DCCSOA: A Dynamic Cloud Computing Service-Oriented Architecture

Mehdi Bahrami 1 and Mukesh Singhal 2

Cloud Lab,
University of California, Merced
1 Senior Member of IEEE, MBahrami@UCMerced.edu
2 IEEE Fellow, MSinghal@UCMerced.edu

Abstract—The emerging field of Cloud Computing provides several advantages over traditional in-house IT services, such as accessing to elastic on-demand computing and storage over the Internet, and cost effective pay-per-use subscription plans. However, according to the International Data Corporation (IDC), cloud computing has several issues, such as a lack of standardization, a lack of customization, and limited interoperability. In addition, there is an increasing demand for introduction and migration of a variety of services to cloud computing systems, which are abstract their offering services into various *-as-a-Services (*aaS) layers. Although each such service provides a new feature (e.g., simulation services in cloud), it aggravates the issues due to the lack of standardization and inability to customize services by a vendor because each *aaS has its own features, requirements and output. In this paper, we propose a cloud architecture to alleviate issues associated with standardization and customization. In the cloud, the proposed architecture uses a single layer, called Template-as-a-Service (TaaS), to provide: (i) a single service layer for interaction with all resources and major cloud services (e.g., IaaS, PaaS, SaaS and *aaS), (ii) a standardization for existing services and future *aaS across different cloud environments, and (iii) a customizable architecture which can be modified on demand by a cloud vendor, and its partners to provide the flexibility on cloud computing systems. A comparison with previous studies show that the proposed architecture provides customization and standardization for cloud services with minimum modifications.

Keywords— Cloud Computing; Cloud Architecture; Cloud Customization; Cloud Standardization; *-as-a-Service; SOA;

I. INTRODUCTION

Cloud computing is based on a distributed and parallel computing systems that provide elastic storage resources and computing resources over the Internet. The cloud computing paradigm allows customers to pay for their resource usage based on pay-per-use model, and enables customers to scale their storage and computing resources up or down on-demand [1].

An important aspect of cloud computing is cloud architecture [1] that refers to the components (e.g., service layers), subcomponents (e.g., security or message passing in service layers), and overall system organization of cloud computing [2, 3]. Moving successfully into cloud computing requires an architecture that will support new capabilities for migrating different traditional services and applications to cloud computing systems. Such an architecture should support all user domains of a cloud computing system which includes cloud vendors, cloud developers, cloud customers or cloud vendors partners, and end-users.

II. MOTIVATION

This dependency on cloud services creates several issues [4, 5, 6]. For example, according to the IDC Survey [4], 79.8% people say “Bringing back in-house may be difficult” is another issue and 76.8% people say “Hard to integrate with in-house IT” is an issue. These issues indicate consumers are afraid of migrating to cloud computing systems because the migration is difficult to integrate with IT department services and it is difficult to return the data back to the IT department. The survey shows 80.2% of people say “Lack of interoperability standards” is another concern. Thus, cloud computing requires interoperability with other cloud computing systems; also as indicated in this report, 76.0% of respondents answer that “Not enough ability to customize” is an issue. Similar significant concerns around cloud computing are reported recently in other studies [5, 6, 7]. Furthermore, all of these concerns show that cloud computing systems require flexibility in defining a variety of services that meet specific cloud users’ requirements. The flexibility in defining services can be implemented by a customizable architecture that allows a vendor to define a service for each group of users.

Cloud vendors provide several services to their customers through a general multi-tier architecture (IaaS, PaaS and SaaS). Although this architecture is useful for several customers’ requests, customers may have own specific request. Customers should adapt his request based on offered services, because each offered service intends to satisfy unique user requests. For example, when a customer requests a service in PaaS for developing an image processing application, the customer has the same accessibility to Application Programming Interfaces (API) as other customers who develop a web mining application on a cloud. However, an image processing application requires specific functions (e.g., spatial transformations) that are different in type and not useful for a web mining application that requires more specific network functionality (e.g., spatial indices). This example shows that the customization of a service by a cloud vendor allows a cloud vendor to provide unique service to each customer. A customized service allows customers to have a simple system or API rather than a complex system or a complex API that intends to satisfy different users’ demands. For example, a cloud vendor could define a customized service that only satisfies a small group of partners or users, such as a group of users who only need Voice-over-IP service (VoIP) in a cloud computing system.
Another concern in cloud computing is an increasing demand for the introduction and migration of a variety of services to cloud computing systems, that are a type of *-as-a-Services (*aaS). Although each service provides a new feature, such as Simulation-as-a-Service [8] or Robot-as-a-Service [9], it aggravates migration issues and complexity issues due to the lack of standardization and customization, respectively because each *aaS has its own features, requirements and output. For example, Robot-as-a-Service provides a platform to control robot devices through a cloud computing system. This service requires different resources and it provides different outputs. A dynamic architecture allows vendors to add/edit their services and future *-as-a-services to their cloud computing systems with ease.

In this paper, we propose a dynamic and customizable architecture that targets mentioned concerns, such as providing customizable and dynamical services, a standardization for different cloud vendors with different solutions, supporting different services (*aaS) in a cloud computing system, and a solution for cloud vendor lock-in issue.

III. RELATED WORK

Currently, we do not have a generally accepted standard for cloud computing. Unlike the Internet which was developed by the U.S. government agencies [10], such as ARPA [11], cloud computing has been developed by several open-source groups and leading business companies, such as Microsoft and Amazon. Therefore, several independent cloud architectures have been developed.

To the best of our knowledge, no effective architecture exists that supports dynamic customization. As previously discussed, the lack of ability for customization is one of the major issues in existing cloud architectures. This drawback of existing cloud architecture creates other issues, which are discussed in Section II, such as migration issues. We have several solution to overcome this drawback by implementing customization at different level of cloud computing systems. As shown in Figure 1, we divided customization of cloud computing systems into conceptual level, architecture level and implementation level. In the following section, we review related work in each level of customization.

A. Conceptual Level

Conceptual level provides a high-level definition of a customized system. Based on customization at the conceptual level, we can define an architecture and its implementation. For example, one of the conceptual customization is Mass Customization (MC) [12] which is based on marketing and manufacturing. MC focuses on developing one product with different features. For instance, Hu et al. [13] proposed a mass customization for their proposed cloud architecture (CCRA), which enables a cloud vendor to define a cloud architecture requirements and its implementation. In their architecture, different models of one object could be defined by a conceptual model. Each object has different features. Their concept provides different services through a dynamic domain with different abstractions which is called a model. Although Hu et al. provide a customization model in cloud computing, the model is not adoptable because authors did not provide the specific detail of implementation methods for a diverse environments.

![Figure 1. Customization Levels](image)

B. Architecture Level

Existing cloud architectures are static, and are divided into the following categories:

(i) Service-Oriented Architectural (SOA) [14] based: Tsai et al. provide SOCCA [15] which is a combination of Enterprise SOA style and cloud style and Zhang et al. provide CCOA [16] architecture based on SOA with a scalable service feature, but these cloud architectures do not provide customization on each service layer;

(ii) Cloud Reference Architecture (CRA) [17] which is developed by NIST. This architecture has five primary actors: Cloud Service, Consumer, Cloud Service Provider, Cloud Broker, Cloud, Auditor and Cloud Carrier;

(iii) Open forums, such as OGF Open Cloud Computing Interface [18], Cloud Computing Interoperability Forum (CCIF), Deltacloud [19], DMTF, Open Stack [19], Open Cloud Consortium and Open Cloud Computing Interface (OCCI) [20].

The idea behind most of these open source clouds is to provide a common interface that includes major cloud platforms. However, in this paper, we propose an architecture that allows vendors to define and implement their own specific service through a standardized layer cross all other vendors’ platforms. In the proposed architecture, the vendor uses a layer to provide standard services to their customers. The vendors are not required to modify their platform and they can provide an extension layer on the top of their cloud platform.

Existing cloud architectures do not provide any solution for facilitating different services, such as *aaS. In addition, existing cloud architectures are static and could not easily provide a customization on services.

---

1 Available from: http://www.cloudforum.org/  
2 Available from: http://dmtf.org/standards/cloud  
3 Available from: http://opencloudconsortium.org  
4 Available from: http://occi-wg.org/about
C. Implementation Level

Customization at implementation level allows a vendor to define several separate services and applications. Customization at this level is often tied to a vendor’s platforms and infrastructures.

Major customization at implementation level has been developed by using Object-Oriented (OO) paradigm. The concept of OO enables developer to implement an application based on different objects which are closely linked. For customization reasons, several implementation models have been developed for cloud computing systems. For example, Bahga et al. [21] provide a Cloud Computing Model (CCM) which is a component-based model for cloud computing systems. The CCM allows a developer to provide multiple components which are connected via Uniform resource identifier (URI) and uses message passing. Although, the model provides a customization for cloud applications, CCM is relied on cloud architecture.

The CCM has several drawbacks. For example, if an architecture is non-functional, then the implementation model cannot provide an efficient model. For example, limitation on network access at PaaS layer it causes limitation on CCM application (i.e., the lack of accessibility to a protocol). Implementation of CCM also has some drawbacks because the model depends on the architecture with specific requirements, such as type of programming language. These issues show disadvantages of a cloud architecture could be caused issue in the implementation.

IV. THE PROPOSED ARCHITECTURE

This section presents a Dynamic Cloud Computing Service-Oriented Architecture (DCCSOA) that allows cloud vendors to analyze, design, develop and implement a cloud computing system. The DCCSOA provides a dynamic service layer that allows a vendor to add new customized services on-demand.

A dynamic architecture for cloud computing allows cloud vendors to customize their services. As shown in Figure 2, the architecture is based on SOA. The SOA features enable an architecture to provide several independent services that work together as a system and can be run on different cloud computing systems. The proposed architecture can customize value-added cloud services (offered resources on a cloud computing system). In the proposed architecture, a dynamic layer represents all heterogeneous services, and it can customize services on-demand.

A. DCCSOA Components

The DCCSOA has several service layers that are discuss as follows:

Dynamic Template Service Layer (DTSL): The DTSL provides a dynamic and customizable bridge between all value-added services in a cloud computing system and all cloud user groups, such as cloud vendor users, cloud customers (partner of cloud vendors), cloud developers and cloud end-users. The DTSL is a primary component of the proposed architecture and it provides a service layer which we call “Template-as-a-Service (TaaS)”. The TaaS provides a dynamic customization on value-added services. The DTSL is divided into two sub-layers as follows: “Front-end of Template-as-a-Service (FTaaS)” and “Back-end of Template-as-a-Service (BTaaS)”. The BTaaS provides customized value-added cloud services to cloud clients by Cloud Client Dashboard. The BTaaS is only available to cloud vendors and it interacts with all cloud services, such as all traditional services.

Cloud Client Dashboard (End-users, Developer and Third-party users with regular end-user or developer role)

Cloud Vendor Dashboard (Cloud vendor users and third-party users with an administrator role)

Dynamic Template Service Layer (DTSL)

Front-end of Template-as-a-Service (FTaaS)

Back-end of Template-as-a-Service (BTaaS)

Cloud Governing Services

Template Management Service

Cloud Subscription Service

Cloud Provisioning Service

Cloud Ecosystem Management

QoS, Monitoring, Metering, Billing

Cloud Brokers

Cloud Value-added Services

Main services: IaaS, PaaS, SaaS

Other Services, such as Business Process-as-a-Service

Virtualization Services

Figure 2. The Architecture of DCCSOA
service (IaaS, PaaS and SaaS), other service layers (e.g., Firmware-as-a-Service, Robot-as-a-Service) and in general *aaS. The classification of DTSL into BTaaS and FTaaS, makes a cloud architecture progressively deployable alongside existing cloud technologies without significant barriers or overhead because the BTaaS defines a dynamic layer which can be modified and customized by a cloud vendor. The BTaaS can be developed alongside of existing cloud service layers. The FTaaS forms the customer interface and subscriber audits.

A cloud vendor defines several different services on-demand at DTSL. Each defined service is a Template [22] which is integrated with one or multiple value-added cloud services. Cloud vendors can set up, configure and provide different templates to their customers based on different value-added service layers in a cloud computing system. As illustrated in Figure 3, a template at the back-end of DTSL is dynamic, and it interacts with one or multiple value-added cloud services.

![Figure 3. One snapshot of DTSL layer and connection to cloud value-added services](image)

A cloud vendor can define several templates at FTaaS where each template provides cloud services to end-users. The FTaaS allows different vendors to define the same template to their customers. This feature provides independent value-added service to customers who need data and applications migration from one cloud to another cloud. Cloud vendor are able to define their own *aaS with a BTaaS. The BTaaS differ from vendor to vendor and they provide a transfer from heterogeneous *aaS to general templates. For example in Figure 4, if two vendors (V1 and V2) provide different IaaSs (IaaS1 and IaaS2), each vendor can provide a template as IaaSx at FTaaS, BTaaS in V1 is different from BTaaS in V2. Both vendors should use his own BTaaS to configure the back-end of his IaaSx.

![Figure 4. An example of a template (IaaS, template) with two different back-ends for two different platforms](image)

The dynamic customization feature of the BTaaS layer enables a cloud vendors to customize their own services and it provides standard services through the templates.

A cloud vendor can edit a layer by adding, editing or removing a template as shown in Figure 5. In this figure, rows represent cloud-value added services (traditional services layers) are static, such as IaaS, PaaS and SaaS or other service layers, such as future services of *aaS. In this figure, columns represent templates and are dynamic that can be defined by a cloud vendor on-demand. Four templates are defined in Figure 5. For example, a user who has access to T1 can use SaaS layer, or a user has access to T3 can access to SaaS and PaaS layers.

![Figure 5. One snapshot of the dynamic BTaaS layer](image)

The templates can be implemented for any kind of cloud services and traditional services. For instance, a cloud vendor can define several services as a template, such as Business-Intelligence-as-a-Service (BIaaS) and IaaS. In this case, the figure 5 will be changed and rows represent IaaS and BIaaS layers, and columns can be defined by a vendor.

The number of columns is dynamic, and is defined by a vendor. Each column stands for a template. The vendor defines several templates which make use of resources in one or multiple layers in a cloud computing system. For example, in Figure 5, T1, T2, T3 and T4 are cloud templates (orange colors). T1 interacts with SaaS value-added service layer, T2 interacts with all value-added service layers in a cloud computing system, T3 interacts with two value-added service layers (SaaS and PaaS) and finally T4 interacts with two lower-level value-added service layers (PaaS and IaaS). The customer groups include end-users, developers and third-party users (with end-user, or developer role). They use Cloud Client Dashboard for interacting with FTaaS to use cloud resources. Each user has an option for working on several cloud value-added services simultaneously by interacting with a template. For instance, a developer who uses T4 template in Figure 5, can work on PaaS and IaaS simultaneously. The developer can work on IaaS to install a new application (App1) on the server and she has access to PaaS simultaneously for developing a Mashups application which is required App1.

Cloud Client Dashboard (CCD): This component provides an interface to a group of end-users, developer and third-party users. Although the third-party users are collaborating with a cloud vendor to develop or provide cloud services or applications, they may use resources as regular cloud users. Each user can subscribe to a template rather than a service in traditional cloud computing systems. The dashboard provides a list of templates that each group of users can subscribe for billing tools to provide billing on resource usage of a template, and monitoring tools to provide monitoring on all subscribed templates. The CCD interacts with FTaaS to provide cloud services based on defined templates.
Cloud Vendor Dashboard (CVD): The CVD component provides an interface to high-level users, such as system administrators and third-party users (with an administrator role). The CVD is isolated from regular users to provide a secure layer to cloud administrators who work on configuring, adding and editing cloud templates.

User Governing Services (UGS): This service provides control and configuration of the DTSL for Pricing, Billing and Subscription services for each defined template. This layer sits on the DTSL because this service requires a list of users who are subscribed to templates. This service also interacts with the Cloud Governing Services (CGS) to enable high-level users to configure and control a cloud ecosystem.

Cloud Governing Services (CGS): This service is accessible through Cloud Vendor Dashboard for administrative users. This service includes the following services: Template Management Service (TMS) which controls the DTSL to develop FTaaS and BTaaS. The TMS interacts with BTaaS and Cloud Value-Added Services layer to provide cloud services through a template. Cloud Subscription Service provides a management service for defining a different type of subscriptions as well as billing and pricing methods for each template. Cloud Provisioning Service provides a management service for resources and it provision elastic services based on Cloud Subscription Service. Cloud Ecosystem Management Service provides an integrated model of cloud interdependent components. Quality of Services (QoS) service provides a control management on overall performance of cloud services. Monitoring Service monitors cloud templates and the customer applications which run on the cloud. Metering and Billing Services provide a payment structure and access to one or multiple templates.

Virtualization Services (VS): This service layer provides a virtualization tools for storage, computing, and other resources. This service includes Dispatcher, Storage and Programming API Tools, and Virtual Machine (VM) Services, such as Virtual Machine Monitors.

B. Advantages of the Proposed Architecture

The DCCSOA has several advantages which are described as follows:

Customizable architecture: The dynamic component of the proposed architecture (DTSL) allows cloud vendors to modify and customize their cloud architecture on demand. This customization improves cloud architectural issues, such as lack of usability of cloud computing because a cloud vendor defines a new template that covers several services for enabling customers to have an integrated service. This offer will be more attractive for a variety group of users because a vendor is able to provide different customized services via different templates. For example, in traditional cloud computing systems, a telecom user who needs one or more network functions should find a cloud vendor who provides IaaS and subscribe to this service. However, a cloud vendor can define multiple services (e.g., a VPN service and a storage service) in a template for a group of users, such as the telecom user.

Flexibility and accessibility: The DTSL gives more flexibility and accessibility to customers through a template that provides several services at the back-end of templates (BTaaS). As a result, cloud vendors are able to offer different cloud templates to their customers. Each template could be an integration of one or more services. For example, in Figure 5, a cloud vendor provides four different templates, and cloud users who work on template T2 can interact with PaaS and SaaS layers simultaneously.

Dynamic Abstraction: The proposed architecture abstracts and encapsulates higher-level service layers from lower-level service layers by defining a template in DTSL that exposes lower-level services to advanced customers, and expose higher-level services to regular users or a customized services from both levels to a group of users. For example, in Figure 5, a vendor offers template T2 to advanced customers who need service in PaaS and IaaS service layers. The reason for this exposure is to improve flexibility and accessibility for some customers who need access to different and multiple services.

The DTSL facilitates the customers’ migration to the cloud and return back to the in-house IT department because a cloud vendor can provide a template at DTSL that has the similar features to in-house IT or other cloud vendors. For example, in Figure 5, customers who interact with T4 can access to IaaS to setup an operating system, and use a cloud platform simultaneously.

Portability of applications and data in cloud: The portability of both applications and data in cloud computing is another advantage of DTSL which is divided into FTaaS as front-end and BTaaS as back-end. As previously described, a lack of portability in cloud for both applications and data, that causes vendor lock-in issues, is a major issue in cloud computing systems. The DCCSOA enables different cloud vendors with heterogeneous infrastructures provide the similar FTaaS to their customers. The similar FTaaS allows customers to migrate data and applications to other vendors. The vendors can configure different BTaaS based on their specific infrastructures, such as hardware or platforms.

Cloud Vendor Devolution: Current existing cloud architectures do not support cloud vendor devolution that allows a partner of a cloud vendor to develop cloud configurations. The DCCSOA enables a cloud vendor to define a template, such as T2 in Figure 5. The cloud vendors can provide full access, and give a devolution role to their partners who uses a template (e.g., T3). The partners can provide new templates which are derived from the main templates (e.g., T4 from T3), to their customers. For security reasons, DTSL manager, DTSL monitor and security monitor control this group of users. The best advantage of this permission is that a cloud partner is able to develop the cloud computing system like a cloud vendor. For instance, a cloud vendor provides a PaaS as a template in DTSL to her partner. Cloud partners can offer a new service to their customers based on an integrated service of PaaS and other services.
**Security**: The DTSL divided into FTaaS and BTaaS. This segmentation improves cloud security because customers have access to the FTaaS services layer and this layer is isolated from other value-added cloud services. This isolation makes the DTSL more secure. In addition, any data security and privacy method, such as [24] that can be implemented as a template in the DTSL.

**Standardization**: One of the major issues in cloud computing is a lack of standardization because this problem causes vendor lock-in issue. However, DCCSOA provides a dynamic service layer (DTSL) to enable different vendors to offer the same front-end (FTaaS) service layer. When different cloud vendors provide the same FTaaS to their customers, the customers could transfer their data and applications to other vendors, or they can transfer data and applications to their private clouds through defining a similar FTaaS.

V. A FRAMEWORK FOR COMPARISON OF DCCSOA WITH RELATED WORK

Several parameters are important to evaluate software architecture, such as quality attributes [25] (i.e., modifiability and system independent). We consider the quality attribute to provide a framework to compare the proposed architecture against related works.

In Table 1, we present a comparison of the proposed architecture (DCCSOA) to existing architectures, methods, and cloud tools. In this table, ‘x’ denotes that the literature did not provide information related to a feature of their platform, or they did not consider the feature. We use the following features in our comparison: (i) customization and standardization with minimal modification to the architecture and services; (ii) the capability of supporting interoperability, and (iii) *aaS* support. In Table 1, each row represents a study or a product of a conceptual model, a cloud architecture, a cloud platform or a tool. Each column represents the following items: the level of customization that indicates the ease of customization with which vendors could customize their own architectures; the level of standardization indicates the level of modifications is needed to provide a standardized cloud computing system; and the last column represents *aaS* capability that shows which architecture, platform or tools could support *aaS* feature with ease.

Low level Customization indicates customization at the Implementation Level because each application or product requires to be modified. For example, MC provides a solution to modify each service to provide a customize cloud computing system. Medium Level indicates customization at the Conceptual and Architecture Level (undatable) because both architecture and the existing applications are required to be modified. For example, CRA provides a new architecture without adopting new features with the existing architecture. High Level indicates customization at the Architecture Level with adopting new features with the existing architecture. For example, CCIF provides adoptable services through a uniform cloud interface. This level requires minimal modifications to achieve customization with standard model. The solutions of interest are high level customization because they provide customized services with minimal modifications to the existing architectures (conceptual level) and existing services (implementation level). DCCSOA provides an independent service (TaaS) to provide customization on existing services. DCCSOA is not required to modify the existing architecture or the existing services to achieve customization with minimal modifications. DCCSOA is only required to modify and adopt the templates of each service.

Low Level standardization represents the maximal modifications to major cloud services to provide a standard service between different cloud computing systems. For example, all components are required to be modified in CCM to provide a standardized cloud computing system. High Level indicates standardization with less modifications. For example, DCCSOA provides a solution for standardization with minimal modifications to the existing architecture or the existing services.
through a uniform cloud interface. The solutions of interest are high level standardization because it does not require modifying the existing architecture or existing services to achieve standard cloud computing. **DCCSOA** provides different templates at the **FTaaS** to provide a uniform interface for different types of the existing services that cloud be implemented in different cloud vendor’s systems with different architectures.

**Low Level Interoperability** indicates interoperability via an external interface that causes high traffic connection to external entity without an ability to control or modify the interface. For example, **CCIF** provides an external uniform interface that is disabled independently of a service and each service is required to connect to the interface to provide interoperability feature. **Medium Level** indicates interoperability through implementation because each component does not rely on an external interface but the correspondence method requires to modify major services. For example, **CCM** provides interoperability through object-oriented paradigm that does not require to connecting to an external interface and in this method each object requires to be modified to achieve interoperability. **High level** indicates interoperability with independent services and minimal modifications. The solutions of interest are high level interoperability solution that minimizes the modifications of the architecture and services to achieve interoperability. **DCCSOA** provides an independent service which is not relied on external interface or required modification of service.

The last column shows the capability of *aaS*. Other related work (methods, platforms and architectures) did not consider this feature as a part of their proposed solution. **DCCSOA** allows a cloud vendor to define, deploy, customize and standardize new services via **FTaaS** and **BTaaS**. **DCCSOA** enables a cloud vendors to add new services as *aaS* by implement a heterogeneous service and adopt the service at the front-end layer (**FTaaS**) to provide a customizable and standardized service with minimal modifications and with ease. This comparison shows our proposed architecture (**DCCSOA**) allows vendors to define a dynamic, standardized and customizable cloud architecture with the capability of supporting interoperability and *aaS*. **DCCSOA** requires minimal modifications to the architecture and services with maximal the customization.

In addition to the framework in Table 1, we evaluate the proposed architecture based on **SOA** evaluation [25]. The evaluation is divided into the following topics: (1) **Target Platform**; (2) **Synchronous versus Asynchronous Services**; (3) **Granularity of services**; (4) **Exception Handling and Fault Recovery**; (5) **HTTPS or Message-Level Security**; (6) **XML optimization**; (7) **Use of a registry of services**; (8) **Legacy Systems Integration**; (9) **Service Orchestration**.

These major topics are divided into minor evaluation items as shown in Table 2. Icon “☺” in Table 2 shows the advantage of the selected parameter topic in **DCCSOA**. For instance, the proposed method in fine-grained services topic provides advantage in flexibility feature. More details about each parameter can be found in [25]. We consider the following general scenario to evaluate the proposed method:

Scenario **SC**: “User **U** uses a platforms as follow: **P** runs on the top of Cloud1 to provide service **S** and **U** is willing to transfer data and application to **P** which is running on the top of Cloud2 for the same service. When **U** needs to transfer data and applications from **P** to **P**, administrator of **P** needs to define the same service on **P**. Both platforms (**P** and **P**) are bound to the target cloud vendor services (**S** and **S**).” Evaluation results

We evaluate the proposed method against a popular cloud architecture, **CCOA** [16], based on **SC** scenario. Each item (parameter) in Table 2 gives one credit, if the scenario passes a given parameter. The evaluation result is shown in Figure 6. The result shows the proposed method provides flexibility with less modification over other existing methods.

![Figure 6. The evaluation results](image)

**VI. CONCLUSION AND FUTURE WORK**

In this paper, we proposed a new Dynamic Cloud Computing Service-Oriented Architecture (**DCCSOA**). The proposed architecture addresses the most existing cloud computing issues, such as data and applications migration between different clouds, transfer to cloud, or return back to in-house **IT**, data and applications lock-in issues, and a lack of standardization and customization. **DCCSOA** provides a dynamic and customizable service layer (**DTSL**). The **DTSL** provides simplicity these issues by defining a layer, **template**, with the same feature in **DTSL**. A **template** is divided into **front-end** (**FTaaS**) and **back-end** (**BTaaS**) layers. The defined **template** can be customized by a cloud vendor for different groups of users. **DCCSOA** also allows different cloud vendors to provide the similar cloud services through a **template** that meets a standardization between different cloud computing systems. We discussed how the proposed architecture supports existing and future services (in general *aaS*) by using the **DTSL** at **BTaaS** that can be configured to a specific cloud services. We evaluated the proposed method.
based on SOA evaluation. The result shows that the proposed architecture, DCCSOA, provides several advantages over existing cloud architectures and platforms, such as minimal modifications for providing standardization and customization. As a future work, we will investigate the quality of each attribute of DTSL, such as performance, reliability, scalability, security by running an evaluation method (i.e., scenario based). We will also consider heterogeneous cloud computing systems to implement DCCSOA with different templates in DTSL.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>DCCSOA</td>
<td>DCCSOA</td>
<td>DCCSOA</td>
<td>DCCSOA</td>
<td>DCCSOA</td>
<td>DCCSOA</td>
<td>DCCSOA</td>
<td>DCCSOA</td>
</tr>
</tbody>
</table>

Table 2. Evaluation parameters for each topic

Based on DCCSOA existing cloud architectures and platforms, such as minimal customization. As a future work, we will investigate the modifications for providing standardization and architecture, heterogeneous cloud computing systems to implement DCCSOA with different templates in DTSL.
Citation (To be appeared in):