aforementioned methods output change scores, it is hard to determine which score above is a significant change in order to automatically apply on the large-scale KPI streams.

Concept drift adaption in other fields: The study of concept drifts 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 [30]. The *change detection* component is always necessary for explicit drift detection in such a quick way [31]. The *learning* component generalizes from examples and updates the predictive models from evolving data, *e.g.*, retraining the new data [32] or incremental adaption updates of the model [33]. Supervised adaptive systems rely on *loss estimation* based on environment feedback [34], but these methods are not suitable for our 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.

7 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, that concept drift detection can be converted into spike detection in the change score stream and that concept drift adaption can be achieved by well utilizing KPIs' characteristics and past behaviors. Based on that, StepWise successfully addresses interesting challenges in the adaption for concept drift in terms of scalability, robustness and adaption delay through two novel approaches, iSST-EVT and Portfolio. 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 47% over a baseline without any concept drift detection, but also is rapid enough with a detection lag of around 6 minutes.

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] 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.

[10] 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.

[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.

[14] 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.

[12] 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.

[13] 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.

[15] Y. Kawahara, T. Yairi, and K. Machida, "Change-point detection in time-series data based on subspace identification," in *icdm*. IEEE, 2007, pp. 559–564.

[16] 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.

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

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

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

[20] 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.

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

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

[23] M. Ma, S. Zhang, D. Pei, X. Huang, and H. Dai, "Robust and rapid adaption for concept drift in software system anomaly detection," in *2018 IEEE 29th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 2018, pp. 13–24.

[24] 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.

[25] 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.

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

[27] 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.

[28] 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.

[29] 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.

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

[31] 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.

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

[33] 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.

[34] 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.