



(51) International Patent Classification:

G06F 11/00 (2006.01) H04L 67/00 (2022.01)

(21) International Application Number:

PCT/US2022/039896

(22) International Filing Date:

10 August 2022 (10.08.2022)

(25) Filing Language:

English

(26) Publication Language:

English

(71) Applicant: **RAKUTEN MOBILE, INC.** [JP/JP]; 1-14-1 Tamagawa, Setagaya-ku, Tokyo, 158-0094 (JP).

(71) Applicant (for SC only): **RAKUTEN MOBILE USA LLC** [US/US]; 800 Concar Dr., San Mateo, California 94402 (US).

(72) Inventors: **NAGAR, Vivek**; c/o Rakuten Symphony India Pte. Ltd., 3rd Floor, C21 Biz. Park, C21 Sq. Opposite Radisson Blu Hotel, MR-10, Indore, MP 452010 (IN). **MO-**

HANANI, Chitra; c/o Rakuten Symphony India Pte. Ltd., 3rd Floor, C21 Biz. Park, C21 Sq. Opposite Radisson Blu Hotel, MR-10, Indore, MP 452010 (IN). **SONI, Taresh**; c/o Rakuten Symphony India Pte. Ltd., 3rd Floor, C21 Biz. Park, C21 Sq. Opposite Radisson Blu Hotel, MR-10, Indore, MP 452010 (IN). **SONI, Mohak**; c/o Rakuten Symphony India Pte. Ltd., 3rd Floor, C21 Biz. Park, C21 Sq. Opposite Radisson Blu Hotel, MR-10, Indore, MP 452010 (IN). **TA-LANKAR, Ashish**; c/o Rakuten Symphony India Pte. Ltd., 3rd Floor, C21 Biz. Park, C21 Sq. Opposite Radisson Blu Hotel, MR-10, Indore, MP 452010 (IN). **VERMA, Jayesh**; c/o Rakuten Symphony India Pte. Ltd., 3rd Floor, C21 Biz. Park, C21 Sq. Opposite Radisson Blu Hotel, MR-10, Indore, MP 452010 (IN).

(74) Agent: **PRITCHETT, Joshua L.**; Hauptman Ham, LLP, 2318 Mill Road, Suite 1400, Alexandria, Virginia 22314 (US).

(81) Designated States (unless otherwise indicated, for every kind of national protection available): AE, AG, AL, AM,

(54) Title: SYSTEM, METHOD, AND NON-TRANSITORY COMPUTER-READABLE MEDIA FOR PROVIDING CLOUD APPLICATION FAULT DETECTION

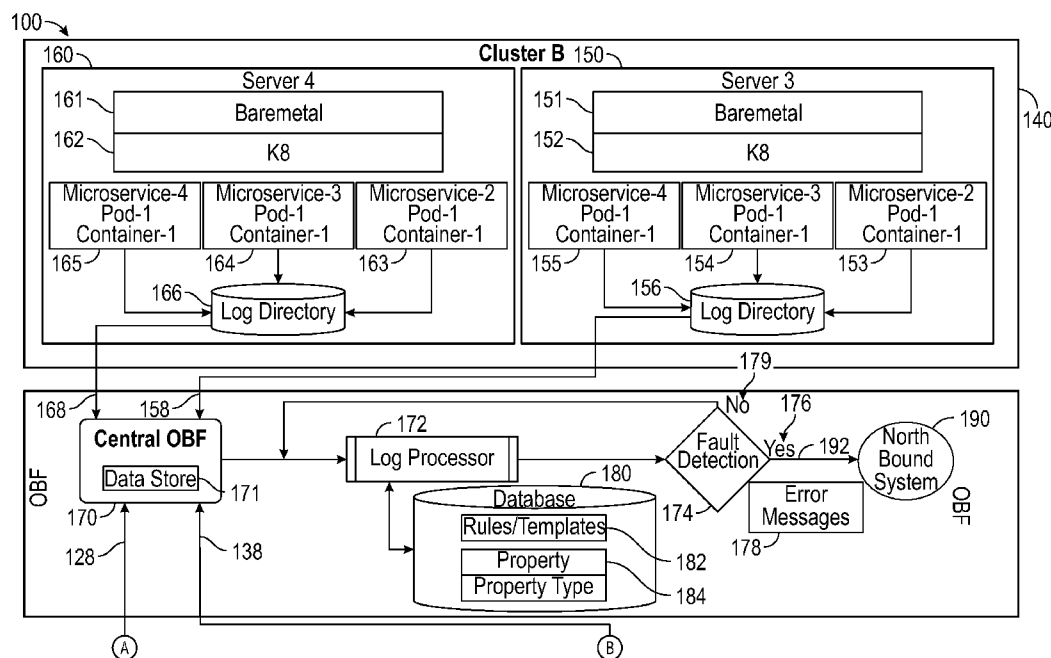


FIG. 1

(57) Abstract: Cloud application fault detection is described. At an observability framework implemented by a processor, log files associated with applications are received, wherein the log files are automatically streamed from at least one log directory. A template is applied to the log files automatically streamed from the at least one log directory as the log files are received to identify faults of the applications associated with the log files. The log files automatically streamed from the at least one log directory are processed as the log files are received based on the faults to generate an error message identifying the faults of the applications associated with the log files.

AO, AT, AU, AZ, BA, BB, BG, BH, BN, BR, BW, BY, BZ, CA, CH, CL, CN, CO, CR, CU, CV, CZ, DE, DJ, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IQ, IR, IS, IT, JM, JO, JP, KE, KG, KH, KN, KP, KR, KW, KZ, LA, LC, LK, LR, LS, LU, LY, MA, MD, ME, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PA, PE, PG, PH, PL, PT, QA, RO, RS, RU, RW, SA, SC, SD, SE, SG, SK, SL, ST, SV, SY, TH, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, WS, ZA, ZM, ZW.

- (84) **Designated States** (*unless otherwise indicated, for every kind of regional protection available*): ARIPO (BW, GH, GM, KE, LR, LS, MW, MZ, NA, RW, SC, SD, SL, ST, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, RU, TJ, TM), European (AL, AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HR, HU, IE, IS, IT, LT, LU, LV, MC, MK, MT, NL, NO, PL, PT, RO, RS, SE, SI, SK, SM, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, KM, ML, MR, NE, SN, TD, TG).

Published:

— with international search report (Art. 21(3))

**SYSTEM, METHOD, AND NON-TRANSITORY COMPUTER-READABLE MEDIA
FOR PROVIDING CLOUD APPLICATION FAULT DETECTION**

TECHNICAL FIELD

[0001] This description relates to a system, method, and non-transitory computer-readable media for providing cloud application fault detection

BACKGROUND

[0002] The scale and complexity of application deployment continues to increase. Understanding how these applications perform, when failure occurs, and what actions need to be taken to address performance is becoming increasingly difficult. To address these issues, characterizing and detecting problems with applications with post-detection diagnostics has become a common task.

[0003] Logs play a key role in identifying the root cause of application faults. Applications write log files to a log directory in response to a fault occurring. The log files from the log directories are capable of providing identification of the faults. Thus, workers, such as application developers and members of the development operations team, access the environment, such as Apache servers, to identify and troubleshoot faults from log files stored at log directories.

[0004] To read the logs from the log directory, workers access the log directories manually, and then refine the manually accessed log records to determine error messages. After the error messages are determined, actions are identified to address the error messages. Quick and easy identification of faults from logs is not currently possible. While manually reading logs can work for smaller applications, large applications can generate thousands of events per second, making manual analysis impossible. Manually reading logs becomes worse when applications are deployed on multiple servers, which may be geographically distributed.

SUMMARY

[0005] In at least embodiment, a method for providing cloud application fault detection includes receiving, at a processor from at least one log directory associated with at least one server, log files associated with applications running on the at least one server, the log files automatically streamed from the at least one directory, storing in memory by the processor, the log files automatically streamed from the at least one log directory as the log files are received, obtaining, by the processor from a database, a template for identifying faults of the applications associated with the log files automatically streamed from the at least one log directory,

applying, by the processor, the template to the log files stored in the memory to identify the faults of the applications associated with the log files, obtaining, by the processor from the database, a configuration file from the database for generating an error message identifying the faults of the applications associated with the log files automatically streamed from the at least one log directory, applying, by the processor, the configuration file to the log files stored in the memory to generate the error message identifying the faults of the applications associated with the log files, and transmitting, by the processor over a network, the error messages to an operations support system for addressing the faults of the applications associated with the log files.

[0006] In at least embodiment, a system for providing cloud application fault detection includes a memory storing computer-readable instructions, and a processor connected to the memory, wherein the processor is configured to execute the computer-readable instructions to receive, from at least one log directory associated with at least one server, log files associated with applications running on the at least one server, the log files automatically streamed from the at least one directory, store in the memory the log files automatically streamed from the at least one log directory as the log files are received, obtain, from a database, a template for identifying faults of the applications associated with the log files automatically streamed from the at least one log directory, apply the template to the log files stored in the memory to identify the faults of the applications associated with the log files, obtain, from the database, a configuration file from the database for generating an error message identifying the faults of the applications associated with the log files automatically streamed from the at least one log directory, apply the configuration file to the log files stored in the memory to generate the error message identifying the faults of the applications associated with the log files, and transmit, over a network, the error messages to an operations support system for addressing the faults of the applications associated with the log files.

[0007] In at least embodiment, a non-transitory computer-readable media having computer-readable instructions stored thereon, which when executed by a processor causes the processor to perform operations including receiving, at an observability framework implemented by the processor, log files associated with applications, the log files automatically streamed from at least one log directory, applying a template to the log files automatically streamed from the at least one log directory as the log files are received to identify faults of the applications associated with the log files, and processing the log files automatically streamed from the at least one log directory as the log files are received based on the faults to generate an error message identifying the faults of the applications associated with the log files.

BRIEF DESCRIPTION OF THE DRAWINGS

[0008] Aspects of the present disclosure are best understood from the following detailed description when read with the accompanying figures. It is noted that, in accordance with the practice in the industry, various features are not drawn to scale. In fact, the dimensions of the various features are able to be increased or reduced for clarity of discussion.

[0009] Fig. 1 illustrates an Observability Framework (OBF) according to at least one embodiment.

[0010] Fig. 2 illustrates a Log Shipper according to at least one embodiment.

[0011] Fig. 3 illustrates a Log Processing System according to at least one embodiment.

[0012] Fig. 4 is a flowchart of a method for providing cloud application fault detection according to at least one embodiment.

[0013] Fig. 5 is a high-level functional block diagram of a processor-based system according to at least one embodiment.

DETAILED DESCRIPTION

[0014] Embodiments described herein describes examples for implementing different features of the provided subject matter. Examples of components, values, operations, materials, arrangements, or the like, are described below to simplify the present disclosure. These are, of course, examples and are not intended to be limiting. Other components, values, operations, materials, arrangements, or the like, are contemplated. For example, the formation of a first feature over or on a second feature in the description that follows include embodiments in which the first and second features are formed in direct contact and include embodiments in which additional features are formed between the first and second features, such that the first and second features are unable to make direct contact. In addition, the present disclosure repeats reference numerals and/or letters in the various examples. This repetition is for the purpose of simplicity and clarity and does not in dictate a relationship between the various embodiments and/or configurations discussed.

[0015] Further, spatially relative terms, such as “beneath,” “below,” “lower,” “above,” “upper” and the like, are used herein for ease of description to describe one element or feature’s relationship to another element(s) or feature(s) as illustrated in the figures. The spatially relative terms are intended to encompass different orientations of the device in use or operation in addition to the orientation depicted in the FIGS. The apparatus is otherwise oriented (rotated 90 degrees or at other orientations) and the spatially relative descriptors used herein likewise

are interpreted accordingly.

[0016] Embodiments described herein describes examples for implementing different features of the provided subject matter. Examples of components, values, operations, materials, arrangements, or the like, are described below to simplify the present disclosure. These are, of course, examples and are not intended to be limiting. Other components, values, operations, materials, arrangements, or the like, are contemplated. For example, the formation of a first feature over or on a second feature in the description that follows include embodiments in which the first and second features are formed in direct contact and include embodiments in which additional features are formed between the first and second features, such that the first and second features are unable to make direct contact. In addition, the present disclosure repeats reference numerals and/or letters in the various examples. This repetition is for the purpose of simplicity and clarity and does not in dictate a relationship between the various embodiments and/or configurations discussed.

[0017] Embodiments described herein provide a system, method, and non-transitory computer-readable media for providing cloud application fault detection. At an observability framework implemented by a processor, log files associated with applications are received, wherein the log files are automatically streamed from at least one log directory. A template is applied to the log files automatically streamed from the at least one log directory as the log files are received to identify faults of the applications associated with the log files. The log files automatically streamed from the at least one log directory are processed as the log files are received based on the faults to generate an error message identifying the faults of the applications associated with the log files.

[0018] Advantages include a system that provides quick and easy identification of faults from logs as log files are received. The cloud application fault detection enables events from large applications to be processes. Further, the cloud application fault detection is able to handle log files received from multiple, geographically distributed servers. The cloud application fault detection provides automated collection of log files for analysis in the cloud, rather than relying on personnel to manually access logs files directly from log directories

[0019] Fig. 1 illustrates an Observability Framework (OBF) 100 according to at least one embodiment.

[0020] In Fig. 1, Cluster A 110 includes Server 1 120 and Server 2 130. Server 1 is implemented using Baremetal operating system 121. Baremetal operating system 121 provides a physical computer specifically designed to run dedicated services without any interruptions for extended periods. The term ‘baremetal’ is used to refer to a physically dedicated server

rather than a virtualized environment and modern cloud hosting forms. Within a data center, servers based on Baremetal operating system 121 are not shared between multiple clients. When using Baremetal operating system 121, other users do not compete on the same system for resources so that users are provided higher performance. A dedicated server based on Baremetal operating system 121 is able to handle a more significant workload than a similar virtual machine in most cases. This allows dedicated hosting to be provided to users that need top levels of performance. Compared to other types of dedicated servers, servers based on Baremetal operating system 121 is often easier to manage.

[0021] Kubernetes (K8) 122 is shown running on Baremetal operating system 121. K8 122 provides an open-source container management platform for managing containerized applications in a clustered environment. K8 122 is configured to instruct a data shipper how to treat logs stored in a log directory. For example, in at least one embodiment, Kubernetes uses pod annotations to tell a data shipper how to treat logs stored in a log directory. Data shippers are configured on a server to send data to a predetermined location. In K8 122 a pod represents a group of one or more containers running together, wherein a pod is the smallest execution unit in Kubernetes 122. A pod encapsulates one or more applications and includes one or more containers. Containers separate functions within a server and run logically in a pod. A container includes storage/network resources, and a specification for how to run the containers.

[0022] Applications represented by Microservices are running in a cloud environment and are writing logs to a log directory. Server 1 120 provides Microservice-2 123, Microservice-3 124, and Microservice-4 125. Microservice-2 123, Microservice-3 124, and Microservice-4 125 are applications and are carried in Container 1 and deployed in Pod-1. Microservice-2 123, Microservice-3 124, and Microservice-4 125 write log files to Log Directory 126 in Server 1 120. Log files are explanatory records of events regarding the application, its performance, and user activities. Examples of events include deleting, or modifying a file for the application. They also include system configuration changes. Thus, log files are useful in identifying and correcting problems.

[0023] Cluster A 110 also includes Server 2 130 that is implemented using Baremetal operating system 131. Kubernetes (K8) 132 is running on Baremetal 131. Server 1 130 provides Microservice-2 133, Microservice-3 134, and Microservice-4 135. Microservice-2 133, Microservice-3 134, and Microservice-4 135 are applications and are carried in Container 1 and deployed in Pod-1. Microservice-2 133, Microservice-3 134, and Microservice-4 135 write log files to Log Directory 136 in Server 1 130.

[0024] Cluster B 140 includes Server 3 150 and Server 4 160. Server 4 150 is implemented using Baremetal operating system 151. Kubernetes (K8) 152 is running on Baremetal 151. Server 1 150 provides Microservice-2 153, Microservice-3 154, and Microservice-4 155. Microservice-2 153, Microservice-3 154, and Microservice-4 155 are applications and are carried in Container 1 and deployed in Pod-1. Microservice-2 153, Microservice-3 154, and Microservice-4 155 write log files to Log Directory 156 in Server 1 150.

[0025] Cluster B 140 also includes Server 4 160. Server 4 160 is implemented using Baremetal operating system 161. Kubernetes (K8) 162 is running on Baremetal 161. Server 1 160 provides Microservice-2 163, Microservice-3 164, and Microservice-4 165. Microservice-2 163, Microservice-3 164, and Microservice-4 165 are applications and are carried in Container 1 and deployed in Pod-1. Microservice-2 163, Microservice-3 164, and Microservice-4 165 write log files to Log Directory 166 in Server 1 160.

[0026] Microservices 123-125, 133-135, 153-155, 163-155 have applications running on the K8 layers 122, 132, 152, 162, respectively. K8 layers 122, 132, 152, 162 manage the applications containers. Applications of Microservices 123-125, 133-135, 153-155, 163-155 write log files to Log Directories 126, 136, 156, 166. For example, applications of Microservices 123-125, 133-135, 153-155, 163-155 generate log files on a regular basis, e.g., how many users have logged into an application, how many users are trying to access a particular feature of the application, etc. The applications are capable of generating thousands of log files. In response to an issue associated with a fault condition occurring in applications of Microservices 123-125, 133-135, 153-155, 163-155, identification of the issue is written in a log file that is stored in Log Directories 126, 136, 156, 166.

[0027] Observability Framework (OBF) 100 provides automated collection of log files for analysis in the cloud, rather than relying on personnel to manually access logs files directly from Log Directories 126, 136, 156, 166. OBF 100 includes a Central OBF device 170. OBF 100 is coupled via Connection 128 to Log Directory 126 of Server 1 120. OBF 100 is coupled via Connections 138 to Log Directory 136 of Server 2 130. OBF 100 is coupled via Connection 158 to Log Directory 156 of Server 3 150. OBF 100 is coupled via Connection 168 to Log Directory 166 of Server 4 160.

[0028] Connections 128, 138, 158, 168 are implemented using at least one of a wireless connection and a wired connection. In at least one embodiment, Connections 128, 138, 158, 168 are implemented as a wireless connection in accordance with any IEEE 802.11 Wi-Fi protocols, Bluetooth protocols, Bluetooth Low Energy (BLE), or other short range protocols that operate in accordance with a wireless technology standard for exchanging data using any

licensed or unlicensed band such as the citizens broadband radio service (CBRS) band, 2.4 GHz bands, 5 GHz bands, or 6 GHz bands. Additionally, in at least one embodiment, Connections 128, 138, 158, 168 are implemented using a wireless connection that operates in accordance with, but is not limited to, RF4CE protocol, ZigBee protocol, Z-Wave protocol, or IEEE 802.15.4 protocol. In at least one embodiment, Connections 128, 138, 158, 168 are implemented using a coax (MoCA) network. In at least one embodiment, Connections 128, 138, 158, 168 are a wired Ethernet connection.

[0029] Central OBF device 170 receives log files from Log Directories 126, 136, 156, 166. For example, Log Directories 126, 136, 156, 166 stream log files to Central OBF 170. By streaming the log files from Log Directories 126, 136, 156, 166, a persistent or continuous flow is provided to Central OBF 170 for processing by Log Processor 172. Central OBF 170 provides automated collection of log files for analysis thereby providing improved operations.

[0030] Central OBF 170 implements Data Store 171 for continuously receiving the log files associated with applications of Microservices 123-125, 133-135, 153-155, 163-165 from Log Directories 126, 136, 156, 166. In at least one embodiment, a shipper that is implemented at Server 1 120, Server 2 130, Server 3, 150, and Server 4 160 forwards the log files to the OBF 100. The shipper monitors the log files in the Log Directories 126, 136, 156, 166, collects log events, and forwards them to Central OBF device 170. For example, Filebeat® software provides a lightweight shipper whose purpose is to forward and centralize logs from a Kafka® server.

[0031] In at least one embodiment, Data Store 171 ingests streaming log files in real-time. Data Store 171 manages the constant influx of log files that are then provided to Log Processor 172. In at least one embodiment, Central OBF 170 implements Data Store 171 as a Kafka® server. In at least one embodiment, Data Store 171 is a distributed system of Kafka® servers communicating with Log Processor 172. For example, Data Store 171 implements Kafka® servers as a cluster of one or more servers that can span multiple datacenters or cloud regions. Kafka® servers receive the log files received at Central OBF 170. Kafka® servers decouple data streams so there is very low latency. Kafka® servers store and process data streams as log files are received.

[0032] Log Processor 172 processes the log files automatically streamed steadily from the Log Directories 126, 136, 156, 166 as the log files are received to identify faults of the applications associated with the log files. For example, once a log file is available in Kafka® server, a Spark job parses the log files to obtain meaningful information about the log file. Log Files are parsed according to recognized formats, such as JSON and XML. Log Files in other

formats are parsed using additional parsing rules.

[0033] A Database 180 stores rules for identifying and generating error messages or alarms identifying the faults of the application associated with the log files. The format of the log files depend on the type used by a particular vendor. For example, a JSON (JavaScript Object Notation) log file provides a structured data format and, in at least one embodiment, includes key value pairs. Vendors using the JSON log file format share the key name used for fault detection and a value identifying the fault.

[0034] Log Processor 172 identifies the log file and the value identifying the fault. Log Processor 172 detects anomalies from analysis of the log files. For example, suspicious events or data associated with error messages are identified. Database 180 stores Rules/Templates 182 that are used to defines rules and log fault patterns, and are applied to the log files to detect faults. In at least one embodiment, Database 180 includes a Logging yaml File 184 that is applied to the log files to identify faults from the log files. Logging yaml File 184 is a regular expression that includes “\$.spec.pattern” and “\$spec.pattern.type” to define patterns and pattern types for identifying faults from the log files. The Logging yaml File 184 defines log file formats, filters, and processing options. Once the regular expression of the Logging yaml File 184 is applied, a key value is obtained that identifies the log as a fault.

[0035] Vendors provide information or identifiers for identifying logs from their applications. Such identifiers are included in the log files for identifying a fault event or alarm. Identifiers include values, such as an alarm or event value to identify the type of fault. The log files are provided by applications in one or more different formats, such as plain-text files, JSON (JavaScript Object Notation) files, XML files, etc. Log Processor 172 accesses Database 180 to obtain a Template 182 to apply to identified faults. Log Processor 172 applies Rules/Template 182 to the identified faults and generates Errors Messages 178.

[0036] In at least one embodiment, the log files automatically streamed from the at least one log directory are processed as the log files are received to generate deserialized log files. Deserializing involves the parsing of the log file to obtain meaningful information from the log file. In at least one embodiment, a Spark job is used to deserialize the data. For example, computer data is usually transmitted in a serialized form. For example, deserialized data is organized in data structures such as arrays, records, graphs, classes, or other configurations for efficiency. Serialization converts and changes the data organization into a linear format. XML, YAML, and JSON are commonly used formats for serialized data. Using Java as an example platform for serialization, an object of type Address would logically have separate objects of Street, City, State, And Postal Code. Once serialized, this data is converted into a linear data

format (such as the XML text form in the diagram) representing the Address object, e.g., `<Address><Street>123 Main Street</Street><City>Smithville</City><State>New York</State><Postal Code>12345</Postal Code></Address>`. Deserialization reverses the process and causes the Address objects to be instantiated in memory as separate objects.

[0037] In at least one embodiment, the data is enriched and the enriched log data is returned for further processing. Enriching the log file data involves processing the deserialized log files to add information. For example, such additional information includes data for supporting searching of the enriched log files.

[0038] Fault Detection 174 is performed to identify faults in the log files. In response to faults not being detected 179, the Log Processor 172 monitors for additional log files that are received by Central OBF 170. In response to faults being identified in the log files 176, Error Messages 178 identifying the faults of the applications associated with the log files are generated. A fault is identified in the log files through a key value in the log file. Error Messages 178 that are generated are based on the fault associated with the key value. Error Messages 178 are converted to a predetermined format and sent to a North Bound Process 190, such as Operations Support System (OSS). Operations Support System (OSS) is used to provide management, inventory, engineering, planning, and repair functions. Upon receipt of logging errors, OSS logs the issue and generates a trouble ticket that is sent to personnel to begin the repair process. Alerts based on the identified faults are used to address the faults.

[0039] North Bound Process 190 receives Error Messages 178 via Connection 192. Connection 192 is implemented using at least one of a wireless connection and a wired connection. In at least one embodiment, Connection 192 is implemented as a wireless connection in accordance with any IEEE 802.11 Wi-Fi protocols, Bluetooth protocols, Bluetooth Low Energy (BLE), or other short range protocols that operate in accordance with a wireless technology standard for exchanging data using any licensed or unlicensed band such as the citizens broadband radio service (CBRS) band, 2.4 GHz bands, 5 GHz bands, or 6 GHz bands. Additionally, in at least one embodiment, Connection 192 is implemented using a wireless connection that operates in accordance with, but is not limited to, RF4CE protocol, ZigBee protocol, Z-Wave protocol, or IEEE 802.15.4 protocol. In at least one embodiment, Connection 192 is implemented using a coax (MoCA) network. In at least one embodiment, Connection 192 is a wired Ethernet connection.

[0040] North Bound Process 190 identifies incidents and appropriate tickets are created in response to the Error Messages 178 along with the identification of the severity of the faults. The ticket identifies the fault and provides other information and is assigned to a developer or

other worker for solving the issue associated with the ticket.

[0041] Log file data, such as log files not identified as being associated with a fault 179, is stored in a storage device, such as Database 180. In at least one embodiment, Database 180 includes an Open Distro Elasticsearch Database. Open Distro is an open source cloud-native tool that is able to store the log file data, which is then able to be visualized on Kibana and other dashboard interfaces. Such dashboard interfaces present a visualization of the log file data on a display device for analysis. For example, Elasticsearch provides a distributed, multitenant-capable full-text search engine with an HTTP web interface and schema-free JSON documents. Elasticsearch includes a distributed search and analytics engine configured for log analytics, full-text search, and operational intelligence use cases. In response to faults not being identified in the log files, the OBF 100 continues to process received log files at the Log Processor 172.

[0042] OBF 100 reduces major debugging time compared to manually accessing log directories to identify faults. In at least one embodiment, the OBF 100 works with any application running on any cloud native environment. In a cloud native environment, there are usually many clusters, wherein some servers or simple machines are part of the clusters. Rather than having to manually access application log files from a log directory, the OBF automates the collection of log files from one or more log directories and analysis of the log files is performed in the cloud.

[0043] Fig. 2 illustrates a Log Shipper 200 according to at least one embodiment.

[0044] In Fig. 2, Log File 1 210, and Log File 2 212 are generated by Application Process 1 220. Log File 3 230 is generated by Application Process 2 240. Log File 1 210, Log File 2 212, and Log File 3 230 are provided to a Spooler 250 in Log Directory 260. A Shipper 270 monitors Log File 1 210, Log File 2 212, and Log File 3 230 in Log Directory 260. In response to Log File 1 210, Log File 2 212, and Log File 3 230 being written to Log Directory 260, Log File 1 210, Log File 2 212, and Log File 3 230 are pushed to the OBF 280 via Connection 272.

[0045] . Connection 272 is implemented using at least one of a wireless connection and a wired connection. In at least one embodiment, Connection 272 is implemented as a wireless connection in accordance with any IEEE 802.11 Wi-Fi protocols, Bluetooth protocols, Bluetooth Low Energy (BLE), or other short range protocols that operate in accordance with a wireless technology standard for exchanging data using any licensed or unlicensed band such as the citizens broadband radio service (CBRS) band, 2.4 GHz bands, 5 GHz bands, or 6 GHz bands. Additionally, in at least one embodiment, Connection 272 is implemented using a wireless connection that operates in accordance with, but is not limited to, RF4CE protocol,

ZigBee protocol, Z-Wave protocol, or IEEE 802.15.4 protocol. In at least one embodiment, Connection 272 is implemented using a coax (MoCA) network. In at least one embodiment, Connection 272 is a wired Ethernet connection.

[0046] In at least one embodiment, Shipper 270 is a Filebeat® process that forwards Log File 1 210, Log File 2 212, and Log File 3 230 to the OBF 280. Shipper 270 monitors Log Directory 260, collects log events, and forwards Log File 1 210, Log File 2 212, and Log File 3 230 to OBF 280. Shipper 270 steadily streams Log File 1 210, Log File 2 212, and Log File 3 230 to OBF 280.

[0047] OBF 280 implements Data Store 282 for continuously receiving the log files associated with applications from log directories. In at least one embodiment, Data Store 282 ingests streaming log files in real-time. Data Store 282 manages the constant influx of log files that are then provided to a log processor for processing. In at least one embodiment, OBF 280 implements Data Store 282 as a Kafka® server. In at least one embodiment, Data Store 282 is a distributed system of Kafka® servers communicating with the log processor. For example, Data Store 282 implements Kafka® servers as a cluster of one or more servers that can span multiple datacenters or cloud regions.

[0048] In Fig. 2, Log File 3 230 includes an Identifier (ID) 232. Identifier 232 are included in the log files for identifying a fault event or alarm. While not shown in Fig 2, Log File 1 210 and Log File 2 212 also include identifiers.

[0049] Fig. 3 illustrates a Log Processing System 300 according to at least one embodiment.

[0050] In Fig. 3, Central OBF Device 302 provides Log Files 310 to Storage Device 322 of Log Processor 320 via Connection 324. The Log Files 310 are written to Storage Device 322 for processing by Processor 330.

[0051] Central OBF Device 302 implements Data Store 304 for continuously receiving the log files associated with applications from log directories. In at least one embodiment, Data Store 304 ingests streaming log files in real-time. Data Store 304 manages the constant influx of log files that are then provided to a log processor for processing. In at least one embodiment, Central OBF 302 implements Data Store 304 as a Kafka® server. In at least one embodiment, Data Store 304 is a distributed system of Kafka® servers communicating with the log processor. For example, Data Store 304 implements Kafka® servers as a cluster of one or more servers that can span multiple datacenters or cloud regions.

[0052] Processor 330 accesses Database 340 via Connection 332 to obtain Logging Yaml File 342 for processing the Log Files 310. Logging Yaml File 342 is a logging configuration file. Logging Yaml File 342 includes \$.spec.pattern 344 and \$.spec.pattern_type 346 that are used

to identify a pattern and pattern type of faults, respectively, in Log Files 310.

[0053] In at least one embodiment, Log Files 310 are automatically streamed from at least one log directory and are processed by Processor 330 at Log Processor 320 as Log Files 310 are received. Processor 330 performs one or more processes to analyze the Log Files 310. For example, in at least one embodiment, Processor 330 performs Deserializing 334 on Log Files 310. Deserializing 334 involves the parsing of the Log Files 310 to obtain meaningful information from the Log Files 310. In at least one embodiment, a Spark job is used to deserialize the data.

[0054] In at least one embodiment, Processor 330 performs Enriching 336 on the data obtain from the Log Files 310. Enriching 336 the data of Log Files 310 involves processing the Log Files 310 to add information. For example, such additional information includes data for supporting searching of the enriched Log Files 310. In at least one embodiment, Processor 330 performs Parsing 338 of the Log Files 310 to identify faults of applications associated with the Log Files 310. Log Files are parsed according to recognized formats, such as JSON and XML. Log Files in other formats are parsed using additional parsing rules. In at least one embodiment, Identifiers 312 are included in Log Files 310 for identifying a fault/event or alarm. Identifiers 312 include values, such as an alarm or event value to identify the type of fault. The Log Files 310 are provided by applications in one or more different formats, such as plain-text files, JSON (JavaScript Object Notation) files, XML files, etc.

[0055] Processor 330 accesses Database 340 to obtain a Template 348 to apply to identified faults. Processor 330 applies Template 348 to the identified faults and generates Logging Errors 350. Processor 330 causes Logging Errors 350 to be sent to North Bound Systems 360 via Connection 362. For example, in at least one embodiment, in response to receiving Logging Errors 350 at North Bound Systems 360, North Bound Systems 360 identify incidents and appropriate tickets are created in response to the incidents along with the identification of the fault and its severity. The ticket identifies the fault and provides other information and is assigned to a developer or other worker for addressing the issue associated with the ticket.

[0056] In at least one embodiment, Storage Device 322 sends log file data to Logging Analysis Database (DB) 370 via Connection 372. In at least one embodiment, log file data, such as log files not identified as being associated with a fault, is stored in Logging Analysis Database (DB) 370. In at least one embodiment, Logging Analysis Database (DB) 370 stores log data associated with log files identified as being associated with a fault. In at least one embodiment, Logging Analysis Database (DB) 370 includes an Open Distro Elasticsearch Database. Open Distro is an open source cloud-native tool that is able to store the log file data, which is then

able to be visualized on Kibana and other dashboard interfaces. Such dashboard interfaces present a visualization of the log file data on a display device for analysis. For example, Elasticsearch provides a distributed, multitenant-capable full-text search engine with an HTTP web interface and schema-free JSON documents. Elasticsearch includes a distributed search and analytics engine configured for log analytics, full-text search, and operational intelligence use cases. In response to faults not being identified in the log files, received log files continue to be processed at the Log Processor 320.

[0057] Connections 324, 332, 362, 372 are implemented using at least one of a wireless connection and a wired connection. In at least one embodiment, Connections 324, 332, 362, 372 are implemented as a wireless connection in accordance with any IEEE 802.11 Wi-Fi protocols, Bluetooth protocols, Bluetooth Low Energy (BLE), or other short range protocols that operate in accordance with a wireless technology standard for exchanging data using any licensed or unlicensed band such as the citizens broadband radio service (CBRS) band, 2.4 GHz bands, 5 GHz bands, or 6 GHz bands. Additionally, in at least one embodiment, Connections 324, 332, 362, 372 are implemented using a wireless connection that operates in accordance with, but is not limited to, RF4CE protocol, ZigBee protocol, Z-Wave protocol, or IEEE 802.15.4 protocol. In at least one embodiment, Connections 324, 332, 362, 372 are implemented using a coax (MoCA) network. In at least one embodiment, Connections 324, 332, 362, 372 are a wired Ethernet connection.

[0058] Fig. 4 is a flowchart of a method 400 for providing cloud application fault detection according to at least one embodiment.

[0059] In Fig. 4, method 400 starts (S410), and log files associated with applications, log files associated with applications automatically streamed from at least one log directory are continuously received from a shipper (S414). Referring to Fig. 2, a Shipper 270 monitors Log File 1 210, Log File 2 212, and Log File 3 230 in Log Directory 260. In response to Log File 1 210, Log File 2 212, and Log File 3 230 being written to Log Directory 260, Log File 1 210, Log File 2 212, and Log File 3 230 are pushed to the OBF 280 via Connection 272. In at least one embodiment, Shipper 270 is a Filebeat® process that forwards Log File 1 210, Log File 2 212, and Log File 3 230 to the OBF 280. Shipper 270 monitors Log Directory 260, collects log events, and forwards Log File 1 210, Log File 2 212, and Log File 3 230 to OBF 280 via Connection 272. Shipper 270 steadily streams Log File 1 210, Log File 2 212, and Log File 3 230 to OBF 280.

[0060] The received log are processed as the log files are received to generate deserialized log files (S418). Referring to Fig 3, in at least one embodiment, Processor 330 performs one or

more processes to analyze the Log Files 310. For example, in at least one embodiment, Processor 330 performs Deserializing 332 on Log Files 310. Deserializing 332 involves the parsing of the Log Files 310 to obtain meaningful information from the Log Files 310. In at least one embodiment, a Spark job is used to deserialize the data.

[0061] Log files, whether serialized or un-serialized, are stored in a storage device (S422). Referring to Fig. 1, Log file data, such as log files not identified as being associated with a fault 179, is stored in a storage device, such as Database 180. Referring to Fig. 3, in at least one embodiment, log file data, such as log files not identified as being associated with a fault, is stored in Logging Analysis Database (DB) 370. In at least one embodiment, Logging Analysis Database (DB) 370 stores log data associated with log files identified as being associated with a fault.

[0062] Search and analytics are performed on the log files stored in the storage device (S426). Referring to Fig. 3, in at least one embodiment, Logging Analysis Database (DB) 370 includes an Open Distro Elasticsearch Database. Open Distro is an open source cloud-native tool that is able to store the log file data, which is then able to be visualized on Kibana and other dashboard interfaces. Such dashboard interfaces present a visualization of the log file data on a display device for analysis. For example, Elasticsearch provides a distributed, multitenant-capable full-text search engine with an HTTP web interface and schema-free JSON documents. Elasticsearch includes a distributed search and analytics engine configured for log analytics, full-text search, and operational intelligence use cases. In response to faults not being identified in the log files, received log files continue to be processed at the Log Processor 320.

[0063] The deserialized log files are processed to generate enriched log files having additional information (S430). Referring to Fig. 3, in at least one embodiment, Processor 330 performs Enriching 336 on the data obtain from the Log Files 310. Enriching 336 the data of Log Files 310 involves processing the Log Files 310 to add information. For example, such additional information includes data for supporting searching of the enriched Log Files 310.

[0064] A logging configuration file is received (S434). Referring to Fig. 1, Database 180 includes a Logging yaml File 184 that is applied to the log files to identify faults from the log files.

[0065] The logging configuration file is applied to the log files as the log files are received to identify a pattern and a pattern type associated with log files (S438). Referring to Fig. 3,

[0066] Logging yaml File 184 is a regular expression that includes “\$.spec.pattern” and “\$.spec.pattern.type” to define patterns and pattern types for identifying faults from the log files. The Logging yaml File 184 defines log file formats, filters, and processing options. Once the

regular expression of the Logging yaml File 184 is applied, a key value is obtained that identifies the log as a fault.

[0067] A template is applied to the log files as the log files are received (S442). Referring to Fig. 3, Log Processor 172 accesses Database 180 to obtain a Template 182 to apply to identified faults. Processor 330 applies Template 348 to the identified faults and generates Logging Errors 350.

[0068] In response to applying the template, an error message identifying the faults of the applications associated with the log files is generated (S446). Referring to Fig. 3, Processor 330 applies Template 348 to the identified faults and generates Logging Errors 350. Processor 330 causes Logging Errors 350 to be sent to North Bound Systems 360 via Connection 362. For example, in at least one embodiment, in response to receiving Logging Errors 350 at North Bound Systems 360, North Bound Systems 360 identify incidents and appropriate tickets are created in response to the incidents along with the identification of the fault and its severity. The ticket identifies the fault and provides other information and is assigned to a developer or other worker for addressing the issue associated with the ticket.

[0069] The method then ends (S450).

[0070] At least one embodiment of a method for providing cloud application fault detection includes receiving, at a processor from at least one log directory associated with at least one server, log files associated with applications running on the at least one server, the log files automatically streamed from the at least one directory, storing in memory by the processor, the log files automatically streamed from the at least one log directory as the log files are received, obtaining, by the processor from a database, a template for identifying faults of the applications associated with the log files automatically streamed from the at least one log directory, applying, by the processor, the template to the log files stored in the memory to identify the faults of the applications associated with the log files, obtaining, by the processor from the database, a configuration file from the database for generating an error message identifying the faults of the applications associated with the log files automatically streamed from the at least one log directory, applying, by the processor, the configuration file to the log files stored in the memory to generate the error message identifying the faults of the applications associated with the log files, and transmitting, by the processor over a network, the error messages to an operations support system for addressing the faults of the applications associated with the log files.

[0071] Fig. 5 is a high-level functional block diagram of a processor-based system 500 according to at least one embodiment.

[0072] In at least one embodiment, processing circuitry 500 provides end-to-end application onboarding. Processing circuitry 500 implements automatic detection of faults in log files steadily received from one or more log directories in a cloud-based system using processor 502. Processing circuitry 500 also includes a non-transitory, computer-readable storage medium 504 that is used to implement automatic detection of faults in log files steadily received from one or more log directories in a cloud-based system. Storage medium 504, amongst other things, is encoded with, i.e., stores, instructions 506, i.e., computer program code that are executed by processor 502 causes processor 502 to perform operations for automatic detection of faults in log files steadily received from one or more log directories in a cloud-based system. Execution of instructions 506 by processor 502 represents (at least in part) a cloud application fault detection system which implements at least a portion of the methods described herein in accordance with one or more embodiments (hereinafter, the noted processes and/or methods).

[0073] Processor 502 is electrically coupled to computer-readable storage medium 504 via a bus 508. Processor 502 is electrically coupled to an Input/output (I/O) interface 510 by bus 508. A network interface 512 is also electrically connected to processor 502 via bus 508. Network interface 512 is connected to a network 514, so that processor 502 and computer-readable storage medium 504 connect to external elements via network 514. Processor 502 is configured to execute instructions 506 encoded in computer-readable storage medium 504 to cause processing circuitry 500 to be usable for performing at least a portion of the processes and/or methods. In one or more embodiments, processor 502 is a Central Processing Unit (CPU), a multi-processor, a distributed processing system, an Application Specific Integrated Circuit (ASIC), and/or a suitable processing unit.

[0074] Processing circuitry 500 includes I/O interface 510. I/O interface 510 is coupled to external circuitry. In one or more embodiments, I/O interface 510 includes a keyboard, keypad, mouse, trackball, trackpad, touchscreen, and/or cursor direction keys for communicating information and commands to processor 502.

[0075] Processing circuitry 500 also includes network interface 512 coupled to processor 502. Network interface 512 allows processing circuitry 500 to communicate with network 514, to which one or more other computer systems are connected. Network interface 512 includes wireless network interfaces such as Bluetooth, Wi-Fi, Worldwide Interoperability for Microwave Access (WiMAX), General Packet Radio Service (GPRS), or Wideband Code Division Multiple Access (WCDMA); or wired network interfaces such as Ethernet, Universal Serial Bus (USB), or Institute of Electrical and Electronics Engineers (IEEE) 564.

[0076] Processing circuitry 500 is configured to receive information through I/O interface 510.

The information received through I/O interface 510 includes one or more of instructions, data, design rules, libraries of cells, and/or other parameters for processing by processor 502. The information is transferred to processor 502 via bus 508. Processing circuitry 500 is configured to receive information related to a User Interface (UI) 522 through I/O interface 510. The information is stored in computer-readable medium 504 as UI 522. A visualization of log file data is presented on Display Device 524 for analysis.

[0077] In one or more embodiments, one or more non-transitory computer-readable storage media 504 having stored thereon instructions (in compressed or uncompressed form) that may be used to program a computer, processor, or other electronic device) to perform processes or methods described herein. The one or more non-transitory computer-readable storage media 504 include one or more of an electronic storage medium, a magnetic storage medium, an optical storage medium, a quantum storage medium, or the like. For example, the computer-readable storage media may include, but are not limited to, hard drives, floppy diskettes, optical disks, read-only memories (ROMs), random access memories (RAMs), erasable programmable ROMs (EPROMs), electrically erasable programmable ROMs (EEPROMs), flash memory, magnetic or optical cards, solid-state memory devices, or other types of physical media suitable for storing electronic instructions. In one or more embodiments using optical disks, the one or more non-transitory computer-readable storage media 504 includes a Compact Disk-Read Only Memory (CD-ROM), a Compact Disk-Read/Write (CD-R/W), and/or a Digital Video Disc (DVD).

[0078] In one or more embodiments, storage medium 504 stores computer program code 506 configured to cause processing circuitry 500 to perform at least a portion of the processes and/or methods for end-to-end application onboarding. In one or more embodiments, storage medium 504 also stores information, such as algorithm which facilitates performing at least a portion of the processes and/or methods for automatic detection of faults in log files steadily received from one or more log directories in a cloud-based system. Accordingly, in at least one embodiment, the processor circuitry 500 performs a method for providing cloud application fault detection that includes at least receiving, at a processor from at least one log directory associated with at least one server, log files associated with applications running on the at least one server, the log files automatically streamed from the at least one directory, storing in memory by the processor, the log files automatically streamed from the at least one log directory as the log files are received, obtaining, by the processor from a database, a template for identifying faults of the applications associated with the log files automatically streamed from the at least one log directory, applying, by the processor, the template to the log

files stored in the memory to identify the faults of the applications associated with the log files, obtaining, by the processor from the database, a configuration file from the database for generating an error message identifying the faults of the applications associated with the log files automatically streamed from the at least one log directory, applying, by the processor, the configuration file to the log files stored in the memory to generate the error message identifying the faults of the applications associated with the log files, and transmitting, by the processor over a network, the error messages to an operations support system for addressing the faults of the applications associated with the log files.

[0079] Advantages include a system that provides quick and easy identification of faults from logs as log files are received. The cloud application fault detection enables events from large applications to be processes. Further, the cloud application fault detection is able to handle log files received from multiple, geographically distributed servers.

[0080] Separate instances of these programs can be executed on or distributed across any number of separate computer systems. Thus, although certain steps have been described as being performed by certain devices, software programs, processes, or entities, this need not be the case. A variety of alternative implementations will be understood by those having ordinary skill in the art.

[0081] Additionally, those having ordinary skill in the art readily recognize that the techniques described above can be utilized in a variety of devices, environments, and situations. Although the embodiments have been described in language specific to structural features or methodological acts, the subject matter defined in the appended claims is not necessarily limited to the specific features or acts described. Rather, the specific features and acts are disclosed as exemplary forms of implementing the claims.

CLAIMS

WHAT IS CLAIMED IS:

1. A method for providing cloud application fault detection, the method comprises:
 - receiving, at a processor from at least one log directory associated with at least one server, log files associated with applications running on the at least one server, the log files automatically streamed from the at least one directory;
 - storing, in memory by the processor, the log files automatically streamed from the at least one log directory as the log files are received;
 - obtaining, by the processor from a database, a template for identifying faults of the applications associated with the log files automatically streamed from the at least one log directory;
 - applying, by the processor, the template to the log files stored in the memory to identify the faults of the applications associated with the log files;
 - obtaining, by the processor from the database, a configuration file from the database for generating an error message identifying the faults of the applications associated with the log files automatically streamed from the at least one log directory;
 - applying, by the processor, the configuration file to the log files stored in the memory to generate the error message identifying the faults of the applications associated with the log files; and
 - transmitting, by the processor over a network, the error messages to an operations support system for addressing the faults of the applications associated with the log files.
2. The method of claim 1, wherein the log files automatically streamed from the at least one log directory are continuously streamed from the at least one log directory and obtained by the processor.
3. The method of claim 1, wherein the applying, by the processor, the template to the log files to identify the faults of the applications associated with the log files further comprises:
 - determining, by the processor based on the template, a value in the log files identifying the fault.

4. The method of claim 1, wherein the receiving, at the processor, the log files further comprises continuously receiving, at the processor, the log files from a log file shipper.

5. The method of claim 1 further comprising:
processing, by the processor, the log files automatically streamed from the at least one log directory as the log files are received, by the processor, to generate deserialized log files;
and
processing, by the processor, the deserialized log files to generate enriched log files having additional information.

6. The method of claim 1, wherein the applying, by the processor, the configuration file to the log files further comprises:
receiving, at the processor, a logging yaml file including patterns and pattern types associated with faults; and
applying, by the processor, the logging yaml file to the log files including at least one of the patterns and at least one of the pattern types to identify a particular error associated with at least one of the faults.

7. The method of claim 1, further comprising:
processing, by the processor, the log files automatically streamed from the at least one log directory as the log files are received to generate un-serialized log files;
storing, by the processor, the un-serialized log files in a storage device; and
performing, by the processor, log analytics on the un-serialized log files to identify information about the log file.

8. The method of claim 1, further comprising:
storing, by the processor, log data associated with the log files in a storage device; and
presenting, by the processor on a display device, a visualization of the log data associated with the log files.

9. A system for providing cloud application fault detection, the system comprising:
- a memory storing computer-readable instructions; and
 - a processor connected to the memory, wherein the processor is configured to execute the computer-readable instructions to:
 - receive, from at least one log directory associated with at least one server, log files associated with applications running on the at least one server, the log files automatically streamed from the at least one directory;
 - store in the memory the log files automatically streamed from the at least one log directory as the log files are received;
 - obtain, from a database, a template for identifying faults of the applications associated with the log files automatically streamed from the at least one log directory;
 - apply the template to the log files stored in the memory to identify the faults of the applications associated with the log files;
 - obtain, from the database, a configuration file from the database for generating an error message identifying the faults of the applications associated with the log files automatically streamed from the at least one log directory;
 - apply the configuration file to the log files stored in the memory to generate the error message identifying the faults of the applications associated with the log files; and
 - transmit, over a network, the error messages to an operations support system for addressing the faults of the applications associated with the log files.
10. The system of claim 9, wherein the processor is further configured to receive continuously streamed log files from the at least one log directory and to determine, based on the template, a value in the log files identifying the fault.
11. The system of claim 9, wherein the processor is further configured to continuously receive the log files from a log file shipper.
12. The system of claim 9, wherein the processor is further configured to:
- process the log files automatically streamed from the at least one log directory as the log files are received to generate deserialized log files; and

process the deserialized log files to generate enriched log files having additional information.

13. The system of claim 9, wherein the processor is further configured to apply the configuration file to the log files by receiving a logging yaml file including patterns and pattern types associated with faults and applying the logging yaml file to the log files including at least one of the patterns and at least one of the pattern types to identify a particular error associated with at least one of the faults.

14. The system of claim 9, wherein the processor is further configured to:
store log data associated with the log files in a storage device; and
present on a display device a visualization of the log data associated with the log files.

15. A non-transitory computer-readable media having computer-readable instructions stored thereon, which when executed by a processor causes the processor to perform operations comprising:

receiving, from at least one log directory associated with at least one server, log files associated with applications running on the at least one server, the log files automatically streamed from the at least one directory;

storing in memory the log files automatically streamed from the at least one log directory as the log files are received;

obtaining, from a database, a template for identifying faults of the applications associated with the log files automatically streamed from the at least one log directory;

applying the template to the log files stored in the memory to identify the faults of the applications associated with the log files;

obtaining, from the database, a configuration file from the database for generating an error message identifying the faults of the applications associated with the log files automatically streamed from the at least one log directory;

applying the configuration file to the log files stored in the memory to generate the error message identifying the faults of the applications associated with the log files; and transmitting, over a network, the error messages to an operations support system for addressing the faults of the applications associated with the log files.

16. The non-transitory computer-readable media of claim 15, wherein the log files automatically streamed from the at least one log directory are continuously streamed from the at least one log directory.

17. The non-transitory computer-readable media of claim 15, wherein the applying the template to the log files to identify the faults of the applications associated with the log files further comprises:

determining, based on the template, a value in the log files identifying the fault.

18. The non-transitory computer-readable media of claim 15, wherein the receiving the log files further comprises continuously receiving the log files from a log file shipper.

19. The non-transitory computer-readable media of claim 15, wherein the applying the configuration file to the log files further comprises:
- receiving a logging yaml file including patterns and pattern types associated with faults; and
 - applying the logging yaml file to the log files including at least one of the patterns and at least one of the pattern types to identify a particular error associated with at least one of the faults.
20. The non-transitory computer-readable media of claim 15, further comprising:
- storing log data associated with the log files in a storage device; and
 - presenting on a display device a visualization of the log data associated with the log files.

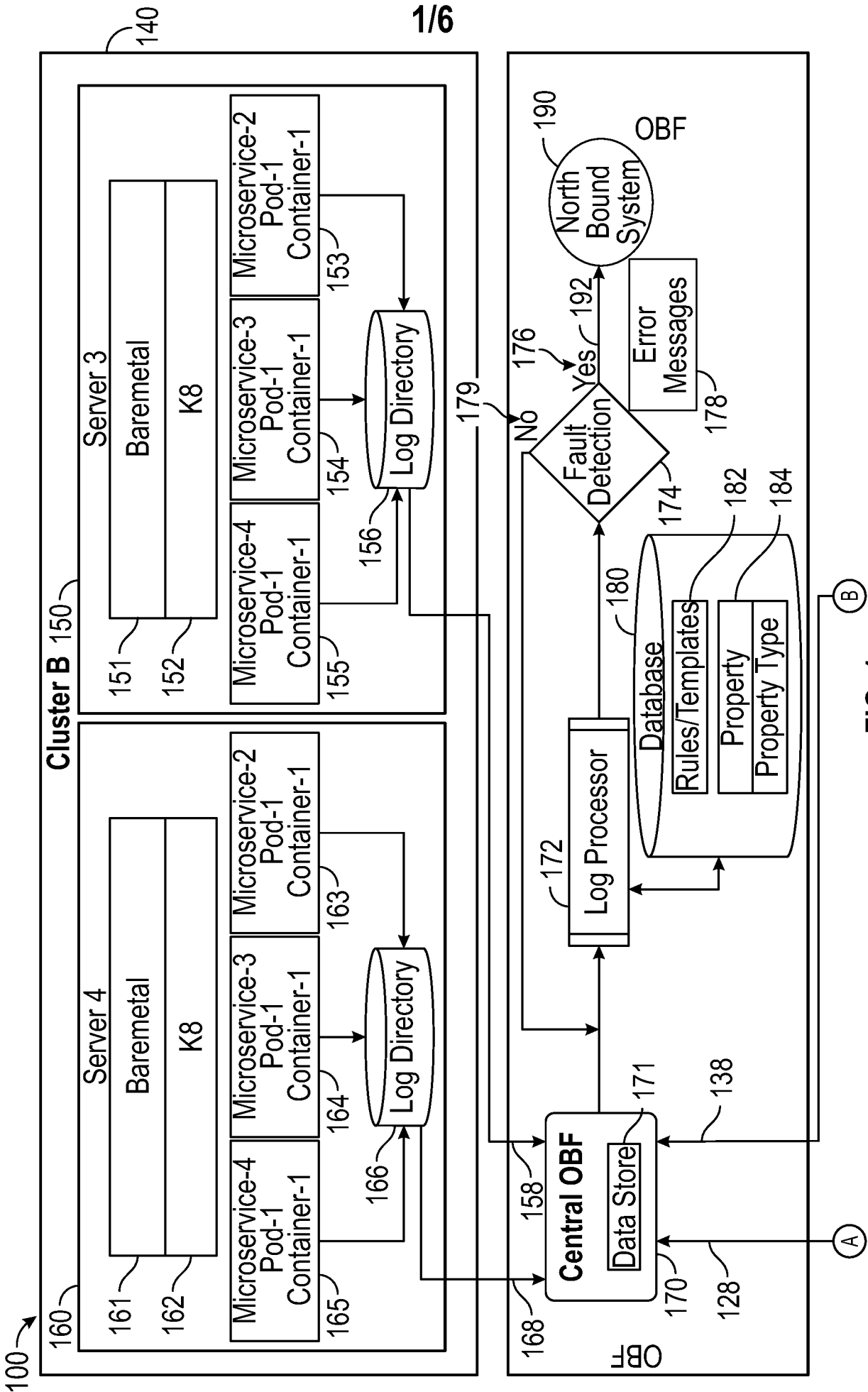


FIG. 1

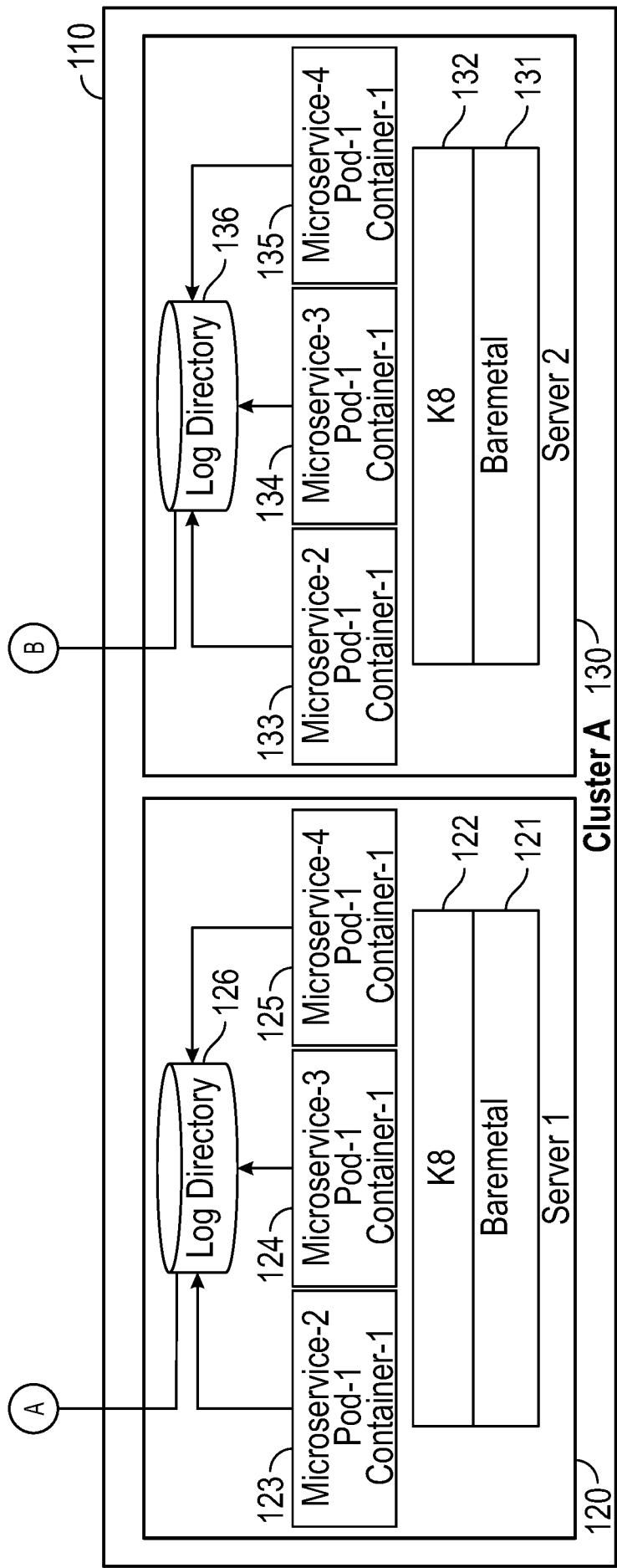


FIG. 1
(Continued)

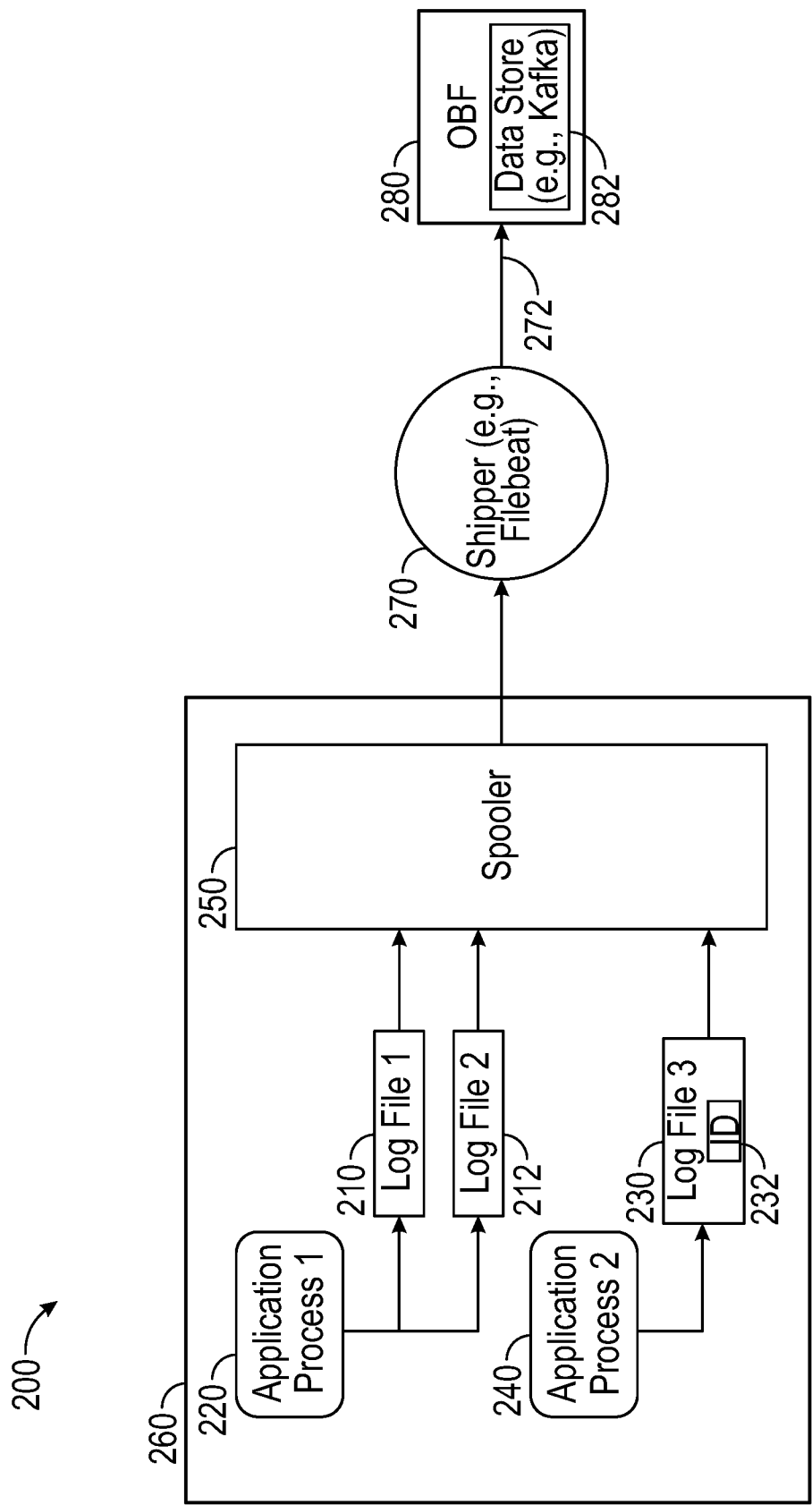


FIG. 2

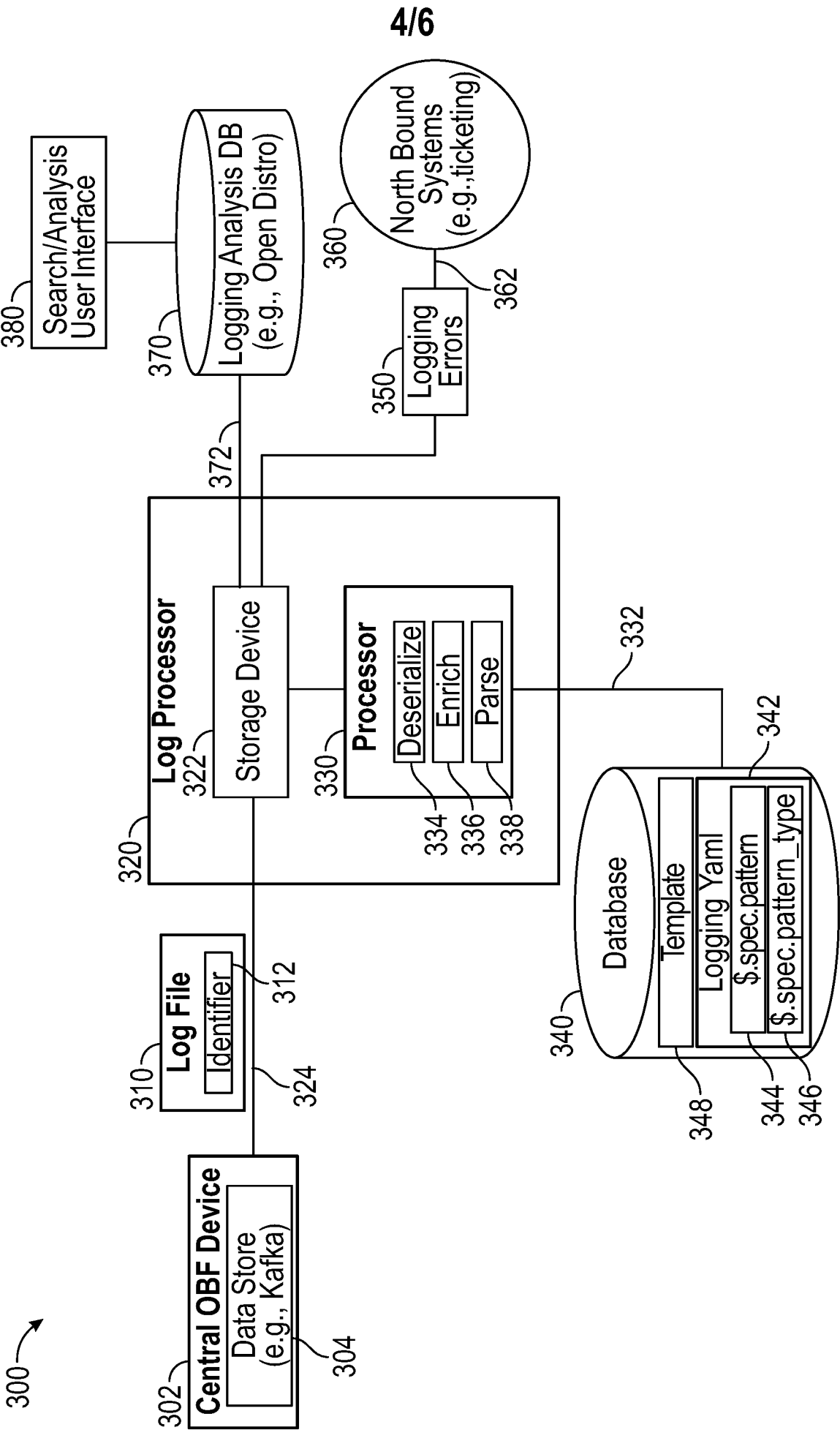


FIG. 3

5/6

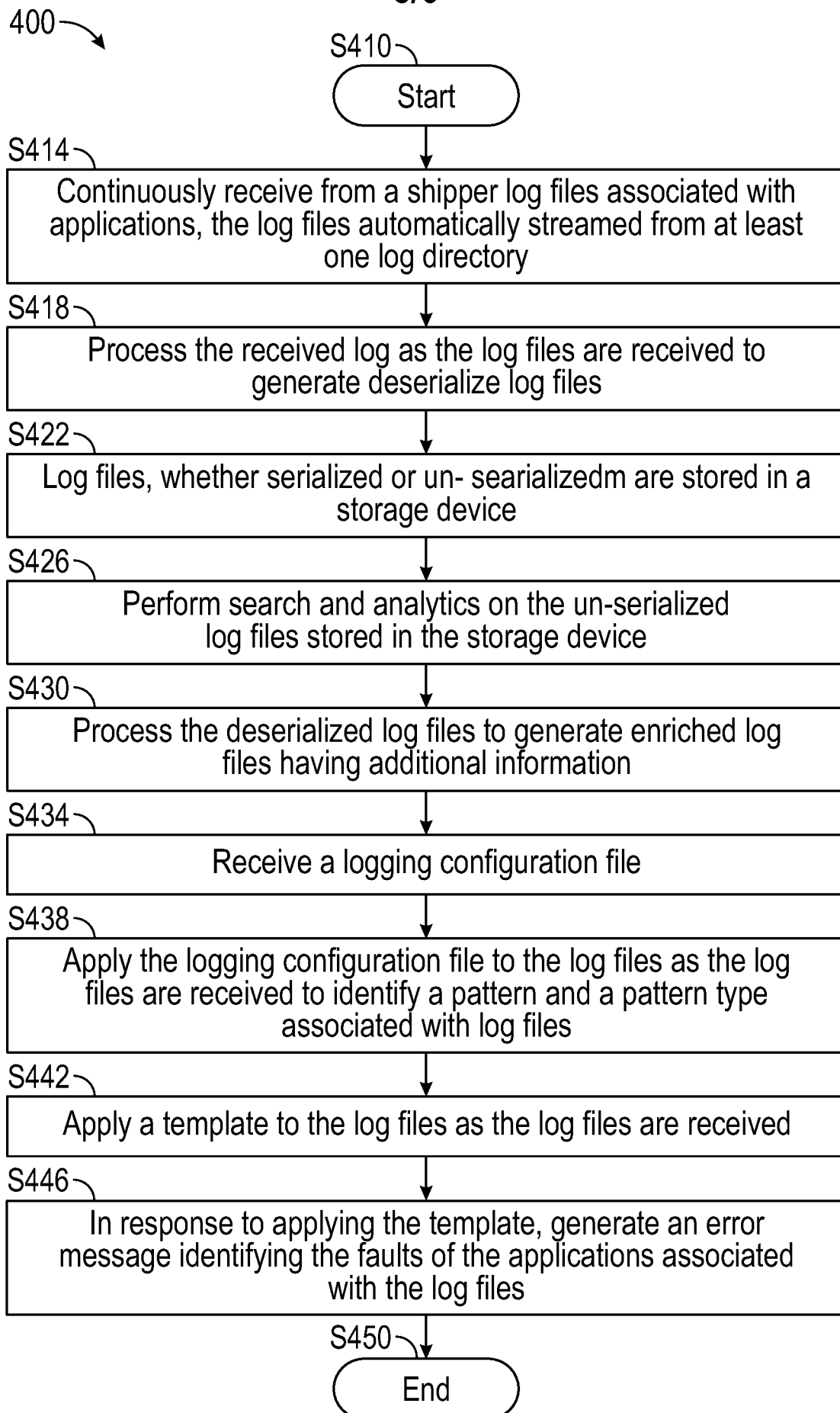


FIG. 4

6/6

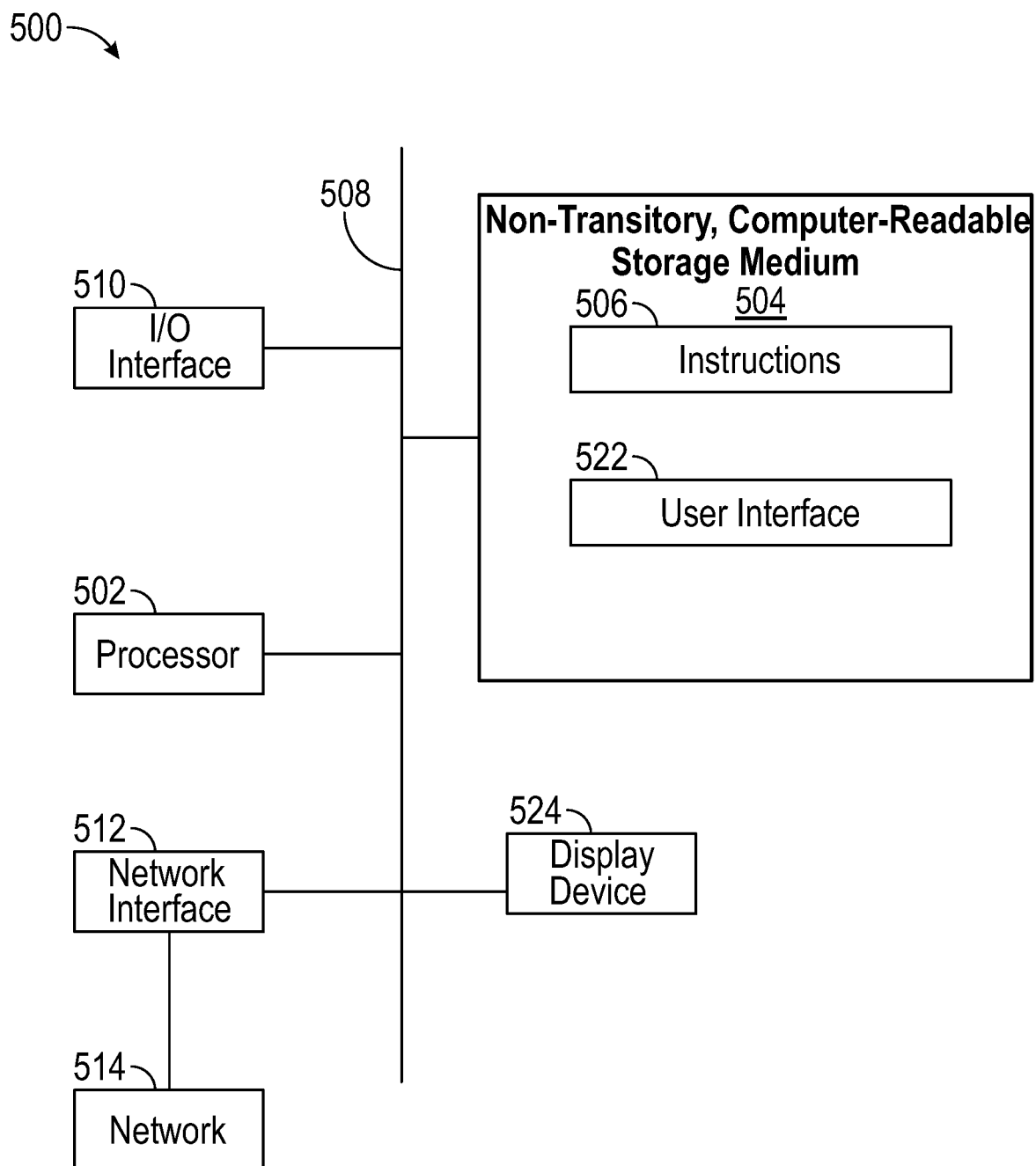


FIG. 5

INTERNATIONAL SEARCH REPORT

International application No.

PCT/US22/39896

A. CLASSIFICATION OF SUBJECT MATTER

IPC - INV. G06F 11/00 (2022.01)
 ADD. H04L 67/00 (2022.01)
 CPC - INV. G06F 11/0703; G06F 11/0775

ADD. H04L 67/00

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
 See Search History document

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched
 See Search History document

Electronic database consulted during the international search (name of database and, where practicable, search terms used)
 See Search History document

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X — Y	US 9,672,085 B2 (ACCENTURE GLOBAL SERVICES LIMITED) 06 June 2017; column 3, lines 10-26; column 4, lines 44-46; column 6, lines 18-40; column 12, lines 64-66	1-3, 5, 8-10, 12, 14-17, 20 — 4, 6-7, 11, 13, 18-19
Y	US 2021/0377156 A1 (VMWARE, INC.) 02 December 2021; paragraph [0414]	4, 11, 18
Y	US 11,025,512 B2 (DRAIOS, INC.) 01 June 2021; column 18, lines 29-30	6, 13, 19
Y	US 7,801,851 B2 (HOLENSTEIN, P ET AL.) 21 September 2010; column 17, lines 23-27	7

☐ Further documents are listed in the continuation of Box C.

☐ See patent family annex.

* Special categories of cited documents:

"A" document defining the general state of the art which is not considered to be of particular relevance

"D" document cited by the applicant in the international application

"E" earlier application or patent but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art

"&" document member of the same patent family

Date of the actual completion of the international search

18 November 2022 (18.11.2022)

Date of mailing of the international search report

DEC 28 2022

Name and mailing address of the ISA/

Mail Stop PCT, Attn: ISA/US, Commissioner for Patents
 P.O. Box 1450, Alexandria, Virginia 22313-1450
 Facsimile No. 571-273-8300

Authorized officer

Shane Thomas

Telephone No. PCT Helpdesk: 571-272-4300