

IJIRE-0000827

by FDRP Journal's

Submission date: 04-Oct-2024 01:15PM (UTC+0700)

Submission ID: 2474628665

File name: IJIRE-0000827.docx (620.9K)

Word count: 2198

Character count: 12985

Efficient Log Parsing and Visualization System for Large-Scale Logs: A Data-Centric Approach Using Custom Data Structures

V M Suhas¹, V M Subash², V S Arjun³

^{1,2} Final Year BTech Students Dept. of Computer Science and Engineering, Reva University, Bangalore, Karnataka, India

³ Technical Program Manager – Wireless, Tessolve Semiconductor Private Limited, Bangalore, Karnataka, India

Abstract: In the realm of debugging and system optimization, testers frequently deal with large-scale logs from systems such as 5G communication networks. Handling, filtering, and visualizing these logs efficiently are crucial for understanding system performance and identifying issues. This paper presents an efficient log parsing and visualization system that leverages custom data structures to handle logs more effectively, with significant improvements in filtering and querying after initial parsing. The system provides multiple ways to filter logs, including by parent attributes, supports inline scripting to manipulate data tables, and offers various visualization techniques. The tool is optimized for fast filtering and querying, compared to other approaches, and will be available as an open-source project to the broader testing and debugging community.

Key Word: PySide6, Matplotlib, MplCursors, Log Parsing, QT Threads.

I. INTRODUCTION

Modern software systems, especially those used in 5G communication networks and large-scale infrastructures, generate vast amounts of log data. These logs are essential for understanding system behaviour, diagnosing issues, and optimizing performance. However, manually analysing such large and complex logs is time-consuming and prone to errors, making automated log analysis tools a necessity.

Traditional log analysis tools struggle to handle the massive data volumes and intricate log structures found in modern systems. This is especially true for testers who need to filter, visualize, and analyse logs efficiently. In response to these challenges, this paper introduces a log parsing and visualization tool designed to help testers debug large-scale logs more effectively.

The tool offers a flexible approach, allowing users to apply custom filters, visualize data through multiple plotting methods, and export logs for further analysis. It also enables users to write and execute their own Python scripts directly within the tool, making it ideal for testers who require custom log manipulation.

This paper outlines the tool's design, key features, and performance, demonstrating its usefulness in simplifying and accelerating log analysis tasks in complex environments like 5G communication systems.

II. LITERATURE SURVEY

1. Tools and Benchmarks for Automated Log Parsing:

This paper, presented at ICSE'19, discusses the performance modeling and failure diagnosis aspects of log parsing. It highlights the use of logs as a valuable data source for performance modeling by extracting all possible event templates from logs for model construction. The paper addresses the challenges of manual failure diagnosis due to the large volume and verbosity of logs, proposing machine learning techniques to automate root cause analysis.

2. System Log Parsing: A Survey:

This survey from ar5iv.org provides a comprehensive overview of system log parsing, introducing basic terminologies and processes involved in log parsing. It explores the various phases of log parsing including preprocessing, data classification, and template extraction, and discusses the challenges of adapting log parsing techniques to diverse log formats and the need for preprocessing to manage the complexity of logs.

3. Automatic Parsing and Utilization of System Log Features in Log Analysis: A Survey:

Published by MDPI in 2023, this paper surveys the field of log parsing and feature extraction. It emphasizes the importance of system logs in anomaly analysis, intrusion detection, and situational awareness. The survey critiques the unstructured nature of system logs and the manual effort required to analyze these logs, suggesting the need for more advanced parsing techniques.

4. Log Parsing Evaluation in the Era of Modern Software Systems:

This paper critiques current evaluation metrics used in log parsing, arguing that they fail to accurately assess the quality of log parsing outputs. It advocates for new metrics such as log template accuracy and edit-distance to better reflect the real-world effectiveness of log parsers. The paper is a call to action to refine evaluation methods to better meet the needs of practical applications in modern software systems.

III. EXISTINGSYSTEM

In the domain of log analysis, traditional systems often rely on static methods to parse and analyze logs. These systems, though functional, struggle with large-scale data generated by modern networks like 5G communication systems. Current tools primarily allow for basic filtering and querying but often lack efficient visualization techniques and advanced data manipulation. While some existing log analysis platforms provide visualizations, they tend to be rigid, with limited support for customizable filtering or querying of complex hierarchical log structures. Moreover, these systems often do not cater well to testers who need to write custom scripts to automate their debugging workflows.

IV. PROPOSEDSYSTEM

The proposed system is an advanced log parsing, visualization, and analysis tool developed using Python, custom data structures, and integrated plotting libraries. Designed specifically for handling large volumes of logs generated in environments such as 5G communication networks, the tool supports rapid querying, filtering, and visualization of logs. It leverages custom-built structures (like MYDS) that efficiently store and index log data, ensuring that users can perform complex queries in minimal time after the initial parsing.

This system introduces several novel features. It allows users to filter logs based on parent-child relationships within log data, provides multiple visualization options, and enables the user to export the logs for external analysis. Furthermore, the tool includes an embedded script execution feature, allowing testers to write and run Python scripts directly within the platform, enhancing their ability to customize the analysis. Additionally, users can save visualizations and retrieve previously saved log views, offering flexibility and convenience for continuous log analysis.

V. ARCHITECTUREDIAGRAM

The process starts with log file acquisition, followed by an efficient parsing step using the MYDS class, which builds a hierarchical structure of the log data. The system then processes these logs, identifying relationships between parent and child logs, attributes, and relevant filters. The next stage allows the user to apply custom filters, after which the data is either visualized using Matplotlib or saved for future reference. Through the interface, users can also write scripts to manipulate the parsed data. Finally, the results can be exported, and visualizations can be saved or shared for collaboration.



VI. SYSTEM OVERVIEW

A. Log File Acquisition and Parsing

The Log File Acquisition and Parsing module is responsible for handling large-scale logs (such as those generated in 5G communication networks). The system leverages custom data structures, specifically **MYDS** (custom hierarchical data structure), **MydsJ** (JSON-based data structure), and **MyDsp** (pandas-based data structure), to efficiently parse and organize the log data. Using MYDS, the system captures and stores a comprehensive hierarchical structure, including parent-child relationships, and indexes all key attributes from the logs for efficient querying. Initial parsing times are slightly higher due to this in-depth data collection, but it ensures faster subsequent querying. For MyDsJ and MyDsp, simpler indexing is applied, leading to faster parsing times but at the cost of losing some parent-child relationship data.

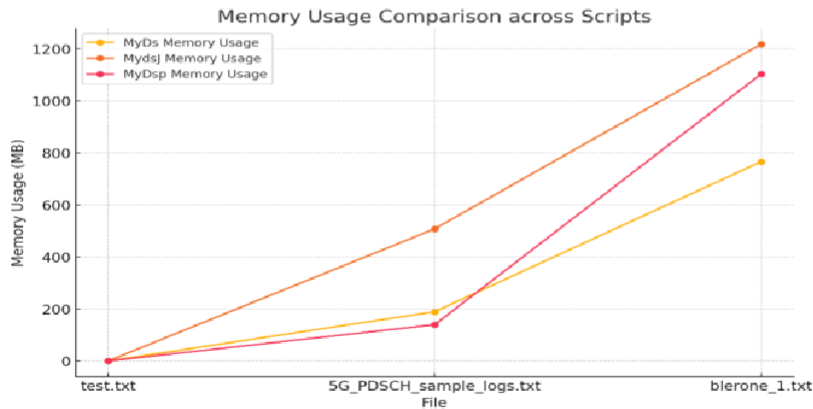
B. Data Structuring and Indexing

The Data Structuring and Indexing module focuses on organizing the parsed log data into efficient data structures for further querying and filtering.

- **MYDS**: Provides extensive data indexing and maintains parent-child relationships, allowing detailed queries across the log structure. This makes MYDS ideal for complex filtering and querying tasks, albeit with a higher memory footprint.
- **MydsJ**: Uses a JSON-based approach for data storage. While it offers faster parsing times due to its lightweight structure, some detailed data (such as parent-child relationships) is lost in the process, making it better suited for simpler queries.
- **MyDsp**: Uses the **pandas** library for data storage and manipulation, offering fast data filtering and manipulation at the expense of higher memory usage. While pandas provides efficient data querying, it also sacrifices some of the detailed indexing found in MYDS.

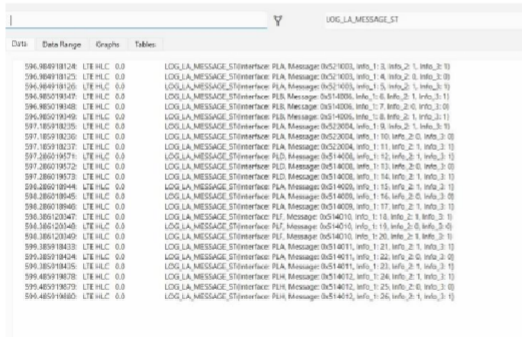
Performance Comparison for Parsing:

- **MyDs**: 481.469 MB log parsed in 51.962 seconds with 765.828 MB memory usage.
- **MydsJ** (JSON): 481.469 MB log parsed in 43.123 seconds with 1217.332 MB memory usage.
- **MyDsp** (pandas): 481.469 MB log parsed in 78.516 seconds with 1102.516 MB memory usage.



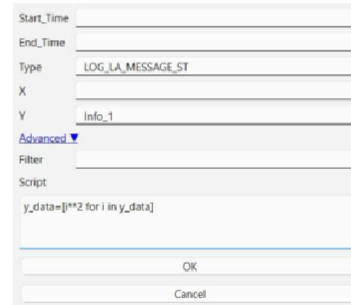
C. Filtering and Querying

The Filtering and Querying module allows users to apply custom filters on the logs. The system supports filtering based on individual attributes or parent attributes, enabling multi-level filtering with MYDS. Filters can target specific log entries and their relationships with other logs, such as filtering by timestamp, message type, or parent attributes. MYDS performs particularly well in this regard due to its comprehensive indexing. On the other hand, MydsJ (JSON) and MyDsp (pandas) are better suited for simpler queries, as they lack some of the detailed indexing and parent-child relations maintained by MYDS.

The interface shows a table with columns: Dots, Data Range, Graphs, Tables. The 'Tables' column is active, displaying a list of log messages. Each row contains a timestamp, a log level (LTL, LTL, LTL), and a log message. The messages are all of type 'LOG_MESSAGE_Statement' and contain various parameters like 'PLA', 'Message', and 'Info'.

Dots	Data Range	Graphs	Tables
596.988918124	LTL LTL LTL	0.0	LOG_MESSAGE_Statement: PLA Message: [00000000, info, 1, 0, info, 2, 0, info, 3, 0]
596.988918125	LTL LTL LTL	0.0	LOG_MESSAGE_Statement: PLA Message: [00000000, info, 1, 0, info, 2, 0, info, 3, 0]
596.988918126	LTL LTL LTL	0.0	LOG_MESSAGE_Statement: PLA Message: [00000000, info, 1, 0, info, 2, 0, info, 3, 0]
596.988918127	LTL LTL LTL	0.0	LOG_MESSAGE_Statement: PLA Message: [00000000, info, 1, 0, info, 2, 0, info, 3, 0]
596.988918128	LTL LTL LTL	0.0	LOG_MESSAGE_Statement: PLA Message: [00000000, info, 1, 0, info, 2, 0, info, 3, 0]
596.988918129	LTL LTL LTL	0.0	LOG_MESSAGE_Statement: PLA Message: [00000000, info, 1, 0, info, 2, 0, info, 3, 0]
596.988918130	LTL LTL LTL	0.0	LOG_MESSAGE_Statement: PLA Message: [00000000, info, 1, 0, info, 2, 0, info, 3, 0]
596.988918131	LTL LTL LTL	0.0	LOG_MESSAGE_Statement: PLA Message: [00000000, info, 1, 0, info, 2, 0, info, 3, 0]
596.988918132	LTL LTL LTL	0.0	LOG_MESSAGE_Statement: PLA Message: [00000000, info, 1, 0, info, 2, 0, info, 3, 0]
596.988918133	LTL LTL LTL	0.0	LOG_MESSAGE_Statement: PLA Message: [00000000, info, 1, 0, info, 2, 0, info, 3, 0]
596.988918134	LTL LTL LTL	0.0	LOG_MESSAGE_Statement: PLA Message: [00000000, info, 1, 0, info, 2, 0, info, 3, 0]
596.988918135	LTL LTL LTL	0.0	LOG_MESSAGE_Statement: PLA Message: [00000000, info, 1, 0, info, 2, 0, info, 3, 0]
596.988918136	LTL LTL LTL	0.0	LOG_MESSAGE_Statement: PLA Message: [00000000, info, 1, 0, info, 2, 0, info, 3, 0]
596.988918137	LTL LTL LTL	0.0	LOG_MESSAGE_Statement: PLA Message: [00000000, info, 1, 0, info, 2, 0, info, 3, 0]
596.988918138	LTL LTL LTL	0.0	LOG_MESSAGE_Statement: PLA Message: [00000000, info, 1, 0, info, 2, 0, info, 3, 0]
596.988918139	LTL LTL LTL	0.0	LOG_MESSAGE_Statement: PLA Message: [00000000, info, 1, 0, info, 2, 0, info, 3, 0]
596.988918140	LTL LTL LTL	0.0	LOG_MESSAGE_Statement: PLA Message: [00000000, info, 1, 0, info, 2, 0, info, 3, 0]
596.988918141	LTL LTL LTL	0.0	LOG_MESSAGE_Statement: PLA Message: [00000000, info, 1, 0, info, 2, 0, info, 3, 0]
596.988918142	LTL LTL LTL	0.0	LOG_MESSAGE_Statement: PLA Message: [00000000, info, 1, 0, info, 2, 0, info, 3, 0]
596.988918143	LTL LTL LTL	0.0	LOG_MESSAGE_Statement: PLA Message: [00000000, info, 1, 0, info, 2, 0, info, 3, 0]
596.988918144	LTL LTL LTL	0.0	LOG_MESSAGE_Statement: PLA Message: [00000000, info, 1, 0, info, 2, 0, info, 3, 0]
596.988918145	LTL LTL LTL	0.0	LOG_MESSAGE_Statement: PLA Message: [00000000, info, 1, 0, info, 2, 0, info, 3, 0]
596.988918146	LTL LTL LTL	0.0	LOG_MESSAGE_Statement: PLA Message: [00000000, info, 1, 0, info, 2, 0, info, 3, 0]
596.988918147	LTL LTL LTL	0.0	LOG_MESSAGE_Statement: PLA Message: [00000000, info, 1, 0, info, 2, 0, info, 3, 0]
596.988918148	LTL LTL LTL	0.0	LOG_MESSAGE_Statement: PLA Message: [00000000, info, 1, 0, info, 2, 0, info, 3, 0]
596.988918149	LTL LTL LTL	0.0	LOG_MESSAGE_Statement: PLA Message: [00000000, info, 1, 0, info, 2, 0, info, 3, 0]
596.988918150	LTL LTL LTL	0.0	LOG_MESSAGE_Statement: PLA Message: [00000000, info, 1, 0, info, 2, 0, info, 3, 0]
596.988918151	LTL LTL LTL	0.0	LOG_MESSAGE_Statement: PLA Message: [00000000, info, 1, 0, info, 2, 0, info, 3, 0]
596.988918152	LTL LTL LTL	0.0	LOG_MESSAGE_Statement: PLA Message: [00000000, info, 1, 0, info, 2, 0, info, 3, 0]
596.988918153	LTL LTL LTL	0.0	LOG_MESSAGE_Statement: PLA Message: [00000000, info, 1, 0, info, 2, 0, info, 3, 0]
596.988918154	LTL LTL LTL	0.0	LOG_MESSAGE_Statement: PLA Message: [00000000, info, 1, 0, info, 2, 0, info, 3, 0]
596.988918155	LTL LTL LTL	0.0	LOG_MESSAGE_Statement: PLA Message: [00000000, info, 1, 0, info, 2, 0, info, 3, 0]
596.988918156	LTL LTL LTL	0.0	LOG_MESSAGE_Statement: PLA Message: [00000000, info, 1, 0, info, 2, 0, info, 3, 0]
596.988918157	LTL LTL LTL	0.0	LOG_MESSAGE_Statement: PLA Message: [00000000, info, 1, 0, info, 2, 0, info, 3, 0]
596.988918158	LTL LTL LTL	0.0	LOG_MESSAGE_Statement: PLA Message: [00000000, info, 1, 0, info, 2, 0, info, 3, 0]
596.988918159	LTL LTL LTL	0.0	LOG_MESSAGE_Statement: PLA Message: [00000000, info, 1, 0, info, 2, 0, info, 3, 0]
596.988918160	LTL LTL LTL	0.0	LOG_MESSAGE_Statement: PLA Message: [00000000, info, 1, 0, info, 2, 0, info, 3, 0]

Figure C.1 Data view Interface

The interface for setting advanced filters. It includes fields for Start_Time, End_Time, Type (set to LOG_MESSAGE_ST), X, Y (set to Info_1), and a Filter section. The Filter section has a dropdown for Filter and a text area for Script with the content 'y_data=[i**2 for i in y_data]'. There are OK and Cancel buttons at the bottom.

Start_Time
End_Time
Type: LOG_MESSAGE_ST
X
Y: Info_1
Advanced
Filter
Script: y_data=[i**2 for i in y_data]
OK
Cancel

Figure C.2 Interface to set advanced filters

Figure C.1 shows the interface of the application after a log file has been successfully loaded, and a filter has been applied, allowing users to view the filtered results and interact with the data.

Figure C.2 shows the interface used to apply complex filters and functions. This interface enables users to set advanced parameters for querying the logs, including custom filters, parent-attribute filtering, and the ability to apply specific functions for enhanced log analysis and visualization.

D. Visualization and Plotting

The Visualization and Plotting module provides various ways to visualize log data using Matplotlib and Seaborn. The system allows users to create scatter plots, time series plots, heatmaps, and more. Logs can be filtered and visualized based on individual attributes or parent attributes. This flexibility in visualization provides users with valuable insights into the logs, helping testers understand patterns or anomalies across time and log types.

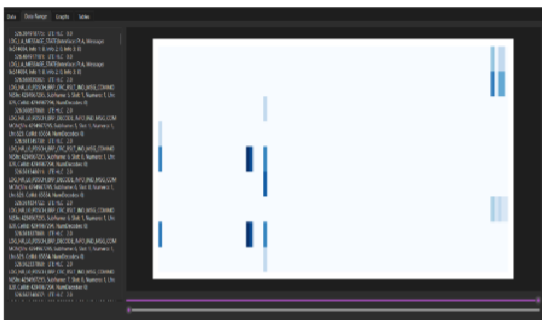


Figure D.1 Heatmap Interface

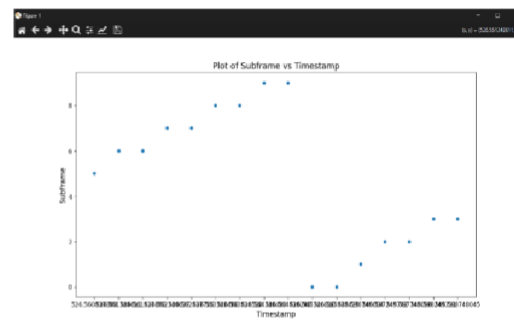


Figure D.2 Sample Scatter plot

Figure D.1 shows the heatmap interface of the application, where the user can visualize log data as a heatmap. The heatmap provides a graphical representation of log activity over time, with colors indicating the density of log events.

Figure D.2 shows a sample scatter plot of the log data generated by the application. This scatter plot allows users to visualize the relationship between different attributes in the log data, with each point representing a specific log event, and the axes representing the selected parameters for analysis.

E. Scripting and Data Export

The Scripting and Data Export module empowers users to write custom Python scripts within the app for manipulating log data, performing advanced queries, or generating additional insights. This feature enables the seamless execution of custom Python scripts to modify log data and generate reports. Additionally, users can export logs or filtered results for further analysis outside the app. The system also supports saving graphs and exporting them in various formats for inclusion in reports or further analysis.



	Subsystem	Size
1	5	8
2	5	0
3	5	1
4	7	0
5	7	1
6	9	0
7	9	1
8	9	0
9	9	1
10	9	0
11	9	1
12	1	0
13	1	1
14	2	0
15	2	1
16	9	0
17	3	1
18	4	0
19	4	1

Figure E.1 Table Interface

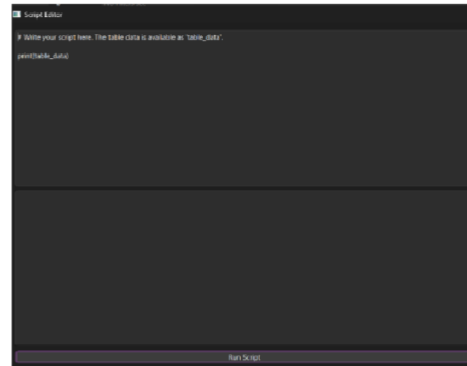


Figure E.2 Script Interface

Figure E.1 shows the interface of the application after log data has been loaded into the table. The data from the log file is parsed and displayed in a structured format. Users can manipulate this data further by applying filters or executing custom queries. Additionally, the data can be exported for use in external applications if needed.

Figure E.2 shows the Script Interface, where users can write and execute custom Python scripts to manipulate the data loaded in the table. This feature allows for dynamic data manipulation, custom filtering, and plotting directly from the table, offering flexibility to testers who wish to implement their own data analysis logic.

VII. FUTURE ENHANCEMENT

Future updates aim to:

1. Add support for additional graph types (e.g., pie charts, bar graphs).
2. Improve data storage by transitioning from dictionaries to more efficient data structures, potentially reducing memory usage and improving performance.
3. Enhance support for more advanced querying and filtering capabilities.

VIII. CONCLUSION

In conclusion, The developed log parsing and visualization tool offers a powerful solution for testers working with large-scale logs, including those from 5G systems. By using custom data structures like MYDS for detailed indexing, and alternatives like MydsJ (JSON-based) and MyDsp (pandas-based), the system achieves a balance between performance and data integrity. MYDS provides deep indexing and retains parent-child relationships for advanced filtering and querying, while MydsJ and MyDsp offer faster, lighter alternatives for simpler tasks. Despite its longer initial parsing time, MYDS excels in post-parsing speed, making it highly efficient for complex queries. The tool supports multiple visualizations, allows custom scripting, and enables data export, giving users flexibility in log analysis. Its open-source nature invites future enhancements, ensuring it remains a valuable resource for log debugging. Performance metrics, visualizations, and source code will be provided to demonstrate its effectiveness.

References

1. Tools and Benchmarks for Automated Log Parsing: Jieming Zhu, Shilin He, Jinyang Liu, Pinjia He, Qi Xie, Zibin Zheng, Michael R. Lyu (2019)"Tools and Benchmarks for Automated Log Parsing" <https://arxiv.org/abs/1811.03509>
2. Automatic Parsing and Utilization of System Log Features in Log Analysis A Survey: Junchen Ma, Yang Liu, Hongjie Wan, Guozi Sun, (2023)"Automatic Parsing and Utilization of System Log Features in Log Analysis: A Survey", DOI: <https://ieeexplore.ieee.org/document/10301279>
3. System Log Parsing: A Survey: Tianzhu Zhang, Han Qiu, Gabriele Castellano, Myriana Rifai, Chung Shue Chen, Fabio Pianese "System Log Parsing: A Survey" <https://ieeexplore.ieee.org/document/10025560>
4. Log Parsing Evaluation in the Era of Modern Software Systems: Stefan Petrescu, Floris den Hengst, Alexandru Uta, Jan S. Rellermeyer "Log Parsing Evaluation in the Era of Modern Software Systems" <https://ieeexplore.ieee.org/document/10301279>
5. Source code for the project : - <https://github.com/VMsubhash/LogPlot>

ORIGINALITY REPORT

4%

SIMILARITY INDEX

3%

INTERNET SOURCES

4%

PUBLICATIONS

1%

STUDENT PAPERS

PRIMARY SOURCES

1

arxiv.org

Internet Source

1%

2

Jieming Zhu, Shilin He, Jinyang Liu, Pinjia He, Qi Xie, Zibin Zheng, Michael R. Lyu. "Tools and Benchmarks for Automated Log Parsing", 2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP), 2019

Publication

1%

3

www.mdpi.com

Internet Source

1%

4

link.springer.com

Internet Source

<1%

5

Marcin Borowiec, Tomasz Rak. "Advanced Examination of User Behavior Recognition via Log Dataset Analysis of Web Applications Using Data Mining Techniques", Electronics, 2023

Publication

<1%

6

"Advanced Parallel Processing Technologies",
Springer Science and Business Media LLC,
2019

Publication

<1 %

7

louisdl.louislibraries.org

Internet Source

<1 %

8

web.archive.org

Internet Source

<1 %

9

Junchen Ma, Yang Liu, Hongjie Wan, Guozi
Sun. "Automatic Parsing and Utilization of
System Log Features in Log Analysis: A
Survey", Applied Sciences, 2023

Publication

<1 %

Exclude quotes On

Exclude matches Off

Exclude bibliography On

