

# ネットワークログメッセージの出力元関数の 自動特定手法の提案

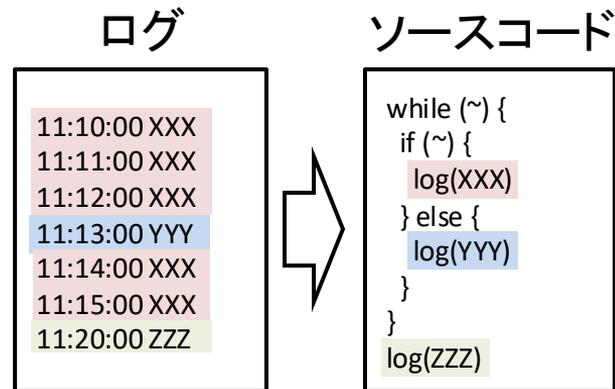
小林 諭<sup>1</sup>, Gaspard Damoiseau-Malraux<sup>2</sup>, 福田 健介<sup>3</sup>

<sup>1</sup>岡山大学, <sup>2</sup>ソルボンヌ大学, <sup>3</sup>国立情報学研究所

IA研究会 2025年3月3日

# 背景

- ネットワークトラブルシューティングにおけるログの調査
  - ログメッセージの記述のみでは何が起きたのか把握困難
  - OSSの場合、ソースコードの調査が必要
- ログメッセージとソースコードの対応付け
  - ログメッセージを出力した元のログ関数を特定する
  - 複数の既存技術の前提として活用される
  - システムの振る舞いを把握する上で有用



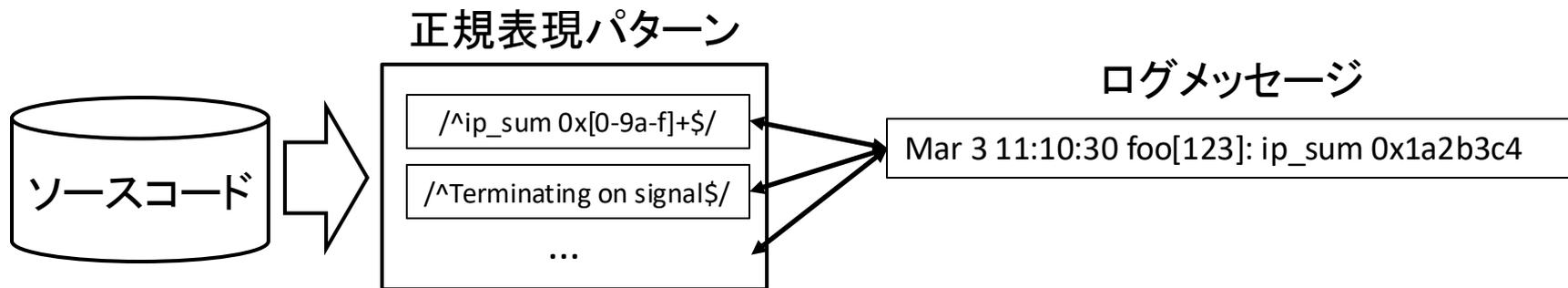
# ログ出力元関数を特定する難しさ

- ログ出力元関数にはログのフォーマットが含まれるので、特定に活用したい
  - 一般的な単語が多い場合、単純な単語検索では候補が多すぎて非効率
  - 単語間にフォーマット指定子がある場合、検索対象に含めることが難しい
- ログメッセージからログ出力元関数を自動特定したい

```
zlog_err("Instance %i out of range (0..%u)",  
         instance, (unsigned short)-1);
```

# 既存のアプローチとその課題

- ソースコードから抽出したログ出力元関数を正規表現パターンに自動変換し、入力のログメッセージと照合する [7]
- 課題: ログメッセージごとにすべての正規表現の候補パターンと照合するため、大きなプロジェクトだと処理時間が大きい



[7] W. Xu, et al. "Detecting large-scale system problems by mining console logs," Proceedings of SOSP '09, pp.117–132, 2009.

# 研究の目的

- ログメッセージを入力として対応するソースコード部分(ログ出力元関数)を自動特定するシステムの設計
  - ログメッセージと出力フォーマットの照合を高速に行うアプローチとして、正規表現によるマッチングとメッセージの単語分割下でのマッチングを組み合わせた新しい手法を提案
- 上記設計に基づく自動特定システムSCOLMを用いた評価
  - オープンソースルータFRRoutingを用いた処理時間の評価

# 既存研究

- 主に前処理としてログとソースコードの対応付けを実施
  1. ログメッセージの分類のための出力フォーマット抽出<sup>[7,8,9]</sup>
    - ログメッセージのみからの推定より信頼性が高い
  2. ログイベントの動作フロー上マッピング<sup>[11,13]</sup>
    - プログラムのソースコード上の振る舞いをログから可視化可能
- いずれも正規表現による手法<sup>[7]</sup>を利用、高速な対応付けについては議論していない

[7] W. Xu, et al. “Detecting large-scale system problems by mining console logs,” SOSP ’09, pp.117-132, 2009.

[8] D. Schipper, et al. “Tracing back log data to its log statement: from research to practice,” MSR’19, pp.545-549, 2019.

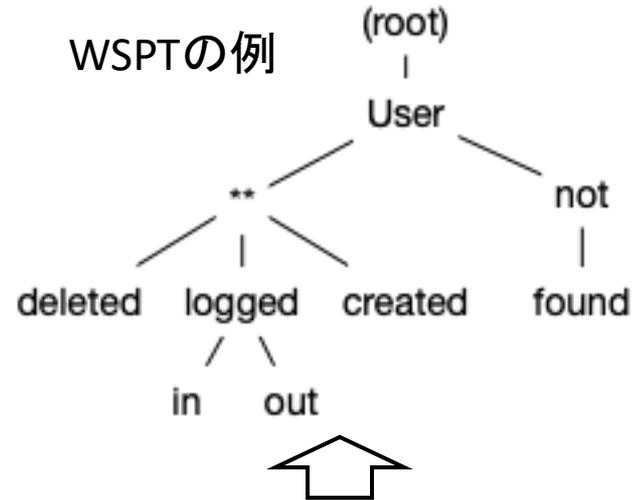
[9] A. Pecchia, et al. “Industry practices and event logging: assessment of a critical software development process,” ICSE’15, pp.169–178, 2015.

[11] D. Yuan, et al. “SherLog : Error Diagnosis by Connecting Clues from Run-time Logs,” SIGARCH Comput. Archit. News, vol.38, pp.143-154, 2010.

[13] Q. Fu, et al. “Execution anomaly detection in distributed systems through unstructured log analysis,” ICDM’09, pp.149–158, 2009.

# 既存のログ照合技術

- WSPT (Word Segmented Prefix Tree) [17]
  - 単語かワイルドカードをノードとして持つプレフィックス木
  - 1つのワイルドカードには1つの単語のみが入る前提(=セパレータを含まない)
    - これを満たす単語分割テンプレート(WS-template)の集合から構築可能
  - ログメッセージのテンプレート分類を正規表現の1/10以下の時間で処理可能 [17]

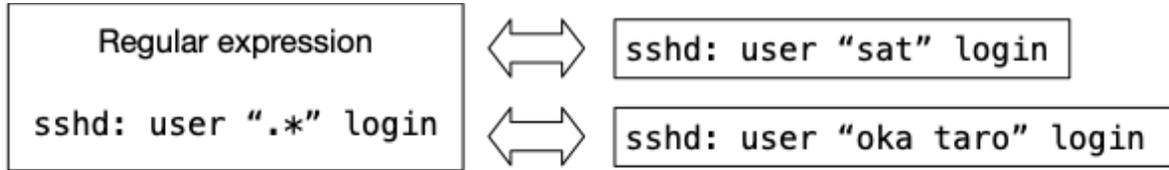


- User not found
- User Alice logged in
- User Bob created
- User Charlie detected

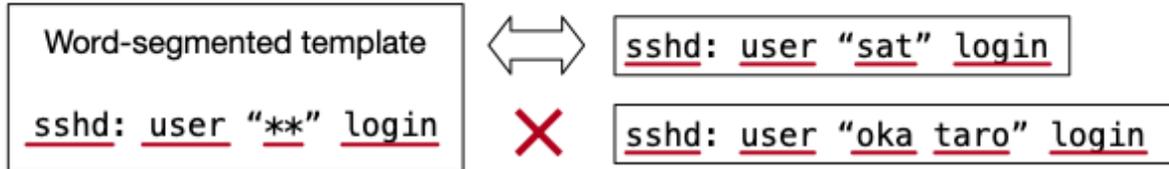
# WSPTの問題点

- ソースコードから抽出した出力フォーマットはそのままでは単語分割テンプレート(WS-template)として利用不可
  - フォーマット指定子には複数の単語が埋め込まれる
    - 「1つのワイルドカードに1つの単語のみが入る」前提を満たさない

正規表現の場合  
フォーマット指定子は  
複数単語とマッチする



WSPTの場合  
ワイルドカードは単一  
単語とのみマッチする



# 提案手法のアプローチ

- 正規表現とWSPTの併用
  - 実ログメッセージと正規表現パターンを対応づけ
    - 単語数が確定、WS-templateを生成可能
- WSPT -> 正規表現の順で照合
  - WSPTで発見 -> WS-templateに対応する正規表現のみ照合
  - WSPTで未発見 -> すべての正規表現を照合
    - 照合すべき正規表現をWSPTで絞り込み、処理を効率化

正規表現手法と同等の信頼性で、WSPTに準ずる処理速度を実現

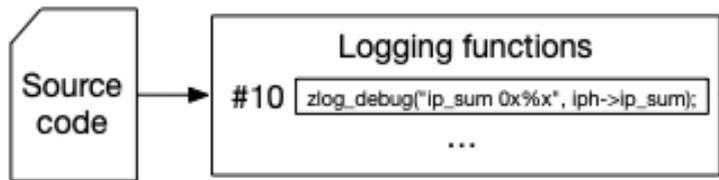
Method	Processing time	External template	False positives
Regular expression	Large	Available	Small
WSPT	Small	Not available	Large
Proposed method	Small	Available	Small

# 提案手法の流れ

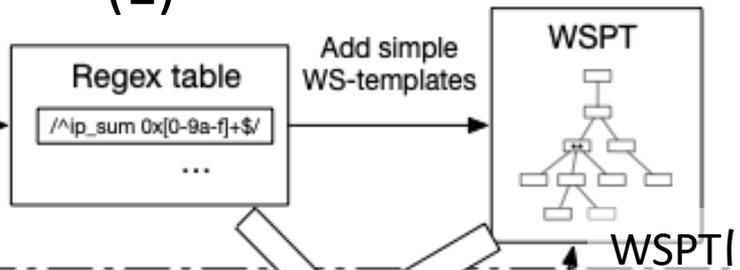
WSPTには単純なテンプレートのみ事前追加  
(高速化のための工夫)

事前にソースコードから  
正規表現パターン生成

Code parsing



(1) Template generation



WSPTに  
テンプレート追加

Preprocessing

Online process



Message parsing

入力ログメッセージのうち  
ヘッダを除くメッセージ部のみ利用

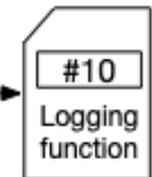
Refer

Generate  
WS-template



(2) Log matching

WSPTと正規表現を併用する  
ログメッセージの照合



Source code

# (1) テンプレート生成の手順

1. ソースコードからログ関数を抽出
2. フォーマット指定子を正規表現パターンに置き換え
3. フォーマット指定子につき1単語の単純なWS-templateを生成し、WSPTに追加

ログ関数

```
zlog_debug("ip_sum 0x%x", iph->ip_sum)
```

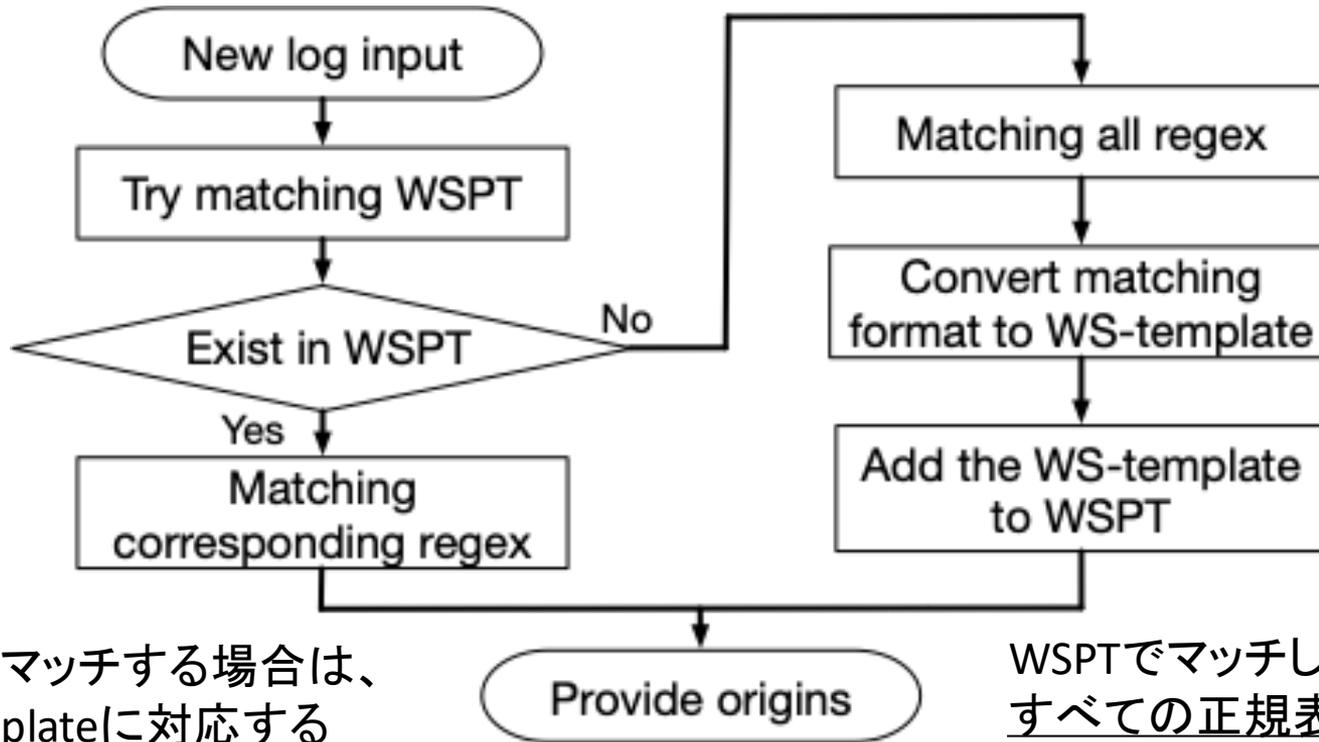
正規表現パターン

```
/^ip_sum\ 0x[0-9a-f]+$
```

(単純な) WS-template

```
ip_sum **
```

## (2) ログマッチングの手順



WSPTでマッチする場合は、WS-templateに対応する正規表現のみを照合

WSPTでマッチしない場合は、すべての正規表現を照合、WS-templateを生成・追加

# 実装

- SCOLM (Source Code Origins of Log Messages)
  - Python 3.10向けに実装
  - ソースコード分析部にはCtags[19]を利用
  - WSPTは、amulog[17, 20]をベースに実装
  - ログメッセージの単語分割にはlog2seq[17, 21]を利用
  - オープンソース公開を計画中

[17] S. Kobayashi, et al. “amulog: A general log analysis framework for comparison and combination of diverse template generation methods,” International Journal of Network Management, vol.32, no.4, p.e2195, 2022.

[19] Universal Ctags Team, “Universal ctags,” <https://ctags.io/>, 2021.

[20] S. Kobayashi, “amulog,” <https://github.com/amulog/amulog>.

[21] S. Kobayashi, “log2seq,” <https://github.com/amulog/log2seq>.

# 予備評価 – 方針

- 既存手法(正規表現)との処理時間の比較を行う
- オープンソースルータFrrouting<sup>[18]</sup>を対象とする
  - ソースコード: FRRouting バージョン8.5
  - ログメッセージ: DockerでFRRoutingを用いた模倣ネットワークを構築し、netroub<sup>[22,23]</sup>を用いた障害再現によりログを収集
    - 6,481のログ出力フォーマット、89,865行の入力ログメッセージ
- 評価環境
  - AMD Ryzen 7 7700, Ubuntu Server 22.04 (x86\_64), メモリ64GB

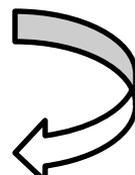
[18] FRRouting Project, “Frrouting,” <https://frrouting.org/>, 2017.

[22] C. Regal-Mezin, et al. “netroub: Towards an emulation platform for network trouble scenarios,” CoNEXT-SW 2023, pp.17–18, 2023

[23] C. Regal-Mezin, “netroub,” <https://github.com/3atlab/netroub>.

# 予備評価 – 処理時間

手法	処理時間 (秒)
正規表現 (既存手法)	517.19
SCOLM (提案手法)	<b>5.44</b>
SCOLM – WSPTの 初期状態が空	5.80
SCOLM – WSPTを すべて事前生成	5.25

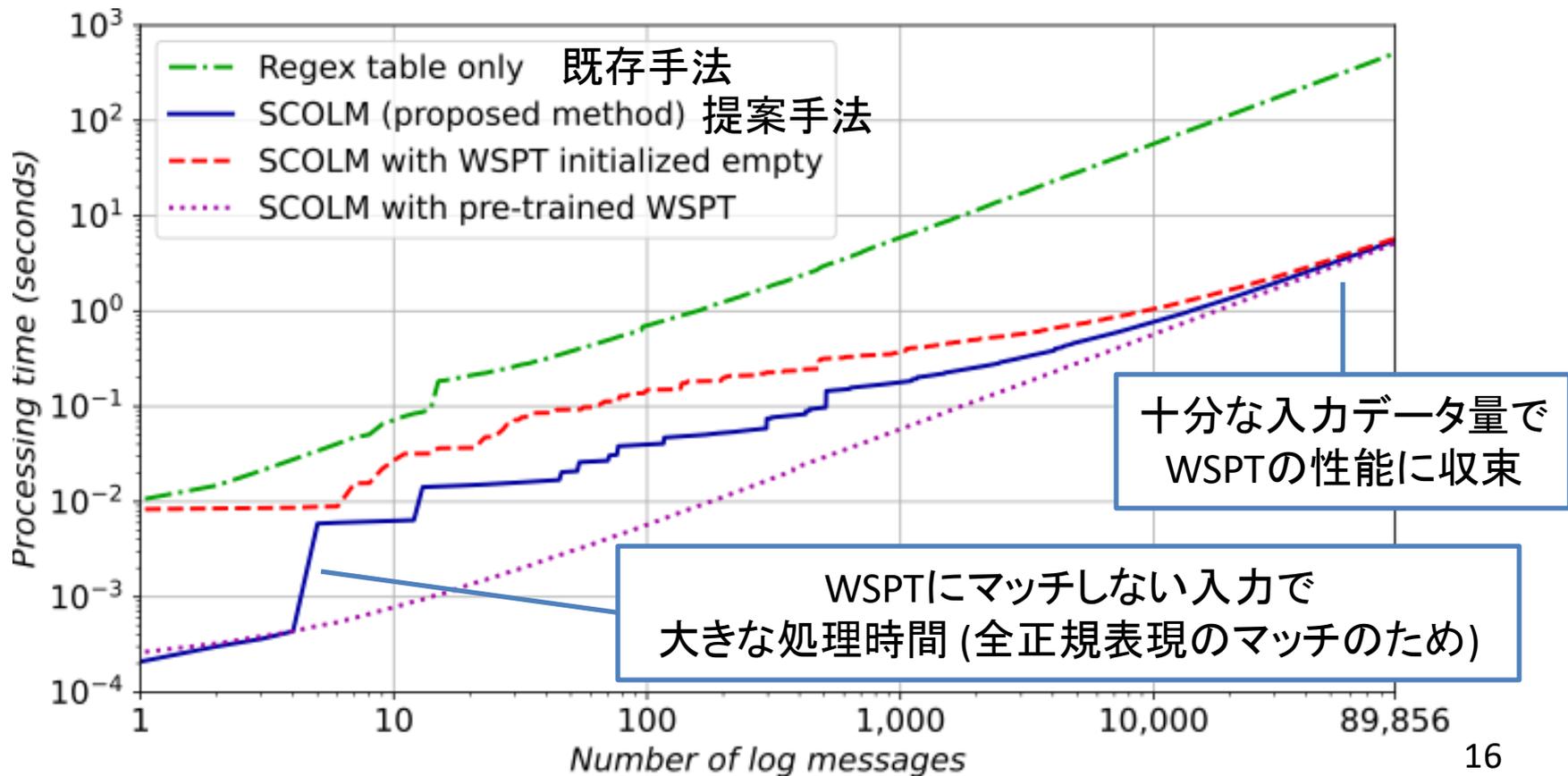


提案手法は既存手法に比べ  
**95倍**の速度で処理可能

単純なWS-templateの生成を  
行わない場合の性能

既に全てのWS-templateがWSPTに  
追加された状態で開始  
≡ WSPTのみの場合 (理想条件) 15

# 予備評価 – 処理時間の推移



```
demo@computer $ python
Python 3.10.12 (main, Nov 6 2024, 20:22:13) [GCC 11.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> |
```

# 動作デモ

# 議論

- 他ソフトウェアへの応用性
  - ソフトウェアの記述言語
    - ソースコード解析部分を拡張すれば、言語の制約はない
  - FRR同様、独自のログ出力元関数を持つ場合 (syslogのwrapper)
    - 事前に候補の関数名を与えれば有効 (例: zlog\_debug, ...)
  - ログのFormattingが出力元関数外で行われている場合
    - 例: 処理過程で文字列が順次結合されてログを生成している場合
    - 既存手法/提案手法のアプローチでは適切なフォーマットが得られない
    - ソースコードの静的解析などが必要か

# まとめ

- ネットワークのトラブルシューティング支援を目指し、ネットワークログの出力元関数を自動特定するシステムを設計
  - 正規表現とWSPTを組み合わせたログとソースコードの高速な照合技術を提案
- SCOLMとして実装、FRRoutingを対象に処理時間評価
  - Dockerベースの仮想ネットワークで収集したログを評価に利用
  - 既存手法と比較して95倍の速度で処理可能
- 今後の課題
  - 出力元関数特定の信頼性・精度の評価