2019 IEEE International Conference on Web Services (ICWS)

AIMS: A Predictive Web API Invocation Behavior Monitoring System

Lanxuan Tong, Jian Cao †, Qing Qi, Shiyou Qian
Shanghai Institute for Advanced Communication and Data Science, Department of Computer Science and Engineering
Shanghai Jiao Tong University
Shanghai, China
Email: lxtong0526@163.com, cao-jian@sjtu.edu.cn, qi_ng616@sjtu.edu.cn, qshiyou@sjtu.edu.cn

***Abstract*—With the extensive applications of Web APIs, to monitor and analyze personal Web API invocation behaviors is becoming more and more important. However, due to the fact that the users' Web API invocation behaviors are highly heterogeneous, it is extremely challenging to provide a unified framework. In the paper, we introduce a predictive Web API Invocation Behavior Monitoring System (AIMS). AIMS automatically analyzes the predictability of users' invocation behaviors and for the user whose behaviors can be predicted, an adaptive strategy is applied to generate the predictions in an adaptive way. In addition, a context-aware K-nearest neighbor classifier is applied to detect the anomaly of user's invocation behaviors. Experiments on both a real-world dataset and synthetic dataset show AIMS is efficient in analyzing personal API invocation behaviors.**

***Keywords*-Web API; Predictability Analysis; An Ensemble Prediction Strategy; A Context-aware K-Nearest Neighbor Classifier**

I. Introduction

Many enterprise IT providers are shifting to an everything-as-a-service (XaaS) model, in which the internal business asserts can be shared through Web APIs. Web APIs, also referred to as RESTful services [1] when conforming to the REST architectural principles, relies on the use of URIs, for both resource identification and interaction, and HTTP for message transmission. Because of their relative simplicity and their natural suitability for the Web [2], the number of Web APIs is increasing very rapidly in recent years and the term of API Economy is being used to describe this movement of the industries [3].

Not surprisingly, a Web API is invoked very frequently by users worldwide in some Internet companies. As an example, Twitter receives 6 billion API calls per day, or 70,000 per second [4]. Once a Web API is invoked, the information about when, where and how an API is invoked can be detected and recorded so that users' Web API invocation behaviors can be collected.

To analyze user's Web API invocation behaviors is becoming more and more important because of the following reasons:

1) It is a critical issue to ensure the quality of Web API. Service providers have to spend a lot on the provision of resources such as hardware, network bandwidth and middleware to render services to clients. At the same time, the resource pool should be adjusted in terms of the changing workloads brought by user's API invocations. Moreover, different users have different priorities regarding to the service qualities, which needs to provide specific resources to them based on the understandings of their invocation behaviors.
2) Although API vendors adopt multiple strategies such as authentications and invocation times limitations to prevent malicious invocations, the risks such as customer account hacking or unusual invocations caused by the bugs in users applications cannot be avoided. If we know the normal invocation behaviors of a specific user, the abnormal invocation behaviors can be detected efficiently.
3) Value-added services can be provided to customers when their invocation behaviors are analyzed and reported such as API usage analysis and prediction, optimizing their expenditures on the service usage if API invocation is charged or warnings of abnormal invocations.

Monitoring individual behaviors on the Web is not a new task and there are already many approaches proposed to analyze user's different web access behaviors. Unfortunately, there still lacks a unified framework to monitor and analyze personal Web API invocation behaviors.

Fig. 1 shows API invocation statistics of Commit API in GitHub from three users. It can be found that some users' behaviors are almost irregular and unpredictable. Although some users' behaviors present approximately regular patterns, their patterns are very different. It means it is not possible to provide a single method to predict and analyze different users' behaviors. Even worse, user behavior patterns are changing with time so that it is difficult if not possible to find a model that always perform well.

We introduce a system named AIMS, which stands for a predictive Web ***A***PI ***I***nvocation Behavior ***M***onitoring ***S***ystem. It automatically analyzes the predictability of users' invocation behaviors and for the user whose behaviors can be predicted, an adaptive strategy is applied to generate the predictions in an adaptive way. In addition, a context-aware K-nearest neighbor classifier is applied to detect the anomaly

† Jian Cao is corresponding author.

978-1-7281-2717-0/19/$31.00 ©2019 IEEE
DOI 10.1109/ICWS.2019.00067

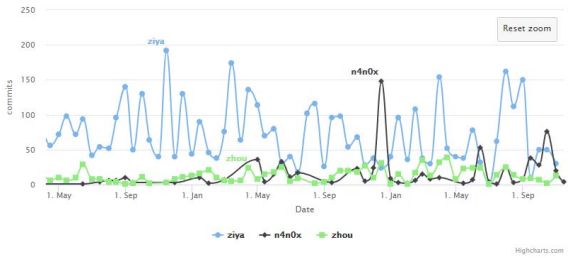

Figure 1. Different "commit" invocation behavior patterns

of user's invocation behaviors. The contributions of the paper include:

1) We propose the structure for a predictive Web API Invocation Behavior Monitoring System.
2) Based on the predictability analysis of the user's invocation behaviors, we propose an adaptive strategy for behavior predictions.
3) We design a context-aware K-nearest neighbor classifier to detect the anomaly invocation behaviors adaptively.

Experiments are performed on both a real-world dataset and synthetic dataset, which shows AIMS is efficient in analyzing API invocation behaviors.

The rest of the paper is organized as follows: Section II reviews some related work. Section III gives a brief description of the structure of AIMS. Section IV shows our Web API invocation behavior prediction methods, including a Web API invocation behavior predictability analysis and an ensemble-based prediction approach. The anomaly detection approach on Web API invocation behavior is described in Section V. The implementations and experiments are presented in Section VI. Finally, conclusions are given in Section VII.

II. Related Work

In this section, we review related work from several parts including Web API monitoring, quality of service monitoring and prediction, behavioral monitoring and analysis, and anomaly detection.

A. Web API Monitoring

Many vendors have developed API monitoring tools together with their Web API platforms. The main functions of these tools include collecting API event data, tracing the status of API executions and presenting the statistic results. Amazon API Gateway, developed by the leader of cloud provider Amazon, is an AWS service that enables developers to create, publish, maintain, monitor, and secure APIs at any scale. AWS X-Ray, AWS CloudTrail, and Amazon CloudWatch are tools that Amazon API Gateway developers can use to trace, log, and monitor Web API execution and management operations [5]. IBM API Connect offers monitoring and built-in analytics of services, for providers to build better APIs with high quality. It helps service providers manage service levels, set quotas, establish controls, set up security policies, manage communities and analyze trends [6]. Obviously, the focus of these API platforms is to provide management tools to maintain the performance of Web APIs, and they cannot monitor and analyze personal Web API invocation behaviors.

B. Quality of Service Monitoring and Prediction

Quality of Services (QoS) is an important research topic in Web Service area. In order to measure the quality of services, monitoring tools are necessary [7]. A novel weighted naïve Bayesian runtime monitoring approach based on information gain theory and sliding window mechanism, called IgS-wBSRM is proposed [8]. Moreover, methods including time series models and machine learning models are proposed for QoS prediction [9] [10]. Since the QoS of services varies widely among users due to the unpredicted network, physical location and other objective factors, many Collaborative Filtering based approaches are recently proposed [11] [12]. Comparing with service quality data, service invocation behavior data is more complex since different people have totally different behavior patterns.

C. Behavioral Monitoring and Analysis

Web API invocations often correspond to user's personal behaviors. There are many approaches proposed for personal behavior monitoring and analysis with different definitions on behaviors in different domains. For example, human activity recognition requires an automated identification of high-level activities, composed of multiple simple (or atomic) actions of persons [13]. In some applications, behavioral monitoring functionality help users understand "normal" system and network activity so that the incident response can be simplified when investigating an operational issue or potential security incident [14]. Unfortunately, Web API invocation behavior monitoring and analysis has not been investigated.

D. Anomaly Detection

Anomaly detection is essentially a classification problem and the main idea is to monitor the running states and classify every state into normal or abnormal ones. There are many machine learning algorithms, which can be applied to this problem. However, to label anomalies is a labor and knowledge intensive job. Therefore, unsupervised machine learning based anomaly detection approaches attract more interests in recent years, in which K-Nearest Neighbor (KNN) classifier has been widely adopted [15]. Improvements are also proposed such as to speed up the process of searching nearest neighbors [16].

Web API Invocation behavior patterns change with time and traditional KNN classifier is hard to adapt to the

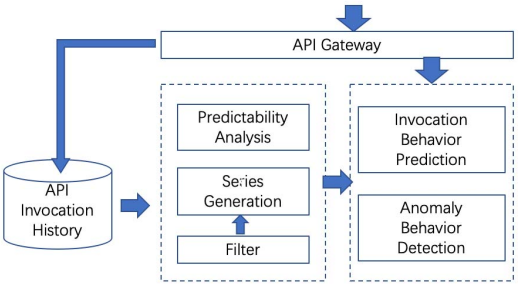

Figure 2. Framework of AIMS

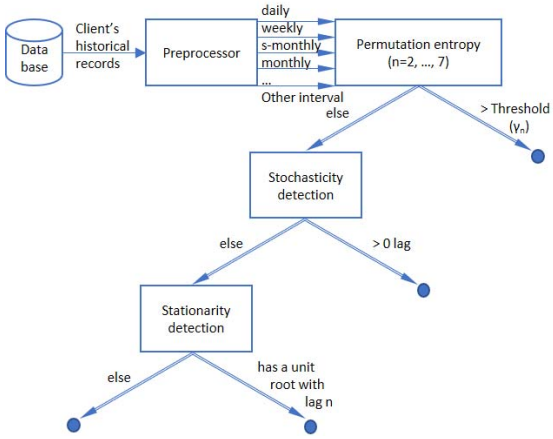

Figure 3. Process of predictability analysis

evolving behavior patterns. In addition, behavior changes when an event happens such as sales promotions doesn't mean there is an anomaly. Therefore, we introduce a context-aware dynamic K-NN classifier both on his own behavior data and similar user's data to detect behavior anomalies.

III. The Architecture of AIMS

The framework of AIMS is illustrated in Fig. 2. It consists of several components, in which different algorithms are embedded.

A. Preprocessor

A user's historical behavior series is firstly converted to series according to different time units. User's behavior, which could sometimes be cyclical or have some short-term trends. For a pattern, daily, weekly, semi-monthly, monthly, quarterly or yearly time unit could be the length of the period. For API invocation, Daily, weekly, semi-monthly and monthly are considered the proper time-slot for making atomic event series.

Moreover, not all parts of the atomic event series are useful. A filter is implemented to extract active series at each time-span.

B. Invocation Behavior Prediction

A Group of time series predicting model are implemented, but it is impossible to go through all of them for each single series. The active atomic event series of each time-span are await here for the prediction analysis. Since not all the series could have the value of predicting, the implementation of permutation entropy, randomness detection and stationarity evaluation are used to see if the series could suit any model and which model could perform better.

C. Anomaly Invocation Behavior Detection

A context-aware K-NN classifier is used in this part. The similarity of the user's current pattern together with his historical patterns is determined. Then, the neighbors of a user are discovered by the similarity of their pattern within the same period. By considering the similarity of himself or the change in his neighbors, the anomaly index is calculated. The model is repeatedly updated over time by inputting new data, as a result, it can adapt to the behavior changes of a user.

IV. API Invocation Behavior Prediction

A. API Invocation Behavior Predictability Analysis

Users API invocation behavior is not necessary regular, i.e., many users invocation behavior is unpredictable. Chaotic, random behaviors or inactive periods could be detected before putting into prediction model. For example, there is no need to detect or analyze a user who seldom submits invocations in the past year. Fig. 3 shows the process of the predictability analysis.

Preprocessing. Firstly, the user's behavior series should be converted to time series of atomic events according to different time units. That is, on daily level for example, the number of behaviors within a day is converted to a data point.

The inactive period of each user's behavior should be omitted and the active periods should be reserved into the user's behavior series. For each time point, this step is independent. The time span covered by daily active periods could be much shorter than by monthly active periods, in the case that invocations are made only on some dates of a month. A common implementation is setting a threshold for the zero-observation percentage in a period, which is described in Algorithm 1.

Fig. 4 shows a series that is stripped to active periods. The active parts are reserved and the inactive parts are removed.

Permutation entropy. In this paper, permutation entropy is used to determine the randomness of a series. However, unlike the original definition of permutation entropy [17], the equal values are kept and the stationarity is determined later.

Permutation entropy with equal value is defined as follows:

Algorithm 1 Active Period Filtering

Input: series s, threshold t
Output: a list of acive periods

function ACTIVEPERIODFILTER(s, t)
  $ei \leftarrow$ LENGTH(s) -1
  while $ei > 0$ **and** $s[ei] == 0$ **do**
    $ei \leftarrow ei - 1$
  end while
  $si \leftarrow ei$
  $nc \leftarrow zc \leftarrow 0$
  repeat
    if $s[si] > 0$ **then**
      $nc \leftarrow nc + 1$
    else if $zc \div nc > t$ **then**
      $inner \leftarrow$ ACTIVEPERIODFILTER($s[: si], t$)
      return $inner$. APPEND($s[si : ei]$)
    else
      $zc \leftarrow zc + 1$
    end if
  until $si < 0$
  return $[s[: ei]]$
end function

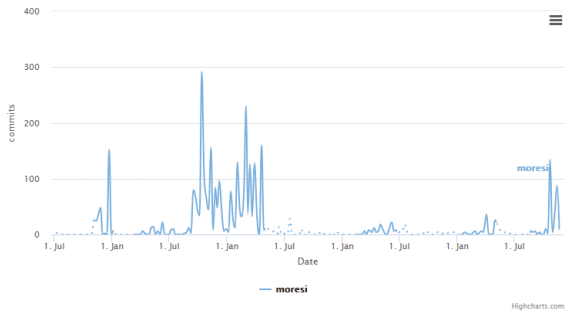

Figure 4. Example of a series after filtering inactive periods

Consider a time series $\{x_1, \ldots, x_T\}$, for every pair x_t and $x_{t+1}, (t, t+1 \in \{1, ..., T\})$, there could be three possible operators: $x_t < x_{t+1}$, $x_t = x_{t+1}$ and $x_t > x_{t+1}$. For order n, there could be 3^{n-1} permutations π. For each π the relative frequency is calculated by Eq. (1):

$$p(\pi) = \frac{\#\{t|0 \leq t \leq T - n, (x_{t+1}, ..., x_{t+n}) \text{ has type } \pi\}}{T - n + 1} \tag{1}$$

And the permutation entropy of order $n \geq 2$ is defined in Eq. (2):

$$H(n) = -\sum p(\pi) \log_2 3n - 1 \tag{2}$$

It is easy to see that (3):

$$0 \leq H(n) \leq log_2 3n - 1 \tag{3}$$

Where the lower bound is reached for an absolutely increasing or decreasing or equal sequence of values, and the upper bound is reached for all of the 3^{n-1} permutations occurs once in the series.

Stochasticity detection. A pure white noise time series has a constant mean and variance, independently distributed and nonauto-correlated. Since the last observation is to predict, the overall randomness of the last few observations are taken into consideration. In this paper, Ljung-Box test is selected for the number of observations is rather small.

The Ljung-Box test consists of two hypotheses to be tested:

$H0$: The data within h lags are independently distributed.

$H1$: The data within h lags are not independently distributed, and there exists serial correlation. The test statistic Q is defined in (4):

$$Q = n(n+2) \sum_{k=1}^{h} \frac{\hat{\rho_k}^2}{n-k} \tag{4}$$

Where n is the sample size,$\hat{\rho_k}$ is the sample autocorrelation at lag k and h is the tested number of lags.

The next step is to determine n. As an empiric value, daily records should have no less than 7 lags; weekly, semi-monthly and monthly records should have at least 3 lags. The largest n for each time-slot that satisfies 95% confidence interval is selected. Otherwise, the null hypothesis should be accepted.

Stationarity detection. There are different types of stationarities:

1) Strict Stationary: The mean, variance and covariance of a series are not function of time.

2) Trend Stationary: A series has no unit root but exhibits a trend. Once the trend is removed, the resulting series will be strict stationary.

3) Difference Stationary: A series can be made strict stationary after a differentiating calculation is applied.

The two common ways to determine whether a given series is stationary or not are by visual test and by statistical test. An implementation of unit root stationary test is adopted in AIMS.

The most commonly used unit test detection methods are ADF and KPSS. They focus on different aspects and do not conflict with each other. By augmenting Dickey-Fuller test (ADF), it could judge if a series is difference stationary. Hence, another method called KPSS can be used to make up. Kwiatkowski–Phillips–Schmidt–Shin (KPSS) tests are used for testing if a series is stationary around a deterministic trend (i.e. trend-stationary) against the alternative of a unit root.

For the ADF test, two hypotheses are:

$H0$: There exists a unit root (value of $a = 1$), so that the series is not stationary.

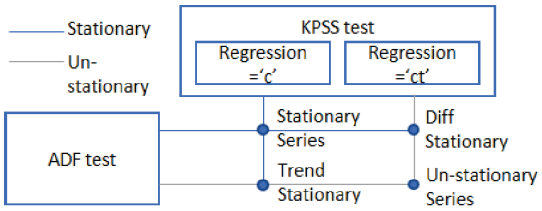

Figure 5. Cross Validation Results of Stationarity Tests of ADF & KPSS

$H1$: Consider the series does not have a unit root. The series could be linear or difference stationary.

While in the KPSS test, two hypotheses are:

$H0$: The series has a trend, so is stationary or trend stationary.

$H1$: The series has a unit root, so is not stationary.

By implementing both ADF and KPSS test, AIMS could easily detect that whether a series is stationary, and at what extend it is stationary, so as to reduce attempts of all the prediction models blindly, shown in Fig. 5.

B. An Adaptive Predicition Approach

We use an adaptive prediction model to deal with different types of series. The corresponding prediction method for the series could be determined by the following way:

Firstly, to check the length of the active period of a user's behavior series. If the length is too short to use any model, all the previous observations of the active series are used to predict the observation. Especially, if $l = 1$, the series is aborted.

The second step is permutation entropy analysis, the value of $H(n)$ should be determined. As an experimental value, on daily level, $H(2)$ and $H(7)$ are considered, elsewise $H(2)$ and $H(4)$. For those series have unstable permutation entropy, should be throw and use a K-means method or the mean value of method, depending on the last active length l.

K-means. The K-means method uses the last few observations correspondingly on weekly, semimonthly or monthly level. On daily or hourly level, the last observations on the border level are used. For example, if we need to predict the value of Monday or 3 P.M., the observation at Monday of last few weeks or at 3 P.M of last few days should be taken into consideration.

If the series is mainly white noise in the last few lags, which consists of random behaviors and has a constant mean and variance. Then K-means or KNN could be the best methods, depends on the last active length l.

KNN. The KNN finds the most similar patterns of the last observations $\{x_{i-k}, \ldots, x_{i-1}\}$ from the series, which are regarded as the neighbors of the last observations. The prediction result is the weighted average of the neighbors' next data points.

If the observations value in the active period are not white noise. The classes of ARIMA models can be used.

ARIMA. The $ARIMA(p, d, q)$ (Auto Regressive Integrated Moving Average) is a combination of $AR(p)$ model, $I(d)$ model and $MA(q)$ model. The $AR(p)$ model means the current observation is the function of the last p history observations, which is autoregressive. The $I(d)$ model means the input series could be integrated by time with d steps. The $MA(q)$ model means the current observation relies on the prediction error of the last q observations. ARIMA has various models that could deal with those series that are obvious stationary, which is suitable in predicting user's behavior.

1) The series is judged stationary by ADF and KPSS test, an $ARMA(p, q)$ ($d = 0$, the ARIMA degenerates to ARMA) model is derived to fit data and find the most appropriate p, q values.
2) The series is difference stationary, which means the series could be made stationary after steps of integrations. The $ARIMA(p, d, q)$ is seleted.
3) The series is trend stationary when the mean of the observations is not constant. It could be seasonal that evolves according to a cyclical pattern, or just has a linear trend. A seasonal $ARIMA(p, d, q)$ or trend $ARIMA(p, 1, q)$ is chosen.
4) When the series is non-stationary, K-means or KNN can be adopted, depending on the last active length l.

V. API Invocation Behavior Anomaly Detection

Anomaly detection is an important issue in every domain. In the aspect of behavioral time series, the order of the data points is the most important feature, hence should be taken into first concern. The anomaly events could be further divided into outlier, abnormal pattern and abnormal series:

Outlier. The atomic event that contains anomaly characteristic or is away from other events on the same time series.

Pattern Anomaly. The pattern (usually contains more than one atomic event) that contains anomaly characteristics or appears not the same with other patterns on the same time series.

Series Anomaly. A series that have very different forms compared by other series in the set. The difference of the series is usually caused by distinct motivations.

In this section, a method that can detect three forms of anomalies, i.e., outlier (it can be regarded as an anomaly pattern with $length = 1$), anomaly pattern and series anomaly, is described.

User's behavior changes with time. The obvious change should be a suspicious pattern anomaly. For example, people rarely work on Christmas, no matter how they behave on normal days. We could not say one who is active on normal days acts abnormally when he is inactive on Christmas, for the people's behaviors may change when an event happens. These false positive errors can be avoided by comparing the

behaviors of people that have similar behaviors on normal days.

The user's data is firstly converted to a series of atomic events in terms of a time unit and striped into series according to the pattern length. Note X_i is the $i-th$ series of a user, and $\overline{X_i}$ is the average value of each atomic event in the series, defined in (5) and (6) respectively. Suppose the length of the pattern is pl.

$$X_i = \{x_{i_1}, x_{i_2}, ..., x_{i_{pl}}\} \tag{5}$$

$$\overline{X_i} = \{avg(x_{1_1}, x_{2_1}, ..., x_{i_1}), ..., avg(x_{1_{pl}}, x_{2_{pl}}, ..., x_{i_{pl}})\} \tag{6}$$

The similarity between the ith series and the previous series is determined by the Euclidean distance of X_i and $\overline{X_i}$, shown in Eq. (7):

$$R_i = \frac{1}{1 + Eu(X_i, overlineX_i)}, R_1 = 1 \tag{7}$$

Notice that $R_i \in (0, 1]$. Since the method is context-aware, the effect of time is taken into consideration. Normally, a user's behavior series last week could be more deterministic than the series last year, thus the series of the past should have an exponential attenuation coefficient to adjust their effects. Notice there is a common knowledge shown in (8):

$$\lim_{i\to\infty}(ea-1)(\frac{1}{ea} + \frac{1}{ea^2} + ... + \frac{1}{ea^i}) = 1 \tag{8}$$

A brief application is that the self-anomaly index follows the exponential attenuation n of the first $i-th$ similarity values, shown in Eq. (9):

$$I_i = (ea-1)(\frac{R_i}{ea} + \frac{R_{i-1}}{ea^2} + ... + \frac{R_1}{ea^i}) = \frac{I_{i-1} + (ea-1)R_i}{ea} \tag{9}$$

Similarly, the neighbor-anomaly index also follows the exponential attenuation of the first $i-th$ similarity values. Note the neighbors of X_i are $Y_{i,j}$, which are defined by (10), (11):

$$Y_{i,j} = \{y_{i_1,j}, y_{i_2,j}, ..., y_{i_{pl},j}\} \tag{10}$$

$$\overline{Y_{i,j}} = \{avg(y_{1_1,j}, y_{2_1,j}, ..., y_{i_1,j}), ..., avg(y_{1_{pl},j}, ..., y_{i_{pl},j})\} \tag{11}$$

So that the neighbor-anomaly index is calculated by Eq. (12):

$$I_{i,j} = (ea-1)(\frac{R_{i,j}}{ea} + ... + \frac{R_{1,j}}{ea^i}) = \frac{I_{i-1,j} + (ea-1)R_{i,j}}{ea} \tag{12}$$

where the $R_{i,j}$ is calculated by Eq. (13):

Table I
STEPS OF CALCULATION $Sim(X_i, Y_{i,j})$

	1	**2**	**3**	**4**	**5**	**6**	**7**	**avg**
x_i	16	20	6	2	10	0	4	8.28
$dist(x_i, avg(x_i))$	$>$	$>$	$<$	$<$	$>$	$<$	$<$	
$y_{i,j}$	29	91	25	20	27	0	1	27.5
$dist(y_{i,j}, avg(y_{i,j}))$	$>$	$>$	$<$	$<$	$<$	$<$	$<$	
$dist$	0	0	0	0	2	0	0	

Table II
PARAMETERS OF METHOD POST

Name	**Type**	**Description**
user	string	**Required**. The action maker.

$$R_{i,j} = \frac{1}{1 + \sqrt{Sim(X_i, Y_{i,j})}} \tag{13}$$

where $Sim(X_i, Y_{i,j})$ depends on the edit distance between the relationship between X_i and $\overline{X_i}$ and the relationship between $Y_{i,j}$ and $\overline{Y_{i,j}}$. Take $l = 7$ for instance, Tab. I shows how $Sim(X_i, Y_{i,j})$ is calculated.

The recursive function for calculating I_i and $I_{i,j}$ follows algorithm 2. (Take $n = 2$.)

The anomaly index of a user is thus generated by Eq. (14), consisting of the changing of self-anomaly index and the retention of his nearest neighbors, according to the neighbor-anomaly index.

$$I = \Delta I_i + NeighborsChanged(\Delta I_{i,j}) \tag{14}$$

VI. IMPLEMENTATIONS AND EXPERIMENTS

A. Implementations

We have implemented AIMS in python Tornado framework and uses Tornadb for storage. AIMS online is a web service accommodating the full process from receiving data to prediction and anomaly detection.

- The user invocation records could be obtained directly from a web service vendor by adding a POST method once the invocation happens. Tab. II shows the parameters of POST:

He should specify the hashed client_id and data, which contains datetime, the number of invocations, and the method invocated in a json body. Moreover, the data_type he should be specified in the body.

AIMS online follows a five-second heartbeat to deal with user invocation records received asynchronously to solve the resource consumption problem.

- Tab. III shows the parameters of GET method:

The UI interface for parameter settings and the confirmation messages are shown in Fig. 6.

AIMS supports predicting the next data point by now. Fig. 7 shows the result of prediction.

- The parameters of detect method is described in Tab. IV:

Algorithm 2 Self-anomaly and Neighbor-anomaly Index Calculation

Input: series s, mean series ms, pattern length pl
Output: the self-anomaly index
function SAINDEX(s, ms, pl)
  $c \leftarrow$ LENGTH(s)$\div pl$
  if $c == 1$ **then**
    return 1
  end if
  $res \leftarrow 0$
  $res \leftarrow res$+SAINDEX($s[:-pl], ms[:-pl], pl$)$\div 2$
  $res \leftarrow res$+SADIST($s[-pl:], ms[-pl:], pl$)$\div 2$
  return res
end function
Input: series si, mean series ms, pattern length pl
Output: the self-anomaly distance
function SADIST(si, ms, pl)
  $dist \leftarrow 0$
  for $s, m \in si, ms$ **do**
    $dist \leftarrow dist$+POW($s - m, 2$)
  end for
  $res \leftarrow 1 \div$ (SQRT(dist)$\div pl + 1$)
  return res
end function
Input: series s, neighbor series ns, pattern length pl
Output: the neighbor-anomaly index
function NAINDEX(s, ns, pl)
  $c \leftarrow$ LENGTH(s)$\div pl$
  if $c == 1$ **then**
    return NADISTANCE(s, ns, pl)
  end if
  $res \leftarrow 0$
  $res \leftarrow res$+NAINDEX($s[:-pl], ns[:-pl], pl$)$\div 2$
  $res \leftarrow res$+NADIST($s[-pl:], ns[-pl:]$)$\div 2$
  return res
end function
Input: series s, neighbor series nsi
Output: the neighbor-anomaly distance
function NADIST(si, nsi)
  $dist \leftarrow 0$
  $ms \leftarrow$ MEAN(si)
  $mns \leftarrow$ MEAN(nsi)
  for $s, ns \in si, nsi$ **do**
    if $s > ms$ **then**
      $dist \leftarrow dist + 1$
    else if $s < ms$ **then**
      $dist \leftarrow dist - 1$
    end if
    if $ns < mns$ **then**
      $dist \leftarrow dist + 1$
    else if $ns > mns$ **then**
      $dist \leftarrow dist - 1$
    end if
  end for
  return $1 \div$ (SQRT($dist$)+1)
end function

Table III
PARAMETERS OF METHOD PREDICT

Name	Type	Description
user	string	**Required**. The user to predict.
cycle	string	If the vender knows the client's behavior pattern cycle, please specifies! Elsewise, AIMS should follow the experimental time intervals: daily, weekly, semimonthly and monthly.
period	string	Follow YYYY-MM-DD-YYYY-MM-DD pattern if needed. The last active period by now is selected by default.
zero-ptg	decimal	Indicates how inactive the user is could be accepted. The default value is 0.25.

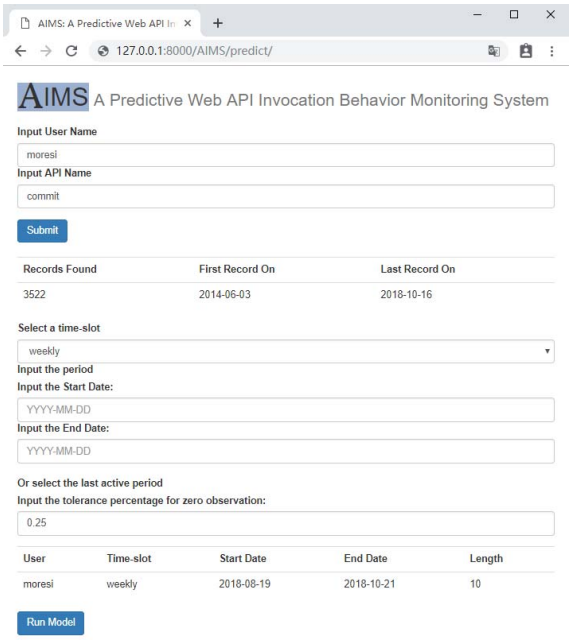

Figure 6. UI for AIMS' Prediction

B. Experiments

In this paper, an experiment on user's "commit" API invocations on GitHub [18], the world's leading software development platform, is performed.

Commit is one of the most common behaviors of developers for which the corresponding API is invoked when a commit is submitted in GitHub. Developers may make a pull request to participate a public repository, or just upload and backup his own code through commits.

Table IV
PARAMETERS OF METHOD DETECT

Name	Type	Description
user	string	**Required**. The user to detect.
n	int	**Required**. The number of neighbors
predict	Bool	If True, the approximate base on similar user's approximate result or historical result is returned.

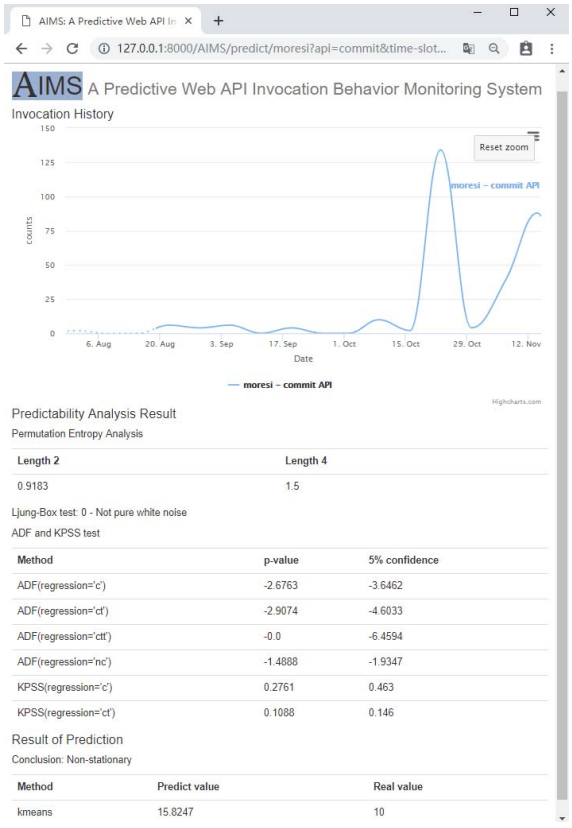

Figure 7. Result page of AIMS' Prediction Method

Sampling. Till October 16, 2018, there are more than 24 million active users on GitHub. It is impossible to settle a general survey, so the sample size should be determined.

The sample size to investigate the developers' commit API invocation behaviors can be determined by Eq. (15):

$$\frac{1}{n_0} = \frac{1}{N} + \frac{d^2}{z_{\frac{\alpha}{2}}^2 S^2} \tag{15}$$

where N is the overall size, $1-\alpha$ is the confidence, d is the absolute error limit, that is, a relative error limit s_0 on the overall mean, and S^2 is the overall variance.

Actually, not all the users on GitHub invoke commit API. During the sampling process, the users that never invoke commit API before October 16, 2018 account to 73.56%.

After eliminating these users that never invoked commit API, there are still about 6.34 million users. Set $\alpha = 0.95$, so that $z_{\frac{\alpha}{2}} = 1.96$. The next step is to determine the overall variance and the mean value of user's commit API invocation times according to sample variance and sample mean. Fig. 8 shows the approximate variance with n_0 grows.

By multiple sampling, S^2 is approximately 706581381, and the overall mean is 1097.7. The relationship between n and s_0 is shown in Tab. V.

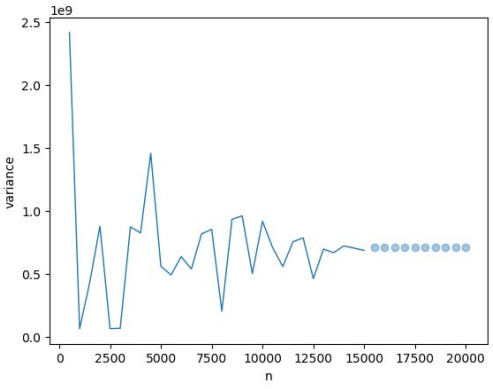

Figure 8. Relationship between n_0 and the variance

Table V
RELATIONSHIP BETWEEN s_0 & n_0

s_0	0.2	0.3	0.4	0.5
n_0	24735	10824	6093	3900

Dataset for prediction. At this stage, 6693 samples are collected. The relationship between the numbers of all the active atomic event series and the threshold t is shown in Tab. VI.

In this experiment, $t = 0.25$ is chosen. The result of the permutation entropy tested on the dataset is presented in Fig. 9. According to the permutation entropy distribution, a threshold of 30% is selected for determining whether the last active period of the user's "commit" API invocation series worth to be predicted or not.

The last active period of each time unit and each user is selected according to the threshold $t = 0.25$ in the experiment.

Dataset for anomaly detection. From the 6693 samples, about 100 samples are selected for being active in each of the three periods 2015.02.02 to 2015.04.26, 2016.10.10 to 2016.12.18 and 2018.07.23 to 2018.09.30. The pattern length is $pl = 7$, and the exponential attenuation $ea = 2$. The 9^{th} and 10^{th} series are considered. Since the dataset is unlabeled, we modified 30% of the samples' 10^{th} series by other's behavior data, and are labeled as the positive samples, which represents abnormal behavior data. The rest remains to be negative samples.

Table VI
RELATIONSHIP BETWEEN t & NUMBER OF ACTIVE SERIES

t	0.1	0.25	0.5
Daily	34435	29995	18819
Weekly	12551	10487	7203
Semi-Monthly	8464	7276	5350
Monthly	6121	5405	4221

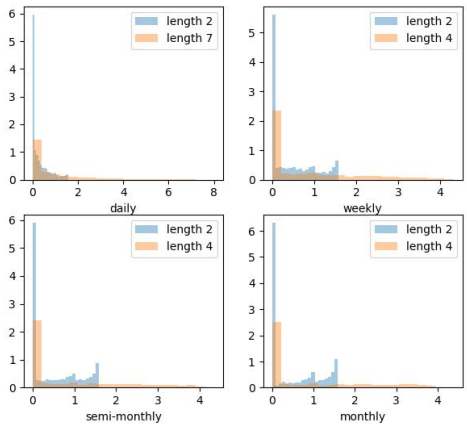

Figure 9. Distribution of Permutation Entropy

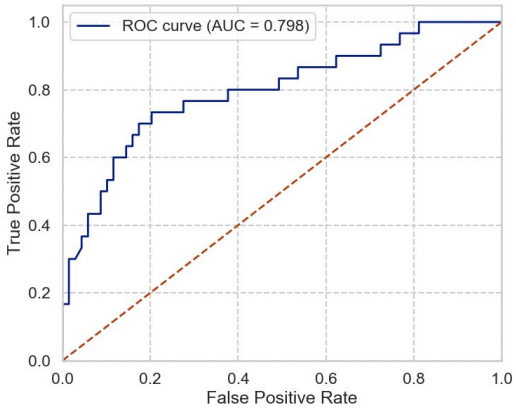

Figure 10. ROC curve

C. Evaluation

Prediction. MSE (Mean Square Error), RMSE (Rooted Mean Square Error) and MRE (Mean Absolute Error) are selected to be the metrics of the evaluation, their definitions are described in Eq. (16), (17) and (18) respectively.

$$MSE = \frac{1}{m}\sum_{i=1}^{m}(y_i - \hat{y_i})^2 \tag{16}$$

$$RMSE = \sqrt{\frac{1}{m}\sum_{i=1}^{m}(y_i - \hat{y_i})^2} \tag{17}$$

$$MAE = \frac{1}{m}\sum_{i=1}^{m} \mid y_i - \hat{y_i} \mid \tag{18}$$

Where m stands for the number of samples, y_i, $\hat{y_i}$ stands for the real and predicted value of each sample.

Tab. VII shows the MSE, RMSE and MAE results in terms of different time units and through different prediction methods. AIMS is stable, even if human's behavior is sometime elusive. The reason is that AIMS selects appropriate prediction method to adapt to different user's invocation behaviors.

Anomaly detection. ROC (Receiver Operating Characteristic curve) is selected to be the evaluation metric, which uses the False Positive Rate and the True Positive Rate, calculated by $FP(FalsePositive)/N(Negative)$ and $TP(TruePositive)/N$. The curve of ROC is shown in Fig. 10. The AUC value is 0.798, which means AIMS has a 79.8% chance to distinguish the positive and negative classes.

VII. Conclusion

Monitoring and predicting personal Web API invocation behaviors are of great value for both service providers and service clients. However, different users have very different API invocation behavior patterns. In the paper, we introduce a predictive Web API Invocation Behavior Monitoring System (AIMS), which can select the prediction approach in terms of the characteristics of each individual's invocation behaviors. Moreover, a context-aware anomaly detection approach is also deployed into the AIMS which can detect different forms of anomaly. Experiments on both a real-world dataset and synthetic dataset show AIMS is efficient. We will further improve AIMS by incorporating more prediction models especially non-linear models such as neural networks. In addition, for each API, the input data of each invocation can also be analyzed so that to understand the behaviors more deeply.

Acknowledgment

This work is supported by National Key Research and Development Plan(No. 2018YFB1003800).

References

[1] L. Richardson and S. Ruby, "Restful web services: O'reilly media," *Incorporated*, 2007.

[2] M. Maleshkova, C. Pedrinaci, and J. Domingue, "Investigating web apis on the world wide web," in *2010 Eighth IEEE European Conference on Web Services*. IEEE, 2010, pp. 107–114.

[3] W. Tan, Y. Fan, A. Ghoneim, M. A. Hossain, and S. Dustdar, "From the service-oriented architecture to the web api economy," *IEEE Internet Computing*, vol. 20, no. 4, pp. 64–68, 2016.

[4] L. Rao, "Twitter seeing 6 billion api calls per day, 70k per second," Website, 09 2010. [Online]. Available: https://techcrunch.com/2010/09/17/twitter-seeing-6-billion-api-calls-per-day-70k-per-second/

[5] Amazon, "Amazon api gateway," Website. [Online]. Available: https://aws.amazon.com/api-gateway/

Table VII
RESULTS OF MSE, RMSE & MAE OF DIFFERENT TIME UNIT THROUGH EACH PREDICTION METHOD

		AIMS	mean	k-nn	k-means	arima	trend-arima	diff-arima
MSE	**Daily**	52.48	49.14	91.29	91.42	88.33	88.77	91.48
	Weekly	665.74	1093.38	2389.11	725.61	1195.28	1163.22	1190.13
	Semi-Monthly	1820.99	23625.57	19285.45	1991.09	23553.23	26320.95	26432.45
	Monthly	9923.76	14070.36	21076.06	11103.39	16688.66	18407.05	19252.18
RMSE	**Daily**	7.24	7.01	9.55	9.56	9.40	9.42	9.56
	Weekly	25.80	33.07	48.88	26.94	34.57	34.11	34.50
	Semi-Monthly	42.67	153.71	138.87	44.62	153.47	162.24	162.58
	Monthly	99.62	118.62	145.18	105.37	129.18	135.67	138.75
MAE	**Daily**	3.44	3.44	4.54	4.47	4.57	4.60	4.78
	Weekly	5.44	6.21	8.34	6.95	7.56	7.77	8.52
	Semi-Monthly	7.89	11.58	13.28	10.50	13.26	13.77	14.93
	Monthly	13.46	14.96	19.54	18.23	17.34	18.17	20.29

[6] IBM, "Ibm api connect," Website. [Online]. Available: https://www.ibm.com/cloud/api-connect

[7] M. H. Hasan, J. Jaafar, and M. F. Hassan, "Monitoring web services' quality of service: a literature review," *Artificial Intelligence Review*, vol. 42, no. 4, pp. 835–850, 2014.

[8] P. Zhang, H. Jin, Z. He, H. Leung, W. Song, and Y. Jiang, "Igs-wbsrm: A time-aware web service qos monitoring approach in dynamic environments," *Information and software technology*, vol. 96, pp. 14–26, 2018.

[9] H. Ma, H. Zhu, Z. Hu, W. Tang, and P. Dong, "Multi-valued collaborative qos prediction for cloud service via time series analysis," *Future Generation Computer Systems*, vol. 68, pp. 275–288, 2017.

[10] H. Wu, K. Yue, C.-H. Hsu, Y. Zhao, B. Zhang, and G. Zhang, "Deviation-based neighborhood model for context-aware qos prediction of cloud and iot services," *Future Generation Computer Systems*, vol. 76, pp. 550–560, 2017.

[11] C. Yu and L. Huang, "A web service qos prediction approach based on time-and location-aware collaborative filtering," *Service Oriented Computing and Applications*, vol. 10, no. 2, pp. 135–149, 2016.

[12] K. Su, B. Xiao, B. Liu, H. Zhang, and Z. Zhang, "Tap: A personalized trust-aware qos prediction approach for web service recommendation," *Knowledge-Based Systems*, vol. 115, pp. 55–65, 2017.

[13] J. K. Aggarwal and M. S. Ryoo, "Human activity analysis: A review," *ACM Computing Surveys (CSUR)*, vol. 43, no. 3, p. 16, 2011.

[14] K. Xu, Z.-L. Zhang, and S. Bhattacharyya, "Internet traffic behavior profiling for network security monitoring," *IEEE/ACM Transactions on Networking (TON)*, vol. 16, no. 6, pp. 1241–1252, 2008.

[15] J. S. Gosavi and V. S. Wadne, "Unsupervised distance-based outlier detection using nearest neighbours algorithm on distributed approach: Survey [j]," *International Journal of Innovative Research in Computer and Communication Engineering*, vol. 2, no. 12, pp. 7510–7514, 2014.

[16] E. Schubert, A. Zimek, and H.-P. Kriegel, "Fast and scalable outlier detection with approximate nearest neighbor ensembles," in *International Conference on Database Systems for Advanced Applications*. Springer, 2015, pp. 19–36.

[17] C. Bandt and B. Pompe, "Permutation entropy: a natural complexity measure for time series," *Physical review letters*, vol. 88, no. 17, p. 174102, 2002.

[18] Github, "Github," Website, 2019. [Online]. Available: https://www.github.com