Distributed Logging Service with Distributed Hash Table for Cloud

Takayuki Kushida

School of Computer Science
Tokyo University of Technology
Katakura, Hachouji, Tokyo 192–0914 JAPAN
kushida@acm.org

Abstract. The logging service on cloud is a critical component for administrators who maintain applications and solutions for end users. The service is usually a central server deployment to accept log messages from leaf computing nodes. Since several leaf computing nodes and their usages are dynamically changed for applications and solutions, the amount of generated log messages is also changed. This paper proposes the architecture and design for Distributed Logging Service (DLS) which can distribute processing powers and storage resources to leaf computing nodes with Distributed Hash Table (DHT). Those nodes generate log messages and also have DLS components locally. The evaluation results with the emulated environment show the feasibility of DLS with the scalability.

Keywords: Distributed Logging · Logging Service · Cloud Management · Distributed Hash Table.

1 Introduction

Cloud has been widely adopted in IT industry since cloud services could solve several issues of IT automation in production systems. One of cloud advantages is to acquire computing resources with user's requests as on-demand basis [18]. As the number of user's requests for cloud is changed as on-demand, the consumed computing resources are increased and decreased with those requests. In addition, the serverless computing which also dynamically manages computing resources has been emerged on cloud native environments [14].

On the other hand, the logging service on systems management is critical for production systems in enterprises, and applications in enterprises are migrated to microservice [3]. The logging service on cloud usually requires the dedicated CPU and storage resources to store log messages and search log messages. Therefore, the logging service has following challenges and issues which will be fixed for production systems in enterprises.

1. Critical backend service: Logging service is one of critical management services for production systems in enterprises. The service is always used for the operation and management [10]. The latest status is collected into the

management system since the production system has to be always up and running without any service disruption.
2. Central Logging Service (CLS): The logging service consists of several central servers on the deployment since it is based on the central model. All log messages are forwarded to the central servers, and they are processed on them. As the single point failure for CLS can be avoided, it is high availability and has a good scalability for both incoming log messages and user's access.
3. Dynamically changing environments: In early days of IT infrastructure, resource requirements for the logging service were static since the configurations of traditional data centers were fixed. As cloud is the dynamic changed computing environment, the estimation of computing and storage resources is one of major challenges for the architecture design. In addition, the resource requirement for the logging service is also changing and it's hard to estimate the resource allocation for the logging service.
4. Dedicated CPU and storage resources: Dedicated CPU resources is required although it is the backend management service [17]. Enough CPU resources can process those log messages for CLS. In addition, dedicated disk storages to store a large amount of log messages are also required in the central servers.

A new architecture is required to fix those issues and challenges for the logging service. This paper proposes Distributed Logging Service (DLS) with the distributed model in Fig 1. Fig. 2 shows the architecture of DLS for solutions and applications on cloud. It adopts the distributed deployment model instead of the traditional central model for the logging service.

The paper consists following sections. The section of "Related Work" describes the related studies for management services which the logging service is included in. The section of "Architecture" describes the architecture and the model of the logging service on cloud. The section of "Design of Distributed Logging Service" describes the required functions and the DLS design. The section of "Evaluation" describes the evaluation and its results for DLS. The section of "Discussions" describes the issues and challenges for this study. The section of "Conclusion" describes the summary and concluding remarks.

2 Related Work

Related studies are described in this section There are two categories such as Central management systems and Distributed Hash Table.

2.1 Central management systems

Apache Hadoop and Spark are software stacks which can process the large amount of data [23] [29], but they are also the central management systems. The well-known software product for the log management is ElasticSearch which is based on the central key-value database [10]. The architecture design is a central server which has the central database to search log messages. The central

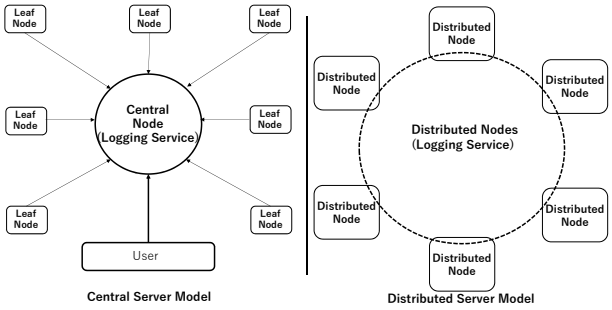

Fig. 1. Central Model and Distributed Model

architecture model for ElasticSearch could be usually applied to usual production systems on both cloud and on-premise data center. ElasticSearch, Logstash and Kibana (ELK) are a typical log management software suite for the central architecture model [1,2]. It collects all log messages from leaf computing nodes since log agents on leaf computing nodes are forwarding to the central servers for ElasticSearch. All search and analysis processes are done in the central servers.

LogDNA is a cloud service provider for the logging service. Once the LogDNA agents are installed on leaf computing nodes, log messages on leaf computing nodes are forwarded to LogDNA service, and those log messages are stored in the central storage. Users subscribe the logging services and install LogDNA agent. The operational dashboard from LogDNA is provided to search, analyze and report those log messages [7]. It is also suitable for Kubernetes/Docker containers to see log messages for the operation and management on the environments. It is a central architecture model for logging service, but all related issues for logging service are managed by LogDNA, log management as a service.

The design of the monitoring system (PCMONS) for private cloud monitoring was proposed with the implementation. Use case scenario with the architecture and the application was described in the paper [5]. It is a centralized model for the monitoring system which receives all monitoring data on the central server.

The integrated management system for log messages from distributed sensors was proposed as well as servers and network devices [12]. It also proposes a cross-processing system for several kinds of log messages. It aims to provide a flexible method to get access to distributed log messages using log attributes and values. Although it proposes the distributed management for log messages, the prototype of the distributed processing system is implemented with the local and the central fluentd manages the log messages. Therefore, the architecture is still the central server model.

Log messages are managed on Kubernetes containers and are forwarded through stdout/stderr to outside of the container environment. The installed logging-agent-pod in Kubernetes can forward log messages to the logging services [6]. Therefore, all log messages can be processed on CLS. Since dockers on Kubernetes environment are rather small than ordinal VMs, many docker

containers can be created and are generating many log messages in overall even if those containers are rather small system.

The robust and scalable technology for distributed monitoring, management and data mining have been proposed as Astrolabe [28]. It collects a large-scale system state, permitting rapid updates and providing on-the-fly attribute aggregation.

The distributed management architecture was proposed and its system performance was evaluated [13]. 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 the distributed management is proposed, it is still a limited distributed management.

2.2 Distributed Hash Table (DHT)

DHT has been used to support distributed contents management [19]. One of DHT implementations, Chord provides to support just one operation: given a key, it maps the key onto a node. Data location can be easily implemented on top of Chord by associating the key with each data item, and storing the key/-data pair at the node to which the key maps [25]. It provides load balance, decentralized, scalability and availability. In addition, Chord has several characters for distributed operation: 1. Node Joins and Stabilizations, 2. Impact of Node Joins on Lookups, 3. Failure and Replication, 4. Voluntary Node Departures. The consistency caching to map index to data contents is used for fault tolerance [15].

The Cassandra project is one of DHS implementations [16]. It is a one hop of DHT and eventually consistent with 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 columnFamilies, which are indexed in Cassandra.

The performance evaluation was reported on the implementation of DHash++ [8]. In this study, measurements with 425 server instances running hosts show that the latency optimizations for DHT++ can reduce the processing time by a factor of two. The time is required to locate and fetch data. In addition, the throughput optimizations 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 for DHT++ implementation can be referred to the performance aspect of DHT.

UsenetDHT is a service aiming to reduce the total storage dedicated to Usenet by storing all articles in a shared DHT [24]. The study describes the design and implementation of UsenetDHT. It allows a set of cooperating sites to keep a shared, distributed copied articles. It uses DHT that provides shared storage of those Usenet articles across the sites. This study utilizes shared storage for those articles with DHT.

Amazon Dynamo is highly available key-store database [9]. It is provided as Platform as a Service (PaaS) from Amazon Web Service. As it is characterized as a zero-hop DHT, each node maintains enough routing information locally

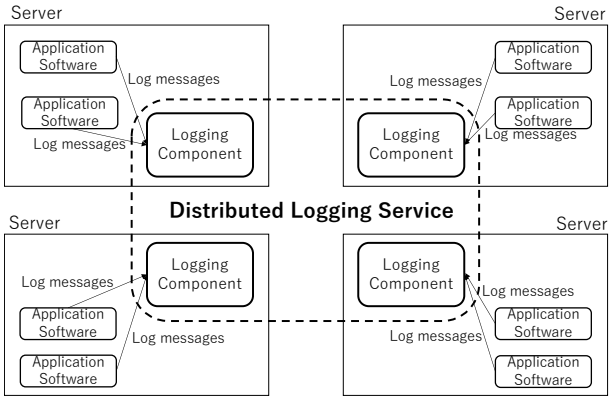

Fig. 2. Architecture of DLS

to route a request to the appropriate node directly. Bigtable is a distributed storage system to manage structured data that is designed to scale to a very large size [4]. It is a central service for end users and a good evidence for the scalable DHT implementation on the local server. Although they are not focused on the distributed logging service on cloud, their studies and implementations are quite valuable for the proposed architecture and design.

3 Architecture

This section describes the architecture of the logging service for the central and distributed model.

3.1 Requirements

The architecture decision for the central and distributed model is done with the investigations.

Two major functions are required for the logging service in general. They are "Store" and "Retrieve" functions for those log messages. The "Store" function is simply to store them in the repository without a longer delay. The "Retrieve" function is to get them with the specified condition of the search. It retrieves log messages which can be used for different kinds of activities which are in development, test and operation phases.

The applications and solutions need to follow the dynamically changing computing environment. It means that the flow of log messages from leaf computing nodes has often a large amount of data for the logging service. It is accepted and processed on the logging service.

The consumption of CPU and storage resources is minimized in general. Logging service needs a lot of dedicated CPU resource to create the index and search the specific log message in the repository. In addition, an amount of

storage to store log messages is required. The efficient resource management for both CPU and storage resource can be considered.

Although the logging service is the backend management service, it is critical for the production system. Because the service provides any past event with log messages. The systems management can find the failure event and the unusual message among them and understand what was happened in the past.

3.2 Central and distributed model

Fig. 1 shows "Central Server Model" (CSM) and "Distributed Server Model" (DSM) for the logging services on the central server. The CSM accepts and processes all log messages The left side of Fig. 1 is the CSM. In the left side of Fig. 1, "Leaf Computing Nodes" send log messages to "Central Node" which is the logging service. Once Log messages are received on the logging service, they are processed to put the index and stored in the storage. User can search and retrieve the log messages through the dashboard on the logging service in the left side of Fig. 1.

The operation and management functions including the logging service usually require the central servers which collect log messages from leaf computing nodes. When a large amount of logging messages is generated and forwarded to the central servers, the large amount of CPU and storage resources are required to process and store them on time. In addition, as the peak flow of log message is supported, the idle or unused resources is reserved for the logging service on cloud. The systems management function is in general a kind of overhead function for the services even when it is mandatory required for the development, test and operation phases.

On the other hand, the DSM can be processed on those distributed nodes. The workload can be assigned to distributed nodes instead of a large central server. If there is additional overhead for DSM on the logging service, the feasibility for the logging service is investigated to support the logging service with DSM.

DSM for logging services can be considered since several drawbacks for CSM can be fixed [27]. The right side in Fig. 1 shows the DSM for the logging service. Those log messages are forwarded and stored on distributed nodes in this model instead of collecting the central server. DLS provides the processing service for log messages on those distributed nodes. It means that there is no central server to store those log messages and search the specific log message.

The management server for the logging service and the access point which supports API can be on distributed servers. They act as a part of the logging service instead of assigning to the dedicated central server which processes the transactions for the logging service.

3.3 DHT

DHT is a solid technology which can share a large amount of data with the robustness and availability on distributed nodes. Chord is one of DHT imple-

mentations [25]. It manages the hash table on the distributed nodes. Their data entries are stored into the distributed nodes with the hash table. There are several advantages for DHT: 1. There is a resilience to join and leave networks. DHT can also support the network changes. 2. Hashed data can be automatically distributed to DHT nodes. Since stored data are replicated across nodes, data loss can be avoided with the lost node. The removal of any single node doesn't have an impact at all.

On the other hand, DHT has several considered items: 1. There is no native support for events or triggers. 2. It doesn't provide absolute guarantees on data consistency and integrity. 3. It isn't efficient for group queries, range queries or other kinds of data lookups. 4. There is no authority in the setup. Therefore, any node must cooperate among them in case certain decisions need to be made. 5. The time range of lookup for data is O(log(n)). It may take several seconds, depending on real location, the number of nodes and those latencies among nodes. As those considered items can be managed, DHT can be applied to the logging service.

3.4 Architecture of DLS

Fig. 2 shows the architecture of DLS. Each server has the process for "Application Software" and "Logging Component". Application software on each server provides the solution services to end users. They also generate a lot of log messages which are stored in local files. The logging component on each server is deployed on the same server where the application software is running. It finds and retrieves those log messages for the request with API. In Fig. 2, those logging components are interconnected among leaf computing nodes.

In general, after the logging service is deployed on the cloud environments, service owner or user require to set up the log agent which collects and forwards log messages for their applications to the central log server.

Therefore, the central server for the logging service require enough computing resources such as CPU, memory and disk since the central server processes a lot of log messages which are forwarded by leaf computing nodes. Although the logging service on cloud is critical for the operation and management, the service is only backend operation function.

3.5 Use case scenarios for retrieval and alert notification

Fig. 4 shows the example of log messages which are generated by Web server on the operation phase. They are forbidden and error messages with date, time and IP address. Log messages with the same and similar format are stored in the logging service.

Once log messages are stored, they are searched and retrieved from the repository. If the specified server on the production system has error as the example of log messages in Fig. 4, the administrator starts investigating to know two aspects briefly: 1. One is to know the current status of the specific server. The

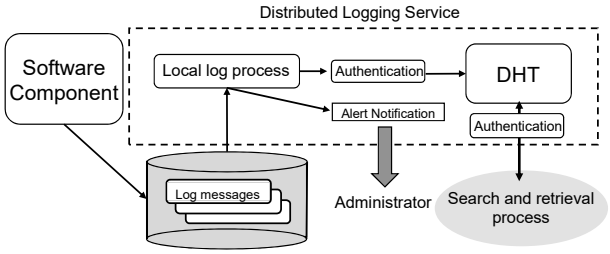

Fig. 3. Local DHT process

```
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/"
```

Fig. 4. Example of log messages

current status can be investigated with the monitoring and logging service. 2. Another is to know the time when the specified server had error or was down.

In addition, if the failure is happened in the operation, log messages include the failure event. Those events are linked to generate the alert notification. Alert notifications is usually linked to failure for system/service, unusual status of the system or security incident. When the alert notification is generated with log messages, operators and administrators review the specific log message for the alert notification and execute the recovery process to solve the incident.

If the unusual failure event on the server is detected in those log messages, the alert notification is generated to notify to the administrator. The administrator fixes the issue and recover the system immediately. In addition, he/she must check those log messages and solve the root cause for the issue.

4 Design of Distributed Logging Service

This section describes the design of DLS for production usages.

4.1 Adoption of DHT and local process

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

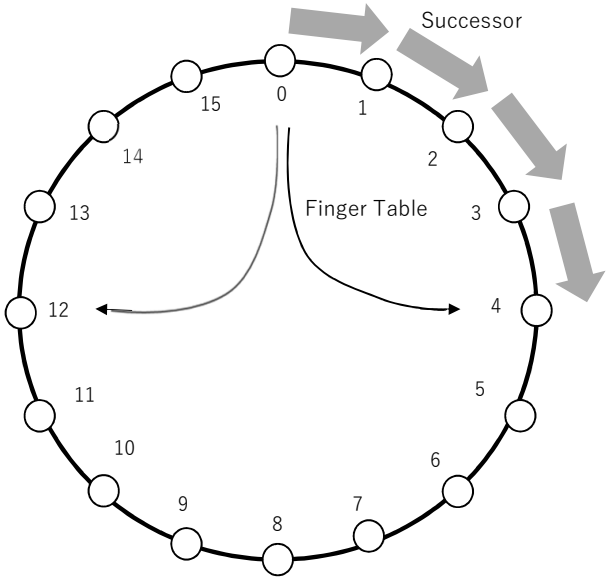

Fig. 5. DHT - Chord

which are stored into local log files at first. Those log messages are accumulated in local log files. The local agent forwards those log messages to DHT. Therefore, log messages are stored into the table in DHT.

The hash table is provided to get keys and values. Both keyes and values are defined for DHT. Keys are timestamps and type of log messages, values are the message itself. The definition can reduce the size of DHT and has the separation between the index for search and actual message contents for the retrieval.

Fig. 5 shows that DHT manages logging messages for DLS. Once the query for DHT are sent to find log messages in DHT [11], it can be retrieved for the search results in DHT since DHT keeps those log messages.

Fig. 3 shows the process on the local host which is one of DHT nodes. Software component in local host stores log messages to the storage. Each local host has DHT component which can accept log messages which are locally generated. They are submitted to the local part of DHT. DHT starts sharing those log messages with other DHT nodes.

DLS can also check them whether the alert notification is generated. If it's not required, the log messages are submitted to DHT. Once they are stored in DHT, search and retrieval process will get the specific log message from DHT. The administrator can search and retrieve it for the further analysis of the infrastructure issues with the operation dashboard. On the other hand, CSM for the logging service adopts to install the log agent which forwards log messages to the central server.

4.2 Lifetime for log messages

Log messages from leaf computing nodes are locally stored in DHT. Since they are created with various kinds of activities in those leaf computing nodes and forwarded to DHT in DLS, the storage sizes for DHT gets are growing so fast. Therefore, the expiration time for those log messages is configured so that the total size of local storage can be managed.

Old entries for log messages are simply moved to inexpensive storage from DHT once the expiration time is reached. After old ones are moved to the inexpensive repository and it takes sometime later, the oldest ones in the repository are permanently removed from DHT.

4.3 Access control

Log messages are kept under the access control. DHT contains the hash table to store and retrieve values with keys. It is deployed to those multiple nodes. Since different users and processes get to access to DLS, local authentication for them is required to valid access to DLS.

The first item (storing) for access control is to store log messages which come from leaf computing nodes. The access control is also required to store log messages. Since local log messages are stored in the system, the process is done by the local log agent only. The second item (retrieving) for access control is to retrieve log messages for analysis on systems management. The method is based on API, and it provides access control when log messages are retrieved

Authentication module of DLS is shown in Fig. 3. In Fig. 3, user authentication can be applied for access control. Authentication is to protect log messages from invalid access.

In other study, there is an authentication method called Hash-based Distributed Authentication Method (HDAM). It realizes a decentralized efficient mutual authentication mechanism for each pair of nodes [26]. As the authentication service on cloud can be relied on, the paper has adopted as usual authentication service.

5 Evaluation

This section describes the evaluation for the proposed architecture and design. The performance test on the emulated environment has been done to evaluate the deployment for DLS. The scalability with a reasonable performance is achieved with the proposed deployment for DLS. As the number of DHT nodes is increased, the capability to process log messages is evaluated. For example, it is examined how much the performance to retrieve log messages on DHT is changed.

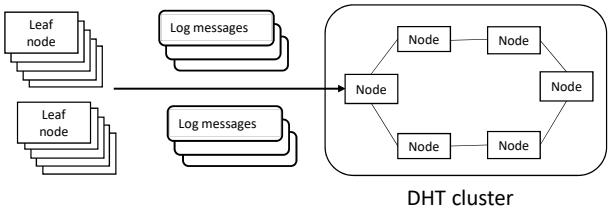

Fig. 6. Typical DLS setup on evaluations

5.1 Emulation toolkit

The emulated environment is constructed with the DHT software emulator called OverlayWeaver [21] [22]. It facilitates the implementation of routing algorithms, and runs with multiple well-known algorithms for DHT just in hundreds of lines of codes with the toolkit [20].

This toolkit also provides a common API to support higher level components which are DHT and multicast functions. It can support multiple DHT algorithms which are Chord, Kademlia, Koorde, Pastry and Tapestry. Those DHT algorithms and routing methods have been discussed and evaluated on the previous study [25].

The proposed architecture has been evaluated with Chord, DHT algorithm. It is well-know DHT algorithm and the emulated environment could be constructed with the toolkit. As it is used for DHT algorithm in the evaluation, the iterative routing is applied to DHT. It is a basic algorithm and routing method, and the evaluation has been focused on the feasibility of DLS with DHT.

5.2 Evaluation Scenarios

Fig. 6 shows a typical configuration and the processing of log messages for DLS. The evaluation has adopted use-case scenarios with the typical configuration on the toolkit. In Fig. 6, those leaf computing nodes generated log messages and forward them to DHT which is a cluster of nodes. The cluster consists of several nodes on DHT. DHT cluster processes incoming log messages from leaf computing nodes. As those nodes are located at distributed locations, the processing for incoming log messages can be distributed to those nodes.

Basic functions for the logging service are “Store” and “Retrieve” for log messages. The “Store” function is to store those incoming log messages into the backend storage which is DHT in the proposed method. This process can be sequential and the submissions of log messages are different time. It can be the batch process and not related to use experience.

Thus, the “Retrieve” function is evaluated for the performance of the proposed method. It takes some time to retrieve the log message which is specified by user since many log messages are stored in DLS with DHT. The performance evaluation is to retrieve log messages from the number of stored messages.

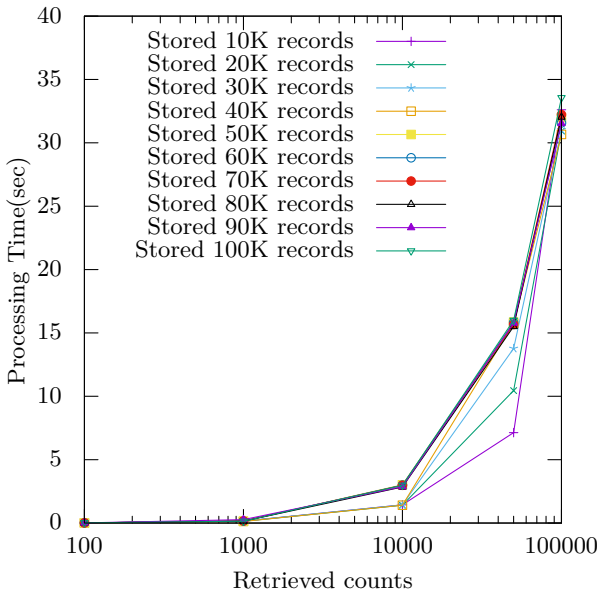

Fig. 7. Processing time for retrieved counts

The emulated environment runs on the single computer which Linux is installed. The connections among DHT nodes are the loopback network interfaces which can avoid the network overhead for the interactions among multiple computers. In this environment, the performance of DHT can be also measured with the different retrieval data set and the different number of stored messages.

5.3 Evaluation Results

Fig. 7 shows the processing times for the count of retrieval records (log messages) which are stored in DLS when the number of DHT nodes is fixed at 10. The number of stored log messages in DLS is changed from 10K-100K records. The retrieved counts are changed at 100, 1000, 10000, 50000 and 100000 in Fig. 7. The processing times to retrieve log messages from those stored messages are measured with the given conditions.

Although the number of stored records which contain log messages are from 10K to 100K with increased 10K, the processing time at 100K is only a slightly different from one at 10K records. Therefore, the processing time isn't directly related to the number of stored recorded. As the number of retrieval counts is increased from 100 to 100000, the processing time is approximately 32 seconds

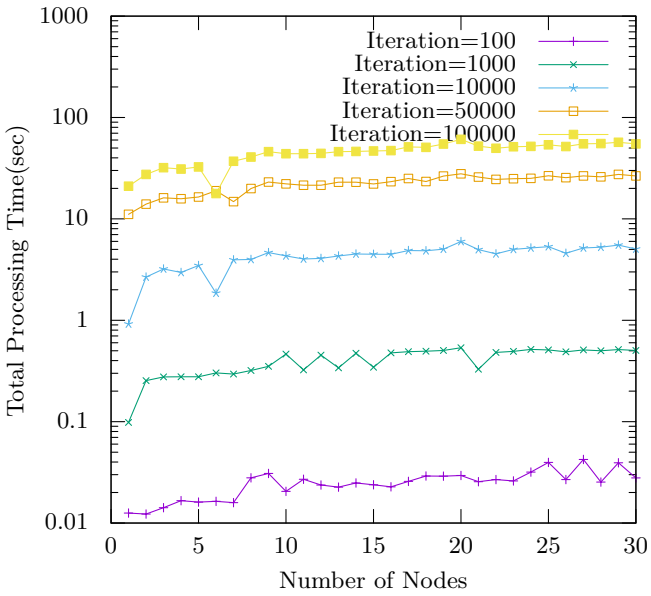

Fig. 8. Processing time for multiple nodes

at stored record 10000 in Fig. 7. The processing times are increased with the number of retrieved counts but not the linear relationship.

In Chord for DHT, each node gets routing entries for few other nodes [25]. 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 information of few other nodes. As each node maintains information only about O(log N) for N, each search process can be completed in no more than O(log2 N) messages.

The overhead for the processing on DHT nodes has been evaluated.

Fig. 8 shows the graph of processing time with the number of nodes which DLS consists of. The x-axis is the number of nodes from 1 to 30, and the y-axis is the processing time for the retrieval of records in DHT. When the specific record is searched, the retrievals of records (log message) are randomly selected. The number of iterations is the count to retrieve the specified log messages. Those iterations are 100, 1000, 10000, 50000 and 100000 in Fig. 8. Those processing times to retrieve log messages is directly linked to the waiting time for applications and users. In Fig. 8, although the number of DLS nodes is increased from 1 to 30, the total processing time to retrieve the specified log message is rather similar or slightly increased for the number of DLS nodes. It means that the overhead of the DLS processing time is minimum. Those iterations are the

number of retrieval trials. As they are changed from 100 to 100000, the processing times to retrieve the specified log message changed as the same as those iterations. The number of iterations to retrieve the specific log messages from DLS is directly related to the processing time.

The evaluation results on the emulated environment show a scalability of DLS without any accumulated overhead.

6 Discussions

The section discusses the considerations and challenges for the proposed method.

The evaluation results exhibit to provide DLS for less overhead of DHT. The balance for the resource allocation for the logging service is considered. Less number of nodes with a large number of log messages causes some issue to process log messages in each node. When the number of leaf computing nodes which generate log messages is increased, DLS requires to increase CPU and storage resources to process them.

In addition, the search processing power is increased when the large number of log messages is stored. There is a balance between several nodes and the number of incoming log messages, and it is the search processing for the retrieved log messages. The dynamic load balance and increased computing resources can be applied to resolve it.

The CPU and storage resources for those leaf computing nodes are different in general. Some node has a large computing resource but some doesn't. The balance of resource consumption for CPU and storage is considered. For example, the node with large computing resource receives a lot of log messages but the node with less computing resource processes less log messages. DLS with 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 is updated with the resource usages instead of equally hashed allocation to store and retrieve those log messages.

7 Conclusion

The paper proposes the distributed logging service with DHT since it meets the requirements for the logging service. It could solve several issues that are faced for the current logging service with CLS. In the proposed method, the workloads to store, process and retrieve those log messages can be distributed among distributed leaf computign nodes which applications and solutions are currently deployed on. The DLS can reduce the dedicated CPU and storage resource for the systems management services and the logging service keeps the same level of service as CLS. Use cases, lifetime management and access control have been designed for DLS. The evaluation results for the retrieval counts and node overheads shows the feasibility with a scalability and a sustainable performance.

References

1. Bagnasco, S., Berzano, D., Guarise, A., Lusso, S., Masera, M., Vallero, S.: Monitoring of iaas and scientific applications on the cloud using the elasticsearch ecosystem. In: Journal of Physics: Conference Series. vol. 608, p. 012016. IOP Publishing (2015)
2. Bagnasco, S., Berzano, D., Guarise, A., Lusso, S., Masera, M., Vallero, S.: Towards monitoring-as-a-service for scientific computing cloud applications using the elasticsearch ecosystem. In: Journal of Physics: Conference Series. vol. 664, p. 022040. IOP Publishing (2015)
3. Balalaie, A., Heydarnoori, A., Jamshidi, P.: Microservices architecture enables devops: Migration to a cloud-native architecture. IEEE Software **33**(3), 42–52 (May 2016). https://doi.org/10.1109/MS.2016.64
4. Chang, F., Dean, J., Ghemawat, S., Hsieh, W.C., Wallach, D.A., Burrows, M., Chandra, T., Fikes, A., Gruber, R.E.: Bigtable: A distributed storage system for structured data. ACM Transactions on Computer Systems (TOCS) **26**(2), 4 (2008)
5. Chaves, S.A.D., Uriarte, R.B., Westphall, C.B.: Toward an architecture for monitoring private clouds. IEEE Communications Magazine **49**(12), 130–137 (December 2011). https://doi.org/10.1109/MCOM.2011.6094017
6. community, K.: Logging architecture in kubernetes (2018), `https://kubernetes.io/docs/concepts/cluster-administration/logging/`
7. company, L.: Logdna web site (2019), `http://logdna.com`
8. Dabek, F., Li, J., Sit, E., Robertson, J., Kaashoek, M.F., Morris, R.: Designing a dht for low latency and high throughput. In: NSDI. vol. 4, pp. 85–98 (2004)
9. DeCandia, G., Hastorun, D., Jampani, M., Kakulapati, G., Lakshman, A., Pilchin, A., Sivasubramanian, S., Vosshall, P., Vogels, W.: Dynamo: Amazon's highly available key-value store. In: Proceedings of Twenty-first ACM SIGOPS Symposium on Operating Systems Principles. pp. 205–220. SOSP '07, ACM, New York, NY, USA (2007). https://doi.org/10.1145/1294261.1294281, `http://doi.acm.org/10.1145/1294261.1294281`
10. Gormley, C., Tong, Z.: Elasticsearch: The Definitive Guide: A Distributed Real-Time Search and Analytics Engine. " O'Reilly Media, Inc." (2015)
11. Harren, M., Hellerstein, J.M., Huebsch, R., Loo, B.T., Shenker, S., Stoica, I.: Complex queries in dht-based peer-to-peer networks. In: Druschel, P., Kaashoek, F., Rowstron, A. (eds.) Peer-to-Peer Systems. pp. 242–250. Springer Berlin Heidelberg, Berlin, Heidelberg (2002)
12. Ikebe, M., Yoshida, K.: An integrated distributed log management system with metadata for network operation. In: 2013 Seventh International Conference on Complex, Intelligent, and Software Intensive Systems. pp. 747–750 (July 2013). https://doi.org/10.1109/CISIS.2013.134
13. Jiang, C.B., Liu, I.H., Liu, C.G., Chen, Y.C., Li, J.S.: Distributed log system in cloud digital forensics. In: 2016 International Computer Symposium (ICS). pp. 258–263 (Dec 2016). https://doi.org/10.1109/ICS.2016.0059
14. 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., Patterson, D.A.: Cloud programming simplified: A berkeley view on serverless computing. Tech. Rep. UCB/EECS-2019-3, EECS Department, University of California, Berkeley (Feb 2019)
15. Karger, D., Lehman, E., Leighton, T., Panigrahy, R., Levine, M., Lewin, D.: 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. pp. 654–663. STOC '97, ACM, New York, NY, USA (1997). https://doi.org/10.1145/258533.258660, http://doi.acm.org/10.1145/258533.258660
16. Lakshman, A., Malik, P.: Cassandra: a decentralized structured storage system. ACM SIGOPS Operating Systems Review **44**(2), 35–40 (2010)
17. Lin, X., Wang, P., Wu, B.: Log analysis in cloud computing environment with hadoop and spark. In: 2013 5th IEEE International Conference on Broadband Network Multimedia Technology. pp. 273–276 (Nov 2013). https://doi.org/10.1109/ICBNMT.2013.6823956
18. Mell, P., Grance, T.: The nist definition of cloud computing. NIST special publication **800**(145), 7 (2011), http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf
19. Rhea, S., Geels, D., Roscoe, T., Kubiatowicz, J., et al.: Handling churn in a dht. In: Proceedings of the USENIX Annual Technical Conference. vol. 6, pp. 127–140. Boston, MA, USA (2004)
20. Shudo, K.: Overlay weaver (2006), http://overlayweaver.sourceforge.net
21. Shudo, K., Tanaka, Y., Sekiguchi, S.: Overlay weaver: An overlay construction toolkit. In: Proceedings of Symposium on Advanced Computing Systems and Infrastructures. pp. 183–191 (2006)
22. Shudo, K., Tanaka, Y., Sekiguchi, S.: Overlay weaver: An overlay construction toolkit. Computer Communications **31**(2), 402–412 (2008)
23. Shvachko, K., Kuang, H., Radia, S., Chansler, R., et al.: The hadoop distributed file system. In: MSST. vol. 10, pp. 1–10 (2010)
24. Sit, E., Morris, R., Kaashoek, M.F.: Usenetdht: A low-overhead design for usenet. In: Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation. pp. 133–146. NSDI'08, USENIX Association, Berkeley, CA, USA (2008), http://dl.acm.org/citation.cfm?id=1387589.1387599
25. Stoica, I., Morris, R., Liben-Nowell, D., Karger, D.R., Kaashoek, M.F., Dabek, F., Balakrishnan, H.: Chord: A scalable peer-to-peer lookup protocol for internet applications. IEEE/ACM Trans. Netw. **11**(1), 17–32 (Feb 2003). https://doi.org/10.1109/TNET.2002.808407, http://dx.doi.org/10.1109/TNET.2002.808407
26. Takeda, A., Hashimoto, K., Kitagata, G., Zabir, S.M.S., Kinoshita, T., Shiratori, N.: 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). pp. 483–488 (March 2008). https://doi.org/10.1109/WAINA.2008.203
27. Tanenbaum, A.S., Van Steen, M.: Distributed systems: principles and paradigms. Prentice-Hall (2007)
28. Van Renesse, R., Birman, K.P., Vogels, W.: Astrolabe: A robust and scalable technology for distributed system monitoring, management, and data mining. ACM Trans. Comput. Syst. **21**(2), 164–206 (May 2003). https://doi.org/10.1145/762483.762485, http://doi.acm.org/10.1145/762483.762485
29. Zaharia, M., Chowdhury, M., Franklin, M.J., Shenker, S., Stoica, I.: Spark: Cluster computing with working sets. HotCloud **10**(10-10), 95 (2010)