# WIDE クラウド WG 2016 年度活動報告

小林 諭 *　　島 慶一 †

2017 年 1 月 6 日

## 1　はじめに

WIDE クラウドワーキンググループは、今後のクラウド技術の研究開発を推進するために 2010 年 1 月に設立された。複数の WIDE 組織間に渡って運用される広域連邦型クラウドシステムである WIDE クラウドシステムの運用と、それを用いた研究開発を行っている。

2016 年度の主な活動は以下の通りである。

- 因果推定に基づくログの関係情報抽出技術 (第 2 章)
- テンプレートを用いないログ異常検知技術の研究 (第 3)
- シンプルなログテンプレート生成技術の研究 (第 4)

以降、それぞれの活動の詳細を報告する。

## 2　因果推定に基づくログの関係情報抽出技術

ネットワークシステムの継続的運用のため、障害発生時の高速な原因究明及び復旧が重要視されている。システム障害の原因究明には主にシステムログなどに代表される運用データを用いるが、これらのデータは近年の複雑化したネットワークシステムにおいて肥大化する傾向がある。よってデータの効率的活用のため、運用データの自動解析技術が求められている。

しかし、システムログは他のデータと違い数値的に扱うことができずスパースかつ多次元的であるなど従来の解析技術の応用が難しい。一方で既存の技術では時系列的変化や相関などシステムログの表面的な性質を抽出するにとどまり、オペレータにとって真に有用な文脈的情報を獲得することが困難である。これに対し、本研究ではシステムログ上のイベント間の因果関係を効率的に推定する技術を提案し、障害原因の推測や有用な情報の提供、障害発生の事前予測などに応用することを目指している。

### 2.1　PC アルゴリズムを用いた因果解析

本研究ではシステムログ中に多数見られる「イベント」についてそのイベント間の因果関係を推定するため、PC アルゴリズム [1] を利用する。PC アルゴリズムはデータにおける因果関係を Directed Acyclic Graph (DAG) として高速に推定するためのアルゴリズムである。このアルゴリズムは条件付き独立性の判定を行うが、本研究においてはログイベントの出現の有無に関する時系列論理値データに対して G2 試験 [2] を用いることで実現した。G2 試験は離散データに対する条件付き独立性判定手法の 1 つである。ノード集合 Z を前提とするときのノード X, Y 間の条件付き独立性判定に用いられる統計量は、

$$G^2 = 2mCE(X, Y \mid Z) \tag{1}$$

で表される。ただし、m は各ノードの統計ベクトルの大きさを、CE() は条件付きクロスエントロピーを指す。

以下に、システムログにおいて因果関係の推定を行う処理の流れを示す。

まず、元データにおけるログの出力テンプレートの生成を行い、それを用いてログメッセージを分類する。システムログにおけるログの出力テンプレート生成については多くの既存研究が存在するが、本研究では小林 [3, 4] らが提案する教師あり学習に基づく手法によ

*東京大学
†IIJ イノベーションインスティテュート

り生成したテンプレートについて、一部修正を加えたものを用いた。

こうしてログの出力テンプレートごとの出現時刻の集合が、テンプレート別に得られることとなる。本研究ではこの出現時刻の集合を「イベント」と呼ぶものとし、因果関係推定にあたっての 1 単位とする。

次に、データセットを一定期間ごとに分割し、その上で各イベントの各時間帯での出現の有無について G2 試験を適用するため時系列論理値データに変換する。最後にこの論理値データ群について PC アルゴリズムを適用することで、各イベントをノード、イベント間の因果関係をエッジとする DAG が得られる。DAG はデータセットの分割単位ごとに 1 つ生成されることになる。

本研究では教育機関向けネットワークである SINET4 の運用ログデータに適用した。図 1 にこの手法によって自動検出された因果情報の例を示す。あるルータ機器で BGP の接続の初期化、再構築が行われた際、一時的にネットワークが不通になったことで接続している別のスイッチ機器においてネットワークインタフェースのエラーが検出された。この 2 つの機器において関連するイベントは時系列的に必ずしも一致していないためこの 2 機器間の因果関係をオペレータが直感的に見出すのは難しく、複数機器にまたがる障害のトラブルシューティングにおいて重要な情報を検出していると言える。

今後は提案手法の優位性を評価するため、母相関推定に基づく条件付き独立性検定手法や、Lift による有向相関に基づくグラフ生成手法などを実装し、それらとの処理時間、有用性の双方からの比較検討を行なっていく。

### 2.2 因果情報の優先度付け

2.1 で得られた因果情報は、SINET4 のログデータ 15ヶ月分 (35,513,125 行) からはのべ 8613 の因果関係が検出された。これは 1 日あたり平均 20 程度の因果情報が提示されることを意味しており、データの大半が障害に関係しないものであることから実際の障害時にはより多くの情報が提示されうる。これらの情報が障害にどれだけ関連するかの優先度付けを行うことができれば、より効率的に因果情報をトラブルシューティングに活用することが可能となる。

提示する因果情報を絞り込む単純な方法としては、「頻繁に得られる情報は障害に関わる可能性が低い」という仮定のもとに複数回現れる情報や連続した期間で検出され続ける情報を除くというものが考えられる。

図 2 に SINET4 のログデータから検出された因果関係における同一のイベント間の因果情報の出現回数を示す。このデータでは 776 の異なる因果情報が見つかったが、そのうち上位 10%に相当する 77 種類を除いたところ因果情報の数は 1722 まで減少した。この除かれた因果情報を調査すると、その大半が NTP と NTP、BGP と BGP など同一サブシステムのイベント間のものであるとわかった。これは確かに除かれた情報がオペレータにとって自明なものである可能性が高いことを示している。

しかし、上のようなパーセンタイルに基づく分別では閾値の設定がデータに大きく依存するという問題点がある。この問題に対して、改善案として「定常時に得られる情報は障害に関わる可能性が低い」という仮定を加え、システムログにおける既存の異常検知手法と組み合わせることが考えられる。異常検知手法によってデータセットを定常時と異常時に分類することで、それぞれにおいて検出される因果情報の優先付けが可能になると期待される。

## 3 テンプレートを用いないログ異常検知技術の研究

syslog メッセージを用いた異常検知手法では、一般的にメッセージのテンプレートを作成し、テンプレートの出現頻度や分布などを解析して異常を発見するのが普通である。しかしながらテンプレートの作成には一定の計算量が必要で、また syslog メッセージ自体が自由文字列を基本としたデータであるため、間違いのないテンプレートを類推することは事実上不可能である。もしテンプレートを用いずに、入力されたメッセージをそのままの形で処理し、その後の解析に利用できればその方が望ましい。

本研究では、syslog メッセージのテンプレート化を実行せずに、メッセージの集合を分類し、異常検知などに応用できないかどうかを検討した。

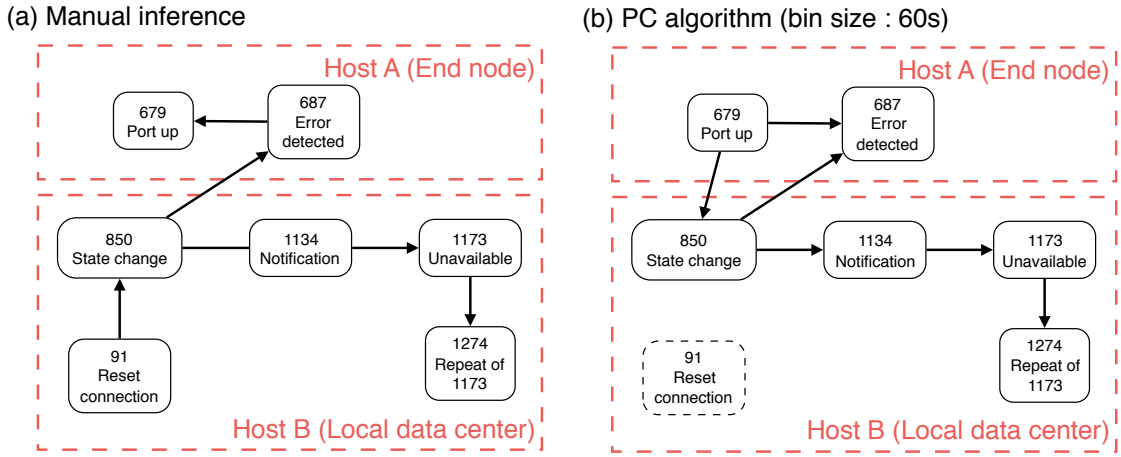



図 1: 検出された因果情報の一例

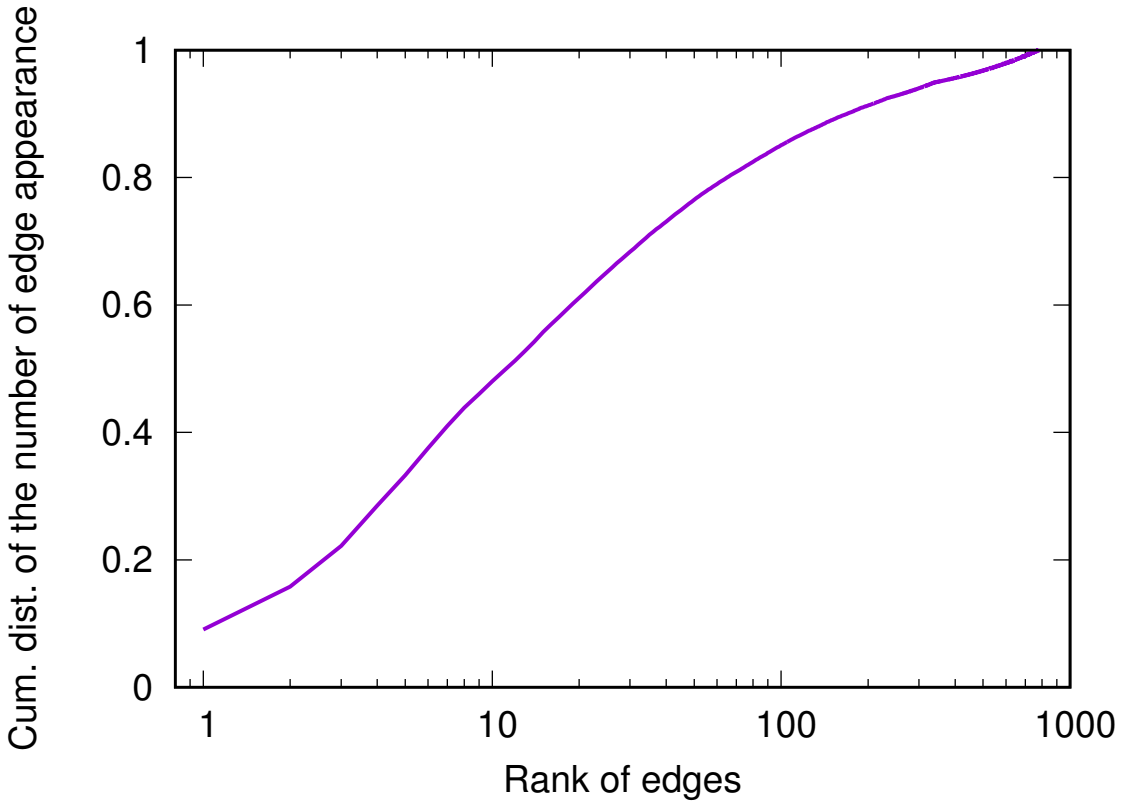



図 2: 同一因果情報の累積分布

### 3.1 状態の定義

syslog メッセージは時系列に発生する文字列の集合である。ある期間のシステムの状態は、その期間に出力された syslog メッセージで表現されていると考えることとする。異なる二つの期間に似たようなメッセージ集合が出力されている場合、その二つの期間の状態も似ていると考えられる。本研究では、メッセージの集合をいかにして状態を表す情報に変換するか、また状態を表す情報の類似度を計算するかに焦点を当てる。

### 3.2 共有ワードグラフ

特定期間の syslog メッセージの状態を定義するために、メッセージ間の共有ワードを元にしたグラフ構造を作成する。以後、このグラフ構造を共有ワードグラフと呼ぶこととする。

図 3 に syslog メッセージの例を示す。この例には 14 のメッセージが記録されている。図に示した通り、それぞれのメッセージはお互いに共有する単語を含んでいる。共有ワードグラフは、各メッセージを一つのグラフの頂点とし、共有ワードを持つ頂点を接続することで生成される。図 3 を共有ワードグラフとして描き直したものが図 4 である。このグラフをシステムの状態みなし、分類を行う。

### 3.3 iijlab データでの実験

iijlab で収集しているサーバーログを用いて共有グラフによる状態分類を実施してみた。一つの状態は 100 メッセージから計算される。すなわち、頂点数が 100 のグラフが連続的に生成されることとなる。ある時間の状態が、他の時間の状態とどれくらい類似しているかは、グラフの形状を比較する手法である Sorted graph histogram[5] を用いた。

図 3.3 は、iijlab で運用しているサーバーの一つの状態変化を共有ワードグラフを元にして表現したものである。左上のヒートマップは時系列に並べられたサーバーの状態を示す。2015 年 10 月 1 日から 11 月 1 日までの状態を示しており、全体で 6000 状態が表現されている。今回は syslog メッセージ 100 行を一つの状態として区切っているため、全体として 600,000 行のメッセージを処理したことになる。Y 軸は X 軸の各状態と他の時間の状態との比較になっている。二つの状態が類似してなければ青く、類似していなければ赤く表現される。左上の図を階層化クラスタリングして並べ直したものが右上の図になる。6,000 ある状態はおおよそ 4 つの大きな類似状態と、その他の小さな状態に分類できることがわかる。全体を 20 のクラスタに分類し、それぞれのクラスタの出現率を計算、さらに出現頻度の低いものを時系列にプロットしたものが左下の図となる。出現頻度が低い状態を、通常とは異なる状態とみなせば、syslog メッセージと共有ワードグラフから稀な状態をふるい出すことができると考えられる。現時点では、通常と異なる状態が即座に異常状態とは判断できない。洗い出された稀な状態が異常状態とどの程度関連しているかは今後の調査が必要になる。

## 4 シンプルなログテンプレート生成技術の研究

ネットワーク機器のログを収集する手法として古くから syslog プロトコル [6, 7] が広く利用されてきた。様々機器でサポートされているため、運用の汎用性が高い反面、記録される情報の形式が統一されておらず、解析が困難であるという問題を抱えている。syslog を活用するための手法として、これまで多くのテンプレート手法が提案されてきた ([8, 9, 10, 11, 4])。テンプレート化とは、自由形式で記述された syslog メッセージを固定語部分と変動語部分に分離し、メッセージを分類する作業を意味する。時に数千万行にも及ぶメッセージをひとつひとつ取り扱うのではなく、特定の（通常は元のメッセージ数に対して非常に少ない）種類のメッセージとして取り扱うことで、メッセージの出現頻度などを観察することが可能となる。

テンプレート手法は、1 パス手法と 2 パス手法に大きく分類される。1 パス手法はオンライン手法とも呼ばれ、事前の知識なく、新たに入力されてくるメッセージを分類していく手法である。これは、メッセージの数が膨大であったり、メッセージの構成が日々変化していくようなシステム、例えばクラウドシステムなどにおいて有用な手法となる。2 パス手法は、一旦すべてのメッセージを解析し、出現頻度の高い語（多くの場合は固定語と予想される）と出現頻度の低い語などを事前に調べることにより、より高精度なテンプレートを作成する。一般的に 2 パス手法の方が間違いの少ないテンプレートを作成可能であるが、事前の解析が必要となるため、適用できる場面が限られる。

本研究では、syslog のメッセージを構成する各単語の長さに注目し、そこから 1 パスでのテンプレート生成技術の実現可能性を検討した。なお、詳細な内容はテクニカルレポートとして公開している ([12])。

```
1    Nov  1 00:01:13 backup sshd[13883]: Connection closed by 211.99.249.89 [preauth]
2    Nov  1 00:10:01 backup CRON[13914]: pam_unix(cron:session): session opened for user root by (uid=0)
3    Nov  1 00:10:01 backup CRON[13916]: (root) CMD (/root/bin/backup-sh1.sh)
4    Nov  1 00:10:35 backup sshd[13922]: Connection closed by 211.99.249.89 [preauth]
5    Nov  1 00:11:05 backup CRON[13914]: pam_unix(cron:session): session closed for user root
6    Nov  1 00:17:01 backup CRON[13949]: pam_unix(cron:session): session opened for user root by (uid=0)
7    Nov  1 00:17:01 backup CRON[13951]: (root) CMD (   cd / && run-parts --report /etc/cron.hourly)
8    Nov  1 00:17:01 backup CRON[13949]: pam_unix(cron:session): session closed for user root
9    Nov  1 00:19:58 backup sshd[13963]: Connection closed by 211.99.249.89 [preauth]
10   Nov  1 00:20:01 backup CRON[13966]: pam_unix(cron:session): session opened for user root by (uid=0)
11   Nov  1 00:20:01 backup CRON[13967]: (root) CMD (/root/bin/backup-playground.sh)
12   Nov  1 00:20:09 backup CRON[13966]: pam_unix(cron:session): session closed for user root
13   Nov  1 00:25:41 backup sshd[13994]: Invalid user admin from 222.186.190.71
14   Nov  1 00:25:41 backup sshd[13994]: input_userauth_request: invalid user admin [preauth]
```

図 3: syslog メッセージ例

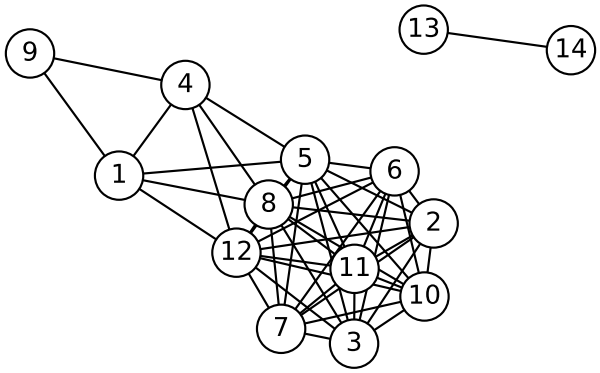



図 4: 図 3 を共有ワードグラフに変換

## 4.1 野生の syslog の観察

### 4.1.1 単語長の観察

syslog メッセージはプログラムによって出力されるため、その形式は事前に定義されたテンプレートに従っている。図 6 に例を示す。

既存研究でも説明されている通り、メッセージは固定部と変動部から構成されている。図 6 の最初の行では、日付とホスト名の部分を除外すると、`sshd`, `6845`, `vyatta`, and `41.190.192.158` が変動部であり、`Invalid`, `user`, and `from` が固定部となる。既存研究では、これらの部分、例えば出現頻度、文字種の比率など、事前に定義された条件によって分類することが多い。我々の疑問は、この分類にそれほど複雑な手法が必要かどうかという点にあった。

図 7 は類似するメッセージを図示したものである。人間が読めば、最初のメッセージ群が “`postfix/cleanup[*]: *: message-id=*`”2、ふたつめのメッセージ群が “`sshd[*]: Invalid user * from *`” に分類されることは明らかである。我々にはプロセス識別子、IP アドレス、メッセージ識別子がなんであるか、またどういった形式かという事前知識があり、どの部分が固定でどの部分が変動部であるかを類推できるためである。しかし、そういった前提知識がない場合でも上記の例には明らかな特徴として単語の長さが挙げられる。固定部の長さは常に一定であることは自明であるし、変動部も、その単語が共通の意味を持っているのであれば、おおよそ同じような長さになると考えられる。

図 8 は一部のメッセージを取り上げ、そこに含まれている単語の長さの分布を調べたものである。図からわかる通り、それぞれの syslog メッセージは特徴的な単語長分布となっている。メッセージの最初の単語は通常はプロセス名であり固定長となる。2 番めはプロセス識別子であり数桁の数値が入ることが多い。図 8(a) の 7 番めの単語は IP アドレスが書かれている部分であり、この部分は変動幅が大きくなっている。IP アドレス部にはホスト名が書かれることや、IPv6 の長い文字列が書かれることがあるためである。同様に、図 8(b) の 9 番めの位置にはホスト名が書かれている。この例では、ほとんどの場合 29 語のホスト名であったようだが、まれに短いホスト名や長いホスト名が入っていたことがわかる。

### 4.1.2 単語の位置の観察

単語の長さの分布がメッセージの特徴を示していることが観測されたが、これだけでは精度の高い分類ができないこともわかっている。

図 9 ふたつの異なるメッセージの単語長の分布を示したものである。本提案では、単語の長さの分布をベ

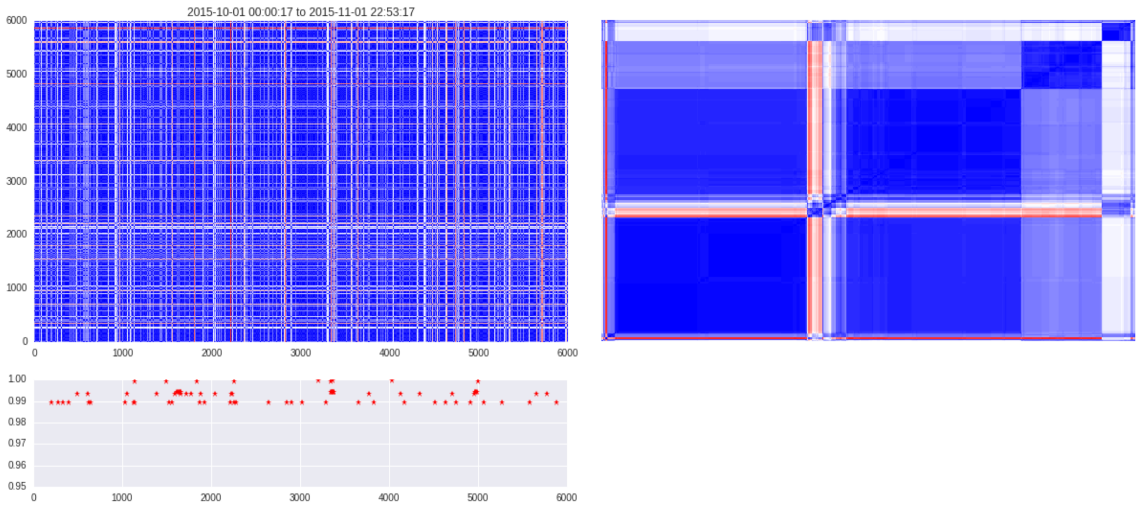



図 5: あるサーバーの共有ワードグラフを用いた状態変化の視覚化例

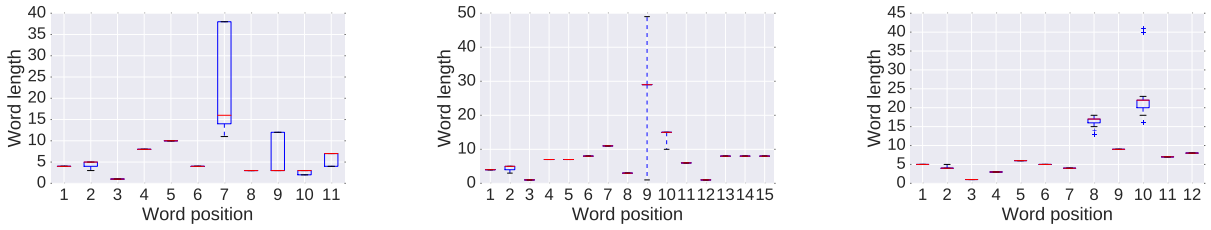
```
Oct  1 00:12:51 backup sshd[6854]: Invalid user vyatta from 41.190.192.158
Oct  1 00:12:51 backup sshd[6854]: input_userauth_request: Invalid user vyatta [preauth]
Oct  1 01:02:55 backup CRON[7069]: pam_unix(cron:session): session closed for user root
```

図 6: syslog メッセージの例

```
Dec  1 00:05:01 vm1.example.com postfix/cleanup[2767]: 7EF561405E3:
message-id=<20151130150501.7EF561405E3@vm1.example.com>
Dec  1 00:10:01 vm1.example.com postfix/cleanup[3247]: 898FD1405E3:
message-id=<20151130151001.898FD1405E3@vm1.example.com>

Dec  1 00:27:27 backup sshd[15406]: Invalid user admin from 222.186.30.174
Dec  1 04:29:58 backup sshd[16287]: Invalid user a from 218.38.12.218
```

図 7: メッセージグループの例



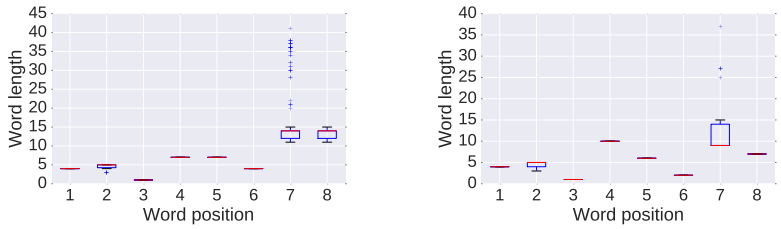

(a) `sshd * : Received disconnect from * 11: * * *` (79182 samples)

(b) `sshd * : reverse mapping checking getaddrinfo for * * failed - POSSIBLE BREAK-IN ATTEMPT!` (11926 samples)

(c) `named * : DNS format error from * resolving * invalid response` (878 samples)

図 8: 単語長の分布の例

クトルとみなし、ベクトルのコサイン近似値で類似度を判定するが、図に示したメッセージは、元のメッセージが全く異なるものの、コサイン近似値的に非常に近い値となる例である。




(a) `sshd * : refused connect from * *` (762 samples)　(b) `sshd * : Connection closed by * preauth` (16192 samples)

図 9: 全く異なるふたつのメッセージの単語長分布が近似してしまう例

人の目で見れば、このふたつのメッセージが異なることは明らかである。メッセージの共通点は最初の単語が一致していることくらいであり、続く単語群は全く異なっている。我々は、コサイン近似値による類似度判定とともに、同じ位置にある単語がどれくらい共有されているかも類似度判定に組み入れることで精度の向上を目指した。

## 4.2 LenMa: Length Matters Clustering

4.1.1 節での観察に基づき、我々はメッセージ間の類似性を比較する際に各単語の長さに注目することにする。

オンラインでメッセージを分類する場合、新しく入力されたメッセージが既存のどの分類に属するのか、あるいは新規に分類を作成すべきなのかを判断しなければならない。

図 7 の一つ目のグループの最初の行は次に示すベクトルによって表現できる。

$$\begin{aligned}[&len(\texttt{postfix/cleanup}),\\ &len(\texttt{2767}),\\ &len(\texttt{7EF561405E3}),\\ &len(\texttt{message-id}),\\ &len(\texttt{<201511...}(snip)\texttt{...example.com>})]\\ &= [15, 4, 11, 10, 44]\end{aligned}$$

これは、二行目のメッセージから生成されるベクトルと同じものである。二つ目のグループに関しては、最初の行のベクトルが $[4, 5, 7, 4, 5, 4, 13]$ であり、二行目のベクトルが $[4, 5, 7, 4, 1, 4, 13]$ となる。もし二つのベクトルが近似していれば、元となった二つのメッセージは同じ分類に属すると判定できる。

近似指数 $S_c$ はコサイン近似値を用いて式 (2) で定義される。

$$\begin{aligned}\boldsymbol{V}_c &= [v_{c,0}, v_{c,1}, ...v_{c,n}]\\ \boldsymbol{V} &= [v_0, v_1, ...v_n]\\ S_c &= CosineSimilarity(\boldsymbol{V}_c, \boldsymbol{V})\\ &= \frac{\boldsymbol{V}_c \cdot \boldsymbol{V}}{|\boldsymbol{V}_c|\ |\boldsymbol{V}|}\\ &= \frac{\sum_{i=0}^{n} v_{c,i} v_i}{\sqrt{\sum_{i=0}^{n} v_{c,i}^2}\sqrt{\sum_{i=0}^{n} v_i^2}}\end{aligned} \tag{2}$$

$\boldsymbol{V}_c$ と $\boldsymbol{V}$ は、それぞれ分類 $c$ と入力メッセージの単語長ベクトルを表す。$v_{c,i}$ と $v_i$ は $i$ 番目の単語の長さを表す。

ある分類を代表する単語長ベクトルは、新しいメッセージが統合されるたびに更新される。新しく入力されるメッセージの単語長ベクトルはアルゴリズム 1 で計算される。もし新規メッセージの $i$ 番目の単語長が既存の分類を代表する単語長ベクトルの $i$ 番目と異なっていれば、新しい値で上書きされる。

単語長ベクトルと同様に、単語の場所を示す単語ベクトルもアルゴリズム 2 の手順に従って管理、更新される。$\boldsymbol{W}_c$ 及び $\boldsymbol{W}$ は、それぞれ分類 $c$ と入力メッセージの単語の順序付きリストを表す。

新しいメッセージが入力されると、以下の手順が実行される。

1. 新しいメッセージの単語長ベクトルと単語ベクトルを生成する

**Algorithm 1** 単語長ベクトルの更新

**procedure** UPDATEWORDLENGTHVECTOR($\boldsymbol{V}_c, \boldsymbol{V}$)
  **for all** $v_{c,i} \in \boldsymbol{V}_c, v_i \in \boldsymbol{V}(i \leftarrow 1 \cdots |\boldsymbol{V}_c|)$ **do**
    **if** $v_{c,i} \neq v_i$ **then**
      $v_{c,i} \leftarrow v_i$
    **end if**
  **end for**
**end procedure**

**Algorithm 2** 単語ベクトルの更新

**procedure** UPDATEWORDVECTOR($\boldsymbol{W}_c, \boldsymbol{W}$)
  **for all** $w_{c,i} \in \boldsymbol{W}_c, w_i \in \boldsymbol{W}(i \leftarrow 1 \cdots |\boldsymbol{W}_c|)$ **do**
    **if** $w_{c,i} \neq w_i$ **then**
      $w_{c,i} \leftarrow *$
    **end if**
  **end for**
**end procedure**

2. 新しいメッセージと既存の分類間の近似指数を計算する
3. 近似指数が事前に定義された閾値 $T_c$ に満たない場合、新しい分類が作成され、入力されたメッセージは新しい分類に統合される
4. そうでない場合は、アルゴリズム 1 と 2 の手順に従ってもっとも近似指数の高かった分類に統合される。

アルゴリズム 3 は上記の手順を定式化したものである。

4.1.2 節で議論した通り、コサイン近似値だけで適切な分類を判断することができない場合が存在するため、本提案では固定部の単語がどれくらい共通しているかを比較し、分類の補助とする。そのため、新たに位置近似指数 $S_p$ を導入した。位置近似指数は固定部で共有された単語の数に関連する値であり、式 (3) で計算される。

$$S_p = |\{w_{c,i} = w_i(w_{c,i} \in \boldsymbol{W}_c, w_i \in \boldsymbol{W})\}| \quad (3)$$

$\boldsymbol{W}_c$ 及び $\boldsymbol{W}$ は分類 $c$ と入力メッセージの単語ベクトル、$i$ はメッセージの中での単語の出現位置を示す。

既存の分類と新規メッセージを比較する際、共通する単語がどの程度あるかを同時に確認する。もしその値が事前に定義された閾値 $T_p$ よりも小さい場合、新規メッセージは既存の分類には属さないと判断される。

**Algorithm 3** 分類の検索または新規作成

$\boldsymbol{C} \leftarrow \emptyset$
**procedure** FINDORCREATECLUSTER($message$)
  $\boldsymbol{V} \leftarrow$ CREATEWORDLENGTHVECTOR($message$)
  $\boldsymbol{W} \leftarrow$ CREATEWORDVECTOR($message$)
  $\boldsymbol{Cand} \leftarrow \emptyset$
  **for all** $[\boldsymbol{V}_c, \boldsymbol{W}_c]$ in $\boldsymbol{C}$ **do**
    **if** SIMILARITY($\boldsymbol{V}_c, \boldsymbol{V}, \boldsymbol{W}_c, \boldsymbol{W}$) $> T_c$ **then**
      $\boldsymbol{Cand} \leftarrow \boldsymbol{Cand} \cap [\boldsymbol{V}, \boldsymbol{W}]$
    **end if**
  **end for**
  **if** $\boldsymbol{Cand} = \emptyset$ **then**
    $\boldsymbol{C} \leftarrow \boldsymbol{C} \cap [\boldsymbol{V}, \boldsymbol{W}]$
    **return** $[\boldsymbol{V}, \boldsymbol{W}]$
  **end if**
  $[\boldsymbol{V}_c, \boldsymbol{W}_c] \leftarrow$ HIGHESTSIMILARITY($\boldsymbol{Cand}$)
  UPDATEWORDLENGTHVECTOR($\boldsymbol{V}_c, \boldsymbol{V}$)
  UPDATEWORDVECTOR($\boldsymbol{W}_c, \boldsymbol{W}$)
  **return** $[\boldsymbol{V}_c, \boldsymbol{W}_c]$
**end procedure**

## 4.3 プロトタイプ実装

我々は提案手法を実装し、既存オンラインテンプレート技術の一つである SHISO [11] と比較した。対象とした 3 種類の出自の異なるデータセットに対して妥当な数のテンプレート生成が可能であることがわかる。SHISO も LenMa も閾値ベースでのテンプレート生成アルゴリズムであるため、実際に生成されるテンプレート数は閾値によって変化し、またその閾値のあたいは対象となるデータセットに依存する。今回、LenMa の実行時に $T_c$ として 0.9 を、$T_p$ として 3 を用いている。

[1]SHISO は二段階のテンプレート生成を提案しており、第二段階で複数のテンプレートを統合する処理を実施しているが、本提案では第一段階の基本テンプレートの生成部分のみで比較している。

**Algorithm 4** 近似指数の計算

**procedure** SIMILARITY($\boldsymbol{V}_c, \boldsymbol{V}, \boldsymbol{W}_c, \boldsymbol{W}$)
  **if** $|\boldsymbol{V}_c| \neq |\boldsymbol{V}|$ **then**
    **return** 0
  **end if**
  **if** $\boldsymbol{W}_c$ matches $\boldsymbol{W}$ **then**
    **return** 1
  **end if**
  $S_c \leftarrow$ COSINESIMILARITY($\boldsymbol{V}_c, \boldsymbol{V}$)
  $S_p \leftarrow |\{w_{c,i} = w_i (w_{c,i} \in \boldsymbol{W}_c, w_i \in \boldsymbol{W})\}|$
  **if** $S_p < T_p$ **then**
    **return** 0
  **end if**
  **return** $S_c$
**end procedure**

表 1: 分類結果

| データセット | 生成されたテンプレート数 | |
|---|---|---|
| | SHISO[1] | LenMa |
| Public Securiy Log Sharing Site[13] | 29 | 26 |
| WIDE Cloud | 1093 | 1075 |
| iijlab servers | 1113 | 891 |

### 4.4 まとめ

テンプレートの生成は syslog メッセージの解析において頻繁に利用される手法である。今回、我々は syslog メッセージを観察することにより、単語の長さの分布がメッセージの分類に関連していることを発見し、この事実を用いたテンプレート生成手法を提案した。単語長と単語の比較のみを用いた手法のため、既存のテンプレート作成方法よりも若干計算量を減らすことができる。今後はテンプレート化されたメッセージを対象とした異常検出などに取り組んでいく予定である。

## 5 今後の展開

今年度はシステムログを用いた異常検知手法の研究開発を実施した。昨年度から引き続き、PC アルゴリズムを用いた因果推定の研究を進め、進捗を ACM CoNEXT の学生ワークショップにて発表した [14]。また、よりオンライン処理に適した syslog 解析手法を模索するため、テンプレート化を用いる既存の方法とは異なる、共有ワードグラフを用いた状態クラスタリングの手法の研究に着手した。引き続き、大規模化するシステムの状態把握のための精度の高い異常検出技法を研究していく。

## 参考文献


[1] P. Spirtes et al. 2nd edition. The MIT Press, 2000.

[2] R. E. Neapolitan. Prentice Hall Upper Saddle River, 2004.

[3] Satoru Kobayashi, Kensuke Fukuda, and Hiroshi Esaki. Towards an NLP-based log template generation algorithm for system log analysis. In *Proceedings of The Ninth International Conference on Future Internet Technologies - CFI '14*, pp. 1–4, 2014.

[4] 小林論. システムログ解析に基づく異常検出・原因究明技術に関する研究. Master's thesis, 東京大学, 2015.

[5] Apostolos N. Papadopoulos and Yannis Manolopoulos. Structure-Based Similarlity Search with Graph Histograms. In *Proceedings of the tenth International Workshop on Database and Expert Systems Applications (DEXA'99)*, pp. 174–178, 1999.

[6] Chris Lonvick. *The BSD syslog Protocol*. IETF, August 2001. RFC3164.

[7] Rainer Gerhards. *The Syslog Protocol*. IETF, March 2009. RFC5424.

[8] Risto Vaarandi. A data clustering algorithm for mining patterns from event logs. In *Proceedings of the 2003 IEEE Workshop on IP Operations and Management (IPOM 2003)*, pp. 119–126, 2003.

[9] Risto Vaarandi and Mauno Pihelgas. LogCluster - A Data Clustering and Pattern Mining Algorithm for Event Logs. In *Proceedings of the 11th International Conference on Network and Service Management (CNSM 2015)*, pp. 1–7, 2015.

[10] Wei Xu, Ling Huang, Armando Fox, David Patterson, and Michael I. Jordan. Detecting Large-Scale System Problems by Mining Console Logs. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles (SOSP'09)*, Vol. 4, pp. 117–132, 2009.

[11] Masayoshi Mizutani. Incremental Mining of System Log Format. In *Proceedings of the IEEE International Conference on Services Computing (SCC 2013)*, pp. 595–602, 2013.

[12] Keiichi Shima. Length matters: Clustering system log messages using length of words. *CoRR*, Vol. abs/1611.03213, , 2016.

[13] Anton Chuvakin. Public Security Log Sharing Site. `http://log-sharing.dreamhosters.com/`, 2010. accessed Jun 2016.

[14] Satoru Kobayashi, Kensuke Fukuda, and Hiroshi Esaki. Causation mining in network logs. In *ACM SIGCOMM CoNEXT 2016 Student Workshop*, 2016.