JKAU: Comp. IT. Sci., **Vol**. **8** No. 1, pp: 59 – 69 (1440 A.H. / 2019 A.D.)
Doi: 10.4197/Comp. 8-1.6

Peer to Peer Cloud Providers Federation

Nourah Fahad Janbi

Dep. of Computer Science, Faculty of Computing and Information Technology, King Abdulaziz University, Jeddah, Saudi Arabia

Noorah.janbi@yahoo.com

Abstract. The increasing demand of the cloud services and with the emergence of many could service providers, the need for cloud federation is inevitable. In cloud federation, many could services providers are collaborating with each other to improve the resources usage, cost, quality of service they provide. To form this federation a management framework is required to facilitate the communication between these providers. This framework can be centralized or distributed, distributed Peer to Peer cloud federation improve extensibility, scalability and fault-tolerant. On the other hand, it is challenging in term of complexity, security and manageability of the federation. In this paper we propose a fully distributed P2P Cloud Federation (PPCF) architecture. PPCF provide a way to connect heterogenous cloud providers to share resources and improve the cloud elasticity. The architecture combines different software technologies to fulfil the cloud federation requirements.

Keywords: Cloud Computing, Cloud Service Providers, Federation, Peer-to-Peer, P2P.

1. Introduction

In cloud computing, computer resources (such as servers, storage and software) are offered as services on demand with pay-as-you-go pricing through Internet. Services are provided by Cloud Service Providers (CSP) on different levels Software as a Service(SaaS), Platform as a Service(PaaS) and/or Infrastructure as a Service(IaaS) depending on the customers' needs [1].

With the popularity of cloud computing and the increasing demand on CSPs, a new range of limitations and issues are raised [2]. Limitations such as the scalability of the provider, interoperability of data and degradation in quality of service(QoS) could be overcome with connecting multiple cloud providers [2-3]. In this way, the CSP can provide a better service to its customers and help other providers to serve their customers in case of idle resources are available.

Cloud federation is one of interconnected-clouds types where a group of clouds (private or public) are connected together to form a federation. In this federation, CSPs work as both providers and consumers [4]. That would maximize the CSPs resource usage, cost efficiency, quality of their services and help them to fulfil their customers Service Level Agreement (SLA) including unexpected or unpredicted demands.

To build and manage a cloud federation the following functional requirements must be considered [5]: (1) Members discovery, management, authentication, and authorization. (2) Resource discovery, selection, allocation, access and pricing. (3) Interoperability mechanism to connect and

exchange data between heterogenous systems. In addition to the functional requirements, the main non-functional requirements are reliability, flexibility, scalability and self-organization.

The cloud federation management architecture can be centralized or distributed (Peer to Peer- P2P). In centralized architecture there is a central manager/s that stores information about the providers and their resources. Usually this manager coordinates resource allocation among providers and perform accounting tasks [6]. Although having a central manager reduce the security risks and guarantee a fair resource allocation, it is like any other centralized system suffer from single points of failure and requires setting up a central server [6-7].

In contrast, in a fully distributed P2P cloud federation there are no central manager and CSPs must negotiate directly to get the required service. P2P federations improve extensibility, scalability and fault-tolerant of the cloud and are easier to deploy [7-8]. On the other hand, the P2P federation add a big challenge in term of complexity, security and manageability of the federation. For instance, discovering CDP member of the federation and their resources is a complex and risky operation as the trustworthiness of this member cannot be guaranteed. This mainly because there is no central registration point that members can refer to discover and authenticate the provider.

In addition, another issue that must be considered in designing P2P federations is the free riders [8]. As peers tend to be selfish by trying to maximize their profit without contributing in the federation. Therefore, there should be a mechanism to encourage contribution, impose fairness and isolate free riders.

In this paper we propose a fully distributed P2P Cloud Federation (PPCF) architecture. PPCF provide a way to connect heterogenous cloud providers to share resources and improve the cloud elasticity. The architecture combines different software technologies to fulfil the cloud federation requirements.

The rest of this paper is organized as follows: Section 2 discuses related works on could federations. Section 3 present our proposed architecture and in Section 4 we evaluate our design. Finally, conclusion is presented in Section 5.

2. Related Works

Many approaches and efforts have already been explored related to our proposed architecture. Each of them focused on different scenarios and have their strength and weaknesses. In this section we summarize some of these proposals and frameworks. Table1 shows a summary of the reviewed designs.

Authors in [9] developed a cloud interconnection agent for OpenStack clouds. They used the peer-to-peer agreements to support the flexibility of the distributed cloud. The agent works on layer 2 using IPsec tunnels to link two agents in different clouds. This is done manually by the administrator using the agent API. The could customers also has to use anther API to expand to one of the available clouds that are set by the administrator. Although their solution enhanced the security for the users, it did not consider the self-organization and dynamic that cloud federation require. Therefore, their secure connection should be automated and used as underlaying technique to connect peers.

BEACON [10] is open source cloud federation framework that suite federation architectures such as peer, hybrid, and

brokered federations. It focuses on intercloud networking and security issues, to support the automated deployment of applications and services. BEACON consist of three main components the Service Manager, the Cloud Manager and the Network Manager [11]. The Service Manager and Cloud Manager are responsible for the instantiation of the service and the placement of VMs into physical hosts. While the Network Manager is responsible for managing the federated cloud network operation and resources allocation. Alternatively, to [9], BEACON automated the deployment and configuration of the security Virtual Network Function (VNF) on Service Function Chaining (SFC). Which are used to enforce a global security policy that is defined in a single service manifest [12].

In [13] they proposed the use of blockchain infrastructure to implement a distributed, democratic and trustworthy governance approach in Federation-as-a-Service (FaaS) federations for SUNFISH platform. They focused mainly on the context of the public sector and its legal requirements. Blockchain provides great features related to the integrity, distribution and control of data that support the approach distribution and trustworthy. Their architecture (FaaS) federations consists of six main components for SUNFISH are the Federated Administration and Monitoring (FAM), the federated identity manager (IDM), the Registry Interface (RI), the Data Security (DS), the Intelligent Workload Manager (IWM), and the Federated Runtime Monitoring (FRM) [14].

FAM represents the logical entry-points for managing the FaaS federation and interacting with the SUNFISH platform. IDM provide authentication services to all the entities within a FaaS federation. IR manage the interaction (e.g. data store or retrieval) with the blockchain registry using crypto-tokens based authorization. DS uses Attribute-based Access Control (ABAC) to enforces the access control policies. IWM is a service broker, that optimize the workload upon service requests. FRM component provides a distributed infrastructure to monitor every access control request received. Although the use of blockchain enhances the security of the distributed system and provide the required democracy among providers using smart-contracts, it has its own drawbacks such as limited speed and computing resources. It also may show scalability issues.

In [15] they designed Fogbow, which is a new middleware to support large federations of (IaaS) cloud providers. Fogbow is designed to support federation of heterogenous clouds which require interoperability mechanism. In contrast to [9] work, the middleware implemented at a higher level at each federation member. This increases the flexibility of the federation as the communication between CSP is standardized at the middleware. Fogbow consists of two main components: The membership manager and the allocation manager. Membership manager keep track of active allocation manager using a gossip-based protocol, which keep their membership information updated by exchanging information periodically.

This makes Fogbow fully-decentralized system unlike BEACON which has a single service manifest. Plugins are included in the allocation manager architecture to provide a communication to various IaaS technologies, such as Identity, Compute, Storage, Network services. In their work they implemented interoperability plugins and behavioural plugins to customize the business logic. Additionally, Fogbow apply three level authentication and authorization at the federation layer, the local cloud layer, and among clouds.

Table 1. Summary of the reviewed works.

	Discovery	***Selection and allocation***	***Monitoring and Pricing***	***Authorization and Authentication***	***Fairness***
BEACON [10]	Members are set by admin	Location-aware elasticity rules and service placement policies	Stored at different level depending on the used subsystems	Global security policy using SFCs	N/A
SUNFISH [13]	Formal agreement SUNFISH Federation Agreement Contract (SFAC)	Service consumer choose one of the operation tenant offering the service	All monitoring information stored at blockchain-based registry	eIDAS – crypto-tokens -blockchain-based registry	Trustworthy repudiation system (only proposed)
Fogbow [15]	Gossip-style synchronization between allocation managers	First come first serve (FCFS) Asynchronous requests	Request data is stored at allocation manager and updated periodically	Three-steps procedure	Fairness-driven Network of Favors (FD-NoF) incentive mechanism

3. P2P Cloud Federation Platform

In this paper we propose a fully distributed P2P Cloud Federation (PPCF) platform architecture. PPCF provide a way to connect heterogenous cloud providers to share resources and improve the cloud elasticity. This section will present the PPCF architecture and discuss its design in detail.

A. System Architecture

P2P Cloud Federation architecture uses different software technologies to fulfil the system requirement. It mainly uses component-based software technology, but web service and software agent software technologies are also used in some areas. PPCF consists of four main components: Federation Web Service (FWS), Request Handler (RH), Distribution Manager (DM), and Native Cloud Adapter (NCA). Figure1 presents the software architecture of PPCF and the components that should be installed for each cloud provider participating in the cloud federation. Next section will discuss each component in detail.

B. Detailed Design

This section will discuss the function of each component of the PPCF platform architecture and the different architecture style it follows.

1- Federation Web Service (FWS)

The Federation web service component is a web-based service that provide an API for peers to receive resources request over the Internet. These requests are sent by Remote Allocation (RA) component at the requester cloud PPCF. When a resources request is received, it is passed to the RH to deal with the request. After that, the response will be returned to the remote peer depending on the action that the HR took. Remote requests can have three types of responses: (1) refused by the authentication and authorization component, (2) the resource allocation succeed and a connection is established between remote user and resources, or (3) the resource allocation failed.

2- Request Handler (RH)

Resource requests in cloud federation can be either fulfilled locally or remotely depending on the current status of the requester local cloud. The Request Handler is responsible for dealing with both local and remote resources requests. Local requests are sent by the cloud provider local users. While remote requests are sent by users of remote

cloud provider in the federation through the FWS. RH consists of three subcomponents Authentication and Authorization (AA), Load Balancing (LB), and Requests Monitoring (RM). First the request will be sent to the AA component to be authenticated and authorized. If it succeeds, it will be passed to the LB which is responsible for making the decision to either allocate the requested resource locally or remotely. Finally, after allocation, the request will be registered at the requests DB by the RM. Figure 2 shows how two clouds are communicating to submit a remote request. While Fig. 3 and Fig. 4 present the sequence diagrams for local and remote requests respectively.

3. Authentication and Authorization (AA)

When a user sends a request to his local PPCF, RH will use AA to authenticate and authorize the request for the local user using the native cloud mechanisms. If the user is an authenticated user and is authorized to acquire the requested resources, the request will be forwarded to LB. On the other hand, if the request is a remote request coming from FWS, the AA will first check if the cloud provider of the remote user is authenticated and authorized. If this is the case, the AA will map the credentials of the remote user to a credential that is used to access the underlying local cloud. Finally, the native cloud mechanisms will be invoked to check if the request can be processed at the local cloud.

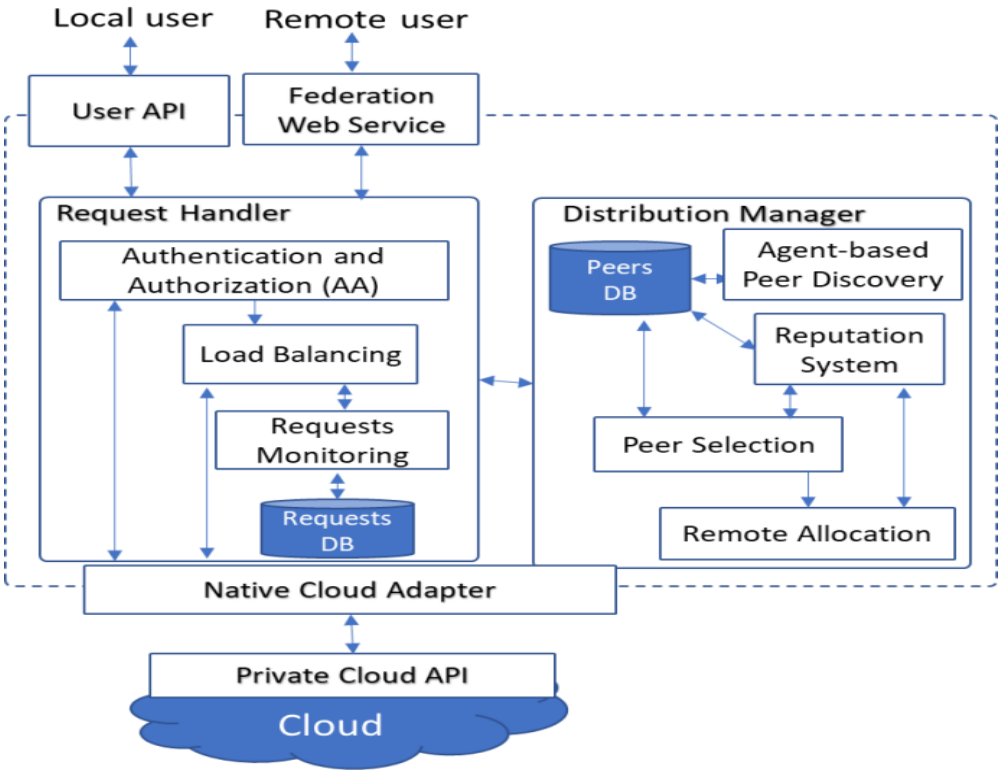

Fig. 1 Software Architecture of PPCF.

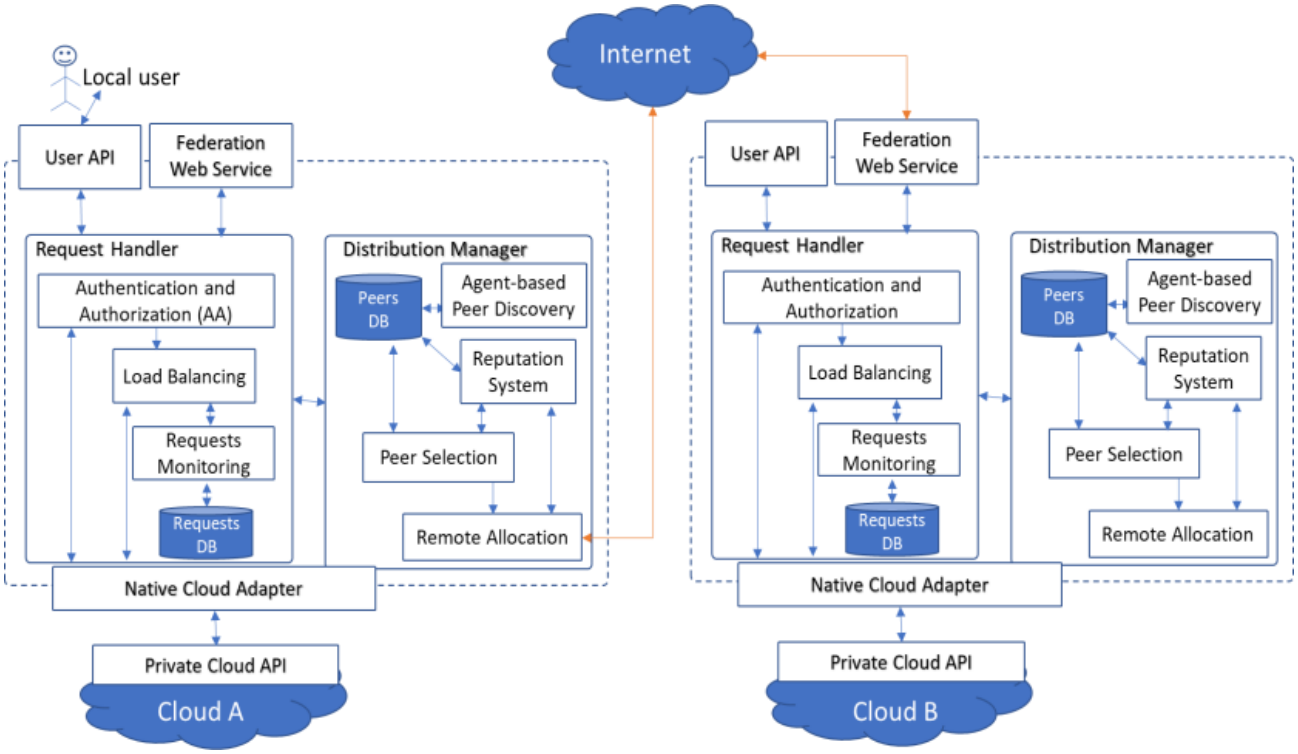

Fig. 2 Remote request between two PPCFs.

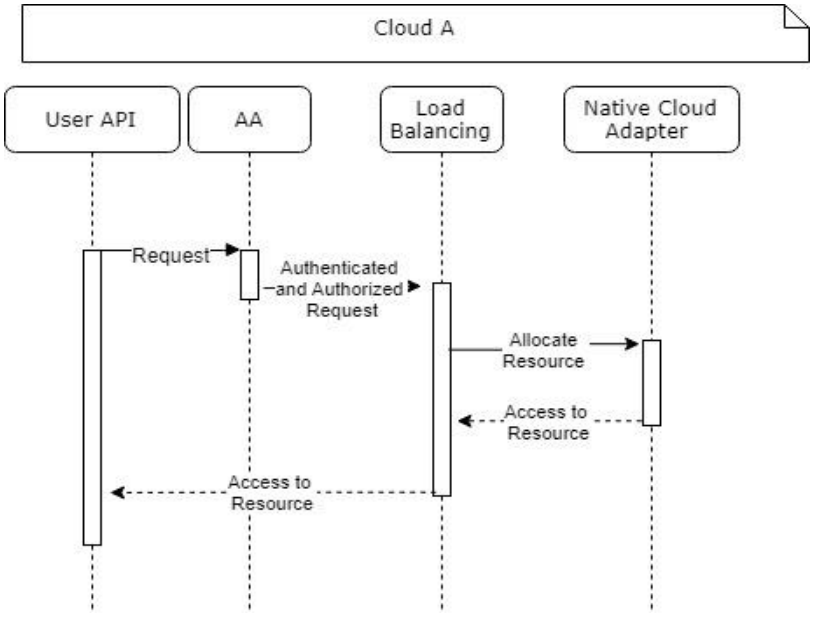

Fig. 3 Sequence diagram for local requests.

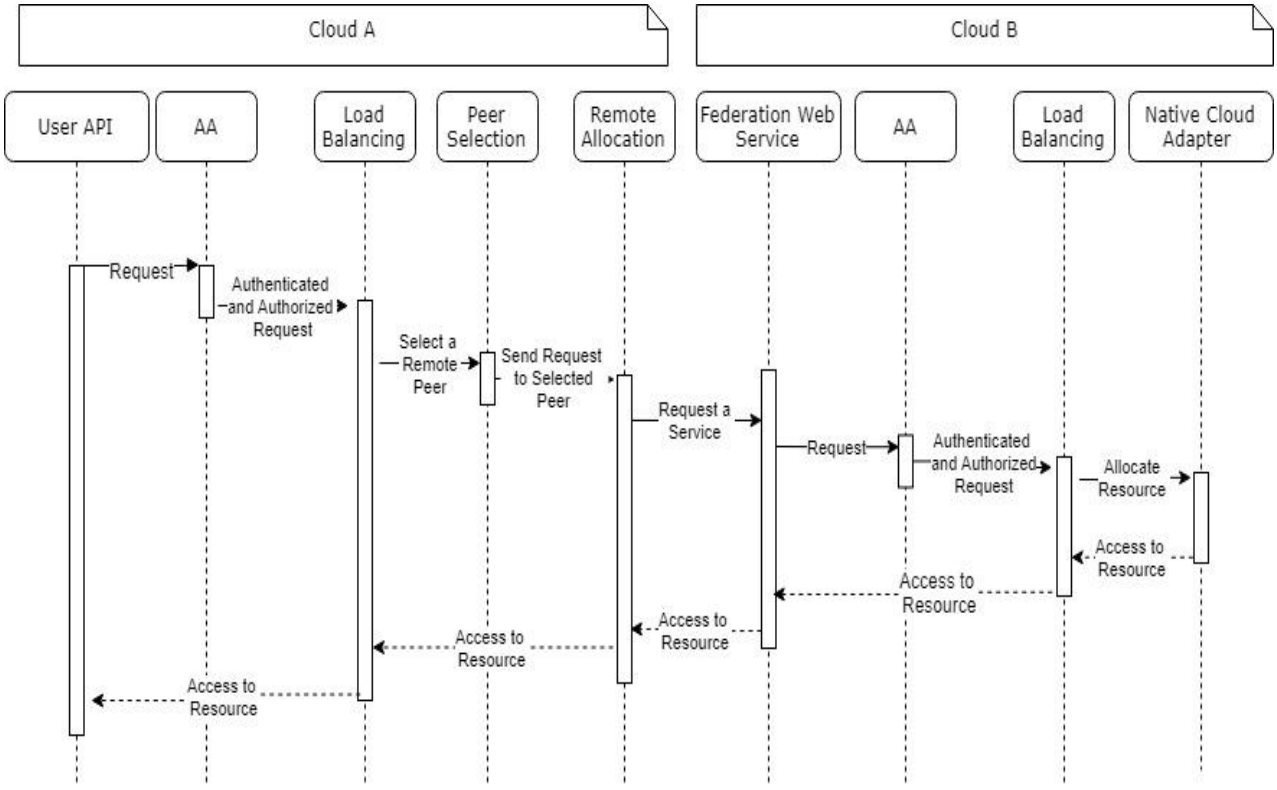

Fig. 4 Sequence diagram for remote requests.

4. Load Balancing (LB)

The Load Balancing component receives authenticated and authorized requests from AA and decide whether to allocate resources locally or remotely. The decision is based on the current state of the local cloud and different parameters on SLA of the user. LB retrieves the current state of the cloud using the NCA and take the decision. If LB decided to allocate resource locally the NCA will be used to communicate with the Private Cloud API to create the proper VM. Then the request will be registered at the Requests DB by RM. If the LB decided to allocate resource remotely the request will be sent to the DM to handle it. Algorithm 1 describes the resources allocation decision procedure.

Algorithm 1: Request Allocation Decision

```
requested ← Received resource request
available ← Get the local resource available
if available > requested then
    Allocate resource locally using the Native Cloud Adapter
else
    Pass the request to the Distribution manager to select a peer and allocate resources remotely
end if
Register requests on the DB using Request Monitoring Component
```

5. Requests Monitor (RM)

The Requests Monitor is the logical component that manage the storing and retrieval operations on the Requests DB. It is responsible for keeping track of all local and remote requests.

6. Distribution Manager (DM)

The Distribution Manager handles requests that LB decided to allocate remotely. DM has to select a peer to send the request to and set the connection between the user and the remote cloud. RH consists of four subcomponents: Reputation System (RS), Agent-based Peer Discovery (APD), Peer Selection (PS), and Remote allocation (RA).

7. Reputation System (RS)

The Reputation System is responsible for recording information about the past behavior of other peers. It also gathers information from other peers, which are then used to determine their reputation. This strategy is used to promote collaboration and discriminate free riders. There are many reputation systems proposed in the literature for P2P networks. Here we follow the Bayesian-based reputation system we proposed in [16] as is it uses a very simple basic calculation to identify good peers. This is important because the system cloud federation is a complex system and we need to minimize the overhead as much as possible in an efficient way. All Reputation values are stored with other Peers information at the Peers DB.

8. Agent-based Peer Discovery (APD)

Agent-based Peer Discovery is component that use mobile agent technology to discover other cloud providers in the federation and their available recourse and store them in the Peer DB. Mobile agent instances are sent to neighbor peers to collect information about available resources they have periodically. These data then are sent back to the home agent and stored in Peer DB. Fig. 5 shows the Peer discovery mobile agent which travels from cloud provider to anther to collect information about providers participating in the federation.

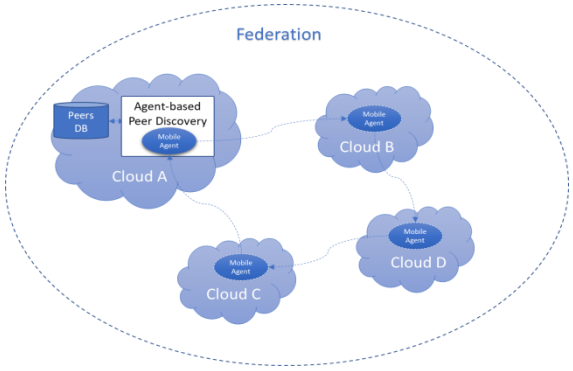

Fig. 5. Peer discovery mobile agent.

9. Peer Selection (PS)

Peer Selection is responsible for selecting the optimal cloud provider (peer) for the remote request. The optimization problem will take a decision depending on three parameters: Available resources, communication cost, and reputation of the peer. Peers available resources are collected using APD and stored on the Peer DB. Communication cost is calculated depending on the location, distance, and type of link between the current peer and destination peer. The reputation is also stored at Peer DB and managed by RS component. Using these parameters APS will choose the best peer to send the remote request to. Algorithm 2 describes the peer selection decision procedure. The selected peer information will be passed to RA to communicate with the chosen remote peer and actually allocate resources.

Algorithm 2: Peer Selection

```
1:  peerSet ← set of all known peers
2:  user ← user requested resource
3:  requested ← requested resource
4:  cw ← communication weight
5:  rw ← reputation weight
6:  candidatePeers ← new empty set
7:  for i=1 to peerSet.size()
8:      peer← peerSet.get(i)
9:      com← peer.getCommunicationCost(user)
10:     rep← peer.getReputation()
11:     if peer.getAvailableRecourse()> requested then
12:         score ←  cw * com + rw * rep
```

```
13:             peer.setScore(score)
14:             candidatePeers.addPeer(peer)
15:         end if
16:     end for
17:     selectedPeer← candidatePeers.getPeerWithHighestScore()
18:     return selectedPeer
```

10. Remote Allocation (RA)

To actually allocate the resource remotely, the RA component submit the request to FWS. After the recourse is allocated and setting up the needed infrastructural layer at the remote cloud provider, RA will send the request and allocation information to RM. In this way local RM will know where their users' remote requests are allocated.

11. Native Cloud Adapter (NCA)

Native Cloud Adapter is required so that PPCF can provide the interoperability requirement. As it is expected from PPCF to form a federation of heterogenous cloud providers platforms. Therefore, NCA works as a middleware between PPCF and the Private Cloud API that facilitate the communication between them.

4. Evaluation of PPCF Design

Cloud federation and combining heterogenous clouds is a complex process and with the absent of a central manager in the peer to peer environment the process is much complicated. Therefore, there was a need to use different architecture styles to fulfil the system requirements. Each architecture style has its own significant contribution for software development but mostly one style is not sufficient to design the whole aspects of system [17], especially when it is a complex and ramify like the one in our case. In our architecture design we adopted three technologies (software component, web service and mobile agent) to fulfil the reliability, flexibility, scalability and self-organization non-functional requirements of our system.

Dividing the complex system into multiple cohesive components that encapsulates a set of related functions and provided an interface to expose its services and hide its implementations. By decomposing the system into a set of independent components the complexity of the system is reduced. In our design we have two subsystems, Request Handler and Distribution Manager, each of them is further consists of a number of components. These two systems are designed in that way to separate the local and remote functionally. Distribution Manager is only used when a remote allocation is required, and it knows how to deals with different cloud providers and choose the best one.

Mobile Agent technology is used in peer to peer distributed networks, as the goal is to build a collaborative environment to facilitate resource sharing [18]. Resources need to be easily located and the discovery process should be done asynchronously without disturbing the system operation. Mobile agents provide asynchronous processing where agents are initialized once and then roam freely through the Internet to do their tasks [19]. In this way the system performance is going to improve, and communication overhead is going to reduce [17]. In addition, the self-organization and dynamism requirements will be met.

In order to satisfy a heterogeneous and loose-coupled software system, web service is used [20]. Furthermore, interoperability will be achieved as any client (in our case provider) can access other (providers) services regardless of their platform, technology, vendors, or language implementations [21]. Web Service in our system provides an interface that defines the data available and how it can be accessed by submitting requests to the federation web service.

5. Conclusion

Cloud federation allows many services providers to collaborate with each other to improve the resources usage, cost, quality of service they provide. Managing this federation and coordinate the communication of different provides is a complex task. The management framework can be centralized or distributed, distributed Peer to Peer cloud federation improve extensibility, scalability, fault-tolerant and overcome some of the centralized issues. On the other hand, it is itself has its own issues regarding complexity, security and manageability of the federation. In this paper we propose a fully distributed P2P Cloud Federation (PPCF) architecture that facilitate the communication of heterogenous cloud providers to share resources and improve the cloud elasticity. The architecture combines different software technologies (software component, web service and mobile agent) to fulfil the cloud federation requirements.

References

[1] **Villegas, D., Bobroff, N., Rodero, I., Delgado, J., Liu, Y., Devarakonda, A., Fong, L., Sadjadi, S.M.** and **Parashar**, **M.,** 2012. Cloud federation in a layered service model. *Journal of Computer and System Sciences*, **78**(5):1330-1344.

[2] **Assis, M.R.** and **Bittencourt, L.F.,** 2016. A survey on cloud federation architectures: identifying functional and non-functional properties. *Journal of Network and Computer Applications*, **72**:51-71

[3] **Shu, J., Liang, C., Wang, B.** and **Xu, J.,** 2018, March. Building the federation of cloud service for big data. In Big Data Analysis (ICBDA), *2018 IEEE 3rd International Conference on IEEE* (pp. 166-169)..

[4] **Liaqat, M., Chang, V., Gani, A., Ab Hamid, S.H., Toseef, M., Shoaib, U.** and **Ali, R.L.**, 2017. Federated cloud resource management: Review and discussion. J*ournal of Network and Computer Applications*, **77**: 87-105.

[5] **Lee, C.A.,** 2016. Cloud federation management and beyond: Requirements, relevant standards, and gaps. *IEEE Cloud Computing*, (1), pp.42-49.

[6] **Xhagjika, V., Navarro, L.** and **Vlassov, V.**, 2015, November. Enhancing real-time applications by means of multi-tier cloud federations. In *2015 IEEE 7th International Conference on Cloud Computing Technology and Science (CloudCom) IEEE*. (pp. 397-404).

[7] **Ranjan, R.** and **Buyya, R.**, 2010. Decentralized overlay for federation of enterprise clouds. In *Handbook of Research on Scalable Computing Technologies* (pp. 191-217). IGI Global.

[8] **Falcão, E., Brasileiro, F., Brito, A.** and **Vivas, J.L.**, 2016. Enhancing fairness in P2P cloud federations. *Computers & Electrical Engineering*, **56**:884-897.

[9] **Kimmerlin, M., Hasselmeyer, P., Heikkilä, S., Plauth, M., Parol, P.** and **Sarolahti, P.,** 2017, June. Network expansion in OpenStack cloud federations. In *Networks and Communications (EuCNC), 2017 European Conference on IEEE* (pp. 1-5)..

[10] **Huedo, E., Montero, R.S., Moreno, R., Llorente, I.M., Levin, A.** and **Massonet, P.,** 2017. Interoperable federated cloud networking. *IEEE Internet Computing*, **21**(5):54-59.

[11] **Montero, R.S., Massonet, P., Villari, M., Merlino, G., Celesti, A., Levin, A., Schour, L., Vázquez, C., Melis, J., Spahr, S.** and **Whigham, D.**, 2016, April. BEACON: A Cloud Network Federation Framework. In *Advances in Service-Oriented and Cloud Computing: Workshops of ESOCC 2015, Taormina, Italy, September 15-17, 2015, Revised Selected Papers* (Vol. 567, p. 325). Springer

[12] **Massonet, P., Dupont, S., Michot, A., Levin, A.** and **Villari, M.**, 2016, October. Enforcement of global security policies in federated cloud networks with virtual network functions. In *Network Computing and Applications (NCA), 2016 IEEE 15th International Symposium on IEEE* (pp. 81-84)..

[13] **Margheri, A., Ferdous, M.S., Yang, M.** and **Sassone, V.**, 2017, June. A distributed infrastructure for democratic cloud federations. In *Cloud Computing (CLOUD), 2017 IEEE 10th International Conference on IEEE* (pp. 688-691)..

[14] **Schiavo, F.P., Sassone, V., Nicoletti, L.** and **Margheri, A.,** 2016. *Faas: Federation-as-a-service.* arXiv preprint arXiv:1612.03937.

[15] **Brasileiro, F., Silva, G., Araújo, F., Nóbrega, M., Silva, I**. and **Rocha, G.,** 2016, May. Fogbow: A middleware for the federation of iaas clouds. In Cluster, Cloud and Grid Computing (CCGrid), *2016 16th IEEE/ACM International Symposium on IEEE* (pp. 531-534).

[16] **Janbi, N.F.** and **Radenkovic, M.**, 2017, July. An enhanced Bayesian-based reputation system for P2P file sharing. In *Computing Conference, 2017* (pp. 1247-1252). IEEE.

[17] **Jeware, A.** and **Dino, N.**, 2013. Hybrid Software Architecture Design Pattern Model. HiLCoE Journal of Computer Science and Technology, p.21.

[18] **Lopes, A.L.** and **Botelho, L.M.,** 2012. Efficient algorithms for agent-based semantic resource discovery. In *Agents and Peer-to-Peer Computing* (pp. 71-82). Springer, Berlin, Heidelberg

[19] **Braun, P.** and **Rossak, W.R.**, 2005. *Mobile agents: Basic concepts, mobility models, and the tracy toolkit*. Elsevier.

[20] **Aboud, N.A., Cariou, E., Gouardères, E.** and **Aniorté, P.,** 2011, July. Service-oriented Integration of Component and Agent Models. In *ICSOFT* (**1**) (pp. 327-336).

[21] **Qian, K., Fu, X., Tao, L.** and **Xu, C.W.,** 2010. *Software architecture and design illuminated.* Jones & Bartlett Learning.

اتحاد مقدمي الخدمة السحابية بطريقة الند للند

نوره فهد عبدالعزيز جنبي

قسم علم الحاسبات، كلية الحاسبات وتقنية المعلومات، جامعة الملك عبدالعزيز، جدة، المملكة العربية السعودية

noorah.janbi@yahoo.com

المستخلص. الحاجة إلى الاتحاد السحابي حتمية مع وجود الطلب المتزايد على الخدمات السحابية، ومع ظهور العديد من مقدمي الخدمات. في الاتحاد السحابي، يمكن للعديد من مقدمي الخدمات التعاون مع بعضهم البعض لتحسين استخدام الموارد والتكلفة وجودة الخدمة التي يقدمونها. لتشكيل هذا الاتحاد، يلزم وجود منصة إدارة لتسهيل الاتصال بين هؤلاء المزودين. هذه المنصة يمكن أن تكون مركزية أو موزعة. توزيع الاتحاد على طريقة الند للند تعمل على زيادة القابلية للتوسعة والتحمل. ومن ناحية أخرى، فإنه من الصعب تصميم هذا النوع من الاتحاد من حيث التعقيد والأمن وسهولة الإدارة. في هذه الورقة نقترح بنية اتحاد تعمل بطريقة الند للند لتكوين اتحاد سحابات موزعة بالكامل يدعى (PPCF). يوفر PPCF وسيلة لربط مقدمي السحابية غير المتجانسة لمشاركة الموارد وتحسين مرونة السحابة. يجمع التصميم المقترح بين تقنيات البرامج المختلفة لتلبية متطلبات اتحاد السحابات.

الكلمات المفتاحية: الحوسبة السحابية، مقدمو الخدمات السحابية، الاتحاد، الند للند، P2P.