DECS: Collaborative Edge-Edge Data Storage Service for Edge Computing

Fuxiao Zhou and Haopeng Chen(✉)

Shanghai Jiao Tong University, Shanghai, China
{zhoufuxiao,chen-hp}@sjtu.edu.cn

Abstract. With the development of IoT and 5G, data are generated by the numerous smart end devices at each moment. Simultaneously, as the improvement of the hardware's performance, computing and storage are partly transferred to the edge of the Internet. However, the core cloud and massive data centers are still responsible for management and coordination. In more and more local-area and small-scale scenarios such as a parking lot, an office building, or a college campus, these scenarios also need the edge nodes to offload computing and storage tasks. Moreover, in order to decrease costs and be lightweight, these scenarios need to decouple with the core cloud partly. In this paper, we proposed a collaborative edge-edge data storage service called DECS for edge computing in local-area scenarios. DECS can make the edge nodes collaborate with others. Such as trade-off to pick the most appropriate edge node to offload storage or computing tasks. DECS can also replicate data or generate forwarding rules in advance by predicting data's popularity proactively.

In this paper, we evaluated DECS at two real scenarios compared with state-of-the-art research. The experiment results proved that DECS was more suitable for the local-area edge cluster. Which lowered the access latency, saved the total bandwidth, and improved the resource utilization of the whole edge cluster.

Keywords: Collaborative storage · Storage as a service · Edge computing · Small-scale · Local area · Resource management

1 Introduction

With the development of 5G and the Internet of Things (IoT), the amount of data is more substantial. Meanwhile, the devices between the cloud and the end devices (IoT devices, mobile phones, or laptops) are getting more powerful, such as base stations, gateway servers, or routers. Fog computing has been researched widely in recent years to solve the problem of how to effectively utilize and manage resources of the devices between the core cloud to the end devices [19].

The traditional Internet logic architecture model has been approximately divided into three layers [19], including the cloud layer, which is composed of

H. Gao et al. (Eds.): CollaborateCom 2020, LNICST 349, pp. 373–391, 2021.
https://doi.org/10.1007/978-3-030-67537-0_23

large data centers, such as Amazon's AWS. The fog layer consists of base stations, RANs, gateway servers, or routers. Moreover, the end layer contains IoT devices such as sensors, monitors, or mobile phones, laptops.

Generally speaking, the scenarios of edge computing are wide-range geographical such as metropolis computing, inter-metropolis computing, or smart city, and which can be seen in Fig. 1(a). In these scenarios, the core cloud and edge devices provide computing and storage resources for connected vehicles, health care, and smart delivery. However, in some small-scale and local area scenarios such as a college campus, a parking lot, a hospital, an industrial park even an office building. Which also need edge computing and multi-access edge computing (MEC) to provide computing and storage resources. These resources on edge nodes can process offloaded tasks from IoT devices or mobile phones.

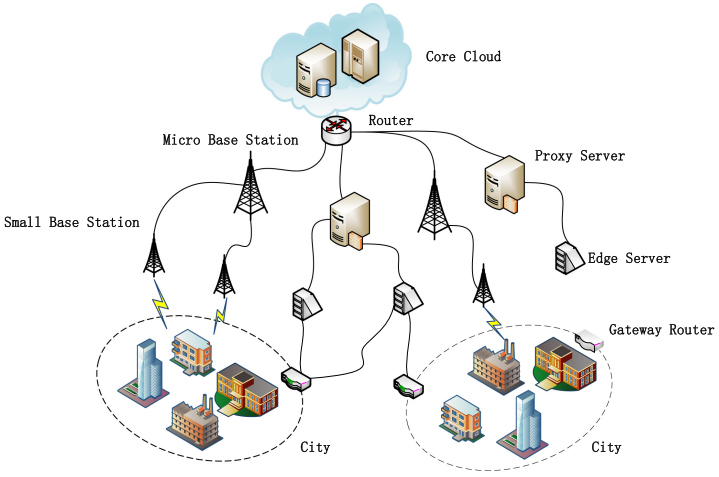
For example, the scenario in an office building, which contains many IoT devices such as video surveillance monitors distributed on every floor, and many mobile phones used by many people worked in this building, which can be seen in Fig. 1(b). Each video monitor or sensor can offload tasks to some closer edge node, such as a router. Moreover, the task may be a realtime computing task such as video violations identification. The edge node which executes the identification task will analyze the video data and utilizes machine learning to identify the wrong behavior such as smoking and stealing. On the other hand, the users who have mobile phone also can offload tasks to a closer edge node to storage or read some data, and the task can be a computing task too.

Traditional distributed databases, file systems such as cassandras [4], or HDFS [8] can not be a solution directly to the local area and small-scale scenarios mentioned above because that these schemas do not consider the features of edge computing. Due to the edge nodes have limited resources, and different edge node's performance may have considerable differences with others. Moreover, the data stored inside the edge node are heterogeneous (may be structured or unstructured). Besides, the storage involved in edge computing considerate the distance and geographically location. Therefore, in this paper, we abstracted the storage as a service and proposed a collaborative edge-edge service. The storage service can be provided to end devices to write or read data. Meanwhile, this service can be an essential service to support other computing services or be a sub-service in more extensive services.

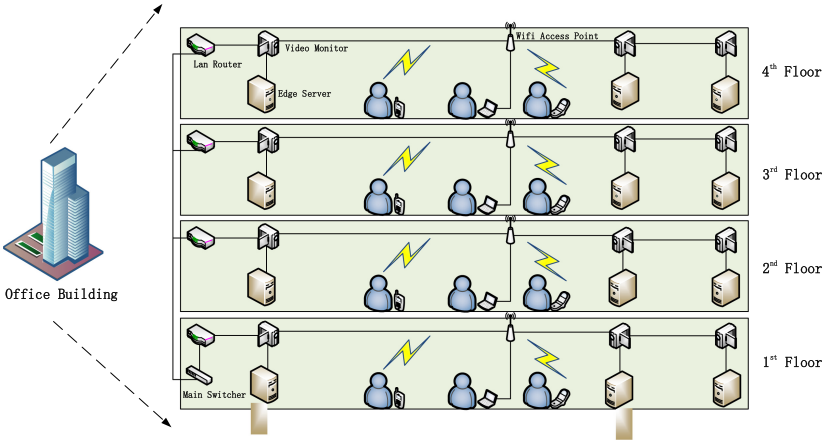
In this paper, we proposed a collaborative edge-edge data storage service for edge computing in the local area and small-scale scenarios called DECS. DECS only runs on the edge cluster, which is closer to end devices. DECS contains some modules such as Data Storage Module, Metadata Module, Data Processing Module, Trade-off Module, Prediction Module.

This paper mainly made these contributions as follows:

(1) This paper abstracted the storage as a service and proposed a collaborative edge-edge data storage service called DECS to utilize the resources in the edge cluster effectively.
(2) This paper designed and implemented the sub-modules of DECS, including the Metadata Module, Data Storage Module, Data Processing Module, Trade-off Module, and Prediction Module.



(a) Three-Layer Edge Architecture In Wide Area



(b) An Example of Local Area Scenario: An Office Building

Fig. 1. Wide area and local area scenarios for edge computing

(3) This paper did experiments and evaluated DECS at two real scenarios compared with state-of-art research. This paper gave explanations and analysis of the experiment results.

The rest of this paper is organized as follows: Sect. 2 introduces the design of DECS, including the overall framework and every sub-module. Section 3 introduces the implementation and evaluation of DECS and its sub-modules. Section 4

mainly introduces the related work, research, and state-of-art. Section 5 is the final section and which introduces the conclusion of this paper.

2 Design

In this section, we will introduce the design of DECS. Firstly this section will describe the overall framework of DECS. This section will then introduce the sub-modules of DECS, including the Metadata Module, Data Storage Module, Data Processing Module, Trade-off Module, Prediction Module.

2.1 Overall Framework of DECS

In the local area and small-scale scenarios such as a building, a park, and a college campus, the infrastructure is relatively lightweight. Therefore the DECS is applicable to edge cluster, which is closer to end devices. DECS is organized as a micro-service running in every edge node of the edge cluster.

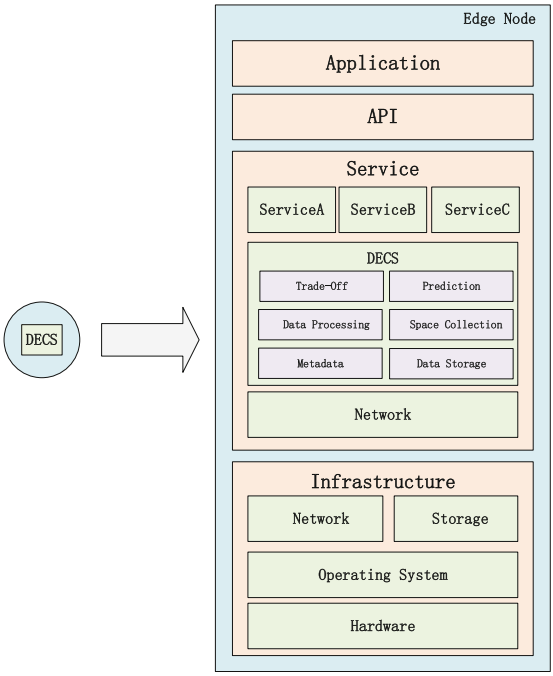


Fig. 2. Inner architecture of edge node and DECS

Edge nodes connect with each other by network cable, and DECS is run on every edge node. End devices are split into two types: the first one is IoT

devices, which connects edge nodes by network cable usually. IoT devices can offload real-time computing tasks to the appropriate edge node, typically such as target recognition. The second type is mobile phone or laptop carried by users, and it is a MEC architecture, multi-users can access the same edge node. The users are movable to every position of the specific local area and small-scale scenario. Mobile phones can access the edge node by wifi or 4G/5G network, and the laptops can access the edge cluster by wifi or network cable.

DECS contains some sub-modules which can be seen in Fig. 2. There are the metadata module, data storage module, data processing module, prediction module, trade-off module, and space collection module. DECS is in the service layer, and there are other services in this layer. Part of them are underlying fundamental services such as network. Moreover, others are top-level and specific services such as some computing services offloaded from users, or inherent services attached to some applications. DECS can be seen as an underlying service because it integrates the fundamental storage as a sub-service. On the other hand, DECS contains some top-level services such as data processing sub-service.

Table 1. Sub-modules of DECS instruction

Module Name	Description
Metadata Module	This module is responsible for managing all the metadata and status of this edge cluster and synchronized meta-information between nodes by the Gossip protocol
Data Storage Module	This module is an underlying module to process the write and read operation received from the storage task sent by end devices
Data Processing Module	This module can execute some computing tasks, especially the data processing task in real-time or asynchronously, and only return the result
Trade-off Module	This module is responsible for weighing the cost against latency or resource utilization in the task offloading or data storage in the whole cluster
Prediction Module	The role of this module is to predict the appropriate place for task offloading and predict the possible access point to migrate or replicate data proactively in order to lower the latency of the whole cluster

The concise introduction of sub-modules in DECS can be seen in Table 1, the rest of this section will introduce every sub-module in detail.

2.2 Metadata Module in DECS

Metadata Module is a core sub-module in DECS. Each edge node shares its status, such as stored data meta-information, access information, machine status information, and resource utilization information to the whole cluster. When an end device accesses some edge node to get data, the edge node will first check its metadata to get whether the wanted data in this node. If not, this node will refer to metadata to get which node has the wanted data and forward the request to the target node.

Table 2. Definition and pattern of metadata

Name	Pattern
liveNodes	[nodeIp,*]
availableSpace	[{nodeIp:space},*]
nodeHasWhatFile	[{nodeIp:[file,*]},*]
nodeCpuInfo	[{nodeIp:cpuInfo},*]
whoStoreFiles	[{devIp:[file,*]},*]
hotNodeForSpecificDev	[{devIp:[nodeIp:[period:acc,*],*]},*]
forwardingRule	[{devIp:[nodeIp,*]},*]
fileStoredWhere	[{file:[nodeIP,*]},*]
fileAccessFromNodeIpEachPeriod	[{file:{NodeIp:[{period:acc},*],*}},*]
filePopularityFromNodeIpLastPeriod	[{file:[{nodeIp:pop},*]},*]

In DECS, we defined some types of metadata to represent each edge node's status, the shared information of the whole edge cluster, and the metadata of stored data. The specific definition and representation of metadata can be seen in Table 2. Such as the *liveNodes* represents the number of live nodes of the edge cluster. Other metadata will be explained in the remainder of this paper when used.

2.3 Data Storage Module in DECS

The data storage module is fundamental and essential in DECS. This module can be accessed individually or supporting the upper sub-module such as the data processing module. The data storage module provides two operations for users as basic operations, which are **write operation** and **read operation**, each operation contains the call to the trade-off module and prediction module.

The process of store data to the edge cluster by DECS can be seen in Algorithm 1, the type of data can be divided into two types as **temp data** and **regular data**. The difference between temp data and regular data is their generated source. Due to the data from IoT devices will be provided to the real-time task usually. After the task completed, the data will be discarded in most cases and only store the computed result. To save limited space in edge nodes, we will

Algorithm 1. The process of write operation.

Require:
End device's IP, ip_f
The type of data wanted to be written, t
Default access the close edge node, ip_c
The file name which can uniquely identify the file, $name$
The binary stream of file wanted to be written, $data$;
Ensure:
Whether the data is written successfully, $result$.

```
1: liveNodes ← GetLiveNodes()
2: threshold ← GetStorageThresh()
3: closeNode ← GetCloseNodeInfo(ip_c)
4: exist ← CheckExist(name)
5: if exist == True then
6:    return False
7: end if
8: UpdateMetadata(data, name, ip_f, t)
9: forwardRule ← QueryMeta(ip_f)
10: if forwardRule != Null then
11:    forwardNodes = GetFwdNodes(forwardRule)
12:    ip_t ← PickNodeW(forwardNodes, data, ip_c, t)
13:    UpdateMetadata(data, name, ip_t, t)
14:    return Store(ip_t, data, name, t)
15: end if
16: ip_t ← PickNodeW(liveNodes, data, ip_c, t)
17: UpdateMetadata(data, name, ip_t, t)
18: return Store(ip_t, data, name, t)
```

mark this kind of data as temp data, which only lives a short time. The regular data is generated from mobile phones or laptops of users, and this kind of data will be stored for a long time unless some extreme cases.

In the process of storing data, this module checks whether this data existed by metadata *fileStoredWhere* firstly at line 4. Then this module will updata the metadata (*availableSpace*, *whoStoreFiles*). After the write operation, this module updates the metadata once again at the end at line 17, which updates the *fileStoreWhere* after tradeoff to record where the file is written.

At the beginning of each time period, the prediction module will predict the storage pattern for the devices that frequently store data in the previous periods. The predicted results will be recorded in the metadata *forwardingRule*, and the storage pattern can help devices to write data to some nodes that most likely to be accessed by other end devices. During the process of the write operation, the storage module will check the *forwardingRule* in metadata at line 11 to proactively forward this storage request to the target node in advance.

The process of write operation calls the trade-off module many times, such as *PickNodeW* at line 12, which needs to pick the most appropriate node to write

Algorithm 2. The process of read operation.

Require:
The type of data wanted to be read, t
Default access the close edge node, ip_c
The file name which can uniquely identify the file, $name$;
Ensure:
The binary stream of file wanted to be read, $result$.

```
1: liveNodes ← GetLiveNodes()
2: existClose ← CheckExistClose(name, closeNode)
3: if existClose == True then
4:     ip_t ← ip_c
5: else
6:     existNodes ← FindExistNode(liveNodes, name)
7:     if existNodes.lenth == 0 then
8:         return Null
9:     else
10:        ip_t ← PickNodeR(existNodes, name)
11:    end if
12: end if
13: updataMetadata(name, ip_t, t)
14: if t == Temp then
15:    return ReadAndDelete(name, ip_t, t)
16: else
17:    return Read(name, ip_t, t)
18: end if
```

data from the candidate nodes. It is necessary to filter the candidate nodes first to exclude the node that has no space to store the data.

The read operation process is relatively simple, which can be seen in Algorithm 2. The module checks whether the accessed edge node has the wanted data firstly. If not, this module will query the metadata *fileStoreWhere* to get the list of edge nodes that contain the wanted data at line 6. Regular data may exist in more than one node, and the reason for this case is that it is popular for this cluster. The prediction module proactively replicates this data to the other nodes in advance. However, the temp data only has one. When the wanted data has more than one replicas, this module will call the *PickNodeR* of the trade-off module at line 10 to determine to pick an appropriate node to read. If the desired data are temp data, these data will be removed at line 15 from the edge cluster to keep space enough.

2.4 Data Processing Module in DECS

The data processing module is a critical top-level sub-module in DECS. Its role is executing the computing tasks that are offloaded from the IoT devices or the users. The data processing task processed in this module can be split into two types, namely **with-data task** and **without-data task**, the difference between them is whether the task with or without accompanying data.

The differences can be seen in Table 3. If a requested task with data means this data processing task needs to process the data attached, the accompanying data will be written as temp data. The without-data task will process the existing data, and this kind of task will not modify the raw data.

Table 3. Differences between the two types of tasks

	With-Data Task	Without-Data Task
Request With Data	Yes	No
Modify Data	May modify carried data	No
End Target	IoT devices or users	Users
Data Type	Temp data	Regular data
Main Resource Needed	Computing Power	Data Resources

The process of processing with-data task can be seen in Algorithm 3. This module needs the supports of the data storage module to write or read temp data. This module will gather real-time CPU idle utilization of the candidate nodes and computes the real-time computing power for each. Then call the function *PickNodeP* in the trade-off module at line 10 to pick the best node to process the with-data task.

Algorithm 3. The process of processing with-data task.

Require:
The data processing task, *task*
The accompanying data with this task, *data*
The data storage module, *module*;
Ensure:
The computed result of this task, *result*.
1: $canStoreNodes \leftarrow module.GetCanStoreNodes(data)$
2: **for** *node in canStoreNodes* **do**
3: $cpuInfo \leftarrow GetCpuInfoFromMeta(node.GetIp)$
4: $cpuIdle \leftarrow GetCpuIdle(node.GetIp())$
5: $idleComputePower = cpuIdle \cdot cpuInfo.GetFreq()$
6: $canComputeNodes.Add(node, idleComputePower)$
7: **end for**
8: $nodes \leftarrow canComputeNodes \cap canStoreNodes$
9: **if** $nodes.lenth$!= 0 **then**
10: $bestNode \leftarrow PickNodeP(nodes)$
11: $module.storeTemp(data, bestNode)$
12: **return** $Process(bestNode)$
13: **end if**

The process of processing without-data task is similar to the process of processing the with-data task. The difference between them is that the latter firstly

traverses all nodes which can store the temp data and compute their computing power to pick the most appropriate node. However, the former only traverses the nodes that have the desired raw data. The rest of them are the same.

2.5 Trade-Off Module in DECS

The trade-off module is an auxiliary module in DECS, and the role of it is decision-making. This module needs to decide where to write or read data and where to offload computing tasks according to the distance, bandwidth, computing power, or free space. This module is called by the data storage module and data processing module, which can be seen in the algorithms mentioned above. The following will introduce the specific details of each function of this module.

The function *PickNodeW* at line 12 of Algorithm 1 means picking the most appropriate node to write data, the specific process is formalized as follows:

In Eq. (1), the l means the length (MB) of data, the s means the real-time network speed (MB/s) from the accessed node to the node$_i$, and the t_i means the estimated time (s) to transfer the data from the accessed edge node to the node$_i$.

$$t_i = \frac{l}{s} \tag{1}$$

This module trades off the most appropriate node to write data by comparing the score of s_i generated from Eq. (2). In the equation, the r_i means the real-time ratio of free space in node$_i$, and $r_i \epsilon [0, 1]$. The n_i means the predicted access amount after normalizing, and $n_i \epsilon [0, 1]$. The T_c is a constant, which means the transferring time (s) between the end devices and the accessed edge node. The α and β are the hyperparameter, which can be adjusted to change the weight, and $\alpha, \beta \epsilon [0, 1]$. The edge node with the highest score will be chosen as the target node.

$$s_i = \frac{\alpha \cdot r_i + (1 - \alpha) \cdot n_i}{\beta \cdot (1 + \frac{t_i}{T_c})} = \frac{T_c \cdot [\alpha \cdot r_i + (1 - \alpha) \cdot n_i]}{\beta \cdot (T_c + t_i)} \tag{2}$$

The function *PickNodeR* at line 10 of Algorithm 2 means picking the most appropriate node to read data, the role of this function is load balancing, and the specific process is formalized as follows:

$$score_i = \frac{\alpha \cdot (1 - p_i)}{\beta \cdot (1 + \frac{t_i}{T_c})} = \frac{T_c \cdot [\alpha \cdot (1 - p_i)]}{\beta \cdot (T_c + t_i)} \tag{3}$$

In Eq. (3), the p_i is the predicted access amount after normalizing which the access will request node$_i$,and $p_i \epsilon [0, 1]$, the higher the predicted access amount, the lower the score of node$_i$, which can balance loads. Other parameters have the same meaning as the previous equation, and the edge node with the highest score will be chosen as the target node.

The function *PickNodeP* at line 10 of Algorithm 3 means picking the most appropriate node to offload the data processing task, and the specific process is formalized as follows:

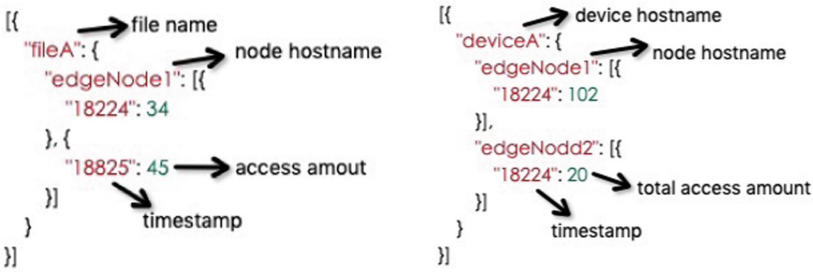
$$s_i = \frac{\alpha \cdot r_i + (1-\alpha) \cdot c_i}{\beta \cdot (1 + \frac{t_i}{T_c})} = \frac{T_c \cdot [\alpha \cdot r_i + (1-\alpha) \cdot c_i]}{\beta \cdot (T_c + t_i)} \tag{4}$$

The Eq. (4) is similar as the Eq. (2) in form, however, the meaning is different. In Eq. (4), the c_i means the free CPU computing power after normalizing. But in Eq. (3), the n_i means the predicted access amount after normalizing, and the $c_i \epsilon [0, 1]$.

2.6 Prediction Module in DECS

The prediction module can proactively replicate hot data in advance and predict the location that most likely to be accessed according to the user's storage pattern to write data by machine learning. This module runs at the beginning of each time period periodically and automatically, which is not called by any other module. The following will introduce the specific details of this module.

At the beginning of each time period, this module will check the metadata to acquire the access information of the files in this edge cluster of the last time period. The length of one period may be one day or be custom set. Each write operation (not temp data) from end device will be recorded in the metadata such as *whoStoredFiles*, and each read operation (not temp data) will increase the access amount record in metadata such as *fileAccessFromNodeIpEachPeriod*.



(a) *fileAccessFromNodeIpEachPeriod* (b) *hotNodeForSpecificDec*

Fig. 3. Examples of metadata helping to predict

The example of the metadata *fileAccessFromNodeIpEachPeriod* can be seen in Fig. 3(a). Which means the requests from end devices to access *fileA* hit the edge node *edgenode1* 34 times in *time period 18224* and 45 times in *time period 18225*. Another example is the metadata *hotNodeForSpecificDec* can be seen in Fig. 3(b). Which means the requests to access files stored by *deviceA* hit edge node *edgenode1* 102 times totally in *time period 18224*, and hit *edgenode2* 20 times in *time period 18224*. These two metadata can help to predict the

access amount in the current time period, and the predicted result can decide the metadata *forwardingRule* to forward the storage request during each write operation and decide whether replicate the data to a hot node.

At the beginning of each time period, when this module predicts the most appropriate node to form the *forwardingRule*. This module gets a total access amount list for each period in each edge node for the files stored by the specific device, and the results are organized as a triple <*deviceIp, edgeNodeIp, accessRecordEachPeriod*>. If the length of *accessRecordEachPeriod* is too short or the last access amount of *accessRecordEachPeriod* is too lower, this item will be excluded and not be predicted, or else, this model will train an LSTM model using the list *AccessRecordEachPeriod* as training data to predict the access amount of node *edgeNodeIp* for the device *deviceIp*. If the predicted access amount greater than the threshold, this item will be recorded in the metadata *forwardingRule*. When the same device stores file again, the request will be forwarded to the hot edge node, and the data will be stored in the selected edge node. The thresholds mentioned above are the hyperparameters and can be adjusted according to the actual situation.

When this module predicts whether to replicate hot data, this module will acquire the metadata *fileAccessFromNodeIpEachPeriod* to get the access amount for each file at each edge node, and the result also organized as a tripe <*fileName, edgeNodeIp, AccessRecordEachPeriod*>. The rest of process of predicting is the same as above, this model will train an LSTM model for the eligible list *accessRecordEachPeriod* and predict the access amount in node *edgeNodeIp*, if the access amount more significant than the threshold, then will be decided whether replicating data to this hot edge node.

$$\begin{cases} s_r = \frac{r_i \cdot T_c}{e^{n+1} \cdot (T_c + t_i)} \\ s_f = \frac{T_c}{p_i \cdot e \cdot (T_c + t_i)} \end{cases} \tag{5}$$

Equation (5) is to trade-off whether replicating data or just forwarding the read requests simply, s_r means the score of replicating data, r_i means the free space utilization of selected node $node_i$, and n is the existing amount of replicas of this data. Moreover, s_f means the score of forwarding read requests, and p_i is the predicted access number of this data in node $node_i$ in the current period. The prediction module can make decisions according to the scores calculated by Eq. (5) in the trade-off module.

3 Implementation and Evaluation

In this section, we will introduce the implementation and performance evaluation of proposed DECS, including the experiment parameters, experimental data, and experiment results analysis.

3.1 Implementation

We implemented the collaborative edge-edge data storage service DECS based on the *Spring Cloud* micro-service framework, and each DECS run as a service in

each edge node. Moreover, each edge node was running on the Ubuntu operating system. We limit the CPU cores, memory size, bandwidth, and disk size differently at each edge node to simulate the practical heterogeneity. In each edge node, we utilized the Redis to store the metadata, and the Redis runs at cluster mode can synchronize information with others using the gossip protocol. DECS provided its service using the RESTful API, and end devices such as IoT devices or smartphones can access these RESTful API by HTTP communication.

3.2 Experimental Parameters and Data

In this experiment, we used fifteen computers with the CPU Intel(R) Xeon(R) E5-2620 v3 with main frequency 2.40 GHz, and this CPU had eight cores. The computer has 32G memory and 2 TB disk space. These fifteen computers are acting as the edge devices and run our data services DECS and other services. In this experiment, to store the heterogeneous data, we stored the data as file, and this paper only considered the data's size, ignoring the data's content and semantics.

All experiment data were generated from the end devices, the end devices, including the video surveillance devices such as *Hikvision DS-2CD3T45(D)-I3/I5/I8*, smartphones such as *iphone 11*, *Mi 8*, and laptops such as *Macbook Pro*. The data generated by the end devices can be many types such as image, text video, music, or others.

Table 4. Edge node types and resource limit

	Small node	Medium node	Large node
Amount	7	5	3
CPU cores	2	4	8
Memory (GB)	8	16	32
Disk size (GB)	100	500	1000

We divided the edge nodes into three categories according to the performance, namely small, medium, and large nodes, and the detailed parameters can be seen in Table 4.

Table 5. Experiment parameters

Parameter	Value
Experimental time (hours)	30
Size of each time period (hour)	1
Epoch of LSTM	100
Batch size of LSTM	1
Step size of LSTM	5
Layers of LSTM	2
Neurons of input layer	16
Neurons of output layer	1

The Table 5 are the detailed experiment parameters. This experiment was running lasting 30 h, and each time period's size was set to one hour. Other parameters were the hyperparameters of LSTM.

3.3 Experimental Scenarios and Applications

- **Path Fitting Application:** We used this application as an experimental scenario at a parking lot. In this scenario, end devices were IoT devices, mostly such as video surveillance or sensors. This application needed computing and storage resources.
- **Edge Dropbox Application:** We used this application as an experimental scenario at our university campus. In this scenario, end devices were mobile phones or laptops used by users. This applicable needed the storage resource.

These two applications ran at the edge devices mentioned above.

3.4 Baseline

In this experiment, we compared DECS with two baselines:

- **No Service:** The first one is that no data service runs on the edge node, which means each edge node does not collaborate to store.
- **ElfStore:** The second one is ElfStore, a resilient data storage service proposed by Sumit Kumar Monga et al. for federated edge and fog [15]. ElfStore also needs collaboration between edge nodes to store data.

3.5 Performance Evaluation

Firstly, we evaluated the data storage module and prediction module in DECS. For the write operation, the experiment result can be seen in Fig. 4. The left figure represents the ratio of forwarded write requests over time. The right figure represents read latency to access the data written by the same end device over time. The result shows that the ratios of forwarded write requests of DECS and ElfStore were increased, and the average read latency of DECS and ElfStore was decreased over time.

These results proved that DECS could build forwarding rules accurately for this end device when the same end device frequently writes data. Which can record this end devices' storage pattern, and forward this end device's write request to the edge node which is most likely to be accessed later. The results proved that DECS forwarded writing operations proactively in advance to decrease the read latency of the edge cluster effectively.

Then we evaluated the read operation in DECS, and the results can be seen in Fig. 5. These two subfigures represented the read latency of the path fitting application and the edge dropbox application. Each subfigure can be split into two parts, namely before replicating and after replicating. The experiment result proved that the prediction module in DECS could replicate the hot data to the

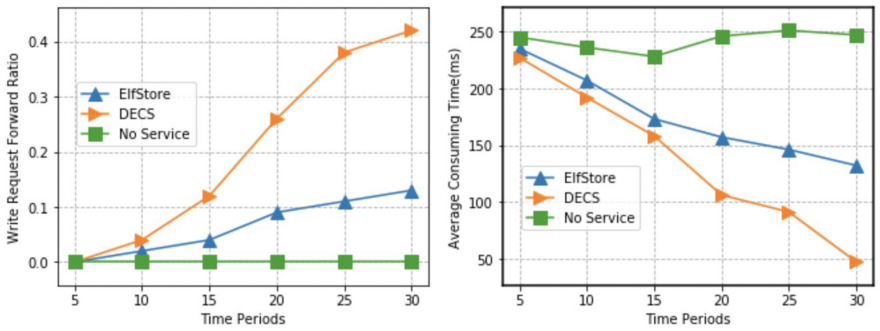


Fig. 4. Read latency for the same writer's data

target edge node in advance, proactively referencing the predicted access amount. In the path fitting application, most of the data were temp data, these temp data was not be replicated, so the improvement of reading performance was not evident. In the edge dropbox application, most of the data were regular data, and the read latency for hot data decreased evidently for DECS after replicating or migrating hot data.

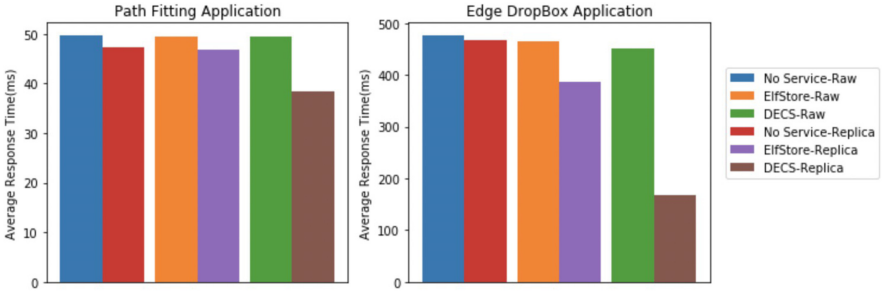
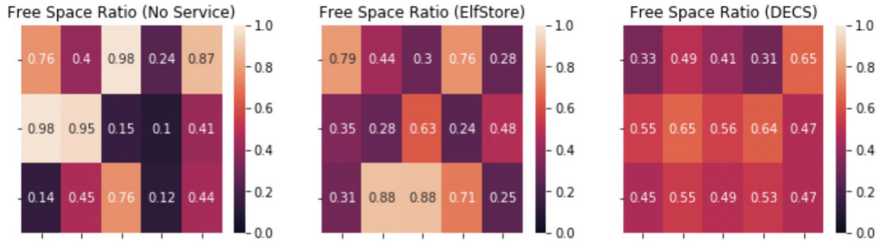


Fig. 5. Read latency before and after replicating or migrating data

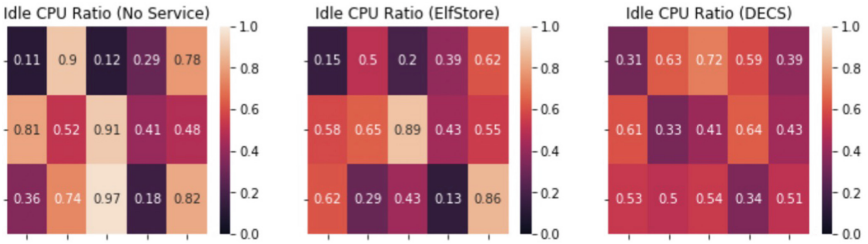
Then we evaluated the trade-off module and data processing module in DECS. The experiment results can be seen in Fig. 6. We collected the free disk space ratio and idle CPU ratio snapshot of all the fifteen edge devices in this experiment.

The Fig. 6(a) represents the free disk ratio collected after all time periods of the experiment. This result proved that the trade-off module could pick edge node which with more free disk size to store data. The Fig. 6(b) shows the idle CPU ratio snapshot of all the fifteen edge nodes at a random time. We can see that when the data processing module process the offloaded tasks, the trade-off module can pick the freest edge node to execute the task. This result proved

that the trade-off module could pick the edge node that idle CPU utilization is lower than others.



(a) Free Disk Space Ratio Heatmap



(b) Idle CPU Ratio Snapshot Heatmap

Fig. 6. Free disk and idle CPU snapshot heatmap

These experimental results proved that DECS could better manage and coordinate all the edge nodes of the whole cluster, decreased the read latency, and saved the bandwidth. DECS was compatible with all types of data generated by end devices. Moreover, DECS can be adaptive to heterogeneous performance of edge nodes, offload storage, and computing tasks more evenly.

4 Related Work

Some current researches such as [1–3,9] proposed approaches to solve the resources management, peers discovering, or self-adaptation problems on the decentralized edge nodes without core cloud. Study [1] proposed a resource management framework in the decentralized edge cluster decoupled from the cloud to deploy latency-sensitive IoT applications. Moreover, research [9] proposed a decentralized storage system, which utilized the proposed peer living loop algorithm to find other live nodes. What is more, the study [3] proposed a self-adaptive computing framework at the edge to manage uncertainties at runtime and satisfy the requirements of a decentralized data-intensive system. Furthermore, study [2] proposed an approach for coordinating resources and

computations in edge clusters, which can tolerate unreliability by self-adaptation to device failure, mobility, and withdrawal.

Other researches implemented the decentralized schemas based on the blockchain technology such as [5,6,17,20], and these studies concentrated on the security and collaboration of edge nodes. Study [6] analyzed the advantages and disadvantages of decentralized cloud storage solutions using blockchain technology. Moreover, research [5] proposed a multiagent and collaborative governance approach of decentralized edge micro clouds with blockchain-based distributed ledgers to improve the efficiency in decision-making. This paper [20] proposed a blockchain-based decentralized architecture called Microthings Chain, which supporting cross-domain data sharing safely. The study [17] proposed that using blockchain technology as a platform hierarchical and distributed control systems, and utilizing the edge nodes as the executive level for the actual process.

Some current researches focused on providing services to end devices based on edge nodes or collaborated with the cloud, or the service framework on edge, such as [11,14,16,18]. Study [14] proposed the deployment of edge-computing in 5G NFV environment and services-based architecture. Research [18] proposed an algorithm called OREO to jointly optimizes dynamic service caching and task offloading to solve service heterogeneity and decentralized coordination challenges. Moreover, another paper [11] used W-DAG to model the data-intensive service or business and make resource allocation between the cloud data center and edge nodes to improve QoS. The study [16] proposed a cloud-edge collaboration framework for IoT data analytics that can provide data analytics services.

There were studies concentrated on the storage at the edge. Which includes collaborated with the cloud, forecasted for pre-store, self-adaptive according to specific situations, such as [7,10,12,13]. The study [10] proposed a single namespace and resource-aware federation file system called EDGESTORE for edge nodes, which can aggregate storage namespace and federate resources of edge nodes to enable high resource-sharing in the federation. The study [12] proposed an effective model for edge-side collaborative storage in data-intensive edge computing, which can forward the offload requests to the with-data edge nodes. The study [7] proposed an edge storage method based on multi-view learning, which predicted the resources that need to be pre-stored based on multi-feature linear discriminant analysis, including their real-time performance and user's location. The study [13] presented a three-layer architecture model for data storage management on edge, including an adaptive algorithm that dynamically finds a trade-off between providing high forecast accuracy necessary for efficient real-time decisions.

5 Conclusion

With the development of IoT and 5G and the improvement of the hardware's performance, edge nodes of the Internet also have the power to store data or process tasks. However, we found that many local-area and small-scale scenarios also need the computing or storage resources of the edge nodes, and for lower costs, these scenarios usually partly decoupled with the core cloud.

In this paper, we proposed a collaborative edge-edge data storage service for edge computing in local-area scenarios called DECS. DECS runs at each edge node, which not only provides the fundamental data storage sub-service but also manages and coordinates the storage and computing resource of the whole cluster by the trade-off module and the prediction module. Especially, the latter can replicate data or forward requests in advance by predicting the access amount proactively. We implemented the proposed service and evaluated the performance compared with state-of-art research. The results proved that DECS was more suitable for these scenarios, lowered the access latency, and saved total bandwidth by the prediction beforehand. Moreover, the trade-off module improved the resource utilization of the whole edge cluster.

Acknowledgment. This paper is supported by Project 213.

References

1. Avasalcai, C., Tsigkanos, C., Dustdar, S.: Decentralized resource auctioning for latency-sensitive edge computing, pp. 72–76, July 2019. https://doi.org/10.1109/EDGE.2019.00027
2. Casadei, R., Viroli, M.: Coordinating computation at the edge: a decentralized, self-organizing, spatial approach, pp. 60–67, June 2019. https://doi.org/10.1109/FMEC.2019.8795355
3. D'Angelo, M.: Decentralized self-adaptive computing at the edge, pp. 144–148, May 2018
4. Decandia, G., et al.: Dynamo: Amazon's highly available key-value store. **41**(6), 205–220 (2007)
5. Freitag, F.: On the collaborative governance of decentralized edge microclouds with blockchain-based distributed ledgers, pp. 709–712, December 2018. https://doi.org/10.1109/WI.2018.000-7
6. Gabriel, T., Cornel-Cristian, A., Arhip-Calin, M., Zamfirescu, A.: Cloud storage. A comparison between centralized solutions versus decentralized cloud storage solutions using blockchain technology, pp. 1–5, September 2019. https://doi.org/10.1109/UPEC.2019.8893440
7. Gao, Q., Gao, L., Xue, T., Zhu, X., Zhao, X., Cao, R.: Multi-view learning based edge storage management strategy, pp. 366–371, August 2018. https://doi.org/10.1109/CBD.2018.00072
8. Ghemawat, S., Gobioff, H., Leung, S.T.: The Google file system. In: Proceedings of the 19th ACM Symposium on Operating Systems Principles, Bolton Landing, NY, pp. 20–43 (2003)
9. Gheorghe, A., Crecana, C., Negru, C., Pop, F., Dobre, C.: Decentralized storage system for edge computing, pp. 41–49, June 2019. https://doi.org/10.1109/ISPDC.2019.00009
10. Khan, A., Muhammad, A., Kim, Y., Park, S., Tak, B.: EDGESTORE: a single namespace and resource-aware federation file system for edge servers, pp. 101–108, July 2018. https://doi.org/10.1109/EDGE.2018.00021
11. Li, X., Lian, Z., Qin, X., Abawajyz, J.: Delay-aware resource allocation for data analysis in cloud-edge system, pp. 816–823, December 2018. https://doi.org/10.1109/BDCloud.2018.00122

12. Li, Y., Luo, J., Jin, J., Xiong, R., Dong, F.: An effective model for edge-side collaborative storage in data-intensive edge computing, pp. 92–97, May 2018. https://doi.org/10.1109/CSCWD.2018.8465306
13. Lujic, I., Maio, V.D., Brandic, I.: Efficient edge storage management based on near real-time forecasts, pp. 21–30, May 2017. https://doi.org/10.1109/ICFEC.2017.9
14. Lv, H., Chen, D., Wang, Y.: Deployment of edge-computing in 5G NFV environment and future service-based architecture, pp. 811–816, December 2018. https://doi.org/10.1109/CompComm.2018.8780937
15. Monga, S.K., Ramachandra, S.K., Simmhan, Y.: ElfStore: a resilient data storage service for federated edge and fog resources, pp. 336–345 (2019)
16. Moon, J., Cho, S., Kum, S., Lee, S.: Cloud-edge collaboration framework for IoT data analytics, pp. 1414–1416, October 2018. https://doi.org/10.1109/ICTC.2018.8539664
17. Stanciu, A.: Blockchain based distributed control system for edge computing, pp. 667–671, May 2017. https://doi.org/10.1109/CSCS.2017.102
18. Xu, J., Chen, L., Zhou, P.: Joint service caching and task offloading for mobile edge computing in dense networks, pp. 207–215, April 2018. https://doi.org/10.1109/INFOCOM.2018.8485977
19. Yousefpour, A., et al.: All one needs to know about fog computing and related edge computing paradigms: a complete survey (2018)
20. Zheng, J., Dong, X., Zhang, T., Chen, J., Tong, W., Yang, X.: MicrothingsChain: edge computing and decentralized IoT architecture based on blockchain for cross-domain data sharing, pp. 350–355, October 2018. https://doi.org/10.1109/NANA.2018.8648780