SaaS application mashup based on High Speed Message Processing

Zhiguo Chen[1,2,*], Myoungjin Kim[4], and Yun Cui[3,*]
[1]School of Computer and Software, Nanjing University of Information Science and Technology
Nanjing, Jiangsu 210044 CHINA
[2]Engineering Research Center of Digital Forensics, Ministry of Education, Nanjing University of Information Science and Technology
Nanjing, Jiangsu 210044 CHINA
[e-mai: chenzhiguo@nuist.edu.cn]
[3] School of Computer, Jiangsu University of Science and Technology
Zhenjiang, Jiangsu 212100, CHINA
[e-mai: ycui@just.edu.cn]
[4]Innogrid
Jung-gu, Seoul 04551, SOUTH KOREA
[e-mai: tough105@innogrid.com]
[*]Corresponding author: Zhiguo Chen and Yun Cui

Received Marchr 2, 2021; revised March 16, 2022; accepted March 19, 2022; Published May 31, 2022

Abstract

Diversified SaaS applications allow users more choices to use, according to their own preferences. However, the diversification of SaaS applications also makes it impossible for users to choose the best one. Furthermore, users can't take advantage of the functionality between SaaS applications. In this paper, we propose a platform that provides an SaaS mashup service, by extracting interoperable service functions from SaaS-based applications that independent vendors deploy and supporting a customized service recommendation function through log data binding in the cloud environment. The proposed SaaS mashup service platform consists of a SaaS aggregation framework and a log data binding framework. Each framework was concreted by using Apache Kafka and rule matrix-based recommendation techniques. We present the theoretical basis of implementing the high-performance message-processing function using Kafka. The SaaS mashup service platform, which provides a new type of mashup service by linking SaaS functions based on the above technology described, allows users to combine the required service functions freely and access the results of a rich service-utilization experience, using the SaaS mashup function. The platform developed through SaaS mashup service technology research will enable various flexible SaaS services, expected to contribute to the development of the smart-contents industry and the open market.

Keywords: SaaS mashup, Apache Kafka, message processing, recommendation, rule matrix.

http://doi.org/10.3837/tiis.2022.05.003 **ISSN : 1976-7277**

1. Introduction

Web-based technologies have attained a considerable stage of development. Especially with the popularity of smart devices, their influence has infiltrated every industry and affects everyone around the world. With the development of web-based technologies, computing powers that most businesses need come from cloud-computing services using the recent concept of "computing as a utility." Customers receive web-based applications by way of a new platform called Software as a Service(SaaS), according to the cloud-computing services. Customers have been fretting about understanding and utilizing a variety of functions that the applications provide. To solve that aspect of this issue, a new type of service was born and named the "cloud service brokerage" (CSB)[1-4]. It provides a platform for agencies with customers and businesses to safeguard the highest profit. In the past few years, CSB has received much attention and accomplished a new type of ecosystem, by merging cloud resources and services. CSB is a service and also a framework that gained various companies' acceptance of agents for their cloud services. With the agency services, it supports cloud services to customers according to their needs and conditions of economics, locations, and functional requirements. Along with the development of CSB, the SaaS mashup service was proposed as a concept, to manage the functions of SaaS applications utilized in an independent environment.

The concept of mashup services had been defined a long time ago, and mashup technology has been studied in many research areas, to improve the quality of life and enhance business values. Mashup services concisely describe various data that can be interconnected with each other, to give customers a novel user experience (UX)[5, 6]. Based on the concept of the mashup, the SaaS application mashup extracts many kinds of functions from SaaS applications and makes SaaS applications function via interconnection rules. These rules, also called "recipes," can be created by customers who can interconnect according to their functional needs, and supporters make regulations to help customers make rules. Depending on the SaaS application mashup service, we propose an extension concept that includes the newest functions—interconnection services and recommender services for supporting customization mashup services, called the SaaS mashup platform. The SaaS mashup platform can integrate a variety of functions of SaaS applications to meet customers' requirements for greatly improving applications' usability and practicability, to enrich customers' UX.

The SaaS mashup platform includes the SaaS aggregation framework (SAF) and log data binding framework (LDBF). SAF comprises an authorization technology and high-performance message-processing technology for interconnecting a variety of functions of SaaS applications[7-9]. The authorization is used to construct web interfaces of customers and administrators, respectively, called "web-based portal service." The Apache Kafka framework is used to classify the messages that declare the functions of applications, and it interconnects the functions based on the content of the messages, called "high-performance message processing[10-17]." Just as the name implies, LDBF collects the log data that records the information, including customers' service-use behaviors. According to the log data, LDBF extracts the data of customers, providing a customized recommendation service and proposing a novel concept called an "event processing rule matrix"[18-24].

The proposed platform focuses on supporting a more convenient environment, by using application interconnection and a rule matrix-based customized recommendation mechanism for satisfying customers' needs and giving customers a better UX. SaaS mashup services differ from general mashup services. The key point is to decide interconnecting more than two functions of any application with the open API that the developers support.

Contribution and paper outline. Our contribution in this paper is as follows.

Innogrid provided an independent cloud-computing environment, established for data processing, storing, and analyzing, and also constructed a web interface framework.To interact with each other's functions, the open APIs provided by the developers have been collected and analyzed, involving the interfaces of the big data analysis system and smart home system. More than 30 services, 130 functions, and 370 rules are declared for providing service mashup.A rule matrix-based recommendation mechanism adapted to the specifications of the SaaS mashup services has been proposed for enhancing the customer's UX satisfaction.

The structure of this paper is as follows. We discuss the differences between general mashup services and proposed SaaS application mashup services in section 2. Section 3 and 4 expose detailed parts of each function of two frameworks in the SaaS mashup platform. We evaluate and discuss in section 5 and 6 the result of the Apache Kafka-based event message-processing system for the functions interconnection and user experience, for utilizing the prototype of the platform. Section 7 concludes the paper.

2. Related Works

Overseas, CSB technology development is being actively carried out under the leadership of global IT companies and government. They have launched numerous services to form a profit structure, and core technologies are under research and development. The cloud service industry will gain momentum through the development of CSB technology around the world[4]. Like most of the IaaS-based service brokerage services, it is expected to grow continuously until 2025. Recently, a new SaaS mashup service that provides a service to relay various SaaS-based services and link SaaS functions is attracting attention. Unlike a general cloud mashup service, the SaaS mashup service provides a new type of service by linking SaaS functions, rather than by integrating data generated from heterogeneous clouds[25, 26]. Several sources describe the existing mashup service providing services that focus on data by combining various data in one web service and expressing it in a new format. AWS Marketplace, Dell Boomi, HP Aggregation Platform for SaaS, and Rackspace Cloud Tools Marketplace are the most representative mashup service types. In particular, the HP Aggregation Platform for SaaS enables the creation of new business models through the integration of various services from service providers. Such data-integration services can create new services, but several sources confirm that the mashup service uses a large amount of data and requires a complex analysis process, creating high demand for it. In addition, changing to a unified data format through data normalization, data sorting, and data format requires applying various algorithms.

The SaaS mashup service can create a connection between service functions through interworking with SaaS functions, with no need for data processing analysis and, thus, a new type of service according to the user's requirements[27-30]. Representative companies that provide such a service are IFTTT and Zapier. IFTTT, whose name is an abbreviation of "IF This, Then That," provides web-based SaaS mashup service that can arbitrarily link several separate services and applications existing on the Internet and computers[31]. In the past, users linked heterogeneous applications through direct coding. Now, this service enables linking service functions in an automated form.

IFTTT defines the fundamental service as "if this condition occurs" in one service, then "do it like this" in another service, forming a kind of automated application service to use through interworking between service functions. A user-defined service function interworking combination is called a "Recipe." Other users' recipes can be shared, and recommendations

are made based on the recipe's number of uses and ratings. This supports service linkages, focusing on such productivity services as email, photo management, storage, and notes.

Zapier, an SaaS interworking service similar to IFTTT, supports more service interworking than IFTTT, up to 100 cases per month for free use, with up to three types of interworking. "Zap" is the interworking rule between services, and the rules to be executed have detailed definitions. However, the user must bear responsibility for the grammar rules the service uses and understanding the API.

The existing SaaS mashup service provides a simple 1:1 service for interworking with SaaS functions but not a 1:N service suitable for diversified user requirements. In addition, as the 1:1 service is the main one provided, the economic burden on the user increases. The recommendation service for the user provides the results of a simple recommendation function, based on the service usage time and service rating[32, 33]. However, it does not support the customized recommendation service for the user through the service usage log analysis. In addition, the existing SaaS mashup service requires users to manage the inconvenience of having to understand and set the service function, interworking rule, and API directly, to create the service linkage using API. In this paper, we propose an SaaS mashup service platform that can compensate for these problems.

3. SaaS Mashup Service Platform

3.1 Main Part of the Platform

The SaaS Mashup Service Platform (SMSP) interconnects a variety of independent SaaS application functions, such as cloud-based business services, social network services (SNS), and legacy services. SMSP consists of an SaaS aggregation and log data binding frameworks. The SaaS aggregation framework includes interconnectable SaaS functions, "SaaS channels" that the administrator of the platform or the companies that establish the SaaS application services define. It also includes modules for message processing, SaaS application authentication, and management. The log data binding framework includes an event processing rule matrix engine, for real-time SaaS mashup events, and an SaaS mashup recommender engine, for service recommendation using collaborative filtering. Also, the construction of the SaaS mashup database enables storing service log data, channel information, and customers' information.

The basis for the design of SMSP was eliciting service requirements to provide the functions interconnecting mashup services and examining the validity of each technology to construct the whole platform. The design of the SaaS aggregation framework enables legacy services or new kinds of service functions to be cross interconnected for improving the utilization factor. Based on the SaaS aggregation framework, functions can be interconnected for data or information integration, every function's cross-connection can trigger corresponding events, and every event would be caught and written on the "log data" disk, including customer information and a time stamp. The event processing rule matching engine can create the log data. Analyzing the log data, the SaaS mashup recommender engine can provide customized recommender service for each user, to enhance UX.

3.2 SMSP Database

SMSP provides the functions interconnection by using the SaaS channel and constructs a recommender mechanism using log data for customers. SaaS channels, log data, and user information all must be stored in a reasonable place; we designed a database for SMSP called

"SaaS Mashup Description Database" (SMDD). SMDD includes 23 tables for SaaS channels, recipes, trigger channels, action channels, user information, historical log, recommender service data, and similar items. SMDD holds all the important data, and all of the important parts of SMSP keep a connection with SMDD all the time.

SMDD includes two parts—one for the legacy and general services and the other for the big data services.

4. SaaS Aggregation Framework

The SaaS Aggregation Framework (SAF) focuses on interconnecting SaaS channels. However, some SaaS channels serve as trigger channels, the initial segment in SaaS mashup services, and the others serve as action channels, the actuating segment. The recipe is just like a guide or a stipulation to interconnect a trigger channel, and action channels must follow. SAF also includes a web-based service portal that collects customer authentication of the SaaS application to the concrete unified authentication management module. The Apache Kafka-based message-processing module is the core of SAF.

4.1 Service Definition and Authorization in SAF

SaaS application gets centralized management and is "on-demand software." Customers must get access licenses to use the functions of SaaS applications on the web. Especially, most SaaS applications support RESTFUL API for additional utilization or extension by other organizations, so this paper calls them "SaaS channels." As mentioned above, trigger channels and action channels belong with SaaS channels, with trigger channels as the initial segment and action channels as actuating segments. A combination of a trigger channel and action channels constitutes a mashup service that comes from a defined recipe.

The word "recipe" refers in IFTTT to combining a function and another function, according to a rule that an administrator or service providers define. To provide a variety of SaaS mashup services, the usability of each channel distinguishes trigger channels and action channels, and they are registered in SMSP. In this paper, the recipe has two defining parts. The first is the basic construction, a trigger channel combined with an active channel and marked 1:1. The other is a trigger channel combined with more than one action channel and marked 1:n. This enhances the usability of various SaaS functions and improves UX.

To utilize the SaaS mashup service, customers must register a personal ID and services authentication that SaaS application supporters provide. SMSP includes a unified authentication module for registering and storing service and user authentication information. Just as its name implies, the SaaS application unified authentication module manages all services and user authentication data.

4.2 Kafka-based Message Processing and Function Interconnecting Module

The message processing and function interconnection module is the core part of SAF. The definition of the message must be clarified at this point. The messages include the user ID and the recipe ID to confirm who requires which services. The Kafka producer generates a message according to a cycle that each recipe defines.

Kafka adapts to process a huge number of logs in a distributed environment, based on the Pub/Sub mechanism[34, 35]. The producer generates messages and transfers the messages to the worker node, known as "the consumer." How to transfer the message from producer to worker node is the key point. As mentioned above, Kafka consists of the producer, the

consumer, and the topic. Topics store the messages already classified in partitions by the trigger channel of the recipe. Trigger channels connect with SaaS channels that are split into three categories, such as SNS, IoT, and business, that include Big Data. Based on the SaaS channel classification, the topics are also defined.

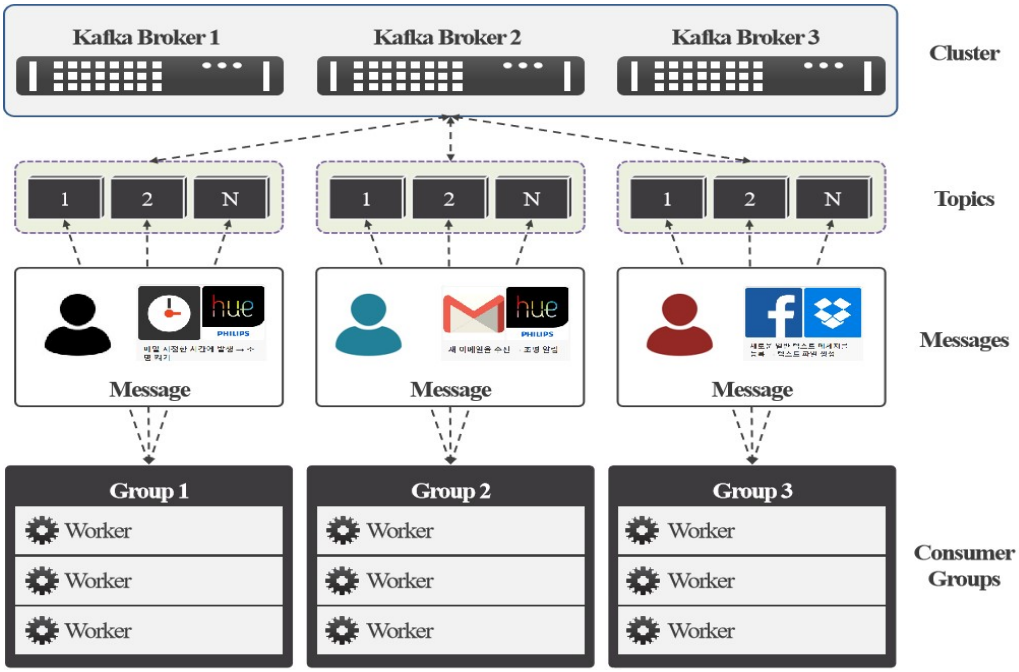

Fig. 1. The workflow of the message processing based on Kafka

Fig. 1 illustrates the workflow of the message-processing module, based on Kafka. As the diagram shows, the cluster includes three brokers consisting of a producer, three topics, and three consumer groups. Each consumer group includes three worker nodes whose main functions are to subscribe to messages from the specified topic, based on the Kafka message transmission mechanism, and interconnect SaaS application functions based on the recipe that the message includes.

Fig. 2 shows the interconnecting SaaS application functions in the worker nodes. The progress of interconnecting the functions is as follows. A worker node gets a message from a specified topic and extracts the recipe and user IDs. Using the recipe ID, the worker node queries the trigger channel and action channel(s) to obtain the token of each trigger and action channel by using the user ID. Then, it requests service permission from the specified SaaS application, using the token and REST API on the trigger side. The worker node gains data and transmits it to the action side. The workflow for obtaining service permission on the action side is the same as on the trigger side. After interconnecting the functions, the data of the end service includes tokens and the user ID (the recipe ID would be dropped), and the worker node starts to reload another message for mashup service. When a user completes a mashup service, we define this is as a created event. After completing the workflow of interconnecting functions, the worker node transfers the user ID and recipe ID to the real-time event handler included in the log data binding framework and drops them. The data that the worker nodes transmit are defined as event information, to be used for customized recommendation services.

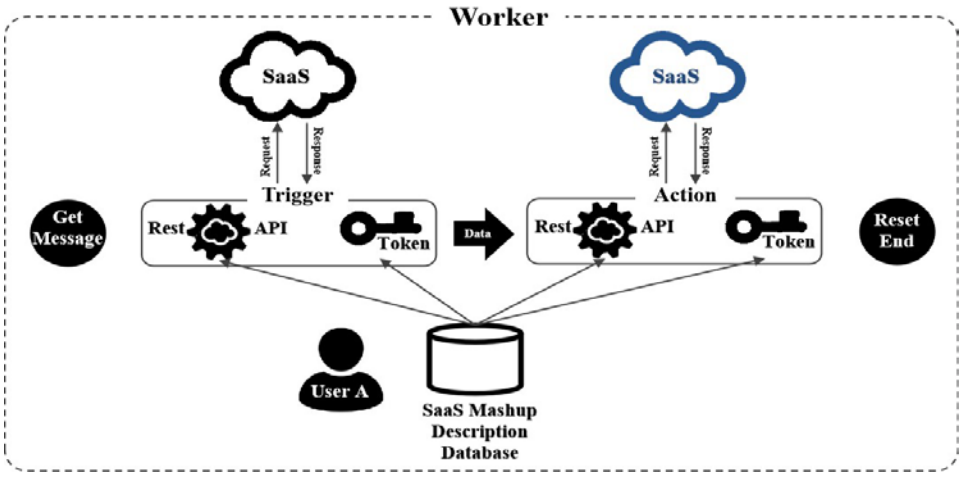

Fig. 2. SaaS mashup process

5. Log Data Binding Framework

The Log Data Binding Framework (LDBF) creates and stores an SaaS mashup service usage log, using the user and recipe IDs included in the event information. It collects and binds log data per user periodically, to support the customized SaaS recommendation service. To recommend the most appropriate mashup service to the user, a rule matrix mechanism is provided that depends on the mashup services usage history. We propose a two-stage approach to solving the cold-start drawback in the memory-based recommendation. The proposed LDBF comprises an event processing rule matrix engine and SaaS mashup recommendation engine. The event processing rule matrix engine includes a real-time event handler, a multiple SaaS functions splitting module, the event processing rule matrix, and a log creating/storing module. The SaaS mashup recommender engine includes a scheduler, log data collecting/processing module, recommender algorithm-based processing module, and individual recommended data storing module. The event processing rule matching engine creates and stores log data, using event information. The SaaS mashup recommender engine creates and stores user-appropriate mashup service recommendation information, using binding data that individual log data binding processing creates.

5.1 Event Processing Rule Matching Engine

The proposed SMSP provides SaaS application mashup services and, especially, also supports SaaS mashup service recommendation functions, based on service usage log analyzing and processing. To provide customized SaaS mashup recommendation services, the event rule matching engine involves a real-time event handler, a multiple SaaS function mashup separating module, and log data creating a module and the Rule Matrix.

SAF completes an SaaS mashup service for a user, then transmits user and recipe IDs to the real-time event handler, whose main job is to distinguish a 1:1 recipe from a 1:n recipe. Then, it transmits the 1:1 recipe ID to the log creating and storing module and the 1:n recipe ID to the multiple SaaS functions mashup separating module. The real-time event handler also transmits the user ID when transmitting the recipe ID.

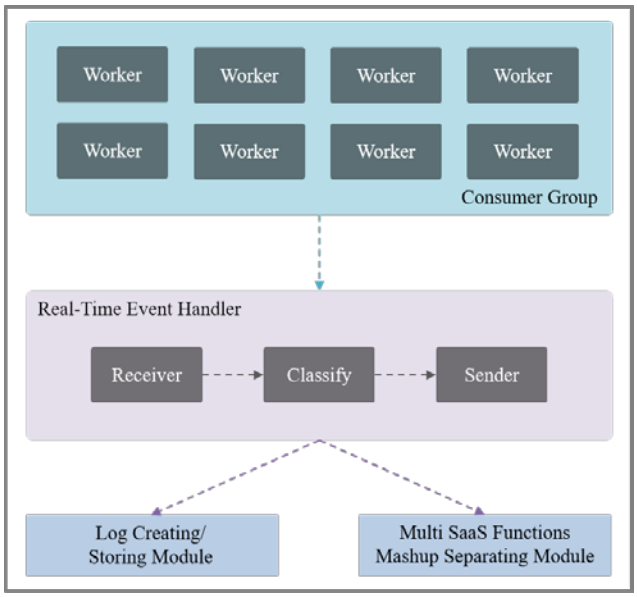

Fig. 3. Real-time Event Handler

The MSFMSM (Multi SaaS function mashup splitting module) splits a 1:n recipe into *n* 1:1 recipes, to support more effective recommendation services. The general SaaS mashup services usually support a 1:1 recipe; however, the proposed SMSP provides a 1:n recipe. The particular service approach can provide an effective mashup service for users, but there is a problem when processing log data binding to support a customized recommendation service based on users' hobbies. The recommendation service function analyzes users' history constructed from log data that includes a 1:1 recipe. However, the 1:n recipe consists of a trigger channel and *n* action channels. Suppose a user utilized a mashup service named "1:n recipe A" that includes a "1:1 recipe B," and the SMSP stores both A and B in the log according to the user. Then, recipe A and B are independent and not relational, even if recipe A includes recipe B and others. We cannot comparatively discover a relationship among the recipes for supporting users with an enhanced recommendation service. Hence, MSFMSM specializes in splitting a 1:n recipe into *n* independent 1:1 recipes. After splitting the 1:n recipe, MSFMSM sends the split recipes to the log creating/storing module with the user and cycle information. **Fig. 3** shows the workflow of MSFMSM.

The log creating/storing module creates log data according to user ID, recipe ID, and cycle information, supported by the event handler and MSFMSM. It also stores the log data in SMDD, to generate the historical log data, and sends the matrix module the log data at the same time for the recommendation service. **Fig. 4** illustrates the workflow of the log creating/storing module.

The proposed rule matrix supports two functions. First, it creates a matrix that includes the user ID and recipe ID, both of which are in the recommendation list that the recommendation engine creates. Second, it receives log data from the log creating/storing module to verify the accuracy of the recommendation list. The rule matrix is defined to provide more customized recommendation services, comparing the usage services with recommended services through analyzing the rate of usage in the recommendation list.

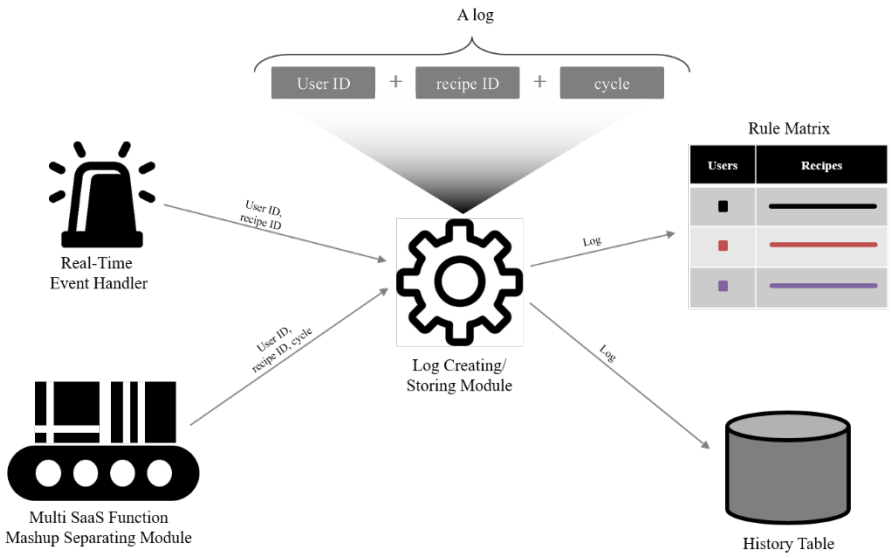

Fig. 4. The Workflow of Log Data Creating and Storing

5.2 Service Recommender Engine

The purpose of the service recommender engine is to recommend a piece of customized service information according to an analysis of the users' behavior log data. The service recommender engine includes a scheduler, the log data collecting and processing module, the recommender algorithm, and an independent user-based behavior storing module. Each part interacts with the others to provide a recommending service.

The recommendation approach consists of two steps. To solve the cold-start issue, the first step offers recommendation services utilizing the applications that users select when they register with SMSP. When users first get in touch with SMSP, there is no log data for them, so SMSP cannot recommend services to those users. Therefore, when the users register with SMSP, they must choose three applications that interest them or that they frequently use. According to the selected applications, SMSP recommends some interconnected services mostly used in SMSP. However, the recommended services cannot precisely fit these users, so the service recommender engine supports the second step.

The second step provides a customized recommendation service to analyze the users' hobby using the log data. The services rating is generated by using service duration accumulative calculation, not based on users' ratings. The range of the rating is from 10 to 1, and 0 expresses unused services. In this paper, Collaborative Filtering is used for the recommendation service. The duration of analyzing log data once is 24 hours.

6. Benchmark of High-Performance Message Processing

SMSP transmits recipes to worker nodes using the Apache Kafka-based high-speed message-processing framework for supporting SaaS mashup services. The framework is one of the most important parts of transferring the message to include the recipe, user information, and cycling from producer to worker nodes. Kafka can more rapidly deliver massive recipes that many users create simultaneously, based on distributed message processing. This section presents

the number of messages that can be transmitted per second, through performance tests on Kafka producers and consumers in a distributed environment. It also shows the processing performance of Kafka through digitized message throughput. The reason Kafka is suitable for the SMSP it proposes appears by comparing Kafka and RabbitMQ, which provides the message-processing function in a similar structure.

6.1 Performance Test System Environment

Kafka, which provides high-speed message processing in SMSP, was installed in a cluster using three virtual nodes. The proposed platform was built as a test node, based on Cloudit 5.0, and tested in an environment where running SMSP can have a significant impact on Kafka's actual performance. Therefore, the environment utilized in this performance evaluation was installed with the same structure as the virtual environment, using three physical computer nodes, and Kafka's performance was tested. **Table 1** shows the specifications of the physical computer nodes used in the Kafka performance test.

Table 1. H/W for Performance Test

Content	Specification
CPU	Intel® Core ™ i3-2120 3.3GH
Memory	8GB
HDD	2T
Network Card	Intel Corporation 82579V Gigabit
OS	Ubuntu 14.04.5 LTS

For the Kafka test, a three-node computer cluster was configured with the same specifications as **Table 1** shows, and **Table 2** shows the software environment for the Kafka installation and performance evaluation. Also, RabbitMQ's software version for comparison with Kafka appears in **Table 2**. For analysis and comparison of actual message throughput, the software itemized in **Table 2** was installed on all nodes.

Table 2. S/W for Performance Test

Software	Version
Apache Kafka	2.11-0.10.2.1
Apache ZooKeeper	3.4.10
Java	1.8.0_131
RabbitMQ	3.5.15

6.2 Experimental Method

SMSP creates a recipe using the trigger channel and action channel the user selects through a web portal. The producer creates a message containing the recipe ID, user ID, and the cycle, and delivers it to the broker. The broker stores the received message in the partition of the predefined topic and waits for the request from the consumer. The requesting consumer

receives a message from the broker with the partition of the specified topic. In SMSP, the consumer includes the worker, and the worker extracts user information, recipe information, and cycle information from the received message, to process the data. The message the producer generates is composed in JSON format, the size is 40 bytes, and the data included appears in the example below.

```
{“user_id”: 36, “recipe_id”: 878, “cycle”:1}
```

Kafka provides two script programs that can test its performance on a system that developers built—more precisely, the programs can test the performance on the respective producer and consumer sides. The performance test program for the producer is the throughput per second of messages transmitted through the partition. The performance test program for the consumer measures the throughput per second of messages received over the partition. The consumer performance test program measures the throughput per second when outputting a message from a partition of a topic that the consumer determines. Through the test programs, we can obtain the result of performance measurement for system evaluation.

The performance test programs Kafka provides are “kafka-producer-perftest.sh” and “kafka-consumer-perf-test.sh,” respectively, included in Kafka's installation package. There are various setting fields for testing the message transmission amount of producers and consumers using a performance test program, some of which are optional and some required. A performance test requires entering items in the fields marked “required” in the remarks column of each table. Also, some fields contain default values when there is no input value.

Kafka's performance varies depending on the partition and replication settings of the topic defined in the broker. To this end, we define the topic with four different setting values, to test the performance of producers and consumers. The created topic appears in **Table 3**.

Table 3. Topic List in Kafka

Topic ID	Configuration
A	3 partitions, 3 replications
B	1 partition, 3 replications
C	3 partitions, 1 replication
D	1 partition, 1 replication

“Topic” is a term for getting messages in Kafka. The producer and the consumer can input/output a message only by specifying a topic. In the setting section of the topic (see **Table 3**), “partition” refers to the storage of the actual message. One topic must define more than one partition and cannot be used without defining one partition. Both producers and consumers either send or receive messages to or from the topic's partition. In Kafka, which has a pub/sub structure, the consumer determines the topic for receiving the message, and the quantity of the consumer determines the message. In **Table 3**, replication refers to the number of copies of the partition. Kafka provides fault-tolerant functions as the general distributed system. Kafka can select another leader to compensate, when the leader operating the partition is down, to receive or output data again.

Performance tests for producer and consumer were of two types: a single test and a simultaneous test. In the single test, the producer was tested for performance, and then the

consumer was tested for performance. In other words, by independently testing the performance of producers and consumers, the input/output of messages proceeds separately, so there is no mutual effect. Simultaneous testing runs producer and consumer performance test programs to affect message input/output performance.

In a single test, the message transmission rate per second of the producer is measured by inputting a message using the producer at the same computer node, and the message transmission rate per second of the consumer is measured by outputting the input message using the next consumer. This enables verifying the absolute performance of producers and consumers through independent tests.

Simultaneous testing verifies each performance by running producer and consumer performance test programs on the same computer node at the same time. The simultaneous test enables checking the relative performance because the producer inputs the message, and the consumer consumes the message.

Since the system that provides the message-queuing function delivers the message (data) from the producer to the consumer according to a certain rule, the message transmission amount per second confirms its performance. The single test and the simultaneous test ran the same script program to verify performance.

In the execution command to perform the producer performance test, “--topic” specifies the name of the topic and enters a message in the topic partition. When the message is entered, “--throughput” is the amount of message input per second. Throughput was set to 2,000,000 for all execution codes, and the maximum throughput was set. The size of a single message is “--record-size,” and the message SMSP generates is 40 bytes, so it is set to 40 for all execution codes. “--Num-records” determines the number of messages to be input. It is set to 100 million in the execution code and measures the number of messages input per second until 100 million messages are input. The expression “--producer-prof” is defined as a URL indicating the location of the broker, but in the case of a cluster, it lists all the URL addresses of the nodes.

“--Topic” in the execution command for conducting the consumer performance test determines the name of the topic to be delivered. “--Show-detailed-stats” specifies that detailed progress is displayed when the message is displayed. “--zookeeper” is a list of URLs of the ZooKeeper nodes, run by specifying the data flow of the message and the ZooKeeper servers that manage the Kafka cluster. “--Message-size” matches the producer's “--record-size,” and the size of the message sent by SMSP to all executable codes is 40 bytes. “--Messages” determines the total number of messages the consumer should consume, set to 100 million in all executable code. The term “--thread” should be defined according to the number of partitions in each topic, i.e., the number of consumers in the consumer group, and delivers messages in association with at least one consumer per partition. Topics A and C are three partitions, so “--thread” was set to 3, and topics B and D were one partition, so 1 was set.

In the performance test of producer and consumer, the message transmission per second measures the average transmission rate per one time. The quantity of messages the producer transmits is measured in units of 5 seconds (once), to calculate the number of messages sent per second, and the consumer checks performance in the same way. In comparison with RabbitMQ, the average performance per rabbit is calculated and compared.

6.3 Kafka Performance Test

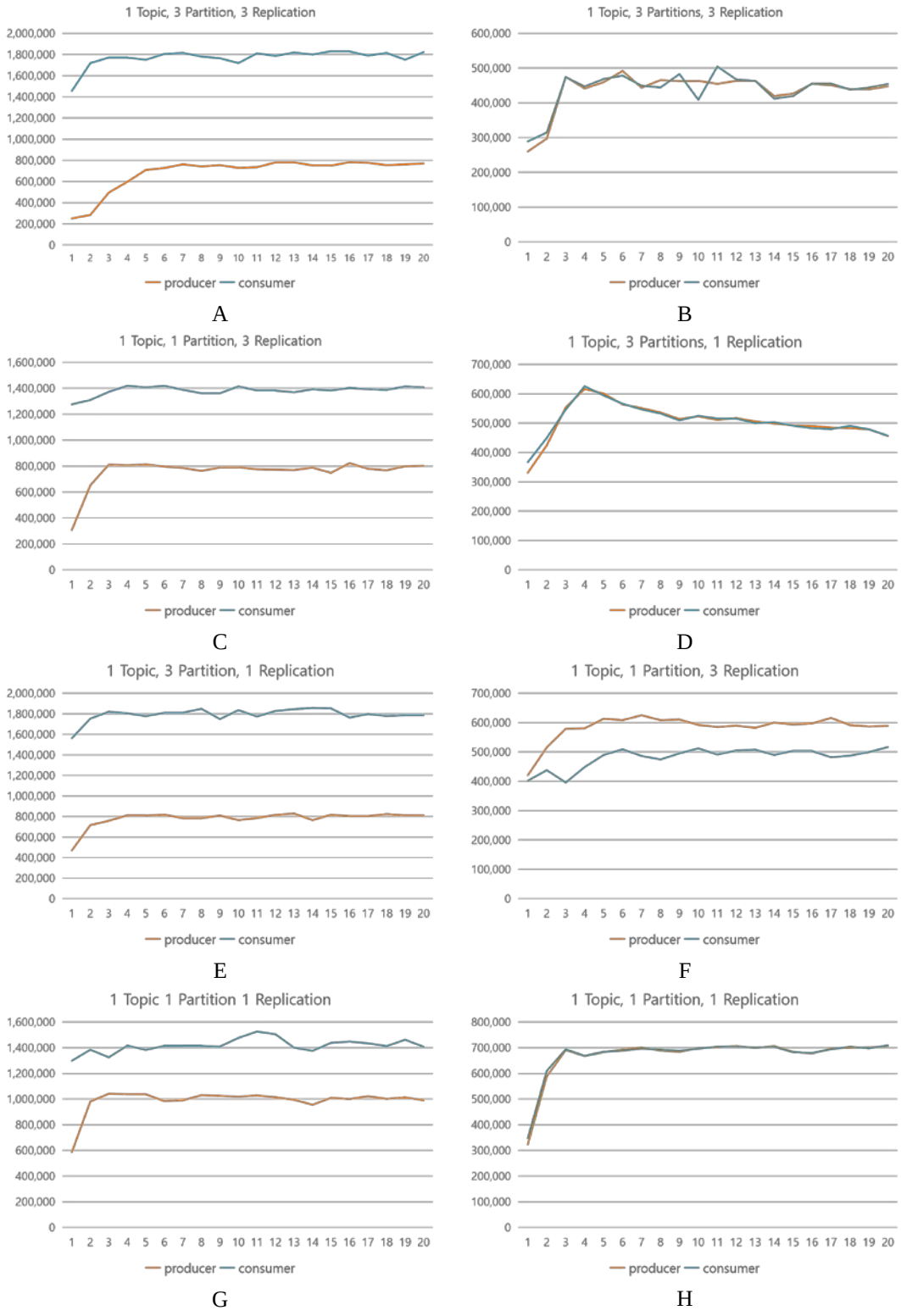

Fig. 5. Performance of Kafka in SMSP

The performance results of the individual tests and joint tests of Kafka producers and consumers using topics A, B, C, and D appear in the charts in **Fig. 5**. There, the x-axis means the number of times to check the average message transmission per second, by defining 5 seconds as one time in the continuous message transmission time. The y-axis means the number of times in the continuous message transmission time for 5 seconds. It represents the calculated average message transmission volume after sending a message. As an example, from the contents of A in **Fig. 5**, the value of the y-axis displayed on the x-axis 1 in the producer's performance is 250,644. This means that through Topic A, the average number of messages transmitted per second by the Kafka producer sending to the Kafka broker in the first 5 seconds is 250,664 messages. The figure shown on the x-axis number 2 (see below) is the average message transmission per second from the producer to the broker in the second 5 seconds. As such, in the chart, the x-axis represents the number of times in units of 5 seconds, and the y-axis represents the average message transmission per second.

According to the performance test results A, C, E, and G in **Fig. 5**, the performance of the consumer exceeds that of the producer. This is in the settings of all topics, so when the producer creates a message and inputs it to the partition, Kafka's controller affects it. The controller manages partition and replication in Kafka and sends partition information to ZooKeeper. Messages the producer generates are input to the partition the controller designates. When one topic consists of multiple partitions, the controller is in charge of distributing the message to be input. In addition, the controller operates the replication mechanism. A copy of the message applied to each partition is distributed to each broker, according to the designated replication quantity, and consuming quantities of computer resources in the distribution process. In addition, the producer receives an acknowledgment from the leader of each partition. However, waiting for this acknowledgment takes time. For this reason, when receiving a message from the producer, Kafka causes much latency, due to the time the action of the controller and the acknowledgment consume, reducing the amount of message transmission per second. A consumer can directly request a message from ZooKeeper, using the partition location and offset information, and receive a message determined to a topic. In addition, the consumer can receive the file directly from the message storage location by using the socket without sending the message through the controller, which is different from the producer. For this reason, the consumer shows better performance than the producer.

Checking the results B, F, and H in **Fig. 5** shows that the message transmission per second of the producer and the transmission of the consumer are insignificant. If this is a concurrent test, the producer inputs a message to the partition; then, the consumer can consume the message, so the consumer waits for the message the producer inputs, and the performance is similar. In conclusion, the producer's performance directly affects the consumer's performance in the concurrent test, compared to the single test. In addition, the producer's performance in the simultaneous test is worse than the producer's performance in the independent test, a result of the controller action, replication action, and ZooKeeper action through the consumer proceeding in a complex way. However, the result of B in **Fig. 5** shows the performance of the producer is better than that of the consumer. The reason is a problem that occurs in the process of copying messages by replication.

As **Fig. 5** illustrates, Kafka's performance shows a significant difference according to topic setting in concurrent tests. Of course, the latency that occurs when inputting messages from the producer determines this difference. The one that showed the best performance was using

topic D, and the producer's performance was the best in an environment with one partition and one replication. Next, the test results using topic C show good performance because much cost occurs in message replication under the load the controller generates. However, the performance deteriorates over time because the load is greater when the controller distributes messages to the partition than the load on replication. According to the test results using topic A, although the performance fluctuates, it maintains a constant performance. The test result using topic B shows the producer's performance is better than the result using topic C and inferior to the result using topic D, but the consumer performance falls between the results using topic A and topic C. The reason is that it degrades the consumer performance in the process of copying the message with the replication setting.

Analysis of the results using topic B in the simultaneous test is as follows. Replication Kafka provides distributes messages stored in partitions to nodes in the cluster, leaving replicas, to provide a fault-tolerance function and select a replacement node when the partition leader is suddenly down, to use as a leader. Here, copying the message stored in the partition to another node or your own node is the same as outputting a message from the partition, just like an actual consumer. Therefore, topic B has three replications per partition, so messages stored in the actual partition are consumed by four consumers. In this process, because one resource is consumed by consumers, it results in performance relatively inferior to the producers'. On the other hand, topic A uses three replications, but this is similar to the producer's performance because a large number of consumers consume a small number of messages in a distributed message storage environment.

6.4 Performance Comparison Between Kafka and RabbitMQ

SMSP provides a message-sending function using Kafka. Kafka and RabbitMQ are the platforms most used to support a message-queue service with many existing pub/sub structures[36]. Therefore, this paper compares Kafka and RabbitMQ, which provide message-queuing service using the message-processing method in the same structure. There is no test program like Kafka in the RabbitMQ package. Therefore, the test was conducted using "rabbitmq-perf test-2.2.0", a RabbitMQ performance test program provided by GitHub. Unlike Kafka, "rabbitmq-perf-test-2.2.0" cannot test the performance of producer and consumer separately. That is, the test is carried out by creating producer and consumer in one node. To compare them under the same conditions, in Kafka, producer and consumer were executed on the same node, and that test result was utilized. The test results of RabbitMQ and Kafka appear in **Fig. 6**. The results in **Fig. 5** show Kafka's much better performance than RabbitMQ's. According to the comparison results, in the same environment, the producer of Kafka shows an average performance of 669.816 Msg/sec, and the producer of RabbitMQ shows a performance of 45,618 Msg/sec. The Kafka consumer shows an average performance of 672.366 Msg/sec, and the RabbitMQ consumer shows an average performance of 41,180 Msg/sec. In these results, the producer standard deviation of Kafka is 83,393, and that of RabbitMQ is 16,068; the consumer standard deviation of Kafka is 77,300, and RabbitMQ's is 2,613. In terms of message transmission rate per second as an evaluation criterion, Kafka shows more than 14 times the performance of RabbitMQ. However, RabbitMQ shows more stable performance.

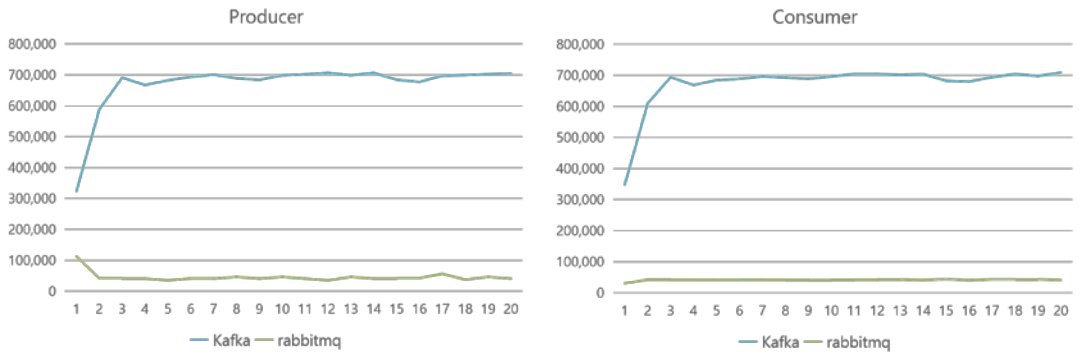

Fig. 6. Performance Comparison

With poor performance and good stability compared to Kafka, the RabbitMQ results can be explained in three main parts. The first is in the way the message is transmitted. Kafka can transmit a large number of messages at a time, based on batch processing, but RabbitMQ uses a general message-queue method. If consumer messages are delivered through a pull method, worse results will be obtained than the measured results, because the consumer consumes only one message at a time during the pull process. In addition, considering the situation where a large number of messages are stored in the broker's queue, but consumers cannot consume them, RabbitMQ provides a QoS limit mechanism. This is the second problem. If the amount of messages that consumers cannot consume is input from the RabbitMQ producer, the QoS limit mechanism reduces the number of producer messages sent per second. This result appears in “A” in **Fig. 6**. The speed of the first producer showed a performance of 112,851 Msg/sec, but in the next stage that dropped to 41,625 Msg/sec. This proves that the QoS limit mechanism reduces the producer transmission amount. Lastly, RabbitMQ provides stability in message delivery by sending an acknowledgment. Both the producer and consumer of RabbitMQ check the acknowledgment for message delivery from the broker and then proceed with message input/output.

Kafka also provides the acknowledgment function, but only for the producer, and it is divided into synchronous and asynchronous. These differ in that the producer stores messages in partitions on the broker and provides replication capabilities. The synchronous type sends the result of checking up to the state of saving replicas in each node, as an acknowledgment of the data stored in the leader's partition. Asynchronously, only the acknowledgment of the data stored in the leader's partition is sent to the producer. Using the asynchronous method can result in irretrievable data loss if the leader goes down. For this reason, we adopt the synchronous method in general and propose the asynchronous method only for systems that need to maximize the performance of Kafka.

SMSP utilizes a message-queue system to transmit JSON data, defined with a single user ID, recipe ID, and cycle structure, as a message. In addition, considering the situation where many users can use the SaaS mashup service by creating recipes in SMSP, this provides a message transmission function using Kafka, which transmits numerous messages per second.

7. Conclusions

The proposed SaaS mashup service platform consists of SaaS aggregation and log data binding frameworks. The SaaS aggregation framework provides web-based portal services, including SaaS service authentication and operation functions, and defines SaaS channels, trigger channels, action channels, and recipes to provide SaaS mashup services. In addition, high-performance message-processing technology using Apache Kafka is supported, to provide a quick SaaS mashup service according to the recipe the user creates. Users' SaaS mashup service information in the SaaS aggregation framework is stored as a history log by the event processing rule matching engine, and user-specific binding data extracted from the log is utilized for customized recommendation service. In addition, a rule matrix was defined to recommend services suitable for users, and an SaaS mashup recommendation engine was implemented, using memory-based cooperative filtering.

The SaaS mashup service platform that this paper proposes has a basic service form in providing SaaS mashup service. However, to provide a better service environment, it derives functional elements necessary for service improvement and includes a policy or legal part as a pilot service. In the later stage, machine learning and knowledge graph technology will be integrated according to the constantly changing service requirements, so as to provide users with better recommendation services and strengthen the shortcomings of the platform in recommendation functions. We will do our best to complete it as a product that can provide the SaaS mashup service platform to be developed through the pilot service and transformed into a product that users expect. The value of the SaaS services that the SaaS mashup service platform generates is likely to exceed everyone's expectations.

Acknowledgement

This research was supported by the Startup Foundation for Introducing Talent of NUIST under Grant No.2019r032, The National Natural Science Foundation of China under grant Numbers 62102190, 62072250.

References

[1] J. Y. Kim, K. Ro, "A Study on The Standard Platform Model for CSB Business," *Indian Journal of Public Health Research and Development*, vol. 9, no. 8, pp. 681-686, 2018.

[2] S. Venkateswaran, S. Sarkar, "Modeling Operational Fairness of Hybrid Cloud Brokerage," in *Proc. of 18th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, CCGRID 2018*, pp. 563-572, 2018. Article (CrossRef Link)

[3] S. Sundareswaran, A. C. Squicciarini, D. Lin, "A brokerage-based approach for cloud service selection," in *Proc. of 2012 IEEE 5th International Conference on Cloud Computing*, No. 6253551, pp. 558-565, 2012. Article (CrossRef Link)

[4] A. Elhabbash, F. Samreen, J. Hadley, "Cloud brokerage: A systematic survey," *ACM Computing Surveys*, vol. 51, no. 6, pp.1-28, 2019. Article (CrossRef Link)

[5] L. Sabatucci, S. Lopes, M. Cossentino, "Self-configuring Cloud Application Mashup with Goals and Capabilities," *Cluster Computing*, vol. 20, no. 3, pp. 2047-2063, 2017. Article (CrossRef Link)

[6] X. Zhou, C. Li, H. Zhang, F. Meng, D. Chu, "A Feature Tree and Dynamic QoS based Service Integration and Customization Model for Multi-tenant SaaS Application," in *Proc. of 2020 International Conference on Service Science (ICSS)*, pp. 107-114, 2020. Article (CrossRef Link)

[7] G. Kesidis, T. Konstantopoulos, M. A. Zazanis, “The distribution of age-of-information performance measures for message processing systems,” *Queueing Systems*, vol. 95, no. 3, pp. 203-250, 2020. Article (CrossRef Link)
[8] P. Bhimani, G. Panchal, “Message delivery guarantee and status update of clients based on IOT-AMQP,” *Intelligent Communication and Computational Technologies*, pp. 15-22, 2018. Article (CrossRef Link)
[9] M. H. Javed, X. Lu, D. K. Panda, “Cutting the tail: designing high performance message brokers to reduce tail latencies in stream processing,” in *Proc. of 2018 IEEE International Conference on Cluster Computing*, pp. 223-233, 2018. Article (CrossRef Link)
[10] C. Esposito, F. Palmieri, K. K. R. Choo, “Cloud Message Queueing and Notification: Challenges and Opportunities,” *IEEE Cloud Computing*, vol. 5, no. 2, pp. 11-16, 2018. Article (CrossRef Link)
[11] Y. Xue, S. Jin and X. Wang, "A Task Scheduling Strategy in Cloud Computing with Service Differentiation," *KSII Transactions on Internet and Information Systems*, vol. 12, no. 11, pp. 5269-5286, 2018. Article(CrossRefLink)
[12] I. Sadooghi, G. Kumar, K. Wang, D. F. Zhao, T. L. Li, I. Raicu, “Albatross: An efficient cloud-enabled task scheduling and execution framework using distributed message queues,” in *Proc. of IEEE 12th International Conference on e-Science*, pp. 11-20, 2016. Article (CrossRef Link)
[13] I. Sadooghi, K. Wang, D. Patel, D. F. Zhao, T. L. Li, S. Srivastava, I. Raicu, “FaBRiQ: Leveraging Distributed Hash Tables towards Distributed Publish-Subscribe Message Queues,” in *Proc. of 2015 IEEE/ACM 2nd International Symposium on Big Data Computing*, pp. 11-20, 2015. Article (CrossRef Link)
[14] M. J. Sax, S. Sakr, A. Zomaya, “Apache Kafka,” 2019. Article (CrossRef Link)
[15] J. W. Bang, S. W. Son, H. J. Kim, Y. S. Moon, M. J. Choi, “Design and implementation of a load shedding engine for solving starvation problems in Apache Kafka,” in *Proc. of 2018 IEEE/IFIP Network Operations and Management Symposium*, pp. 1-4, 2018. Article (CrossRef Link)
[16] H. Wu, Z. Shang, K. Wolter, “Learning to reliably deliver streaming data with apache kafka,” in *Proc. of 2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pp. 564-571, 2020. Article (CrossRef Link)
[17] C. N. Nguyen, J. S. Kim, S. W. Hwang, “KOHA: Building a Kafka-Based Distributed Queue System on the Fly in a Hadoop Cluster,” in *Proc. of 2016 IEEE 1st International Workshops on Foundations and Applications of Self* Systems*, pp. 48-53, 2016. Article (CrossRef Link)
[18] T. Badriyah, S. Azvy, W. Yuwono, I. Syarif, “Recommendation System for Property Search Using Content Based Filtering Method,” in *Proc. of 2018 International Conference on Information and Communications Technology*, 2018. Article (CrossRef Link)
[19] X. Y. Su, T. M. Khoshgoftaar, “A Survey of Collaborative Filtering Techniques,” *Advances in Artificial Intelligence*, vol. 2009, 2009. Article (CrossRef Link)
[20] Y. Yu, Y. Gu, H. Zuo, J. Wang, D. Wang, “Social recommendation algorithms with user feedback information,” *Concurrency and Computation Practice and Experience*, vol. 33, 2021. Article(CrossRefLink)
[21] E. Lee and J. Jang, "Research Trend Analysis for Sustainable QR code use - Focus on Big Data Analysis," *KSII Transactions on Internet and Information Systems*, vol. 15, no. 9, pp. 3221-3242, 2021. Article(CrossRefLink)
[22] L. Liu, W. Li, L. Wang and H. Jia, "PCRM: Increasing POI Recommendation Accuracy in Location-Based Social Networks," *KSII Transactions on Internet and Information Systems*, vol. 12, no. 11, pp. 5344-5356, 2018. Article(CrossRefLink)
[23] Z. Cui, X. Xu, X. U. E. Fei, X. Cai, Y. Cao, W. Zhang, J. Chen, “Personalized recommendation system based on collaborative filtering for IoT scenarios,” *IEEE Transactions on Services Computing*, vol. 13, no. 4, pp:685-695, 2020. Article (CrossRef Link)
[24] D. Kluver, M. D. Ekstrand, J. A. Konstan, “Rating-based collaborative filtering: algorithms and evaluation,” *Social Information Access*, pp. 344-390, 2018. Article (CrossRef Link)

[25] A. Soylu, F. Mödritscher, F. Wild, P. D. Causmaecker, P. Desmet, “Mashups by Orchestration and Widget-based Personal Environments: Key Challenges, Solution Strategies, and an Application,” *Program: Electronic Library and Information Systems*, vol. 46, no. 4, pp. 383–428, 2012. Article (CrossRef Link)
[26] B. Cheng, S. Zhao, J. Qian, Z. Zhai, J. Chen, “Lightweight service mashup middleware with REST style architecture for IoT applications,” *IEEE Transactions on Network and Service Management*, vol. 15, no. 3, pp. 1063-1075, 2018. Article (CrossRef Link)
[27] P. A. Bernstein, L. M. Haas, “Information integration in the enterprise,” *Communication ACM*, vol. 51, no. 9, pp. 72-79, 2008. Article (CrossRef Link)
[28] Y. Lei, Y. C. Duan, K. C. Li, “A real-world service mashup platform based on data integration, information synthesis, and knowledge fusion,” *Connection Science*, vol. 33, no. 3, pp. 463-481, 2021. Article (CrossRef Link)
[29] H. S. Seok, Y. J. Lee, “Ontology-based IoT context information modeling and semantic-based IoT mashup services implementation,” *The Journal of the Korea institute of electronic communication sciences*, vol. 14, no. 4, pp. 671-678, 2019. Article (CrossRef Link)
[30] W. Q. Lin, C. N. Wang, W. Wang, “Mashup-based Architecture for Social Trends Analysis System,” in *Proc. of 2019 IEEE 8th Joint International Information Technology and Artificial Intelligence Conference (ITAIC)*, 2019. Article (CrossRef Link)
[31] M. F. Huang, "A queuing delay utilization scheme for on-path service aggregation in services-oriented computing networks," *IEEE Access*, vol. 7, pp. 23816-23833, 2019. Article (CrossRef Link)
[32] N. Y. Cao, J. S. Kim, J. H. Lee, S. W. Hwang, “A Case Study of Leveraging High-Throughput Distributed Message Queue System for Many-Task Computing on Hadoop,” in *Proc. of 2017 IEEE 2nd International Workshops on Foundations and Applications of Self* Systems (FAS*W)*, pp. 257-262, 2017. Article (CrossRef Link)
[33] N. Kulathuramaiyer, “Mashups: Emerging application development paradigm for a digital journal,” *Journal of Universal Computer Science*, vol. 13, no. 4, pp. 531–542, 2007.
[34] J. Y. Byun, Y. K. Kim, A. Y. Son, E. N. Huh, J. H. Hyun, “A real-time message delivery method of publish/subscribe model in distributed cloud environment,” in *Proc. of 2017 IEEE International Conference on Cybernetics and Computational Intelligence*, pp. 102-107, 2017. Article (CrossRef Link)
[35] J. Kreps, N. Narkhede, J. Rao, “Kafka: a Distributed Messaging System for Log Processing,” in *Proc.of 6th International Workshop on Networking Meets Databases*, 2011.
[36] P. Dobbelaere, K. S. Esmaili, “Industry Paper: Kafka versus RabbitMQ,” in *Proc. of 11th ACM International Conference on Distributed and Event-based Systems*, pp. 227-238, 2017. Article (CrossRef Link)

Zhiguo Chen received the M.S. and Ph.D. degree from the division of Internet and Multimedia Engineering at Konkuk University, Seoul, Korea, in 2014 and 2019, respectively. He is an Associate Professor with the School of Computer Science, Nanjing University of Information Science & Technology, Jiangsu, China. His research interests include artificial intelligence, information security and cloud computing, etc.

Myoungjin Kim received the M.S. and Ph.D. degree from the division of Internet and Multimedia Engineering at Konkuk University, Seoul, Korea, in 2009 and 2019, respectively. He is CEO with Innogrid, Seoul, South Korea. His research interests include cloud computing, big data, high performance computing, etc.

Yun Cui received the M.S. and Ph.D. degree from the division of Internet and Multimedia Engineering at Konkuk University, Seoul, Korea, in 2010 and 2019, respectively. He is an Associate Professor with the School of Computer Science, Jiangsu University of Science and Technology, Jiangsu, China. His research interests include cloud computing, big data, information security, etc.