An Intelligent Anomaly Detection Scheme for Micro-services Architectures with Temporal and Spatial Data Analysis

Yuan Zuo, Yulei Wu, Geyong Min, Chengqiang Huang, and Ke Pei

***Abstract*—Service-oriented 5G mobile systems are commonly believed to reshape the landscape of the Internet with ubiquitous services and infrastructures. The micro-services architecture has attracted significant interests from both academia and industry, offering the capabilities of agile development and scale capacity. The emerging mobile edge computing is able to firmly maintain efficient resource utility of 5G systems, which can be empowered by micro-services. However, such capabilities impose significant challenges on micro-services system management. Although substantial data are produced for system maintenance, the interleaved temporal-spatial information has not been fully exploited. Additionally, the flooding data impose heavy pressures on automatic analysis tools. Automated digestion of data is in an urgent need for system maintenance. In this paper, we propose a new learning-based anomaly detection framework for service-provision systems with micro-services architectures using service execution logs (temporally) and query traces (spatially). It includes two major parts: logging and tracing representation, and two-stage identification via a sequential model and temporal-spatial analysis. The experimental results show that the temporal-spatial features can accurately capture the nature of operational data. The proposed framework performs well on anomaly detection, and helps gain in-depth insights of large-scale systems.**

***Index Terms*—Service-oriented 5G systems, micro-services management, log analysis, representation learning, anomaly detection**

I. Introduction

The fifth generation mobile system (5G) has drawn substantial attention, thanks to its better performance over the fourth generation system (4G) for realizing the next generation of telecommunication systems [1]. The 5G system is reasonably deemed as the versatile services enabled large-scale architecture through ubiquitous infrastructures from the service-oriented perspective [2], [3]. The service-oriented architecture (SOA) helps decouple complex services into separated but intercommunicated components [4]. Recently, a newly-architectural style, called micro-services architecture, has gained large popularity, which makes use of lightweight communication style to decouple closely related and overlapped services [5]. Even though the sceptics claimed that the micro-services scheme only accounts for a sub-type of SOA with web service communication prototypes [6], it is no doubt that systems, including 5G, substantially benefit from shifting to micro-services [7], [8]. The improved traceability and scalability enable micro-services to handle complicated, decentralized and modular service platforms. However, scalable and lightly-coupled distributed systems largely increase the difficulty and complexity of their monitoring, maintenance and management, due to interleaved data and the ambiguous communication processes [6]. Another problem is that the inner mechanisms for most of the commercial off-the-shelf platforms are black-box to users and administrators. Although the open-source software prevails today, it is still not easy to have a thorough knowledge of understanding these software. Therefore, it requires well devised mechanisms integrated into the system to well track the execution status, and well maintain functionality and stability [9].

The purpose of this paper is for intelligent network management [10], [11], [12], [13] of micro-services through learning cognitive knowledge from enormous and multi-source networking data. Specifically, it is to identify suspicious events based on the available execution status, such as service execution logs and service query traces, and to raise necessary alerts to avoid catastrophic failures for a system. The idea consists of two parts: using service execution logs to capture functional behaviors in a temporal manner; using query traces to capture systematic behaviors in a spatial manner. Combining them is supposed to enable expected detection conduction, and our experiment results well validate this hypothesis.

To meet the above requirements, the logging module has become indispensable when developing a reliable and robust system. It is because logging is able to probe record writers inside the core functional components [14]. The writers always gather designated elements and instantly store them in logs to notify IT administrators on what is happening [15]. An effective analysis of logs can reduce the difficulty of identifying abnormal system behaviors [16]. The primary matter of processing logs is to digest text formats written in a human readable way. A simple case is shown in Table I, which is a block of raw log examples collected from IBM BlueGeneL [17]. One log entry corresponds to a low level event. Compared to a high-level functionality response, like a user access query, the low-level operation only involves fine-grained actions, thus called *meta-execution* in this work. A method of better understanding a system is to mine the underlying patterns of chaotic messages to group into structural templates. This method is called *template extraction*; its details will be elucidated in

Y. Zuo, Y. Wu, and G. Min are with the College of Engineering, Mathematics and Physical Sciences, University of Exeter, Exeter, EX4 4QF, UK. (Corresponding authors: Y. Wu, G. Min.)

C. Huang, and K. Pei are with RAMS Lab, Huawei Technologies Co. Ltd., Shenzhen, 518129, China.

E-mail: yz506@exeter.ac.uk, Y.L.Wu@exeter.ac.uk, G.Min@exeter.ac.uk, huangchengqiang@huawei.com, peike@huawei.com.

TABLE I: Raw log message samples collected from BlueGene/L supercomputer

Log head entry	Log message entry
2005-06-05-07.43.29.199690 R35-M0-N1-C:J02-U01 RAS KERNEL INFO	instruction cache parity error corrected
2005-06-05-07.43.29.223683 R35-M1-N9-C:J03-U01 RAS KERNEL INFO	instruction cache parity error corrected
2005-06-07-22.17.51.624699 R03-M0-N7-C:J02-U01 RAS KERNEL INFO	instruction cache parity error corrected
2005-06-07-22.17.51.651067 R06-M1-NC-C:J11-U01 RAS KERNEL INFO	generating core.53489
2005-06-07-22.17.51.677398 R06-M1-NC-C:J06-U01 RAS KERNEL INFO	generating core.53489
2005-06-07-22.17.51.703324 R06-M1-NC-C:J02-U11 RAS KERNEL INFO	disable store gathering.................0
2005-06-07-22.17.51.728819 R05-M0-N1-C:J17-U11 RAS KERNEL INFO	CE sym 2, at 0x0b85ea80, mask 0x08
2005-06-07-22.17.51.754779 R05-M0-N1-C:J08-U11 RAS KERNEL INFO	Node card is not fully functional
2005-06-07-22.17.51.784808 R05-M0-N1-C:J08-U01 RAS KERNEL INFO	CE sym 21, at 0x110035e0, mask 0x80
2005-06-07-22.17.51.850973 R14-M1-NF-C:J03-U11 RAS KERNEL INFO	CE sym 2, at 0x0b85ea80, mask 0x08

Section IV.

Only using *meta-execution* logs could lose system-scale interaction information. This interaction in micro-services is vital, since lightly-coupled services depend on their back-end query responses. A distributed tracing mechanism is of essence to keep track of the interaction behaviors [18]. It indicates that the presence of hidden communication issues can affect the interaction duration. The authors in [19] discussed the fundamental design and deployment factors of distributed tracing mechanisms, where this tracing mechanism should impose less burden on the current system.

The tracing mechanism is helpful on system management, while the current works lack full exploitation of this mechanism [16]. The lack of awareness of connection and coupling status may give rise to inconsistency and instability. Combining the above two types of information (i.e., service execution logs and query traces) can lead to a comprehensive and robust anomaly detector. We presume that the input data show no direct sign of errors and faults, for example, log severities being *info* and *warning*. The failures may not occur instantly but hide the symptoms in a number of service executions. Therefore, we propose an anomaly detection framework from front-end data collection, feature construction to back-end outliers classification. The overview of this new framework is presented in Section III. The main contributions of this paper are summarized as follows:

- We propose a general-purpose anomaly detection framework with two-stage validation targeting at micro-services architectures, using temporal service execution logs and spatial service query traces.
- An efficient and on-line template extraction method for log messages written by natural languages is proposed to obtain service execution behaviors, based on iterative partitioning and segment adjustment.
- We propose a temporal data learning method with sequential deep learning model to predict the deviation between normal expectations and real outputs with corresponding anomaly degree calculation. This process forms the first phase of the two-stage validation and manifests potential abnormal candidates.
- We propose a temporal and spatial data learning method, implementing behaviors and queries representative features to unveil hidden systematic status. The fusion of two representations is taken into effect at the following anomaly detector based on a one-class classifier, which is able to discover insights from empirical data. This process forms the second phase and jointly determines the abnormal tasks with anomaly candidates.

The rest of this paper is organized as follows: Section II introduces related work. Section III describes the proposed framework, followed by the template extraction in Section IV, temporal log model in Section V, and temporal-spatial joint analysis in Section VI. Section VII analyzes experimental results, and Section VIII draws the conclusions.

II. Related Work

In this section, the related work, including template extraction, document representation, tracing system, and learning based log anomaly detection are reviewed.

A. Template Extraction

There have been numerous research efforts on log template extraction. At an early stage, statistics-based methods are of more interest. The authors in [20] made an effort to template extraction based on word frequency, called Simple Logfile Clustering Tool (SLCT). Basically, it endeavored to scan the whole log data in limited times and to collect the frequency of individual log words with their positions in a template with a particular length. The SLCT sets a frequency threshold to determine what words should be kept in that particular position. Makanju et al. [21], [22] proposed an iterative partitioning based algorithm, called Iterative Partitioning Log Mining (IPLoM) tool, gaining efficient template extraction. IPLoM consists of four steps: partitioning based on log length; partitioning based on the least count of unique items in one column; partitioning based on bijection pairing; shrinking variable terms with asterisks. IPLoM is efficient and its computation grows linearly with data size, however the bijection partitioning gives rise to ambiguity about very similar templates separation. To overcome these issues of IPLoM and gain efficiency in on-line working, we propose a segment-based recursive partitioning method in this work. More relevant algorithms are introduced and evaluated in [23], [24].

B. Document Representation

In document representation, the Latent Dirichlet Allocation (LDA) [25] provides a classic probabilistic way to model topics of text collections with mixtures. A numerical representation of a topic could be explicitly computed by it.

In essence, LDA remains as a hierarchical Bayesian model, whose parameters are estimated based on variational and EM methods. Recently, text representations have been improved based on machine learning, especially promoted by neural networks for word distributed representations (also called word embedding and word2vec) [26], [27], [28]. An extensive bonus of word embedding is that the representation of word collections (phrases, sentences, paragraphs, and corpora) is also stimulated by deep learning. The authors in [29] were inspired by word embedding and offered a simple and effective approach for training alongside word2vec to accomplish topic modeling. The authors in [30] and [31] focused on convolutional neural networks (CNN) to grasp a representation of a sentence, meanwhile [32] and [33], leveraged recurrent neural networks (RNN) to model word sequences. Other studies reached the same achievement, e.g., [34] and [35] by adapting deep variation inference into latent topic modeling.

C. Tracing System

A fundamental feature in this work is the tracing module in the micro-services architecture. In early research, X-trace [18] monitored general service protocols in a classical networking or a software system. A Google team in [19] deployed a tracing tool, Dapper, inserting lightweight probes in key service points in a large-scale distributed system, e.g., searching service, to collect and track a complete request tree. It shared the concepts with X-trace and dedicated to low overhead, low-level application transparency, easy deployment and scalability. Other pragmatic efforts were designed based on the Dapper tracing mechanism. For instance, the Jaeger tool [36] open-sourced by the Uber Technologies and OpenZipkin[37] originally developed by Twitter. All of them have gained popularities and reputations in large-scale distributed micro-services practices. As the counterpart in the OpenStack cloud architecture, the Osprofiler [38] is specifically developed for building a query tree. The built-in component manages to construct service-point level calling relations for authentication, imaging, computing, networking and etc., instead of programming level in python invocation.

D. Learning Based Log Anomaly Detection

For the anomalies detection in micro-services, Shilin el at. [39] conducted a comprehensive empirical evaluation based on several open-sourced log analysis toolkits, including supervised learning: logistic regression, decision tree, and support vector machine (SVM); unsupervised learning: log clustering, principal component analysis (PCA), and invariants mining. These detection algorithms all conformed to four steps framework, from log collection, log parsing, and feature extraction to the final anomaly detection with two public datasets. Yu el at. [40] aimed at building a tool called CloudSeer for cloud computing platform monitoring. They took OpenStack as the test platform. With assumed interleaved log sequences, they tended to mine dependency relation in terms of time and built an automaton machine, which shared conceptual similarity with the finite state machine. On top of deep learning, DeepLog [16] was proposed to fit a long term short memory (LSTM) model in natural language-like sequential log data. The perspective of log records from DeepLog is similar to our proposal. Both of the frameworks endeavor to model the hidden transactional patterns over abstract events that are transferred from raw messages. Nevertheless, DeepLog as well as CloudSeer merely processes pure text logs regardless of informative service traces. In addition, the authors in [41] attempted to interpret semantical implications from logging sequences by "*template2vec*", which focused on logging word embeddings and finally was extended to a log event as "*sentence*" embeddings. In our framework, we take task-bounded semantics associated with particular tasks into consideration, rather than plain logs. We further consider the text log features and traces integrated information to gain deeper insight of micro-services mechanisms.

III. An Overview of the System Architecture

We firstly introduce our proposed detection framework for the system with micro-services architectures, which include two core aspects: individual service execution and services remote query. The former refers to whether a local single service can handle an incoming task properly within a reasonable time scale. The latter refers to whether the global lightly-coupled services as a whole can process a user request smoothly in an acceptable time. The local and global functionalities can be regarded as *two-tier behavior features* being encoded into mathematical and learnable representations in a continuous vector space. To achieve the primary goal of runtime anomaly detection, this paper builds representation learning based procedures to capture temporal and spatial information. Fig. 1 illustrates the diagram of the proposed framework. The red arrows are the data processing flows.

In detail, it is divided into two main steps: data representation learning and two-stage outliters identification. Firstly, in data representation learning, temporal log sequences are collected by the embedded logging module in the micro-services architecture, and the hierarchical trace data sheets are collected by tracing probes (the probes are deployed inside the micro-services architecture). In addition, event templates are extracted from logs, and in turn chaotic logs shall be transferred into the template formation. Furthermore in representation learning, the log template formation is mapped into distributed vectors to collectively infer a "*topic*" vector for a particular transaction. In the tracing module, the tracing information is extracted from the trace data sheet which is organized in hierarchy to expose request-response time. The tracing information is a matrix, each entry of which indicates a service duration queried from a front-end service to a certain back-end service. Then, at the second anomaly detection step, it consists of two stages for anomaly detection. Stage 1 focuses on a temporal model which extracts execution sequential patterns from temporal operation histories. This stage will give out a collection of anomaly candidates by calculating anomaly degrees. Stage 2 focuses on local and global information joint detection through log representation features and tracing representation features. This stage will also allow anomaly candidates to assembly validate the existence of fault examples. The stage 2 will jointly make use of temporal and spatial

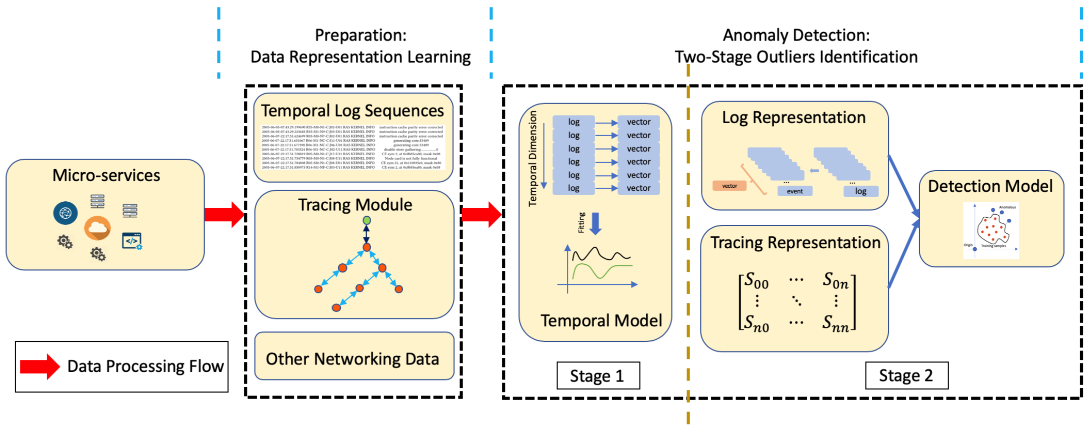


Fig. 1: The proposed framework for runtime system anomaly detection in a Micro-services architecture

information by merging the two representation features and the inputs into a detection model to learn an implicit normal sample distribution, with which the detection model keeps the capability of marking data out of the implicit distribution as anomalies.

In this work, we focus on an OpenStack cloud platform [42], which implements each service component, e.g., nova-compute, neutron-networking, and keystone-authorization, by RESTful API based on the micro-services architecture. Templates are extracted based on partitioning, and logs are vectorized by the word embedding. The tracing matrix is constructed based on the OpenStack embedded tracing mechanism, osprofiler [38]. To model the temporal characteristics of sequential execution log records, the neural networks based Long Short Term Memory (LSTM) is applied to reveal temporal dependencies. To jointly reveal temporal-spatial information, the outliers classifier employs a one-class support vector machine (SVM) as the anomaly detector.

IV. Template Parsing and Extraction

Log preprocessing is the foundation of downstream tasks, as raw log messages are unstructured, disordered and even possibly corrupted [17]. The analysis of log data is inclined to take numerical and categorical data as input, which also requires raw log information being cleaned, ordered, and normalized.

A log record[1] consists of two parts: *log head entry* and *log message entry*. For log head entry, there exit several segments, e.g., time stamps, host names, and severity of events. The specific settings depend on the operating system, but they will keep consistent with the standard and unified format across all the stored logs. In the meantime, log message entry is predefined manually by developers, which may differ significantly among systems, even inside one system.

The unstructured *log message entry* usually draws more attention to administrators, which may contain abundant fragmented clues to system status at one particular time step. The log message style can vary among systems or even inside one large-scale system. Because of the unstructured property and trivial numerical variables, it is difficult to mine coarse-grained and high-level interpretable systematic events. To form comprehensive systematic states, the work aims at mining compact, stable and interpretable message representations, called *message patterns and templates* [22]. A common idea is treating log message as two elements, *constant* expressions and *variable* values, respectively. The constant expressions are consistent repeating information body, which are usually fixed in the system source code, e.g., message wrapped as constant string-type in “print”-style functions. In contrast, the variable values are reserved entries inside the “print” functions, varying based on the incoming specific conditions, e.g., the integer type replacement as ”%d”. Therefore, the main task becomes to construct an abstraction in which the constant entries are kept, and the variables are reduced into a substitution, such as wild card “$*$” with the corresponding numbers and positions. The extracted message abstraction is called “template” or “pattern”[2].

Extraction examples based on Table I are illustrated in Table II. From Table II, five templates are extracted, wherein three of them remain the same as the raw log, and the rest two have been reduced to a compact style. With such a compressed format, the downstream task takes the extracted types as input to enable deeper analysis. As aforementioned, the raw log message refers to the unstructured part written in natural language.

In [22], the authors proposed a successful log entry partitioning based method, called IPLoM to iteratively refine abstract candidates and use invariant entry pair mapping to obtain templates. The method proposed in this section is inspired by the IPLoM. A drawback from [22] is that the IPLoM assumes that the message contains stable structures of words pairs. This assumption fails to hold if developers tend to mark divisive executions with the same syntax and prefix. The same syntax will appear in the IPLoM partitioning candidates, entail pair mapping and eventually mislead the discriminator to unify variant templates. IPLoM is at the risk of masking key information. The proposed partitioning-based extraction

[1]Some may also consider “log record” as “log entry”, therefore the two expressions will be used interchangeably.

[2]In this work, the two terminologies are used interchangeably.

TABLE II: Template/pattern examples

Raw log message part	Extracted patterns
instruction cache parity error corrected	instruction cache parity error corrected
CE sym 2, at 0x0b85ea80, mask 0x08	CE sym *, at *, mask *
generating core.53489	generating *
disable store gathering.................0	disable store gathering.................0
Node card is not fully functional	Node card is not fully functional

in the section discards the intuitive mapping and recursively partitions candidate sub-groups by the same mechanism as the first partitioning trial. As the algorithm here targets at the whole log data set, it is called the bulk-oriented recursive partitioning algorithm.

A. Bulk Recursive Partitioning

Firstly, it is broadly accepted that the raw logs which entail the identical templates would keep the same length, namely entry count. Recall that the count relates to splitting a text sentence by given separators, i.e., space. Thus, one reasonable way to reduce entire raw log intersection chaos is to coarsely segment the equal length raw logs as several sub groups. The sub groups with the equal length help identify and locate stable constants by columns alignment.

After grouping raw logs with the identical sentence length (word count separated by space), the first partitioning is based on the most stable entry position, which can be decided by the count of unique words occurring in that particular position. On top of the same sentence length, the group of messages can be aligned and fitted in a symbolic "matrix", each entry of which is a discrete word. This "matrix" is reduced into a vector with respect to column, by filling each place with the unique word count in the corresponding column. The column of the least one is chosen, where individual logs with the same entry word will be moved into one sub-group. The above operation is taken subsequently and recursively over the newly created sub-groups until a limit is reached.

That is, for one sub-group, one symbolic "matrix" is also temporally instantiated and compressed into one vector. At this point, any column already being summed up to one will be ignored to ensure that the partitioning effect will not be taken repeatedly and unnecessarily over analyzed columns. A threshold of the number of the processed columns is set to terminate the recursion process. The threshold controls how many seemingly constant words should remain to represent the event and how much information should be exposed to analyzers. The recursion process and threshold indicate the balance between minimizing cardinality of template set and maximizing the reserved information. The low threshold is inclined to extend one type as broadly as possible, meanwhile the high threshold tends to increase template diversity.

Once all the sub-groups become indivisible, the partitioning phase comes to the end, and the final step results in generating a template set or so-called template library, one template for each sub-group. The first one message will then be examined along with the unique word counts vector, and the words at any position corresponding to non-one, will be directly substituted by $*$, otherwise are kept. The template set storing all extracted templates is the ultimate outcome of the recursive processing.

Algorithm 1: Bulk-oriented Recursive Partitioning Algorithm

```
Data: Entire log message part, $R$
Parameter: Threshold, $\theta$
Result: A template library, $\zeta$
Initialize a set of candidate sets $\Omega$;
for message $l \in R$ do
    $\nu = split(l)$, via pre-defined symbols;
    $\mathcal{L} = len(\nu)$, length of the list $\nu$;
    if $\mathcal{L}$ not match any set in $\Omega$ then
        create a empty set $S_\mathcal{L}$ in $\Omega$;
        append $l$ in $S_\mathcal{L}$
    else
        append $l$ in the matched $S_\mathcal{L}$
    end
end
Initialize a group of partitioned sets $\mathcal{G}$;
for candidate set $S_\mathcal{L} \in \Omega$ do
    $\mathcal{G} = \text{RECUR\_PARTITION}(S_\mathcal{L}, \theta)$;
end
Initialize a library of templates;
for $one\_set \in \mathcal{G}$ do
    Replacing columns without count-1 unique terms
      with $*$;
    $\zeta$ appends the processed content;
end
return $\zeta$;
```

The partitioning detail is presented in **Algorithm 1**. A parameter, θ is required to determine the remaining information that is ready for downstream tasks or semantic analysis. The function *Recur_Partition* from **Algorithm 2** takes as input a group of candidate sets and recursively outputs partitioned results. Afterwards, a template library, ζ is returned and each entry of the library refers to one template.

B. Segmented Library Iteration

The bulk-oriented recursive partitioning algorithm will return satisfactory template library which is optimized against the original messages. However, one crucial disadvantage is the operation time consumption. Though there is a threshold controlling the process depth, the chaotic and fragmented log sequences will complicate the recursion process and largely reduce the efficiency. For example, because the recurring will only end until a threshold has been reached within each partitioned block, if every single message is a block,

```
Algorithm 2: Recursive Partitioning
  Data: A set of logs
  Parameter: log_subset, INFO_threshold
  Result: A template set, inner_set
  Function
   Recur_Partition($log\_subset, INFO\_threshold$):
      $split\_set \leftarrow log\_subset$ split by the column of the
        least unique terms;
      Initializing $inner\_set$ to hold partitioned set;
      for $one\_block \in split\_set$ do
          if Count 1 terms exceeds threshold then
              append the block into $inner\_set$;
              continue;
          else
              RECUR_PARTITION(one_block,
                INFO_threshold);
          end
      end
      Return $inner\_set$;
  End Function;
```

```
Algorithm 3: Segmentation and Aggregation
  Data: Entire log message part, $R$
  Parameter: Threshold, $\theta$;
             Log count in segments, $Seg\_interval$.
  Result: A template library, $\zeta$
  Separating the data set sequentially by $Seg\_interval$.
   $Seg\_list$ = split($R$);
  Initializing a list, $temporal\_lib\_list$;
  for $block \in Seg\_list$ do
      Applying Algorithm 1 to the block.
       $temporal\_lib$ = Algorithm 1($block$);
      Adding $temporal\_lib$ into $temporal\_lib\_list$.
  end
  Applying Aggregation to $temporal\_lib\_list$.
   $\zeta = Aggregation(temporal\_lib\_list)$;
  return $\zeta$;
```

the computation of log sequence processing grows rapidly. Another disadvantage is that bulk partitioning requires an off-line environment feeding the entire log dataset, which may compromise the log collection module and its efficiency.

Therefore, we propose a segmentation style log partitioning and an aggregation scheme to increase utility. The segmentation and aggregation extend the off-line scheme to not only on-line but also a parallel architecture, as the on-line scheme iterates streaming data to complete the template library, and the parallel architecture encourages local computing and low communication overhead.

The downward closure property [43] is introduced, also known as Apriori property, by which unstructured and natural language-related log messages can be compressed to compact text form. The downward closure property claims if a term set is frequent, any subset should be at least as frequent as this term set. The property makes it possible that the frequent terms are obtained in segmentation, and the shared subsets are extracted in aggregation. In the segmentation step, **Algorithm 1** is only applied to a fraction of original logs, for example 1000 samples, which is determined by a split parameter. If in an on-line manner, the fraction depends on the buffered streaming events, pre-set by a buffer parameter. Apparently, the outputs are locally optimal, since no global information is considered. The intermediate template sets are stored as inputs to the aggregation step. Subsequently, the aggregation step takes as input the gathered intermediate templates to apply **Algorithm 1** again. At this point, the intermediate templates are considered as "incomplete logs" and entail the global information to approach more abstract forms. Details are described in **Algorithm 3**.

In addition to directly applying **Algorithm 1** in an on-line scheme, it is also reasonable to consider that the constant terms could exceed a fraction of variable terms in aggregation rather than adhering to a fixed information threshold. A case is that the count of constants should not be less than a percentage of the count of variables. The modification benefits relatively long logs from the potential discovering of the templates with the majority of variables. Obviously, it provides an interface of some adjustments to flexibly tackle extreme conditions, denoted as *Aggregation* in **Algorithm 3**. We refer to this modification as *elastic aggregation*. The downward closure property guarantees that log templates will gradually converge to its optimal abstraction. It is because within a local part of logs, variables, e.g., host names, are likely remain stable and are therefore inseparable. Nevertheless, on a longer timescale, the inseparable positions become chaotic and separable in terms of unique counts. The segmentation focuses on small fractions of data to relief computing pressure, meanwhile aggregation gathers intermediate results from segmentation to approximate optimal abstraction. The two steps manage to avoid large time consumption while gaining theoretical convergence.

V. Temporal Log Features Modeling

This section presents the temporal model that learns sequential execution patterns from system operation history data in Stage 1 of Anomaly Detection processing. The conjecture is that there is a chance that faults incur disorder of logs and infrequent system execution records. Other than solely interpreting the static information, the sequential model endeavors to obtain dependency from dynamic context to predict regular and expected behaviors. The discrepancy between predictions and actual log symptoms implies hidden suspicious anomalies, and even the erroneous marks. Accordingly, the core problem is to setup a well-performed sequential model. The adopted sequential model is one of the RNN models, i.e., LSTM [44]. Moreover, the model usually predicts the future operations by giving out a ranking list that holds the probability for each potential behavior. The real data may appear in several top positions if no severe issue affects the running task. Simply setting top-k occurring positions is straightforward, however a soft and elastic score can be assigned as anomaly degrees to reflect the seriousness. Lastly, the temporal model will output abnormal sample candidates for the double validation jointly with the following outliers detector. The detail will be elucidated below.

The RNN model prevails in sequential data analysis in deep learning community [45] and can be visualized in Fig. 2, where there are two diagrams illustrating the structures of the basic forms of the RNN model, the unrolled architecture (left-sided) and the rolled architecture (right-sided), respectively. Theoretically, the processed sequences can be extended to unlimited length with rolled loop, which recurrently takes single entries as input. However, the input sequences are truncated with fixed lengths either to adapt to realizable model in physical machines or to grasp latent semantics by considering task specialization, where the unrolled form RNN takes into effect. In detail, x_t, o_t, and s_t denote the input data, output data and hidden state, respectively. The input x_t is the log templates representation learned by word embedding [27] similar to the transactional representation construction, which will be presented in Section VI-A. The templates remain the same order as the raw logs and the sequential data are turned into a sequence of distributed vectors as the time series input of the RNN. The outputs also construct a sequence that should match a given sequence of labels to guide the neural networks training direction. Generally, an input sequence and its labels pair is called a training pair. A conventional way to accomplish it is shifting an input sequence forward one time step to be its output label.

As for the cell, it is the processing core of RNN which transforms the input data into the implicit forms (hidden state s_t) in a high-dimensional vector space, and is capable of projecting the implicit forms into output representation. The left red dotted rectangle covers the unrolled structure of the RNN, and the right rectangle revealing one cell unit recurrently wraps up the previous hidden state to merge into the current processing.

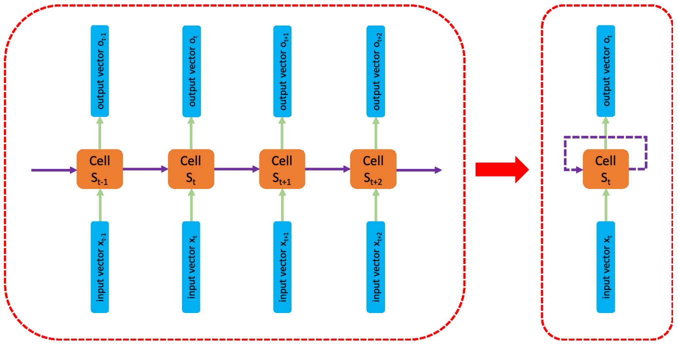


Fig. 2: Visualization of Recurrent Neural Networks Structure

The formulation of the RNN can be defined as below:

$$s_t = f_{cell}(x_t), o_t = f_{out}(s_t) \tag{1}$$

In Eq. (1), function $f_{cell}(x_t)$ passes x_t at step t to compress into the hidden state s_t, and $f_{out}(s_t)$ transforms the hidden state s_t into o_t. In effect, f_{cell} is chosen as a Hyperbolic function $tanh$ in original RNN, while in LSTM cell, the f_{cell} is divided as three processing gates: the input gate determines how much input information should be received; the forget gate determines how much history information should be omitted; the output gate determines how much current information should be outputted. The LSTM is formulated in the following:

$$f_i = G_f(x_t, s_{t-1}) \tag{2}$$
$$i_t = G_i(x_t, s_{t-1}) \tag{3}$$
$$o_t = G_o(x_t, s_{t-1}) \tag{4}$$
$$c_t = Cell(c_{t-1}, x_t, i_t, f_t, s_{t-1}) \tag{5}$$
$$h_t = State(o_t, c_t) \tag{6}$$

In Eqs. (2-4), the gate functions G are Sigmoid functions, containing three groups of parameters for forget gate, information gate and output gate, respectively. Different from the original RNN, LSTM has a cell state c_t that helps store sequence relations. Then, the outputs are obtained directly from the output gate. Furthermore, for complex data structure, standard LSTM models can be stacked to enhance the capacity of capturing complex regularity. Considering the learning and testing efficiency, two LSTM blocks are stacked and the stacked entity is called a two-layer LSTM model. As such, a two-layer LSTM model is employed to learn sequential characteristics from a sequence of extracted templates in normal tasks. A well trained LSTM model is then utilized to compute the anomaly degree of one task sample. The idea is that the LSTM model attempts to predict the future templates that are compared with ground truth during the corresponding task. The predicted results are in a list ranked with probability. The comparison is to match the ground truth with the list position and the anomaly degree is exponential to the matched position.

$$list_{o_t}^{(m)} = SeqModel(o_{0:t-1}) \tag{7}$$
$$Pos_{l_t} = \begin{cases} f_{arg}(list_{o_t}^{(m)}, l_t), & \text{if } l_t \in list_{o_t}^{(m)} \\ m+1, & \text{otherwise} \end{cases} \tag{8}$$
$$Dgr = exp(Pos_{l_t} - m) \tag{9}$$

Eq. (7) formulates the sequential model and its *top-m* list $list_{o_t}^{(m)}$, where o_t denotes the template at time step t, and $o_{0:t-1}$ denotes the previous templates as a time series from step 0 to $t-1$. In Eq. (8), if the template label l_t belongs to the $list_{o_t}^{(m)}$, the corresponding positioning Pos_{l_t} is recorded by an argument search function $f_{arg}(\cdot)$, otherwise, the positioning is assigned by $m+1$. The positioning Pos_{l_t} is subsequently subtracted by m in an exponential expression in Eq. (9) to compute how anomalous the template is at time t, which is called anomaly degree. The calculated degree projects the prediction ranking list into measurable values to reflect the deviation between the expectations and the real task records. Through the values, the anomaly candidates can be filtered out based on applying a degree threshold.

VI. Temporal And Spatial Feature Joint Analysis

From this section, we target at the second stage in the two-stage outliers identification, which combines and integrates both log semantics and services correlations. This part serves as a complementary processing for abnormal samples filtering in case that false alarms would happen in Stage 1, since sequential logs might have slight turbulence in occurrence order. The front-end of Stage 2 includes transactional topic representation inferred from logs and tracing representation inferred from services interactions. The back-end detector then underlies the weakly supervised one-classification methods to filter out outliers.

A. Transactional Topic Representation

This section is to discuss the semantic representation of grouped logs, which are always entangled within a short period. The semantic learning collects execution information and will be the first half of the eventual transaction representation.

Though the preprocess mainly involves in individual log record structures, the downstream tasks require deep analysis and comprehensive explanation in higher hierarchy. In other words, single individual log messages only record meta-execution, i.e., add and delete, send and receive, etc. The issues of meta-executions can express *error* or *critical* in log severity, while no evident signs will probably be expressed in case of system-level faults. For instance, the chaotic execution order causes service failure. The semantic analysis is to observe and discover hidden running patterns and make use of them to diagnose root causes.

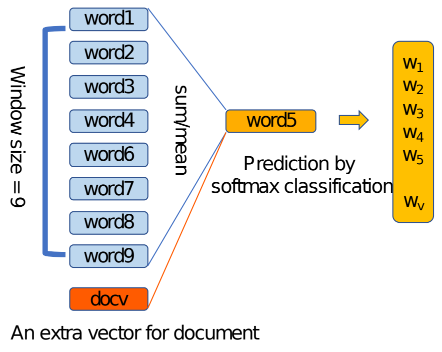
A reasonable solution to high level events representation is to gather a group of logs sequentially occurring in a time-window and summarize its "*topic*" to a more compact and expressive format. The expressive formats shall compactly capture the corresponding systematic information and avoid effect of absolute message volume. In this manner, several fundamental factors should be taken into consideration. Firstly, logs shall be converted into templates to indicate events instead of unique individuals. The conversion depends on the outputs of the template extraction method and informative templates prevents identical events divergence. Secondly, the collected logs shall keep the original order since the execution sequence is one of the dominant factors which heavily influences decision. A widely accepted argument is that the execution order entails essential work-flow, and its disorder and incompleteness possibly imply an abnormal task. Thirdly, the time-window shall be set properly and flexibly to ensure covering the close related events and omitting the long-standing disturbance issues, e.g., daemon procedures. Last but not least, the expressive "*topic*" shall be capable of keeping core content, compressing in a compact space and simplifying subsequent numerical computation.

We employ the template library, transaction request information and natural language analysis to address the previous issues. As for the first two points, the issues are straightforward. All the raw messages are scanned line by line, each of which is aligned with one example in the template library. In practice, one way to reduce computation complexity is to only consider examples with the same length. Afterwards, all positions corresponding to asterisks would be masked to give way to matching other positions. Once conversion is finished, the whole data set becomes a sequence of meta-executions excluding unstable variables. In terms of the time window, a basic idea is to use a pre-defined fixed scale to separate logs. However, the segments either may lose a significant part of logs or cover irrelevant logs mistakenly. Therefore, the logs of interest should be determined by request time duration collected based on practical transaction time-stamps. The runtime duration will be discussed later along with the request tracing in Section VI-B.

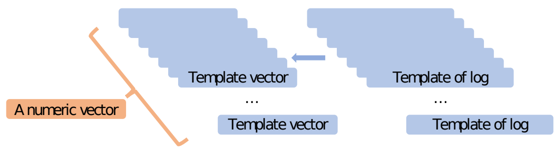
To obtain densely distributed representation of transactions, the document representation approach from natural language processing is introduced [29]. Document distributed representation, called *doc2vec*, stems from word distributed representation [26], which is called *word2vec*. The word2vec succeeds in not only capturing word-level semantic meaning, but also adapting semantic transferring into word vector calculation. As aforementioned in Section V, the *word2vec* is adopted to vectorize log templates, which in turn serves as input entries for the temporal model. Furthermore, the doc2vec inherits the mechanism of word2vec to wrap up the whole paragraph or document vector to entail a *topic*.

Generally, the word embedding learning refers to word2vec-like algorithms, learning distributed representation of text elements. Fig. 3a illustrates the fundamental concepts of word2vec and the following doc2vec. To simplify the theory, the Continuous Bag of Words (CBOW) in [26] is introduced. The embedding learning attempts to converge each word vector within a time window by aggregating the context information and predicting the central word. At the beginning, each word will be assigned with a randomly initialized vector. Choosing a window size, e.g., $win = 9$, will constrain and determine the scope and context of interest. As shown in Fig. 3a, *word5* takes the center position, hence being taken as a prediction objective. The other 8 words make up the context information, which are aggregated by summing or averaging the vectors. The prediction is accomplished by softmax function with the input vector $\boldsymbol{w} = (w_1, w_2, ..., w_v)$, where v is the size of the entire vocabulary. The window will slide through the entire sentence to ensure learning every word, and each moving step produces a training example. On top of the CBOW paradigm, the doc2vec simply adds a fixed extra vector $docv$ into the training process. The basic idea is very similar except that $docv$ is chosen identically across the sentence. Note that $docv$ will not remain fixed in training but will keep using the same vector instance across window sliding.

In contrast to the intuitive and generic natural language processing, our model utilizes *doc2vec* over log template sentence-level rather than constant word-level. That is in the log analysis, the analysis grain is log templates rather than general language words. In essence, a language sentence can be seen as a sequence of words. Similarly, it is reasonable to imagine that a complete transaction or task is a sequence of meta-executions, recorded by a sequence of logs that can be substituted by intrinsic templates. Thus, in log analysis, one template corresponds to one "word", and one transaction log collection corresponds to one "sentence". Then, such a "sentence" (an actual transaction or task) is fed into a *doc2vec* model to learn its distributed vector representation. As can be


Window size = 9
word1
word2
word3
word4
word6
word7
word8
word9
sum/mean
word5
Prediction by softmax classification
w_1 w_2 w_3 w_4 w_5 w_v
docv
An extra vector for document

(a) Word2vec and doc2vec



(b) Executions representation

Fig. 3: Illustration of embedding learning and task-level representation

seen from Fig. 3b, each template is embedded as a template vector, and via *doc2vec* a sequence of log template is also embedded as a transaction embedding vector.

B. Service Query Tracing

In this section, we will introduce the service tracing module to extract routine and subroutine response duration to construct the temporal information. Service tracing is of special interest in large scale and distributed server clusters as well as the micro-services architecture, which couple each other with a communication mechanism, i.e., RESTful API. As a web-search example described in [19], the basic front-end functionality in distributed platforms heavily depend on frequent service queries, whose responses normally consist of a stable relation chain.

The targeted scenario is that user-oriented queries are sent from the very first front-end to all computation provider back-ends, which is inclined to form a spanning tree-like structure. The motivation is that if a complete query path can be well tracked, a malfunction action can be well identified and located. The “Dapper” tool depicted in [19] provides a robust, scalable and effective implementation as well as “Zipkin” in [37], and “Osprofiler” in [38]. Practically, a tracing component will insert a light weight collector into each service point, and immediately return service activities to central monitor. The light weight collectors should only cause negligible overhead and be transparent to application developers [19]. Fig. 4 illustrates service query chain, a spanning tree structure.

From Fig. 4, users launch a service request through an interface, the application entry. In the following, the request is processed as several parts which are delivered to back-end service point via service queries, as depicted by light blue arrows. It is possible that the first tier receiving queries merely conducts intermediate analysis so that the first tier queries are mapped to the next service tier. Queries are eventually disassembled to multiple primitive ones, sent to the last tier, and the results are reversely responded to the application entry, where all messages are assembled and replied to users. In this case, there are three levels of service back-end points.

In the proposed framework, the tracing infrastructure is utilized for structure-related information, because service requests and back-end query durations implicitly reflect the function integrity. With the assistance of a tracing module, the whole query durations are stored to construct a service query matrix to record one transaction inner component query histories. Eq. 10 below shows a simple example of service query matrix. Here, services are denoted as numbered columns and rows. Columns are the query sender point and rows are the receivers. Each entry indicates the response duration from column to row. For instance, $Service\ Query\ Matrix(1,2) = 0.4$ means service 1 calling service 2 with the response duration 0.4 seconds. The matrix explicitly contains all the necessary structural time information extracted from the tracing module, and will combine its corresponding transaction “*topic*” vector as the complete temporal-spatial representation.

$$Service\ Query\ Matrix = \begin{bmatrix} 0 & 2 & 0 & 0.1 & 0 \\ 0.4 & 0 & 0.3 & 0 & 0 \\ 0 & 0 & 0 & 0.78 & 1.7 \\ 1.5 & 0 & 0.9 & 0 & 0 \\ 0 & 1.1 & 0 & 0.33 & 0 \end{bmatrix} \tag{10}$$

C. Outliers Detector

Based on the behavior and query trace representation, the two parts should be integrated as a synthesized feature vector. The feature vector will be taken as input to an anomaly detector. The analysis step serves as the last session of jointly temporal and spatial detection at the Stage 2 of Anomaly Detection step. The joint detection process shall give the second identification of abnormal task samples, complementing the temporal model.

For representation integration, firstly the matrix should be converted into a vector. In general, a matrix is flattened with respect to column or row. That is, for row, placing all rows elements in one single row but keeping their relative context position. In our work, the service query matrix is flattened across row and the dimension is $dim_{query} = n \times n$, where n denote the number of services. To keep the dimension consistent through all data samples, the number is for the total service. In one transaction, it is possible that not all services are involved, where irrelevant columns and rows are set as 0. Afterwards, to aggregate two parts, transaction “*topic*” vector and service query vector can be concatenated seamlessly.

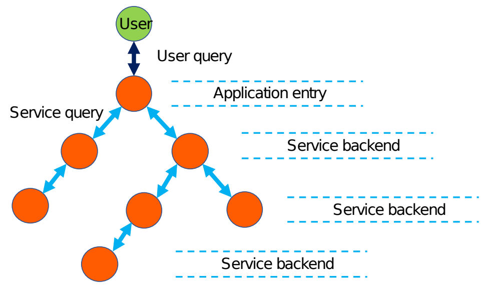


Fig. 4: A user request from front-end to back-end path

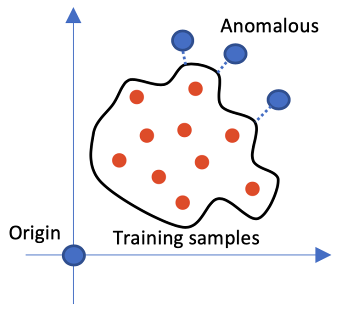


Fig. 5: The OSVM is training samples against the origin

The concatenation results in a hybrid vector with dimension $dim_{concat} = dim_{query} + dim_{log}$.

As for the anomaly detector, the basic idea is to absorb the features of normal samples and clarify the normality boundary in a feature space in training, and identify outliers in practice. Here, in training, all input samples are assumed normal, and for test samples, outliers will be considered as anomalies against the normal sample distribution. One noteworthy issue is to find a proper high-dimensional area to encompass the training cluster. To this end, one-class classification (OCC) [46] is investigated, which is prevalent in anomaly detection sphere [46] [47]. When there is a newly incoming data point, a classifier is able to identify whether it is belonging to the training set. From the OCC algorithm, the one-class support vector machine (OSVM) [48] [49] is of special interest.

The OSVM adopts the similar idea to standard support vector machine, by drawing an optimal boundary to separate two categories. In OSVM, only one category is specified, therefore the other one consists of the origin in the feature space. Essentially, OSVM finds a least radius sphere in hyperplane to encompass the training data, if kernel trick is used for linear non-separable situation. As shown in Fig. 5, red circles denote the training samples, which should be separated against the origin. Note that the boundary drawn is not a strict sphere in the 2D space, but remain spherical in a high dimensional space. The OSVM is robust to noise in training dataset since it mainly concentrates on flexible data distribution boundary, and the training is effective and fast given a small size of dataset. It is reasonable to consider the advantage of OSVM, provided that collectible and deterministic fault samples in a complex architecture. Therefore, we employ the OSVM algorithm as the outliers classifier to fit in small size training data and detect the abnormal tasks. Three blue-colored points are marked as anomalous points due to their locations. The blue dash lines denote the distance from boundary to outliers.

VII. Experimental Results and Analysis

In this section, we will present our empirical experiments of template extraction, the temporal log modeling, and the integrated outliers detector with concatenated features. The data processing machine is with 16-core Intel Xeon E5-2630 v3 CPU, 64GB memory, and Nvidia GTX 1080Ti GPU.

The data are collected from a public dataset and the distributed platform in our lab. The public dataset comes from the BlueGene/L [17], a supercomputer developed by IBM Watson Research Center. The BlueGene/L data will be utilized in template extraction as there lacks high level transactional label and essential service tracing records for the tracing matrix. The OpenStack log data is collected from OpenStack services, i.e., nova-computing, neutron-networking, and etc., deployed in three physical computing nodes in our lab. One of the three nodes plays the role of both the controller and compute node, which enables keystone authentication service, glance image service, nova-api daemon service, nova computing service and neutron-networking service. The other two solely participate in compute node with nova computing service and neutron networking service. As described before, we make use of Osprofiler, the OpenStack tracing module, to store hierarchical request trace paths. For evaluating anomaly detection, we collect 122 user transaction samples from virtual instance creation and deletion. All data are labeled as 100 normal samples, and 22 outliers. As for labeling, once a request is sent into OpenStack, the trace module will start to construct a request tree via inserted probes and assign a unique trace ID to it. The corresponding logs can be directly located based on the trace ID and starting time.

Table III depicts the summary of the volume of the two datasets. BlueGene/L contains more than 4 million logs in total, two thirds of which stay with “info” and “warning” severity. Similarly, OpenStack has nearly 270 thousand records, most of which have “info” and “warning” severity fields. These “info” and “warning” records are important information in our work. This is fairly reasonable, because we assume that if “error” appears, it surely points to an anomaly, however if only “info” and “warning” appear, abnormal events probably are hidden under the seemingly healthy records.

Table IV describes the transaction sample dataset for the two-stage anomaly detection. In total, 122 samples are collected, 100 normal samples and 22 anomalies, through basic virtual instance creation and deletion operations requested by users. In abnormal conditions, interferences have been inserted to make instance creation unsuccessful or long delay. Wherein, unsuccessful creation is triggered by not enough user quote, and the long delay is due to the broken physical network links. Note that the inadequate quote is not a system anomaly, but it still can be marked as failed requests with “info” and “warning” records. The proposed model can successfully identify the exceptions.

TABLE III: Log datasets in experiments

Dataset	BlueGene/L		OpenStack	
Volume	Total 4,747,963		Total 269,169	
	Info/Warning 3,759,170	Error 988,793	Info/Warning 243,780	Error 25,389

TABLE IV: Transactional data description

Log Volume	243,780		
Sample Volume	Total 122	Normal 100	Abnormal 22
Sample Type	Total 122	Creation 96	Deletion 26

A. Evaluation of Template Extraction

In the beginning, we show the efficiency and analyze the concrete outcomes of template extraction between IPLoM and our method. Some evaluations have been done in [24] and [23]. We take advantage of the open-source material with default settings for BlueGene/L logs and implement our own method. The parameter settings are: the information threshold, θ, is set as 0.5 to keep at least 50% terms; the log count in one segment or buffered size in an on-line scheme, $Seg\_interval$, is set as 10000 and 100000, respectively; furthermore, in the *long template adjustment*, the exceeding percentage is set as 0.9. The results are shown in Table V. The rows represent investigated algorithms including the original IPLoM and our proposal with two aggregation strategies, fixed information threshold and elastic aggregation, respectively. The columns denote the segment setting, extracted accuracy and complete time. In detail, the *Segmented Iteration* corresponds to the intuitive fixed information threshold strategy, and the *Elastic Aggregation* corresponds to the elastic aggregation modification. In terms of the accuracy, the proposed method is competitive and even performs slightly better than the original IPLoM. The highest one is the elastic aggregation with accuracy 0.61. As for the time cost, both of our proposals are nearly as twice as faster than IPLoM.

TABLE V: Template extraction results

Method	Block Size	Accuracy	Time (s)
IPLoM		0.54	31.9
Segmented Iteration	10000	0.57	16.44
	100000	0.56	**12.26**
Elastic Aggregation	10000	0.53	16.92
	100000	**0.61**	12.38

B. Evaluation of Temporal Log Feature Model

After the data preparation step, the whole raw log messages are converted into their corresponding templates. Each transactional logs block (logs in one task) is located by request time stamps labeled in tracing modules. Within one task, the sequential templates are pre-processed as sequences samples for the input of the LSTM learning model. To be more specific, the temporal model is comprised of a two-layer bi-directional LSTM block with 512-sized hidden states and 256-sized input units. In the learning phase, all the normal samples are utilized to train the baseline model associated with two types of tasks, i.e., virtual instance creation and deletion. The training accuracy and epochs are presented in Fig. 6a, where the training accuracy promptly reaches nearly 1.0 within 500 epochs while the whole training continues with 3000 epochs. In the testing phase, normal tasks and abnormal tasks are both fed into anomaly degree/score calculation with the definition in Section V. The results are illustrated in Fig. 6b, Fig. 7a, and Fig. 7b. It is manifest that the abnormal task samples attain above the anomaly degree of 5, nonetheless the normal creation and deletion tasks are orders of magnitude lower than that of anomalous ones, maintaining less than 0.1. Particularly, the first anomaly reaches more than 300 because the task suffers low term networking disconnection, while others encounter short term task termination.

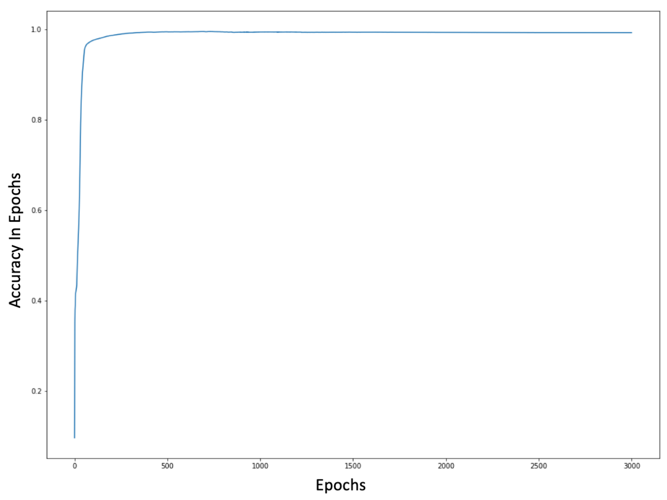
C. Evaluation of Temporal And Spatial Feature Joint Model

At Stage 2, the two modules of system information representation are jointly taken into effect for global and in-depth anomalies recognition. Similar to the temporal model preprocessing, log templates are the abstract input data for transactional interpretation. The *doc2vec* algorithm takes as input the located normal transactional log blocks to train the *document* representation with $dim_{log} = 150$. As for tracing features, the tracing mechanism recognizes 20 services in OpenStack, therefore one 20×20 trace matrix is flattened as a $dim_{query} = 400$ vector. Concatenated with *document* representation, the ultimate integrated vector is with $dim_{concat} = 550$. A high dimensional space needs to be visualized to verify its interpretability. To this end, the t-SNE algorithm [50] is adopted to map high dimensional representations into 2D space and properly visualize the space structure. Fig. 8a illustrates all the samples with ground truth labels distributed in 2D space. The cyan blue circles are normal points, and the magenta circles refer to inserted abnormal requests. Obviously, the combined features clearly reflect the distinct cluster natures, which enables the following outliers classifier.

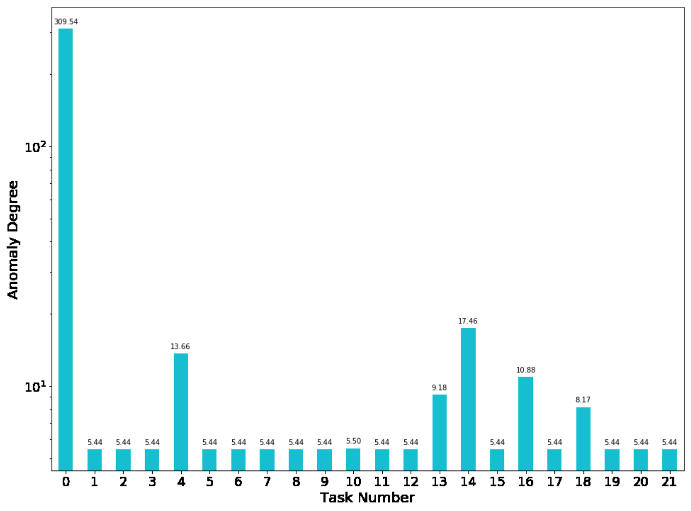
By the OSVM method with radial basis function (RBF) kernel, the detection results are visualized in Fig. 8b. In training, we use 90 normal samples to train the boundary while in evaluating, 10 normal samples and 22 abnormal samples are tested. As shown in this figure, training with the normal data marked as cyan has 0.91 training accuracy and all anomalies are successfully identified, marked as magenta on the left. Compared to Fig. 8a, several normal data are mistakenly identified as anomalies. We conjecture that in high dimensional training, these data are placed near the periphery of normal distribution boundary, despite being distant from anomalies. Without an elastic sphere boundary, possibly a small number of normals are segregated out of the normal scope.

D. Discussion

The previous experiments clearly emphasize the effectiveness of three essential components, template extraction, se-

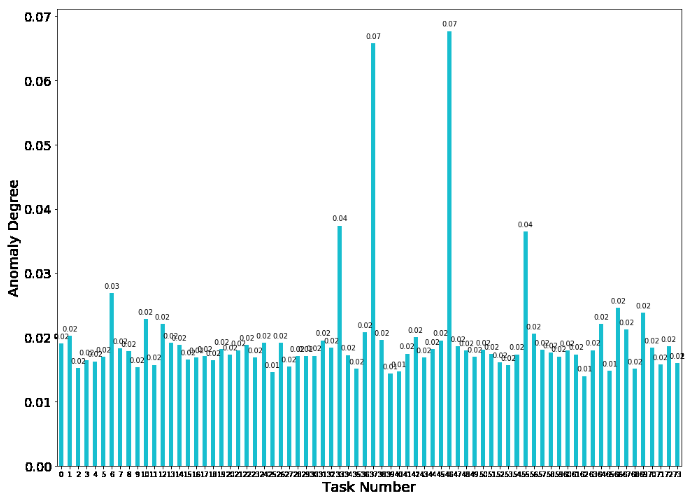


(a) The training accuracy and epochs

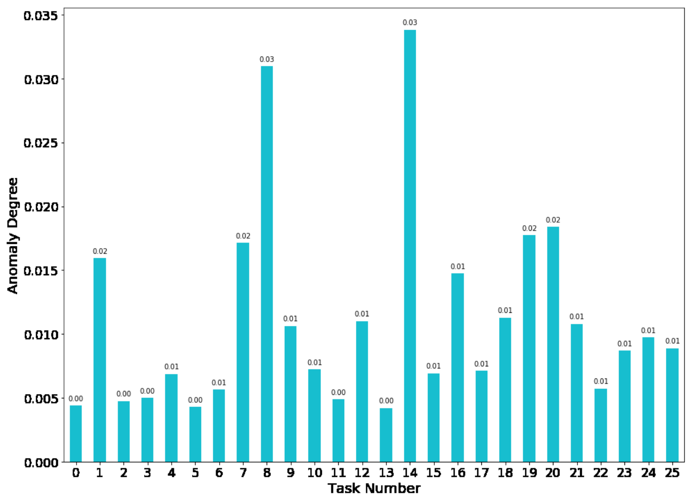


(b) Anomaly degree bar chart of all abnormal tasks

Fig. 6: Anomaly degree bar chart of normal instance creation and deletion tasks

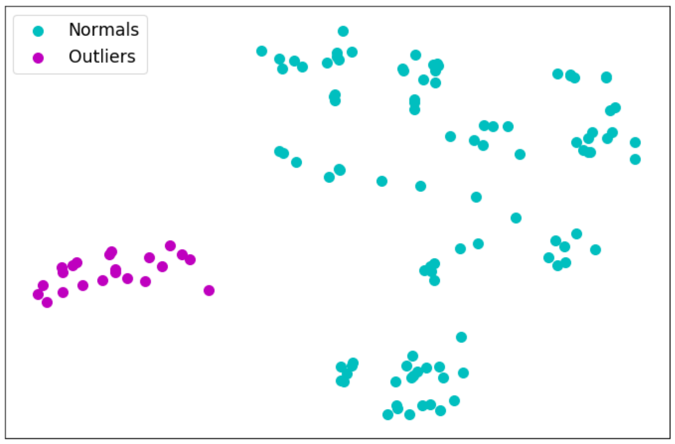


(a) Normal instance creation tasks

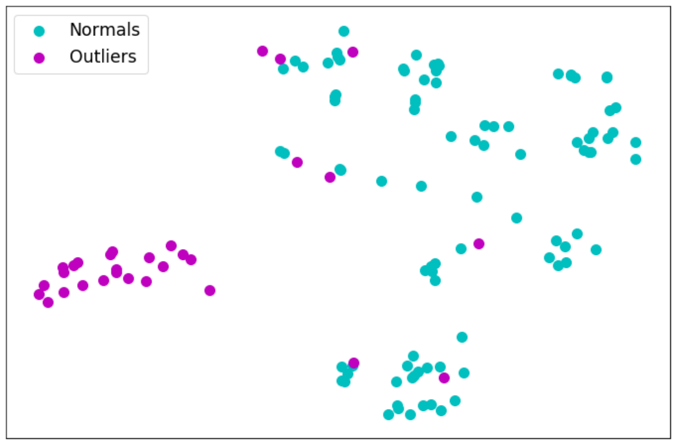


(b) Normal instance deletion tasks

Fig. 7: Anomaly degree bar chart of normal instance creation and deletion tasks



(a) Ground truth



(b) Detection results

Fig. 8: Visualization of all samples with ground truths and detection results

quential anomaly degree model, and temporal and spatial joint anomaly detection. Each discrete module performs well in efficiency, availability and interpretability. It is worth noting that the complete framework presented in Fig. 1 smoothly integrates all the three processing components instead of being intuitively isolated. The templates closely underpin the latter abstract event representation, and tracing data facilitates spatial information analysis. The sequential degree calculation captures the temporal execution patterns and produces high degree suspicious task candidates, against which the results of the outliers classifier are compared for double anomalies validation. The two-stage anomaly validation makes use of the most of stable temporal and spatial features collected from the micro-services to avoid accidental mistakes due to single decision-making.

VIII. Conclusion

The service-oriented 5G system dedicates to providing diverse and enormous services in a large range, which inevitably requires a sustainable operation mechanism. Efficient and reliable management for a large-scale micro-services architecture has become extremely significant. In this paper, we have proposed a general-purpose anomaly detection framework targeting at micro-services architectures, using temporal service execution logs and spatial service query traces. The framework is comprised of informative data representation, temporal logs modeling and temporal-spatial joint analysis. It works compatibly with but not limited in well-performed LSTM model, *doc2vec* and *tracing matrix*, and unsupervised outliers identification algorithms. Experimental results have shown competitive effectiveness of the proposed scheme: the template extraction achieves competitive accuracy and high efficiency; the temporal model successfully acquires sequential patterns for anomaly degree computation; the integration of the transactional representation and spatial service query traces, segregates anomalies from normal points, visualizes the distribution that data entail, and helps the outliers classifier to highlight those anomalies. Our future work will root in exploring an effective way to not only detect anomalies but also localize and rank the root causes of faults.

References

[1] J. G. Andrews, S. Buzzi, W. Choi, S. V. Hanly, A. Lozano, A. C. Soong, and J. C. Zhang, “What will 5g be?” *IEEE Journal on selected areas in communications*, vol. 32, no. 6, pp. 1065–1082, 2014.

[2] X. Foukas, G. Patounas, A. Elmokashfi, and M. K. Marina, “Network slicing in 5g: Survey and challenges,” *IEEE Communications Magazine*, vol. 55, no. 5, pp. 94–100, 2017.

[3] M. Yang, Y. Li, B. Li, D. Jin, and S. Chen, “Service-oriented 5g network architecture: an end-to-end software defining approach,” *International Journal of Communication Systems*, vol. 29, no. 10, pp. 1645–1657, 2016.

[4] M. Bell, *SOA Modeling patterns for service-oriented discovery and analysis*. John Wiley & Sons, 2009.

[5] C. Pautasso, O. Zimmermann, M. Amundsen, J. Lewis, and N. Josuttis, “Microservices in practice, part 1: Reality check and service design.” *IEEE Software*, vol. 34, no. 1, pp. 91–98, 2017.

[6] P. Jamshidi, C. Pahl, N. Mendonça, J. Lewis, and S. Tilkov, “Microservices: The journey so far and challenges ahead,” *IEEE Software*, vol. 35, no. 3, pp. 24–35, 2018.

[7] O. Zimmermann, “Microservices tenets,” *Computer Science-Research and Development*, vol. 32, no. 3-4, pp. 301–310, 2017.

[8] D. Bhamare, M. Samaka, A. Erbad, R. Jain, and L. Gupta, “Exploring microservices for enhancing internet qos,” *Transactions on Emerging Telecommunications Technologies*, vol. 29, no. 11, p. e3445, 2018.

[9] S. Hassan and R. Bahsoon, “Microservices and their design trade-offs: A self-adaptive roadmap,” in *2016 SCC*. IEEE, 2016, pp. 813–818.

[10] Y. Zhu, Y. Wu, G. Min, A. Zomaya, and F. Hu, “A survey of big data and computational intelligence in networking,” in *Big Data and Computational Intelligence in Networking*. CRC Press, 2017, pp. 3–26.

[11] H. Wang, Y. Wu, G. Min, J. Xu, and P. Tang, “Data-driven dynamic resource scheduling for network slicing: A deep reinforcement learning approach,” *Information Sciences*, vol. 498, pp. 106–116, 2019.

[12] H.-N. DAI, R. C.-W. WONG, H. WANG, and A. V. VASILAKOS, “Big data analytics for large scale wireless networks: Challenges and opportunities,” 2019.

[13] Y. Zuo, Y. Wu, G. Min, and L. Cui, “Learning-based network path planning for traffic engineering,” *Future Generation Computer Systems*, vol. 92, pp. 59–67, 2019.

[14] A. Oliner, A. Ganapathi, and W. Xu, “Advances and challenges in log analysis,” *Communications of the ACM*, vol. 55, no. 2, pp. 55–61, 2012.

[15] S. Sabato, E. Yom-Tov, A. Tsherniak, and S. Rosset, “Analyzing system logs: A new view of what’s important,” in *Proceedings of the 2nd USENIX workshop on Tackling computer systems problems with machine learning techniques*, vol. 7, no. 6. USENIX Association, 2007, pp. 1–7.

[16] M. Du, F. Li, G. Zheng, and V. Srikumar, “Deeplog: Anomaly detection and diagnosis from system logs through deep learning,” in *Proceedings of the 2017 ACM SIGSAC*. ACM, 2017, pp. 1285–1298.

[17] A. Oliner and J. Stearley, “What supercomputers say: A study of five system logs,” in *2007 37th DSN*. IEEE, 2007, pp. 575–584.

[18] R. Fonseca, G. Porter, R. Katz, S. Shenker, and I. Stoica, “X-trace: A pervasive network tracing framework,” in *Proceedings of the 4th USENIX conference on Networked systems design & implementation*. USENIX Association, 2007, pp. 20–20.

[19] B. Sigelman, L. Barroso, M. Burrows, P. Stephenson, M. Plakal, D. Beaver, S. Jaspan, and C. Shanbhag, “Dapper, a large-scale distributed systems tracing infrastructure,” *Google Technical Report*, 2010. [Online]. Available: https://research.google.com/archive/papers/dapper-2010-1.pdf

[20] R. Vaarandi, “A data clustering algorithm for mining patterns from event logs,” in *IP Operations & Management, 3rd IEEE Workshop on*. IEEE, 2003, pp. 119–126.

[21] A. Makanju, A. Zincir-Heywood, and E. Milios, “Clustering event logs using iterative partitioning,” in *Proceedings of the 15th ACM SIGKDD*. ACM, 2009, pp. 1255–1264.

[22] A. Makanju, A. Zincir-Heywood, and E. Milios, “A lightweight algorithm for message type extraction in system application logs,” *IEEE Trans. Knowledge and Data Engineering*, vol. 24, no. 11, pp. 1921–1936, 2012.

[23] J. Zhu, S. He, J. Liu, P. He, Q. Xie, Z. Zheng, and M. Lyu, “Tools and benchmarks for automated log parsing,” in *IEEE/ACM 41st International Conference On Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. IEEE, 2019, pp. 121–130.

[24] P. He, J. Zhu, S. He, J. Li, and M. Lyu, “An evaluation study on log parsing and its use in log mining,” in *2016 46th DSN*. IEEE, 2016, pp. 654–661.

[25] D. Blei, A. Ng, and M. Jordan, “Latent dirichlet allocation,” *JMLR*, vol. 3, no. Jan, pp. 993–1022, 2003.

[26] T. Mikolov, K. Chen, G. Corrado, and J. Dean, “Efficient estimation of word representations in vector space,” in *Processings of the International Conference on Learning Representations (ICLR)*, 2013, pp. 1–14.

[27] T. Mikolov, I. Sutskever, and et al., “Distributed representations of words and phrases and their compositionality,” in *NIPS*, 2013, pp. 3111–3119.

[28] J. Pennington, R. Socher, and C. Manning, “Glove: Global vectors for word representation,” in *Proceedings of the EMNLP*, 2014, pp. 1532–1543.

[29] Q. Le and T. Mikolov, “Distributed representations of sentences and documents,” in *ICML*, 2014, pp. 1188–1196.

[30] Y. Kim, “Convolutional neural networks for sentence classification,” in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*, 2014, pp. 1746–1751.

[31] N. Kalchbrenner, E. Grefenstette, and P. Blunsom, “A convolutional neural network for modelling sentences,” in *Proceedings of 52nd Annual Meeting of the Association for Computional Linguistics*, 2014, pp. 655–665.

[32] A. Dieng, C. Wang, J. Gao, and J. Paisley, “Topicrnn: A recurrent neural network with long-range semantic dependency,” in *Proceedings of the International Conference on Learning Representations*, 2016, pp. 1–13.

[33] H. Lu, L. Xie, N. Kang, C. Wang, and J. Xie, "Don't forget the quantifiable relationship between words: Using recurrent neural network for short text topic discovery," in *Proceedings of the 31st AAAI*, 2017.
[34] Z. Yang, Z. Hu, R. Salakhutdinov, and T. Berg-Kirkpatrick, "Improved variational autoencoders for text modeling using dilated convolutions," in *Proceedings of the 34th ICML-Volume 70*. JMLR. org, 2017, pp. 3881–3890.
[35] Y. Miao, E. Grefenstette, and P. Blunsom, "Discovering discrete latent topics with neural variational inference," in *Proceedings of the 34th ICML-Volume 70*. JMLR. org, 2017, pp. 2410–2419.
[36] U. Technologies, "Jaeger: open source, end-to-end distributed tracing," 2019. [Online]. Available: https://www.jaegertracing.io/
[37] Zipkin, "Openzipkin: A distributed tracing system," 2019. [Online]. Available: https://zipkin.io/
[38] Openstack, "Openstack: Osprofiler," Apr 2019. [Online]. Available: https://github.com/openstack/osprofiler
[39] S. He, J. Zhu, P. He, and M. Lyu, "Experience report: System log analysis for anomaly detection," in *2016 IEEE 27th ISSRE*. IEEE, 2016, pp. 207–218.
[40] X. Yu, P. Joshi, J. Xu, G. Jin, H. Zhang, and G. Jiang, "Cloudseer: Workflow monitoring of cloud infrastructures via interleaved logs," in *ACM SIGPLAN Notices*, vol. 51, no. 4. ACM, 2016, pp. 489–502.
[41] W. Meng, Y. Liu, Y. Zhu, S. Zhang, D. Pei, Y. Liu, Y. Chen, R. Zhang, S. Tao, P. Sun *et al.*, "Loganomaly: Unsupervised detection of sequential and quantitative anomalies in unstructured logs," in *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19. International Joint Conferences on Artificial Intelligence Organization*, vol. 7, 2019, pp. 4739–4745.
[42] K. Jackson, *OpenStack cloud computing cookbook*. Packt Publishing Ltd, 2012.
[43] R. Agrawal, T. Imieliński, and A. Swami, "Mining association rules between sets of items in large databases," in *Acm sigmod record*, vol. 22, no. 2. ACM, 1993, pp. 207–216.
[44] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.
[45] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, p. 436, 2015.
[46] S. Khan and M. Madden, "One-class classification: taxonomy of study and review of techniques," *The Knowledge Engineering Review*, vol. 29, no. 3, pp. 345–374, 2014.
[47] E. J. Pauwels and O. Ambekar, "One class classification for anomaly detection: Support vector data description revisited," in *ICDM*. Springer, 2011, pp. 25–39.
[48] L. Manevitz and M. Yousef, "One-class svms for document classification," *JMLR*, vol. 2, no. Dec, pp. 139–154, 2001.
[49] B. Schölkopf, R. Williamson, A. Smola, J. Shawe-Taylor, and J. Platt, "Support vector method for novelty detection," in *NIPS*, 2000, pp. 582–588.
[50] L. Maaten and G. Hinton, "Visualizing data using t-sne," *JMLR*, vol. 9, no. Nov, pp. 2579–2605, 2008.