J Grid Computing (2020) 18:441–469
https://doi.org/10.1007/s10723-020-09519-x

Approach for Selecting and Integrating Cloud Services to Construct Hybrid Cloud

Joonseok Park · Ungsoo Kim · Donggyu Yun · Keunhyuk Yeom

Received: 28 September 2018 / Accepted: 26 April 2020 / Published online: 13 May 2020

Abstract With the popularization of cloud computing, various cloud services have emerged, and hybrid clouds that can take advantage of combining public and private clouds are attracting attention. However, because of their variety, determining a combination of cloud services suited to the user's current environment and requirements is expensive when deploying a hybrid cloud. Even if the required services are available, there is a lack of tools to connect them, manage them in batches, and utilize the integrated environment. To solve these problems, this paper proposes a cloud selection and integration process (C-SIP), which selects and integrates a combination of cloud services through a hybrid cloud service broker (hybrid CSB), which is an automation solution supporting hybrid cloud deployment. Moreover, the proposed method is realized using a script including the application programming interface of each cloud service. The proposed C-SIP will be used as a core approach toward the hybrid CSB, which is expected to facilitate the introduction of hybrid clouds and the acquisition of cloud strategies.

Keywords Cloud service brokerage · Cloud service selection · Cloud service integration · Cloud service evaluation · Hybrid cloud

Ungsoo Kim and Donggyu Yun contributed equally to this work.

J. Park
Research Institute of Intelligent Logistics Big Data, Pusan National University, Busandaehak-ro 63beon-gil, Geumjeong-gu, Busan 46241, South Korea
e-mail: pjs50@pusan.ac.kr

U. Kim
SK Holdings Co Ltd, 9 Seongnam-daero 343beon-gil, Bundang-gu, Seongnam-si, Gyeonggi-do 13558, South Korea
e-mail: kus9010@sk.com

K. Yeom (✉)
School of Computer Science and Engineering, Pusan National University, Busandaehak-ro 63beon-gil, Geumjeong-gu, Busan 46241, South Korea
e-mail: yeom@pusan.ac.kr

D. Yun
Department of Electrical and Computer Engineering, Pusan National University, Busandaehak-ro 63beon-gil, Geumjeong-gu, Busan 46241, South Korea
e-mail: lodestar692@pusan.ac.kr

1 Introduction

Cloud computing is a computing paradigm that allows remote access via Internet without the need to directly install information technology resources such as servers, platforms, and software [1]. Cloud computing can be classified regarding the service provision method into public and private clouds. According to the latest edition of Flexera's "Rightscale 2019 state of the Cloud Report," [2] 84% of companies have a multi-cloud strategy, and 58% have a hybrid cloud strategy. Marketwatch [3] states that the global hybrid market is expected to grow at an average annual rate of 22.8% by 2025, reaching USD 140.86 billion by 2025.

A hybrid cloud is a cloud deployment model that combines a public and a private cloud, or a public cloud with an existing on-premise environment [4]. With a

hybrid cloud, both cost-effectiveness and security can be achieved by storing security-sensitive data such as personal information or company confidential documents in a private cloud or on-premise environment and storing other data in a public cloud. Additionally, there are different forms of utilizing the private cloud, such as switching to a public cloud backup system in the event of a disaster. With the advent of the fourth industrial revolution, cloud computing is valuable as a foundational technology and technical studies are required to promote the adoption of hybrid cloud.

There are various problems involved in introducing a hybrid cloud and securing an optimized cloud strategy. The current state of private infrastructure in the enterprise and the characteristics of public cloud should be understood. Furthermore, the cloud architecture should be constructed by appropriately combining private and public clouds. In addition, each public cloud environment that constitutes the cloud architecture must be connected to the private cloud environment and used as if it was a single cloud environment. There are many issues to be resolved, such as consolidation of different clouds to combine complex configurations, metadata movement between clouds, migration complexity problems related to workload distribution and execution depending on the nature and type of the workload, and security problems [5–7].

However, not all companies or organizations currently have cloud experts, and there are inadequate solutions to solve these problems and help them transition to a hybrid cloud environment. Although some public cloud companies such as Amazon, Microsoft, and Google provide consulting services on building a cloud environment, there is a limit to passive consulting, which depends on the experience of cloud experts, and the hybrid cloud that is built in this manner is limited to those provided by the vendors. In other words, problems that depend on the vendor should be solved when building a hybrid cloud.

To solve these problems, we proposed the concept of a hybrid cloud service broker (CSB) [8, 9]. A CSB is an intermediary between cloud users and providers, which selects or serves and manages cloud services to users [1]. Traditional CSBs focus on recommending a single cloud service and do not employ an approach for recommending a combination of individual vendors' cloud service configurations, even if recommending multiple cloud services. To realize an optimal hybrid cloud environment, it is necessary to utilize various cloud services from different vendors.

The hybrid CSB proposed in this paper extends previous CSB research [10]. It selects a combination of cloud services according to the user's requirements and supports the establishment of a hybrid cloud environment through a combination of selected services [8, 9]. The proposed method differs from other approaches to construct hybrid cloud environments, as it is used to select an architecture structure, which is difficult for hybrid clouds, and enhances the convenience of building a customized hybrid cloud environment considering the purpose and cost. Therefore, in this paper, we propose a cloud selection and integration method as an important technology of the hybrid CSB for building a hybrid cloud environment.

Cloud vendors such as Amazon, Microsoft, and IBM provide the types (e.g., "designs" or "architectures") used to build the cloud which are called "reference architectures" [11–13] or "cloud design patterns" (CDPs) [14, 15]. However, providing an automated solution that supports hybrid cloud deployment by automating the types of cloud services offered by vendors without the involvement of vendor experts in hybrid cloud deployments is a new approach. In this paper, we propose a cloud service selection approach, in which a combination of various clouds is selected according to the user's requirements in the core function of the hybrid CSB, and a script generation approach is used to solve the problem of complexity between clouds.

The cloud selection approach presented in this paper provides a cloud service combination type expressed in the form of a pattern by abstracting the cloud coupling type according to the purpose and requirements of building a hybrid cloud environment. Additionally, by matching the services of different cloud vendors to patterns, it allows users to combine environments to create a suitable hybrid cloud environment. Moreover, there is a need for an approach that realizes the capabilities of a hybrid cloud environment based on the recommended combination. Thus, we present a script generation approach for creating an integration script to implement the functions performed in a hybrid cloud environment. The script generation approach that supports cloud-to-cloud coupling allows the control of different cloud services in a hybrid cloud environment. Additionally, it allows users who can only implement one service to implement the desired function of the hybrid cloud through an application programming interface (API) combination of various services.

The proposed approach can be applied as a base technology and template of the hybrid CSB, which forms a

hybrid cloud environment by combining various individual services of many different vendors. By using the selection and script generation approach of the hybrid CSB proposed in this paper, users can build their own hybrid cloud environment, which is expected to facilitate the adoption of hybrid clouds and the acquisition of cloud strategies.

The major contributions of this study are as follows.

- In this paper, we proposed a process and technique to determine the cloud combining structure when implementing hybrid cloud and to achieve the cloud-to-cloud integration according to the combining structure. This can be applied as a foundational model that can realize automation of hybrid cloud introduction;
- We proposed a pattern-based service selection technique that recommends a combination of public cloud services that meets the cloud construction objectives when introducing hybrid cloud using hybrid CSB;
- Existing service selection techniques research [16, 17] mostly select a single cloud service, but the method proposed can recommend a combination of services that are selected considering the hybrid cloud environment. The user can also configure cloud service combinations to suit user requirements by reflecting different selection criteria, such as service scores, single vendor, and multi-vendor type selection;
- We proposed a script generation method that can perform the integration between clouds by applying the design pattern. In other words, it is an executable script that implements the integration function between clouds;
- Existing integration technique research [18–20] is proposed as a method to support or control the integration between clouds by using defined APIs, methods, libraries, etc.; it cannot create new functions. In contrast, the integration technique proposed in this paper can flexibly create new integration functions desired by the user by defining methods, script connections and integration relationships.

2 Related Work

2.1 Related Research on Cloud Service Selection and Cloud Combination Pattern

Garg et al. [16] proposed an "SMICloud" framework for selecting cloud services. The architecture of the SMICloud framework consists of three parts: SMI cloud broker, monitoring, and service catalog. It has the form of a basic CSB. The service selection method proposed in this study employs the analytic hierarchy process (AHP) [21]. This is one of the representative multi-criteria decision-making (MCDM) algorithms [22, 23]. It uses evaluation criteria that are layered in several levels. In this study, the quality of service (QoS) attributes were evaluated using the service attribute index (SMI) [24]. The SMI is a standardized method for measuring and comparing the QoS of cloud services. It is defined by high-quality attributes such as accountability, agility, cost, performance, assurance, security and privacy, and usability. According to the calculation method of the AHP, it is possible to obtain the score of the upper quality attribute by measuring each of the measurable lower quality attributes and calculating the scores reflecting the importance of each lower quality attribute. Similarly, the score obtained for the upper quality attribute reflects its importance, and if it is repeatedly raised to the highest level, the final score of the service can be obtained. This method can evaluate the service by reflecting different requirements of each user. However, the intuitiveness of the pairwise comparison matrix [25] to be input is low when the user inputs a value without sufficient understanding of the pairwise comparison matrix.

Zheng et al. [17] proposed a method of selecting a cloud service using a collaborative filtering algorithm [26] based on the QoS of the cloud service. The Pearson [27] and the Spearman correlation coefficients [28] are calculated according to the QoS value measured by each user. The similarity between users is calculated using the correlation coefficient to find a user similar to the new user and select a cloud service for the new user. However, the collaborative filtering algorithms cannot be used until a sufficient amount of user data is obtained.

The aforementioned studies focused on the selection of a single cloud service; multi-cloud and hybrid cloud environments, which have been discussed recently, were not considered. Service selection using QoS is the most commonly used method in the service selection and cloud fields, utilizing the MCDM algorithm [22, 29, 30]. In addition, the cloud computing service selection comparative study presented by Sirohi [31] et al. classifies MCDM and optimization, trust [32], and incentive based service selection techniques as cloud service selection research types. In this study, MCDM was analyzed as the most widely used technique.

Martino et al. [33] proposed a semantic expression method for representing cloud patterns in machine-readable form and proposed an approach to support application development in a multi-cloud environment. In this study, they defined a set of cloud services as a cloud pattern and a set of cloud patterns as an application pattern and aimed to deal with these patterns using semantic technology. They classified cloud patterns into two types: vendor-dependent and agnostic. The vendor-dependent cloud pattern, as the name implies, is a cloud vendor-dependent pattern, such as a pattern composed of AWS or Azure services. In contrast, the agnostic cloud pattern is defined independently of the cloud vendor. Additionally, according to the defined semantic model, the semantic web rule language and SPARQL (SPARQL protocol and RDF query language) [34] are used to identify and search for patterns and services. Hence, it is possible to express and search the combination service for the hybrid cloud, but the point of selecting the service is not revealed.

Liu et al. [35] proposed a method employing a social learning optimization (SLO) algorithm to select a combination of cloud services. The SLO is an evolutionary algorithm [36] that imitates a human social learning process. After the fitness is calculated using the QoS of the candidate services, a combination of cloud services is selected using variation and intersection techniques such as evolutionary algorithms, and the learning and observation learning mechanisms added in the SLO are simulated. However, because only the QoS was considered as an attribute of the cloud service in this study, and the semantic part, such as the type and purpose of the cloud service, was not considered, it was difficult to combine different types of services considering the respective roles.

2.2 Cloud Service Integration

The cross-platform cloud API provides a higher level of abstraction than the cloud API provided by the cloud provider and allows unified API calls to leverage the resources of one or more cloud services. This reduces the complexity of the code within the script and reduces the development costs by eliminating the need to implement access to a variety of cloud services.

Apache Libcloud [37] is one of the representative cross-platform cloud APIs. It is a library of cloud services in various service categories and their APIs are written in Python. Developers can easily use Libcloud to develop products that support a variety of services. Libcloud acquires the driver of the corresponding cloud provider and performs the work of the corresponding cloud service, such as compute or storage, through the API. It offers service categories such as compute, block storage, object storage, CDN (content delivery network), load balancer, and DNS (domain name system), and other services are provided for each category. Additionally, various methods are supported for each service. In this study, we created an integration script with executable code through an integration script generation model built using Libcloud.

Apache jclouds [38] is a Java-based open-source API library that facilitates the development of a common API for making Apache's cloud services compatible with each other. Jclouds provides a framework for accessing each cloud service and provides a common set of APIs available through drivers that are provided for each service. Jclouds supports service categories such as compute, blobstore, and load balancer, and supports fewer service categories and cloud services than Libcloud.

Deltacloud [39] is an open-source project for the interoperability of public and private clouds. The most important goal is to integrate the public and the private clouds and manage them with the same interface. Deltacloud is available through a REST API and can be used with libraries written in various languages. However, the project was suspended on July 20, 2015.

Grozev et al. [18] proposed four types of inter-cloud and multi-cloud coupling schemes: centralized and peer-to-peer coupling structures, and coupling structures using a multi-cloud service and a multi-cloud library. In this study, a hybrid combination of the CSB of the hybrid cloud service adopts a combination structure using the multi-cloud library, which provides APIs of various cloud services as a library.

Markoska et al. [19] proposed a method for developing interoperable cloud services using software design patterns. Their method employs an adapter interface that can simplify the cloud service development process by integrating APIs of OpenStack [40] and Eucalyptus, which are heterogeneous clouds. The adapter class allows users to configure scripts on a function-by-function basis using OpenStack and Eucalyptus cloud services. However, this study considered only these two cloud services and only a few compute-related functions.

Meireles [20] presented a cloud resource integration management architecture using Deltacloud, which is

one of the multi-cloud libraries. Users can invoke the architecture in a REST fashion or through an interface and use the OpenNebula, OpenStack, CloudStack, and LunaCloud APIs within the architecture through Deltacloud. However, Deltacloud has been a deprecated project since 2015, and the architecture presented in that paper supports only private clouds such as OpenStack.

Silva et al. [41] proposed a service delivery cloud platform in which applications can interoperate with different cloud services through a common API and a standardized interface. The proposed platform separates APIs of various cloud services into a unique abstraction and a normalized interface for the supported cloud service. However, this platform cannot support all of the various unique APIs of the cloud service, because it provides a common abstraction of the API of the cloud service.

Related works [18–20, 41] have not provided a method for creating new functions by combining methods or scripts. They suggest techniques for controlling various cloud services through a method that is commonly abstracted.

2.3 Multi-Cloud Supporting Tools

AWS offers VMWare Cloud on AWS, which can be scaled to seamlessly migrate a specific cloud environment, i.e., a VMware VSphere-based environment, to the AWS cloud. Additionally, the AWS Outpost service provides a hybrid cloud solution that virtually brings AWS operational models to on-premise facilities [42].

Microsoft Azure provides API Service, Logic Apps, a Service Bus [43], and an Event Grid as integration services. API Service allows APIs to be published and managed. This API service [43, 44] makes software services accessible to other software, and the services may run in the cloud or on-premise. The Service Bus allows secure communication between hybrid cloud solutions and allows a variety of apps to be accessed through a single connection. Logic Apps support orchestration of business processes and workflows. The Event Grid allows events to be raised and forwarded.

Google offers technology to support Anthos and hybrid connections [45–47]. Anthos is a platform for building and managing modern hybrid applications on existing on-premise devices or public clouds [46]. Connectivity products [47] that support hybrid connectivity include Cloud Interconnect, Cloud VPN (virtual private network), and peering. Cloud Interconnect provides enterprise-class connectivity to Google virtual private cloud (VPC). Cloud VPN allows an on-premise or other public cloud network to be securely connected to a GCP (Google Cloud platform) VPC via IPSec VPN over the Internet when a data bandwidth of 3.0 Gbps is needed. Google and Google Cloud features can be accessed through a peering VPN or the Internet.

As suggested in related works [42–47], there are tools for linking a public cloud with existing on-premise systems. However, to understand these technologies, users must analyze the services of the vendor, and it is not easy to apply the services without the help of a cloud vendor expert. Additionally, it is difficult to construct a hybrid environment through a combination of various types of services provided by each vendor.

Terraform [48] is an open-source infrastructure-as-code software developed by HashiCorp. It allows developers to define and deploy data center infrastructure using the high-level configuration language called HashiCorp used in Terraform or JSON. Terraform focuses on the deployment of the infrastructure and only has a command-line interface [49]. In contrast, the integration script proposed in this paper can flexibly define the customized function by selecting the functions provided by the cloud service in the hybrid cloud environment composed of the recommended patterns. In creating a script, it is convenient to generate a script with an XML structure by choosing a method selected by the user using a graphical user interface and a connection relationship (e.g., sequence, fork and join) between methods. When a new cloud vendor is created and a corresponding new cloud service is created, it can be easily applied by additionally mapping service categories and new service and vendor information to reference patterns and CDPs managed as resources in the hybrid CSB. For generating the integration script, the integration script generation model with the design pattern is employed, as shown in Fig. 6. Therefore, when a new cloud vendor opens and provides a service API, as shown in Fig. 6, it has the scalability to create and support a vendor-specific ServiceDriver in the same structure as the currently supported Libcloud-based connected structure.

Caballer et al. [50] presented a mechanism called the INDIGO-DataCloud [51] Project to orchestrate computing resources across a heterogeneous cloud. In this study, they used the description language TOSCA, which is a standard designed specifically to model cloud-based application architectures [52, 53]. In

contrast, in our study, the model employed for cloud service selection is defined using the unified modeling language [54–56]. The pattern of the defined model is represented in the form of nodes and edges, and the internal structure is transformed and managed in XML. In the case of using TOSCA, tools such as TOSCA Parser, Heat Translator, etc. to interpret and process TOSCA should be linked and employed [57]. Our proposed method creates a combined script that defines the function by connecting the necessary methods to create the supported hybrid cloud function after the hybrid cloud combining structure is selected. Additionally, the integration script generates Python code that is directly mapped to the component-defined APIs of each existing cloud vendor service.

Kovács et al. [49] presented Occopus, which is an open-source cloud orchestration and management framework for heterogeneous multi-cloud platforms. This framework uses two types of design patterns—abstract factory and strategy—to implement the proposed architecture model. Additionally, it defines the description language to specify a virtual infrastructure. We also used a design pattern to generate integration scripts. However, Occopus does not consider the combination of methods or the selection of abstract patterns according to the user's business goal.

3 Selection and Integration Process in Hybrid CSB

3.1 Hybrid CSB

The CSB is a concept or service that is responsible for brokering cloud service selections, contracts, and usage between cloud service providers and users [1]. However, existing CSBs are mainly focused on brokering public clouds. We previously proposed Virtual Cloud Bank (VCB), which is an architecture model for a CSB [10]. Many companies want to adopt a hybrid cloud; however, considering the current situation of hybrid cloud deployment and the complexity of the hybrid cloud environment, the CSB must support the establishment and operation of a hybrid cloud environment. We refer to this as a hybrid CSB, which is an expansion of the concept of our proposed VCB [10]. In our previous study [8], we presented the requirements and conceptual architecture of this hybrid CSB, which are shown in Fig. 1. The functional requirements for supporting hybrid cloud environments are classified into intermediation, management, and optimization.

Intermediation is performed in the service intermediation layer, which includes the ability to recommend public clouds, combine private and public clouds, and support migration and scheduling in a combined environment. Management is required to operate a hybrid CSB. It includes a function to manage resources, contracts, and policies, which are components of a hybrid CSB, and is a layer of operation support. Optimization is part of the evolution management layer, with features to improve the quality of hybrid cloud environments built with hybrid CSBs. It detects errors or performance issues in the cloud environment through monitoring and performs service evaluation and architecture improvement through feedback.

In this study, among the functions of the hybrid CSB, we focused on service selection and integration. Therefore, among the various modules shown in Fig. 1, the service selection and service integration modules are described in detail. The service selection module recommends the appropriate integration pattern and public cloud service considering the private cloud environment and requirements. The service integration module integrates multiple cloud services constituting the hybrid cloud environment to be built after the contract is concluded through an integration script.

Section 3.2 describes the service selection and service integration modules. In Sections 4 and 5, we describe the methods used for service selection and integration within the modules and present experimental results.

3.2 Selection and Integration Processes

In this section, we first present the interaction and overall process of each module of the hybrid CSB used for service selection and integration. If a user accesses a hybrid CSB, receives a service selection, which is a basic scenario from combining selected services to building a hybrid cloud environment; the functions of the modules shown in Fig. 2 are used.

The operation of each module is as follows.

- Registration: The user enters the requirements through the registration module to select and merge the cloud services.
- Resource management: The input requirements from the registration module are stored, along with

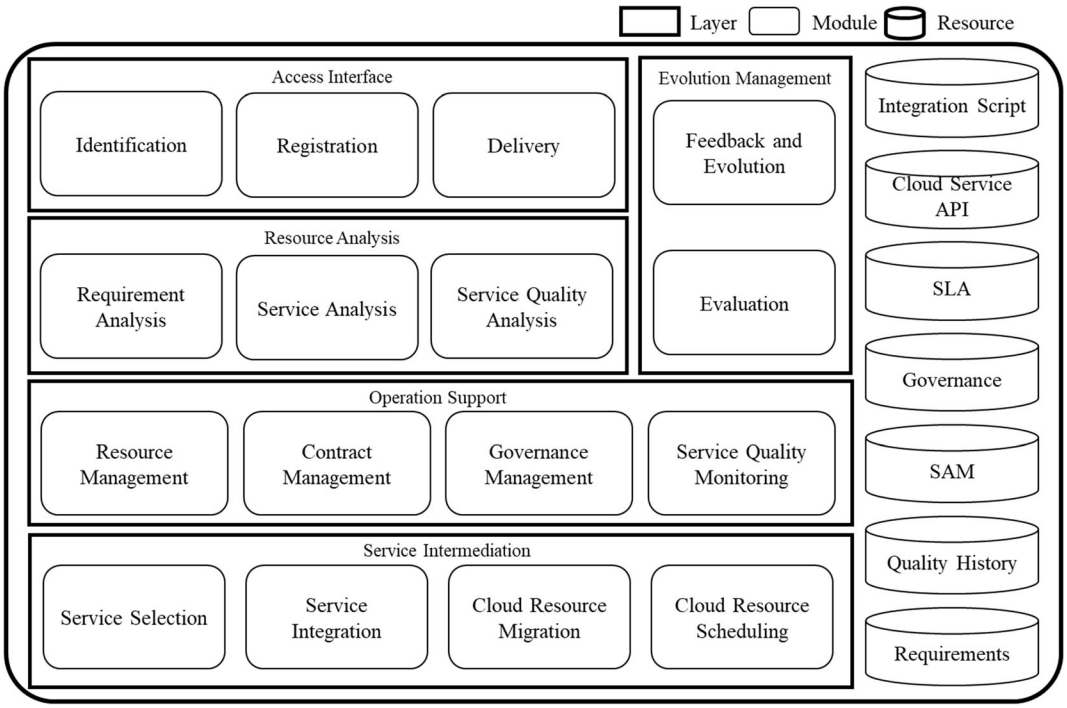


Fig. 1 Conceptual architecture of the hybrid csb that reflects requirements

the specification of the service received from the cloud service provider. Additionally, the requirements and service information, such as evaluations, requirement analysis, and service analysis, are delivered to other modules.

- Requirement analysis: The user requirements are analyzed and reworked into a form that can be used for service selection and combination.
- Service analysis: The specifications of the service are analyzed, the service selection is scored according to analyzed service specification, and the information necessary for the combination is reprocessed.
- Service selection: Information regarding requirements is received from the requirement analysis module, and the service description is received from the service analysis module. The selection result is output through the service selection mechanism.
- Service integration: The combination service selection result is received, and the integration script is generated using APIs of the corresponding services.

The detailed operation process of the service selection and integration modules is shown in Fig. 3. A brief description of each step in the process is presented below, and additional details are provided in Sections 4.

3.2.1 Patten-Based Cloud Service Selection

- Select pattern: Selects a pattern for a service combination based on the purpose for which the user wants to build a cloud environment. Details are presented in Section 4.1.1.
- Modify pattern: Adds or deletes the components of a pattern according to functional requirements entered by the user. Details are presented in Section 4.1.2.
- Bind existing service: Adds the cloud service that the user already has to the pattern. Details are presented in Section 4.1.3.
- Select service: Completes the service combination by selecting and adding a new public cloud service to the pattern. Details are presented in Section 4.1.4.

3.2.2 Cloud Service Integration

- Decide script elements: Determine the components of the integration script, including services, methods, and parameters to be controlled by the integration script. Details are presented in Section 4.2.1.
- Generate integration script: Generates an integration script that can perform the functions of a user's

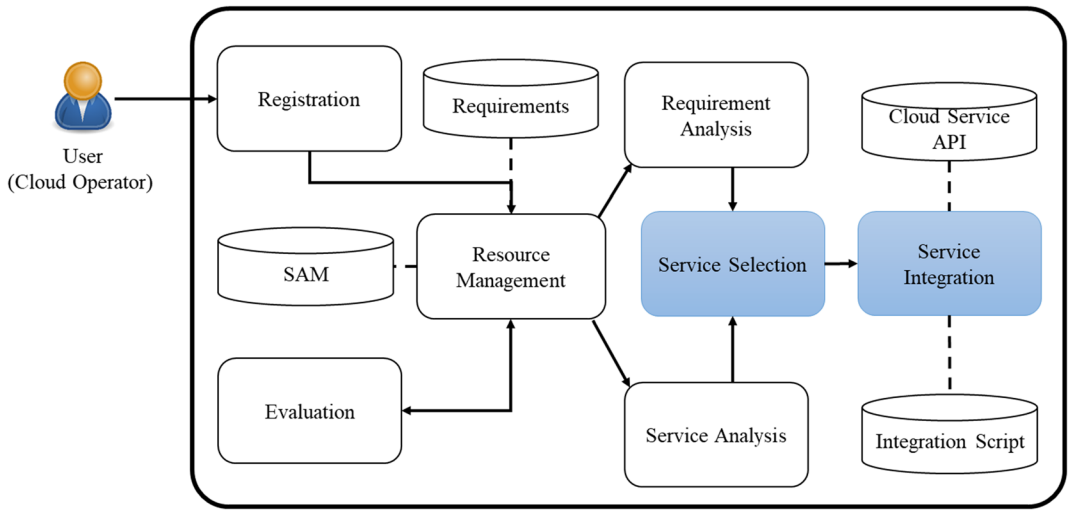


Fig. 2 Module interaction and related resources for usage scenario of hybrid CSB

desired hybrid cloud. Details are presented in Section 4.2.2.

3.3 Basic Concept and Model for Pattern-Based Cloud Service Selection

We constructed the model shown in Fig. 4 by summarizing the elements that should be defined for selecting the pattern-based cloud service. The model was divided into two parts: one that specifies requirements and other that specifies patterns. The details are as follows.

3.3.1 Requirements

- Business requirement: This section covers the user's requirements for building a cloud environment in which the hybrid CSB receives input. It includes information regarding what type of system is required to be built through the cloud (target system) and the maximum budget for building the cloud environment (maximum budget). Additional quality constraints and functional requirements may also be included. Furthermore, QoS priorities to be applied in service selection can be input, which are converted into a weight value in the service selection algorithm.
- Quality constraints: Users can enter quality constraints to be applied when a service is selected. This is the minimum condition that should be satisfied for quality. For example, there may be a requirement that an availability of ≥99.9% is required for a particular type of service, and this is reflected in the service selection phase.
- Functional requirements: Users can enter requirements for functional characteristics of the cloud environment. To satisfy these requirements, it is often necessary to use additional cloud services. For example, to add the ability to dynamically increase or decrease the number of servers according to the amount of traffic, an autoscaling service [58] should be used. When a user enters a functional requirement, a corresponding CDP containing a solution to satisfy the corresponding requirement is found and applied.

3.3.2 Pattern

- Integration pattern: We define the concept of a combining pattern (hereinafter referred to as "pattern") to represent a combination service. A pattern is a framework required for a service combination, consisting of a node and an edge. A node is an abstraction of a cloud service and contains information about a service type, but not a specific service. A node may be bound to an actual cloud service of the same type as the service about which it contains information. An edge represents a connection relationship between nodes, through which interactions can be made. A pattern comprises one or more nodes connected, and the binding of actual services to each

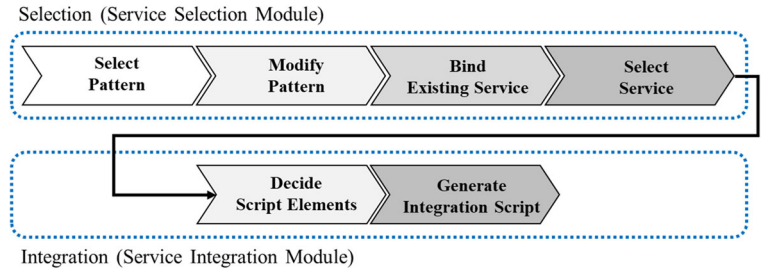


Fig. 3 Process for service selection and integration

node constituting the pattern is a combination service for a hybrid cloud, as shown in Fig. 5. Additionally, when each vendor's cloud service corresponding to the recommended node type is bound, information such as the average cost of using the individual cloud service or the measurement cost of moving data can be specified in the additional information attribute part of the class called "Integration Pattern," as shown in Fig. 4.

- Reference pattern: The reference pattern (or reference architecture) defines a set of commonly used services depending on the type of cloud system to be built. AWS, Azure, and IBM provide such reference patterns [11–13]. However, the reference patterns that they provide are vendor-dependent and use their services. In contrast, the technique presented in this study is for a hybrid cloud environment and is multi-vendor. Therefore, each service constituting the reference pattern is abstracted and stored in the node.
- CDPs: Summarize the solutions that are available for common issues that often arise when building a cloud environment. Here, the solution is generally presented as a combination of services similar to a reference pattern or as additional settings for a particular service. There are vendor-dependent CDPs, such as AWS and Azure [14, 15], and generic CDPs that are independent of vendors, such as those reported in [59, 60]. As in the case of the reference pattern, the vendor-dependent CDP is abstracted and stored as a combination of nodes.
- Master/slave service: There is a service dependency that should be considered when combining cloud services. Although there is no problem in connecting at the service-type level, there may be cases where the actual service is not technically connected if it is a service of a different vendor when the service is bound. For example, AWS's autoscaling service is designed for EC2 [61], which is the compute service of AWS, and cannot be combined with compute services of other vendors. Therefore, when binding a service to a pattern, this

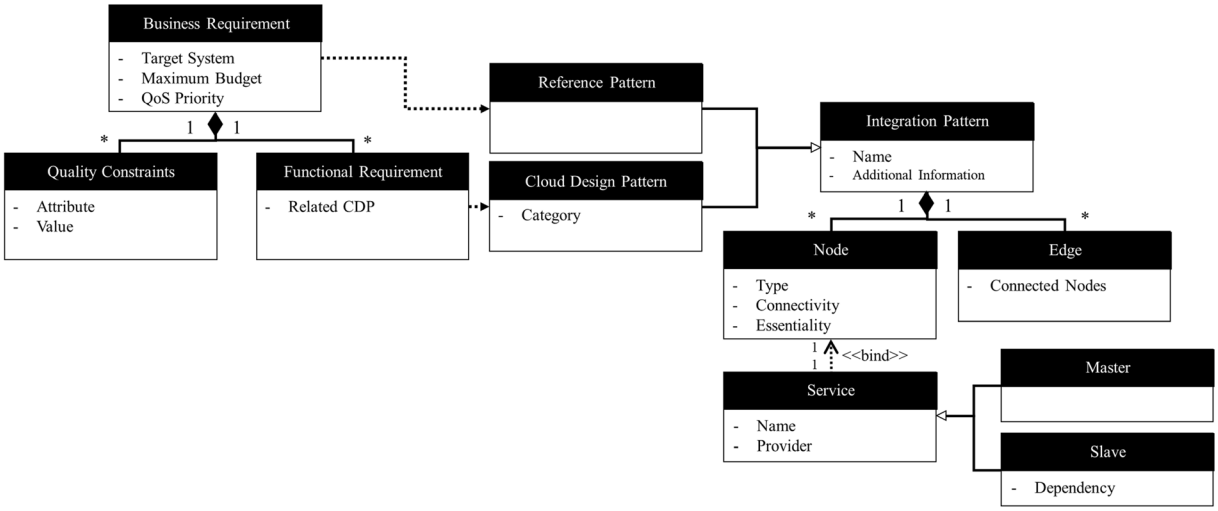


Fig. 4 Model for pattern-based cloud service selection using uml(unified modeling language)

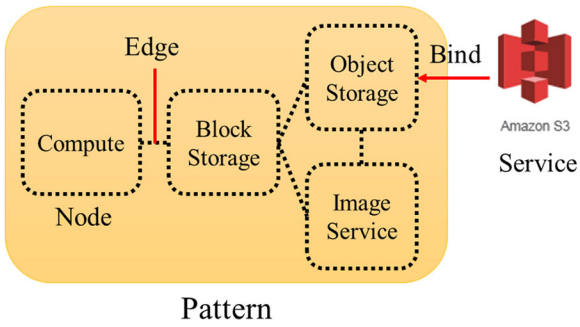


Fig. 5 Structure and representation example of a pattern

service dependency should be considered. To deal with this, the service is classified as a "master service" or a "slave service." A master service can be independently selected because it can perform functions or roles independently, and a slave service must be used with other specific services.

3.4 Basic Concept and Model for Cloud Service Integration

3.4.1 Definition of Integration Script and Template

In a hybrid CSB, cloud combining is performed when the services that compose the hybrid cloud and their relationship are determined through the selection function. In a hybrid CSB, cloud binding is script-based. An integration script implements the functionality provided by a user's hybrid cloud built with a hybrid CSB. Cloud combining generates integration scripts that satisfy the user's requirements for a hybrid cloud service. Figure 6 shows the components of an integration script and the templates. As shown in Fig. 6, the structure of the integration script is composed of method, parameter, service endpoint, and authentication information.

"Method" is a function desired by the user of the hybrid cloud service, and the integration script can have several methods. Method basically supports the Libcloud method, and the methods supported by different cloud services may differ. "Parameter" refers to the data instance to be used in the method. "Service Endpoint" is the endpoint of each cloud service, and the authentication information is the API key, token, etc. required to access the cloud service. The Service Endpoint and Authentication Information can be acquired through user input or through a method.

As shown on the right side of Fig. 6, the integration script has representation templates called Method, Script, and Script_Topology (Topology). An integration script can have two or more methods in a single script and can consist of a topology composed of multiple scripts.

The Method template in Fig. 6 shows the function of each cloud service in the hybrid CSB. Here, "id" and "name" are the attributes for managing the method in the hybrid CSB, and "service_name" indicates the function of the service. The method consists of two types, as shown in Fig. 6. Type 1 is the type of result of the method that affects the service or service resources, such as "create," "delete," "update," and "upload" parameters. Type 2 obtains the result of a method through a script, such as "read," "obtain," or "download" parameters.

The method type eases the selection of a method by allowing the user to select the method before the script element determination process. The input and output of a method refer to its input and output parameters and may have more than two items. A precondition is a condition that a method assumes to be performed normally, and a postcondition is an attribute described in order to grasp the result of a method execution. Two or more situations can be defined in the precondition and postcondition.

Figure 7 shows examples of templates, in accordance with Fig. 6. The left side of Fig. 7 shows the object_download() method template of S3, which is the object storage service of AWS. The hybrid CSB stores Method, Script, and Script_Topology in XML form. As shown in Fig. 7, the script template consists of one or more methods. It has its own id and name, and it refers to numerous methods. It also has a method composition structure. The upper right side of Fig. 7 shows a script template composed of the object_download() method of S3. Script templates are stored as XML-like resources and then converted into executable integration scripts. In the hybrid CSB, the topology consisting of a combination of scripts has attributes of the Script_Topology template. Script has a script composition structure. The lower right of Fig. 7 shows a move_object() topology template that combines the object_download() script in S3 and the object_upload() script in Swift.

Thus, a script and a topology can be generated by a combination of methods and scripts, respectively. The composition structures of the script and topology (each

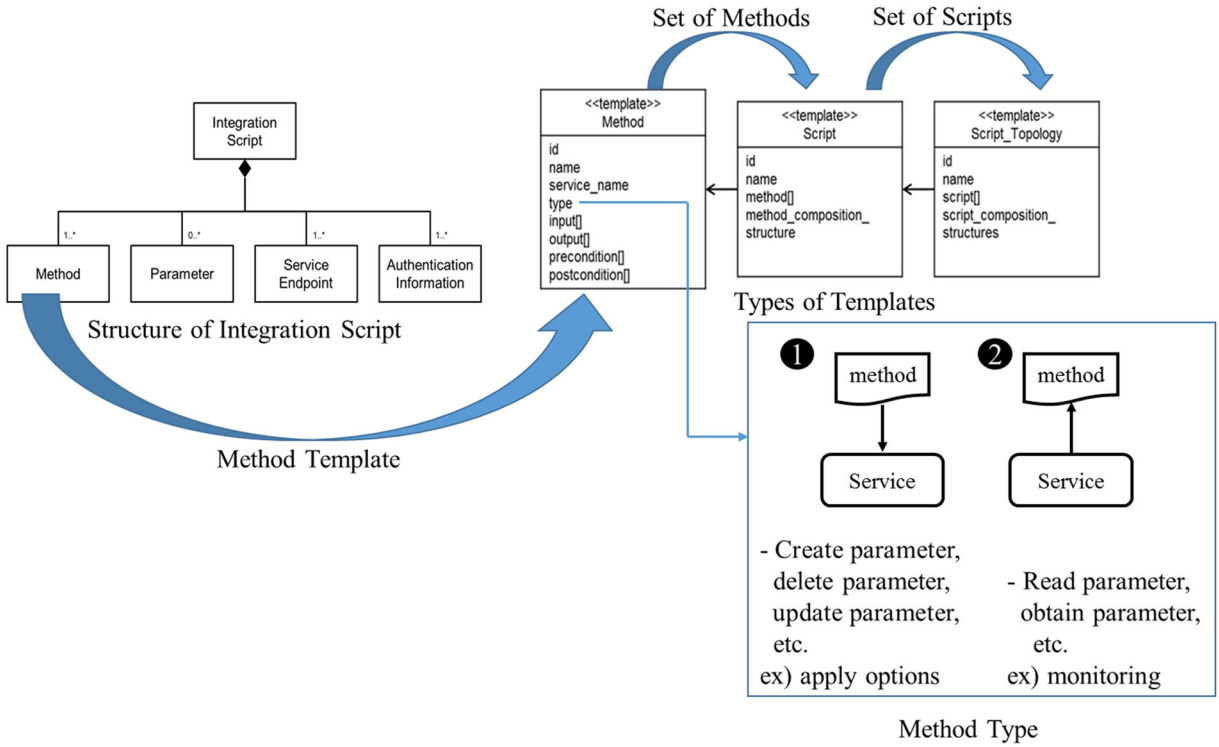


Fig. 6 Structure of integration script, templates, and method type

denoted as method_composition_structure and script_composition_structure, as shown in Fig. 6) are explained in sequence. Script and topology templates are converted into executable Python scripts to realize real-world functionalities.

3.4.2 Integration Script Configuration Structure

As shown in Fig. 8, there are four types of configuration structures that can be combined when a method or script is combined in an integrated script combination structure: sequence, loop, choice and merge, and fork and join. Sequence is a structure in which a method or a script is sequentially executed, as shown in Fig. 8.

In the sequence structure, a link that connects a method or a script is required. The sequence example shown in Fig. 8 is a function to download an object from S3, which is an object storage cloud service; move the object to Swift; and reboot the node of EC2, which is a computing service of AWS. When creating such a function, we connect each method through a link.

A loop is a structure in which a method or script within a loop range is repeatedly executed according to a condition, a period, and a count. The condition is the output of the method or script included in the loop. Among the condition, period, and frequency, at least one should be expressed in the loop structure. The loop in Fig. 8 shows an example of repeatedly executing the object list of the container with a period of 3 s and a count of 3 in OpenStack Swift, where the service category is object storage.

Choice and merge are structures that represent selective branches, and they have different paths depending on the conditions. The path is selected according to the condition through the choice, and it becomes one path through merge, as necessary. Choice and merge are expressed through a link and a condition, and the condition is the output of the preceding method in which the branch occurs. As shown in an example of choice and merge in Fig. 8, if attach_volume's output is true, the virtual machine node is deployed, and if it is false, the node is rebooted. Finally, we run list_nodes() to examine the state of the node.

Fork and join is an element that supports multiple operations of a method or a script. It separates each method or script operation through a fork and merges the result through a join after the separated operation is completed. The fork and join structure is represented by

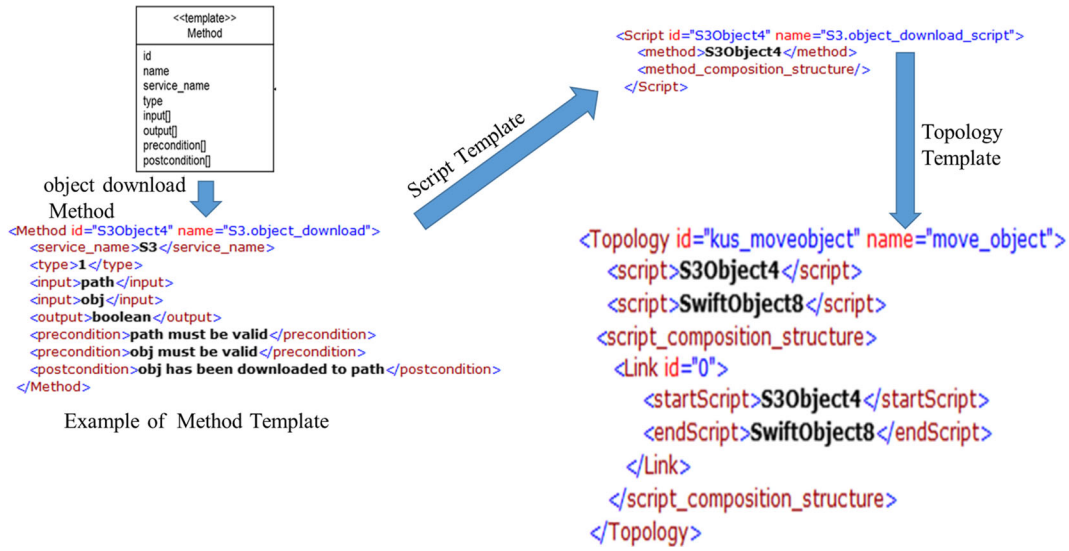


Fig. 7 Method, script, and topology template representation example

a link. When two or more start and end elements are present in a link, they represent a fork or join, as shown in an example of the fork and join structure in Fig. 8. In the compute service EC2, generating a node, creating a virtual machine, and generating a volume are performed (fork), and a method of attaching the created volume to the generated node is executed after all the operations are completed.

The integration script can implement new functions by combining various methods and scripts through sequence, loop, choice and merge, and fork and join.

3.4.3 Combination Structure of Integration Script

The integration script features a method unit combination structure in which numerous methods are combined into one script, as well as a script unit combination structure in which topologies are formed by combining multiple scripts.

In the method unit combination structure, an integration script is composed of a combination of methods. The left side of Fig. 9 shows an example of a structure that combines functions in a method unit. Here, OpenStack Swift and AWS S3, which are cloud object storage services, are used. The figure corresponds to a function to download object A from OpenStack Swift and move object A from OpenStack Swift to AWS S3 by uploading object A downloaded to AWS S3.

The function in Fig. 9 receives a token to access OpenStack Swift through OpenStack Swift's getToken() method and downloads object A from Swift using the token and download_object() method. Next, it obtains the API key of AWS S3 and uploads object A using the API key and upload_object() method. In the method unit combination structure, as shown in the figure, numerous methods are combined into one integration script.

The right side of Fig. 9 shows the structure of the script unit combination, where one integration script is

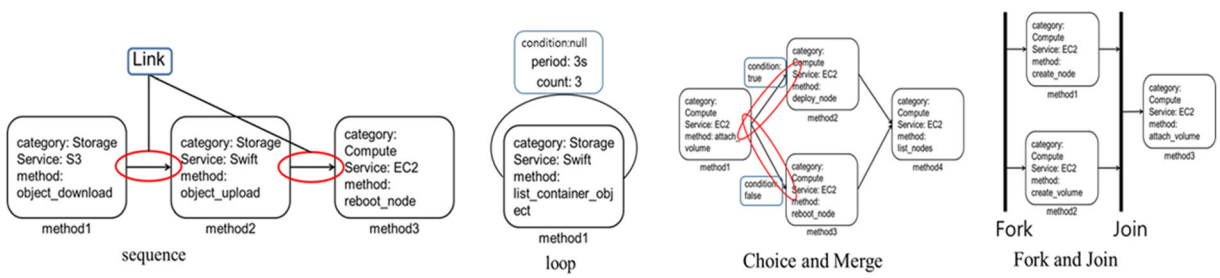


Fig. 8 Four types of script configuration structures

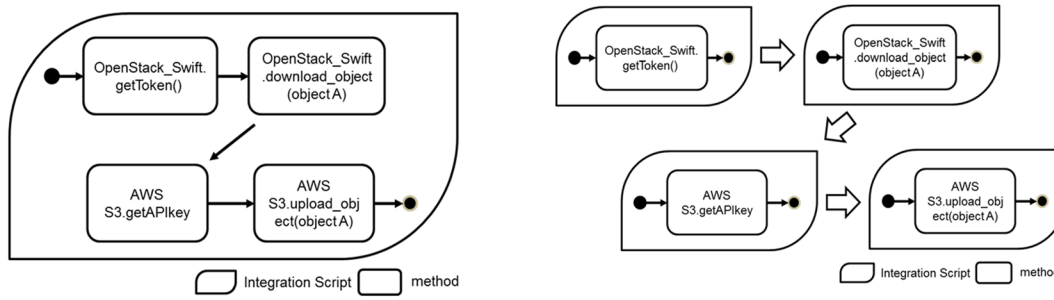


Fig. 9 Example of method unit and script unit combinations

composed of a combination of different integration scripts. The functions shown in the figure are the same as those described in the method unit combination structure.

The methods in the method unit combination structure are implemented as one integration script, and the actual function is implemented as a combination of scripts.

4 Pattern-Based Cloud Service Selection and Cloud Integration

4.1 Process and Method for Pattern-Based Cloud Service Selection

We propose a method and a process for selecting a combination of cloud services according to a pattern using the model presented in Section 3.3. The proposed process is shown in Fig. 10.

As shown in Fig. 10, the hybrid CSB receives the business requirements, as shown in Fig. 4, from a user who wants to receive a combination service through the registration module. The input business requirements consist of the target system, budget, QoS importance, functional requirements, and quality constraints, as described in the model in Section 3.3, which are processed through the requirement analysis module and used at different stages of the selection process. Additionally, the registration module provides an interface for entering and selecting business-requirement elements. In this study, for constructing the hybrid CSB based on the proposed selection process, the XML format is used to process data internally, but format can be extended to define and convert to the TOSCA format, JSON structure, etc.

The selection process consists of four steps: select pattern, modify pattern, bind existing service, and select new service. Details of each step are presented in Sections 4.1.1–4.1.4. When the selection process of step 4 is

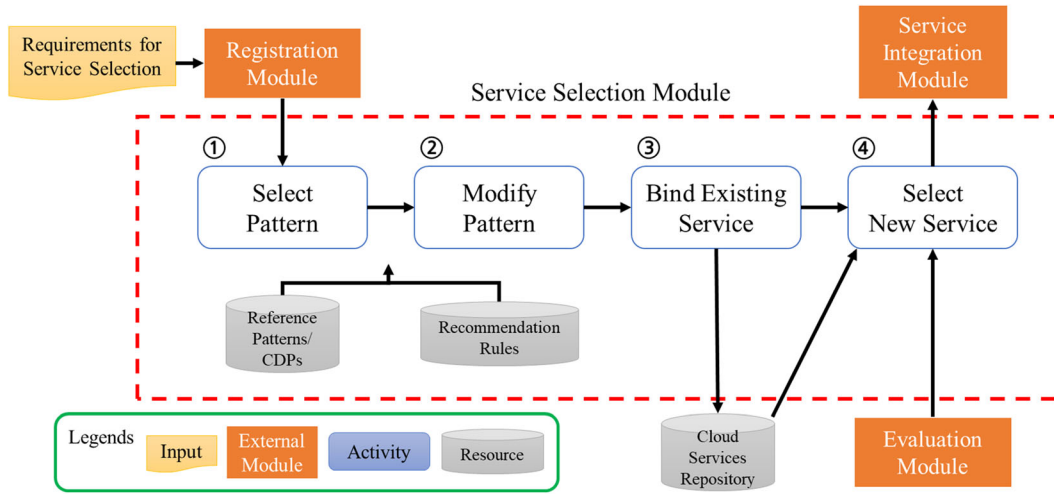


Fig. 10 Selection process and interaction in service selection module

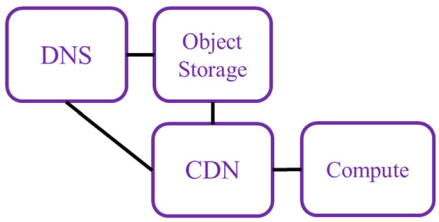


Fig. 11 Pattern selection example for content and media serving

completed and the combination service is selected, the service selection module transmits information regarding the combination service to the service integration module, which uses this information to create a connection script (integration script) to allow connections between the selected services.

4.1.1 Select Pattern in C-SIP

In the first part of the pattern selection phase, we refer to the reference pattern provided by cloud service vendors and select a pattern that matches the user's purpose. A reference architecture is a collection of commonly used services for building a cloud environment for a specific purpose. As shown in Fig. 10, the reference pattern and CDPs are internally stored as resources. Additionally, a recommendation rule that relates the user's purpose and suitable patterns is defined. In the pattern selection step, a user inputs the type of system to be constructed according to the purpose, that is, a cloud, and selects a pattern in which the reference architecture is abstracted. Pattern selection is a rule-based one-to-one correspondence method, and a rule is defined according to information provided by each cloud service vendor.

For example, if a user wants to deploy a cloud to build a service that distributes content or media, such as a video-streaming service, the user can select the "content and media serving" pattern among the AWS reference patterns [62]. Figure 11 shows an example. However, the AWS reference pattern is a combination of AWS services: Route 53, S3, CloudFront, and EC2. Table 1 presents an example of the mapping of AWS services to generalized service categories. Therefore, as shown in Fig. 11, a combination of nodes, i.e., a generalized pattern, is selected.

Table 1 Mapping example of AWS services and generalized service categories

Generalized service category	AWS service
DNS	Amazon Route 53
CDN	Amazon CloudFront
Compute	Amazon EC2
Object Storage	Amazon S3
Load Balancing	Elastic Load Balancing
Autoscaling	Autoscaling
RDB	Amazon RDS

Table 2 presents a mapping example for the same types of services provided by different cloud vendors: OpenStack, AWS, Google Cloud, and MS Azure. In the phase called "select new service in C-SIP," the service binding to the pattern is performed according to the selected vendor service using the information in Table 2, which is stored in reference patterns/CDP resources, as shown in Fig. 10.

4.1.2 Modify Pattern in C-SIP

In the second step, i.e., the pattern correction step, the patterns selected in the pattern selection step can be customized according to the user's additional requirements.

When modifying the pattern, we refer to the CDP. For each additional requirement that the user chooses, a CDP that can be employed to satisfy the requirement is used. When the user selects a requirement, the CDP corresponding to the pattern is compared with the pattern selected in the pattern selection step. If there is a node that exists in the CDP but does not exist in the selected pattern, this node is added and is connected to the combinable node.

The requirements satisfied by the components of the pattern that were selected in the pattern selection step are checked by default; the user can opt for deleting the service from the pattern to eliminate the check to satisfy the requirement. This is because the reference architecture may contain a mix of unwanted services, or the user may not want to use many services, owing to a lack of budget.

An example of pattern modification is presented as follows. The user decides to employ the AWS content and media serving pattern, as shown in Fig. 12(a). Users want cloud servers to be flexible with regard to traffic volume, it means, they want the number of virtual machines to be automatically adjusted when large and small amounts of traffic comes in. However, the combination of the currently selected nodes cannot be used to implement this function. A CDP that satisfies these

Table 2 Mapping example of vendor services and generalized service categories

Generalized service category	OpenStack	AWS	Google cloud	MS azure
Compute	Nova	EC2	Compute Engine	Virtual machine
Container	Magnum (optional)	ECS	Container Engine	Azure Container Service & Registry
Object Storage	Swift	S3	Storage	Azure storage
RDB	Trove (optional)	RDS	SQL	SQL Database
Monitoring	Horizon	CloudWatch	StackDriver	Azure Portal
Networking	Neutron	VPC	Networking (Cloud Virtual Network)	Virtual Network

requirements is the scale-out pattern, which is one of the AWS CDPs [14]. The AWS scale-out pattern consists of EC2, CloudWatch, Autoscaling, and AMI. Figure 12(b) shows the scale-out pattern. Here, nodes that are present in this pattern but not in the currently selected pattern are indicated by dotted lines.

4.1.3 Bind Existing Service in C-SIP

If a pattern is selected through the procedures in Sections 4.1.1 and 4.1.2, there remains a process of binding the actual service to each node constituting the pattern. Users may wish to introduce a new cloud without existing infrastructure, but they may want to build a hybrid cloud or a multi-cloud environment by combining their existing private or public clouds with new public cloud services.

In the existing service binding stage, information regarding the existing infrastructure is input to these users, and the corresponding service is bound to the appropriate node of the pattern. If there is no existing user infrastructure and a new service is selected at all nodes, this step is omitted. In the service binding step, information regarding the cloud service that is already held is input. The input information is presented in Table 3. This information is used to combine the services in the service integration module after the service selection is completed, as described in Section 4.2.

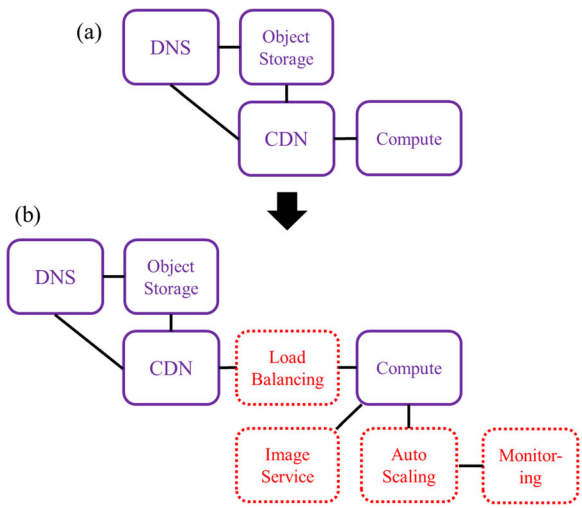


Fig. 12 Pattern modification example applying the scale-out pattern

4.1.4 Select New Service in C-SIP

In the service selection phase, which is the last step, the service selects a new cloud service to be bound to the remaining nodes to which the service was not bound in the service binding step, described in Section 4.1.3.

In the service selection method, the service score is calculated to reflect both the objective and subjective evaluation factors of the service. This process is performed in the evaluation module of the hybrid CSB module, as shown in Fig. 10, and the service selection module receives the final calculated service score.

The objective evaluation method is to score the QoS

Table 3 Required information for binding service

Information	Description
Service name	Name of service
Service type	Type of service (e.g., compute, storage, load balancing)
Endpoint	URL to access the service
Authentication information	Authentication information required to access the service. Depending on the service, the authentication information may have different names, e.g., "access key," "secret key," or "token," and there may be one or two pairs of keys.

Fig. 13 Sentiment-analysis example

User Rating	Review Text	Sentiment Score
5	Good product	+0.125
5	Excellent service for companies to provide excellent Customer Services	+0.480
5	Zendesk – powerful, elegant, and fun to use	+0.420

of the cloud service. Various types of QoS can be used as evaluation criteria. In this study, the QoS is scored using a simple additive weighting (SAW) MCDM algorithm [29].

Thus, the score and the weight of each criterion are calculated and added. However, the scale and distribution of the numerical values differ according to the type of QoS, and the QoS value cannot be directly used in SAW. Therefore, we need a method to standardize the score. In this study, we propose the standard and percentile scores.

The standard scores ease the expression of comparative advantages over other services. We use the T-score system with an average of 100 and a standard deviation of 20. Thus, if the availability of all services belonging to the same type of service A is 99.5%, the T-score of the availability of service A is 100 points. If the weight value for availability determined according to the requirements entered by the user is 0.25, 25 points (100 points multiplied by 0.25) is the availability score of service A.

However, when the standard score is used, it is necessary to assume that the evaluation value is a normal distribution. If the distribution of the QoS is far from the normal distribution, the discrimination power of the standard score may deteriorate. In this case, instead of the standard score, we use the percentile score [63].

In summary, we propose a method to use the following scores in SAW according to QoS metric characteristics:

- If the distribution of QoS values is close to the normal distribution, we use the standard score;
- If the distribution of QoS values is far from the normal distribution, we use the percentile score;
- If the current distribution of values is unknown, we use the standard score + percentile score.

The subjective evaluation method reflects the user feedback. The score is calculated by summing the results of applying sentiment analysis to the rating and reviewing the text entered by the user for each service. The most common method to provide feedback is by entering a rating and a review (text). In the rating evaluation, the method of using a score ranging from 1 to 5 (or 1 to 10) is the most widely used in various evaluation systems, e.g., for shopping malls and movie review sites, and is the simplest way to reflect user feedback.

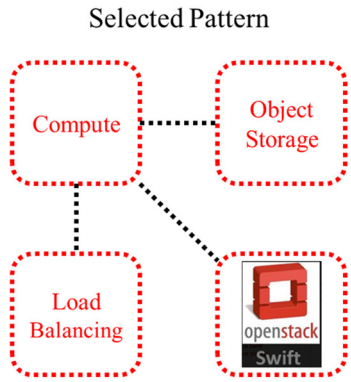
We propose a method of rating user feedback by applying sentiment analysis to review text with a rating. This allows us to identify the well-graded and weighted reviews to some extent, and we can increase the precision if we give precise corrections using the sentiment score/polarity score. Figure 13 shows an example of a cloud service review and sentiment analysis. The user ratings are all the same (5 points), but reviews that include *Excellent* twice, *Powerful*, *Elegant*, and *Fun* yield a slightly higher score. The user rating starts with user input to the hybrid CSB in the form of review text. As an algorithm for sentiment analysis, Valence Aware Dictionary for Sentiment Reasoning (VADER) [64]—a sentiment-analysis tool in the Python NTK library—was used. In VADER, the sentiment score is calculated in the range of −1 to +1.

Each service has a score that corresponds to the sum of the two aforementioned evaluation scores. Finally, on the basis of the service score, the service selection result is calculated according to the service dependency and the vendor. In the example, a pattern consisting of two objects (public and private), compute, and load balancing is selected, and the OpenStack Swift (private cloud storage) owned by the user is bound to the private object storage node.

The process of calculating the combination service selection result for the example shown in Fig. 14 is described below.

-Basic selection method: This is the most basic method to independently select the service with the highest score among the candidate services for each node and select a combination of selected services. In this case, the selected services are B storage, A compute, and B load balancing.

-Consideration of service dependencies: As described in Section 4, there are service-dependency issues that may hinder the combination of services. For example, suppose that the load-balancing service of company



Object Storage	Price	Score
'B' Storage	$100	145
'A' Storage	$100	138
'C' Storage	$90	137
Compute	**price**	**Score**
'A' Compute	$120	148
'B' Compute	$150	145
'C' Compute	$120	138
Load Balancing	**price**	**Score**
'B' Load Balancing	$30	120
'A' Load Balancing	$20	115
'C' Load Balancing	$22	111

Fig. 14 Selected pattern and calculated score example for each candidate service

B is designed only for company B's computing service. Therefore, it is not possible to select the load balancing of compute B from company A as above; thus, it should be selected and recommend to be bundled with the load balancing of compute A from company A or load balancing of compute B from company B.

-One-vendor selection: There is a form to uniformly select vendors of all services to bind to the pattern. The following is a method of collecting the scores of the same vendors, calculating the score for each vendor, and selecting the combination of their services with the highest total score. For example, storage, compute, and load-balancing services of company B are selected. With one-vendor selection, the aforementioned service dependency problem does not apply, and the service of the same vendor is generally connected to a leased line, which is advantageous with regard to network latency. Additionally, to use the service, only the method of using the service for one vendor needs to be learned, and the burden of operation can be reduced by facilitating the deployment and cost management. However, there is increased dependency on the vendor, as well as disadvantages with regard to availability and disaster recovery, and the total price can be higher than that in the case of multiple-vendor services [65].

- Company A: Storage 138 points + Compute 148 points + Load Balancing 115 points = 401 points.
- Company B: Storage 145 points + Compute 145 points + Load Balancing 120 points = 410 points.
- Company C: Storage 137 points + Compute 138 points + Load Balancing 111 points = 386 points.

Review of quality constraints: Users can enter quality constraints for a certain type of service, and services that do not satisfy them are excluded from the selection. For example, suppose that in the example in Fig. 10, the user inputs the requirement "Object storage service must guarantee a response time of 500 ms or less." At this time, if the response time of B storage is 600 ms and the response time value of A storage is 490 ms, B storage is excluded from the selection although the service score of B storage is high.

Considering the maximum budget: If we enter a constraint on the maximum budget, we should select the service so that the sum of the scores is the maximum within the range satisfying the constraint. In the example in Fig. 14, the total price of the currently selected services is $250. If the user has entered $240 as the maximum budget, the service combination with the highest score among the possible combinations of $240 should be output. In the example, we can use company A's instead of company B's load balancing (because company A's load balancing is $10 less expensive than company B's load balancing).

4.2 Process and Method for Cloud Service Integration

This section describes the services that compose the hybrid cloud in a hybrid CSB and how to combine clouds with scripting, which is performed when the relationship is determined by the selection functionality.

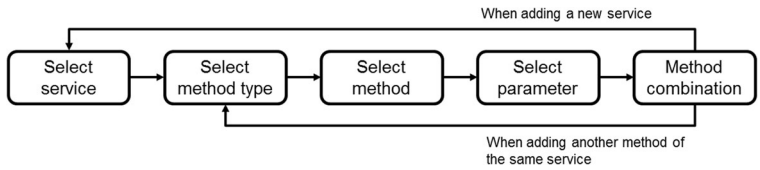


Fig. 15 Deciding script element process

4.2.1 Deciding Script Element in C-SIP

The "decide script element" step in Fig. 3 follows the process shown in Fig. 15. In the hybrid CSB, the user can generate the integration script that went through the script element decision process in Fig. 15. The first step— "select service"—determines the service category and the corresponding service in the category. Next, the method type is selected as one of the two types shown in Fig. 6, in section 3.4. In the next step, the method is output according to the method type, and the user selects the desired method.

When two or more methods are selected, the combination of methods is determined in the method combination. If the user needs an additional method, we proceed from the select service step according to the desired service or proceed from selecting the method type. Script unit or method unit combinations are performed according to the number of methods selected by the user, and script or topology templates are created. The generated script and topology templates are stored in the hybrid CSB for reuse. This creates a real executable integration script written in Python or Java. In Fig. 20 of Chapter 6, a case study, i.e., a prototype of a support tool, is presented to validate the proposed process.

4.2.2 Generating Integration Script in C-SIP

When a script or topology template of XML type generated through the script element determination process is created and stored in the hybrid CSB, an integration script that can be executed is generated through an integration script generation model.

Figure 16 shows the integration script generation model, which is based on Apache Libcloud and software design patterns. It generates real executable code-level integration scripts based on script and topology templates generated by the script element determination process.

Table 4 presents each element of the integration script generation model. Each element was constructed by applying GoF design pattern [66]. Table 5 presents the five design patterns applied to the integration script generation model.

The integration script generation model can be used as a basic generation model of integration script

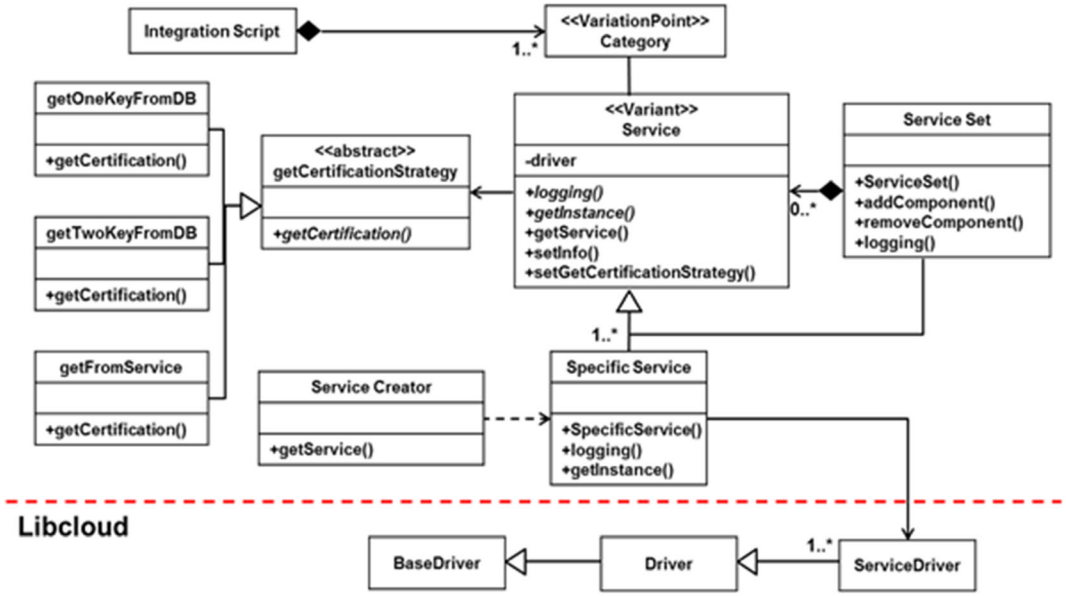


Fig. 16 Integration script generation model using software design pattern

Table 4 Elements of the integration script generation model

Element	Description
Integration Script	It is a script generated through an integration script generation model and has one or more service instances.
<<VariationPoint>> Category	A category is a variation point where variability occurs. It is instantiated and bound to a variant service. The category is typically compute, block storage, or object storage.
<<Variant>> Service	It is an element that is bound to a variation point category and acts as a parent of a specific service belonging to the same category.
Specific Service	It is an element of the actual script, which is a specific cloud service. It executes the method in the actual script.
Service Creator	The service creator is responsible for creating instances of the selected service.
Service Set	The service set is an element that has services as components. Services can be added to and removed from the set through "add" and "remove" methods.
<<abstract>> getCertificationStrategy	It is a class that obtains a strategy to be certified by the cloud service.
getOneKeyFromDB	It is a strategy to acquire an already stored key from the DB; services using one key are used.
getTwoKeyFromDB	It is a strategy to acquire a key already stored in the DB; services using two keys are used.
getFromService	It is a strategy to regain the key from the cloud service when no key is stored or the expired key is in the DB.

Table 5 Integration script generation model elements with design pattern concepts

Pattern	Applicable element	Role
Factory method	<<class>> Service Creator	Creator
	<<class>> Service	Product
	<<class>> Specific Service	Concrete Product
Template method	<<method>> getService()	Template Method
	<<method>> getInstance()	Hook Method
	<<class>> Service	Abstract Class
	<<class>> Specific Service	Concrete Class
Strategy	<<class>> Service	Context
	<<class>> getCertificationStrategy	Strategy
	<<class>> getOneKeyFromDB	Concrete Strategy A
	<<class>> getTwoKeyFromDB	Concrete Strategy B
	<<class>> getKeyFromService	Concrete Strategy C
Composite	<<class>> Service	Component
	<<class>> Service Set	Composite
	<<class>> Specific Service	Leaf
Adapter	<<class>> Service	Target
	<<class>> Specific Service	Adapter
	<<class>> ServiceDriver	Adaptee

generation for building a hybrid cloud service. We can use all the services supported by Libcloud and the methods of these services built on Libcloud. The model can be used for Libcloud and also various multi-cloud library methods. Additionally, it has a structure that can easily expand to include new categories, cloud services, and service methods; thus, it can support various hybrid cloud services.

The hybrid CSB supports the creation of customized hybrid cloud services by creating scriptable element decision processes and models to create executable scripts.

5 Evaluation

5.1 Experiments and Evaluation for Selection

In this section, we present experiments and evaluation results for the pattern-based service selection method presented in this study. In Section 5.1.1, we show that

Table 6 Evaluation results: AWS customer cases

Case	Applied Reference Pattern	Applied CDP	a (%)	b (%)	c (%)
1	Web Application	Scale Out	85.7	66.7	76.2
2	Web Application, Content, and Media Serving	Monitoring Integration	46.2	75	60.6
3	e-Commerce	Scale Out	60	60	60
4	Content and Media Serving	Scale Out, In-Memory DB Cache, DB Replication, Cache Distribution	53.8	77.8	65.8
5	Web Application		100	85.7	92.9
6	Web Application	In-Memory DB Cache	100	75	87.5
7	Web Application	Snapshot, Floating IP, Backnet, Scale Out	81.8	75	78.4
8	Online Games	Scale Out, Cache Distribution	87.5	63.6	75.6
9	Online Games	Scale Out, Cache Distribution	100	63.6	81.8
10	Online Games		75	37.5	51.3
11	Online Games, Log Analysis	Scale Out	100	66.7	83.4
12	Online Games	Queuing Chain	77.8	77.8	77.8
13	Ad Serving, Log Analysis		83.3	71.4	77.4
14	Ad Serving, Log Analysis	Floating IP, In-Memory DB Cache	63.6	63.6	63.6
15	Web Application	Queuing Chain, Web Storage	83.3	62.5	72.9
16	e-Commerce	On-Demand Disk, Backnet	75	66.7	70.9
17	Ad Serving, Log Analysis	In-Memory DB Cache, Backnet	60	60	60
18	Web Application	Queuing Chain, In-Memory DB Cache	80	88.9	84.5
19	Media Sharing	In-Memory DB Cache	87.5	77.8	82.7
20	Web Application	Scale Out, In-Memory DB Cache	90	90	90
21	Large Scale Processing	Multi-Server, In-Memory DB Cache	60	50	55
22	Web Application, Content and Media Serving	Backnet, Storage Index	88.9	88.9	88.9
23	Web Application	Backnet	62.5	62.5	62.5
24	Web Application	Queuing Chain, Storage Index	66.7	44.4	55.6
25	Web Application	Scale Out, Queuing Chain	100	80	90
Average			**78.7**	**69.2**	**73.8**

the patterns selected by the users after pattern selection and revision are reliable via a comparison with the cloud architecture in the AWS casebook. Next, Section 5.1.2 shows how to use the standard and percentile scores in the SAW algorithm, among the various methods for service selection used in the service selection process.

5.1.1 Evaluation for Pattern Selection

In this section, we evaluate the pattern selection part (corresponding to stages 1–2 of the 4th stage service selection process). The evaluation method is as follows.

Among the cloud use cases of 31 companies introduced in the "2017 AWS Korea Customer Casebook" [67], we used 25 cases in which the architecture (AWS service combination) was illustrated. The "2017 AWS Korea Customer Casebook" consists of a company introduction, a challenge for each case, the reason for choosing AWS, the architecture, and the benefits. From the "challenge" and "reason for choosing AWS" sections, we derived the requirements of the company and selected the pattern by applying the pattern selection method. Then, we computed the similarity by comparing the proposed pattern with the architecture of the actual customer case. The similarity was calculated using the following three metrics.

a. The percentage of services that are selected through the selection techniques that constitute the architecture of the actual customer case;

Table 7 Standardization comparison: using standard and percentile scores

QoS Metric	Evaluation CRITERIA	Conventional method [29]	Using standard score	Using percentile score	Using (Standard score + percentile score)
Throughput	Stdev	44.43	22.60	37.14	34.23
	#best	147.8	254.4	127.0	174.6
	#worst	10.4	0.0	1.0	0.0
Fault Rate	Stdev	0.027	0.01	0.002	0.003
	#best	290.8	277.0	299.3	295.6
	#worst	4.0	0.2	0.0	0.0
Response Time	Stdev	0.648	0.260	0.161	0.173
	#best	101.2	70.8	112.6	100.3
	#worst	22.2	0.0	0.33	0.0

b. Among the services selected through the selection techniques, the ratio of services that exist in the architecture of the actual customer case;
c. Arithmetic mean of a and b.

The pattern was selected for all 25 cases, and the similarity was calculated as shown in Table 6. The average values of the accuracy metrics a, b, and c defined for evaluating the pattern selection performance were calculated as 78.7%, 69.2%, and 73.8%, respectively. Thus, according to the metric defined in this experiment, the proposed pattern selection method exhibited an average accuracy of 73.8%.

5.1.2 Evaluation for Service Selection

The SAW score calculation method most commonly used in related studies [29] involves calculating the ratio of the QoS value of the corresponding service to each QoS (maximum value – minimum value). The formula is as follows:

$$r_{ij} = \frac{x_{ij}}{x_j^*} i = 1, \ldots, m; j, =, \ldots, n.$$

We conducted experiments to compare the performance of the conventional calculation method and the calculation method using the standard and percentile scores proposed in this study. We created a Java program that generates 2000 arbitrary services for the experiment. A score can be calculated for all the measurable QoS metrics, which were described in our previous work [10], and the QoS score of service A can be obtained by summing all of the calculated scores. For evaluating our approach, we selected three QoS metrics: throughput, fault rate, and response time, as shown in Table 7. Each service had these three QoS metrics, and the user's weight for each QoS was set as 1:1:1. The scope of generation by the QoS metric was created by referring to the data distribution of the web service QoS dataset used in the study performed by Zheng et al. [68]. For example, according to [68], approximately 17% of services were in the throughput range of 4–8 kbps, and approximately 19% of services were in the throughput range of 8–16 kbps. In each section, uniform random quantities were generated.

In these experiments, we analyzed the characteristics of the top-ranked services when ranking was performed using standard scores and methods used in related studies. The main focus of the evaluation was to confirm that the service with high average quality was stably recommended to the top level. The standard deviations (stdev) of the top 300 selected services (with regard to the total score) calculated using the SAW were measured, including their QoS metrics. The number of cases belonging to the highest (#best) and lowest (#worst) levels were measured. For example, with regard to the response time, the highest and lowest frequencies were < 0.1 and 3.2 ms, respectively. A #worst value of 10 for the response time indicates that for the 2000 services generated, the response time exceeded 3.2 ms, but there were 10 services ranked within the top 300 (with regard to the total score). The "good" selection method assumed in this experiment ensured that the overall QoS was stable and that an excellent service was selected at the top level. Therefore, it aimed to reduce the value of #worst and increase the value of #best. The average value was calculated by repeating the experiment five times. The experimental results are presented in Table 7.

Table 8 Comparison with related research

	[19]	[20]	Script generation model
Service category	Compute	Compute, block, object storage, load balancer	Compute, block, object storage, load balancer, CDN, DNS, container, backup
Number of services	2	8	97
Number of parameters	1	11	18
Number of methods	5	54	75

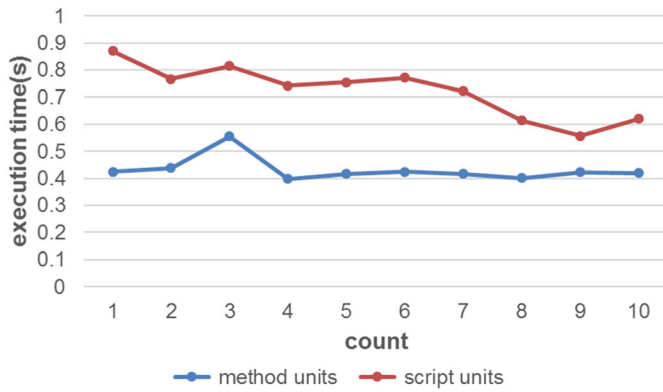
In the case of the method used in a previous related study [29], it was often found that the best quality appears at the top of the selection result if the QoS values are significantly lower than other QoS values (#worst). However, when the standard score was used, it was rarely found that only one QoS was selected within the 300th rank of the lowest class level. The standard deviations of the top 300 services (with regard to the QoS) were also lower when the standard scores were used. Thus, when the standard scores were used, all the QoS values were stable, and the stable services were concentrated at the top of the selection results.

However, the #best values for the fault rate and response time were lower than those for the standard score. In the data used in this experiment, the distribution of the fault rate was significantly shifted to one side, and the response time was close to the normal distribution. However, the scale of the value was smaller than the throughput. In this case, it was judged that it would be more effective to use the percentile score. Table 7 presents the results of using the percentile score and standard score + percentile score instead of the standard score while performing the foregoing experiment.

Regarding the fault rate and response time, when the percentile score was used instead of the standard score, both the #best evaluation criteria and the other two evaluation criteria exhibited good results. However, the standard score yielded better performance when the distribution of values, such as the throughput and the scale of the scores, was above average. Using both standard and percentile scores can compromise the features of the two methods, both of which exhibited better performance than the conventional method.

5.2 Integration Approach Evaluation

The script generation method to support cloud combining, proposed in this study, was realized by creating scripts using a multi-cloud library that employs the inter-cloud and multi-cloud coupling structure introduced in Grozev's study [18]. Table 8 compares previously reported models [19, 20] and the integration script generation model of this study with regard to the service category, number of services, number of supporting parameters, and number of methods. The support parameter refers to the object to which the method is



	method units	script units
Average time(s)	0.4316	0.7233

Fig. 17 Execution time of integration script combination structure

Table 9 User requirements for selection

Type of requirement	Requirements
Target system	Media streaming service
Quality constraints	Response time of CDN: <100 ms
Functional requirements	Server status backup Simultaneously use public and private storage
Maximum budget	$1000/month
Existing infrastructure	OpenStack Nova, Glance, Cinder, Swift

applied, for example, a node representing a virtual machine of a compute service, and a container representing a compartment of data in a storage service. The number of parameters excludes the key or token that accesses the service's resources. Table 6 indicates that the integration script generation model supports more categories, services, parameters, and methods than the previously reported models [19, 20].

Compared with Silva's platform [41], which provides an integration script generation model and an abstraction function, the generation model supports more methods, because abstraction methods common to Libcloud and specific methods of services can be used.

Additionally, the generation model adopts the software design pattern and is constructed according to the model; thus, the service category, service, and method can be easily expanded. For example, to add a service category, we create a new model with the structure of the integration script generation model. To add a new service, we add a specific service of the integration script generation model and define a new method in the corresponding specific service to newly define the method.

The previously reported models [19, 20, 41] did not introduce an approach to combine the provided method or a script to create a new function and a technique to control various cloud services through a common abstraction method. However, the integration script generation method can realize new functions and reuse by combining various methods supported by Libcloud, as well as techniques for combining scripts that have already been generated.

In addition, to compare and analyze the two types of combinatorial scripts, we measured their execution speeds. Each combination structure was written in Python, and 10 functions were sequentially performed in OpenStack Swift. The execution-time measurement was performed 10 times, and the measurement results are shown in Fig. 17.

The combined structure of method units was approximately 1.67 times faster than the combined structure of script units. It is expected that the difference in execution speed is larger when the number of combined functions is increased, complex functions are performed, and functions of multiple cloud services are executed.

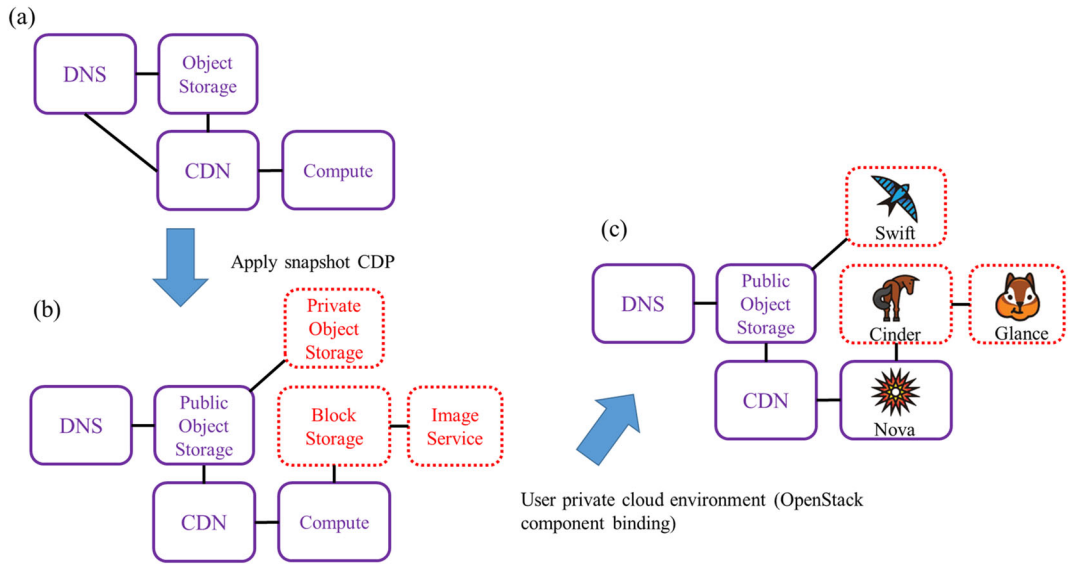
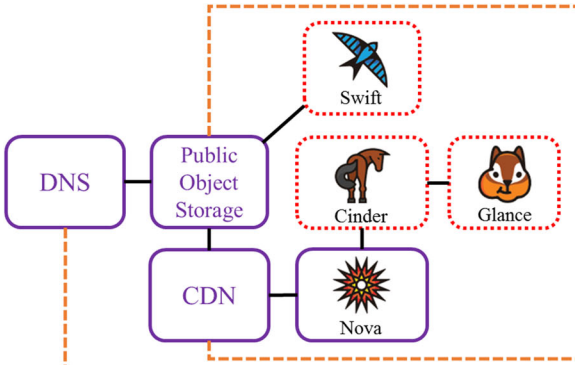


Fig. 18 Pattern selection, modification, and service binding of the case study



Object Storage	Price	ResponseTime	Reliability	Score
AWS Storage	$450	230 ms	99.9995 %	235
Google Storage	$430	250 ms	99.999%	217
Azure Storage	$415	275 ms	99.9995 %	212
Bluemix Storage	$410	280 ms	99.999%	200

DNS	Price	Usability	Reliability	Score
AWS DNS	$200	8	99.995%	215
Google DNS	$220	7	99.999%	216
Azure DNS	$220	9	99.999%	225
Bluemix DNS	$240	10	99.999%	236

CDN	Price	availability	ResponseTime	Score
AWS CDN	$300	99.9%	99 ms	210
Google CDN	$310	99.95%	108 ms	214
Azure CDN	$330	99.99%	96 ms	231
Bluemix CDN	$360	99.99%	92 ms	236

Fig. 19 New service selection

The execution speed of the script unit combination structure is lower than that of the method unit combination structure, but when the pool of the script is formed later, the new function can be quickly implemented by combining the existing script. Therefore, the reusability of the script can be increased, and the resources of the hybrid CSB can be efficiently used.

If we allow the user to choose a combination structure, most users will choose method unit combination because it is faster. Therefore, to efficiently manage scripts in the hybrid cloud service, the number of methods to be combined in the script is set as the criterion for the combination structure selection according to the experiment results, as shown in Fig. 17. If the number of combined methods is smaller than the standard, we configure the script with a combination of script units that can reuse the script.

Table 10 Selected options: multi-vendor vs. one-vendor

Options	Cost	Score	Remarks
Multi-vendor option	**$1000**	**691**	
AWS	**$950**	**660**	
Google	$960	647	Google CDN: Quality requirements not satisfied
Azure	**$965**	**668**	
Bluemix	$1010	672	Overbudget

6 Case Study of C-SIP

This section presents an example of building a hybrid cloud environment via the pattern selection and integration process described in Section 4.

The user in this case is a cloud administrator of a company that is building a private cloud based on OpenStack. The company is planning a video-streaming service with new services. Although they already have computing resources, OS, and storage, they wish to add the infrastructure to quickly transfer media content to customers on video-streaming sites worldwide using the public cloud. The user enters the requirements presented in Table 9 into the hybrid CSB.

6.1 Pattern and Combination Service Selections

First, in the pattern selection step, a reference pattern suitable for a video-streaming service is found and se-

Table 11 Function of the script to be created

Function	Required services	Required method
Create a new virtual machine (VM) and a new volume and attach the volume to the VM	Nova, Cinder	create_node, create_volume, attach_volume

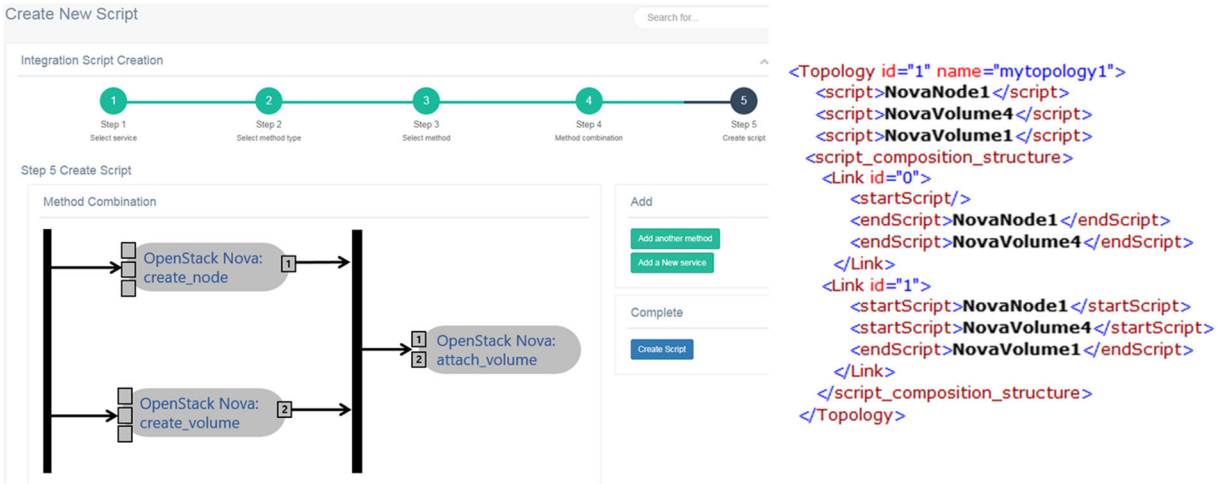


Fig. 20 User interface for creating a script and the topology template

lected. It is similar to the user's goal of building a video-streaming service with the "Content and Media Serving" architecture, which is one of the reference architectures provided by AWS. Figure 18(a) shows an abstraction of the pattern defined in this study. This pattern is selected by the user.

In the pattern modification phase, the functional requirement entered by the user is checked to find a CDP that can satisfy the requirement. There is a snapshot pattern of the AWS CDP in the CDP that can be employed to add a server state backup function. The snapshot pattern consists of compute (EC2), block

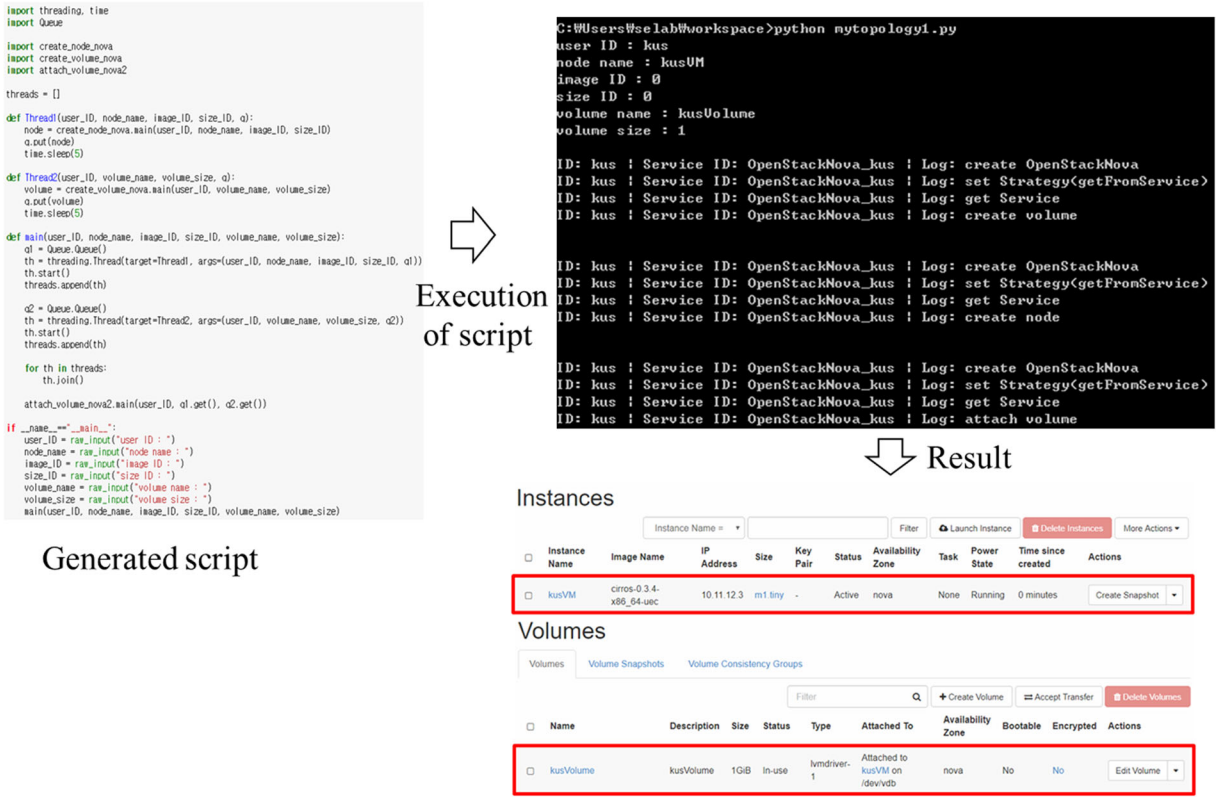


Fig. 21 Executable topology script, executions, and result

storage (EBS), and image service (AMI). We add block storage and image service nodes that do not exist in the currently selected pattern. The requirement to use both public and private storage can be addressed by using the hybrid data pattern of the CDP provided in [12–14]. This pattern consists of two types of object storage (public and private); thus, we add another object storage node to the pattern. The modified pattern using these two CDPs is shown in Fig. 18(b).

The next step is the existing service binding step. Users have an OpenStack-based private cloud environment, and four types of services are in operation: Nova (compute), Glance (image), Cinder (block storage), and Swift (object storage). Figure 18(c) shows the service corresponding to each node of the selected pattern after it is bound.

Finally, there is a process of selecting a new service for each of the remaining three nodes. In this case study, it is assumed that there are candidate services with the same price and QoS for each service type, as shown in Fig. 19 (each QoS is arbitrary, not a measured value). Each candidate service has an SAW score based on the QoS and a feedback score based on user feedback. The service score is the sum of the SAW and feedback scores.

Once the service scores for each service have been calculated, the combination service is finally selected through the following procedure.

- Confirmation of quality requirement: Google CDN's response time was 103 ms and did not satisfy the quality requirements entered by the user. Therefore, they are excluded from referrals.
- Consideration of maximum budget: We use the MCKP algorithm to find the service combination with the highest total score within the maximum budget of $1000 per month. For example, if we individually select the service with the highest score for each service type, the sum of the prices exceeds $1050.
- Service dependency checking: AWS storage and Azure CDN cannot be combined, assuming that the CDN and object storage services have dependencies on the combination of services from the same vendor. In this case, a combination of AWS storage and AWS CDN ($750, 445 points), or a combination of Azure storage and Azure CDN ($745, 443 points) is available.
- One-vendor option offered: Through the above process, an optimal multi-vendor combination is selected. However, if the user wants to take advantage of the one-vendor combination even though the total service score is low, the one-vendor combination can be selected. Combining services with a single vendor can benefit from administrative and operational advantages, such as managing the entire service through only the console provided by the vendor, and advantages related to network latency. However, the price-to-performance ratio of individual services and the ability to endure disasters may be inferior to those of the multi-vendor combination.

Table 10 presents the sum of the price and service points for each of the one-vendor combinations and the multi-vendor combinations selected above. For the Google service combinations, Google CDNs do not satisfy the quality requirements, and the Bluemix service combinations exceed the budget. Therefore, AWS, Azure, and multi-vendor combinations are preferred (as highlighted in bold in Table 10).

6.2 Service Integration

In this section, we present an example of prototyping a script that implements a function in a selected hybrid cloud environment. The function for the case is described in Table 11. In the current prototype, scripts with corresponding functions are created via a combination of script units. Figure 20 shows the Create Script screen of the integration script generation. If we need another service or method in the process, we can add elements as they are. When we click the Create Script button, an integration script is created through the elements that we have determined.

The right side of Fig. 20 shows the XML topology template (indicated in Table 11), which is created through the process of generating the integration script of the prototype. The fork and join structure is represented by Link. Additionally, this process creates an executable topology script, as shown on the left side of Fig. 21. It imports three scripts and executes the main function. The fork and join structure is implemented via Python threading. OpenStack volumes can be implemented via OpenStack Cinder (block storage) and can also be implemented through Nova. The script manages volumes with Nova.

The upper right side of Fig. 21 shows the screen where the function is executed. We ran the Python script generated from the command prompt port of Windows. The required input was obtained via user input, and it executed the methods. The log of each method execution result was output. The bottom right side of Fig. 21 shows the resources of OpenStack after the actual function was executed. As shown in Fig. 21, a virtual machine called kusVM and a volume called kusVolume were created, and we see that kusVolume was connected to kusVM.

Thus, the hybrid CSB can combine the selected services and their methods, and it can combine the methods and scripts that users want with the script generation prototype.

7 Conclusion

We proposed a C-SIP and a hybrid CSB, which are automation solutions that support hybrid cloud deployment. The C-SIP consists of a cloud service selection method that chooses a combination of various clouds according to user's requirements and a script generation method to solve the complexity problem between clouds. In the selection technique, we refer to the reference patterns and CDPs provided by major public cloud vendors such as AWS and Azure. We defined a generalized pattern and proposed a method for selecting patterns, suggesting that the service derived from the selection result is bound to a pattern. Finally, the accuracy of the proposed pattern selection method for evaluation was measured through comparison with the architecture in the AWS casebook. We proposed a combination of cloud services with customized patterns, so that users can simply input the purpose and requirements of building the cloud services. Script generation techniques that support cloud-to-cloud integration include a process in which a user determines the elements of a script, as well as a model that supports executable code-level script generation. The integration script generation model has a structure that can easily expand the services and methods belonging to various service categories by applying a design pattern. When a user selects a service, method, and method structure through the integration script generation process, a script having a desired function is generated through the combination of a script unit or a method unit. A case study of script generation via method unit and script unit combination was presented. Additionally, we showed that the integration script generation technique can support various service categories, services, and methods through comparative analysis with related studies. The script generation method allows the user to conveniently utilize various cloud functions. The case study was conducted for an integrated scenario of the selection and script generation techniques. The processes of selecting an actual cloud and generating a script were presented for a prototype. Through the proposed C-SIP, users can build their own hybrid cloud environment, which is expected to facilitate the introduction of hybrid clouds and the acquisition of cloud strategies. In future research, we plan to improve the cloud environment by analyzing the user feedback and monitoring the log of the established cloud environment.

Acknowledgements This work was supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIP) and the Ministry of Education (No. NRF-2016R1D1A1B03935865, No. NRF-2017R1D1A1B03030243).

References

1. Liu, F., Tong, J., Mao, J., Bohn, R., Messina, J., Badger, L., Leaf, D.: NIST cloud computing reference architecture. NIST Special Publication. (2011)
2. Flexera, RightSclae 2019 State of the Cloud Report form Flexera, https://www.flexera.com/about-us/press-center/rightscale-2019-state-of-the-cloud-report-from-flexera-identifies-cloud-adoption-trends.html
3. Marketwatch, Hybrid Cloud Market Size, Growth, Opportunity and Forecast. https://www.marketwatch.com/press-release/hybrid-cloud-market-size-growth-opportunity-and-forecast-2019-11-04
4. Dillon, T., Wu, C., Chang, E.: Cloud computing: issues and challenges. IEEE International Conference on Advanced Information Networking and Applications. (2010). https://doi.org/10.1109/AINA.2010.187

5. Micore Solutions, What are the major challenges of adopting a hybrid cloud approach? http://www.micoresolutions.com/major-challenges-adopting-hybrid-cloud-approach
6. Informa PLC, Managing Hybrid Cloud: 3 Challenges, https://www.networkcomputing.com/cloud-infrastructure/managing-hybrid-cloud-3-challenges/2100227984
7. OTAVA, Resources, https://www.otava.com/about/resources/videos/
8. Park, J., Yun, D., Kim, U.: Approach for cloud recommendation and integration to construct user-centric hybrid cloud. IEEE Conference on SmartCloud. (2017). https://doi.org/10.1109/SmartCloud.2017.11
9. Park, J., Yun, D., Kim, U., Yeom, K.: Pattern-based cloud service recommendation and integration for hybrid cloud. IEEE Symposium on Cloud and Service Computing. (2017). https://doi.org/10.1109/SC2.2017.40
10. Park, J., Kim, U., Yun, D., Yeom, K.: C-RCE: an Approach for Constructing and Managing a Cloud Service Broker. C-RCE: An approach for constructing and managing a cloud service broker. J. Grid Comput. **17**, 137–168 (2019). https://doi.org/10.1007/s10723-017-9422-2
11. AWS, AWS Reference Architecture, https://aws.amazon.com/architecture/?nc1=h_ls
12. Microsoft Azure, Azure Reference Architecture, https://docs.microsoft.com/en-us/azure/architecture/reference-architectures
13. IBM, IBM Bluemix Reference Architecture. https://www.ibm.com/cloud/garage/architectures
14. Young, M.: Implementing cloud design patterns for AWS. O'REILLY. (2015)
15. Microsoft Azure, Azure Cloud Design Pattern. https://docs.microsoft.com/en-us/azure/architecture/patterns
16. Garg, K.S., Versteeg, S., Buyya, R.: SMICloud: a framework for comparing and ranking cloud services. IEEE International Conference on Utility and Cloud Computing. (2011). https://doi.org/10.1109/UCC.2011.36
17. Zheng, X., Xu, D.L., Chai, S.: QoS recommendation in cloud services. IEEE Access. **5**, 5171–5177 (2017). https://doi.org/10.1109/ACCESS.2017.2695657
18. Grozev, N., Rajkumar, B.: Inter-cloud architectures and application brokering: taxonomy and survey. Softw.: Pract. Exp. **44**, 369–390 (2014). https://doi.org/10.1002/spe.2168
19. Markoska, E., Ackovsak, N., Ristov, S., Gusev, M.: Software design patterns to develop an interoperable cloud environment. IEEE Telecommun. Forum Telfor. (2015). https://doi.org/10.1109/TELFOR.2015.7377630
20. Meireles, F.: Integrated Management of Cloud Computing Resources. Diss. Instituto Superior de Engenharia do Porto (2014)
21. Saaty, R.: The analytic hierarchy process – what it is and how it is used. Math. Model. **9**, 161–176 (1987). https://doi.org/10.1016/0270-0255(87)90473-8
22. Gal, T., Stewart, T., Hanne, T.: Multicriteria Decision Making: Advances in MCDM Models. Theory, and Applications. Kluwer Academic Publishers, Algorithms (1999)
23. Whaiduzzaman, M., Gani, A., Anuar, N., Shiraz, M., Haque, M., Haque, I.: Cloud service selection using multicriteria decision analysis. Sci. World J. **2014**, 1–10 (2014). https://doi.org/10.1155/2014/459375
24. Carnegie Mellon University, Service Measurement Index Framework Version 2.1 (2014)
25. Taira, H., Fan, Y., Yoshiya, K., Miyagi, H.: A method of constructing pairwise comparison matrix in decision making. In: Proc. IEEE International Conference on Systems, Man and Cybernetics. Information Intelligence and System. pp. 2511–2516 (1996)
26. Su, X., Khoshgoftaar, T.: A survey of collaborative filtering techniques. Adv. Artif. Intell. **2009**, 1–19 (2009). https://doi.org/10.1155/2009/421425
27. Benesty, J., Chen, J., Huang, Y., Cohen, I.: Pearson Correlation Coefficient. In: Noise Reduction in Speech Processing. Springer Topics in Signal Processing, Vol 2. Springer, Berlin, Heidelberg (2009)
28. Myers, L., Sirois, M.: Spearman correlation coefficients. Differences Between. In: Wiley StatsRef: Statistics Reference Online. (2006). https://doi.org/10.1002/0471667196.ess5050.pub2
29. Saripalli, P., Pingali, G.: MADMAC: multiple attribute decision methodology for adoption of clouds. IEEE International Conference on Cloud Computing. (2011). https://doi.org/10.1109/CLOUD.2011.61
30. Sidhu, J., Singh, S.: Design and comparative analysis of MCDM-based multi-dimensional trust evaluation schemes for determining trustworthiness of cloud service providers. J. Grid Comput. **15**, 197–218 (2017). https://doi.org/10.1007/s10723-017-9396-0
31. Sirohi, P., Agarwal, A., Maheshwari, P.: A comparative study of cloud computing service selection. Int. J. Eng. Adv. Technol. **8**, 259–266 (2019)
32. Sidhu, J., Singh, S.: Improved TOPSIS method based trust evaluation framework for determining trustworthiness of cloud service providers. J. Grid Comput. **15**, 81–105 (2017). https://doi.org/10.1007/s10723-016-9363-1
33. Martino, D.B., Esposito, A., Cretella, G.: Semantic Representation of Cloud Patterns and Services with Automated Reasoning to Support Cloud Application Portability. In: Semantic Representation of Cloud Patterns and Services with Automated Reasoning to Support Cloud Application Portability. IEEE Trans, Cloud Comput (2015). https://doi.org/10.1109/TCC.2015.2433259
34. W3C, SPARQL Query Language for RDF, http://www.w3.org/TR/rdf-sparql-query
35. Liu, Z., Chu, D., Song, C., Xue, X., Lu, B.: Social learning optimization (SLO) algorithm paradigm and its application in QoS-aware cloud service composition. Inf. Sci. **326**, 315–333 (2016). https://doi.org/10.1016/j.ins.2015.08.004
36. Câmara, D., :1- Evolution and Evolutinary Algorithms, in Bio-inspired Network, https://www.sciencedirect.com/science/article/pii/B9781785480218500016#!
37. Apache, Apache Libcloud, http://libcloud.apache.org
38. Apache, Apache Jcloud, https://jclouds.apache.org
39. Apache, Apache Deltacloud, https://deltacloud.apache.org
40. Chadwick, D., Siu, K., Lee, C., Fouillat, Y., Germonville, D.: Adding federated identity management to OpenStack. J. Grid Comput. **12**, 3–27 (2014). https://doi.org/10.1007/s10723-013-9283-2
41. Silva, L., Costa, C., Oliveira, J.: A common API for delivering services over multi-vendor cloud resources. J. Syst. Softw. **86**, 2309–2317 (2013). https://doi.org/10.1016/j.jss.2013.04.037

42. AWS, Hybrid Cloud with AWS, https://aws.amazon.com/hybrid/?nc1=h_ls
43. Microsoft Azure, Service Bus, https://azure.microsoft.com/en-us/services/service-bus/
44. Microsoft Azure, Integration Services, https://azure.microsoft.com/mediahandler/files/resourcefiles/azure-integration-services/Azure-Integration-Services-Whitepaper-v1-0.pdf
45. Google, Manage hybrid cloud, https://cloud.google.com/solutions/manage-hybrid-cloud/
46. Google, Anthos, https://cloud.google.com/anthos/
47. Google, Google cloud Hybrid Connectivity, https://cloud.google.com/hybrid-connectivity/
48. Hashicorp, Terraform. https://www.terraform.io/
49. Kovács, J., Kacsuk, P.: Occopus: a multi-cloud orchestrator to deploy and manage complex scientific infrastructures. J. Grid Comput. **16**, 19–37 (2018). https://doi.org/10.1007/s10723-017-9421-3
50. Caballer, M., Zala, S., García, Á., Moltó, G., Fernández, P., Velten, M.: Orchestrating complex application architectures in heterogeneous clouds. J. Grid Comput. **16**, 3–18 (2018). https://doi.org/10.1007/s10723-017-9418-y
51. Salomoni, D., Campos, I., Gaido, L., de Lucas, J.M., Solagna, P., Gomes, J., Matyska, L., Fuhrman, P., Hardt, M., Donvito, G., Dutka, L., Plociennik, M., Barbera, R., Blanquer, I., Ceccanti, A., Cetinic, E., David, M., Duma, C., López-García, A., Moltó, G., Orviz, P., Sustr, Z., Viljoen, M., Aguilar, F., Alves, L., Antonacci, M., Antonelli, L.A., Bagnasco, S., Bonvin, A.M.J.J., Bruno, R., Chen, Y., Costa, A., Davidovic, D., Ertl, B., Fargetta, M., Fiore, S., Gallozzi, S., Kurkcuoglu, Z., Lloret, L., Martins, J., Nuzzo, A., Nassisi, P., Palazzo, C., Pina, J., Sciacca, E., Spiga, D., Tangaro, M., Urbaniak, M., Vallero, S., Wegh, B., Zaccolo, V., Zambelli, F., Zok, T.: INDIGO-DataCloud: a platform to facilitate seamless access to E-infrastructures. J. Grid Comput. **16**, 381–408 (2018). https://doi.org/10.1007/s10723-018-9453-3
52. OASIS, OASIS: TOSCA Simple Profile in YAML Version 1.1, http://docs.oasis-open.org/tosca/TOSCA-Simple-Profile-YAML/v1.1/TOSCA-Simple-Profile-YAML-v1.1.html (2018)
53. OASIS, Topology and Orchestration Specification for Cloud Applications Version 1.0, http://docs.oasis-open.org/tosca/TOSCA/v1.0/TOSCA-v1.0.html (2013)
54. Bernal, A., Cambronero, E., Núñez, A., Cañizares, P., Valero, V.: Improving cloud architectures using UML profiles and M2T transformation techniques. J. Supercomput. **75**, 8012–8058 (2019). https://doi.org/10.1007/s11227-019-02980-w
55. Bergmayr, A., Breitenbücher, U., Ferry, N., Rossini, A., Solberg, A., Wimmer, M., Kappel, G., Leymann, F.: A systematic review of cloud modeling languages. ACM Comput. Surveys. **51**, 1–38 (2018). https://doi.org/10.1145/3150227
56. OMG, OMG Unified Modeling Language Version 2.5, https://www.omg.org/spec/UML/2.5/PDF
57. Katsaros, G., Menzel, M., Lenk, A., Rake-Revelant, J., Skipp, R., Eberhardt, J.: Cloud application portability with TOSCA, Chef and Openstack: Experiences from a proof-of-concept implementation. In: Proc. IC2E 2014. pp. 295–302 (2014)
58. Galante, G., Erpen De Bona, L.C., Mury, A.R. et al. : An Analysis of Public Clouds Elasticity in the Execution of Scientific Applications: a Survey, J. Grid Comput. (2016) doi: https://doi.org/10.1007/s10723-016-9361-3
59. Arcitura, Cloud Computing Design Patterns and Mechanisms, https://patterns.arcitura.com/cloud-computing-patterns
60. Cloud Computing Patterns, Cloud Computing Patterns, http://www.cloudcomputingpatterns.org
61. Expósito, R.R., Taboada, G.L., Ramos, S., González-Domínguez, J., Touriño, J., Doallo, R.: Analysis of I/O performance on an amazon EC2 cluster compute and high I/O platform. J. Grid Comput. **11**, 613–631 (2013). https://doi.org/10.1007/s10723-013-9250-y
62. AWS, AWS Architecture Center, https://www.umbrellainfocare.com/wp-content/uploads/2017/07/architecture
63. Neukrug, E., Fawcett, R.: Essentials of testing and assessment: a practical guide for counselors, social works, and psychologists. CENGAGE Learning. (2006)
64. Hutto, C., Gilbet, E.: VADER: A Parsimonious Rule-Based Model for Sentiment Analysis of Social Media Text, International AAAI Conference on Weblogs and Social Media, pp. 216–225 (2014)
65. Blogs, C.: Multi-Vendor vs. Single-Vendor: Making the Choice. https://blogs.cisco.com/smallbusiness/multi-vendor-vs-single-vendor-making-the-choice
66. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley Reading, Massachusetts (1995)
67. AWS, AWS Customer stories, https://aws.amazon.com/ko/blogs/korea/now-available-aws-korean-customer-cases (in korean)
68. Zheng, Z., Zheng, Y., Lyu, R.M.: Investigating QoS of real-world web services. IEEE Trans. Serv. Comput. **7**, 32–39 (2014). https://doi.org/10.1109/TSC.2012.34