

Log Analyzer

SANGEETA SINHA

Barclays

Abstract- Log Analyzer utility is a python utility which provides web-service response time metrics for all the transactions processed by the application in specified duration. It parses the application logs to identify the response time logs, processes all the data from all the servers and provides minimum, average, maximum, percentile and outliers details broken down at transaction type as well as at server level. This paper is created to get familiar with Log Analyzer Utility & understand the usage so that if any other application requires similar solution, the utility can be customized for other applications & be utilized.

Index Terms - Python, Web Service, Response Time, Load Balancer, Load Runner, Load Test, Load Generator, Batch Server, Percentile Response Time.

I. BACKGROUND OF STUDY

In today's distributed computing environments, enterprise applications are often hosted across multiple servers, platforms, and cloud-based systems. These environments generate vast amounts of log data—structured, semi-structured, or unstructured—that are essential for monitoring system health, debugging issues, performing audits, and ensuring cyber security compliance [1]. However, manual inspection of such distributed logs is not only time-consuming but also error-prone, especially when logs come from heterogeneous sources such as web servers, application servers, databases, and operating systems.

Log data contains valuable insights about user behaviour, system performance, and error trends. The challenge lies in collecting, parsing, correlating, and analysing these logs in real-time or near-real-time to extract actionable intelligence. Traditional log monitoring tools often lack scalability or the ability to correlate logs from multiple machines effectively. Furthermore, the sheer volume of data can

overwhelm system administrators or developers who need to trace issues quickly across distributed infrastructures [2].

A centralized log analyser offers a solution by automating the log collection from different servers, normalizing formats, and applying rule-based or AI-powered analysis to detect anomalies or generate alerts. These systems can streamline operational workflows, reduce downtime, and improve overall observability. With the growing demand for DevOps and continuous monitoring, log analysis has become a foundational component of modern system management and incident response [3].

This study presents the design and implementation of a log analyser that can collect logs from multiple servers, aggregate them in a central location, and provide insights through a user-friendly interface. The system focuses on scalability, simplicity, and extensibility to accommodate various server configurations and log types [4].

II. SIGNIFICANCE OF THE STUDY

In distributed computing environments, log files serve as critical components for system monitoring, troubleshooting, and performance evaluation. As modern systems grow increasingly complex, managing and analysing logs across multiple servers becomes a challenging task. Manual log inspection is not only inefficient but often fails to detect patterns and anomalies in real-time [5].

This study is significant because it introduces a centralized log analyser capable of collecting, aggregating, and analysing logs from various servers in an automated and scalable manner. Unlike traditional tools that may be resource-intensive, expensive, or complex to configure, the proposed solution is lightweight, extensible, and designed to be easily deployed across diverse environments.

III. STATEMENT OF THE PROBLEM

In modern software development, performance testing is a critical step to ensure that applications meet scalability and reliability requirements. One of the primary sources of insights during performance testing is log data, which captures real-time system behavior, error occurrences, resource utilization, and response times. However, due to the massive volume and complexity of logs generated during tests, manual analysis is impractical and prone to human error. This log analyzer is a tool designed to process and interpret log files efficiently. Manually inspecting logs across multiple servers is the time-consuming and inefficient. To address this challenge, we have developed a Python-based log analyzer utility, designed to automate log processing across multiple AIS application servers.

This utility is executed remotely using a shell script, allowing it to be triggered seamlessly from a centralized batch server. The shell script automates the execution of the Python analyzer across distributed environments, enabling real-time log collection, parsing, and analysis. By automating log analysis using Python utility techniques, this research aims to enhance the efficiency and accuracy of performance testing, reducing debugging time and improving system reliability.

IV. PURPOSE OF THE STUDY

The primary purpose of this study is to design and develop a centralized log analyser capable of collecting, processing, and analysing final logs from multiple servers in a distributed system. The study aims to address the challenges associated with manual log inspection, such as inefficiency, delayed troubleshooting, and difficulty in identifying anomalies across different system components.

V. WHY THIS UTILITY WAS NEEDED?

- Absence of any external app monitoring tool- FPP application is C++ based application with Java/Scala libraries packaged & provided by Actimize. Due to its custom libraries nature, there are no standard monitoring tools installed which can help with identifying the true web-service

response time from application logs. Due to this constraint, we were completely dependent on Load Runner to provide response time of the processed transactions for FPP initiated traffic.

- Unable to run the Load Tests from all LG's – Response Time provided by Load Runner includes the additional latency from Load Generator to application Load Balancer which is ~500ms & ~300ms for some listed LGs and which was providing unrealistic results.
- Unable to extract application performance analysis for End to End Load Tests – Further, Load Runner could provide the stats FPP initiated traffic & we were additionally dependent on upstream database analysis for analyzing FPP application performance during End to End test even though same information was getting recorded in FPP application logs [6] .
- Only a sample of the application logs could be analyzed manually - Manual analysis on the application logs could be performed only on a sample of the logs due to multiple servers & huge file size. Application logs also occupied a considerable amount of disk space which require time to time cleanup and once the logs removed, there was no scope to get the required metrics again in future.

Therefore to address all the above problem statements, Log Analyzer utility was created in python after which all of above constrains are answered & now there is no manual effort involved in identifying response time metrics at application server level with additional analysis.

VI. HOW THIS UTILITY HAS HELPED?

- Provides the complete analysis of application response time logs on all the 7 application servers & provides consolidate output in less than 5 seconds vs. 3-4 hours of manual efforts.
- This has helped in completing the overall Load Tests analysis without any manual efforts in data extraction after which overall analysis is completed in less than 2 hours.
- This has helped in running the FPP load tests irrespective of the Load Generator location & now any LG can be used to run the load test while

performance analysis can be done from application logs.

- This has helped in identifying any performance bottlenecks being caused at server level due to environmental issues & reporting it to the team accordingly.
- This has also helped in identifying the FPP application performance during E2E load tests without any dependency on upstream component analysis.
- This has also helped in keeping the data completely customized without any additional overhead on application servers so that it can be run any no of times without existing application.

VII. OLD APPROACH

Earlier approach was to login & analyze the application logs manually on each server, extract the logs for required duration & collate it in excel to perform final analysis. In absence of any automation utility, it was laborious work to analyze each and every file manually for all the 7 application servers. Also note, each processed transaction writes 1 line for each transaction in the line & during load test, each server processed at least 50000 transaction which made the manual log analysis even harder. This is represented in Fig. 1.

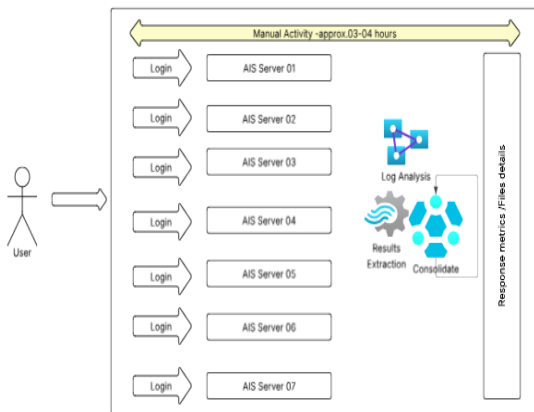


Fig. 1. Old Approach

VIII. NEW APPROACH

Now with the help of automated utility, response time analysis is just a click away. It takes test duration as the input and after performing complete logs

analysis, provides the collated result from all the 7 servers in customized format which is then used to generate a graphical representation to be provided to stakeholders. This is represented in Fig. 2.

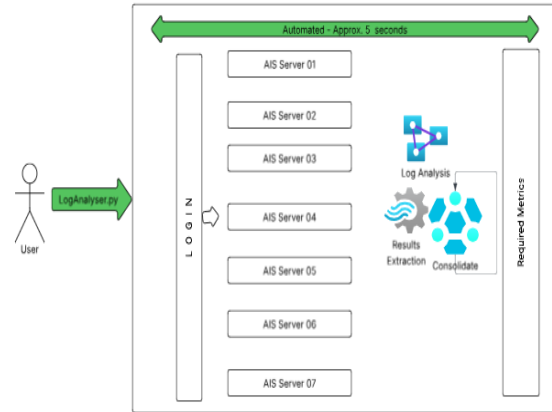


Fig. 2. New Approach

IX. TOOL DETAILS

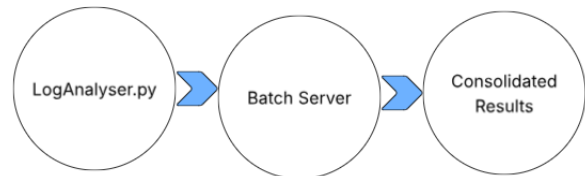


Fig. 3. Tool Details

X. UTILITY IMPLEMENTATION & PRE-REQUISITE

Complete utility code is written in Python 2.0 (Based on the installed python version on Linux Servers). Additional Wrapper is written in shell Script to run the script remotely on all the AIS application servers from AIS Batch server which has SSH trust established enabling the password less execution. Execution Logic & Log Files parsing logic is coded on the log file format which can be customized along with results output.

XI. UTILITY RESULTS

Output	Description
Server Name	Server Name from which application logs are analysed
Transaction Type	Transaction Type based on the transaction details which are getting logged in the log file.
Minimum	Minimum response time (in ms) of all the processed transactions from the analysed server
Average	Average response time (in ms) of all the processed transactions from the analysed server
Maximum	Maximum response time (in ms) of all the processed transactions from the analysed server
50th Percentile	50th percentile response time (in ms) of all the processed transactions from the analysed server
80th Percentile	80th percentile response time (in ms) of all the processed transactions from the analysed server
90th Percentile	90th percentile response time (in ms) of all the processed transactions from the analysed server
95th Percentile	95th percentile response time (in ms) of all the processed transactions from the analysed server
99th Percentile	99th percentile response time (in ms) of all the processed transactions from the analysed server
100th Percentile	100th percentile response time (in ms) of all the processed transactions from the analysed server
1s Outliers	No of Transactions exceeding 1s Response Time of all the processed transactions from the analysed server
2s Outliers	No of Transactions exceeding 2s Response Time of all the processed transactions from the analysed server
3s Outliers	No of Transactions exceeding 3s Response Time of all the processed transactions from the analysed server
4s Outliers	No of Transactions exceeding 4s Response Time of all the processed transactions from the analysed server
5s Outliers	No of Transactions exceeding 5s Response

Outliers	Time of all the processed transactions from the analysed server
Total Transactions	Total Transactions processed/observed in logs from the analysed sever

Fig. 4. Utility Results

XII. WORKFLOW

- Login into the server – Batch Server
- Run the Log analyzer utility – “./LogAnalyzer “Start time” “End time” – Time format (YYYY- MMM-DD hh:mm)

```

gbrdr000003558:/home/sysBIFM/NFT$ ./LogAnalyzer
-----
./LogAnalyzer "<Start_Time>" "<End_Time>"
-----
./LogAnalyzer "2019-Sep-01 08:40" "2019-Sep-05 09:49"
-----
Note: During Day Light Saving enabled, run the script with
1 hour prior time due to known issue around MMR timestamp.
-----
Start,Transaction Type,Min,Avg,Max,50th,60th,70th,80th,90th,95th,99th,100th,1s Outliers,2s Outliers,3s Outliers,4s Outliers,5s Outliers>Total
-----
gbrdr000001193,MMT,64,665,11172,715,1276,894,1703,062,2176,3394,3596,55929,11172,8280,1635,463,170,66,25485
gbrdr000001193,MMT,74,467,13923,310,493,791,1429,094,2843,3004,15825,804,232,79,35,30,10371
gbrdr000001194,MMT,66,709,7266,606,0276,1081,1316,1345,6502,1411,3094,2303,30388,7266,6130,599,76,17,6,25362
gbrdr000001194,MMT,76,474,15491,307,483,6658,796,2319,1529,3388,2793,77192,11491,624,285,37,76,66,10469
gbrdr000001195,MMT,66,866,7902,761,1291,2834,1648,9394,2057,4452,3212,3194,7902,6677,1363,322,86,28,25189
gbrdr000001195,MMT,55,607,15085,326,570,052,870,4934,1549,7463,3122,37818,15085,920,303,112,45,70,10455
gbrdr000001196,MMT,64,775,7320,661,849,1164,8076,1491,831,1843,24315,2877,77162,7320,7093,946,210,42,17,28328
gbrdr000001196,MMT,73,610,15874,326,577,2339,889,8222,1841,916,9126,79612,15874,917,306,117,84,69,10585
gbrdr000001197,MMT,61,731,9134,627,0615,1119,0116,1450,5268,1695,84395,2440,70797,9134,6677,437,102,27,8,25400
gbrdr000001197,MMT,68,495,15643,313,304,7852,829,5174,1520,524,3053,14340,15643,946,284,108,45,72,10457
gbrdr000001198,MMT,63,697,11439,675,856,1040,1838,1279,6248,1524,06985,2462,29375,11439,5637,470,189,42,27,25310
gbrdr000001198,MMT,77,490,15549,301,275,484,6962,800,14832,1466,54155,3582,77439,15549,839,252,125,101,83,10540
    
```

Fig. 5. Log Analyzer Utility

- Copy the output data and paste it in the customized Log Analyzer.xlsx spreadsheet which will automatically consolidate and generate the graphical representation of the data.

Customized Output:

BNET Transactions(Ref_MMR_access.log)															
Server Name	M in	A vg	Ma x	50 th	80 th	90 th	95 th	99t h	100 th	1st Outliers	2s Outliers	3s Outliers	4s Outliers	5th Outliers	Total Transactions
gbrdsr000001193	51	295	2014	224	432	559	711	1111	2014	102	1	0	0	0	7828
gbrdsr000001194	58	325	1901	240	478	629	799	1267	1901	173	0	0	0	0	7918
gbrdsr000001195	56	318	2271	239	469	616	770	1212	2271	161	1	0	0	0	7975
gbrdsr000001196	55	328	2183	247	480	626	805	1263	2183	181	2	0	0	0	7906
gbrdsr000001197	51	318	2019	241	469	609	769	1166	2019	123	1	0	0	0	7862
gbrdsr000001198	53	284	1869	214	418	540	689	981	1869	72	0	0	0	0	7892
gbrdsr000001883	53	319	2557	240	458	618	798	1199	2557	168	3	0	0	0	7900
Total	511	3157	25057	2400	476	627	803	1267	2557	980	8	0	0	0	55281

Fig. 6. Customized Output

Server Name	M in	A vg	Ma x	50 th	80t h	90t h	95t h	99t h	100 th	1st Outliers	2s Outliers	3s Outliers	4s Outliers	5th Outliers	Total Transactions
gbrdsr000001193	52	629	5743	539	849	1077	1333	2137	5743	1805	187	33	5	2	14427
gbrdsr000001194	60	880	8081	651	1263	1742	2236	3594	8081	4117	1000	308	85	39	14353
gbrdsr000001195	58	815	9011	606	1113	1619	2161	3451	9011	3398	892	237	87	17	14283
gbrdsr000001196	53	805	8202	650	1125	1499	1870	2883	8202	3627	593	115	30	10	14341
gbrdsr000001197	54	749	10865	594	981	1347	1798	3169	10865	2770	537	164	65	17	14397
gbrdsr000001198	53	559	4164	499	741	952	1165	1696	4164	1222	60	5	1	0	14378
gbrdsr000001883	59	753	9722	573	1015	1469	1934	3036	9722	2956	655	153	48	21	14373
Total	521	7465	10865	594	1123	1668	2214	3585	10796	19895	3924	1015	321	106	100552

Fig. 7. Customized Output

Consolidated metrics:

Overall Response Time Summary															
Percentile Stats										No. Of Transaction Exceeding Response Time					
Transaction Type	Min	Avg	Max	50th	80th	90th	95th	99th	100th (Max)	>1s	>2s	>3s	>4s	>5s	Total Transactions
BNET	51	312	2557	240	476	627	804	1266	2557	980	8	0	0	0	55281
Offline	52	741	10865	594	1123	1668	2214	3585	10865	19895	3924	1015	321	106	100552
Total	51	527	10865	417	9937	1564	2143	3562	10865	20875	3932	1015	321	106	155833

Fig. 8. Consolidated Metrics

Graphs:

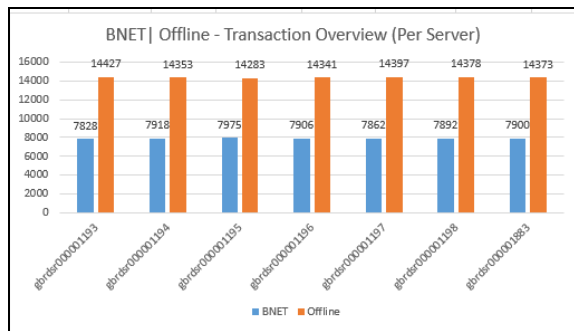


Fig. 9. Transaction Overview

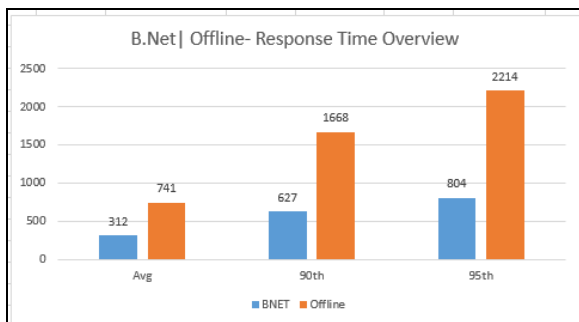


Fig. 10. Response Time Overview

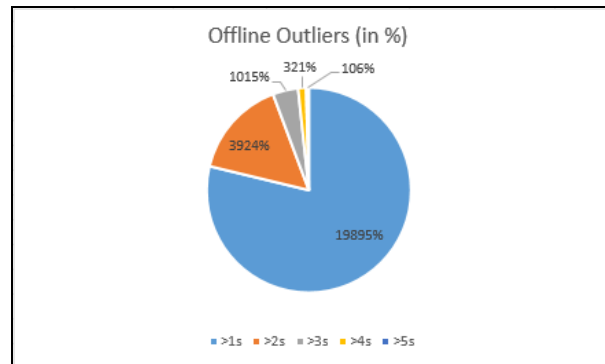


Fig. 11. Offline Outliers

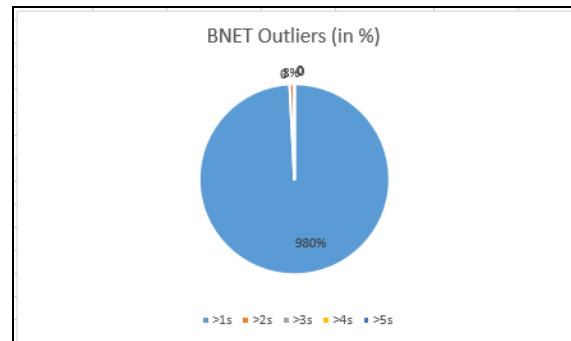


Fig. 11. Offline Outliers

XIII. IMPLEMENTATION

This Utility is purely configurable and can be implemented in any project where Log files parsing are required for any information.

Please find below files for reference:



LogAnalyser_Results_Template.xlsx

<https://github.com/Sangeeta29071986/Log-Analyser>

CONCLUSION

The log analyser parsing utility effectively consolidates and standardizes log data collected from multiple sources, including application servers, web servers, performance tools, and system-level logs. By offering a unified approach to parsing diverse log formats, the utility simplifies log management and enables consistent data extraction across heterogeneous environments.

Its capability to normalize varied log structures into a common schema allows for more efficient querying, filtering, and correlation of events. This not only improves troubleshooting and monitoring but also lays the groundwork for integrating advanced analytics and machine learning for anomaly detection.

Overall, the utility serves as a critical component in modern observability stacks, ensuring that disparate log data can be processed, analysed, and visualized effectively ultimately enhancing system reliability, performance diagnostics, and operational awareness.

REFERENCES

- [1] A. Kent, M. S. Rajarajan, and C. Maple, "Log analysis techniques for cybersecurity compliance in cloud computing," *IEEE Access*, vol. 9, pp. 55235–55249, 2021, doi: 10.1109/ACCESS.2021.3071031.
- [2] Y. He, J. Zhu, J. Wang, and M. R. Lyu, "Loghub: A large collection of system log datasets towards automated log analytics," *arXiv preprint arXiv: 2008.06448*, 2020. [Online]. Available: <https://arxiv.org/abs/2008.06448>
- [3] F. E. Haron, N. Mustaffa, and M. N. Hamidon, "A review of centralized log management for security monitoring," *Indonesian Journal of*

Electrical Engineering and Computer Science, vol. 21, no. 2, pp. 996–1003, Feb. 2021. Doi: 10.11591/ijeecs.v21.i2.pp996-1003

- [4] M. Al-Ayyoub, A. Alsmirat, Y. Jararweh, and M. Al-Habashneh, "A centralized log management solution for enterprise networks," *Journal of King Saud University - Computer and Information Sciences*, vol. 34, no. 4, pp. 1310–1319, May 2022. doi: 10.1016/j.jksuci.2020.02.002
- [5] M. Du, F. Li, G. Zheng, and V. Srikumar, "DeepLog: Anomaly Detection and Diagnosis from System Logs through Deep Learning," *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pp. 1285–1298, 2017. doi: 10.1145/3133956.3134015
- [6] <https://www.keycdn.com/blog/log-analysis-tools>
- [7] D. Oliner, A. Aiken, and I. Stoica, "A Scalable, Language-Independent Approach to Trace Analysis," in **Proc. 2007 Int. Conf. Dependable Systems and Networks (DSN)**, Edinburgh, UK, 2007, pp. 219–228.