Computer Systems
Science and Engineering

Tech Science Press

DOI: 10.32604/csse.2023.035428
Article

Data Utilization-Based Adaptive Data Management Method for Distributed Storage System in WAN Environment

Sanghyuck Nam[1], Jaehwan Lee[2], Kyoungchan Kim[3], Mingyu Jo[1] and Sangoh Park[1,*]

[1]School of Computer Science and Engineering, Chung-Ang University, Seoul, 06974, Korea
[2]Department of Computer Science and Engineering, Kongju National University, Cheonan, 31080, Korea
[3]Qucell Networks, Seongnam, 13590, Korea
*Corresponding Author: Sangoh Park. Email: sopark@cau.ac.kr
Received: 20 August 2022; Accepted: 13 January 2023

Abstract: Recently, research on a distributed storage system that efficiently manages a large amount of data has been actively conducted following data production and demand increase. Physical expansion limits exist for traditional standalone storage systems, such as I/O and file system capacity. However, the existing distributed storage system does not consider where data is consumed and is more focused on data dissemination and optimizing the lookup cost of data location. And this leads to system performance degradation due to low locality occurring in a Wide Area Network (WAN) environment with high network latency. This problem hinders deploying distributed storage systems to multiple data centers over WAN. It lowers the scalability of distributed storage systems to accommodate data storage needs. This paper proposes a method for distributing data in a WAN environment considering network latency and data locality to solve this problem and increase overall system performance. The proposed distributed storage method monitors data utilization and locality to classify data temperature as hot, warm, and cold. With assigned data temperature, the proposed algorithm adaptively selects the appropriate data center and places data accordingly to overcome the excess latency from the WAN environment, leading to overall system performance degradation. This paper also conducts simulations to evaluate the proposed and existing distributed storage methods. The result shows that our proposed method reduced latency by 38% compared to the existing method. Therefore, the proposed method in this paper can be used in large-scale distributed storage systems over a WAN environment to improve latency and performance compared to existing methods, such as consistent hashing.

Keywords: Distributed system; distributed storage; distributed computing; object storage

1 Introduction

With the recent development of big data and artificial intelligence, the global demand for data is increasing. According to the Internet Data Center(IDC) Global Datasphere, global data generation is expected to increase from 64 ZB in 2020 [1] to 175 ZB in 2025 [2]. A High-performance distributed storage system is required in places that manage petabyte-scale large-capacity data, such as the Worldwide Large Hadron Collider Computing Grid (WLCG) [3]. A distributed storage system is a technology for storing and managing large-capacity data for artificial intelligence and extensive data analysis. Such data is held across numerous nodes in a distributed manner and used like a single storage system.

Methods for distributing data in a distributed storage system include table-based [4–10], hash-based [11–18], and subtree-based [19–23] methods. Table-based data distribution technology manages one central table in multiple servers, and the central table maintains data information and identifiers for data storage. Each data storage places data according to a central table and references it to search for the data. Although the table-based method has the advantage of the free arrangement of data, the synchronization cost caused by maintaining the central table limits its scalability. The size of the central table increases as the number of data identifiers increases. The hash-based data distribution method arranges data with a hash key obtained through a hash function using an identifier of data. The hash-based method has good data search efficiency. However, unlike the table-based data distribution method, the storage device corresponding to the hash key changes when there is a change in the number of storage nodes constituting the data storage. Moreover, these hash key changes require a data migration to operate normally. The subtree-based data distribution method distributes data in a tree structure so that the subtrees do not overlap. Thus, the data is distributed in non-overlapping subtrees and mapped to data storage to maintain load balancing. However, since the subtree-based data distribution method distributes data in units of subtrees, it is not easy to keep the locality of data.

To safely provide storage with a larger capacity than can be provided by a single data center, it is necessary to deploy large-scale distributed storage to multiple data centers interconnected to each other. In this case, relatively high latency and low bandwidth must be considered for a Wide Area Network (WAN) connection between data centers. However, existing distributed storage system studies [4,5,11–15,19–21] not only did not consider the network latency between data centers but also did not consider the locality of the data request. This can increase data exploration and transfer costs due to the high latency and low bandwidth of the WAN environment.

This paper proposes a distributed storage system that considers network latency and data locality in a WAN environment to solve these problems. The proposed system tracks the data utilization of users by the data center for efficient data placement. Based on this utilization, each data is placed in a data center frequently accessed by users. Therefore, data can be placed in a data center close to the users to reduce data access latency and improve the data transmission rate. Performance evaluation is performed in a real-world latency measurement-based environment to test the proposed method in a real-world environment and reflect distributed storage systems deployed globally.

The structure of this paper is as follows. Chapter 2 describes data management methods of existing distributed storage systems. Chapter 3 describes the proposed data distribution method considering network latency and data locality in a WAN environment. In Chapter 4, performance evaluation and analysis of the proposed data distribution method are described, and finally, in Chapter 5, the conclusion and future research are discussed.

2 Literature Review

A distributed storage system is a system that distributes and stores data across multiple connected nodes and allows users to use it as a single storage space. Large-capacity distributed storage configured across countries, such as WLCG, maintains connectivity between data centers through a WAN. However, the existing distributed storage system that does not consider the WAN environment may have a problem providing a service for user requests due to the high latency of the WAN environment because user requests are searched and transmitted via the network. Therefore, it is essential to place data on nodes that can minimize network latency to decrease data exploration and transfer costs. Thus, in a large-capacity distributed storage system considering a WAN environment, many user requests must be processed in high latency, so a distributed storage architecture must be constructed considering the data exploration cost.

xFS [4] distributes data by using the centralized table. The central table, called the Manager Map, is shared or replicated between clients and servers. The central table-based distribution method has limitations in scalability because as the system scales, a bottleneck occurs in the search request processing of the node managing the central table. BEAG [5] utilizes data age to estimate popularity. It distributes data accordingly to balance the load on overall nodes. However, only age information is considered and has a scalability problem as it is a centralized table-based approach. CORE [6] distributes data by analyzing data locality and distributes related data together to improve I/O performance. However, CORE needs to analyze historical data to calculate cohesion relations and map accordingly to distribute data. As CORE uses a centralized table, it also has a scalability problem. Other research [7–9] distributes data with the user's social information and latency and cost of data centers to optimize data placement. However, it requires the user's interaction information to estimate the access pattern to optimize a specific data set. It also has limited scalability as it is table based approach, and heavy computation analysis on user's interaction and data access pattern is required. RENDA [10] uses computation resources, network latency, and bandwidth to optimize data placement for periodic workloads in the cloud storage system. However, as it is optimized for recurrent workloads, it is unsuitable for generic data distribution methods. Also, it has scalability problems as it is a centralized table-based approach.

Cassandra [11] is a typical hash-based data distribution method. Cassandra improves data access latency in large-scale distributed storage systems through Consistent Hashing [12]. However, because Cassandra is hashing-based, it is challenging to relocate data considering data usage or network latency. Another hash-based data distribution method, AbFS [13], handles data storage by dividing it into units called volumes and manages segments containing multiple files as hash tables. However, when the volume size changes, the entire remapping occurs, and the mapping relationship between the segment identifier and the volume changes. The segment hashing technique of AbFS has a problem in that locality of data is lowered by performing hashing only with the identifier of the corresponding segment. In Luster [14], one management server manages multiple Object Storage Targets (OST), and efficient data access is possible by hashing the file name. However, a single point of failure (SPoF) problem may occur due to one management server. There is a problem in that the data locality is lowered by hashing only with the file name. Directory Level Based Hash (DLBH) [15] uses directory-based hashing to remove bottlenecks and provide high-speed data exploration. However, the hash-based data distribution method has a disadvantage. High latency may occur in accessing data in a WAN environment because data are placed without considering regional characteristics for data usage. EdgeKV [16] proposed distributed storage system for Edge Computing that distributes data to multiple edge groups using a hash-based method. However, the proposed method assumes data demand coincides with the edge group, so it does not consider the data demand outside the edge

group. Cache-based approach [17] to lower latency in distributed coded storage systems is proposed. However, it only suggests caching the coded data chunk to the frontend server, and only caching is considered. A more general distributed storage system may not use coded data replication schemes, so this method can not be applied to all distributed storage systems. And as only caching is used, the proposed method does not improve write operations as the write operation invalidates cached data chunks in the frontend server. Another cache-based approach [18] uses load imbalance caused by the skewness of incoming requests to cache more popular files to improve cache hit rate and mitigate I/O latencies. It partitions the files into partitions proportional to the expected load and places them on different servers to distribute the load. However, this approach does not consider network latency in a WAN environment, and the benefit of caching diminishes as the latency of accessing distributed partitions can be high. Also, the transmission time taken to retrieve data can be smaller than the network latency to request data as the bandwidth of the network infrastructure increases.

Subtree-based data distribution method Ceph [19] achieves load balancing through parameters such as available capacity or data load of each subtree. However, since Ceph utilizes hashing for data placement, the randomness of the hash function makes it impossible to map data to the desired location. Object-based file system (OBFS) [20] provides high scalability through dynamic tree structures. Still, because of the time-consuming search of trees distributed to nodes for data discovery, there is a problem that search performance decreases due to high latency in WAN environments. Dynamic Dir-Grain (DDG) [21] dynamically divides namespaces by the number of files, directories, and the depth of directories to maintain uniform locality and data distribution. However, as the number of files and directories increases due to the increase in file system size, the tree structure divided based on the file system structure becomes complicated, making it challenging to apply to large storage systems. In a study [22], data is distributed in consideration of data access time and geographic characteristics. Because the distributed storage node creates a replica by examining the ordered pair consisting of the most frequently accessed data and the access location in the cluster, it was possible to place the data in consideration of data access locality. However, since all data centers in the cluster must know the location information of all data, as the number of replicas increases, synchronization overhead increases, and system performance decreases. In another study [23], data is placed to optimize transmission time, and cloud service cost. However, it requires preknowledge of data as dataset and job that requires such dataset to compute optimal placement of datasets. This approach can only be applied to some use cases of the distributed storage system and can not be applied to the generic distributed storage system. Also, it does not consider modifying datasets or changing jobs and can not adapt to these changes.

For the scalability of a large-capacity distributed storage system, the WAN environment for data storage communication must be considered. However, network latency and bandwidth that may occur in the WAN environment are not considered in the existing system. So, this paper proposes a highly scalable distributed storage system considering the WAN environment to solve problems that may arise when applying existing distributed storage systems to the WAN environment.

3 Data Utilization-Based Adaptive Data Management Method

Since a large-capacity distributed storage system based on a WAN environment has high latency between each data center as it is a physical limitation that each data center has a long distance between them. Furthermore, as networking infrastructure evolves, high bandwidth shortens the time to transmit data. However, the latency stays constant. So, it is necessary to minimize the exchange of information between data centers when searching and retrieving for data to reduce the effect of

latency. This paper proposes a distributed storage structure that enables fast data access even with high latency in a WAN environment by placing data based on location and data usage utilization of clients and distributed storage servers, as shown in Fig. 1.

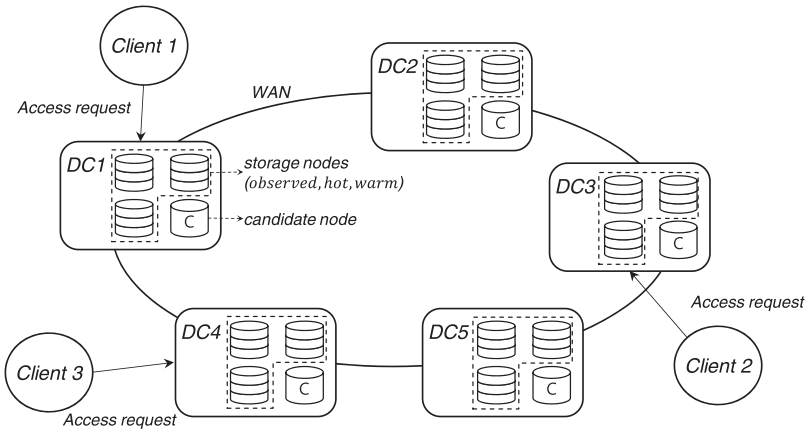

Figure 1: Overview of the proposed distributed storage system

Data centers are connected by WAN, and storage nodes in the data center are connected by Local Area Network (LAN). A client requests data access to the nearest data center to itself. The data center receiving the request searches whether the nodes in the data center have the data or whether the data is in another data center. For placement in consideration of data utilization and latency, storage nodes, as shown in Table 1, where data are stored for each data are divided into *observed*, *hot*, and *warm*.

Table 1: Node description

Node	Description
observed (*k*) storage node	storage node & storage node determined by Consistent Hashing of the key of data k
hot (*k*) storage node	The storage node in the data center with the highest utilization of data k. If not specified, point to the same node as *observed* (*k*)
warm (*k*) storage node	The storage node in the data center with the second highest utilization of data k. If not specified, point to the same node as *observed* (*k*).
candidate node	If the data requested by the user does not exist in the storage node in the data center, the node receives the data from another data center and temporarily stores it. Also tracks data utilization.

The *observed* storage node is a storage node derived by hashing the key of the data to be accessed. The *hot* storage node is the storage node with the highest data utilization, and the *warm* storage node is the second busiest storage node. Besides the storage node, the *candidate* node is a node that temporarily stores data when requested data does not exist in the data center by retrieving data from another data center. Since the *observed*, *hot*, and *warm* storage nodes are separated for each data, the *observed*, *hot*, and *warm* storage nodes of different data may be different. On the other hand, the *candidate* node is fixed and designated as a specific node within the data center.

The *observed* storage node is a storage node derived from Consistent Hashing of the data key. Consistent Hashing is a commonly used hashing algorithm in distributed storage systems to divide data evenly across storage nodes. Unlike the traditional hashing method, a hash ring is created, nodes are placed on the ring, and a hash value is assigned to the first encountered node. Through this, even if node configuration changes, the key to be migrated is less than the existing hashing method. When Consistent Hashing is performed, data is uniformly distributed across all storage nodes. The data center directly determines the *observed* node of the data through a hash function. The *observed* node also stores information about the *hot* and *warm* storage nodes of the stored data. When data is initially stored, the *observed* storage node of the data is initialized to the *hot*, *warm* storage node, and when the data is changed to the new *hot* and *warm* storage node, the *observed* storage node updates *hot* and *warm* storage node information. The *hot* storage node for specific data is a storage node belonging to the data center with the highest data utilization. It may be changed to another storage node as the data utilization changes. The *warm* storage node is a storage node belonging to the data center with the second-highest data usage rate.

The *candidate* node performs the role of temporarily storing the requested data from the client, tracking the data usage, and selecting the *hot*/*warm* storage. Data that does not exist in the storage node in the data center is maintained in the *candidate* node for the time specified by the administrator from the time the data center processes the client's request and records the number of data accesses of the client while each data is maintained. When the specified time elapses, the utilization of the *hot*, *warm* storage nodes is compared with the existing *hot* and *warm* storage nodes for the corresponding data to reselect *hot* and *warm* storage nodes. To prevent all infrequently used data from being managed, the *candidate* node of each data center stores only the M data with high utilization.

candidate nodes create the usage information list L with the top M data that they manage with high utilization every t_{tr} time specified by the administrator to reselect the *hot* and *warm* storage nodes and broadcast it to other *candidate* nodes. *candidate* nodes collect and aggregate L s broadcast by other *candidate* nodes through Algorithm 1 to create a L_{aggr} list that includes all top data utilization of other data centers.

Algorithm 1: Processing of the received usage list

```
1:  procedure OnReceiveUsageList(L_recv, L_aggr)
2:    for each l in L do
3:      if ∃l'.key (l'.key = l.key ∧ l' ∈ L_recv) then
4:        if ∃l'.key (l'.key = l''.key ∧ l'' ∈ L_recv) then
5:          if l'.usage > l''.usage_1stmax then
6:            l'.usage2ndmax ← l''.usage1stmax
7:            l'.usage1stmax ← l'.usage
8:          else if l'.usage > l''.usage2ndmax then
9:            l'.usage2ndmax ← l'.usage
10:         else
11:           continue
```

(Continued)

Algorithm 1: Continued

```
12:                end if
13:                L_aggr \ {l''}
14:              else
15:                l'.usage1stmax ← l'.usage
16:                l'.usage2ndmax ← 0
17:              end if
18:              {l'} ∪ L_aggr
19:            end if
20:          end for
21:          return L_aggr
22:        end procedure
```

In Algorithm 1, the data usage list L received from other *candidate* nodes is called L_{recv}, and it is aggregated in the list L_{aggr}. At this time, only the data included in the data usage list L managed by *candidate* nodes are collected. As shown in lines 4–13, if data contained in L_{recv} already exists in L_{aggr}, the largest value and the second largest value are compared and updated. Also, as shown in lines 13 and 18, if L_{aggr} already has utilization information. It is updated with new information. Otherwise, the largest and second-largest values are initialized and added to L_{aggr}, as shown in lines 14–18. When initialization is performed because there is no utilization information in L_{aggr}, the data utilization included in L_{recv} is set to the maximum value, and since there is no second-highest value, it is set to 0.

Each *candidate* node uses the usage list L of the data it manages and the aggregated usage list L_{aggr} to reselect the *hot* and *warm* storage nodes through Algorithm 2.

Algorithm 2: Determining target data for hot and warm by data center

```
1:    procedure OnAggregateUsageList(L, L_aggr)
2:      L_hot ← {∅}
3:      L_warm ← {∅}
4:      for each l in L do
5:        if ∄l'.key(l'.key = l.key ∧ l' ∈ L_aggr) then
6:          {l} ∪ L_hot
7:        else
8:          if l.usage > l'.usage1stmax then
9:            {l} ∪ L_hot
10:           else if l.usage > l'.usage2ndmax then
11:             {l} ∪ L_warm
12:           end if
13:       end if
14:     end for
15:     return L_hot, L_warm
16:   end procedure
```

Algorithm 2 compares L_{aggr} with L after aggregation to determine which data in the data center needs to be placed in newly selected *hot* and *warm* nodes. For example, if the utilization of any data A in L is greater than the maximum utilization of the same data A in L_{aggr}, the storage node *hot* for this data is designated as the storage node in that data center. Therefore, to select *hot* and *warm* storage

nodes, initialize the data lists L_{hot} and L_{warm} for *hot* and *warm* as shown in lines 2–3, and then compare the utilization for each data as shown in lines 4–14. As shown in line 5, if there is no element l of the data usage list L in L_{aggr}, it means that the storage node in the other data center does not have the data l, so as in line 6, designate the storage node in the data center to which current *candidate* node belongs as the *hot* node for l. If data utilization information is included in L_{aggr}, the L_{hot} or L_{warm} list is updated by comparing the utilization as in lines 8–12. Finally, two created lists containing data to be allocated to *hot*, *warm* storages are returned as a result, as shown in line 15. The data center then assigns a storage node and selects it as a new *hot*/*warm* storage node for the data. The *candidate* node migrates the temporarily stored data to the selected *hot* and *warm* storage nodes. Then, the *candidate* node notifies the *observed* storage node of the data to store the information about the new *hot*/*warm* storage node and informs the old *hot*/*warm* storage node for the corresponding data to remove stored data. When all *candidate* nodes have done this, they know the *hot*, *warm* storage node information for each data.

Fig. 2 describes the structure of accessing *observed*, *hot*, and *warm* storage nodes and data placement by Consistent Hashing in the proposed distributed storage system.

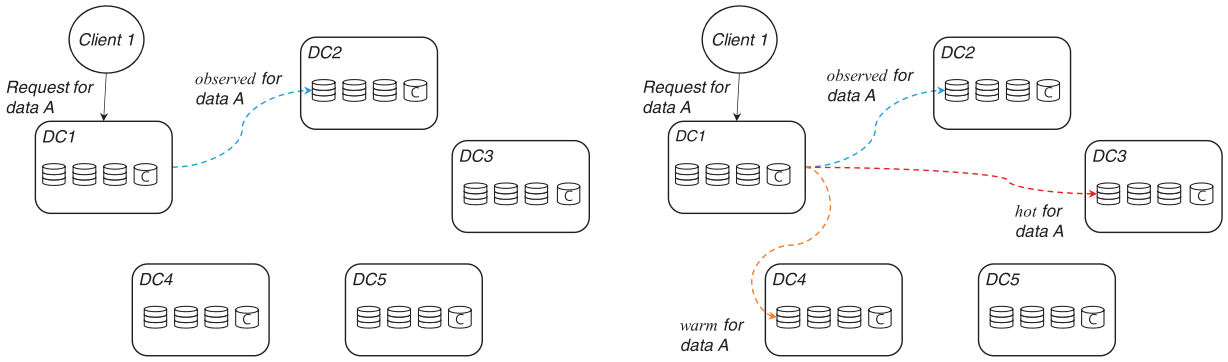

Figure 2: Data placement (a) without ***hot, warm*** (b) with ***hot, warm***

Fig. 2 shows Client 1 accesses data A to the distributed storage system through data center DC1. In this case, Fig. 2a is a case in which DC1 receives a request data A for the first time. DC1 finds out the *observed* storage node of the data through Consistent Hashing, requests data A from the data center to which the *observed* storage node belongs, and receives information on the *hot* and *warm* storage nodes that store data A, including *observed*. DC1 replicates data from the closest of the *observed*, *hot*, and *warm* storage nodes to the *candidate* node and sends it to Client 1. The proposed system reduces the overall access delay time because frequently used data is managed in the frequently used data center.

4 Performance Evaluation and Discussion

This paper analyzed the latency of data access based on the actual network latency data between data centers to evaluate the performance of the proposed distributed storage system. The data set used for performance analysis was collected by Verizon on latency that occurred in April 2021 from designated routers in major network hubs worldwide [24]. Next, consider how data is placed on a distributed storage node. Data is assumed to be evenly distributed among distributed storage nodes through Consistent Hashing. Since the latency for the client to access specific data follows a normal

distribution, the average access time of the system can be obtained as in Eq. (1).

$$\delta_{average} = \frac{\delta_1 + \delta_2 + \ldots + \delta_n}{n} \tag{1}$$

δ_n is the n th data center access delay time for the client. Based on the formula Eq. (1), the result of calculating the average data access latency by country of the Verizon dataset [24] is shown in Table 2.

Table 2: Worldwide network average latency measurement data

	Korea	Singapore	Hongkong	Sydney	Tokyo	India	UK
Korea	5	102.491	40.778	147.541	38.790	153.160	233.883
Singapore	102.491	5	33.297	92.060	76.465	39.106	163.298
Hongkong	40.778	33.297	5	124.516	51.034	83.923	275.279
Sydney	147.541	92.060	124.516	5	113.647	141.059	251.639
Tokyo	38.790	76.465	51.034	113.647	5	123.646	222.504
India	153.160	39.106	83.923	141.059	123.646	5	119.062
UK	233.883	163.298	275.279	251.639	222.504	119.062	5

The existing method used only consistent hashing without considering the data utilization, and the proposed method were compared based on latency data on Table 2. The distribution of which data the client accesses when accessing the data follows the Zipf distribution ($s = 1.01$).

This paper performed an empirical test based on average latency measurement data and compared the proposed method and the existing consistent hashing-based method. The results of analyzing the average data access latency time by country of the two methods and where the data centers are only located in Korea are shown in Fig. 3. The proposed method has about 38% latency performance improvement compared to the existing method. Furthermore, where the data centers are only located in Korea show very low latency when accessed from Korea, as expected, and high latency when accessed from the UK.

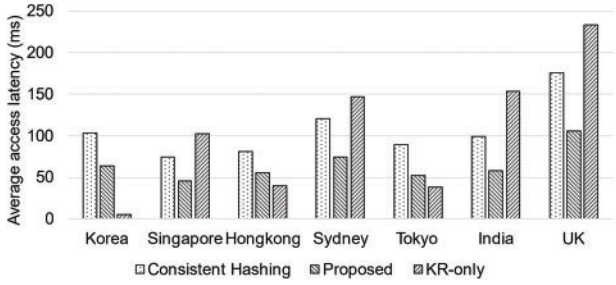

Figure 3: Average latency of client accessing the distributed storage

The proposed method reduces the overall access latency because the data is placed in a data center with high utilization, and excess latency introduced by accessing data in other data centers is reduced. Moreover, the proposed method can improve the overall system performance of the distributed storage system of WAN-connected data centers. However, the consistent hashing-based method shows higher latency because the data is placed by a hashing function, which is random, so accessing data in other data centers is higher than the proposed method.

Next, to evaluate the performance of the proposed system in various aspects, the average data access latency was measured while varying (a) the number of data centers, (b) the number of objects stored in the data center, and (c) the size of the list managed by *candidate* node M. As shown in Fig. 4, the proposed distributed storage system did not show any significant effect, such as increasing or decreasing the average access latency, even if the number of data centers increased.

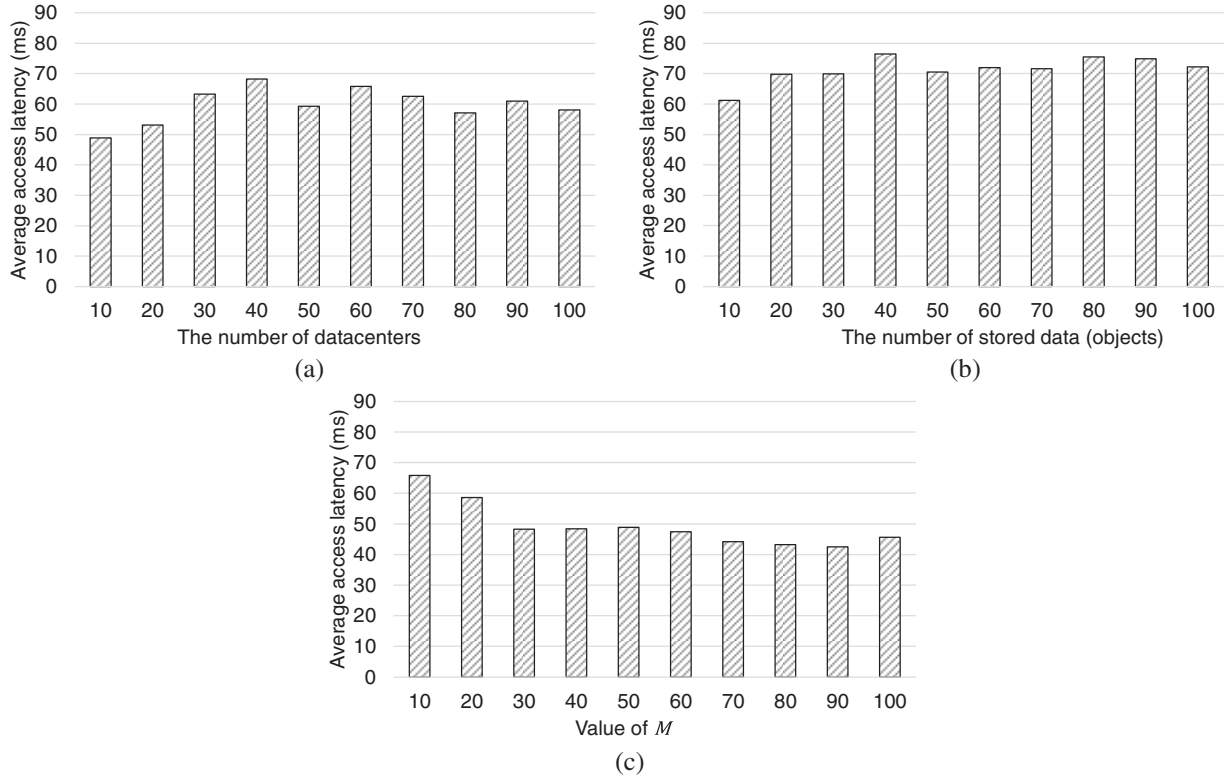

Figure 4: Average latency of client accessing the distributed storage with varying parameters

This is because even as the number of data centers increases, data is consistently replicated around frequently accessed data centers. Similarly, it was found that an increase in the number of stored data had no significant effect on the average access latency because the number of client requests is constant regardless of the total number of stored objects. Next, the average data access latency according to the change in the list size M managed by the *candidate* node tended to decrease the average access latency as M increased because the more data *candidate* held, the faster nearby clients accessed the data.

The average data access latency according to the time change is shown in Fig. 5. Here, the unit of time is t_{tr}, which is the period during which the *candidate* nodes broadcast the usage information list L.

In the distributed storage system to which only Consistent Hashing is applied, the average data access latency is measured as constant, but in the distributed storage system to which the method proposed in this paper is used, the average data access latency decreases rapidly from immediately after the first t_{tr}. Therefore, if the data distribution method proposed in this paper is used, data service with lower latency is possible even in a WAN environment. And also limit of data utilization can be higher as the waiting time induced by latency is lowered. As physical network latency also plays a role in

the data transmission rate, the data transmission rate between the user and distributed storage system is also higher. Therefore, in a distributed storage system deployed over nodes connected with a WAN environment, the proposed method can improve system performance and enable the deployment of a large-scale distributed storage system.

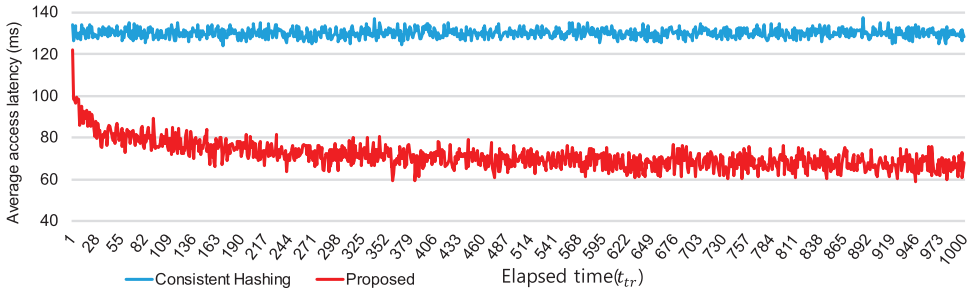

Figure 5: Average latency of client access according to time change

5 Conclusion

Establishing distributed storage system through the connection between data centers is necessary to overcome the storage capacity limitation of a single data center. Because WAN connections between data centers have high latency, the location where data is stored is vital to reduce the impact of latency and bandwidth. However, the existing distributed storage system does not consider the network latency between data centers.

This paper proposes a distributed storage system that provides low-latency data search even in a WAN environment by storing data in locations based on the user's data utilization. For this, each data center tracks the user's data utilization and places the data in an area with high utilization, demonstrating that low-latency data exploration and retrieval is possible. Analysis of average latency using global major network hub latency data showed that the proposed system reduced latency by 38% compared to the existing system. This improvement can be applied to large-scale distributed storage systems where the storage node is deployed globally and connected via WAN to lower latency and achieve higher data utilization.

However, the proposed method has a limitation in that the user must request the data, and when the number of requests is sufficient, the system migrates that data near the user. The time needed for the system to be efficient can be adjusted through T_{tr}. However, it has its limitation, as lower T_{tr} means frequent data migration is required. For future research, it is necessary to study a way to reduce the overall latency while reducing data migration and improving system reaction time to the data demand from the user.

Funding Statement: This research was supported by the Chung-Ang University Graduate Research Scholarship in 2021. This study was carried out with the support of 'R&D Program for Forest Science Technology (Project No. 2021338C10-2223-CD02)' provided by Korea Forest Service (Korea Forestry Promotion Institute).

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding the present study.

References

[1] M. Shirer, *Data Creation and Replication will Grow at a Faster Rate than Installed Storage Capacity*, San Francisco, CA, USA: Business Wire, 2021. [Online]. Available: https://www.businesswire.com/news/home/20210324005175/en/

[2] D. Reinsel, J. Grantz and J. Rydning, *Data Age 2025: The Evolution of Data to Life-Critical*, Needham, MA, USA: IDC, 2017. [Online]. Available: https://www.seagate.com/files/www-content/our-story/trends/files/idc-seagate-dataage-whitepaper.pdf

[3] M. Gaillard, *CERN Data Centre passes the 200-petabyte milestone*, Geneva, Switzerland: Cern, 2016. [Online]. Available: https://cds.cern.ch/record/2276551

[4] T. E. Anderson, M. D. Dahlin, J. M. Neefe, D. A. Patterson, D. S. Roselli *et al.,* "Serverless network file systems," in *Proc. of the Fifteenth ACM Symp. on Operating Systems Principles*, Copper Mountain, Colorado, USA, pp. 109–126, 1995.

[5] X. Y. Luo, G. Xin and X. L. Gui, "Data placement algorithm for improving I/O load balance without using popularity information," *Mathematical Problems in Engineering*, vol. 2019, pp. 2617630, 2019.

[6] S. Vengadeswaran. and S. R. Balasundaram, "CORE-An optimal data placement strategy in hadoop for data intensive applications based on cohesion relation," *Computer Systems Science and Engineering*, vol. 34, no. 1, pp. 47–60, 2019.

[7] J. Zhou, J. Fan, J. Jia, B. Cheng and Z. Liu, "Optimizing cost for geo-distributed storage systems in online social networks," *Journal of Computational Science*, vol. 26, pp. 363–374, 2018.

[8] B. P. Shankar and S. Chitra, "Optimal data placement and replication approach for SIoT with edge," *Computer Systems Science and Engineering*, vol. 41, no. 2, pp. 661–676, 2022.

[9] A. Atrey, G. Van Seghbroeck, H. Mora, F. De Turck and B. Volckaert, "SpeCH: A scalable framework for data placement of data-intensive services in geo-distributed clouds," *Journal of Network and Computer Applications*, vol. 142, pp. 1–14, 2019.

[10] H. K. Thakkar, P. K. Sahoo and B. Veeravalli, "Renda: Resource and network aware data placement algorithm for periodic workloads in cloud," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 12, pp. 2906–2920, 2021.

[11] A. Lakshman and P. Malik, "Cassandra: A decentralized structured storage system," *ACM SIGOPS Operating Systems Review*, vol. 44, no. 2, pp. 35–40, 2010.

[12] D. Karger, E. Lehman, T. Leighton, R. Panigrahy, M. Levine *et al.,* "Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web," in *Proc. of the Twenty-Ninth Annual ACM Symp. on Theory of Computing*, El Paso, Texas, USA, pp. 654–663, 1997.

[13] A. F. Díaz, M. Anguita, H. E. Camacho, E. Nieto and J. Ortega, "Two-level hash/table approach for metadata management in distributed file systems," *The Journal of Supercomputing*, vol. 64, no. 1, pp. 144–155, 2013.

[14] P. Braam, "The Lustre storage architecture," *arXiv preprint arXiv:1903.01955*, 2019.

[15] L. Ran and H. Jin, "An efficient metadata management method in large distributed storage systems," in *Proc. of the Int. Conf. on Human-Centric Computing 2011 and Embedded and Multimedia Computing 2011*, China, pp. 375–383, 2011.

[16] K. Sonbol, Ö. Özkasap, I. Al-Oqily and M. Aloqaily, "EdgeKV: Decentralized, scalable, and consistent storage for the edge," *Journal of Parallel and Distributed Computing*, vol. 144, pp. 28–40, 2020.

[17] K. Liu, J. Peng, J. Wang and J. Pan, "Optimal caching for low latency in distributed coded storage systems," *IEEE/ACM Transactions on Networking*, vol. 30, pp. 1132–1145, 2021.

[18] Y. Yu, W. Wang, R. Huang, J. Zhang and K. B. Letaief, "Achieving load-balanced, redundancy-free cluster caching with selective partition," *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 2, pp. 439–454, 2019.

[19] S. A. Weil, S. A. Brandt, E. L. Miller, D. D. Long and C. Maltzahn, "Ceph: A scalable, high-performance distributed file system," in *Proc. of the 7th Symp. on Operating Systems Design and Implementation*, Seattle, Washington, USA, pp. 307–320, 2006.

[20] F. Wang, S. A. Brandt, E. L. Miller and D. D. Long, "OBFS: A file system for object-based storage devices," *in Proc. of the 21st IEEE / 12th NASA Goddard Conf. on Mass Storage Systems and Technologies*, Greenbelt, Maryland, USA, pp. 283–300, 2004.

[21] J. Xiong, Y. Hu, G. Li, R. Tang and Z. Fan, "Metadata distribution and consistency techniques for large-scale cluster file systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 22, no. 5, pp. 803–816, 2010.

[22] P. Matri, A. Costan, G. Antoniu, J. Montes and M. S. Pérez, "Towards efficient location and placement of dynamic replicas for geo-distributed data stores," in *Proc. of the ACM 7th Workshop on Scientific Cloud Computing*, New York, USA, pp. 3–9, 2016.

[23] C. Li, Q. Cai and Y. Lou, "Optimal data placement strategy considering capacity limitation and load balancing in geographically distributed cloud," *Future Generation Computer Systems*, vol. 127, pp. 142–159, 2022.

[24] *IP Latency statistics*. [Online]. Available: https://www.verizon.com/business/terms/latency/