International Journal of Swarm Intelligence Research
Volume 13 • Issue 1

Log File Template Detection as a Multi-Objective Optimization Problem

Mathi Murugan T., Sathyabama Institute of Science and Technology, India
E. Baburaj, Department of Computer Science Engineering, Marian Engineering College, India

ABSTRACT

There is a need for an automatic log file template detection tool to find out all the log messages through search space. On the other hand, the template detection tool should cope with two constraints: (1) it should not be too general, and (2) it should not be too specific. These constraints contradict to one another and can be considered as a multi-objective optimization problem. Thus, a novel multi-objective optimization-based log-file template detection approach named LTD-MO is proposed in this paper. It uses a new multi-objective-based swarm intelligence algorithm called chicken swarm optimization for solving the hard optimization issue. Moreover, it analyzes all templates in the search space and selects a Pareto front optimal solution set for multi-objective compensation. The proposed approach is implemented and evaluated on eight publicly available benchmark log datasets. The empirical analysis shows LTD-MO detects a large number of appropriate templates by significantly outperforming the existing techniques on all datasets.

KEYWORDS

Chicken Swarm Optimization, Log Messages, Log Parsing, Multi-Objective Optimization, Pareto-Front, Software Management, Swarm Intelligence, Template Detection

INTRODUCTION

Nowadays, large-scale computer processors are comprised of number of software applications and components running on thousands of operating nodes. The runtime statistics of these processors are continuously gathered and accumulated in the form of log files and analyzed thereof to detect the cause and exact location of issues during system failure and malfunctioning (Bao et al., 2018). In general, runtime storage or logging is a usual process to store system functional data helpful for developers as well as support engineers to analyze the behavior of systems and track down the difficulties that may arise in the future. Thus, log files play an essential role in the maintenance and development of software-based computing systems. Additionally, the rich data present in the log files facilitate a huge variety of system analytic practices like ensuring software security (Latib et al., 2018), analyzing application statistics (Patel & Parikh, 2017), detecting performance anomalies (Vaarandi et al., 2018), identifying crashes and errors (Suman et al., 2018; Adam et al., 2016) and so on. In spite of the remarkable data available in logs, it is a great deal to perform effective analysis due to the following

DOI: 10.4018/IJSIR.2022010107

challenges. Firstly, recent software systems regularly produce tons of logs (e.g., a commercial cloud device can generate about gigabytes of data every hour) (Astekin et al., 2019). These high volume logs make it impossible to do a manual inspection for key diagnostics even if provided with search and grep utilities. As a result, traditional log analysis methods that mostly depend on manual operation have become unfeasible and prohibitive (Jia et al., 2018; Li et al., 2018). Secondly, the messages in log files are intrinsically unstructured as developers normally store the system activities in a free-text format for better accessibility and flexibility (Rath, 2016). Therefore, there exists a great demand for automated log analysis for all kinds of applications (He et al., 2017; El-Masri et al., 2020). An automatic log analysis based on keyword searches with ad hoc scripts like "CRITICAL" or "ERROR" is found to be inadequate for fixing several problems (Baudart, 2018; Vega et al., 2017). Further, rule-based methods are an advanced technique; however it is difficult to formulate each and every rule throughout the analysis (Khan & Parkinson, 2018). The drawbacks of these initial approaches significantly increased the difficulty in log file data analysis. To overcome these limitations, recent studies as well as industrialists provide different alternatives with a powerful word search and machine learning analytics like Splunk (Carasso, 2012), ELK (Smith, 2015), Logentries (Jaunin & Burdick, 2011), etc. Nevertheless, the first and foremost step to enable these log analysis is log parsing through which free-text raw log data are parsed into a stream of structured data (He et al., 2016).

A log file usually contains a batch of single or multi-lined characters listed in the form of an inherent chronological pattern. This pattern is normally underpinned by a timestamp inserted to each message of the log files. The messages might be highly structured (e.g. values separated with commas), semi-structured (e.g. attribute-value format), unstructured (e.g. free-text of random length) or a combination of these (Landauer et al., 2020). Therefore, log parsers are applied to transfer the log messages in unstructured pattern into a structured pattern. The parsed messages then enable the advantage of certain characteristics like filtering, effective search, counting, grouping and sophisticated log mining (Zhu et al., 2019). An example of unstructured HDFS benchmark log message is shown in Figure 1. It consists of the following fields recorded during system runtime: (a) timestamp (stores the time in which an event is occurred), (b) verbosity level (describes the severity level of occurring event, e.g. "FATAL", "ERROR", "WARN", "INFO") and (c) raw message data (free-text depiction of system task). Generally, log parsing with traditional methods require heavy usage of regular expressions in the source code. These expressions are manually derived by the developers for each log event and added to the parsing rule list. During system runtime storage, the incoming log messages are compared with the derived expressions to find a matching log event from the parsing list. Besides, traditional methods are found to be impractical and error-prone due to the following limitations. First, todays systems are large-scale which in turn produce huge volume of rapidly increasing logs. Second, the design as well as computing task of the modern systems is very complex which makes logging difficult. Third, the source code of a system can be written by any developer who has even no idea about the purpose of log parsing. Finally, the logging statements are updated frequently due to the wide adoption of agile software and so, developers must update the parsing rules to guarantee the new version (He et al., 2018).

The lack of structured traditional log data format leads to the development of automated log data template detection. It is the process of identifying different events contained in log files through automated template extraction. It is also the modern way of converting unstructured log data into structured form for a high-scale platform. Many automated log parses are presented in the literature namely SLCT (Vaarandi, 2008) and its extension Logcluster (Vaarandi & Pihelgas, 2015), IPLoM

Figure 1. Format of an unstructured log message

```
081110 103812 9115 INFO dfs.DataNode$PacketResponder: Received block
blk_-5668584889663038916 of size 67108864 from /10.250.9.207
```