Log message with JSON item count for root cause analysis in microservices

Tomoyuki Koyama
Graduate School of Computer Science
Tokyo University of Technology
Tokyo, Japan
g21210247f@edu.teu.ac.jp

Takayuki Kushida
Graduate School of Computer Science
Tokyo University of Technology
Tokyo, Japan
kushida@acm.org

***Abstract*—System administrator takes time to find application error in one microservice caused by HTTP response from another microservice. Although Istio default log message includes the data length of HTTP response body, it doesn't indicate the structure of HTTP response body such as key-value pair counts. Therefore, system administrator takes time to identify root cause of error by Istio default log message and needs to have skills in fault diagnosis. This paper proposes an algorithm that calculates an indicator for application error identification. The algorithm counts items such as map or list in JSON format from HTTP response body as the indicator. The indicator represents the body structure of HTTP response and is recorded in log messages as an additional field. The indicator enables system administrator to identify application error by log messages. The experiment measures log message length increase by the indicator and response time increase by the indicator calculation. The average response time of the proposed log format increases by 7% compared with Istio default log format. The average log message length increase in the proposed method is 216 bytes compared to Istio default log format.**

***Index Terms*—Logging, Message format extension, Root cause analysis**

I. Introduction

Commercial software for production systems has a function that generates log messages [1]. Events occurring in the software are recorded as log messages [2], [3]. Log message is utilized for software debugging, system monitoring, and fault localization [4]–[7]. Experienced developers answer that log messages are the primary source of problem diagnosis according to the survey by Microsoft [8]. When system failure occurs such as database connection failure, system administrator finds and analyzes log messages for troubleshooting [9]–[11].

Istio[1] has a logging feature to track request handling processes [12]. Istio is software that implements service mesh and is utilized by several companies such as Airbnb and eBay[2]. Service mesh is a dedicated layer for handling communication among microservices [13]. Microservice is an architecture style [14]. Each microservice often has RESTful API as interface to receive requests from other microservices [15]. Istio proxy is one of the components in Istio. It is deployed beside each application container [16].

[1]https://istio.io/
[2]https://istio.io/latest/about/case-studies/

Fig. 1 shows the use case scenario. "Doktor" is a web service that is implemented as microservices. Each microservice includes application(app) and Istio proxy. End user accesses "Doktor" to find and download technical papers. Front and fulltext service are microservice in Doktor. Front service provides user interface for end user. Fulltext service serves keyword-based full-text search as backend for front service. When Istio proxy receives or sends HTTP requests, it generates log messages. Log server receives and stores the log messages. Monitoring server checks whether "Doktor" replies with HTTP response corresponding to HTTP request as external monitoring. When the web service replies with HTTP response including error status code such as "500 Internal Server Error", the monitoring server sends a notification to system administrator. System administrator has to diagnose root cause of the HTTP response including error status code. System administrator issues search query to log server in order to find log messages which are related to the HTTP response.

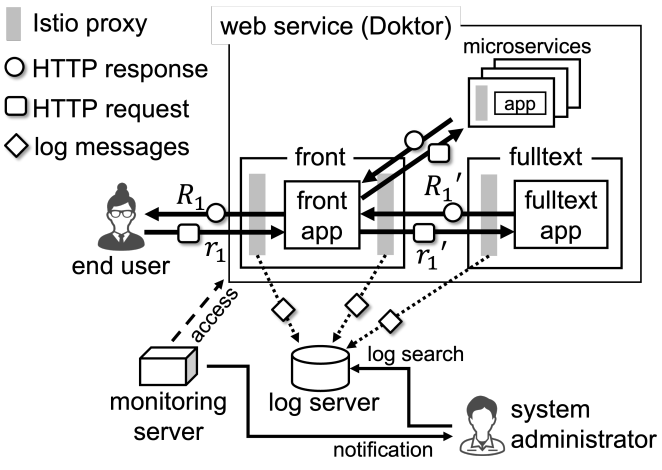

Fig. 1: Use case scenario

Issue

When the monitoring server sends a notification of a failure to system administrator in **Fig. 1**, system administrator analyzes root cause of the failure by log messages. System administrator takes time to find root cause of the application

```
[2022-11-27T08:27:21.565Z] "GET /?keyword=IoT
HTTP/1.1" 200 - via_upstream - "-" 0 18772 61 60
"192.168.200.1,10.42.0.1" "Mozilla/5.0 (Macintosh;
Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/107.0.0.0 Safari/537.36"
"03a2bb05-ff33-46ad-90b8-1e43622c0024"
"doktor.tak-cslab.org" "10.42.0.65:8000"
outbound|4000||front-app.front.svc.cluster.local
10.42.0.247:44120 10.42.0.247:8080 10.42.0.1:44255
- -
```

Code. 1: Example of log message generated from Istio proxy beside front app

error corresponding to the failure by log messages that have Istio default log format. The time to recover system failure depends on the time for root cause analysis. Therefore, it should be short in order to prevent SLA violations. The time that system administrator takes on root cause analysis is caused by Istio log message format. Istio default log format doesn't include the attributes that represent HTTP response body structure. Therefore, system administrator takes time for root cause analysis.

When end user finds technical reports with specified keywords, "front app" replies with error HTTP response. The error is caused by the software bug in "front app" that software developer unintentionally added. For example, "front app" replies with normal HTTP response such as status code 200 when the keyword "IoT" is specified on technical papers search. In contrast, "front app" replies with error HTTP response such as status code 500 when the keyword "recall" is specified. Therefore, the error occurs when no papers are matched with the specified keyword. Whether the error occurs or not depends on paper list in HTTP response body which is received from "full-text app". When the HTTP response body includes 1 or more papers in paper list, "front app" replies with normal HTTP response. When the HTTP response body includes an empty paper list, "front app" replies with error HTTP response. System administrator has to focus on items in HTTP response body such as list or map in order to identify the condition that the error occurs.

Code. 1 shows an example of log message generated from Istio proxy beside "front app" in **Fig. 1**. The log message corresponds to the response R_1' and is made from Istio default log message format[3]. The log message is generated when end user finds technical papers including keyword "IoT" on the web service. The log message doesn't represent data structure and items in HTTP response body. Although HTTP response structure such as array item counts is changed, system administrator cannot observe the changes by Istio default log messages.

One of the methods for error identification by log messages is to record the entire HTTP response body in log messages. System administrator is able to check HTTP response body by log message. The disadvantage of the method is that the total log file size increases as the log message length increases. Log file size fast grows in big data industry [17]. Log file consumes local storage on log server, so that log message length increases pressure on the storage capacity.

[3]https://istio.io/latest/docs/tasks/observability/logs/access-log/

II. Related Study

The study defined log message format identification problem and tried to solve the problem as multi-objective problem [6]. The number of log messages extracted by the proposed method was higher than the other two methods. Although the proposed method is applied to log messages generated from Istio, the log messages don't include fields for identification. Therefore, the proposed method is insufficient for identification with HTTP response body. The study proposed log message transformation to reduce log file size [18]. The study replaced the common message pattern as a message identifier, so that duplicated message patterns were reduced. System administrator needs to prepare message pattern rules to match the common message patterns. The operation editing the message pattern rule is heavy work for system administrator. The method of operational profiles for large deployment added flags on program execution log messages to reduce profile creation time [19]. This study removed and compressed redundant log messages. Although the flags are added to Istio log messages, system administrator takes time to identify different errors by log messages because the flags don't represent HTTP response details.

III. Proposed Method

This study aims to reduce the time to identify the HTTP requests that are the root cause of system failure. This paper proposes Response Body Summary Index (RBSI hereafter). RBSI represents the number of items in HTTP response body which has JSON format. RBSI enables system administrator to identify different HTTP response body structures. The proposed method adds RBSI into log messages. As a result, the method reduces the time for root cause analysis. RBSI is calculated by Response Body Summary Algorithm (RBSA hereafter).

Fig. 2 shows overview of the proposed method. End user accesses web service to see contents. The web service consists of two microservices such as microservice 1 and microservice 2. Each microservice has application container and Istio proxy. For instance, microservice 1 consists of app1 and Istio proxies, and microservice 2 consists of app2 and Istio proxy. Istio proxy generates log message when it receives HTTP requests. The proposed method adds the algorithm RBSA to Istio proxy. RBSA is an algorithm to obtain the indicator RBSI. RBSI is an indicator and represents the structure of JSON in HTTP response body such as key-value pairs or nested items. Istio proxy with RBSA generates log messages including RBSI when Istio proxy receives HTTP request. There is a part of the extended log message in **Fig. 2**. The extended log message includes RBSI in addition to fields for the default log message on Istio proxy. RBSI is appended to the end of the default log message and saved as extended log message.

Algorithm. 1 shows Response Body Summary Algorithm as pseudo code. The algorithm takes HTTP response body

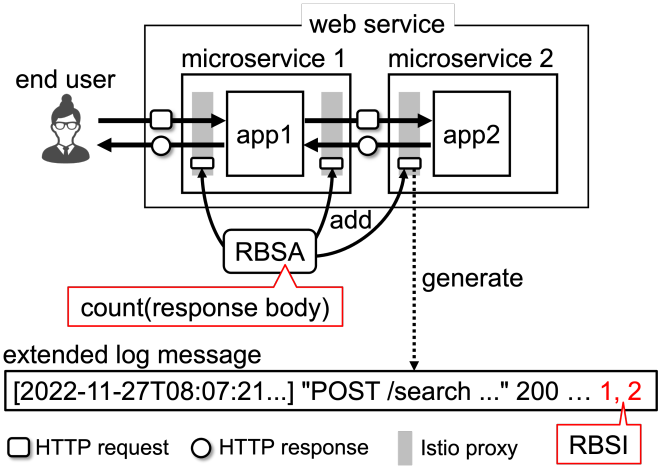

Fig. 2: Overview of the proposed method

Algorithm. 1 Response Body Summary Algorithm

Input: String $responseBody$
Output: List $rbsi$

```
1:  Map total ← ∅
2:  List rbsi ← ∅
3:  function CALCULATE_RBSI(Map payloads, Int depth)
4:      Int counter ← 0
5:      for all k, v ← payloads do
6:          counter ← counter + 1
7:          if TYPE(v) == "map" then
8:              CALCULATE_RBSI(v, depth + 1)
9:          end if
10:     end for
11:     if total[depth] == nil then
12:         total[depth] ← counter
13:     else
14:         total[depth] ← total[depth] + counter
15:     end if
16: end function
17: Map parsed_body ← PARSE_JSON(responseBody)
18: CALCULATE_RBSI(parsed_body, 1)
19: for all k, v ←SORT_BY_KEYS_ASC(total) do
20:     rbsi ← rbsi + v
21: end for
```

of type string as the input variable "responseBody." The algorithm returns RBSI of type list as output variable "rbsi." The variable "total" in line 1 has the number of items in HTTP response body per depth. This variable is type "Map" and takes key/value pairs. The key is the depth of JSON in HTTP response body as string and the value is the number of items in the depth as integer. The variable is initialized with empty. The variable "rbsi" in line 2 has a "List" including RBSI values which is type integer. The variable is initialized with empty. The function "CALCULATE_RBSI" counts items in the argument "payloads" and records item counts for the variable "total." The function takes two arguments. The first

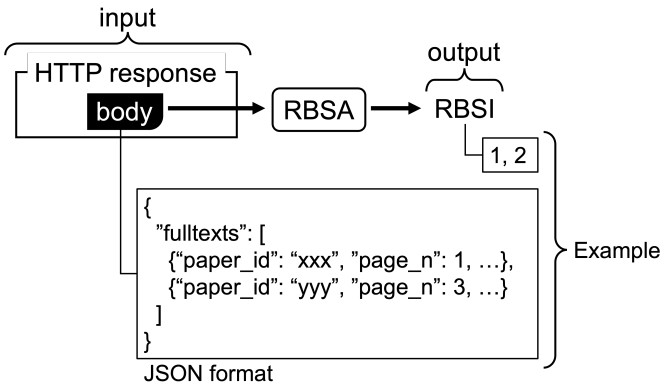

Fig. 3: Model of the proposed method

argument "payloads" takes type "Map" and denotes parsed JSON in HTTP response body. The second argument "depth" takes type "Int" and denotes depth in nested call. This function is designed for recursive calls, so that this variable is required for having current call depth. The variable "counter" in line 4 denotes the number of top level items in "payloads" as type "Int." This variable is initialized with zero. The loop from line 5 to line 10 takes out top level items in "payloads" as key-value pairs. The code in line 6 adds 1 to "counter" and subsequently updates "counter" with the obtained value. This code operates for counting top level items in "payloads." The conditional branch from line 7 to line 9 is executed when the variable "v" is type map. The function "CALCULATE_RBSI" is recursively called with "v" and "depth + 1" as arguments in order to sum up the total number of items per depth in HTTP response body. The conditional branch from line 11 to line 15 updates the variable "total." When "total[depth]" is empty, "counter" is set as the variable "total[depth]." When "total[depth]" isn't empty, "total[depth]" is updated with the sum of "counter" and "total[depth]." The variable "responseBody" is parsed by JSON parser "PARSE_JSON" and is substituted in "parsed_body" in line 17. The function "CALCULATE_RBSI" is called in line 19. The call takes "parsed_body" and 1 as arguments. The loop from line 19 to line 21 joins items in the variable "total" by ascending order.

Fig. 3 shows model of the proposed method. "HTTP response" represents HTTP response with which a microservice replies. HTTP response has contents in body such as JSON or HTML. Example of HTTP response body represents the result of paper search as JSON format. The key "fulltexts" in body of HTTP response includes paper list. The list includes key-value pairs as follows: "paper_id" represents a unique identifier; "page_n" represents page number. RBSA is an algorithm that allows JSON format as input. Microservice supports RESTful API and has JSON based response format [20]. Therefore, the algorithm allows JSON format as input. RBSA exports RBSI which represents input summary. Example of RBSI "1, 2" is calculated by following procedures.

1) RBSA parses HTTP response body from type string to

structured data on programming language such as Map or List.

2) RBSA reads the structured data, and subsequently counts the number of items in first depth. For example, first depth in **Fig. 3** has 1 item ""fulltexts":", so that the number of items in first depth is 1.
3) Second depth of the structured data is counted by RBSA after first depth counts. For example, the second depth in the figure includes 2 items such as "{"paper_id": "xxx", "page_n": 1, ...}" and "{"paper_id": "yyy", "page_n": 3, ...}."
4) Finally, the values of item count per depth are combined into comma-separated values which is called RBSI.

RBSA calculates Response Body Summary Indicator (RBSI). The indicator represents the number of items in input data per depth. System administrator is able to identify different HTTP response body by recording the indicator into log message. RBSI consists of a list of integer values. Each value in the list is separated by commas and empty spaces.

One advantage of RBSI is that log messages with RBSI are shorter message length than logging the entire HTTP response body. **Code. 2** shows an example of HTTP response body on fulltext search app. The fulltext search app provides papers search over RESTful API. When the fulltext search app receives HTTP requests including search keywords, the app finds papers including the search keywords and subsequently replies with HTTP response. The HTTP response body includes paper list that is pairs of paper titles and page numbers. The HTTP response includes a single paper including search keywords on two pages such as 3 and 5. The format of the HTTP response body is JSON. The length of the HTTP response body is 201, so that additional message length of it is 201 bytes. When the HTTP response body is recorded in log message, log message length increases 201 bytes. In contrast, RBSI for the HTTP response body is "[1, 2]." The length of RBSI is 6, so that additional message length is 6 bytes. When the HTTP response body is recorded into message, the log message length increases 6 bytes. Although the structure of the HTTP response body is complex such as deep nested JSON, RBSI only adds a small amount to log message. Log message length depends on storage capacity. When single log message length increases, additional storage capacity is required for local storage on log server [17]. Although the additional storage capacity is small per log message, the total consumed storage capacity increases with increasing the number of log messages. Therefore, reducing log message length is advantage for storage capacity.

Another advantage of RBSI is to support a variety of JSON structures without configuration file edits. RBSA counts items in JSON per depth and obtains RBSI. RBSA doesn't depend on the JSON schema in HTTP response body. Although system administrator doesn't know JSON schema in HTTP response body, RBSA enables RBSI calculation and records it in log message. Several traditional software requires system administrator to edit configuration files for different application support. For example, one web application replies with

```
{
  "fulltexts": [
    {
      "paper_id":
  "4e85846f-ad34-4883-a9cd-fac273351d62",
      "page_n": 3
    },
    {
      "paper_id":
  "4e85846f-ad34-4883-a9cd-fac273351d62",
      "page_n": 5
    }
  ]
}
```

Code. 2: Example of HTTP response body on fulltext search app

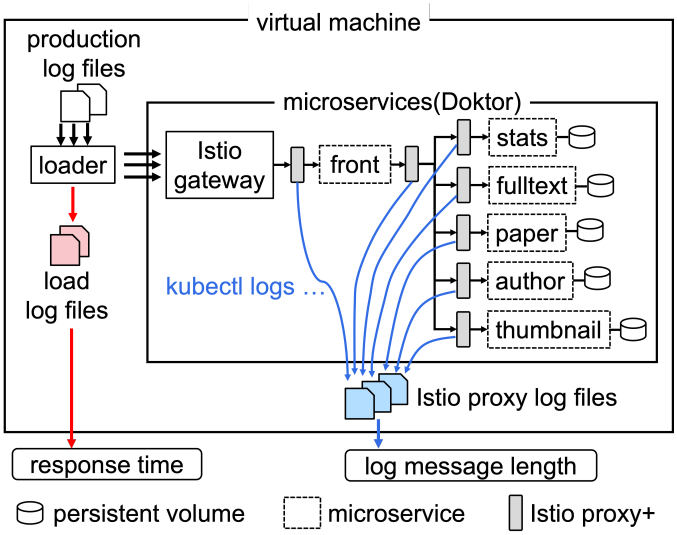

Fig. 4: Experimental environment

HTTP response that includes JSON "{"age": 20}" on body, and another web application replies with HTTP response that includes JSON "{"address": ["Tokyo", "Japan"]}." When system administrator uses traditional software to get a summary of HTTP response body, system administrator needs to edit two different configuration files. While traditional software requires system administrator to edit configuration files, RBSA is not necessary for configuration edits. System administrator saves time because no configuration edits are required. When RBSA bundles with Istio, the advantage is more effective. Istio doesn't require application source code fixes and configuration [12].

Modern commercial software for production systems is required for continuous update to meet business requirements because of significant business changes [21]. Therefore, the structure of HTTP response body such as JSON schema is dynamically changed because of software updates. RBSI doesn't depend on JSON schema and doesn't require system administrator to edit configuration files.

IV. Experiments

Fig. 4 shows the experimental environment. Web service

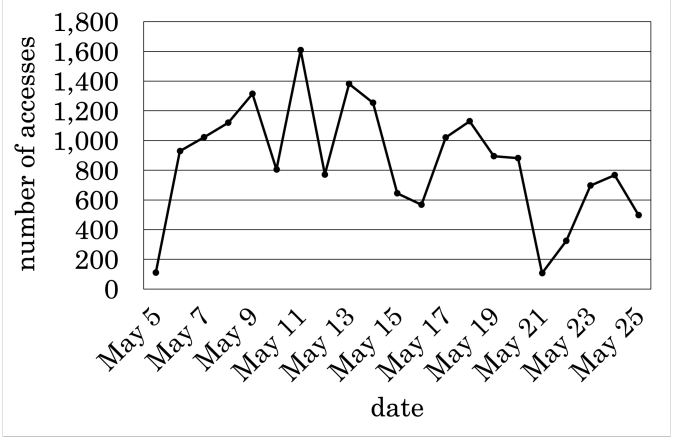

Fig. 5: The number of accesses within 21 days

"Doktor" is deployed to a single virtual machine. The web service provides public access for technical reports over the internet and has Istio gateway and 6 microservices (front, stats, fulltext, paper, author, and thumbnail). The Istio gateway acts as a gateway proxy and provides public access for end user over the internet. When Istio gateway receives an access, it forwards the access to "front" service. Each microservice includes an application and Istio proxy+. Istio proxy+ is an extended implementation of Istio proxy and generates log messages with RBSI. The extension function in Istio proxy+ is implemented by Envoy Filter written in Lua[4]. Envoy Filter is a mechanism that provides a custom function on Istio proxy. It is used for monitoring and traffic shaping [22]. Microservices like "stats" that require a persistent data store use persistent volumes. "Loader" works as an HTTP client to simulate end user access. Loader imports "production log files" collected from Doktor as datasets and sends HTTP requests based on the log files to Istio gateway. The dataset includes 17,845 log messages generated within 21 days from Istio proxy in front of "front" service and in back of "front" service. The log messages are collected from May 5, 2022 to May 25, 2022. **Fig. 5** shows the number of accesses within 21 days. The x-axis is the date, and y-axis is the number of log messages. Each point in the figure equals the number of received HTTP requests in "front" service per day. When Istio proxy+ receives HTTP request, it calculates RBSI from HTTP response body corresponding to the HTTP request and subsequently generates "Istio proxy log files" including RBSI. The method to collect "Istio proxy log files" from Istio proxy+ is to use the "kubectl logs ..." command. The collected log files are stored on the machine's local storage. **Table. I** shows software list for the experimental environment. K3s[5] is utilized for Kubernetes engine, and Istio is utilized for Service mesh. Longhorn[6] provides storage on Kubernetes.

The experiment measures two metrics, which are log message length and response time. Log message length represents byte counts per log entry. It is calculated from "Istio proxy log files". Response time represents the consumed time from HTTP request sent to HTTP response received. It is calculated from "load log files." Shell commands such as curl are utilized for measuring these metrics.

[4]https://www.lua.org/
[5]https://k3s.io/
[6]https://longhorn.io/

TABLE I: Software list for experimental environment

#	Software	Version
Kubernetes engine	K3s	v1.25.5
Service mesh	Istio	v1.16.1
Storage	Longhorn	v1.4.0

V. Experimental Results

Fig. 6 shows comparison of response time overhead by log message format extension. The x-axis is HTTP status code in HTTP response, and the y-axis is response time (unit: millisecond) from HTTP client. The legend is log format type. "default logging" represents Istio default log format. "proposed logging" represents the proposed log format with RBSI. The proposed log format adds RBSI to Istio's default log format. "response body full logging" represents Istio default log format with the entire text of HTTP response body. Response time represents the consumed time from HTTP request sent to HTTP response received. When HTTP status code is 200, "proposed logging" is 4 milliseconds shorter than "response body full logging" and is 7 milliseconds longer than "default logging." This result indicates that log message extension makes response time slow when HTTP requests are correctly processed. When HTTP status code is 307, "proposed logging" is 7 milliseconds longer than "default logging." The difference in response time between "proposed logging" and "response body full logging" is 1 milliseconds. When HTTP status code is 400 or 404 or 405, there is no difference among "default logging", "proposed logging", and "response body full logging" in response time. This result indicates that errors caused by HTTP requests don't affect response time. When HTTP status code is 500, "proposed logging" is 6 milliseconds longer than "default logging." The difference in response time between "proposed logging" and "response body full logging" is 2 milliseconds. The average response time of "proposed logging" increases by 7% compared with "default logging." The average response time of the format increases by 10% compared with "default logging." The result indicates that response time slightly increases by the proposed method.

Fig. 7 shows comparison of log message length among 2 log formats. The x-axis is log message length (unit: byte), and y-axis is counts. The legend is log format type. "default logging" represents Istio default log format. "proposed logging" represents the proposed log format with RBSI. The result for "default logging" indicates that log message length of default logging is shorter than 340 bytes. The most frequent log message length in default logging is from 300 bytes to 319 bytes. This result is compared with other results in the following. The result for "proposed logging" indicates that the most frequent log message length is from 340 bytes to 359 bytes. This result

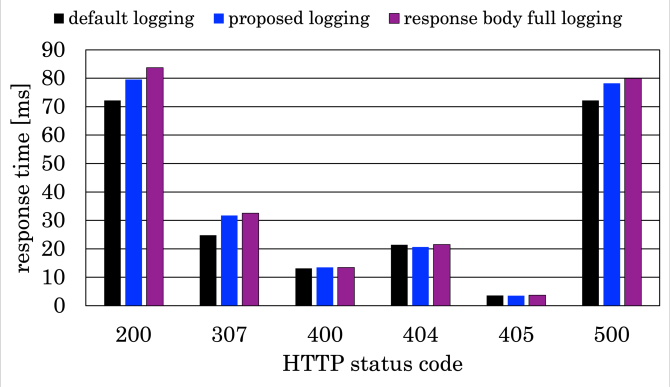

Fig. 6: Comparison of response time overhead by log message format extension

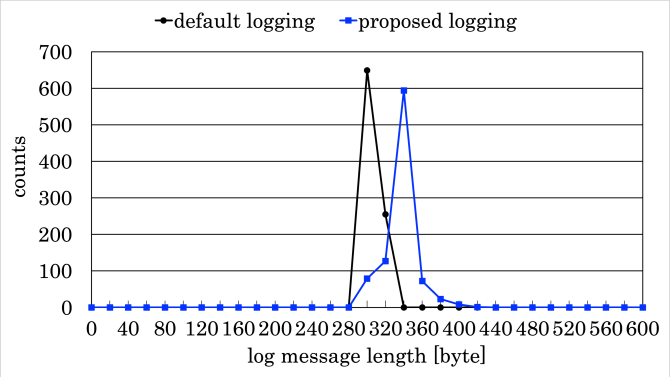

Fig. 7: Comparison of log message length among 3 log formats

suggests that the proposed log format slightly increases log message length. The average log message length increase in "proposed logging" is approximately 216 bytes compared to "default logging."

VI. Discussion

The proposed method counts the number of items in JSON in order to obtain the feature of JSON in HTTP response body such as key-value pairs or nested items. The applicable scope of the indicator RBSI is described below. For example, there is a microservice for book search that returns a list of books that match a search keyword. The microservice provides keyword-based search in ascending or descending order such as publication year. Database server is connected to the microservice and stores book data such as publication year and title. The microservice for book search receives HTTP requests including keywords. When the microservice is faced with error caused by the database failure and replies with an empty book list on the situation expecting 3 books list, system administrator is able to observe the empty HTTP response body by log messages with RBSI. RBSI provides observability to differences in the number of HTTP response items. On the other hand, RBSI cannot represent that a problem of items order in JSON is not correct. On the other hand, RBSI cannot represent that a problem of item order in JSON is not correct. When the microservice for book search receives HTTP request and subsequently replies with an out-of-ordered list of books because of application implementation bugs, RBSI is the same value between out-of-ordered list and correctly ordered list. While RBSI enables representing the difference in the number of items in JSON, RBSI is insufficient to represent order differences in list items.

VII. Conclusion

The goal of this study is to reduce the time for system administrator to identify HTTP requests that are the root cause of system failure. When application error occurs on microservice, log message is utilized for root cause analysis. The default log messages generated from Istio don't include the attribute for HTTP response identification, so that system administrator takes time on identifying the difference between error HTTP response and normal HTTP response. This paper proposes the indicator RBSI that represents the number of items in HTTP response body which has JSON format. The indicator is added to log message. System administrator can identify the difference between error HTTP response and normal HTTP response on microservice by log messages including RBSI. The experiment measures two metrics, which are log message length and response time. The average response time of the proposed log format increases by 7% compared with the default log format.

Acknowledgement

This work was supported by JSPS KAKENHI Grant Number JP20K11776.

References

[1] C. Zhi, J. Yin, S. Deng, M. Ye, M. Fu, and T. Xie, "An exploratory study of logging configuration practice in java," in *2019 IEEE international conference on software maintenance and evolution (ICSME)*. IEEE, 2019, pp. 459–469.

[2] K. A. Kent and M. Souppaya, "Guide to computer security log management:." 2006.

[3] A. Makanju, S. Brooks, A. N. Zincir-Heywood, and E. E. Milios, "Logview: Visualizing event log clusters," in *2008 Sixth Annual Conference on Privacy, Security and Trust*. IEEE, 2008, pp. 99–108.

[4] X. Zhou, X. Peng, T. Xie, J. Sun, C. Ji, D. Liu, Q. Xiang, and C. He, "Latent error prediction and fault localization for microservice applications by learning from system trace logs," in *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2019, pp. 683–694.

[5] S. Alspaugh, A. Ganapathi, M. A. Hearst, and R. Katz, "Better logging to improve interactive data analysis tools," in *Proceedings of the ACM SIGKDD Workshop on Interactive Data Exploration and Analytics (IDEA ' 14)*, 2014, pp. 19–25.

[6] S. Messaoudi, A. Panichella, D. Bianculli, L. Briand, and R. Sasnauskas, "A search-based approach for accurate identification of log message formats," in *2018 IEEE/ACM 26th International Conference on Program Comprehension (ICPC)*. IEEE, 2018, pp. 167–16 710.

[7] S. Khan, A. Gani, A. W. A. Wahab, M. A. Bagiwa, M. Shiraz, S. U. Khan, R. Buyya, and A. Y. Zomaya, "Cloud log forensics: Foundations, state of the art, and future directions," *ACM Computing Surveys (CSUR)*, vol. 49, no. 1, pp. 1–42, 2016.

[8] Q. Fu, J. Zhu, W. Hu, J.-G. Lou, R. Ding, Q. Lin, D. Zhang, and T. Xie, "Where do developers log? an empirical study on logging practices in industry," in *Companion Proceedings of the 36th International Conference on Software Engineering*, 2014, pp. 24–33.

[9] S. Alspaugh, B. Chen, J. Lin, A. Ganapathi, M. Hearst, and R. Katz, "Analyzing log analysis: An empirical study of user log mining," in *28th Large Installation System Administration Conference (LISA14)*, 2014, pp. 62–77.

[10] A. Oliner and J. Stearley, "What supercomputers say: A study of five system logs," in *37th annual IEEE/IFIP international conference on dependable systems and networks (DSN'07)*. IEEE, 2007, pp. 575–584.

[11] F. Skopik, M. Landauer, and M. Wurzenberger, "Online log data analysis with efficient machine learning: A review," *IEEE Security & Privacy*, vol. 20, no. 03, pp. 80–90, 2022.

[12] A. Koschel, M. Bertram, R. Bischof, K. Schulze, M. Schaaf, and I. Astrova, "A look at service meshes," in *2021 12th International Conference on Information, Intelligence, Systems & Applications (IISA)*. IEEE, 2021, pp. 1–8.

[13] D. Cha and Y. Kim, "Service mesh based distributed tracing system," in *2021 International Conference on Information and Communication Technology Convergence (ICTC)*. IEEE, 2021, pp. 1464–1466.

[14] N. Alshuqayran, N. Ali, and R. Evans, "A systematic mapping study in microservice architecture," in *2016 IEEE 9th International Conference on Service-Oriented Computing and Applications (SOCA)*. IEEE, 2016, pp. 44–51.

[15] N. Kratzke, "About microservices, containers and their underestimated impact on network performance," in *Proceedings of CLOUD COMPUTING 2015 (6th. International Conference on Cloud Computing, GRIDS and Virtualization)*, 2015, pp. 165–169.

[16] O. Sheikh, S. Dikaleh, D. Mistry, D. Pape, and C. Felix, "Modernize digital applications with microservices management using the istio service mesh," in *Proceedings of the 28th Annual International Conference on Computer Science and Software Engineering*, 2018, pp. 359–360.

[17] J. Liu, J. Zhu, S. He, P. He, Z. Zheng, and M. R. Lyu, "Logzip: extracting hidden structures via iterative clustering for log compression," in *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2019, pp. 863–873.

[18] A. Makanju, A. N. Zincir-Heywood, and E. E. Milios, "Storage and retrieval of system log events using a structured schema based on message type transformation," in *Proceedings of the 2011 ACM Symposium on Applied Computing*, 2011, pp. 528–533.

[19] A. E. Hassan, D. J. Martin, P. Flora, P. Mansfield, and D. Dietz, "An industrial case study of customizing operational profiles using log compression," in *Proceedings of the 30th international conference on Software engineering*, 2008, pp. 713–723.

[20] N. Kratzke, "About microservices, containers and their underestimated impact on network performance," *arXiv preprint arXiv:1710.04049*, 2017.

[21] L. Leite, C. Rocha, F. Kon, D. Milojicic, and P. Meirelles, "A survey of devops concepts and challenges," *ACM Computing Surveys (CSUR)*, vol. 52, no. 6, pp. 1–35, 2019.

[22] H. Lee, S. Noghabi, B. Noble, M. Furlong, and L. P. Cox, "Bumblebee: Application-aware adaptation for edge-cloud orchestration," in *2022 IEEE/ACM 7th Symposium on Edge Computing (SEC)*. IEEE, 2022, pp. 122–135.