



WYDZIAŁ ELEKTRONIKI,
TELEKOMUNIKACJI
I INFORMATYKI

Imię i nazwisko studenta: Mateusz Mazur
Nr albumu: 180352
Poziom kształcenia: Studia drugiego stopnia
Forma studiów: stacjonarne
Kierunek studiów: Informatyka
Specjalność: Algorytmy i technologie internetowe

PRACA DYPLOMOWA MAGISTERSKA

Tytuł pracy w języku polskim: Wykrywanie i wizualizacja znaczących informacji w logach aplikacji

Tytuł pracy w języku angielskim: Discovery and visualization of meaningful information in applications logs

Opiekun pracy: dr inż. Krzysztof Manuszewski

Do zrobienia: Abstrakt

SPIS TREŚCI

SPIS TREŚCI	4
1 WSTĘP	5
1.1 Struktura logów	6
2 PRZEGLĄD LITERATURY	8
3 ZAPROPONOWANE ROZWIĄZANIE	9
4 EKSPERYMENTY	10
5 PODSUMOWANIE	11
SPIS RYSUNKÓW	12
SPIS TABEL	13
BIBLIOGRAFIA	14

1. WSTĘP

Współczesne systemy informatyczne są złożone i często rozproszone pomiędzy różne niezależne komponenty, które działają asynchronicznie. Zrozumienie ich zachowań i interakcji w tych systemach staje się dużym wyzwaniem, zwłaszcza podczas diagnozowania problemów. Biorąc pod uwagę ograniczenia tradycyjnych narzędzi do rozwiązywania problemów, takich jak GDB, dla tak skomplikowanych środowisk, inżynierowie często uciekają się do bardziej dostępnej, a jednak skutecznej techniki: przechwytywania i zapisywania śladów aplikacji jako plików tekstowych, wzbogaconych o dodatkowe dane, takie jak znaczniki oznaczające czas. Te pliki, powszechnie znane jako "logi" lub "pliki logów", służą jako kluczowe zasoby do monitorowania, diagnozowania i rozumienia wewnętrznych mechanizmów współczesnych systemów informatycznych.

Pliki logów zazwyczaj przyjmują formę półstrukturalnych plików tekstowych, gdzie każda nowa linia odpowiada nowemu zdarzeniu aplikacji. Takie zdarzenie zazwyczaj również bezpośrednio wskazuje na konkretną lokalizację w kodzie źródłowym aplikacji. Dodatkowo, każda linia może zawierać dodatkowe informacje (1.1), takie jak znacznik czasu, powagę informacji, lokalizację itp. Problem jednak polega na tym, że nie ma ujednoliconego sposobu strukturyzowania takich plików, więc w praktyce większość aplikacji ma zupełnie różne struktury plików logów[1], m. in. w zależności od:

- Języka programowania
- Bibliotek cyfrowych
- Systemu operacyjnego
- Dostępnych zasobów

Logi stanowią nieocenione źródło danych dla większości nowoczesnych aplikacji, oferując głęboki wgląd w działanie aplikacji bez generowania znaczących dodatkowych kosztów. Ich wszechstronność sprawia, że znajdują zastosowanie w różnych rozwiązaniach technologicznych, od zintegrowanych systemów wbudowanych po zaawansowane rozproszone serwisy oraz usługi internetowe. Ta uniwersalność i bogactwo zawartych informacji sprawiają, że logi są przedmiotem intensywnych badań, zwłaszcza w kontekście ekstrakcji istotnych danych, które mogą pomóc w usprawnieniu procesów diagnostycznych, monitoringu wydajności oraz wykrywaniu potencjalnych problemów.

Analiza logów stanowi kluczowy element utrzymania i diagnozowania współczesnych systemów informatycznych. Logi te służą jako cenne źródło informacji dla programistów, administratorów systemów i zespołów wsparcia, umożliwiając im zyskanie wglądu w zachowanie aplikacji i skuteczne identyfikowanie problemów, bez potrzeby ponownego uruchamiania systemu. Niestety, pomimo powszechności logów we współczesnym oprogramowaniu, brak ustandaryzowanego formatu stwarza interesujące wyzwania w tej dziedzinie.

W tej pracy zaproponowano przykładowy system, próbujący rozwiązać część opisanych powyżej problemów, jednocześnie minimalizując ilość założeń początkowych, w celu uzyskania jak najlepszej uniwersalności. System, jako swoje wejście przyjmuje dwa pliki wejściowe:

- *Baseline* - Plik ten powinien posiadać logi z przebiegu aplikacji, w momencie gdy działała ona w sposób prawidłowy. Preferowane jest aby okres zbierania logów z systemu dla tego pliku, był podobny bądź lekko dłuższy od pliku *checked*.

- *Checked* - Ten plik zawierać powinien logi z przebiegu aplikacji, która zachowała się w nieporządku bądź niespodziewany sposób. Każda linijka w tym logu, w wyniku działania systemu, będzie posiadała odpowiadającą sobie linijkę w pliku *baseline* (o ile zostanie znaleziona) oraz heurystykę jak istotna jest dana linijka.

Dodatkowo użytkownik ma możliwość dostrojenia dodatkowych parametrów, w celu dalszego usprawnienia skuteczności systemu. Umożliwia to wykorzystanie wiedzy domenowej, kiedy użytkownik ma taką możliwość.

Postawionym zadaniem systemu jest **asysta** użytkownika, tak aby znalezienie istotnych informacji była ułatwiona w stosunku do innych, bardziej manualnych metod.

Do zrobienia: TEZA

1.1. Struktura logów

Jak wspomniano wcześniej, pliki logów często przyjmują format semi-strukturalny. Oznacza to, że w tym samym pliku, linijki logu powinny posiadać wspólną strukturę. Struktura ta opisuje to jak wygląda linijka oraz jakie informacje muszą w niej się znajdować (pomijając samą informację związaną z logiem). W praktyce dane takie obejmują informacje takie jak obecny czas, istotność wydarzenia (ang. Severity), nazwę modułu, etc.

Problem jednak polega na tym, że nie istnieje uniwersalny sposób na definiowanie powyższej struktury. Sprawia to więc, że zadanie polegające na jej automatycznym zrozumieniu przez program komputerowy jest nietrywialne. Zilustrować to można na dość prostym przykładzie, na którego potrzeby sfabrykowano przykładową linijkę 1.1, oznaczoną dalej jako l' 1.1. Ma ona dość łatwą strukturę z dwoma metadanymi: datą oraz informacją o ważności.

01.01.2024 INFO Client connected

Rysunek 1.1: Przykładowa linijka logu

Warto w tym miejscu zastanowić się, jaki wzór (ang. pattern) algorytm powinien wykryć w podanym (1.1) przykładzie. Dość prostym wzorem mógłby być: `<timestamp> <severity> <*> connected`, gdzie nazwy wokół trójkątnych nawiasów reprezentują parametry dla danej linijki logu. Oznacza to więc, że dla dowolnej linijki logu l , parser \mathbb{P} powinien umieć znaleźć w niej wzór $p = \mathbb{P}(l)$, przyjmujący jako argument parametry x , z wykorzystaniem których odtworzyć można daną linijkę logu $l = p(x)$. Operacje te, wykorzystując podany przykład, opisane zostały poniżej: 1.1, 1.2 i 1.3

$$l' = '01.01.2024 \text{ INFO Client connected}' \quad (1.1)$$

$$p', x' = \mathbb{P}(l') = '<timestamp> <severity> <*> connected', ['01.01.2024', 'INFO', 'Client'] \quad (1.2)$$

$$l' = p'(x') \quad (1.3)$$

Łatwo wyobrazić sobie linijkę logu l^* , która przekazuje identyczną informację co linijka l' , jednak ma inną strukturę.

Do zrobienia: Dokończyć przykład

```
01.01.2024 INFO Client connected  
01/01/24 Information [thread 0] - Client connected  
Info - 01 January 2024 - Client connected
```

Rysunek 1.2: Przykładowa linijka logu - inne warianty

2. PRZEGLĄD LITERATURY

Do zrobienia:

3. ZAPROPONOWANE ROZWIĄZANIE

Do zrobienia:

4. EKSPERYMENTY

Do zrobienia:

5. PODSUMOWANIE

Do zrobienia:

SPIS RYSUNKÓW

1.1 Przykładowa linijka logu	6
1.2 Przykładowa linijka logu - inne warianty	7

SPIS TABLIC

BIBLIOGRAFIA

- [1] Pratibha Sharma, Surendra Yadav i Brahmdutt Bohra. „A review study of server log formats for efficient web mining”. W: *2015 International Conference on Green Computing and Internet of Things (ICGCIoT)*. 2015, s. 1373–1377. doi: [10.1109/ICGCIoT.2015.7380681](https://doi.org/10.1109/ICGCIoT.2015.7380681).