

Layer-Integrated Edge Distributed Data Store for Real-time and Stateful Services

Koichiro Amemiya^{*†} and Akihiro Nakao[†]

^{*}Fujitsu Laboratories Ltd., Kawasaki, Japan, Email: amemiya.kouichi@fujitsu.com

[†]The University of Tokyo, Tokyo, Japan, Email: nakao@nakao-lab.org

Abstract—Network operations for real-time and stateful Internet-of-Things (IoT) services, such as Cyber Physical System (CPS) are considered a necessity for driving further innovation in smart and inclusive society. MEC (Multi-access Edge Computing) is one of the essential enablers for those services. But moving devices cannot enjoy the MEC efficacy of low-latency because the geographical coverage area of a single MEC is limited. In this paper, we posit that one of the most significant challenges for realizing real-time and stateful services utilizing MEC is a geographically distributed data store that is designed to support the three requirement, low-latency, stateful, and data locality. Especially the control of the location of data and the directory service that enable quick data access are required to enable real-time and stateful IoT services. Our contributions are four-fold. First, we define the architecture of a geographically distributed data store providing low-latency and consistent data access. Second, we propose in-network directory cache that accelerates the search for the distributed data. Third, we propose a novel algorithm which decides data location to achieve low-latency data access. Finally, we define an index to evaluate the efficacy of the geographically distributed data store. Our evaluation result shows that the proposed method enables low-latency and consistent data access, even when mobility devices move across edge coverage areas.

Keywords—edge computing, stateful, real-time, distributed data store, IoT, Internet of Mobile Things

I. INTRODUCTION

Network operations for Real-time Internet-of-Things (IoT) services, such as Cyber-Physical System (CPS), are considered a necessity for driving further innovation in smart and inclusive society. The network operations for Real-time IoT requires continuous collection of the state data of objects in the real world, analysis of the data, and the feedback control and notification in a real-time manner. For example, a safe driving service keeps the state data of vehicles and pedestrians on the road and updates the state data periodically in response to the state change notification from the field. The vehicles and pedestrians mutually refer and notify their state data through the service system to avoid accidents and achieve safe driving.

However, we posit that there are several requirements in enabling such real-time IoT network operations in terms of the following three aspects: *low-latency*, *statefulness*, *data locality*. First, *low-latency service delivery* is required. The system for the real-time IoT service must complete collecting, analyzing data, and issuing feedback control or notification within low latency that services require. The system can

interact with the physical world in a real-time manner by satisfying the latency requirement.

Second, the system must support *stateful services*. When CPS services prevail, most of the services are expected to be stateful. The stateful services should keep the state consistent. Additionally, stateful services that require low-latency must select the location of the state data store properly to achieve low-latency and consistency simultaneously.

Third, the system must have the ability to control data distribution so that *data locality* is assured. Keeping data in the vicinity of the data source enables to control the geographical range of data distribution besides providing low-latency and traffic reduction. There are regulations to control data distribution, e.g., GDPR (General Data Protection Regulation), that restricts the range of data distribution. And the concept of DFFT (Data Free Flow with Trust) promotes the use of data among data providers and users based on the policy and reliability between data providers and users.

In the light of these observations, we posit that *geographically distributed data store* must be designed to support the three requirements described above, low latency, stateful, and data locality simultaneously. Utilization of MEC (Multi-access Edge Computing) is a practical approach for achieving low latency. But when mobile terminals are the source of data, read and write the data access delay will be deteriorated as the terminal moves if the location of the data storage is fixed. Therefore when data access sources move and the service requires low-latency, stateful, and consistency, a new architecture for a distributed data store is required. We also believe the need for *generic index* that measures the effectiveness of the newly proposed geographically distributed data store considering latency, locality, and consistency.

In this paper, in response to the *three requirements* and *two features* to fulfill the requirements, we propose *Layer-Integrated Edge Distributed Data Store* that satisfies low-latency, stateful, and data locality. We also propose *Delay Area-of-interest Index (DAI)*, a generic index to evaluate geographically distributed data store.

There are proposals about geographically distributed data stores that use space-filling curves, such as Hilberts curve[1][2]. Those works map a geographical location to a value on a space-filling curve. And the value is used as a key for the distributed data store. But geographic proximity does not always guarantee low latency because the latency depends on distance on network topology of the distributed data store.

NetCache[3] is an in-network cache technology for a distributed key-value store (KVS). It is a technology to improve the read latency utilizing cache data on the on-chip memory in programmable switches. The size of the value in KVS is limited because the size of the on-chip memory is limited. And the performance of data update can not be improved as the update message need to be sent to the fixed node where the master entry is stored. As for the evaluation index for the geographically distributed data store, FogStore[2] proposes a notion Context of Interest that defines the geographical range where the data should be consistent. But the Context of Interest is only a requirement for the range of data consistency and can not be used for evaluating the system.

Our proposed Layer-Integrated Edge Distributed Data Store provides low-latency data access with fulfilling consistency for stateful services, and data locality simultaneously. This data store dynamically deploys data entries where the read-write performance for the data entry is high by considering the location of the data read-write sources. The data store is equipped with a directory function that manages the location of distributed data. For better performance of searching the data location, the data store also provides in-network directory cache functionality. We also propose a Delay Area-of-interest Index (DAI) for evaluating the geographically distributed data store. The index quantifies the performance of the geographically distributed data store from the viewpoints of latency and data locality.

Our contributions in this paper are four-folds. First, we propose Layer-Integrated Edge Distributed Data Store. The data store provides low-latency data access with fulfilling consistency for stateful services, and data locality simultaneously. Second, we propose in-network directory cache functionality that accelerates the search of distributed data. Third, we develop an algorithm that decides the optimal arrangement of data entries to achieve low-latency data access securing the data locality policy. Finally, we define an index that quantifies the performance of the geographically distributed data store. Our evaluation result shows that our proposed method enables clients to access data with a latency of 13ms, excluding wireless communication delay even when the client is moving. And in-network directory cache reduces the response time to 9ms on average.

The rest of the paper is organized as follows. Section II describes the architecture of the proposed distributed data store. We present a data deployment algorithm for optimal data location arrangement, in-network directory cache, and evaluation index for evaluating geographically distributed data store, in Section III, IV, V, respectively. Section VI shows the evaluation result of the proposed method, and we evaluate the result using the proposed index in section VII. We discuss the related works in Section VIII and conclude the paper in Section IX.

II. LAYER-INTEGRATED EDGE DISTRIBUTED DATA STORE

In this section, we show the architecture of Layer-Integrated Edge Distributed Data Store (LI-EDDS) and describe how

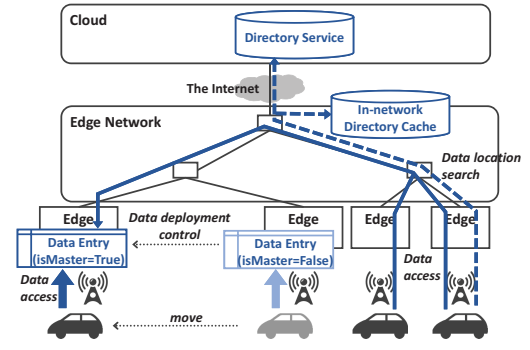


Fig. 1. LI-EDDS system overview

to achieve low-latency, data locality, and consistency. Fig. 1 shows overview of the proposed system, and 2 shows the functional architecture.

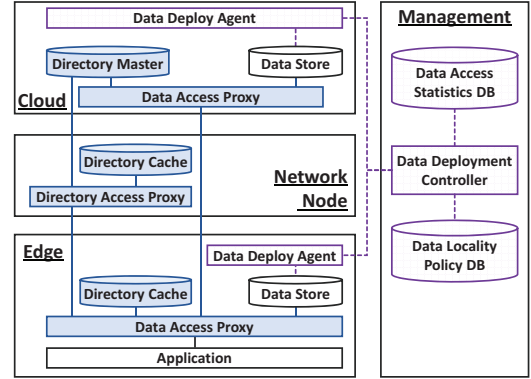


Fig. 2. LI-EDDS architecture

To achieve low-latency, the LI-EDDS controls the location of data entries among cloud and edge servers. Data deployment controller in Fig. 2 arranges the location of data entries to minimize the average latency of data access from clients that is weighted with access frequency as shown in Fig. 1. The detailed algorithm for data deployment algorithm is shown in section III. For data accessibility to moving data, LI-EDDS provides directory service that manages the location of the distributed data entries and is used to search for the data. The master data of the directory service is stored in the cloud to keep the information consistent. Each edge server keeps copies of a part of directory entries, directory cache. Network node on the way from the edge servers to the cloud also keeps directory cache. The in-network directory cache enables data access clients to search for the location of the data entries quickly.

To achieve data locality requirements, LI-EDDS explicitly controls and manages the location of the data entries. And the data location information is stored in the Directory service.

To achieve consistency, the data update is allowed on a single but moving data master. The directory service manages the location of the master data. And we set a *isMaster* flag to

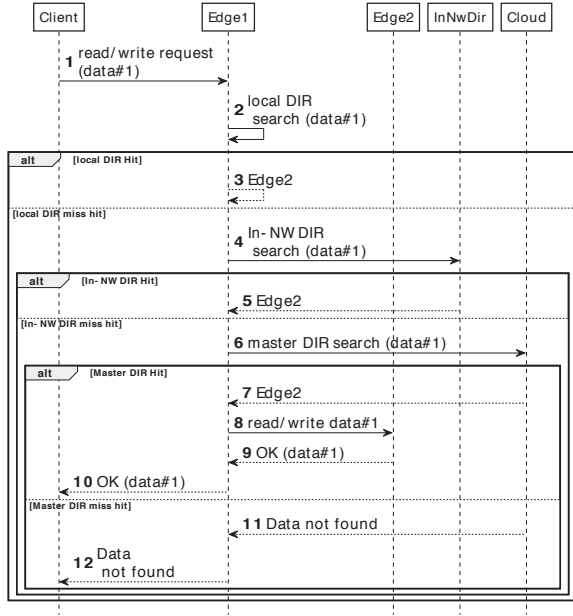


Fig. 3. Read/Write Sequence

each data entry as metadata indicating that the data entry is updatable. The data entry with $isMaster=False$ is a staled and is not allowed to be updated.

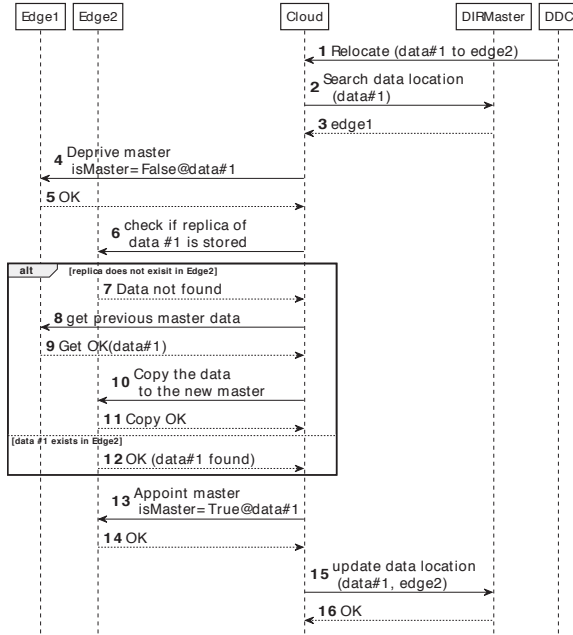


Fig. 4. Data Relocation Sequence

Fig. 3 shows the basic data access sequence. Data access proxy in the edge server (e.g., Edge1) receives data access requests from the client and determines the location of the data by querying the data location to the directory services. And the proxy transfers the request to the remote proxy in the other edge server (e.g., Edge2) when the remote node is the

master of the specified data entry. The remote proxy executes data handling and sends back the data access response to the local client through the local proxy.

Fig. 4 shows a sequence to update the location of the data entry to provide low-latency data access. Data deployment controller, DDC in Fig. 4, order to relocate the data entry to data deployment agent in the cloud. The data deployment agent search for the location of the master for the entry (e.g., data entry #1) and get the location(e.g., Edge1). And the data deployment agent unset $isMaster$ flag on the current master data entry and set $isMaster$ flag *True* in the newly selected master data entry on the specified edge server(e.g., Edge2). This sequence with $isMaster$ flag in data entry prevents write access to other than the master data and keeps the data consistent.

III. DATA DEPLOYMENT ALGORITHM

This section describes the data deployment algorithm for providing low-latency data access, even when the data access client is moving.

The proposed algorithm decides the location of data entry using the aggregated data access statistics information and network latency between edge servers and cloud. The objective function of the proposed algorithm is the summation of the number of data access weighted by network latency between the data access source and destination. And the algorithm decides the location of the data entry such that the objective function becomes minimum.

Eqn. (1)-(9) are the formula for the deployment algorithm. The variables used are defined as follows. i is the identifier of the edge server, and e denotes data entry. j is the edge server where the master data entry is deployed, and k is the edge server where the replica data entry is deployed. The aggregated read access per data entry e , and edge server i is denoted by R_i^e , and aggregated write access is denoted by W_i^e . D_{ij} is the network delay between the edge server i and edge server j . There define three variables, X_j^e , Y_k^e , and $Z_{i,l}^e$. $X_j^e = 1$ when data entry e for write is deployed in edge server j , and $Y_k^e = 1$ when replica of the data entry e is deployed in edge server k . And $Z_{i,l}^e$ means that read access from edge i uses the master or replica data on edge l . $\mathcal{P}_e$ is a set of data locality policy that defines the possible locations where the data entry can be deployed.

$$\sum_j \{W_i^e \times D_{i,j} \odot X_j^e\} + \quad (1)$$

$$X_j^e \times D_{j,k} \times \text{Tr} Y_k^e \cdot \sum_i W_i^e +$$

$$\sum_{i,l} \{(D_{i,l} \odot Z_{i,l}^e) \odot R_i^e\} \rightarrow \text{Min}$$

$$\sum_j X_j^e = 1, \quad \sum_k Y_k^e = m \quad (2)$$

$$\forall i, l (X_i^e + Y_i^e) \cdot Z_{i,l}^e - Z_{i,l}^e = 0 \quad (3)$$

$$\forall j, k \notin \mathcal{P}_e | X_j^e = 0, Y_k^e = 0 \quad (4)$$

Eqn. (1)-(4) shows optimization formula for data entry deployment. This objective function includes three variables, $\mathbf{X}_j^e$, $\mathbf{Y}_k^e$, $\mathbf{Z}_{i,l}^e$, and their product, such as $\mathbf{X}_j^e \times \mathbf{D}_{j,k} \times \text{Tr} \mathbf{Y}_k^e$. Here $\text{Tr} \mathbf{Y}_k^e$ means transpose of matrix $\mathbf{Y}_k^e$. Because of the nonlinearity, the scale of the calculation is $O(n^2)$ where n is the number of edge servers. It is not preferable for the data deploy controller that uses this algorithm because the controller must determine the data arrangement following the changes of the data access pattern according to the travels of the data access clients.

$$\sum_j \{ \mathbf{W}_i^e \times \mathbf{D}_{i,j} \odot \mathbf{X}_j^e \} \rightarrow \text{Min} \quad (5)$$

$$\sum_j \mathbf{X}_j^e = 1, \quad \forall j \notin \mathcal{P}_e | \mathbf{X}_j^e = 0 \quad (6)$$

$$\bar{\mathbf{X}}_j^e \times \mathbf{D}_{j,k} \times \text{Tr} \mathbf{Y}_k^e \cdot \sum_i \mathbf{W}_i^e + \quad (7)$$

$$\sum_{i,l} \{ (\mathbf{D}_{i,l} \odot \mathbf{Z}_{i,l}^e) \odot \mathbf{R}_i^e \} \rightarrow \text{Min}$$

$$\sum_k \mathbf{Y}_k^e = m, \quad \forall k \notin \mathcal{P}_e | \mathbf{Y}_k^e = 0 \quad (8)$$

$$\forall i, l | (\bar{\mathbf{X}}_i^e + \mathbf{Y}_i^e) \cdot \mathbf{Z}_{i,l}^e - \mathbf{Z}_{i,l}^e = 0 \quad (9)$$

We propose an algorithm to determine the placement of data rapidly. In this algorithm, we define the writing process is the priority and determine the master data placement in the first step considering loads of data write access. And as the second step, we determine the locations of the replica data considering copying the replica data and read data access. By this separation of steps, we reduce non-linear terms from the objective function, and the calculation can be done in real-time.

Eqn. (5)-(9) shows the detailed formulation of the proposed algorithm. We determine the location of the master data by using Eqn. (5)-(6) and get the solution $\bar{\mathbf{X}}_j^e$. And we determine the location of the replica data by using Eqn. (7)-(9) using the result of the Eqn. (5)-(6), $\bar{\mathbf{X}}_j^e$.

IV. IN-NETWORK DIRECTORY CACHE

This section describes the in-network directory cache that improves the data access performance by accelerating the response time of directory access.

We propose the data deployment controller that arrange the placement of the data entry following the moving data access clients. That means every time when the data deployment controller changes the location of the data entry we need to search for the data entry using the directory service. Because of the data placement control, the frequency of the searching process is high. If the searching process takes time, the performance of data acquisition will be deteriorated.

Our proposed in-network directory cache functionality is implemented in a programmable node in the network between edge servers and cloud. This deployment structure reduces directory access latency as much as the latency of the wide-area

network between the edge servers and cloud. And processing time of directory service is reduced because the in-network directory cache processes the protocol frame as it is without terminating.

A. Frame format

Fig. 5(a) shows the frame format for the directory access. The protocol for directory access is implemented on UDP over IP over Ethernet. The protocol frame includes fields of TYPE, LEN, KEY, ID, and Payload. The TYPE field is for determining if the frame is for directory access request or response. The KEY field is the primary key of the data entry that is the target of location search. The ID is an identifier that is used to map the directory access request and response within the directory access client. And the payload is the field for storing the information of data location.

B. Functional blocks

Fig. 5 (b) shows the block diagram of the in-network directory cache. The incoming frame is sorted out by its source or destination UDP port number. L3/L4 Classifier identifies directory access frame by looking up the source or destination UDP port number. The frame with specific UDP port number, e.g., 33333, is the frame of directory access and is sent to the next block of the cache system. The other frame is sent to the routing process. Then TYPE Classifier in the cache system determines the type of the frame, request or response, by looking up the type field of the directory access protocol frame.

The cache system searches for the directory entry with the key that is extracted from the KEY field in the request frame. When there is a directory entry corresponding to the key, the cache system creates the response frame with data location information in the payload field. In the case of miss-hit, the cache system sends the frame as it is to the routing process, and the frame is sent to the original destination.

The cache system extracts the key and the data entry location information from the KEY field and payload in the response frame, respectively. The extracted information is registered in the hash table as a directory cache entry.

The proposed in-network directory cache is implemented using the software-defined router Click[4] and can work on general-purpose white box switches. It is possible to design this system scalable by using a multi-core processor such as FLARE[5] for better performance.

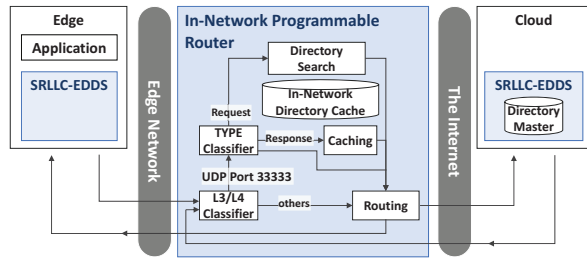
V. DELAY AREA-OF-INTEREST INDEX

This section defines the Delay Area-of-interest Index (DAI) that is a newly proposed index to evaluate geographically distributed data store from the viewpoints of low-latency and data locality.

DAI quantifies the effects of applying technology for the geographically distributed data store, and the value is large when the technology provides less latency for the broader area of services.

Fig. 6 shows the method for calculation of the DAI. We use performance space for defining the DAI. The vertical axis

ETH	IP	UDP	LEN	TYPE	KEY	ID	Payload
-----	----	-----	-----	------	-----	----	---------



DAI = $\frac{\text{A: Extended Area by applying the method}}{\text{B: Extensible area from the base line performance}}$

The diagram illustrates the system architecture across three layers:

- Cloud:** Contains a **Cloud Node** (with **Data**) and a **Directory Master**.
- Edge System:** Contains a **Router #1** (with **In-network Directory Cache**) connected to **Router #2** and **Router #3**.
 - Router #2** is connected to **Edge Node #7** and **Edge Node #3**.
 - Router #3** is connected to **Edge Node #8** and **Edge Node #6**.
 - Edge Node #9** is also connected to **Router #1**.
- Field:** Shows a road with cars. Vertical lines indicate the **Edge Node Cover Areas** for the edge nodes.

Fig. 6(b) and (c) show performances of sample methods for distributed data store, and X1-3 and Y in Fig. 7 show their data access patterns, respectively. We get extended areas from the baseline performance when applying these methods, A . And the DAI is calculated by Eqn. (10).

$$\text{DAI} = \frac{\text{A:Extended area by applying the technology}}{\text{B:Extensible area from the baseline performance}} \quad (10)$$

and the cloud. Each edge node and cloud nodes expose data access APIs and data store. Directory service for determining the location of the data entry keeps its master data in the cloud and directory cache in the edge nodes and programmable router node, e.g., Router#1 in Fig. 8. And in this evaluation, we do not deploy replica and only use master data.

We use a traffic model emulating state reading and writing from the connected vehicles for evaluation. We build vehicle movement models by using a traffic simulator SUMO¹. We suppose each vehicle sends its position, speed, acceleration, and the other information at regular intervals. Additionally, each vehicle gets the state information of vehicles around periodically. This traffic model is created supposing safe driving support service. We get location related traffic model for evaluation by combining the movement model and the traffic model.

¹<https://sumo.dlr.de/index.html>

TABLE I
SYSTEM SPECIFICATIONS

Items	Specifications
VM Host	Intel Core i7-4770, 32GB MEM
Cloud VM	CPU 2 Cores, 2GB MEM
Router1 VM	CPU 2 Cores, 8GB MEM
Edge, Router VM	CPU 1 Cores, 2GB MEM
Latency	Internet: 20ms, b/w Router 2ms
Map	1kmx1kmx5, 200mx200mx5
Vehicle Frequency	40/s, 8/s
Avg. velocity	36km/h
Communication Periods	Update:1s, Read within/between Edges: 1, 10s

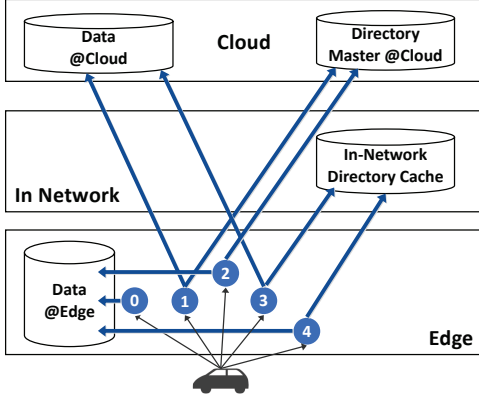


Fig. 9. Communication Pattern for in-network directory cache evaluation

The field where vehicles move around is divided into multiple coverage areas, and each edge server is in charge of each coverage area. The vehicle who wants to access the state information connects to the edge server that is responsible for the current location of the vehicle.

Fig. 9 shows data and directory access patterns occurring during the evaluation. Pattern#0 shows both directory entry and data entry is in the storage within the edge server where the data access client connects. Pattern#1 shows both the target directory entry and the data entry is in the cloud. Pattern#4 is the proposed one, and the directory entry is in the network node, and the data entry is in the edge server. The other patterns are data in the cloud and directory in-network for pattern#2, and data in the edge and the directory in the cloud for pattern#3.

C. Performance of the Deployment Algorithms

We show the performance of the proposed data deployment algorithm. Fig. 10 shows the calculation time and the value of the objective function for the optimal algorithm defined with Eqn. (1)-(4) and the proposed heuristic algorithm defined with Eqn. (5)-(9). We use open source modeling language *pyomo*², and solver *cbc*, *couenne*³ to solve the formulas. We use a traffic pattern of local read/write intensive. 80% of read/write traffic

²<http://www.pyomo.org/>

³<https://www.coin-or.org/>

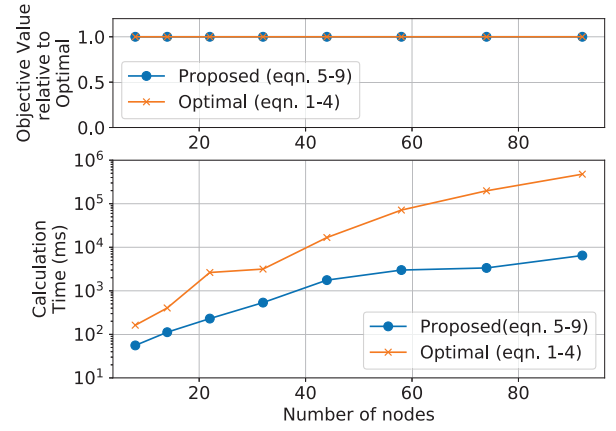


Fig. 10. Calculation time for deployment algorithms

is from an edge server and 20% is from the other remote edge servers.

When the number of nodes is under 40, the proposed algorithm can complete the calculation within 1 second, though the optimal algorithm requires over 10s (Fig. 10). And the proposed algorithm can derive the same objective function value as that of the optimal one. Each vehicle, as a data access client, changes the edge server that the vehicle belongs to every 5-25 sec, and it is preferable to decide the data location under 1 second considering the delay of a few seconds for the deployment of the data after the calculation. This result shows that our proposed algorithm can decide the optimal data arrangement following the change of the data access clients.

D. Evaluation Results

We show the effectiveness of the proposed dynamic data deployment mechanisms following the movement of the data access clients.

Fig. 11 shows changes in data access response time. The case (A) is that data entry is fixedly placed in the cloud. The data entry is located at the gateway to the Internet and in the middle of the edge network for the case (B) and (C), respectively. The case (D) and (E) are the cases where our proposed method is applied. And each edge server covers the area of 1km² for the case (D) and 200m² for the case (E). The charts show that LI-EDDS keeps the response time around 20ms following the movement of the data access clients. (A)-(C) are the data store where the data is fixedly located in the cloud, internet-gateway, in the edge network, respectively, regardless of the location of the data access clients. They provide poor response time, over 40ms. Even with the proposed method the response time is deteriorated around 40ms sometime in time series though the proposed method provides better performance most of the time. This is because clients must access to the directory to search for the new location of data at the first access after the proposed LI-EDDS re-arrange the location of data for better performance.

Fig. 12 shows the distribution of data access delays for the following cases. The proposed method (D) provides the

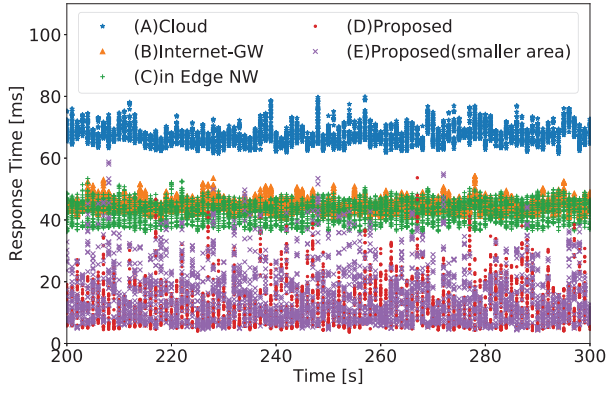


Fig. 11. Timeseries Delay

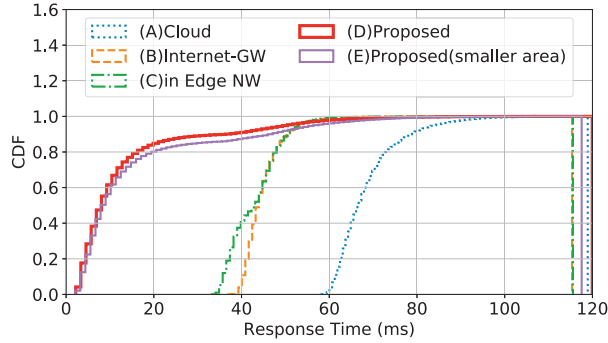


Fig. 12. Delay Distributions

average response time of 13ms thanks to the dynamic data deployment mechanisms following the movement of the data access clients. For fixed deployment, (A) the case of cloud deployment provides 66ms, 45ms for (B) the deployment besides the Internet gateway, and 43ms for (C) the deployment in the edge network.

By comparing the results of (D) and (E), we can observe the deterioration due to frequent data location control and directory search. In the case (E) the coverage area of each edge servers is smaller than that in the case (D). Optimal location of data entry changes more frequent when the coverage area of each edge server is smaller. Because of these conditions average response time for the case (E) is 14ms, although that for the case (D) is 13ms. This result indicates that reducing the delay for directory access is also important to provide moving data accessing clients with faster data access performance.

Next, we evaluate the performance of the proposed in-network directory cache. Fig. 13 shows the average response time for each data access patterns defined in Fig. 9. In the case (E) of evaluation above, there are more data accesses of the pattern (3) than in the case (D). We can reduce the average response time from 32ms of the pattern (3) to 9ms of the pattern (4) by applying in-network directory cache technology. In other words, we can enjoy quick data acquisition with 9ms if directory access request hits the cache entry in-network even when data location control and directory access querying the

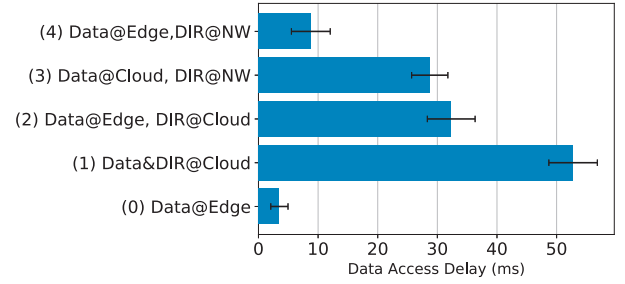


Fig. 13. In-Network Directory Cache

location of data entry occurs frequently.

VII. DISCUSSION

A. Scalability of Deployment Algorithm

The proposed algorithm achieves the same precision as the optimal one and reduces calculation time from 10s to 1s for the system with 40 nodes. But the calculation time for the system with 90 nodes is 7sec and is not sufficient for real-time data deployment. We need to deal with the MEC system as a target for applying our proposed method, and the total number of MEC servers will be 10,000 when they are installed beside SGW in the LTE system. For practical use, we adopt a technique that uses our proposed algorithm with node excluding method. The method excludes the edge servers that cannot be the target of the data deployment because the edge server is far from the location of the data access client. By applying our proposed algorithm after the exclusion, we can reduce the calculation time for the larger system and decide the location of the data in a real-time manner following the move of data access clients.

B. Applicability for the other traffic pattern

The proposed algorithm is the most effective for traffic with write-intensive. We need to replace the order of the calculation for the read-intensive traffic. For the read-intensive traffic, the locations of the replicas are calculated first. Then the location of the master data for writing is calculated.

C. DAI

This section evaluates the system using the proposed index DAI. We compare (D) proposed system performance with (B) fixed deployment at the Internet gateway, and (C) fixed deployment in the edge network. Fig. 14 (a) shows extended area on performance space, and (b) shows the data accessing routes for each case. The case (B) provides the average response time of 44.7 ms, and the case (C) provides that of 43.1 ms. The response time of the proposed method is 10.3ms when data access client connects to the edge server where the target data entry is stored. When accessing from the adjacent edge server of the edge server that keeps the data entry, the average response time is 12.8ms, and otherwise, the average response time is 13.6 ms.

Fig. 6 shows the evaluation result of the geographically distributed data store using the proposed DAI. Our proposed

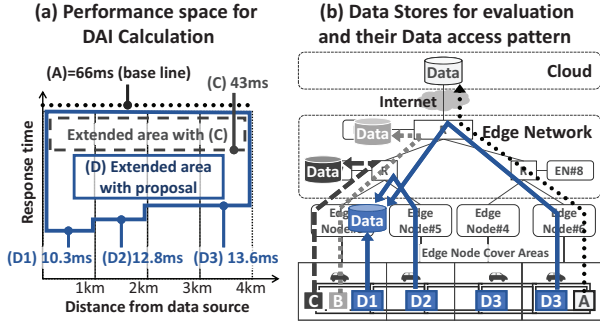


Fig. 14. Calculation of DAI

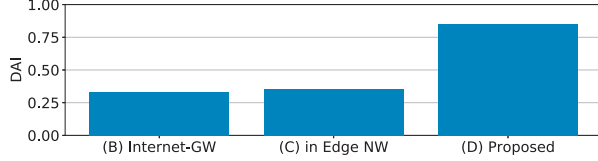


Fig. 15. Evaluation with Delay Area-of-interest Index (DAI)

method provides DAI=0.8, while the methods with fixed data deployment provide DAI=0.3.

This result means that our proposed DAI can evaluate the performance of the geographically distributed data store. And the result also shows that our proposed method provides better performance than the conventional methods quantitatively. If the ratio of edge local read/write is low, the performance of the proposed method (D) can be the same as the conventional methods (B) and (C). But the superiority of our proposed method is ensured because edge local data access cannot be zero for our supposing services, such as CPS for vehicles.

VIII. RELATED WORK

There are proposals about geographically distributed data store that use space-filling curves, such as Hilbert's curve [1][2]. Those works map a geographical location to a value on a space-filling curve. And the value is used as a key for distributed data store. By this hashing, data that are generated neighborhood will be stored in geographically close edge servers. But geographic proximity does not always guarantee low latency because the latency depends on distance on network topology of the distributed data store.

Globally distributed data store is also proposed in [6][7]. They provide strong consistency even if their clients update data in a distributed manner. Their target is distributed data centers, not edge distributed servers, and the data is distributed with hash. Because of those mechanisms, the location of data cannot be controlled explicitly to achieve low-latency and data locality. SwiftCloud [8] proposes a distributed data store with causal consistency that distributes local data on clients and allows updates in a distributed manner for quick data access. But it does not consider the location of the data, especially when multiple clients want to update the data.

NetCache [3] is an in-network cache technology for a distributed key-value store (KVS). It is a technology to improve

the read latency utilizing cache data on the on-chip memory in programmable switches. The size of the value in KVS is limited because the size of the on-chip memory is limited. And the performance of data update can not be improved as the update message need to be sent to the fixed node where the master entry is stored. On the other hand, this work is implemented on a programmable switch [9] and we can achieve higher performance by combining our proposal with the hardware like FLARE [5] or P4 [9].

As for the evaluation index for the geographically distributed data store, FogStore [2] proposes a notion "Context of Interest" that defines the geographical range where the data should be consistent. But the Context of Interest is only a requirement for the range of data consistency and can not be used for evaluating the system.

Regarding optimal placement technology for processing and data, [10] assumes a system model with hierarchical edge servers and controls the location of processing considering the proximity in the network and computing resource capacity of edge servers. But the placement of data and stateful service is not considered.

IX. CONCLUSION

Our contributions in this paper are fourfolds. First, we propose a Layer-Integrated Edge Distributed Data Store. The data store provides low-latency data access with fulfilling consistency for stateful services, and data locality simultaneously. Second, we propose in-network directory cache functionality that accelerates the search of distributed data. Third, we develop an algorithm that decides the location of data to provide low-latency data access. The algorithm achieves the same precision as the optimal algorithm reducing the calculation time from 10s to 1s for the system with 40 nodes. And our evaluation result shows that the data arrangement provided by the proposed algorithm enables data access clients to access data with a latency of 13ms, excluding wireless communication delay even when the client is moving. And in-network directory cache reduces the response time to 9ms on average. Finally, we define an index that quantifies the performance of the geographically distributed data store. Our proposed method provides DAI=0.8, while the methods with fixed data deployment provide DAI=0.3. This result means that our proposed DAI can evaluate the performance of the geographically distributed data store. And the result also shows that our proposed method provides better performance than the conventional methods quantitatively.

For future work, we will evaluate the performance overhead caused by data transfer for optimal data location control, the performance improvement by the effect of replica placement, and the performance overhead for consistent data write when race condition between data write accesses occurs. We will study the deployment algorithm to deal with the larger-scale system and the other traffic pattern. We will also study DAI considering communication and storage costs and in-network directory caching algorithms utilizing hierarchical network topology.

REFERENCES

- [1] Pandey, A. Kipf, D. Vorona, T. Mhlbauer, T. Neumann, and A. Kemper, "High-Performance Geospatial Analytics in HyPerSpace," ACM SIGMOD, pp. 2145–2148, 2016.
- [2] H. Gupta and U. Ramachandran, "FogStore: A Geo-Distributed Key-Value Store Guaranteeing Low Latency for Strongly Consistent Access," DEBS 2018, pp. 148–159, 2018.
- [3] X. Jin et al., "NetCache: Balancing Key-Value Stores with Fast In-Network Caching Key-value stores power online services," SOSP 17, 2017.
- [4] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek, "The click modular router," ACM Trans. Comput. Syst., vol. 18, no. 3, pp. 263–297, Aug. 2000.
- [5] Akihiro Nakao, "FLARE: Open Deeply Programmable Network Node Architecture Extending SDN, " MPLS 2012, Wasington DC, October 2012.
- [6] A. Verbitski, et al., "Amazon Aurora: Design Considerations for High Throughput Cloud-Native Relational Databases," Proceedings of the 2018 International Conference on Management of Data - SIGMOD '18, pp. 789–796, 2018.
- [7] James C. Corbett, et al., "Spanner: Google's Globally Distributed Database," ACM Trans. Comput. Syst., vol. 31, no. 3, pp. 8:1–8:22, August 2013.
- [8] N. Penguica et al., "SwiftCloud: Fault-Tolerant Geo-Replication Integrated all the Way to the Client Machine," 2014 IEEE 33rd International Symposium on Reliable Distributed Systems Workshops, pp 30–33, Oct. 2014.
- [9] Bosshart, Pat and Daly, Dan and Gibb, Glen and Izzard, Martin and McKeown, Nick and Rexford, Jennifer and Schlesinger, Cole and Talyco, Dan and Vahdat, Amin and Varghese, George and Walker, David, "P4: Programming Protocol-independent Packet Processors," SIGCOMM Comput. Commun. Rev., vol. 44, no. 3, pp. 87–95, July 2014.
- [10] Q. Fan and N. Ansari, "Workload Allocation in Hierarchical Cloudlet Networks," IEEE Commun. Lett., vol. 22, no. 4, pp. 820–823, 2018.