Distributed log management for dynamically changing computing environments on cloud

Takayuki Kushida

School of Computer Science,
Tokyo University of Technology,
1404-1 Katakura, Hachouji,
Tokyo 192-0914, Japan
Email: kushida@acm.org

Abstract: The cloud logging service is a core component for the operation and management of the production system. The service is usually a central server deployment whereby the dedicated central servers accept all log messages from leaf computing nodes. As the number of applications and solutions on the cloud changes dynamically, the amount of log messages that are forwarded to the logging service is also changed. The paper proposes a distributed logging service (DLS) that distributes log messages to multiple leaf computing nodes. No central server is required to manage the logging service. DLS also provides alert notifications, authentication, lifetime management and resilience, which are required for the production system. Results of an evaluation of the emulated environment show that DLS is suitable for use with applications and solutions which are used for production usages.

Keywords: distributed log; logging service; log management; cloud management; distributed hash table; DHT.

Reference to this paper should be made as follows: Kushida, T. (2020) 'Distributed log management for dynamically changing computing environments on cloud', *Int. J. Big Data Intelligence*, Vol. 7, No. 3, pp.157–168.

Biographical notes: Takayuki Kushida is a Professor at the School of Computer Science, Tokyo University of Technology. His research interests include cloud computing, distributed computing, IoT and service enablement. He is an author of many research studies published at international journals, conferences and symposiums. He was the Chaired of several workshops and symposiums, and served as the Chief-in-Editor for *Journal of Information Processing (JIP)*. He has also served IPSJ Senior Auditor which oversees the strategy and operation for the academic society.

This paper is a revised and expanded version of a paper entitled 'Distributed logging service with distributed hash table for cloud' presented at SC2: International Symposium on Cloud and Service Computing, Kaohsiung, Taiwan, 18–21 November 2019.

1 Introduction

Cloud computing has been widely adopted in the IT industry since its ability to imporve IT automations in operation and management of production systems has been recognised. Among the advantages of cloud computing is its ability to make the computing resources required by user's on demand (Mell and Grance, 2009; Liu et al., 2011).

As the demand for cloud computing has changed, so too has the consumption of computing resources are dynamically increased and decreased. In addition, the serverless computing, which can dynamically assign computing resources to application software, is emerging in some cloud-native environments (Jonas et al., 2019). Enterprise applications have also migrated to cloud-native environments with microservices architecture (Balalaie et al., 2016). Therefore, systems management software for cloud computing should be improved to ensure better operation and management for cloud instances.

Logging services are among the core services for systems management. It is usually based on the central server model (CSM) (Sohlich et al., 2014), which consists of several central servers, as illustrated by Figure 1. Logging services of this nature are called central logging service (CLS). All log messages from leaf computing nodes are forwarded to the central servers, which process and store them immediately. Several challenges and issues affect CLS in cloud computing, and it is necessary to address these challenges as CLS is deployed as part of cloud-based production systems. These challenges and issues inculde the following:

1 Continuous backend services: The logging service is one of core systems management services since the production system in enterprises is required to support the continuous operation without any service disruption. Thus, logging services serve applications, solutions and administrators without any downtime so that the application's software status can be viewed at any time using the logging service (Gormley and Tong, 2015). As CLS prevents single point of failure, it usually has high availability (HA) and disaster recovery (DR) for processing both incoming log messages and requests from the administrator.

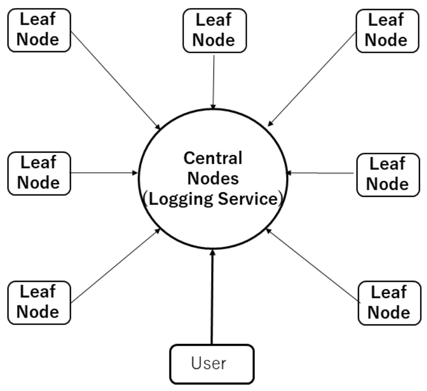
2 Dynamically changing environments: In legacy systems, the resource requirements for the logging service were static since the configurations were fixed and log messages to the central servers could be easily estimated. As cloud computing has been introduced and is a dynamically changing environment in response to users' demands, the allocation of computing resources for the logging service is one of the major challenges faced by the resource planning in systems management. As the large amount of log messages for the logging service changes dynamically, it is often hard to estimate the computing resources that will be required during the planning phase.

3 Dedicated CPU resources: The dedicated CPU resources on central servers are required to process all incoming log messages though the logging service is the backend management component (Lin et al., 2013). The servers for the logging service must categorise and index incoming log messages, which are stored in the storage space. Therefore, sufficient CPU power to process incoming log messages on time is assigned to the central servers.

4 Large amount of dedicated storage space: Dedicated storage space are required to store large amount of log messages (Yue et al., 2010), for example, if the amount of log messages is approximately 100 GB per day and they are forwarded to CLS from all leaf computing nodes. The logging service has at least the 100GB storage space for log messages, and they are kept for one day only. If the retention period for log messages is one week, at least approximately 700 GB of storage space is required.

To address these issues and challenges are solved, this paper proposes a distributed logging service (DLS). The proposed architecture adopts the distributed server model (DSM) to process log messages from leaf computing nodes although the legacy CLS adopted the CSM. Figure 2 shows a DSM that has multiple servers at distributed locations. DLS includes distributed hash table (DHT) from which log messages are distributed to multiple nodes in key/value pairs. Those log messages can be retrieved with the key. DLS also has several functions, such as alert, authentication, lifetime management and resilience, which are required for the production system. The remainder of this paper is structured as follows. The section entitled 'related studies' describes the related fields of central log systems management for logging services and DHT studies. The section entitled 'requirements and use case scenarios' describes a set of requirements for the logging service and use case scenarios for typical usages of log messages. The section entitled 'architecture' describes CSM/DSM, DHT, the process of log messages and DLS architecture. The section entitled 'design for production systems' describes the required functions and the DLS design for the production system. The 'evaluation' section describes the evaluation of the emulated environment and its results. The 'discussions' section describes the challenges faced in completing this study. Finally, the 'conclusion' presents a summary of the study and concluding remarks.

Figure 1 Central server model



2 Related studies

This section describes related studies for central log management systems and DHT.

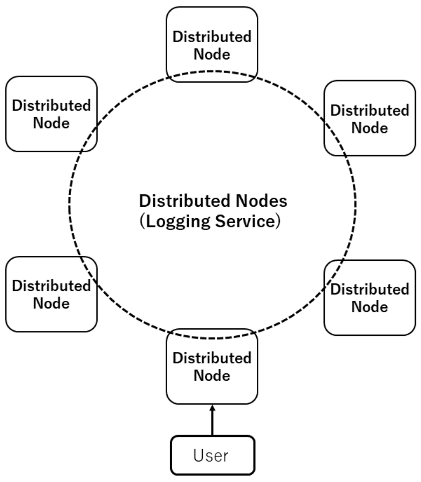
2.1 Central log management systems

The central log management system is used to deploy the log management system on the central location in Figure 1. Leaf computing nodes send log messages to the central log management system. Apache Hadoop and Spark can process large amounts of data on the central servers (Shvachko et al., 2010; Zaharia et al., 2010).

The best known software product for log management is elastic search which is based on the central key/value database (Gormley and Tong, 2015). The architecture design for elastic search is a central server that has a central database to search for log messages. This CSM can usually be applied to production systems of both cloud and on-premise data centres. Elastic search, Logstash and Kibana (ELK) are a typical log management software suite for central log management systems (Bagnasco et al., 2015a, 2015b). The ELK suite collects all log messages from leaf computing nodes since log agents on leaf computing nodes forward to elastic search. All search and analysis processes for the logging service are performed on the central servers. Therefore, they need numerous dedicated CPUs and storage space on the central servers.

LogDNA is a cloud service provider for logging services. After the LogDNA agents have been installed on leaf computing nodes, those log messages on leaf computing nodes are forwarded to the LogDNA service which has the central servers with storage. Users can subscribe to LogDNA's logging services and install the agent to forward them. The operational dashboard from LogDNA is provided to search, analyse and report those log messages (LogDNA, 2019). The service is also suitable for Kubernetes containers to collect log messages for the operation and management of cloud-native environments. LogDNA is a central architecture model for logging services. Therefore, it also needs numerous CPUs and storage space for service on the central servers.

Figure 2 Distributed server model



The design of the monitoring system called private cloud monitoring system (PCMONS) was proposed for private cloud monitoring with its implementation. The study describes a use case scenario using the architecture and the application (Chaves et al., 2011). CSM receives all monitoring data on the central server.

An integrated management system for log messages from distributed sensors was proposed as well as servers and network devices (Ikebe and Yoshida, 2013). They also proposes a cross-processing system for several kinds of log messages. The system aims to provide a flexible method to provide access to distributed log messages using log attributes and values. Although it proposes distributed management for log messages, the prototype of the distributed processing system is implemented using the local and the central fluent server, which manages the log messages. Therefore, its architecture is still the central management model.

Log messages are managed in Kubernetes containers and are forwarded through stdout/stderr to outside the container environment. The installed logging-agent PoD in Kubernetes can forward log messages to the logging services (Xu et al., 2017). Therefore, all log messages can be processed on CLS. Containers on Kubernetes environments are considerably smaller than typical VMs. Once many containers have been created in the environments, they generate numerous log messages which will generate a large amount of data.

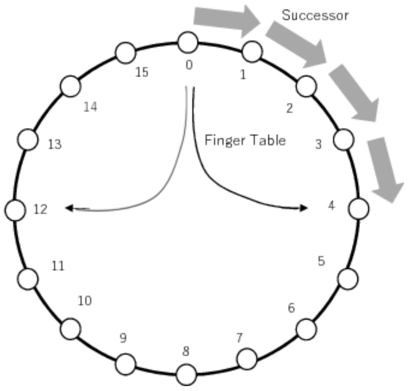
Astrolabe was proposed as a robust and scalable technology for distributed monitoring, management and data mining (Van Renesse et al., 2003). It collects a large-scale system state, permitting rapid updates and providing on-the-fly attribute aggregation. A distributed management architecture and its system performance was evaluated in this study (Jiang et al., 2016). The basic idea is to introduce multiple cell heads in front of the management server. The management workload can be managed on those cell heads. Although distributed management was proposed, it is a limited distributed management since it has a three-level zone tree and monitoring data is aggregated to top-level root zone.

Since those related studies are a related to CLSs, their findings and implementations for logging services can be referred to for the proposed architecture and design. One drawback to the centralised deployment model for those logging services is its requirement of resource assignments. CSM for the logging service requires sufficient computing resources such as CPU, memory and disks since the central server processes many log messages, which are forwarded by leaf computing nodes. Thus, elastic search can have scalability with multiple servers as a server cluster.

If copious amounts of log messages is flowing into the central servers, the central servers will overflow with the incoming log messages. In addition, the central servers should be robust with HA and DR without any service disruption or data loss. Since the log management system is a critical backend service for IT operation and management, those central servers must maintain consistent functionality to provide the logging service.

Although cloud logging services are critical for the operation and management, the service is merely a backend operation function. After the logging service is deployed in the cloud environments, service owners are required to set up the log agent, which collects and forwards log messages for their applications to the central log server.

Figure 3 DHT – Chord



2.2 Distributed hash table

Chord is a service used to implement DHT (Stoica et al., 2003). It is used to support distributed contents management (Rhea et al., 2004). Chord supports just one operation: given a key, it maps the key onto a node. Data location can be easily implemented using Chord by associating the key with each data item, and storing the key/value pair at the node to which the key maps (Stoica et al., 2003). It provides load balance, decentralisation, scalability and availability. In addition, Chord has several characteristics that support distributed operation:

1 joins and stabilisations for nodes

2 impact of node joins on lookups

3 failure and replication

4 voluntary node departures.

It has consistency caching, which maps index to data contents for fault tolerance (Karger et al., 1997). As Chord offers several advantages, DLS could adopt Chord DHT to manage log messages on distributed servers. Each DLS node has the DHT function that generates log messages.

Cassandra is another database that facilitates DHS implementation (Lakshman and Malik, 2010). It is a one-hop DHT that maintains consistent tunable trade-offs between consistency and latency. In addition, it is more than a simple DHT because the values are not opaque, but they are structured into columns and column families, which are indexed in Cassandra.

Dabek et al. (2004) evaluated the performance of DHash++ implementation. In this study, measurements of 425 server instances running hosts show that the latency optimisations for DHT++ can reduce the processing time by a factor of two. The processing time is required to locate and fetch data. In addition, the throughput optimisations result in a sustainable bulk read throughput, which is related to the number of DHT hosts times the capacity of the slowest access link. Since the results of performance evaluation can be referred for DHT++ implementation to the performance aspect of DHT, our study focuses on data center servers, which are connected via high-speed network links. Therefore, no slower access link is present among DHT nodes.

UsenetDHT is a service that aims to reduce the total storage space dedicated to Usenet by storing all articles in a shared DHT (Sit et al., 2008). This study describes the design and implementation of UsenetDHT. It allows a set of cooperating sites to maintain shared, distributed and copied articles. It uses DHT, which provides shared storage for those Usenet articles across the sites. The present study utilises shared storages for those articles with DHT.

The public DHT service that runs on PlanetLab was proposed to support application based on DHT (Rhea et al., 2005). The management of storage allocation for untrusted clients and an interface allowing access to DHT were provided. The paper describes two additional interfaces: lookup in DHT and join outside of DHT, to support OpenDHT. OpenDHT can support multiple separate applications with identifiers for each application. In addition, it supports fair storage allocation among DHT nodes. Although OpenDHT support multiple applications, it is unclear whether it can support log messages and co-exist with applications on the same computing node.

Amazon Dynamo is a highly available key-store database (DeCandia et al., 2007). It is provided as platform as a service (PaaS) from Amazon Web Service. As it is characterised as a zero-hop DHT, each node maintains enough routing information locally to route a request directly to the appropriate node. Bigtable is a distributed storage system for managing structured data that is designed to scale to a very large size (Chang et al., 2008). It is a central service for end users and good evidence for scalable DHT implementation on the local server.

Although several studies have examined DHT, no study has directly investigated logging services in relation to DHT. The present study proposes the logging service with DHT and aims to develop a production system so several management functions are added to DLS.

3 Requirements and use case scenarios

This section describes the requirements for the logging service and use case scenarios to which the logging service can be applied.

3.1 Requirements

Two major functions are required for the logging service in general. They are the 'store' and 'retrieve' functions for those log messages. The 'store' function simply stores them into the storage without any significant delay. The 'retrieve' function is to get them with the specified condition of the search. In addition, the logging service meets the following requirements.

1 The cloud's dynamically changing environment: The applications and solutions must follow the dynamically changing computing environment on cloud. It means that the sum of incoming log messages from leaf computing nodes often has a large amount of data for the logging service. It must be accepted and processed without any delay.

2 Minimised dedicated CPU and storage resources: The consumption of dedicated CPU and storage resources is generally minimised. The logging service requires many dedicated CPU resources to create the index and search for the specific log message in the repository. In addition, the same amount of dedicated storage required to keep those log messages is also required for future search and retrieval. Efficient resource management for both CPU and storage resources should be considered as their minimisation assists with cost reduction.

3 Critical management service: Although the logging service is only one systems management functions, it is a critical backend service for the production system because the logging service can recall any past event with the set of log messages. The systems management function can identify the failure event and the unusual message in log messages and understand what occurred in the past.

3.2 Use case scenarios

The logging service has the following use case scenarios.

3.2.1 Check of log messages

Figure 6 shows an example of log messages generated by the web server during the operation phases. These log messages include forbidden and error messages with date, time and IP address. Other log messages are similar in format and are stored by the logging service.

As those log messages are stored in the repository, the search and retrieval process retrieve the specific log message from the repository. The administrator can search and retrieve it for further analysis of the infrastructure issues using the operation dashboard. In addition, those log messages can be used for different kinds of activities in the development, test and operation phases.

When the specified server on the production system generates an error like the log message in Figure 6, the administrator launches an investigation to quickly understand two aspects:

1 Whether the current status of the specified server down, up or unknown. The current status can be investigated with the monitoring and logging service.

2 When the specified server encountered an error or was down. The log message is useful for investigating such past events

3.2.2 Alert notification

When a failure has occurred in the operation, log messages include the failure event. Those events are linked to generate the alert notification with the log message. Alert notifications are usually linked to system/service failure, an unusual system status or security incidents. When the alert notification is generated with log messages, operators and administrators will review the specific log message for the alert notification and begin to execute the recovery process for the alert to resolve the incident.

3.2.3 Root cause analysis

As the alert notification is received, the administrator begins to address the issue and recovers the system immediately. In addition, the administrator checks the log messages to determine the root cause for the issue. It is a root cause analysis based on log messages.

For example, an error event indicating the disk is full will occur due to the large amount of log messages stored locally. If it occurred at 11 PM yesterday, the event may have been detected and notified to administrator for the further attention. The notification is issued swiftly after the even has been detected. The administrator looks at the log message on the operation dashboard and checks the failure log message to ascertain whether the disk is full. Based on an analysis of the log messages, the administrator understands that the full disk is the root cause.

3.2.4 Development and test phases

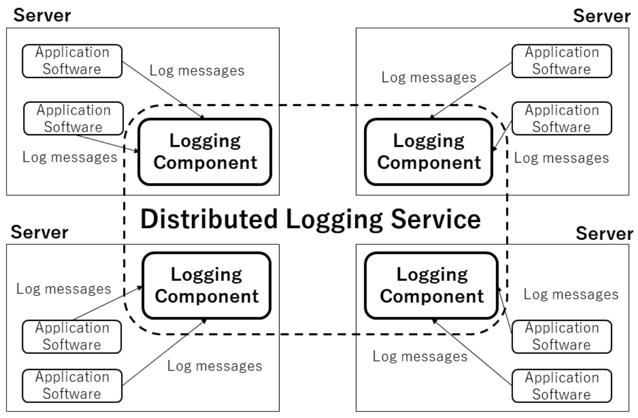
The logging service is used for multiple purposes. In the development phase, the logging service provides log messages for debugging the software components. Development engineers check those log messages to test and debug to the purpose of updating their developed program codes for their software components.

The logging service can also be used for program testing during the system test phase. Test engineers usually check those log messages to verify that the functions have worked correctly. The activities for the test phase differ from those for the development phase since test scenarios and cases are executed to check those functions. Therefore, the logging service is suitable for both the development and test phases.

3.2.5 Security incident and event management

Log messages are also used for security incident and event management (SIEM) (Yen et al., 2013). If unusual access to the computer is detected, the specific log message shows the access. It can be detected in SIEM with log messages. A compliance audit is also required to store the past audit activity events (Murugesan and Ray, 2014). For example, log messages for user access are stored when the audit record is required as a past event.

Figure 4 DLS architecture



4 Architecture

This section describes CSM/DSM, DHT and the architecture for the logging service illustrated in Figure 4.

4.1 Central and distributed server model

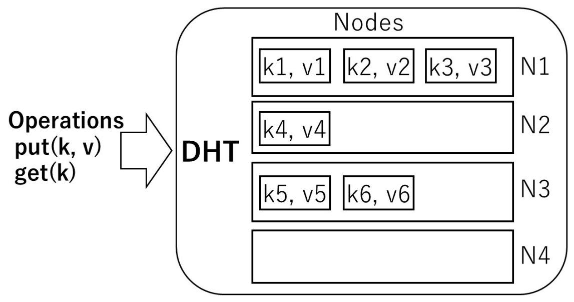
Two models exist for logging services: the CSM and the DSM.

4.1.1 Central server model

Figure 1 shows the CSM, which accepts and processes all log messages on the central servers. 'Leaf nodes' in Figure 1 send log messages to 'central nodes', which are used for the logging service. Once log messages are received by the logging service on the addspan central nodes in Figure 1, they are processed and indexed. The user can search for and retrieve the log messages through the dashboard on the logging service in Figure 1.

The operation and management components, including the logging service, are usually CSM, which receives log messages from leaf computing nodes. When a large number of logging messages is forwarded to the central servers, numerous CPU and storage resources are required to process and store them without any delay or bottleneck. In addition, as a peak log message transactions is supported, the unused CPU and storage resources is always reserved for the logging service on cloud. The systems management function is generally a kind of overhead function for the core services, even when it is a mandatory requirement for the development, test and operation phases. Therefore, the dedicated resources for the log service are minimised and the utilisation of the servers is increased.

Figure 5 DHT operations



4.1.2 Distributed server model

Figure 2 shows a DSM for the logging services and all servers for the logging service are distributed. The DSM can process log messages on the distributed nodes in Figure 2. Therefore, the workload for processing can be assigned to distributed nodes instead of the large central servers. If there are overheads for DSM, the feasibility of the logging service is investigated with DSM. The present study investigates it for the production systems.

DSM can be adopted for logging services as it remedies several drawbacks of the CSM (Tanenbaum and Van Steen, 2007). The log messages are forwarded and stored on distributed nodes, which are based on the DSM. The DSM provides the processing power for log messages on the distributed nodes, and it has no central server to store log messages or search for specific log messages. It provides them with the distributed servers.

The management server for the logging service and the access point that supports application program interface (API) runs on distributed servers. They can act as a part of the logging service rather than being assigned to the dedicated central server that processes the incoming requests for the logging service.

4.2 DLS architecture

The architecture decision for the DSM has been done with the investigations. Figure 4 shows the architecture of DLS. Each server has the processes for 'application software' and 'logging component'. 'Application software' provides the solution service to end users. It generates numerous log messages, which are usually stored in local files. The logging component for DLS is deployed on the same server on which the application software is running. It retrieves local log messages and forwards them to 'logging component' on the local server. It is interconnected with the other 'logging component' on the different servers in Figure 4, and they are a part of the DLS.

4.3 Distributed hash table

DHT is a solid technology that can share large amounts of data with robustness and availability among distributed nodes, illustrated by Figure 5. DHT operations in Figure 5 are the put (k, v) operation to store data (v) with key (k) and the get (k) operation to retrieve data with key (k). The operation get (k) returns data (v) when they are found on DHT. DHT manages the hash table on the distributed nodes. Data entries are stored in the distributed nodes with the hash table. Figure 5 shows an example of DHT that consists of four nodes that have $(k1, v1), (k2, v2), (k3, v3)$ in the first node, $(k4, v4)$ in the second node and $(k5, v5), (k6, v6)$ in the third node.

There are two major advantages in DHT:

1 Join/leave: Resilience is required to join and leave networks. DHT can support the network link changes for the nodes.

2 Data loss: Hashed data can be automatically distributed to DHT nodes. Since stored data are replicated across nodes, data loss as a result of lost nodes can be avoided. The removal of any single node has no impact at all.

On the other hand, DHT has the following limitations:

1 Triggers and events: The function for events and triggers is not supported.

2 Data consistency and integrity: DHT does not provide absolute guarantees on the consistency and integrity of data.

3 Group queries: DHT does not have efficient group queries, range queries or other data lookups.

4 No central authority: No central authority is present in the configuration setup. The multiple nodes must cooperate when decisions need to be made.

5 Longer retrieval time: The retrieval time for looking up log messages is $O(\log N)$. It takes a few seconds depending on the specified location, the number of nodes and the latencies among those nodes.

If these limitations can be managed, DHT technology can be applied to the logging service to realise DLS.

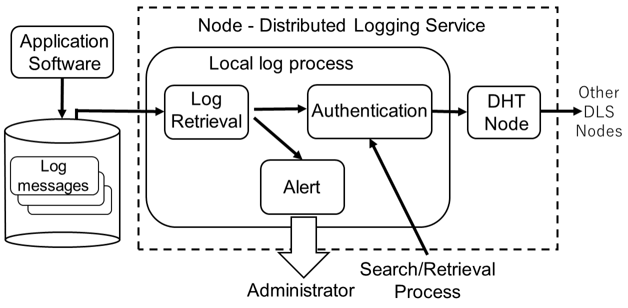
4.4 Log messages with DHT

As DHT can process log messages on distributed nodes, each node can store and retrieve them without the central server. Every node generates log messages, which are stored into local log files at first, and the format of those log messages is shown in Figure 6. The local agent forwards those log messages to DHT in the distributed logging service. Consequently, those log messages are stored into the table in DHT.

Figure 6 Example of log messages

```
2019/05/24 22:08:22 [crit] 24841#24841: \
 *5444 SSL_do_handshake() failed \
 (SSL: error:1420918C:SSL \
 routines:\
 tls_early_post_process_client_hello\
 :version too low) while SSL handshaking, \
 client: 192.168.1.1, server: 0.0.0.0:443
2019/05/25 12:14:40 [error] 24841#24841: \
 *5664 access forbidden by rule, client: \
 192.168.1.1, server: test, request: \
 "GET /", host: "server", referrer: \
 "https://server/documents/"
```

Figure 7 DLS local process



When the log message is stored in the DHT, key/value pairs are configured for the log messages. The key is the timestamp at which the log message is generated, and the value can be the log message itself.

The hash table is provided to generate keys and values. Both keys and values are defined for DHT. Figure 8 shows the process by which the log message timestamp and contents are stored as the key and value for DLS. In Figure 8, log messages are divided into timestamps and contents. Keys are timestamps, and values are log message contents. This distinction can reduce the size of the DHT. It separates between the index for the search and the actual message contents for the retrieval.

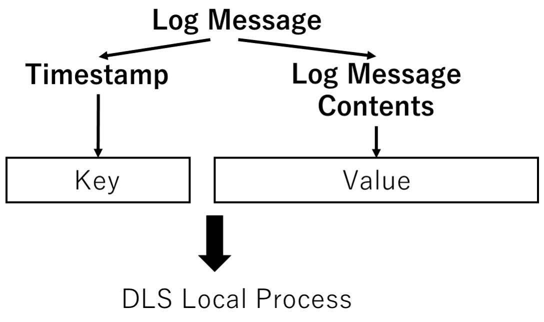
Figure 3 shows that DHT manages logging messages for DLS. Once the query for DHT has been sent to find log messages in DHT (Komarek et al., 2018), it can be retrieved for the search results in DHT since DHT stores those log messages.

5 Design for production systems

This section describes the DLS design for production systems.

Figure 7 shows the process by which the local host becomes one of DHT nodes. 'Application software' in Figure 7 stores their log messages in the local file. 'Log retrieval' in Figure 7 retrieves log messages from the log file and selects the key/value pairs for log messages. It forwards them to 'authentication' and 'alert' as shown in Figure 7. The local DLS has a DHT node, which can accept log messages that are locally generated. The 'DHT node' interacts with 'other DLS nodes', which includes the DHT Node as shown in Figure 7 so DHT can share those log messages among the DHT nodes.

Figure 8 Key and value for DLS



5.1 Alert with log messages

Figure 9 shows the alert component for log messages. Log messages (#1, #2, ..., #n) in Figure 9 are forwarded to 'alert process'. It checks log messages with 'common rules' and 'local rules' to determine whether the alert notification has been generated, as shown in Figure 9. If the specific log message is matched with one of those alert rules, the alert notification is forwarded to another process, such as e-mail, SMS or Web notification on the dashboard. The administrator can detect the alert notification and will fix the issue for the specific component that the alert notification has highlighted. As illustrated by Figure 9, 'common rules' are common rules for log messages among DLS nodes, and 'local rules' are only applied to the local software component to check its issues.

5.2 Authentication for access

Figure 10 shows the local authentication processes. Log messages are checked by the authentication process whether they can be stored. Since DHT contains the hash table to store and retrieve the value with the key, the key/value pair is deployed to multiple DHT nodes. Since different administrators and processes have access to DLS, the local authentication for the process is required to have a valid access to DLS.

Authentication is required to store the log messages that come from 'log retrieval' as shown in Figure 7. When log messages are submitted to DLS, like log message (#2) in Figure 10, 'authentication process' checks the valid access to DLS with the 'ID/password' database in Figure 10. Once it has been authenticated, the log message is forwarded to the 'DHT node' as the log message (#1) in Figure 10.

Another mode of authentication is to search and retrieve the log message from other local software. They are 'search and retrieve' requests as shown in Figure 10. Therefore, the API access to retrieve the log message from local DHT is granted with the authentication. After the request has been validated, the log message can be retrieved with the key as the log message (#3) as shown in Figure 10.

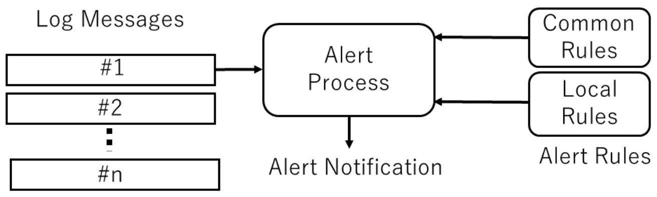
Another authentication method is hash-based distributed authentication method (HDAM). It realises a decentralised efficient mutual authentication mechanism for each pair of nodes (Takeda et al., 2008). As the cloud authentication service can be relied on, the usual API authentication is adopted in this paper.

5.3 Lifetime for log messages

Log messages from leaf computing nodes are locally stored in the local DLS. Since the log messages are created with various kinds of activities in those leaf computing nodes and forwarded to the local DLS, the required storage size on the local DLS increases rapidly. Therefore, lifetime management is required to manage the stored log messages. The expiration time is added to each log message. In addition, the expiration time is often updated since the cloud is a dynamically changing environment and the amount of log messages increases and decreases.

When the log message has reached its expiration, it is simply moved from DHT to the inexpensive storage. Later, they are permanently removed. Consequently, the required size in the local storage can be controlled with the lifetime management.

Figure 9 Local alert process



5.4 Resilience

The availability of DLS is also required by the production service since a continuous service is required to provide for applications and solutions. The staging and production environments usually have HA and DR. They can support continuous services for applications, solutions and administrators. Duplicated computing resources are usually required at different locations.

5.4.1 HA

In the legacy production system, the logging service usually requires HA as a non-function on the different physical hardware in the same data centre. Twin servers must adopt the same setup but one hardware with an independent network connection and a power source from another hardware.

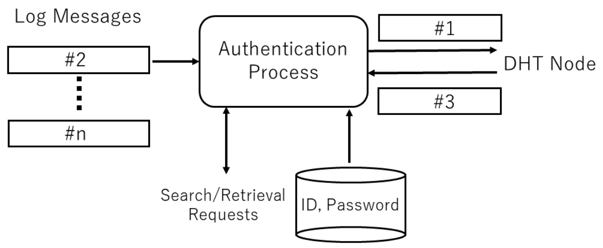
When the distributed logging service is deployed as a production service, it supports the continuous service in case of failure of the distributed node (Tanenbaum and Van Steen, 2007). DLS has HA since DHT Chord has the successor lists on those DHT nodes (Stoica et al., 2003). The list supports the connection of multiple alternative DHT nodes if the successor node is unreachable.

5.4.2 DR

In addition to the native DHT function to support the HA, the duplicated systems are used for each service. For example, when DLS receives the log message, it is duplicated for a primary and backup node. One log message can be stored in the primary node, and the same log message is forwarded to the backup node. Both the primary and backup nodes on DLS are deployed to the isolated underlying resources. They are deployed to two physical locations in different countries.

As applications and solutions have the DR site at another data centre. DLS is also deployed to the same configuration as the primary data centre. As the logging service has log messages for operation and systems management, there is no need to copy those log messages to the data centre in the DR site since the backup data centres has an entirely different operation management from the primary data centre.

Figure 10 Local authentication process



6 Evaluation

This section describes the evaluation of DLS which has a DHT to store and retrieve log messages of distributed nodes. The performance test on the emulated environment was conducted to evaluate the DLS deployment. The scalability for the retrieval of log messages, the stored log messages and the number of nodes is examined on the emulated environment.

6.1 Emulation toolkit for evaluation

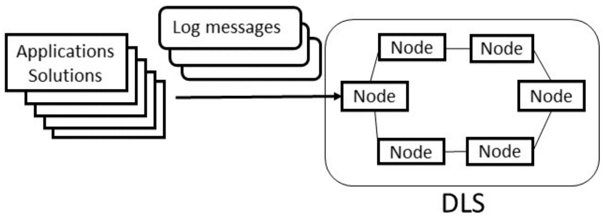
The emulated environment was constructed using the DHT emulator called Overlay Weaver (Shudo et al., 2006, 2008). It facilitates the implementation of routing algorithms and runs with multiple well-known DHT algorithms that can be evaluated with hundreds of code lines (Shudo, 2006).

The toolkit also provides a common API that allows access to high-level components including DHT and multicast functions. It is implemented using Java codes. It supports multiple DHT algorithms, such as Chord, Kademlia, Koorde, Pastry and Tapestry. Those DHT algorithms and their routing methods have been studied in earlier research (Stoica et al., 2003).

DLS is evaluated with Chord, which a DHT algorithm. It is also a well-known algorithm and the emulated environment can be constructed using the Overlay Weaver toolkit. As it is used for the DHT algorithm in the evaluation, the iterative routing is applied to DHT. Chord is also a basic algorithm and routing method; therefore, the evaluation could focus on the capability of DLS for the production system.

The connections among DHT nodes may be the loopback network interfaces that avoid the network overhead for the interactions among DHT nodes. Therefore, the essential processing capability could be evaluated. The performance of DHT can also be measured using the random retrieval data set based on the number of stored messages. The emulated environment runs on virtual machine (memory 4 GB, CPU 2 cores) on which Ubuntu distribution is installed.

Figure 11 DLS emulation setup



6.2 Evaluation setup

Figure 11 shows the emulation setup for the DLS evaluation to process incoming log messages. This evaluation has adopted use case scenarios with configurations on the Overlay Weaver toolkit. In Figure 11, the leaf computing nodes generate log messages and forward them to the local logging component in DLS which consists of multiple distributed nodes. Each node processes incoming log messages on the same nodes. A DLS node processes them, as shown in Figure 11. As those nodes are located at distributed locations, the processing for incoming log messages can be distributed to other nodes.

The basic functions for the logging service are ‘store’ and ‘retrieve’ for those log messages. The ‘store’ function is to store those incoming log messages into the backend storage, which is DHT in the proposed method. The process for the ‘store’ function can be sequential, and the submissions for log messages have different time. It may be the batch process and is not related to user experience. On the other hand, the ‘retrieve’ function can be evaluated for the DLS performance. It takes some time to retrieve the specified log message from the many log messages since many log messages are stored in the DLS. The performance evaluations are to measure the retrieval time with several condition parameters.

Figure 12 Processing time for random retrieval (fixed 1,000 records) (see online version for colours)

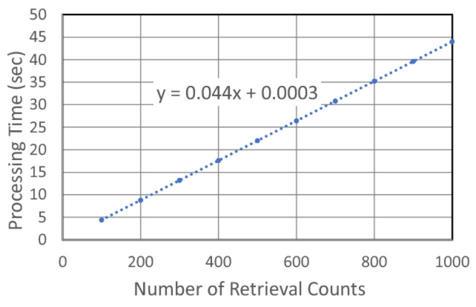
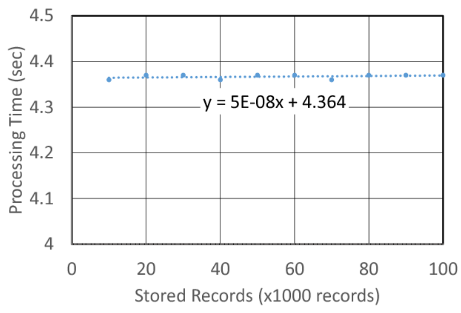


Figure 13 Processing time for stored records (fixed 100 random retrievals) (see online version for colours)



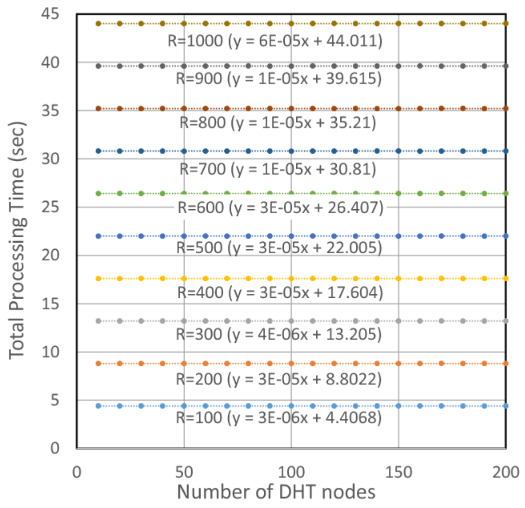
6.3 Evaluation results

- *Processing time for retrieval counts:* Figure 12 shows the processing times required to retrieve the log message for the count of retrieval records (log messages) that are stored in DLS. The number of DHT nodes is fixed at 100 for the evaluation as shown in Figure 12. As the retrieved counts are changed from 100 to 1,000, the total processing times is evaluated for the retrieval of log messages. In this evaluation, the total processing time has a linear relation with the number of retrieval counts when the stored records and the number of nodes are fixed. The value $0.044x$ with the linear fitting means that the retrieval is approximately 44 (msec) per record on the emulated environment as shown in Figure 12. The processing time y intercept 0.0003 (sec) = 0.3 (msec) in Figure 12 may be negligible since the precision of the measurement time is approximately 1 (msec), which is greater than 0.3 (msec).

- *Processing time for stored records:* Figure 13 shows the total processing times for the number of stored records. The number of stored records for log messages ranges from 10 K to 100 K, and the processing times have been always flat for the number of stored records. The evaluation is performed when the retrieval count is fixed at 100. In the evaluation, the processing time is flat for the number of stored records. In Figure 13, the y intercept 4.364 (sec) for the linear fitting is shown for 100 retrieval counts. Therefore, it can be estimated at 43.64 (msec) per retrieval, and the value in Figure 13 is nearly equal to 44 (msec) per record in Figure 12.

  The processing times can be recognised with a flat line since the slope x is $5 * 10^{-8}$ (sec) in Figure 13. Its value may be negligible since the precision of the measurement time is approximately 1 (msec), which means an order of 10^{-3} (sec).

Figure 14 Processing time for DHT nodes (fixed 10,000 records) (see online version for colours)



- *Processing time for number of nodes:* As the number of DHT nodes is increased, the capability to process incoming log messages is evaluated based on the change in the number of DHT nodes. For example, its capability to retrieve the log messages is assessed. The overhead of the processing on DHT nodes is evaluated. Figure 14 shows the processing time for the number of DHT nodes. The x-axis is the number of DHT nodes from 1 to 200, and the y-axis is the total processing time for the retrieval of records. The stored records are changed from R = 100 (records) to R = 1,000 (records) in each graph plot. The key for the retrieval is randomly selected for each retrieval process. For example, when the stored records R = 400 (records), the total processing time with the linear fitting is $y = 3 * 10^{-5}x + 17.604$, 17.604 (sec) is approximately four times 4.4 (sec) per 100 (records). It is thus approximately 44 (msec) per record. The value 44 (msec) per record is the same in Figure 12 and 13. As the y intercept is $3 * 10^{-5}$ (sec), its value may be negligible since the resolution of the measurement time is approximately 10^{-3} (sec). Although the number of DHT nodes is changed from 1 to 200, the total processing time remains flat. The stored records are changed from R = 100 (records) to R = 1,000 (records), and the total processing times are always flat with the increased multiplication of retrieval counts in Figure 14.

The processing time required to retrieve log messages is directly related to the waiting time for applications and administrators. As the number of retrieval counts is increased, the linear relationship with the processing time is shown in Figure 12. It can be easily estimated even if the number of retrieval counts has increased and decreased.

Although the number of stored records for log messages is increased to 100 K (records) in Figure 13, the total processing time for the retrieval is flat at 43.64 (msec) per record. The operation for DHT uses a random key to retrieve the log message. It shows that the process of the retrieval operation is not directly related to the number of stored records.

The number of DLS nodes is increased from 1 to 200 in Figure 14, but the total processing time to retrieve the specified log message is flat. This means that there is less overhead of the processing time for retrieval when the number of DLS nodes is increased.

7 Discussion

The section discusses the further studied items and challenges for DLS in the paper.

The evaluation results show that DLS has fewer overheads when retrieving log messages. The balance for the resource allocation is also considered for the logging service. Fewer of nodes with a large number of log messages causes some difficulties in processing log messages in each node. When the number of leaf computing nodes that generate log messages is increased, DLS must increase CPU and storage resources to process them.

The number of nodes in DHT will be optimised to support DLS. A balance exists between the number of nodes and the number of log messages. The dynamic load balance and increased computing resources can be applied to resolve it.

The emulated environment uses the memory data for log messages and local loopback for the network connections among DHT nodes. Therefore, the overheads of storage and network time may be avoided. They would be included in the evaluation in addition to DHT processing time. Future evaluations will include the network overhead and the storage I/O since the actual system has these overheads. In real computing environments, the storage and network overheads are affected by the entire performance of the logging service. In addition, the emulated environment is the Java toolkit, and the retrieval processing time per record is rather slow but stable. Therefore, the toolkit will be updated to improve the retrieval processing.

The CPU and storage resources for leaf computing nodes are different in general. Some nodes have large computing resources while others do not. The balance of resource consumption for CPU and storage can be considered. For example, nodes with large computing resources receive numerous log messages but nodes with smaller computing resources process fewer log messages. A DLS with a hybrid approach, which consists of dedicated nodes for the log service and shared nodes with applications, can be considered in this case. In the consideration of those approaches, the DHT algorithm will be updated with the resource usages rather than equally hashed allocation to store and retrieve the log messages. This presents the key challenge with respect to DLS.

Each node in DHT gets routing entries for several other nodes (Stoica et al., 2003). It maintains the routing information as those nodes join and leave the system. As the routing table is updated, each node can resolve the hash function with the information of several other nodes. As each node maintains information only about $O(\log N)$ for N, each search process can be completed for no more than $O(\log 2N)$ messages.

8 Conclusions

The paper proposes a DLS which has adopted DHT. It can be applied for the production system since DLS meets the set of requirements for requirements for cloud logging service. It can also resolve the issues and challenges that current logging services with CSM face. In the proposed DLS, the workloads to store, process and retrieve log messages can be distributed among distributed leaf computing nodes which applications and solutions are also deployed and running. DLS can operate distributed nodes without the dedicated CPU and storage resources for the logging service. The service level for DLS can be kept identical to that of CLS. Lifetime management for log messages, authentication to control access and resilience to ensure continuous service are proposed for DLS. Results of the evaluation of the emulation for the retrieval counts, stored records and node overheads shows that DLS is feasible for production usage since it exhibits a sustainable performance with respect to storing and retrieving log messages.

Acknowledgements

This work was supported by JSPS KAKENHI Grant No. 20K11776.

References

Bagnasco, S., Berzano, D., Guarise, A., Lusso, S., Masera, M. and Vallero, S. (2015a) 'Monitoring of IaaS and scientific applications on the cloud using the elasticsearch ecosystem', in *Journal of Physics: Conference Series*, Vol. 608, pp.12–16, IOP Publishing.

Bagnasco, S., Berzano, D., Guarise, A., Lusso, S., Masera, M. and Vallero, S. (2015b) 'Towards monitoring-as-a-service for scientific computing cloud applications using the elasticsearch ecosystem', in *Journal of Physics: Conference Series*, Vol. 664, pp.22–40, IOP Publishing.

Balalaie, A., Heydarnoori, A. and Jamshidi, P. (2016) 'Microservices architecture enables devops: migration to a cloud-native architecture', *IEEE Software*, May, Vol. 33, pp.42–52.

Chang, F., Dean, J., Ghemawat, S., Hsieh, W.C., Wallach, D.A., Burrows, M., Chandra, T., Fikes, A. and Gruber, R.E. (2008) 'Bigtable: a distributed storage system for structured data', *ACM Transactions on Computer Systems (TOCS)*, Vol. 26, No. 2, p.4.

Chaves, S.A.D., Uriarte, R.B. and Westphall, C.B. (2011) 'Toward an architecture for monitoring private clouds', *IEEE Communications Magazine*, December, Vol. 49, pp.130–137.

Dabek, F., Li, J., Sit, E., Robertson, J., Kaashoek, M.F. and Morris, R. (2004) 'Designing a DHT for low latency and high throughput', in *NSDI*, Vol. 4, pp.85–98.

DeCandia, G., Hastorun, D., Jampani, M., Kakulapati, G., Lakshman, A., Pilchin, A., Sivasubramanian, S., Vosshall, P. and Vogels, W. (2007) 'Dynamo: Amazon's highly available key-value store', in *Proceedings of Twenty-first ACM SIGOPS Symposium on Operating Systems Principles, SOSP'07*, ACM, New York, NY, USA, pp.205–220.

Gormley, C. and Tong, Z. (2015) *Elasticsearch: The Definitive Guide: A Distributed Real-Time Search and Analytics Engine*, O'Reilly Media, Inc., 1005 Gravenstein Highway North Sebastopol, CA 95472, USA.

Ikebe, M. and Yoshida, K. (2013) 'An integrated distributed log management system with metadata for network operation', in *2013 Seventh International Conference on Complex, Intelligent, and Software Intensive Systems*, July, pp.747–750.

Jiang, C.B., Liu, I.H., Liu, C.G., Chen, Y.C. and Li, J.S. (2016) 'Distributed log system in cloud digital forensics', in *2016 International Computer Symposium (ICS)*, December, pp.258–263.

Jonas, E., Schleier-Smith, J., Sreekanti, V., Tsai, C-C., Khandelwal, A., Pu, Q., Shankar, V., Menezes Carreira, J., Krauth, K., Yadwadkar, N., Gonzalez, J., Popa, R.A., Stoica, I. and Patterson, D.A. (2019) *Cloud Programming Simplified: A Berkeley View on Serverless Computing*, Tech. Rep. UCB/EECS-2019-3, EECS Department, University of California, Berkeley, February.

Karger, D., Lehman, E., Leighton, T., Panigrahy, R., Levine, M. and Lewin, D. (1997) 'Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the world wide web', in *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing, STOC'97*, ACM, New York, NY, USA, pp.654–663.

Komarek, A., Pavlik, J., Mercl, L. and Sobeslav, V. (2018) 'Metric based cloud infrastructure monitoring', in Xhafa, F., Caballé, S. and Barolli, L. (Eds.): *Advances on P2P, Parallel, Grid, Cloud and Internet Computing*, Springer International Publishing, Cham, pp.391–400.

Lakshman, A. and Malik, P. (2010) 'Cassandra: a decentralized structured storage system', *ACM SIGOPS Operating Systems Review*, Vol. 44, No. 2, pp.35–40.

Lin, X., Wang, P. and Wu, B. (2013) 'Log analysis in cloud computing environment with Hadoop and Spark', in *2013 5th IEEE International Conference on Broadband Network Multimedia Technology*, November, pp.273–276.

Liu, F., Tong, J., Mao, J., Bohn, R., Messina, J., Badger, L. and Leaf, D. (2011) 'NIST cloud computing reference architecture', *NIST Special Publication*, Vol. 500, No. 2011, p.292.

LogDNA (2019) *Log Management for the Kubernetes Age*.

Mell, P. and Grance, T. (2009) 'The NIST definition of cloud computing. National Institute of Standards and Technology', *Information Technology Laboratory, Version*, Vol. 15, No. 10, p.7.

Murugesan, P. and Ray, I. (2014) 'Audit log management in mongodb', in *2014 IEEE World Congress on Services*, June, pp.53–57.

Rhea, S., Geels, D., Roscoe, T., Kubiatowicz, J. et al. (2004) 'Handling churn in a DHT', in *Proceedings of the USENIX Annual Technical Conference*, Boston, MA, USA, Vol. 6, pp.127–140.

Rhea, S., Godfrey, B., Karp, B., Kubiatowicz, J., Ratnasamy, S., Shenker, S., Stoica, I. and Yu, H. (2005) 'OpenDHT: a public DHT service and its uses', in *Proceedings of the 2005 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, SIGCOMM'05*, Association for Computing Machinery, New York, NY, USA, pp.73–84.

Shudo, K., Tanaka, Y. and Sekiguchi, S. (2006) 'Overlay Weaver: an overlay construction toolkit', in *Proceedings of Symposium on Advanced Computing Systems and Infrastructures*, pp.183–191.

Shudo, K., Tanaka, Y. and Sekiguchi, S. (2008) 'Overlay Weaver: an overlay construction toolkit', *Computer Communications*, Vol. 31, No. 2, pp.402–412.

Shudo, K. (2006) *Overlay Weaver* [online] http://overlayweaver.sourceforge.net (accessed 1 August 2020).

Shvachko, K., Kuang, H., Radia, S., Chansler, R. et al. (2010) 'The Hadoop distributed file system', in *MSST*, Vol. 10, pp.1–10.

Sit, E., Morris, R. and Kaashoek, M.F. (2008) 'UsenetDHT: a low-overhead design for Usenet', in *Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation, NSDI'08*, USENIX Association, Berkeley, CA, USA, pp.133–146.

Sohlich, R., Janostık, J. and Spacek, F. (2014) 'Centralized logging system based on websockets protocol', in *13th International Conference on Telecommunications and Informatics*, Istanbul, Turkey.

Stoica, I., Morris, R., Liben-Nowell, D., Karger, D.R., Kaashoek, M.F., Dabek, F. and Balakrishnan, H. (2003) 'Chord: a scalable peer-to-peer lookup protocol for internet applications', *IEEE/ACM Trans. Netw.*, February, Vol. 11, pp.17–32.

Takeda, A., Hashimoto, K., Kitagata, G., Zabir, S.M.S., Kinoshita, T. and Shiratori, N. (2008) 'A new authentication method with distributed hash table for P2P network', in *22nd International Conference on Advanced Information Networking and Applications – Workshops (AINA Workshops 2008)*, March, pp.483–488.

Tanenbaum, A.S. and Van Steen, M. (2007) *Distributed Systems: Principles and Paradigms*, Prentice-Hall, Upper Saddle River, NJ 07458, USA.

Van Renesse, R., Birman, K.P. and Vogels, W. (2003) 'Astrolabe: a robust and scalable technology for distributed system monitoring, management, and data mining', *ACM Trans. Comput. Syst.*, May, Vol. 21, pp.164–206.

Xu, J., Chen, P., Yang, L., Meng, F. and Wang, P. (2017) 'LogDC: problem diagnosis for declartively-deployed cloud applications with log', in *2017 IEEE 14th International Conference on e-Business Engineering (ICEBE)*, November, pp.282–287.

Yen, T-F., Oprea, A., Onarlioglu, K., Leetham, T., Robertson, W., Juels, A. and Kirda, E. (2013) 'Beehive: large-scale log analysis for detecting suspicious activity in enterprise networks', in *Proceedings of the 29th Annual Computer Security Applications Conference, ACSAC'13*, ACM, New York, NY, USA, pp.199–208.

Yue, Y., Tian, L., Jiang, H., Wang, F., Feng, D., Zhang, Q. and Zeng, P. (2010) 'ROLO: a rotated logging storage architecture for enterprise data centers', in *2010 IEEE 30th International Conference on Distributed Computing Systems*, June, pp.293–304.

Zaharia, M., Chowdhury, M., Franklin, M.J., Shenker, S. and Stoica, I. (2010) 'Spark: cluster computing with working sets', *2nd USENIX Workshop on Hot Topics in Cloud Computing*, Boston, MA, USA, 22–25 June, Vol. 10, p.95.