Hindawi
Scientific Programming
Volume 2017, Article ID 2824782, 17 pages
https://doi.org/10.1155/2017/2824782

Research Article

Adaptive Data Placement for Improving Performance of Online Social Network Services in a Multicloud Environment

Seunghee Han, Bosung Kim, Jaemin Han, Kyehee Kim, and JooSeok Song

Yonsei University, 50 Yonsei-ro, Seodaemun-gu, Seoul, Republic of Korea

Correspondence should be addressed to Seunghee Han; alphahacker@yonsei.ac.kr

Received 28 March 2017; Revised 9 June 2017; Accepted 20 June 2017; Published 1 August 2017

Academic Editor: Iria Estevez-Ayres

The existing online social network (OSN) services in a multiple-cloud (Multicloud) environment use replications to store user data for improving the service performance. However, it not only generates tremendous traffic for synchronization between data but also stores considerable redundant data, thus causing large storage costs. In addition, it does not provide dynamic load balancing considering the resource status of each cloud. As a result, it cannot cope with the degradation of performance caused by the resource contention. We introduce an adaptive data placement algorithm without the replications for improving the performance of the OSN services in the Multicloud environment. Our approach is designed to avoid server overhead using data balancing technique, which locates data from a cloud to another according to the amount of traffic. To provide acceptable latency delay, it also considers the relationship between users and the distance between user and cloud when transferring data. To validate our approach, we experimented with actual users' locations and times of use collected from OSN services. Our findings indicate that this approach can reduce the resource contention by an average of more than 59%, reduce storage volume to at least 50%, and maintain the latency delay under 50 ms.

1. Introduction

Recently, online social network (OSN) services such as Twitter, Facebook, and Instagram have dramatically spread over the world. Many OSN service providers prefer to deploy their applications on a cloud environment to reduce costs associated with installing and running applications and to gain scalability via using the cloud. These days, OSN services generate tremendous data and traffic, which differ by time and location. Generally, users of these services are located across the globe.

The main actors throughout this paper are Cloud Service Providers (CSPs), Service Providers (SPs), and users. We consider the following definitions for these actors: CSPs offer computing, storage, and network resources required for hosting online services on an Infrastructure as a Service (IaaS) basis. SPs offer online services such as Twitter, Facebook, and Instagram using hardware resources provisioned by CSPs. And SPs utilize multiple clouds without relying on any interoperability functionalities implemented by the CSPs [1]. The online services are accessed by users. Users access and use the online services.

In this situation, a traditional single cloud computing system has a limitation regarding performance of the OSN service in terms of latency delay and execution delay. Because latency delay is the time lost in network transfer between a user and a cloud, if the user's location is far from the single cloud, the latency delay can increase. Execution delay consists of processing delays and queueing delays. Processing delay refers to the time required for any processing of a packet in the cloud server system. Queueing delay accounts for the time a packet waits in memory, before it is processed. Thus, queueing delay can increase, due to resource contention, according to server overhead. Consequently, execution delay can increase as queueing delay increases. Hence, if all users have access to a single cloud, execution delays can increase.

To mitigate such problems, many researchers have envisioned the use of multiple clouds (Multicloud). By building a Multicloud around the user, resource contention in each cloud can be reduced, and communication distance between

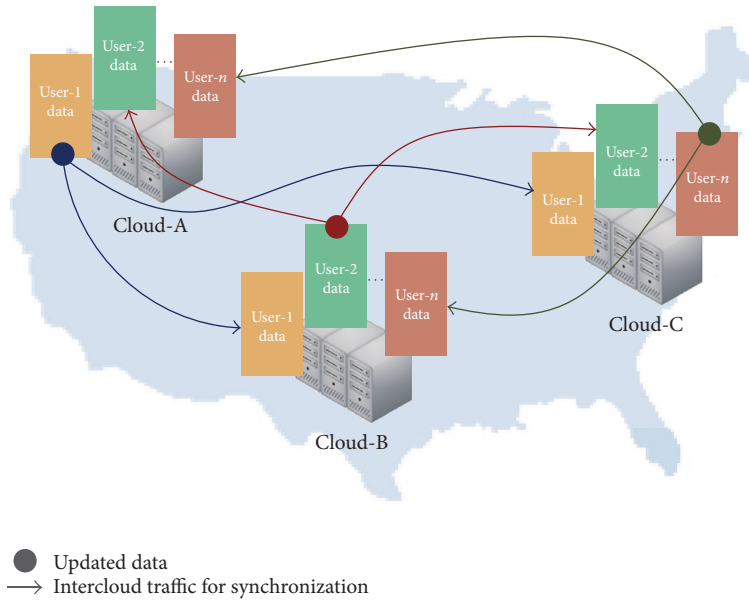

Figure 1: The existing method for storing user data using the replication technique.

the user and each cloud can be reduced. Therefore, execution delay and latency delay also can be reduced.

There have been many studies to make online services, deployed in a Multicloud, more efficient in terms of performance. Some studies have addressed this issue by continuously monitoring the service's performance and provisioning resources additionally as needed. Other studies have addressed this issue by applying a more efficient replication technique [1–6].

However, there are still following limitations. First, in most Multicloud studies and projects, user data are redundantly stored in several clouds using the master-slave replications technique [3]. In particular, where there are more than 20 clouds, such as the United States [7–9], the users benefit in terms of latency delay by accessing the closest cloud to use the OSN service [1, 2]. However, in this case, unnecessary storage waste may occur by storing user data into the cloud, which are located in a place that users rarely access. Moreover, as shown in Figure 1, tremendous intercloud traffic can be generated for synchronization between data. The intercloud traffic is due to communication between the clouds in the Multicloud environment. Second, in the location where there are many users who want to use the OSN service, the possibility of server overhead can be still high because all user requests go to the same, closest cloud. As a result, it may increase execution delay and cause degradation of OSN service's performance.

To resolve these problems, as shown in Figure 2, user data should be stored in several clouds in a nonduplicated way and transferred to other clouds according to the amount of traffic, guaranteeing acceptable latency delay.

In this paper, we introduced an approach that avoids server overhead through adaptive data placement using the data balancing technique, which moves data from a cloud to another, according to the amount of traffic in real-time and, simultaneously, provides acceptable latency delay using social relationship between users in the OSN service.

We assume that CSPs and clouds to deploy the OSN service have been selected in advance, considering cost, performance, reliability, user distribution, and so on. It is also assumed that SPs do not use additional clouds or change to another cloud for the purpose of improving performance of the OSN service.

Furthermore, as with the existing method, our approach also assumes that the availability of the cloud servers is not compromised due to data loss in a certain cloud server because there is backup data.

We demonstrated our approach by modeling a novel architecture of Multicloud system and implementing an algorithm of each component. We collected data from Twitter, which is a representative OSN service, to capture the characteristics of traffic patterns by time and location. Furthermore, we validated that our approach is applicable in the real-world by generating a workload with the collected data.

The key contributions of this paper are twofold. First, we provided traffic pattern information based on time, location, and relationships between users via data collected from a real-world OSN service. Such information is meaningful since it is difficult to be obtained without help from the OSN SP. Second, we provided a novel approach of an adaptive data placement algorithm considering real-time traffic and the social relationships between users. Consequently, our approach prevents performance degradation of the OSN service by distributing traffic requested by users and reducing unnecessary intercloud traffic, while considering latency delay.

The rest of this paper is organized as follows. In Section 2, we describe related works and compare them to ours. In Section 3, we provide an analysis of traffic pattern by time and location with the collected data. Section 4 outlines

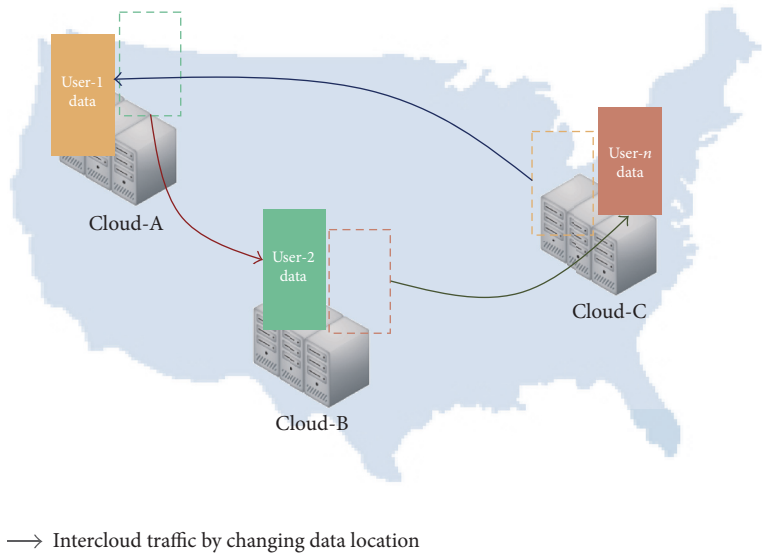

FIGURE 2: Our approach for storing user data in a nonduplicated way and changing data location.

our proposed architecture and introduces each architectural component. Section 5 details our algorithm for adaptive data placement considering social relationships. Section 6 discusses the settings and results of our evaluation, and Section 7 concludes this paper and highlights future work.

2. Related Works

Considerable efforts have been made to efficiently use resources to increase performance of online services in the Multicloud system. In this section, we focus on what efforts have been made to maintain and improve the service's performance using existing clouds without finding new CSP or cloud. The related works can be categorized into three types.

2.1. Resource Provisioning. Several projects facilitate resource provisioning in the Multicloud environment [4–6]. They provide components dedicated to continuously monitoring online service's status and provisioning resources additionally as needed in order to improve and maintain the service's performance.

However, such previous studies did not deal with dynamic load distribution among clouds according to resource status and cannot consider the geographical data location constraints of the clouds and users. Thus, it is hard to cope with the bottleneck at the data layer, and it is hard to improve the service's performance in terms of the latency delay which is affected by user and data location. In contrast, our approach finds the optimal cloud for storing each user's data, considering the user and cloud location, and social relationship on the OSN services, without provisioning additional resources. Additionally, our approach facilitates load distribution according to data location.

2.2. Load Distribution. Grozev and Buyya proposed an adaptive dynamic provisioning and autonomous workload redirection algorithm [1]. When user comes to the cloud system for the first time, the user is mapped onto an appropriate cloud based on the users' location, identity, and information about each cloud. This cloud selection process can be likened to a load balancer because it redirects users to serving clouds. However, after the cloud selection, the user is served within the selected cloud and has no further interaction for new mapping between users and clouds. Hence, there is still a risk of a bottleneck that may occur in the data layer because it cannot adequately to cope with the different traffic occurring in each cloud. Amazon Route 53 is a domain name system (DNS) web service that distributes user's request into several clouds with the lowest latency using the latency-based routing (LBR) system [10]. However, it does not consider traffic overhead at each cloud.

These works allow the OSN SP to replicate data in several or all of the clouds. If each cloud has a copy of the data, when some data are updated, the other data also have to be updated. It obviously generates tremendous load, and the replication is a waste of storage volume itself. Moreover, because these do not consider application-specific data deployment, there is limitation of optimizing the system to improve performance of OSN services. In contrast, our approach does not replicate user data when storing them onto the clouds. Thus, we can reduce waste of storage and traffic overhead due to the update of replicated data. Furthermore, the OSN service's performance can be improved by placing data onto the proper cloud in terms of latency delay.

2.3. Data Placement. Jiao et al. provided a data placement technique to place the data of user and their friends as close as possible [3]. This is because if the user accesses their friends' data without relaying the user's request to another cloud,

User place/time	1 a.m.	2 a.m.	3 a.m.	4 a.m.	5 a.m.	6 a.m.	7 a.m.	8 a.m.	9 a.m.	10 a.m.	11 a.m.	12 p.m.	1 p.m.	2 p.m.	3 p.m.	4 p.m.	5 p.m.	6 p.m.	7 p.m.	8 p.m.	9 p.m.	10 p.m.	11 p.m.	0 a.m.
Kansas	269	418	316	281	240	105	22	22	7	5	33	120	111	287	276	285	277	276	296	309	250	179	312	300
Texas	6166	6761	7135	6891	5058	3254	2090	1381	1262	1218	1381	1934	3051	3810	4871	5616	5574	5350	5404	5477	4668	4780	4755	5309
Minnesota	127	144	152	108	127	120	77	54	48	51	56	85	90	107	137	152	145	112	76	62	86	66	81	115
Iowa	35	22	36	26	18	11	5	4	13	18	109	90	61	58	41	37	56	44	33	22	48	58	57	43
Missouri	468	600	545	419	276	196	158	103	112	152	184	209	311	450	491	573	563	549	483	487	495	389	411	410
Louisiana	354	411	382	244	192	209	132	100	85	78	77	128	183	274	318	334	277	241	228	310	244	227	236	261
Wisconsin	4311	4532	4178	3094	1968	1169	862	637	687	729	1025	1748	2252	3234	3655	3860	3740	3754	3594	3771	3994	3430	3300	3508
Illinois	281	372	333	227	158	89	96	84	68	74	120	268	310	428	438	452	392	425	370	398	359	336	247	255
Mississippi	19	15	18	7	2	0	2	1	0	2	2	3	16	55	50	83	57	47	75	41	30	29	22	26
Michigan	7	6	1	0	10	5	7	23	15	30	34	39	78	101	89	69	60	68	57	65	34	39	7	9
Tennessee	488	451	399	256	99	52	37	30	71	69	197	260	311	395	470	426	443	336	347	394	391	415	424	424
Ohio	542	481	586	562	509	413	320	486	398	449	615	640	663	746	763	881	632	589	585	594	480	403	417	449
Georgia	7616	7458	7444	6553	4735	2910	1835	1613	1565	1601	1949	2929	4678	5838	6848	6708	6690	6522	6134	5975	5922	5206	5439	6337
Newyork	4773	4842	4329	3515	3098	2445	2018	1781	1715	1780	2258	2888	4192	4977	5699	5851	6094	6211	6454	6192	5569	4853	4752	4805
Pennsylvania	772	758	811	642	488	263	233	176	107	152	287	514	659	788	807	786	768	788	823	865	667	761	629	722
Virginia	1368	1433	1287	870	781	640	644	571	581	653	1019	1231	1660	1945	1900	1818	1559	1689	1579	1650	1474	1365	1364	1352
Northcarolina	3308	3328	3208	2842	2344	1759	1363	1539	1376	1723	1906	2677	3188	3522	3869	4230	4135	4306	4407	4296	4178	3941	3301	3224
Florida	1975	2130	2131	1651	1014	665	368	311	389	540	755	1113	1273	1508	1558	1759	1750	1749	1870	1972	1946	1803	1715	1826
Newjersy	842	669	675	530	391	346	274	242	321	443	500	800	835	1107	1023	1090	1054	849	808	989	996	804	847	786
Newhampshire	9	22	26	11	2	5	3	6	4	2	14	19	21	30	32	17	21	13	16	9	10	5	12	14
Vermont	21	9	8	12	6	10	7	7	6	8	7	38	48	69	55	44	32	37	52	33	24	29	48	100
Massachusetts	5033	4842	4728	3524	2519	1844	1724	1511	1746	1873	2472	3182	3776	4661	5237	5220	5441	5688	5795	5833	5192	5033	4701	4650
Connecticut	763	818	756	568	440	251	149	109	122	143	179	363	548	719	781	821	746	728	679	673	670	757	828	842
Rhode	2434	2730	2721	2347	1848	1225	898	836	843	966	1109	1622	2076	2281	2643	2953	3101	2846	2683	2736	2726	2451	2176	2402

Traffic low — Traffic high

Figure 3: Amount of traffic generated by locations and times in the US.

the additional latency delay is avoided. Additionally, traffic associated with further operations that find data stored in the other cloud is reduced. Liu et al. proposed that a cloud can only replicate its frequently requested data from other clouds to reduce intercloud communication due to the update of the replications [2].

These two studies mentioned above consider application-specific factors, particularly the social relationship between users. However, there is still the problem of storage wastage due to replication. Moreover, there is an assumption that clouds can provide infinite resources on demand. In actuality, such resources are not infinite and there are complex issues that limit full exploitation of the resources [11].

In contrast, our approach provides an algorithm for several clouds considering traffic in real-time and the resource status to avoid increasing execution delay. Moreover, it considers geographical locations of users and clouds to maintain acceptable latency delay, even when compared to existing approaches in which the user's request accesses the closest cloud.

Furthermore, the abovementioned approaches relay a user's request to another cloud if there are no data in which the user wants to access the closest cloud. This makes the user's request access to the closest cloud at least once without considering the existence of the data, causing unnecessary traffic to that cloud. In contrast, in our approach, we introduce the redirection layer, which enables the user to be directly routed to the cloud that has the data the user wants to access. Therefore, resource contention can be reduced and server overhead can be avoided within each cloud.

In summary, the existing research works generally consider latency delay or management of resource provisioning with the redundant data and the exception of considering the resource contention in the data layer of the three-tier architecture. However, the network bottleneck that confers performance degradation occurs in the data layer. Therefore, user data should be distributed and placed in a nonduplicated way and balanced according to the resource status of each cloud to avoid increasing execution delay upon the bottleneck at the data layer, guaranteeing acceptable latency delay according to the distance between the user and the cloud. To the best of our knowledge, there is no adaptive dynamic data placement approach that simultaneously considers execution delay and latency delay.

3. Traffic Pattern Analysis

To capture the characteristics of an OSN service, we collected the data of Twitter users via Tweepy API [12]. From June to October 2016, 1,297,163 tweets were collected via depth-first search (DFS). Each tweet has information regarding the user ID of who wrote the tweet, user location, time for which each tweet was written, and the user who received the tweet. As a result of analyzing using the collected data, some characteristics were found as shown in Figure 3.

Figure 3 depicts the amount of traffic by time and location. As shown in the figure, the amount of traffic is generated differently by location at the same time, as well as by time at the same location. These features also appear in other OSN services such as Facebook and Instagram [2, 13–15].

This feature shows that the performance of the OSN service on the cloud can be affected by the state of the resource if the traffic load is not dynamically managed according to the time and location. In other words, the load should be distributed into several different clouds to make the OSN service users feel that the cloud's resources are unlimited. If user data are redundantly stored in several clouds, it will cause unnecessary intercloud traffic due to data updates and waste

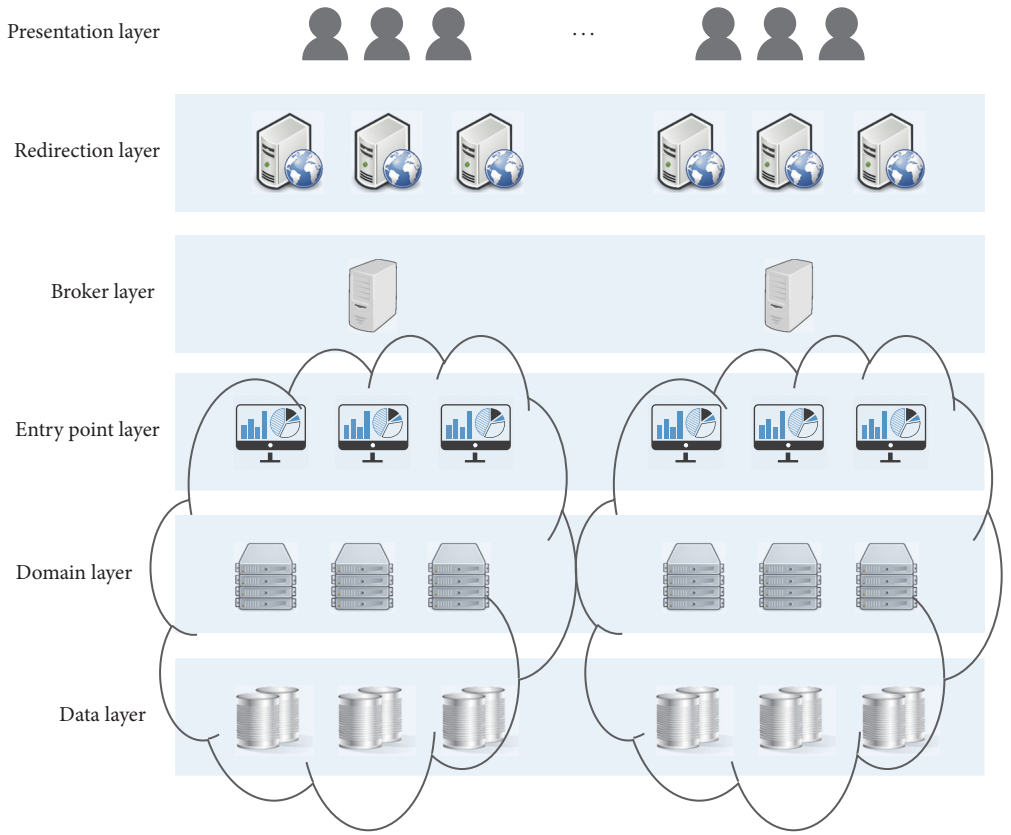

Figure 4: A six-layer architecture with three layers added to the existing three-tier architecture. The cloud shape indicates one cloud. A cloud consists of several entry point servers, OSN service servers, and database servers.

of storage. Therefore, data balancing is required to divide the database horizontally and store it in multiple clouds without replications.

Furthermore, most OSN services such as Twitter, Facebook, and Instagram are designed to allow users to access more of their friends' data than their own data or a non-friends' data [16]. Therefore, when choosing the cloud in which the user's data will be stored, it is important to consider not only the location of the user herself/himself but also where the user's friends are located. It is necessary to store the data in an optimal cloud in terms of latency delay according to distance.

4. Proposed Multicloud Architecture

4.1. Components. The three-tier architecture, which is used by most online applications, comprises three major parts: a presentation layer corresponding to a user terminal, a domain layer with the server logic of an application, and a data layer to serve as a data store. In our approach, to enable adaptive dynamic data placement and to effectively and dynamically distribute user requests via that data placement, a six-layer architecture shown in Figure 4, which is an extension of the traditional three-tier architecture, is needed.

The basic elements of the three-tier architecture remain unchanged, but the existing architecture is modified by adding the following novel elements: redirection layer, broker layer, and entry point layer. Since the CSPs provide the cloud service on an IaaS basis, the elements are built by each SP.

The role of each element is as follows.

(i) Presentation Layer. It represents the user's terminal and is the starting point of the user's request and the end point of response for the user's activity (read/write operation).

(ii) Redirection Layer. It consists of several redirector servers. The redirector server informs the user where the data he/she wants to access is located. This server can be built and operated on a proxy server. If the redirector server is in multiple locations near the users, the performance bottleneck and latency delay can be reduced. In order to minimize costs, the redirector server can be built on each cloud. Since the user simply obtains information on the cache of the redirector server, the time complexity is O (1). In addition, to minimize the load on the redirector server, the redirection is performed only for the user's first request to get the location of each user's data that the user wants to access after the broker performs the matching algorithm, which determines user data placement. After this, the user who sent the request message has the location of the data and can directly access the cloud that has the data without accessing the redirector server.

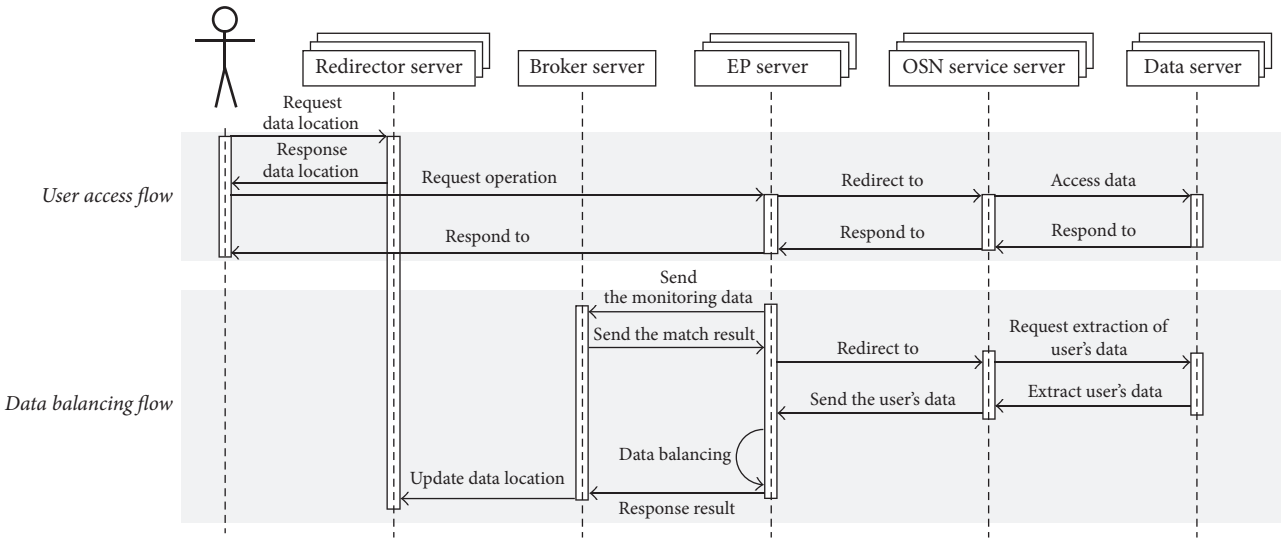

Figure 5: Communication flow between components.

(iii) Broker Layer. It can be clustered to handle large amounts of data and can be deployed in the clouds for better scalability. In the broker server, an algorithm searching for an optimal match between the user and the cloud is performed to store user data into multiple clouds, considering the distance between the user and the cloud, the relationship between users, and real-time traffic.

(iv) Entry Point Layer. It consists of several entry point (EP) servers. The EP server monitors a user's activities, including his/her identity during a unit of time, and performs the data balancing between the clouds. In addition, the standard load balancer, in round-robin way, can be built on this layer. The unit of time can vary depending on the needs of the SP.

(v) Domain Layer. This layer consists of several OSN service servers. In each server, the request that the user sends is processed, which means the service logic that the OSN SP created is processed in the servers for the user's activity.

(vi) Data Layer. This is where user data are stored. There are data servers in the data layer.

4.2. Component Interactions. Figure 5 depicts the process sequence of our suggested system. The sequence can be divided into the following: (i) user access flow shows the process that a user takes when using OSN service and (ii) data balancing flow shows a process for determining the data placement of users.

A detailed description of each flow is as follows.

(i) User Access Flow. The user finds out through the redirector server where the target data to access are located among multiple clouds. It then sends the user's request to the EP server of the cloud where the OSN service server is located. The EP server records the user's identity, where the request came from (identifying the region via IP), and how much data were sent or requested and redirects those data to the OSN service server. When the OSN service server writes or reads the data corresponding to the request in the database and sends a response to the OSN service server, the response finally reaches the user terminal through the EP server.

(ii) Data Balancing Flow. The EP server sends the monitoring results collected in the user access flow process (total traffic at each cloud, traffic generated by each user, and the location of the user who sent the request) to the broker server. The broker server calculates where each user's data have to be stored (that is, it finds a best match to store user data. See Section 5 for more details), taking into account the distance between the user and each cloud, the distance between the user's friend and each cloud, and the total amount of traffic generated during a unit of time in each cloud. Such information can be inferred from the monitoring results. Then, when the match result is sent to the EP server, the EP server extracts the data requiring the balancing and sends them to the EP server of the destination cloud. When the data balancing is completed, information on the placement of the user's data is updated in the redirector server.

5. Matching Algorithm in Broker Server

Determining in which cloud each user's data are stored without replications is equivalent to matching between users and clouds, as shown in Figure 6. When the users and clouds are arrayed as vertex sets (user set and cloud set), a bipartite graph can be developed by connecting every vertex that can be matched. Each edge has a weight, whose value can be expressed according to the location of the user, the locations of the user's friends, and the degree of familiarity between the user and each friend. The matching algorithm finds an optimal match between the user and cloud using the weight value. In Figure 6, the red lines indicate the optimal match.

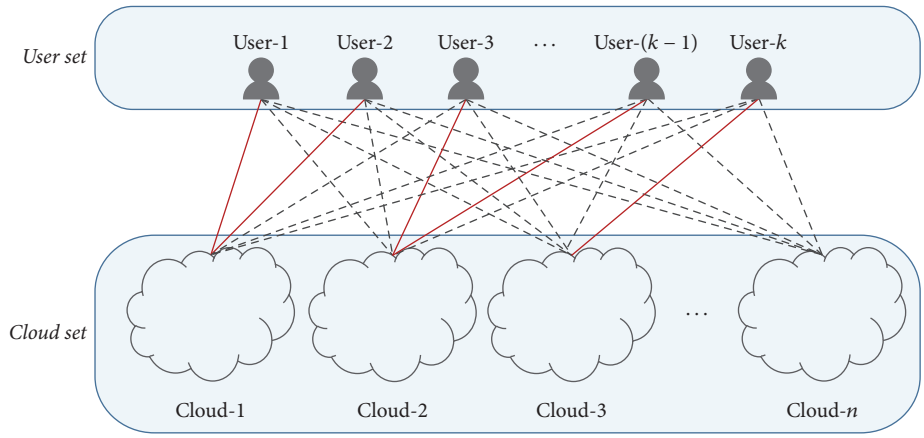

Figure 6: Matches between users and the clouds.

Thus, User-1's data are stored in Cloud-1, User-2's data are stored in Cloud-1, and User-k's data are stored in Cloud-3.

The matching algorithm for finding a match consists of an L-match and a T-match. For each user, a match is made to improve performance by selecting the edge that has the minimum weight value from the viewpoint of latency delay. We refer to this as the L-match.

If the data are stored by the L-match, it is checked whether the amount of traffic expected to occur at each cloud is within an acceptable range considering the capacity of each cloud based on the amount of traffic generated by each user in a previous unit of time. The capacity indicates the volume of traffic that each cloud can stably process without the queueing delay which is caused by the resource contention.

If the capacity of a cloud is less than the traffic expected to occur at the cloud in the following unit of time, the L-match must be changed so that the user's data can be stored in another cloud with sufficient capacity. We refer to this as the T-match. After the T-match process, the matching algorithm ends. Then, according to the match between the users and clouds, user data are stored in each cloud and then relocated from the cloud where user data were originally stored to the other cloud. We refer to this process as data balancing. Thus, user data placement is determined as a result of data balancing. In Sections 5.1 and 5.2, L-match and T-match are described in detail. Notations to describe the matching algorithm are summarized in Notations.

5.1. L-Match Algorithm for Minimum Latency. From the viewpoint of latency delay, when each user accesses the user's own data, the best cloud that stores each user's data can be found by measuring the distance between the user's location and each cloud's location.

In Section 5.1.1, we measure the distance between each user and the clouds. In a similar way, when each user's friends access the user's data, the best cloud that stores each user's data can be found by measuring the distance between the user's friends and each cloud. In Section 5.1.2, each user's data are stored in an optimal cloud, taking into account the distance between the user's friends and the clouds and the degree of familiarity between users in the OSN service.

Normalization is performed to apply the weight values obtained in those processes together to a linear programming algorithm [17, 18] which can find the L-match result. As a result of the normalization, all distance values are set from 0 to 1. The normalization formula [19] is as follows:

$$\widetilde{d_i} := \frac{d_i - \min\{d\}}{\max\{d\} - \min\{d\}}. \quad (1)$$

5.1.1. Calculating Distance Weight. A distance value indicates the distance between a user and a cloud. The distance weight is a normalized value used to apply the distance value as a weight to the linear programming algorithm. We assume that the user's location can be grasped using IP address.

For example, assuming that there are three clouds, as shown in Figure 7, if the distance between the user and each of the three clouds is 10 km, 50 km, and 100 km, the normalized distance weight value of each cloud is $\widetilde{d_0} = 0$, $\widetilde{d_1} = 4/9$, and $\widetilde{d_2} = 1$, respectively.

5.1.2. Calculating Social Weight. In OSN services, since the user usually accesses the friend's data more than their own data, it is difficult to obtain an optimal match considering only the location of the user.

Therefore, we measure the distance between the user's friends and each cloud in a similar way to the method of calculating the distance weight. Additionally, we make the friends with a lot of interaction to have more influence in finding the optimal cloud for storing the user's data.

For example, as shown in Figure 8, we can measure the distances between User-1's friends and Cloud-1. Then, to reflect intimacy, we multiply each distance by the social level value, which is the number of communications between User-1 and each of User-1's friends until just before the algorithm is executed. Therefore, the social weight value between User-1 and Cloud-1 before normalization is $9 * 10\,\text{km} + 6 * 20\,\text{km} + 5 * 30\,\text{km} + 3 * 40\,\text{km} + \cdots + 1 * 50\,\text{km}$.

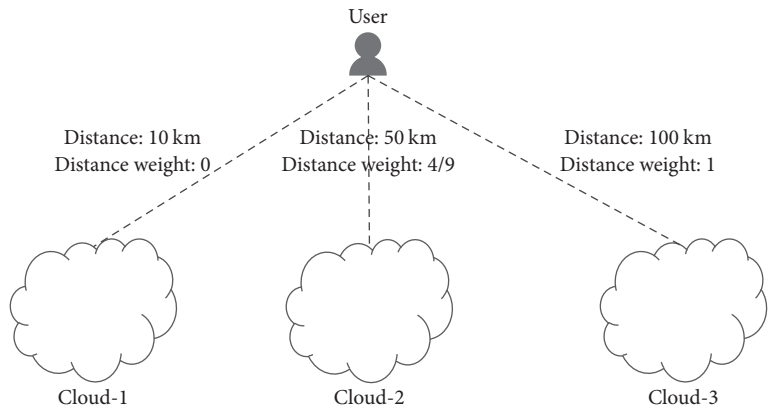

Figure 7: Examples of distances and distance weights between a user and multiple clouds.

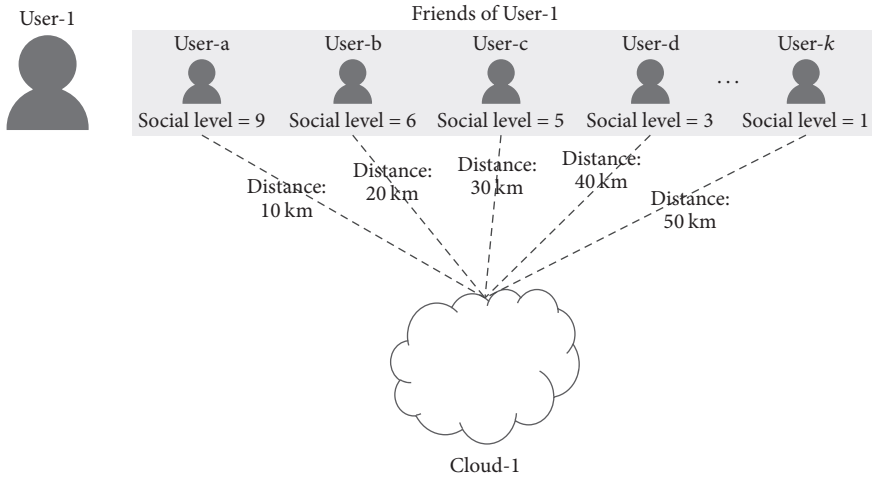

Figure 8: Social level and distance examples that can emerge between a user's friends and the cloud.

The social weight values for other clouds based on User-1 can also be obtained. By normalizing all social weight values, a normalized social weight value which is applicable to the linear programming can be obtained.

Therefore, the social weight value between a user and a cloud can be expressed as formula (2)

$$\sum_{f_k \in \mathrm{Fr}(u_i)} \mathrm{SL}\left(f_k, u_i\right) \cdot \mathrm{Dist}\left(f_k, c_j\right). \tag{2}$$

5.1.3. Calculating the Completed Bipartite Graph. The weight values for the edges between a user and a cloud can be obtained based on the distance weight value and the social weight value. The edge weight $w(u_i, c_j)$ between the user u_i and the cloud c_j can be expressed as formula (3), where $\mathrm{NormDist}(u_i, c_j)$ and $\mathrm{NormSocialLvl}(u_i, c_j)$ are the normalized values of the distance weight value and the social weight value, respectively,

$$w\left(u_i, c_j\right) = \alpha \cdot \mathrm{NormDist}\left(u_i, c_j\right) + \beta \cdot \mathrm{NormSocialLvl}\left(u_i, c_j\right). \tag{3}$$

α and β values may vary depending on whether the OSN service is more heavily handling the user's own data access or his/her friend's data access. If most of the user's activity in the OSN service is communication with a friend, it is better to increase the β value, and if the user accesses a lot of his/her own data, it is better to increase the α value. In terms of latency delay, formula (4) is a linear programming that is used to find the L-match. U and C indicate all users and the cloud

$$\begin{aligned} \min \quad & \sum_{u_i \in U} \sum_{c_j \in C} w\left(u_i, c_j\right) \cdot e\left(u_i, c_j\right) \\ \text{s.t.} \quad & \text{(a)} \ \sum_{c_j \in C} e\left(u_i, c_j\right) \le 1 \\ & \text{(b)} \ \sum_{u_i \in U} e\left(u_i, c_j\right) \le U \\ & \text{(c)} \ \forall e\left(u_i, c_j\right) \in \{0, 1\}. \end{aligned} \tag{4}$$

Constraints (a) and (b) mean that one user can be matched to only one cloud and (c) means all the edges can have a value of 0 or 1. If the value of each edge is 1, the vertices connected to the edge are matched. Among the edges

connected to each user, a certain edge having a value of 1 indicates that the edge has a smaller weight value than the others. Otherwise, if the value of each edge is 0, the vertices are not matched.

These constraints do not include any regulations about the location of user data, because most of the users do not have any data locality constraints and let the service to autonomously decide where to locate their data [20]. However, if it is necessary to regulate or limit the data location of a particular user, constraints for each user can be added to the constraints of formula (4).

In conclusion, the L-match process makes a match that consists of edges having a minimum weight value between each user and cloud.

After the L-match process, each user can be matched with a cloud that guarantees a minimum latency delay when data of all users are stored in multiple clouds in a nonduplicated manner.

5.2. T-Match Algorithm. The amount of traffic that occurs over time varies by location, and each cloud has a capacity limitation.

In this situation, in order to prevent the queueing delay due to the resource contention at the data layer, it is necessary to balance data into several clouds according to the capacity of each cloud and the amount of traffic. Because the traffic is generated by users accessing data, the traffic can be controlled by balancing the data.

We assume that SPs know the maximum response time that should be guaranteed to provide their service without problems in terms of performance. It is also assumed that the SPs empirically know the maximum amount of traffic allowed per unit time in order to ensure such performance with the specifications of cloud servers already in use. Therefore, the SPs can set the capacity of each cloud server using the traffic information.

Thus, the existing match result, that is, L-match result, should be properly changed considering the capacity of each cloud and the amount of traffic which occurred at each cloud. We refer to this process as the T-match. The T-match process can be divided into three stages: (1) classification of traffic type, (2) extraction of users, and (3) cloud selection.

In Section 5.2.1, the algorithm determines whether data balancing is necessary or not. In Section 5.2.2, when the amount of traffic generated at a cloud exceeds the capacity of the cloud, the algorithm decides which user's data should be extracted to balance the data. In Section 5.2.3, the algorithm determines a cloud to move the extracted user data.

After data balancing according to the T-match result, if the traffic that occurred in the following unit of time does not exceed $\mathrm{Ca}(c_j)$ for all clouds, the match is kept. Otherwise, if the traffic of c_j has exceeded $\mathrm{Ca}(c_j)$, the T-match is processed again considering $\mathrm{GAP}(c_j)$.

5.2.1. Classification of Traffic Type. The algorithm is performed differently according to traffic type, which is divided into two cases: *MINIMAL_TRAFFIC_CASE* and *OVERHEAD_CASE*.

(i) MINIMAL_TRAFFIC_CASE. MINIMAL_TRAFFIC_CASE does not require the T-match process. This is because TT is less than *Min-Ca*. In other words, this means that each cloud has enough capacity to accept the expected traffic without any overhead, no matter what the L-match result is. Therefore, the system maintains the L-match result which is made considering the latency delay.

(ii) OVERHEAD_CASE. When TT exceeds *Min-Ca*, it enters the *overhead_case* process. This is because there are chances that $T(c_j)$ exceeds $\mathrm{Ca}(c_j)$. The *overhead_case* is divided into two types: (1) *first_overhead_type* and (2) *continuous_overhead_type*. The *first_overhead_type* is the case where the algorithm encounters the *overhead_case* for the first time. The *continuous_overhead_type* is where the algorithm encounters *overhead_case* more than once.

For the *first_overhead_type*, T-match is performed according to $\mathrm{ET}(c_j)$ and $\mathrm{Ca}(c_j)$ based on the result of the L-match. For *continuous_overhead_type*, the T-match result created in the previous unit of time is used without the L-match result. Hence, for the *continuous_overhead_type*, the T-match is processed again considering $\mathrm{GAP}(c_j)$ only if $T(c_j)$ exceeds $\mathrm{Ca}(c_j)$. As shown in Figures 3 and 13, the amount of traffic does not change abruptly in general and naturally increases or decreases. Thus, it is possible to minimize the operation by performing T-match again using the existing T-match result, which is created in the previous unit of time, rather than using the L-match result each time.

5.2.2. Extraction of Users. Figure 9 shows the process of extracting users to make the T-match result when $\mathrm{ET}(c_j)$ exceeds $\mathrm{Ca}(c_j)$ in the *first_overhead_type* (or when $T(c_j)$ exceeds $\mathrm{Ca}(c_j)$ in the *continuous_overhead_type*).

The extraction process operates as follows:

(1) If $T(c_j)$ exceeded $\mathrm{Ca}(c_j)$, then $\mathrm{GAP}(c_j)$ should be calculated. It can be derived by subtracting $\mathrm{Ca}(c_j)$ from $T(c_j)$. If it is the *first_overhead_type*, $\mathrm{ET}(c_j)$ should be applied instead of $T(c_j)$.

(2) As shown in Figure 9, $t(u_i, c_j)$ is summed in order of the amount of traffic by each user until the summed value reaches $\mathrm{GAP}(c_j)$.

By transferring data of the users (ExtUsr), who are the target of the summation to the other cloud, the amount of traffic at c_j will be reduced in the following unit of time. At this time, the user who generated the maximum traffic at the previous unit of time is first chosen to balance the user's data to the other cloud. This is because users who use the OSN service the most at a certain unit of time are more likely to use the service at other times. Furthermore, in the situation shown in Figure 9, if $\mathrm{GAP}(c_j)$ is 20 and the users who used the service the most are extracted first, only User-A will be extracted. We refer to this as the most-first way. Otherwise, if the users who used the service the least are extracted first, only User-F, User-E, and User-D will be extracted. We refer to this as the least-first way. Whether it is the user group extracted by the most-first or the user group extracted by the least-first, the degree of influence on the service at the unit of

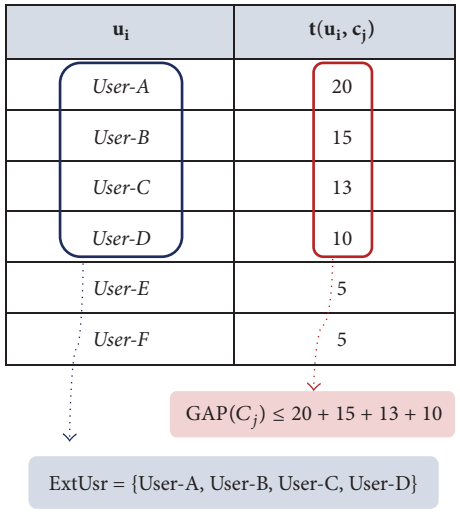

Figure 9: Extraction of users from the clouds with server overhead according to the amount of traffic by each user.

time is equal (20), considering the traffic generated by each group. In this situation, if the least number of user data can be moved, intercloud traffic can be reduced. Thus, we need to extract the users in the most-first way and transfer their data to the other cloud.

Such extraction can be implemented as shown in Figure 10. If there are clouds with server overhead *(overheadClouds)* caused by the traffic generated in the previous unit of time, it picks a cloud from *overheadClouds* and gets a list of users who accessed the cloud server in the previous unit of time *(userList)*. If there is no user in *userList*, then it picks another cloud from *overheadClouds* and gets *userList*. Otherwise, if there is user(s), it picks u_i from *userList* and gets $t(u_i, c_j)$. After then, the user is added to the extracted users list (ExtUsr), and $t(u_i, c_j)$ is added to *trafficSum* that is sum of $t(u_i, c_j)$ of extracted users. If *trafficSum* is less than GAP(c_j), it extracts another user. Otherwise, it picks another cloud from *overheadClouds* and repeats this process again. And, if there are no more clouds in *overheadClouds*, the algorithm ends.

After all users whose data should be balanced to the other cloud are extracted, the process explained in Section 5.2.3 should be used to determine which cloud each user's data will be stored in.

5.2.3. Cloud Selection. The cloud selection steps needed at the T-match process involve a "which cloud" to select in a situation where the user must be matched to another cloud other than the previously matched cloud.

Figure 11 describes the cloud selection algorithm for selecting a second-best (or third-best) cloud in terms of latency delay in the T-match process. The factors for selecting a cloud can be divided into the following four categories: (1) social weight + distance weight, (2) social weight, (3) distance weight, and (4) traffic applied in this order. If the cloud selection is not completed despite all four factors were applied, a cloud will be selected randomly.

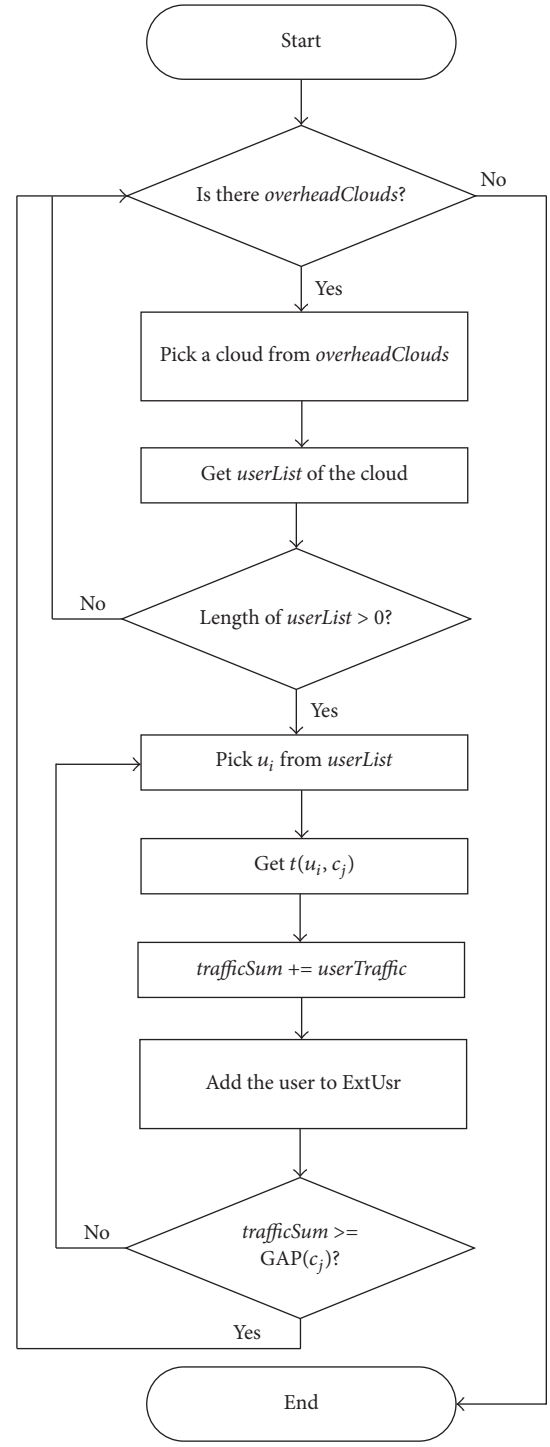

Figure 10: Algorithm flowchart for extracting users from the clouds with server overhead.

First, the algorithm determines whether the situation corresponds to *OVERHEAD_CASE* or *MINIMAL_TRAFFIC_CASE*, based on TT.

If it is *OVERHEAD_CASE*, it gets a user list *(userList)* from ExtUsr. If there are no users in *userList*, the algorithm ends. Otherwise, it picks a user from *userList*. After that, it checks

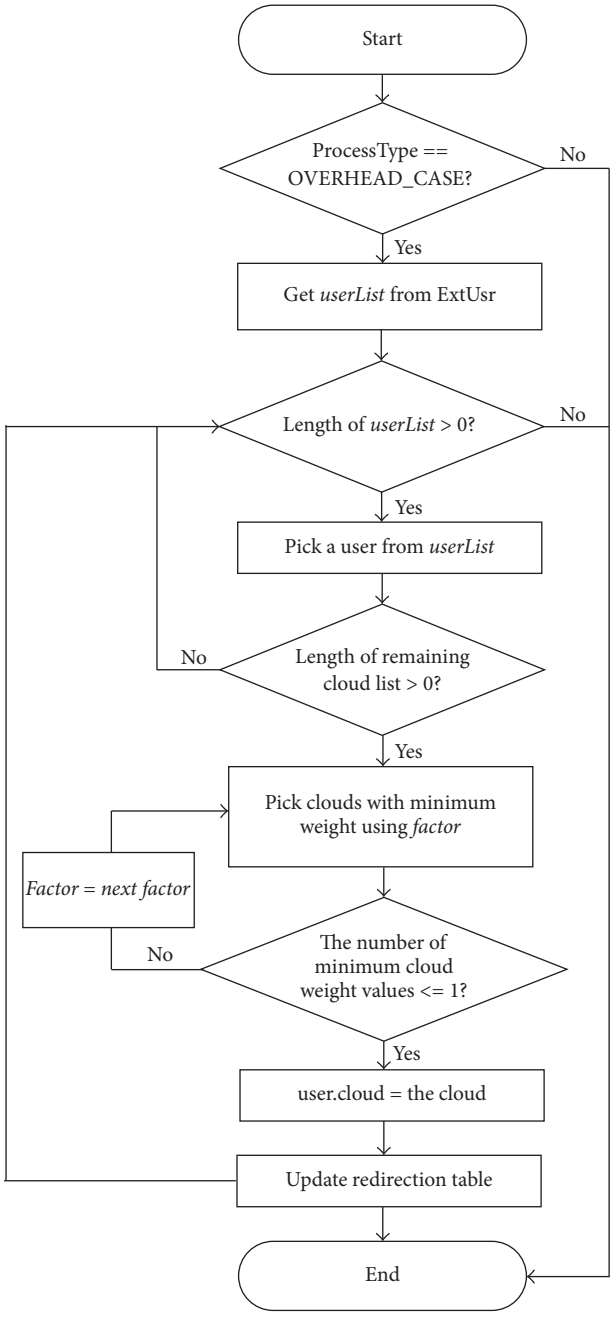

Figure 11: Algorithm flowchart for selecting a second-best (or third-best) cloud in terms of latency delay.

whether there are the remaining clouds that still have space to store user data according to $\mathrm{Ca}(c_j)$ and $t(u_i, c_j)$. If there is a remaining cloud, the cloud selection process starts to select the cloud to store the picked user's data. The algorithm repeats the cloud selection process for all users extracted in Section 5.2.2.

The cloud selection process operates as follows for each factor:

(1) Find the cloud having the minimum weight factor value among the remaining clouds except for the cloud that was matched.

(2) Match the cloud with the user if there is only one with the minimum weight factor value.

(3) If there is more than one cloud with the minimum weight factor value, repeat steps (1) and (2) using the next factor in order.

The reason for applying the factors in that order is the same as that for considering the social weight and the distance weight together in Section 5.1. As mentioned in this section, in OSN service, the optimal cloud for storing user data can be found in consideration of both the user's location and his/her friend's location. The ratio of accessing the data of the friend is larger than that of accessing the data of the user in an OSN service. This is why (2) social weight has higher priority than (3) distance weight. If the cloud selection is not made through the "social_and_distance," "social_only," and "distance_only" factors, the cloud with least traffic occurrence at the unit of time is selected to store the user's data. It may be the safest option to avoid server overhead, given $\mathrm{Ca}(c_j)$. In addition, as mentioned in Section 5.1.3, it is assumed that there is no regulation or constraints of user data location.

Consequently, we can minimize resource contention by considering $T(c_j)$ and $\mathrm{Ca}(c_j)$, while balancing the data with efficiency in terms of latency delay.

6. Evaluation

In this section, we demonstrate that our approach achieves better results compared to the baseline approach, which is used in both industry and academia. At first glance, Section 6.1 introduces how many clouds are set, how the interaction workloads are generated, what those components comprise, and what the parameters are. Then, Section 6.2 demonstrates that our approach can reduce the resource contention of the data layer at each cloud and provide an acceptable latency delay given the workloads and settings.

6.1. Experiment Setting. To validate our approach, we built a Multicloud infrastructure environment using 15 commodity computers (Intel i5-6600 3.3 GHz, 8 G RAM, and 1 TB hard drive), interconnected via a Cisco SF220-24 smart plus switch. Broker, EP, redirector, and service server software that we implemented was built on the abovementioned hardware environment.

6.1.1. System Configuration. Figure 12 depicts the system configuration used to evaluate our approach. We created three clouds to construct the Multicloud environment using eight computers. A cloud consists of two computers: the EP server and service server.

The service server has OSN service logic to deal with read/write operations. The database, as the data server at the data layer, was installed in each service server. We also made a broker server running the matching algorithm and a redirector server redirecting the user's request to a cloud. We used six computers to generate workloads on the basis of real-world data collected from Twitter, as described in Section 3, and stored them in the collected data server.

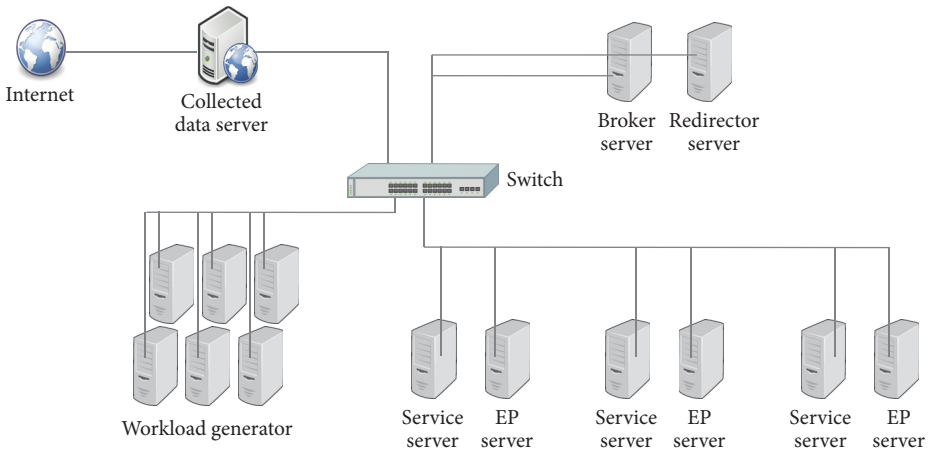

Figure 12: System composition for evaluation.

Table 1: Capacity of each cloud ($Ca(c_j)$) by traffic case.

Traffic case	Eastern US	South Central US	Western US
MINIMAL_TRAFFIC_CASE			
Our approach-min	10000 MB/h	10000 MB/h	10000 MB/h
OVERHEAD_CASE			
Baseline approach-90, our approach-90	117 MB/h	130 MB/h	130 MB/h
Baseline approach-70, our approach-70	90 MB/h	130 MB/h	130 MB/h
Baseline approach-50, our approach-50	65 MB/h	130 MB/h	130 MB/h

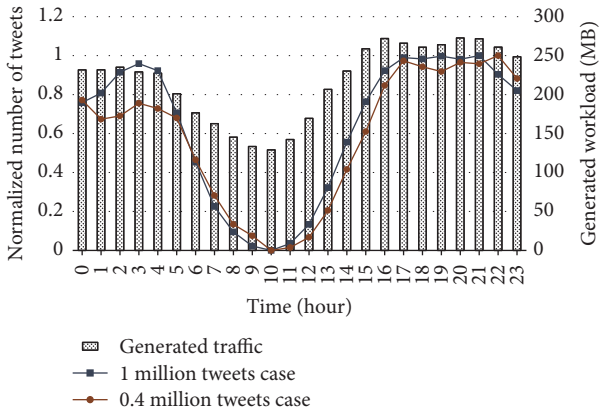

Figure 13: Amount of generated workload and pattern at each time point. The bar graph indicates the amount of generated workload at each time. The line graph indicates the normalized 1 million and 0.4 million tweets.

6.1.2. Interaction Workload. In our experimental setup, we generated interaction workloads using about 410,000 tweets, which are the writing operations of 2,500 users of the Twitter service. As shown in Figure 13, the number of generated tweets and the patterns according to time in 410,000 tweets are similar to those of 1,000,000 tweets as well as to all tweets on Twitter [21].

We used the ratio "R/W" to decide the total number of reading operations over that of writing operations. In our experiment, we evaluated the case of "$R/W = 100$" to reflect the fact that OSN services have much more reading operations than writing operations [22]. We set the size of each read and write operation to 1 KB, which is the average request size of OSN services [2]. As a result, the workloads were generated by time, as shown in Figure 13.

6.1.3. Cloud Settings. We set the three clouds virtually located in the Eastern, South Central, and Western US with its latitude and longitude values. These are actual locations of Microsoft Azure [7]. In order to make the distance between a user and a cloud seem to be geographically distant, we used formula (5), which creates a latency delay according to the distance between the user and the cloud [23]

$$\text{RTT (ms)} = 0.02 \times \text{Distance (km)} + 5. \quad (5)$$

The EP server sends the monitoring data to the broker server every hour. This means the matching algorithm and data balancing are also operated once an hour. We also assumed that the redirector server is close to the users and has enough resources to redirect users to the cloud with no delay.

Table 1 depicts $Ca(c_j)$ for each cloud, which is set in our experiments. We assume that if the amount of traffic which occurred at each cloud does not exceed $Ca(c_j)$, queueing delay of execution delay by resource contention will not occur. In order to evaluate *MINIMAL_TRAFFIC_CASE*, we set enough capacity to all clouds in order for all clouds to be able to accept the workloads without the queueing delay, whatever the results generated by the matching algorithm.

In order to evaluate *OVERHEAD_CASE*, we divided capacity into three types based on the maximum amount of traffic that occurred in the Eastern US. The maximum amount of traffic was about 130 MB/h. The numbers (90, 70, and 50) in the traffic case indicate the percentage of the maximum amount of traffic.

The reason for dividing *OVERHEAD_CASE* based on the Eastern US is that it generates the maximum traffic in Eastern US compared to the South Central and Western US in our experiments. Also, we could see how our approach works depending on the capacity and amount of traffic while gradually decreasing the capacity.

For example, in *baseline approach-90* and *our approach-90* of the *OVERHEAD_CASE*, we set the capacity of the cloud in the Eastern US to 90% of the maximum amount of traffic. On the other hand, the capacity of the others was set to the value of the maximum amount of traffic in the Eastern US. We assumed that the federation capacity of all clouds had enough resources to accept the total traffic generated by all users. Therefore, if some clouds did not have enough resources to accept $T(c_j)$, the others should be able to deal with $\mathrm{GAP}(c_j)$. Additionally, we assumed that if $T(c_j)$ exceeded $\mathrm{Ca}(c_j)$, it may cause performance degradation in terms of execution delay because queueing delay increase.

6.1.4. Baseline Approach. We compared our approach to a baseline approach, which stores all data into all clouds via master-slave replications. In our experiment, the master replication was stored in a local cloud with respect to each user and the two slave replications were stored in the other clouds. All the user's requests were processed at a local cloud. In addition, all the write operations of users resulted in intercloud traffic for updating slave replications.

6.2. Results

6.2.1. Comparison of the Intercloud Traffic. Figures 14 and 15 depict how much intercloud traffic is generated during each time duration. Figure 15 provides a more detailed view of the features from 11 h to 20 h in Figure 14. In the baseline approach, the intercloud traffic is generated by updating slave replications according to the write operations on the master replication in each cloud to the other clouds. Thus, intercloud traffic of the baseline approach during each unit of time can be calculated by

$$\sum_{c_j \in \mathrm{Cs}} \mathrm{WO}\left(c_j\right) \cdot N_{\mathrm{sl}} \cdot S_{\mathrm{msg}}, \tag{6}$$

where Cs and c_j denote all clouds and each cloud that is included in the clouds, respectively, $\mathrm{WO}(\cdot)$ denotes the number of write operations at a certain cloud, N_{sl} denotes the number of slaves, and S_{msg} denotes the size of each message.

In our approach, intercloud traffic occurs by moving all data created by the user, that is, data balancing. Thus, intercloud traffic in our approach during each unit of time can be calculated by

$$\sum_{u_i \in U_{\mathrm{db}}} \mathrm{TC}\left(u_i\right) \cdot S_{\mathrm{msg}}, \tag{7}$$

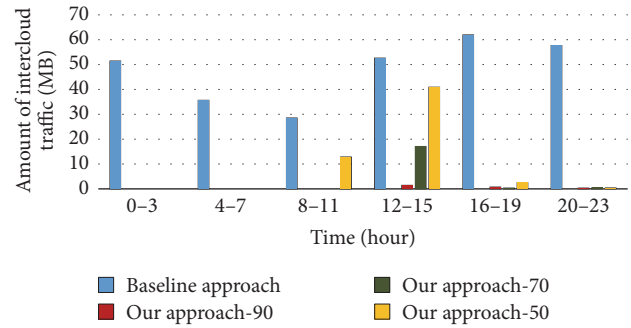

Figure 14: Amount of intercloud traffic by time during the day based on baseline approach and our approach.

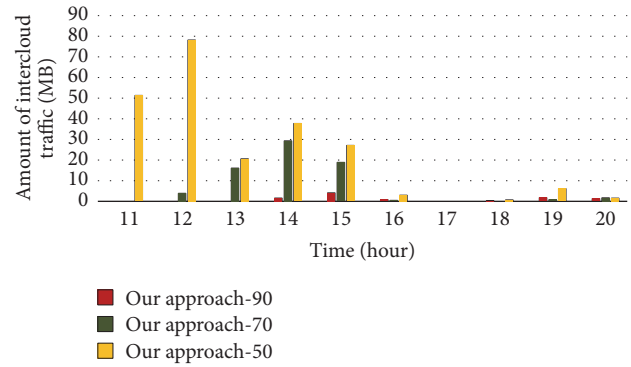

Figure 15: Amount of intercloud traffic by time during the day within our approach according to the size of cloud capacity.

where u_i denotes each user, U_{db} denotes users whose data should be balanced to the other cloud, and $\mathrm{TC}(\cdot)$ denotes the total number of contents of each user. In our experiment, as mentioned above, N_{sl} is set to 2 and S_{msg} is set to 1 KB.

As shown in Figure 14, in the baseline approach, the intercloud traffic occurs in proportion to the total amount of traffic occurring at the time because the write operation occurs every hour. Therefore, the intercloud traffic also increases at the time when the total amount of traffic is increased as well as when the peak is reached. It increases the resource contention of the data layer. Consequently, it will make server overhead.

Additionally, in our experiment, the number of slave replications in the baseline approach was set to "2," but the intercloud traffic will increase if the number of slave replications increases. In contrast, our approach generated intercloud traffic only when the data of users were balanced. Data balancing occurs only when the capacity of a cloud is not enough to accept traffic stably. This is why the intercloud traffic did not occur in the *MINIMAL_TRAFFIC_CASE*.

In the *OVERHEAD_CASE*, as shown in Figure 15, we saw that the smaller the capacity of Eastern US, the more data balancing occurred from earlier. The reason for this is that the traffic mainly increased during the day from 10 h to 19 h as shown in Figure 13, and the smaller the capacity is, which is set for each cloud, the faster the amount of traffic arriving reached its capacity. In addition, after data

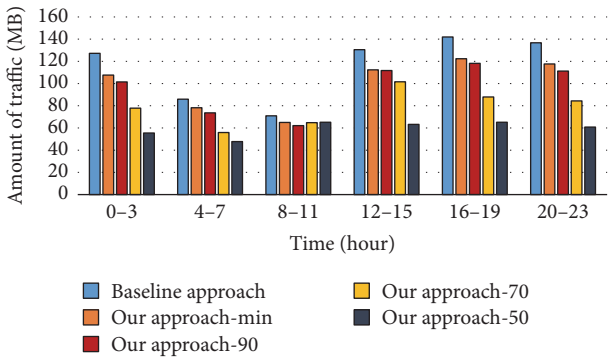

Figure 16: Amount of traffic generated by time during the day in Eastern US.

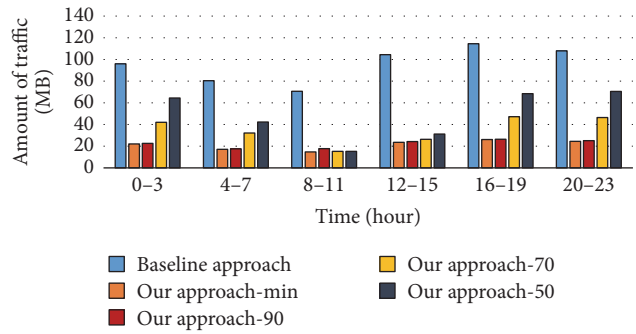

Figure 17: Amount of traffic generated by time during the day in Western US.

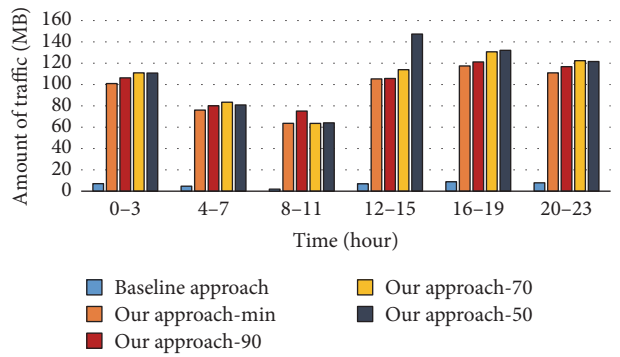

Figure 18: Amount of traffic generated by time during the day in South Central US.

balancing is performed for each capacity, the reason the data balancing continues (our approach-50: 12 h–16 h, our approach-70: 13–16 h, and our approach-90: 15-16 h) is that the total amount of traffic increased more than the amount of traffic in the previous unit of time. The fact that the amount of traffic of the cloud has increased after data balancing means that each user whose data are still stored in the cloud was not the target of data balancing in the previous unit of time (that is, the user did not use the OSN service at the previous unit of time) or the user has generated more traffic than that in the previous unit of time. Because our approach's data balancing is to optimize the system according to the capacity of the cloud and the amount of traffic that occurs in a certain unit time, if the amount of traffic in the following unit of time is greater than in the previous unit of time, then it can be a little over the capacity of that cloud. In order to mitigate this problem when applying it to commercial services, the capacity, which is set in the matching algorithm, should be set slightly lower than the actual capacity of each cloud.

Data balancing does not occur after the total amount of traffic is at its maximum at 20 h, because the data balancing was performed in order for each cloud to handle traffic at peak time. After peak time, the amount of traffic at each cloud is reduced. Therefore, the amount of traffic that occurs at each cloud does not exceed the capacity of the corresponding cloud. However, in our approach, the disadvantage is that the larger the total amount of data of each user whose data should be balanced to the other cloud, the greater the amount of intercloud traffic that will be generated.

6.2.2. Comparison of the Amount of Traffic by Users. Figures 16, 17, and 18 show the amount of traffic in our approach and the baseline approach by the capacity case. In our approach, the amount of traffic rarely exceeded the capacity of each cloud, which is defined in our experiment. On the other hand, the baseline approach almost exceeded the capacity because the baseline approach does not dynamically balance load or data among the clouds according to the amount of traffic.

Figures 17 and 18 show that the amount of traffic between our approach and the baseline approach was significantly different. This is because in the baseline approach, each user always sends the request to the closest cloud. That is, the baseline approach considers each user's own location only, whereas, in our approach, each user sends the request to the cloud with simultaneous consideration of the location of the user and the location of the user's friends. In other words, this result shows that, on the OSN service, not all of the user's friends are close to the user.

As shown in Figure 18, the amount of traffic in our approach was larger than that in the baseline approach. Nevertheless, the amount of traffic in the South Central US rarely exceeded the capacity of the cloud. The reason that the amount of traffic exceeded the capacity like the case of our approach-50 at 12–15 h is that our approach does not precisely control the traffic load itself, like a load balancer. Our approach predicts the load based on the usage of users in the previous unit of time and for changing the location of user data. Therefore, the amount of traffic can sometimes exceed our algorithm's expectations if certain users generate more traffic than the amount generated in the previous unit of time. Nevertheless, as shown by the amount of traffic at 16–23 h in our approach-50 in the South Central US, it can be seen that the amount of traffic was adjusted to the capacity via data balancing of our algorithm.

6.2.3. Comparison of the Response Delay. Figure 19 depicts execution delay according to the amount of traffic shown in Figures 16, 17, and 18. Execution delay consists of processing delay and queueing delay. Processing delay refers to the time

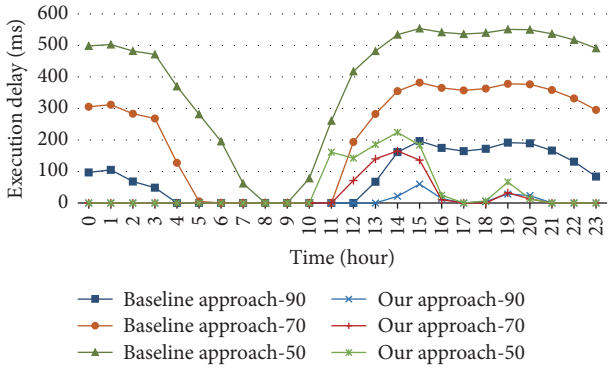

Figure 19: Execution delay by time during the day.

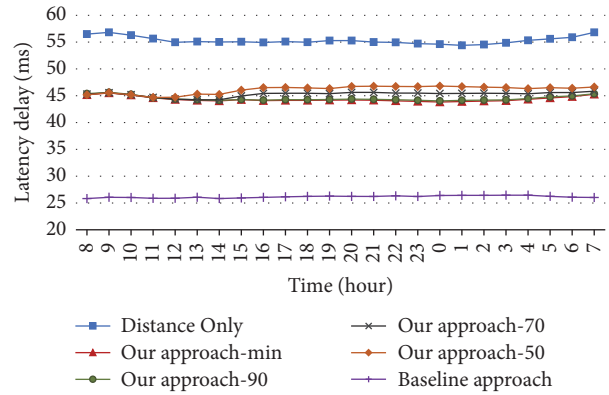

Figure 20: Latency delay by time during the day.

required for any processing of a packet. Queueing delay refers to the time a packet waits in memory before it is processed because another packet is currently being processed.

Processing delay was not considered because it is constant regardless of resource contention. Hence, we assume that processing delay is zero in our experiment. Therefore, the execution delay indicates the queueing delay by the resource contention. Execution delay was calculated by using Little's theorem [24]. The formula of Little's theorem can be expressed as follows:

$$\mathbf{N} = \boldsymbol{\lambda}\mathbf{T}, \tag{8}$$

where $\mathbf{T}$ denotes execution delay, $\mathbf{N}$ denotes the number of packets that was not processed immediately due to resource contention, and $\boldsymbol{\lambda}$ denotes the amount of traffic that occurred at each cloud in the unit of time.

Regardless of the baseline approach and our approach, we can see that execution delay increased as the server capacity decreased. For example, the execution delay of the baseline approach-50 was much higher than that of the baseline approach-90.

In the case of the baseline approach, execution delay was the lowest at around 9 h, when the amount of traffic was low during the day. This is because the traffic that occurred at each cloud did not exceed the capacity of that cloud. In our approach, execution delay occurred from 10 h to 20 h, which is the time of traffic increase (see Figure 13), but the increase was much smaller than the baseline approach. In addition, execution delay did not occur after 20 h, when the amount of traffic peaked. This is because, as mentioned in Section 6.2.1, the amount of traffic occurred at each cloud was reduced after 20 h. Thus, the amount of traffic which occurred at each cloud did not exceed the capacity of the cloud.

As a result, execution delay of our approach-90 was reduced by an average of more than 59%, compared to that of baseline approach-90. Execution delay of our approach-70 was reduced by an average of more than 67% over that of baseline approach-70. Execution delay of our approach-50 was reduced by an average of more than 82%, compared to that of baseline approach-50.

Figure 20 depicts a comparison of latency delay between users and clouds. "Distance Only" is a way of storing a user's data in a cloud close to the user in a nonduplicated manner. Through the results of "our approach" and "Distance Only," we can see the difference in latency delay between them, considering or not considering the social factor, when storing the user's data in the clouds. When our approach was used, the latency delay was lower than that when only the distance factor was used.

In our approach, the latency delay between 8 and 11 h was almost similar even when the capacity of each cloud changed, because the start time of our experiments was set to 8 h. The reason for setting the start time of the experiment at 8 h was that this was the time when traffic was the least, and traffic increased onward, peaking at 20 h. In this situation, we observed how the latency delay changes according to data balancing. The latency delay of the baseline approach was somewhat lower than that of our approach. Because, in the baseline approach, all user requests unconditionally access the closest cloud.

Users experience response delays through the summation of latency delay and execution delay. As shown in Figures 19 and 20 in our approach, the latency delay is higher than that of the baseline approach, but execution delay can be much lower than that of the baseline approach, resulting in lower overall response delay and better performance. In addition, the latency delay of our approach is still acceptable for both *MINIMAL_TRAFFIC_CASE* and *OVERHEAD_CASE* [25]. Therefore, OSN service's performance does not decrease even when execution delay does not occur.

In particular, in the *MINIMAL_TRAFFIC_CASE* where all clouds have enough resources to deal with the generated traffic at the unit of time, we observed that the latency delay was the lowest in the unit of time among our approach's results.

In the *OVERHEAD_CASE* (our approach-50, our approach-70, and our approach-90), the latency increased somewhat. This is because the T-match process selected the second-best (or third-best) cloud for the user in terms of latency delay. In the our approach-50, latency delay started to increase at 12 h because data balancing started to occur at 11 h. Likewise, in the case of our approach-70 and our approach-90, latency delay increased thereafter. In case of our approach-90, data balancing rarely occurred.

Thus, latency delay was almost the same as that in the *MINIMAL_TRAFFIC_CASE*.

In summary, our approach improved OSN service's performance compared to the baseline approaches by reducing resource contention of cloud servers and by guaranteeing the acceptable latency delay using social factors, simultaneously. In addition, since the amount of total data to store is less than the baseline approach, it can benefit more from the storage cost for storing the data. If the number of clouds used to store duplicated data is 1, then the storage cost can be reduced to 1/2. If the number of clouds is 2, then the cost can be reduced to 1/3. In other words, if the number of clouds is N, then the cost can be reduced to $1/(N+1)$.

7. Conclusion

Nowadays, there are numerous OSN services in a cloud environment, and users of these services are geographically disbursed across the world. By analyzing user traffic, we figured out key features generated by time, location, and social relationship levels between users.

In this paper, we introduced a novel approach of data placement for improving OSN service performance in a Multicloud environment. The redirector server, broker server, and EP server ensure minimal execution delay considering the amount of traffic in real-time and acceptable latency delay via discerning the distance between the users and clouds.

To validate our approach, we performed simulations with actual user data. We compared our approach to the other approaches that are using the replications in the Multicloud environment. Results indicated that our approach can reduce execution delay. In addition, we compared our approach to ones that are not using the relationship between the users. Results indicated that our approach can maintain acceptable latency delay.

In the future, we plan to improve our algorithm to more accurately adjust traffic volume via applying the characteristics of the OSN services to the machine learning technique. Furthermore, we will extend it to the dynamic VM management system that considers each cloud's condition.

Notations

d_i: Each distance value
$\max\{d\}$: Maximum distance value of all distance values
$\min\{d\}$: Minimum distance value of all distance values
$\widetilde{d}_i$: Normalized distance value of each distance value.
u_i: Each user of the total of I users on the OSN service, $i = 1, \ldots, I$
$\mathrm{Fr}(u_i)$: Friend set of u_i
f_k: Each friend in the $\mathrm{Fr}(u_i)$ of user u_i. $k = 1, \ldots, K$
$\mathrm{SL}(f_k, u_i)$: Social level value between f_k and u_i
c_j: Cloud among J multiple clouds, $j = 1, \ldots, J$
$\mathrm{Dist}(f_k, c_j)$: Geographical distance between f_k and c_j
$w(u_i, c_j)$: Weight value according to geographical distance between u_i and c_j
$\mathrm{NormDist}(u_i, c_j)$: Normalized values of the distance weight value between u_i and c_j
$\mathrm{NormSocialLvl}(u_i, c_j)$: Normalized values of the social weight value between u_i and c_j
$e(u_i, c_j)$: Edge between u_i and c_j
$t(u_i, c_j)$: The amount of traffic generated by each user in the previous unit of time
$\mathrm{ET}(c_j)$: The expected amount of traffic at c_j in the following unit of time via the existing match result and $t(u_i, c_j)$
$\mathrm{Ca}(c_j)$: Capacity of c_j for the unit of time
TT: The total amount of traffic for all clouds occurred at a unit of time
Min-Ca: Smallest capacity among the capacities of all clouds
$T(c_j)$: The amount of traffic that occurred at c_j in a unit of time
$\mathrm{GAP}(c_j)$: The volume of traffic that exceeded $\mathrm{Ca}(c_j)$.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Acknowledgments

This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (NRF-2015R1D1A1A01058928).

References

[1] N. Grozev and R. Buyya, "Multi-cloud provisioning and load distribution for three-tier applications," *ACM Transactions on Autonomous and Adaptive Systems*, vol. 9, no. 3, article 13, 2014.

[2] G. Liu, H. Shen, and H. Chandler, "Selective data replication for online social networks with distributed datacenters," in *Proceedings of the 2013 21st IEEE International Conference on Network Protocols, ICNP*, Goettingen, Germany, October 2013.

[3] L. Jiao, J. Lit, W. Du, and X. Fu, "Multi-objective data placement for multi-cloud socially aware services," in *Proceedings of the 33rd IEEE Conference on Computer Communications, IEEE INFOCOM 2014*, pp. 28–36, Toronto, Canada, May 2014.

[4] A. J. Ferrer, F. Hernández, J. Tordsson et al., "OPTIMIS: a holistic approach to cloud service provisioning," *Future Generation Computer Systems*, vol. 28, no. 1, pp. 66–77, 2012.

[5] D. Petcu, B. Di Martino, S. Venticinque et al., "Experiences in building a mOSAIC of clouds," *Journal of Cloud Computing*, vol. 2, no. 1, 2013.

[6] P. Pawluk, B. Simmons, M. Smit, M. Litoiu, and S. Mankovski, "Introducing STRATOS: a cloud broker service," in *Proceedings of the IEEE 5th International Conference on Cloud Computing (CLOUD '12)*, pp. 891–898, IEEE, Honolulu, Hawaii, USA, June 2012.

[7] Region of Azure, https://azure.microsoft.com/ko-kr/regions/.

[8] Google., Google Data Centers - Data center locations" https://www.google.com/about/datacenters/inside/locations/index.html.

[9] AWS Global Infrastructure" https://aws.amazon.com/ko/about-aws/global-infrastructure/.

[10] Amazon Route 53" http://aws.amazon.com/route53/.

[11] A. Amato and S. Venticinque, "Multiobjective optimization for brokering of multicloud service composition," *ACM Transactions on Internet Technology*, vol. 16, no. 2, article 13, 2016.

[12] Tweepy: Python library for accessing the Twitter API" https://dev.twitter.com/.

[13] Kissmetrics Blog:A blog about analytics, marketing and testing" https://blog.kissmetrics.com/science-of-social-timing-1/.

[14] "HubSpot: Where marketers go to grow" https://blog.hubspot.com/marketing/best-times-post-pin-tweet-social-media-infographic#sm.0000ddcgqm1w5de2w831dgm9tlqbd.

[15] "Optimizely : Grow your optimization and A/B testing skills" https://blog.optimizely.com/2015/07/08/how-to-find-the-best-time-to-post-on-facebook/.

[16] "PewResearchCenter: Numbers, facts and trends shaping your world" http://www.pewresearch.org/fact-tank/2014/02/03/6-new-facts-about-facebook/.

[17] Luenberger D. G., *Introduction to Linear And Nonlinear Programming*, vol. 28, Addison-Wesley, Reading, Mass, USA, 1973.

[18] Introduction to lp solve, http://lpsolve.sourceforge.net/5.5/.

[19] "Wikipedia: The free encyclopedia" https://en.wikipedia.org/wiki/Normalization (statistics).

[20] C. Esposito, M. Ficco, F. Palmieri, and A. Castiglione, "Smart cloud storage service selection based on fuzzy logic, theory of evidence and game theory," *Institute of Electrical and Electronics Engineers. Transactions on Computers*, vol. 65, no. 8, pp. 2348–2362, 2016.

[21] Buffer Social" https://blog.bufferapp.com/best-time-to-tweet-research.

[22] N. Bronson, Zach A., George C. et al., "Tao: Facebooks distributed data store for the social graph," in *Proceedings of the as part of the 2013 USENIX Annual Technical Conference (USENIX ATC 13)*, 2013.

[23] Y. Wu, C. Wu, B. Li, L. Zhang, Z. Li, and F. C. M. Lau, "Scaling social media applications into geo-distributed clouds," *IEEE/ACM Transactions on Networking*, vol. 23, no. 3, pp. 689–702, 2015.

[24] D. P. Bertsekas, G. Robert, and H. Pierre, *Data Networks*, vol. 2, Prentice-Hall International, Upper Saddle River, NJ, USA, 1992.

[25] Y. Li, G. Zhou, and B. Nie, "Improving Web Performance in Home Broadband Access Networks," *Wireless Personal Communications*, vol. 92, no. 3, pp. 925–940, 2016.