# Logparser Documentation

*Release 0.1*

**LogPAI**

**Mar 30, 2021**

# Get Started

Logparser provides a toolkit and benchmarks for automated log parsing, which is a crucial step towards structured log analytics. By applying logparser, users can automatically learn event templates from unstructured logs and convert raw log messages into a sequence of structured events.

CHAPTER 1

# Demo

The logparser toolkit is implemented with Python and requires a number of dependency requirements installed. Users are encouraged to set up the local environment for logparser with Anaconda. However, for ease of reproducing our benchmark results, we have built docker images for the running evironments. Docker is a popular container technology used in production. If you have docker installed, you can easily pull and run docker containers as follows:

```
$ mkdir logparser
$ docker run --name logparser_py2 -it -v logparser:/logparser logpai/logparser:py2␣
↪bash
```

Note that if you are going to try MoLFI, which requires Python 3, please run the following container:

```
$ mkdir logparser
$ docker run --name logparser_py3 -it -v logparser:/logparser logpai/logparser:py3␣
↪bash
```

After starting the docker containers, you can run the demos of logparser on the HDFS sample log:

```
$ git clone https://github.com/logpai/logparser.git /logparser/
$ cd /logparser/demo/
$ python Drain_demo.py
```

The logparser demo/benchmark scripts will produce both event templates and structured logs in the result directory:

- HDFS_2k.log_templates.csv
- HDFS_2k.log_structured.csv

# CHAPTER 2

# Dependency

The logparser toolkit has the following requirements by default. We recommend users to use Anaconda, which is a popular Python data science platform with many common packages pre-installed.

- python 2.7
- scipy
- numpy
- scikit-learn
- pandas

Some tools have additional dependency requirements:

- SLCT: gcc 4.8.5
- LogCluster: perl 5.22
- MoLFI: python 3.6, deap 1.2.2
- POP: pyspark

CHAPTER 3

# Install docker

This is a note showing the steps of installing docker on Ubuntu platforms. If you need more detailed information, please check docker documentaion at: https://docs.docker.com/install/linux/docker-ce/ubuntu

**Note:** Uninstall old docker versions if any:

```
$ sudo apt-get remove docker docker-engine docker.io
```

- **Ubuntu 14.04**

  Install `linux-image-extra-*` to allow Docker to use the aufs storage drivers.

  ```
  $ sudo apt-get update

  $ sudo apt-get install \
      linux-image-extra-$(uname -r) \
      linux-image-extra-virtual
  ```

  Download docker package file docker-ce_17.03.2~ce-0~ubuntu-trusty_amd64.deb.

  ```
  $ sudo dpkg -i ~/docker/docker-ce_17.03.2~ce-0~ubuntu-trusty_amd64.deb
  ```

- **Ubuntu 16.04**

  Download docker package file docker-ce_17.03.2~ce-0~ubuntu-xenial_amd64.deb.

  ```
  $ sudo dpkg -i ~/docker/docker-ce_17.03.2~ce-0~ubuntu-xenial_amd64.deb
  ```

Verify that Docker CE is installed correctly by running the hello-world image:

```
$ sudo docker run hello-world
```

Add user to the docker group to run docker commands without sudo:

```
$ sudo groupadd docker
$ sudo usermod -aG docker $USER
```

# CHAPTER 4

# Build docker images

Build logpai/logparser:py2

```
$ sudo docker run --name logparser_py2 -it ubuntu:16.04 bash

$ apt-get update
$ apt-get install -y wget bzip2
$ apt-get install -y gcc perl git
$ rm -rf /var/lib/apt/lists/*

$ cd /
$ mkdir anaconda
$ cd anaconda
$ wget https://repo.anaconda.com/archive/Anaconda2-5.2.0-Linux-x86_64.sh
$ bash Anaconda2-5.2.0-Linux-x86_64.sh
$ source ~/.bashrc
$ cd ..
$ rm -r anaconda
$ exit

$ docker commit logparser_py2 logpai/logparser:py2
$ docker login
$ docker push logpai/logparser:py2
```

Build logpai/logparser:py3

```
$ sudo docker run --name logparser_py3 -it ubuntu:16.04 bash

$ apt-get update
$ apt-get install -y wget bzip2 git
$ rm -rf /var/lib/apt/lists/*

$ cd /
$ mkdir anaconda
$ cd anaconda
```

(continues on next page)

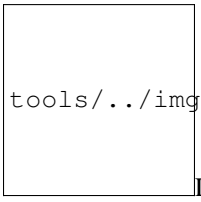
(continued from previous page)

```
$ wget https://repo.anaconda.com/archive/Anaconda3-5.1.0-Linux-x86_64.sh
$ bash Anaconda3-5.1.0-Linux-x86_64.sh
$ source ~/.bashrc
$ cd ..
$ rm -r anaconda

$ pip install deap
$ exit

$ docker commit logparser_py3 logpai/logparser:py3
$ docker login
$ docker push logpai/logparser:py3
```

CHAPTER 5

# Overview

Logparser aims to provide a set of open-source tools that are ready for use in production. By applying logparser, users can automatically learn event templates from unstructured logs and convert raw log messages into a sequence of structured events. The following figure illustrates an overview of log parsing.




Logparser overview

# CHAPTER 6

# SLCT

SLCT is a simple logfile clustering tool designed to find clusters in logfile(s), so that each cluster corresponds to a certain line pattern that occurs frequently enough. With the help of SLCT, one can quickly build a model of logfile(s), and also identify rare lines that do not fit the model (and are possibly anomalous).

**Step 1**: Word vocabulary construction. SLCT makes a pass over the words in all the logs and count the occurence of them. In this step, the position of the word is also considered. For example, "send" as the 1st word of a log and "send" as the 2nd word of a log are considered different. Word occurs more than support threshold, say N, is defined as frequent word.

**Step 2**: Cluster candidates construction. In this step, SLCT makes the second pass over all the logs, while at this time it focuses on frequent words. All the frequent words in a log will be extracted be the log template of itself. The number of logs that match a certain log template is counted, and each log template represents a cluster candidate.

**Step 3**: Log template extraction. SLCT goes through all cluster candidates and log templates whose corresponding cluster contains more than N logs are selected as the output templates. The logs of clusters which are not selected are placed into outlier class.

**Step 4**: Cluster combination. This step is optional. SLCT could make a pass through all selected clusters and combine two clusters if one of them is the subcluster of the other. For example, cluster "PacketResponder 1 for block * terminating" is the subcluster of "PacketResponder * for block * terminating". Therefore, these two clusters will be combined in this step.

To provide a common interface for log parsing, we write a Python wrapper around the original SLCT source code in C (released under GPL license). This also eases our benchmarking experiments. Same with the original release, our implementation has only been tested successfully on Linux (compiled with GCC). We tried running the tool on Windows using cygwin with GCC installed, but failed with a crash. You are advised to use the SLCT tool on Linux. But it is still possible to work around the issue if some efforts are made.

Read more information about SLCT from the following paper:

- Risto Vaarandi. A Data Clustering Algorithm for Mining Patterns from Event Logs, *IEEE Workshop on IP Operations & Management (IPOM)*, 2003.

# CHAPTER 7

# IPLoM

IPLoM (Iterative Partitioning Log Mining) is one of the state-of-the-art algorithms for log parsing. It leverages the unique characteristics of log messages for iterative log partitioning, which thus enables efficient message type extraction. Since the original open-source implementation is not available anymore, we re-implement the algorithm using Python with a nice interface provided. We describe the process of IPLoM as follows.

**Step 1**: Partition by event size. Logs are partitioned into different clusters according to its length. In real world logs, it is possible that logs belong to one template are in variable length. In this case, the result of IPLoM should be postprocessed manually.

**Step 2**: Partition by token posistion. At this point, each cluster contains logs with the same length. Assuming there are m logs whose length are n in a cluster, this cluster can be regarded as an m-by-n matrix. This step based on the assumption that the column with least number of unique words (split word position) is the one contains constants. Thus, the split word position is used to partition each cluster, i.e. each generated cluster has the same word in the split word position.

**Step 3**: Partition by search for mapping. In this step, two columns of the logs are selected for further partitioning based on the mapping relation between them. To determine the two columns, the number of unique words in each column is counted (i.e. word count) and the two columns with the most frequently appearing word count are selected. There are four mapping relations: 1-1, 1-M, M-1, M-M. In the case of 1-1 relations, logs contains the same 1-1 relations in the two selected columns are partitioned to the same cluster. For 1-M and M-1 relations, we should firstly decide whether the M side column contains constants or variables. If the M side contains constants, the M side column is used partition logs in 1-M/M-1 relations. Otherwise, the 1 side column is used. Finally, logs in M-M relations are partitioned to one cluster.

**Step 4**: Log template extraction. IPLoM processes through all the clusters generated in previous steps and generates one log template for each of them. For each column in a cluster, the number of unique words is counted. If there is only one unique word in a column, the word is regarded as constant. Otherwise, the words in the column are variables and will be replaced by a wildcard in the output.

Read more information about IPLoM from the following papers:

- Adetokunbo Makanju, A. Nur Zincir-Heywood, Evangelos E. Milios. Clustering Event Logs Using Iterative Partitioning, *ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)*, 2009.
- Adetokunbo Makanju, A. Nur Zincir-Heywood, Evangelos E. Milios. A Lightweight Algorithm for Message Type Extraction in System Application Logs, *IEEE Transactions on Knowledge and Data Engineering (TKDE)*,

2012.

# CHAPTER 8

# LKE

LKE (Log Key Extraction) is one of the representative algorithms for log parsing. It first leverages empirical rules for preprocessing and then uses weighted edit distance for hierarchical clustering of log messsages. After further group splitting with fine tuning, log keys are generated from the resulting clusters.

**Step 1**: Log clustering. Weighted edit distance is designed to evaluate the similarity between two logs, WED=\sum_{i=1}^{n}\frac{1}{1+e^{x_{i}-v}} . n is the number of edit operations to make two logs the same, x_{i} is the column index of the word which is edited by the i-th operation, v is a parameter to control weight. LKE links two logs if the WED between them is less than a threshold \sigma . After going through all pairs of logs, each connected component is regarded as a cluster. Threshold \sigma is automatically calculated by utilizing K-means clustering to separate all WED between all pair of logs into 2 groups, and the largest distance from the group containing smaller WED is selected as the value of \sigma .

**Step 2**: Cluster splitting. In this step, some clusters are further partitioned. LKE firstly finds out the longest common sequence (LCS) of all the logs in the same cluster. The rests of the logs are dynamic parts separated by common words, such as "/10.251.43.210:55700" or "blk_904791815409399662". The number of unique words in each dynamic part column, which is denoted as |DP| , is counted. For example, |DP|=2 for the dynamic part column between "src:" and "dest:" in log 2 and log 3. If the smallest |DP| is less than threshold \phi , LKE will use this dynamic part column to partition the cluster.

**Step 3**: Log template extraction. This step is similar to the step 4 of IPLoM. The only difference is that LKE removes all variables when they generate log templates, instead of representing them by wildcards.

Read more information about LKE from the following paper:

- Qiang Fu, Jian-Guang Lou, Yi Wang, Jiang Li. Execution Anomaly Detection in Distributed Systems through Unstructured Log Analysis, *IEEE International Conference on Data Mining (ICDM)*, 2009.

# CHAPTER 9

# LogSig

LogSig is a message signature based algorithm to generate system events from textual log messages. By searching the most representative message signatures, logSig categorizes log messages into a set of event types. logSig can handle various types of log data, and is able to incorporate human's domain knowledge to achieve a high performance. We implemented LogSig using Python with a nice interface for benchmarking purpose.

**Step 1**: Word pair generation. In this step, each log is converted to a set of word pairs. For example, "Verification succeeded for blk_904791815409399662" is converted to the following word pairs: (Verification, succeeded), (Verification, for), (Verification, blk_904791815409399662), (succeeded, for), (succeeded, blk_904791815409399662), (for, blk_904791815409399662). Each word pair preserves the order information of the original log.

**Step 2**: Clustering. LogSig requires users to determine the number of clusters, say k, which leads to k randomly partitioned clusters of logs at the beginning of clustering. In each iteration of clustering, LogSig goes through all the logs and move them to other clusters if needed. For each log, potential value, which is based on word pairs generated in step 1, is calculated to decide to which cluster the log should be moved. LogSig keeps clustering untill no log is decided to move in one iteration.

**Step 3**: Log template extraction. At this point, there are k clusters of logs. For each cluster, words in more than half of the logs are selected as candidate words of the template. To figure out the order of candidate words, LogSig goes through all the logs in the cluster and count how many times each permutation appears. The most frequent one is the log template of the cluster.

Read more information about LogSig from the following papers:

- Liang Tang, Tao Li, Chang-Shing Perng. LogSig: Generating System Events from Raw Textual Logs, *ACM International Conference on Information and Knowledge Management (CIKM)*, 2011.

# CHAPTER 10

# Spell

Spell is an online streaming method to parse logs, which utilizes a longest common subsequence based approach. The key observation is that, if we view the output by a log printing statement (which is a log entry) as a sequence , in most log printing statements, the constant that represents a message type often takes a majority part of the sequence and the parameter values take only a small portion. If two log entries are produced by the same log printing statement stat , but only differ by having different parameter values, the LCS of the two sequences is very likely to be the constant in the code stat , implying a message type.

Initially, the LCSMap list is empty. When a new log entry $e_i$ arrives, it is firstly parsed into a token sequence $s_i$ using a set of delimiters. After that, we compare $s_i$ with the LCSseq's from all LCSObjects in the current LCSMap, to see if $s_i$ "matches" one of the existing LCSseq's (hence, line id $i$ is added to the lineIds of the corresponding LCSObject), or we need to create a new LCSObject for LCSMap.

Spell's wordflow is as follows:

image-20180801205611035

Read more information about Drain from the following paper:

- Min Du, Feifei Li. Spell: Streaming Parsing of System Event Logs, *IEEE International Conference on Data Mining (ICDM)*, 2016.

# CHAPTER 11

# Drain

Drain is one of the representative algorithms for log parsing. It can parse logs in a streaming and timely manner. To accelerate the parsing process, Drain uses a fixed depth parse tree (See the figure below), which encodes specially designed rules for parsing.

Sturcture of parse tree in Drain

Drain first preprocess logs according to user-defined domain knowledge, ie. regex. Second, Drain starts from the root node of the parse tree with the preprocessed log message. The 1-st layer nodes in the parse tree represent log groups whose log messages are of different log message lengths. Third, Drain traverses from a 1-st layer node to a leaf node. Drain selects the next internal node by the tokens in the beginning positions of the log message.Then Drain calculate similarity between log message and log event of each log group to decide whether to put the log message into existing log group. Finally, Drain update the Parser Tree by scaning the tokens in the same position of the log message and the log event.

Read more information about Drain from the following paper:

- Pinjia He, Jieming Zhu, Zibin Zheng, and Michael R. Lyu. Drain: An Online Log Parsing Approach with Fixed Depth Tree, *IEEE International Conference on Web Services (ICWS)*, 2017.

# CHAPTER 12

# Papers

**A list of papers about log parsing**

1. [**ICSE'19**] Jieming Zhu, Shilin He, Jinyang Liu, Pinjia He, Qi Xie, Zibin Zheng, Michael R. Lyu. Tools and Benchmarks for Automated Log Parsing, *International Conference on Software Engineering (ICSE)*, 2019.
2. [**TDSC'18**] Pinjia He, Jieming Zhu, Shilin He, Jian Li, Michael R. Lyu. Towards Automated Log Parsing for Large-Scale Log Data Analysis, *IEEE Transactions on Dependable and Secure Computing (TDSC)*, 2018.
3. [**TKDE'18**] Min Du, Feifei Li. Spell: Online Streaming Parsing of Large Unstructured System Logs, *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 2018.
4. [**ICPC'18**] Salma Messaoudi, Annibale Panichella, Domenico Bianculli, Lionel Briand, Raimondas Sasnauskas. A Search-based Approach for Accurate Identification of Log Message Formats, *IEEE/ACM International Conference on Program Comprehension (ICPC)*, 2018.
5. [**SIGMOD'18**] Yihan Gao, Silu Huang, Aditya Parameswaran. Navigating the Data Lake with DATAMARAN: Automatically Extracting Structure from Log Datasets, *International Conference on Management of Data (SIGMOD)*, 2018.
6. [**MASCOTS'18**] Nicolas Aussel, Yohan Petetin, Sophie Chabridon. Improving Performances of Log Mining for Anomaly Prediction Through NLP-Based Log Parsing, *International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, 2018.
7. [**ICICS'18**] Zhiyuan Zhao, Chenxu Wang, Wei Rao. Slop: Towards an Efficient and Universal Streaming Log Parser, *International Conference on Information and Communications Security (ICICS)*, 2018.
8. [**ICWS'17**] Pinjia He, Jieming Zhu, Zibin Zheng, and Michael R. Lyu. Drain: An Online Log Parsing Approach with Fixed Depth Tree, *IEEE International Conference on Web Services (ICWS)*, 2017.
9. [**Benelearn'17**] Stefan Thaler, Vlado Menkovski, Milan Petkovic. Towards Unsupervised Signature Extraction of Forensic Logs, *The Twenty-Sixth Benelux Conference on Machine Learning (Benelearn)*, 2017.
10. [**ISDFS'17**] Stefan Thaler, Vlado Menkonvski, Milan Petkovic. Towards A Neural Language Model for Signature Extraction From Forensic Logs, *International Symposium on Digital Forensic and Security (ISDFS)*, 2017.

11. [**DSN'16**] Pinjia He, Jieming Zhu, Shilin He, Jian Li, Michael R. Lyu. An Evaluation Study on Log Parsing and Its Use in Log Mining, *IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2016.

12. [**ICDM'16**] Min Du, Feifei Li. Spell: Streaming Parsing of System Event Logs, *IEEE International Conference on Data Mining (ICDM)*, 2016.

13. [**KDD'16**] Animesh Nandi, Atri Mandal, Shubham Atreja, Gargi B. Dasgupta, Subhrajit Bhattacharya. Anomaly Detection Using Program Control Flow Graph Mining from Execution Logs, *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, 2016.

14. [**MILCOM'16**] Risto Vaarandi, Markus Kont, Mauno Pihelgas. Event Log Analysis with the LogCluster Tool. *IEEE Military Communications Conference (MILCOM)*, 2016.

15. [**arXiv'16**] Keiichi Shima. Length Matters: Clustering System Log Messages using Length of Words, *arXiv:1611.03213 (arXiv)*, 2016.

16. [**CIKM'16**] Hossein Hamooni, Biplob Debnath, Jianwu Xu, Hui Zhang, Guofei Jiang, Abdullah Mueen. LogMine: Fast Pattern Recognition for Log Analytics, *ACM International Conference on Information and Knowledge Management (CIKM)*, 2016.

17. [**IDDPSW'16**] Yining Zhao, Haili Xiao. Extracting Log Patterns from System Logs in LARGE, *IEEE International Parallel and Distributed Processing Symposium Workshops (IDPSW)*, 2016.

18. [**MILCOM'15**] Jing Ya, Tingwen Liu, Haoliang Zhang, Jinqiao Shi, Li Guo. An Automatic Approach to Extract the Formats of. Network and Security Log Messages, *IEEE Military Communications Conference (MILCOM)*, 2015.

19. [**CNSM'15**] Risto Vaarandi, Mauno Pihelgas. LogCluster - A Data Clustering and Pattern Mining Algorithm for Event Logs, *International Conference on Network and Service Management (CNSM)*, 2015.

20. [**CNSM'15**] Tatsuaki Kimura, Akio Watanabe, Tsuyoshi Toyono, Keisuke Ishibashi. Proactive Failure Detection Learning Generation Patterns of Large-Scale Network Logs, *International Conference on Network and Service Management (CNSM)*, 2015.

21. [**WISTP'15**] David Jaeger, Amir Azodi, Feng Cheng, Christoph Meinel. Normalizing Security Events with a Hierarchical Knowledge Base, *The 9th Workshop on Information Security Theory and Practice*, 2015.

22. [**WHL'14**] Xia Ning, Geoff Jiang, Haifeng Chen, Kenji Yoshihira. HLAera System for Heterogeneous Log Analysis, *SDM Workshop on Heterogeneous Learning (WHL)*, 2014.

23. [**MSR'14**] Ghazaleh Khodabandelou, Charlotte Hug, Rebecca Deneckere, Camille Salinesi. Unsupervised Discovery of Intentional Process Models from Event Logs, *International Working Conference on Mining Software Repositories (MSR)*, 2014.

24. [**CFI'14**] Satoru Kobayashi, Kensuke Fukuda, Hiroshi Esaki. Towards an NLP-based log template generation algorithm for system log analysis,*International Conference on Future Internet Technologies (CFI)*, 2014.

25. [**2014**] Basanta Joshi, Manoj ghimire. A Data Streaming Algorithm for Signature Generation and Clustering of Log Messages, *Logpoint Technical Report*, 2014.

26. [**BigMine'13**] Farhana Zulkernine, Patrick Martin, Wendy Powley, Sima Soltani, Serge Mankovskii, Mark Addleman. CAPRI: A Tool for Mining Complex Line Patterns in Large Log Data, *International Workshop on Big Data, Streams and Heterogeneous Source Mining: Algorithms, Systems, Programming Models and Applications (BigMine)*, 2013.

27. [**TKDE'12**] Adetokunbo Makanju, A. Nur Zincir-Heywood, Evangelos E. Milios. A Lightweight Algorithm for Message Type Extraction in System Application Logs, *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 2012.

28. [**CIKM'11**] Liang Tang, Tao Li, Chang-Shing Perng. LogSig: Generating System Events from Raw Textual Logs, *ACM International Conference on Information and Knowledge Management (CIKM)*, 2011.

29. [**Euro-Par'11**] Ana Gainaru, Franck Cappello, Stefan Trausan-Matu, Bill Kramer. Event Log Mining Tool for Large Scale HPC Systems, *InEuropean Conference on Parallel Processing (Euro-Par)*, 2011.

30. [**SIGOPS'10**] Kathleen Fisher, David Walker, Kenny Q. Zhu. Incremental Learning of System Log Formats, *ACM SIGOPS Operating Systems Review (SIGOPS)*, 2010.

31. [**MSR'10**] Meiyappan Nagappan, Mladen A. Vouk. Abstracting Log Lines to Log Event Types for Mining Software System Logs, *International Working Conference on Mining Software Repositories (MSR)*, 2010.

32. [**ICDM'10**] Liang Tang, Tao Li. LogTree: A Framework for Generating System Events from Raw Textual Logs, *InData Mining (ICDM)*, 2010.

33. [**KDD'09**] Adetokunbo Makanju, A. Nur Zincir-Heywood, Evangelos E. Milios. Clustering Event Logs Using Iterative Partitioning, *ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)*, 2009.

34. [**ICDM'09**] Qiang Fu, Jian-Guang Lou, Yi Wang, Jiang Li. Execution Anomaly Detection in Distributed Systems through Unstructured Log Analysis, *IEEE International Conference on Data Mining (ICDM)*, 2009.

35. [**SOSP'09**] Wei Xu, Ling Huang, Armando Fox, David Patterson, Michael Jordan. Mining Console Logs for Large-Scale System Problem Detection, *ACM SIGOPS Symposium on Operating Systems Principles (SOSP)*, 2009.

36. [**ISSRE'09**] Meiyappan Nagappan, Kesheng Wu, Mladen A. Vouk. Efficiently Extracting Operational Profiles from Execution Logs Using Suffix Arrays, *InSoftware Reliability Engineering (ISSRE)*, 2009.

37. [**JSME'08**] Zhen Ming Jiang, Ahmed E. Hassan, Gilbert Hamann, Parminder Flora. AAn Automated Approach for Abstracting Execution Logs to Execution Events, *Journal of Software Maintenance and Evolution: Research and Practice (JSME)*, 2008.

38. [**QSIC'08**] Zhen Ming Jiang, Ahmed E. Hassan, Parminder Flora, Gilbert Hamann. Abstracting Execution Logs to Execution Events for Enterprise Applications, *International Conference on on Quality Software (QSIC)*, 2018.

39. [**NOMS'08**] Risto Vaarandi. Mining Event Logs with SLCT and LogHound, *Network Operations and Management Symposium (NOMS)*, 2008.

40. [**INTELLCOMM'04**] Risto Vaarandi. A Breadth-First Algorithm for Mining Frequent Patterns from Event Logs, *Intelligence in Communication Systems (INTELLCOMM)*, 2004.

41. [**IPOM'03**] Risto Vaarandi. A Data Clustering Algorithm for Mining Patterns from Event Logs, *IEEE Workshop on IP Operations & Management (IPOM)*, 2003.

# CHAPTER 13

# Benchmarks

All the log parsers have been evaluated on loghub log samples. We report parsing accuracy as the percentage of accurately parsed log messages. Note that accuracy values above 0.9 are marked in bold, and the best accuracy results achieved are marked with *.

| Tools | HDFS | Hadoop | Spark | Zookeeper | Open-Stack | BGL | HPC | Thunder-bird |
|---|---|---|---|---|---|---|---|---|
| SLCT | 0.545 | 0.423 | 0.685 | 0.726 | 0.867 | 0.573 | 0.839 | 0.882 |
| AEL | **0.998** | 0.538 | **0.905** | **0.921** | 0.758 | **0.957** | **0.903** | **0.941** |
| IPLoM | **1*** | **0.954** | **0.920** | **0.962** | 0.871 | **0.939** | 0.824 | 0.663 |
| LKE | **1*** | 0.670 | 0.634 | 0.438 | 0.787 | 0.128 | 0.574 | 0.813 |
| LFA | 0.885 | **0.900** | **0.994** | 0.839 | 0.200 | 0.854 | 0.817 | 0.649 |
| LogSig | 0.850 | 0.633 | 0.544 | 0.738 | 0.866 | 0.227 | 0.354 | 0.694 |
| SHISO | **0.998** | 0.867 | **0.906** | 0.660 | 0.722 | 0.711 | 0.325 | 0.576 |
| LogClus-ter | 0.546 | 0.563 | 0.799 | 0.732 | 0.696 | 0.835 | 0.788 | 0.599 |
| LenMa | **0.998** | 0.885 | 0.884 | 0.841 | 0.743 | 0.690 | 0.830 | **0.943** |
| LogMine | 0.851 | 0.870 | 0.576 | 0.688 | 0.743 | 0.723 | 0.784 | **0.919** |
| Spell | **1*** | 0.778 | **0.905** | **0.964** | 0.764 | 0.787 | 0.654 | 0.844 |
| Drain | **0.998** | **0.948** | **0.920** | **0.967** | 0.733 | **0.963** | 0.887 | **0.955** |
| MoLFI | **0.998** | **0.957** | 0.418 | 0.839 | 0.213 | **0.960** | 0.824 | 0.646 |

| Tools | Win-dows | Linux | Mac | Android | HealthApp | Apache | OpenSSH | Proxifier |
|---|---|---|---|---|---|---|---|---|
| SLCT | 0.697 | 0.297 | 0.558 | 0.882 | 0.331 | 0.731 | 0.521 | 0.518 |
| AEL | 0.690 | 0.673 | 0.764 | 0.682 | 0.568 | **1*** | 0.538 | 0.518 |
| IPLoM | 0.567 | 0.672 | 0.673 | 0.712 | 0.822 | **1*** | 0.802 | 0.515 |
| LKE | **0.990** | 0.519 | 0.369 | **0.909** | 0.592 | **1*** | 0.426 | 0.495 |
| LFA | 0.588 | 0.279 | 0.599 | 0.616 | 0.549 | **1*** | 0.501 | 0.026 |
| LogSig | 0.689 | 0.169 | 0.478 | 0.548 | 0.235 | 0.582 | 0.373 | **0.967** |
| SHISO | 0.701 | 0.672 | 0.595 | 0.585 | 0.397 | **1*** | 0.619 | 0.517 |
| LogClus-ter | 0.713 | 0.629 | 0.604 | 0.798 | 0.531 | 0.709 | 0.426 | **0.951** |
| LenMa | 0.566 | 0.701 | 0.698 | 0.880 | 0.174 | **1*** | **0.925** | 0.508 |
| LogMine | **0.993** | 0.612 | 0.872 | 0.504 | 0.684 | **1*** | 0.431 | 0.517 |
| Spell | **0.989** | 0.605 | 0.757 | **0.919** | 0.639 | **1*** | 0.554 | 0.527 |
| Drain | **0.997** | 0.690 | 0.787 | **0.911** | 0.780 | **1*** | 0.788 | 0.527 |
| MoLFI | 0.406 | 0.284 | 0.636 | **0.788** | 0.440 | **1*** | 0.50 | 0.013 |

CHAPTER 14

# Contributors

- Jieming Zhu, The Chinese University of Hong Kong, currently at Huawei 2012 Labs.
- Pinjia He, The Chinese University of Hong Kong
- Shilin He, The Chinese University of Hong Kong
- Jinyang Liu, Sun Yat-Sen University