LogAno – A library for the analysis of and anomaly detection in unstructured log files

Christopher Vahl, Christian Pfeiffer, Sebastian Schmitz, Jochen Garcke
Fraunhofer Institute for Algorithms and Scientific Computing SCAI

June 5, 2019

Abstract

LogAno is a software library for analysis of and anomaly detection in unstructured log files. An essential feature of this library is the ability to convert unstructured into structured log files. For this purpose, filters for common parameter types and a component to detect remaining parameters by machine learning are included.

The anomaly detection component of the library uses structured log files as an input to calculate an anomaly score on a per line or per file basis.

1 The importance of logging

In today's software industry, monolithic architectures are more and more replaced by smaller, distributed services. Paradigms such as cloud computing and microservices are central to this development.

While small programs providing one or few well defined services are easy to understand in isolation, real-life systems frequently comprise a multitude of such programs, resulting in a high level of complexity and possibly unforeseen problems. A hard part of fixing issues in such a system consists of reconstructing the exact interactions and conditions under which the issue occurred. Logging often is the only way by which this information can be made available. But log files do not only provide crucial information for problem-solving in distributed systems; they additionally are a data source from which valuable business insights can be obtained.

Modern large-scale web services produce vast amounts of logs, which makes manual inspection and analysis unfeasible. Automated analysis techniques such as machine learning can be applied to solve this problem. Cheap storage to save the logs and cheap computing power to analyze them are both available.

2 Structured and unstructured logs

To efficiently apply automated analysis techniques to log files, they must be available in a structured format. However, most logs produced by today's software are unstructured.

An example for an unstructured log message is

```
2019-03-01 10:12 host-x: user jane_doe logged on succesfully
```

Using the structured JSON format, the above message might be written as

```
{{time: 2019-03-01 10:12},
 {event: logon attempt},
 {host: host-x},
 {user: jane_doe},
 {status: successful}}
```

The advantage of structured log messages is apparent: Parameter names (or keys) and parameter values can be accessed directly for further analysis, whereas in unstructured log messages, this information must be derived from the raw text.

Best practices on how to format log messages can be found here.

3 LogAno Log Templater

The central component of LogAno is the Log Templater. This tool derives log templates from log files and, by way of these templates, can convert unstructured to structured log files. This feature is essential for the automated analysis of large-scale log files, since today, most software produces logs in an unstructured format.

3.1 The concept of log templates

A log template is derived from a log message by replacing all parameters in the message by wildcards. A parameter is a part of the message which corresponds to a variable in the source code statement by which the message was produced. The following example illustrates the relation between log message, source code statement, and log template:

Log message	`2018-09-18 16:10`	`File`	`myfile.txt`	`written to`	`mydisk`
	timestamp	const	par	const	par
Log template		File	*	written to	*
Source code	`print('File', filename, 'written to', location)`				

Templates are derived directly from log files without requiring the source code of the log-producing application. This makes the Log Templater applicable to logs produced by both open and closed source software. The only data necessary are the log files to be analyzed.

3.2 Filters for common parameter types

Log Templater includes a set of filters for common parameter types, such as IPv4/v6 and MAC addresses, dates and times, and hexadecimal numbers. Additional filters can be added in form of regular expressions or as general string manipulations in C++ code. Filters are applied as a first step of parameter detection.

3.3 Automated detection of parameters

Parameters remaining after the filter step are detected automatically by the machine learning component of Log Templater. This component processes log messages online, meaning that the inputs to the detection are individual log messages. The set of log templates is updated after every processed log message and is available at any time.

Contrarily, an offline detection would need to process a complete sample of messages at once and cannot be updated by single messages. The benefit of an offline detection is the following: Information which can only be obtained from a large set of messages and not from individual ones can be used in the detection process.

Log Templater incorporates such information in its online detection by efficiently storing relevant information gained from previously analyzed messages and applying it to the templates in a consolidation step. Log Templater thus combines the advantages of online and offline detection.

3.3.1 Customization options

The standard configuration of Log Templater works reliably on a large variety of unstructured logs. Customization is possible when required, for instance when working with very complex log formats. Important customization options are:

- **Choice of delimiters**: Log Templater analyzes a message by splitting it into tokens. The delimiters at which a split occurs can be customized.
- **Aggressiveness of parameter detection**: Controls the amount of message parts which are detected as being parameters. This setting only has a noticeable effect when working with small log samples. With increasing sample size, the effect is counterbalanced by the consolidation step described above. Consequently this setting is useful when working with a small log sample containing an unusually high or low rate of parameters per message.

In cases where the above options are insufficient to produce satisfactory results, all low-level components of Log Templater can be adapted to the needs of the specific use case.

3.4 Log file conversion

Log Templater can use the templates derived from log samples to convert unstructured to structured log messages. A disadvantage of this conversion opposed to having structured messages directly produced by the logging software is the lack of meaningful parameter names. In general, it is not possible to derive these names from unstructured log messages. The converter outputs dummy values which then can be manually replaced by appropriate names.

The following example illustrates the conversion process. The initial unstructured message is:

```
2019-03-01 10:12 host-x: user john_doe logon succesful
```

After the conversion, the message is:

```
{{template: user * logon *},
 {time: 2019-03-01 10:12},
 {host: host-x},
 {par1: john_doe},
 {par2: successful}}
```

The parameter names for `time` and `host` could be inferred due to log format conventions, but `par1` and `par2` need to be manually replaced by `user` and `status`, respectively, if desired.

The converted log files can be analyzed by the Anomaly Detector or other programs capable of working with structured logs, such as ElasticSearch.

4 LogAno Anomaly Detector

The Anomaly Detector component of LogAno creates an anomaly score for individual log messages, complete log files or sets of log messages produced in a specific time interval. To do this, Anomaly Detector makes use of the log templates created by Log Templater and additionally needs a training sample of log messages. Ideally, this sample should be anomaly-free, but a small percentage of anomalies can be tolerated.

High anomaly scores indicate a high probability of unusual events. At the current development state of Anomaly Detector, these scores are best used by developers, administrators or analysts as a first filter to reduce the amount of log messages which need to be inspected manually.