Taiwan UniCloud: A Cloud Testbed with Collaborative Cloud Services

Wu-Chun Chung[*], Po-Chi Shih[*], Kuan-Chou Lai[¥], Kuan-Ching Li[§],
Che-Rung Lee[‡], Jerry Chou[‡], Ching-Hsien Hsu[£] and Yeh-Ching Chung[‡]

[*‡]Department of Computer Science, National Tsing Hua University, Hsinchu 300, Taiwan.
[¥]Department of Computer and Information Science, National Taichung University, Taichung 403, Taiwan.
[§]Department of Computer Science and Information Engineering, Providence University, Taichung 433, Taiwan.
[£]Department of Computer Science and Information Engineering, Chung Hua University, Hsinchu 300, Taiwan.
Email: {[*]wcchung, [*]shedoh}@sslab.cs.nthu.edu.tw, [¥]kclai@mail.ntcu.edu.tw, [§]kuancli@gm.pu.edu.tw,
[£]chh@chu.edu.tw, {[‡]cherung, [‡]jchou, [‡]ychung}@cs.nthu.edu.tw

***Abstract*—This paper introduces a prototype of Taiwan UniCloud, a community-driven hybrid cloud platform for academics in Taiwan. The goal is to leverage resources in multiple clouds among different organizations. Each self-managing cloud can join the UniCloud platform to share its resources and simultaneously benefit from other clouds with scale-out capabilities. Accordingly, resources are elastic and sharable with each other such as to afford unexpected resource demands to each cloud. The proposed platform provides a web portal to operate each cloud via a uniform user interface. The construction of virtual clusters with multi-core VMs is supplied for parallel and distributed processing models. An object-based storage system is also delivered to federate different storage providers. This paper not only presents the architectural design of Taiwan UniCloud, but also evaluates the performance to demonstrate the possibility of current implementation. Experimental results show the feasibility of the proposed platform as well as the benefit from the cloud federation.**

Keywords: community cloud; virtual cluster; vm migration; federated sotrage; portal

I. INTRODUCTION

Cloud computing is an emerging topic recently. The core concept of cloud computing is to provision the resources of computer software and hardware as services. Users can access a variety of virtualized resources and services by various client devices anytime and anywhere. Benefiting from mature open-source cloud software, many organizations or campuses are able to build their clouds on premises. However, when lots of resource demands come together or a service request with a large resource requirement is arrived at a cloud system, it may lead the cloud in unexpected overloading or oversubscription situations. Consequently, single cloud could not support sufficient resources due to its physical hardware limit. The SLA (Service Level Agreement) would be violated [14, 15] while the cloud is neither scale-up with the capacity of available resources nor scale-out with the capability of federated clouds. To alleviate the predicament, Taiwan UniCloud (University Cloud) is proposed to leverage resources of different single clouds in Taiwan to overcome the sudden overloading of each cloud.

Taiwan UniCloud is a community-driven hybrid cloud platform for academia to support cloud education, research, and application development. The self-managing cloud in each campus can join the UniCloud to contribute its cloud resources and benefit the distributed sharing resources from other participants. However, there are some challenges, including:

- Different clouds may have various user interfaces, which results in a user having to be familiar with varied dashboard operations.
- Each cloud may have different system calls or access APIs, which causes many efforts to leverage the existence of multiple clouds.
- The inter-cloud cooperation is not mature in open-source software, which makes a single cloud platform hard to supply the collaborative services among multiple clouds.
- The resource information is usually maintained by each self-managing cloud, which needs extra efforts to retrieve the monitoring information from multiple clouds.
- The live migration of a virtual machine (VM) is sometimes necessary across clouds. The big challenge is to online migrate a VM across clouds while keeping the service hosted by the VM is still available.

Our contribution involves tackling the above issues while developing the Taiwan UniCloud platform. The current prototype enables the collaborative cloud services with the development of several major components: web portal, federated computation, and federated storage. In addition, based on the resource monitoring information, our platform could further supply SLA-based resource provisioning. This paper not only presents the architecture of Taiwan UniCloud, but also evaluates the performance to demonstrate the feasibility of our current implementation.

In the current implementation, our platform could leverage the OpenStack-based campus clouds and the public Amazon clouds. Users could easily operate multiple cloud systems on a single portal and browse the resource status of different clouds. Users may further demand their needs through the UniCloud portal or deploy their cloud applications among multiple clouds. Experimental results demonstrate the feasibility of the proposed platform as well as the benefit from the cloud federation.

978-1-4799-3766-0/14 $31.00 © 2014 IEEE
DOI 10.1109/IC2E.2014.28

IEEE computer society

The rest of this paper is organized as follows. Section II presents the related work and distinguishes our work from existing research. Section III introduces the architecture of proposed cloud environment. Section IV shows the feasibility and experimental results. The conclusions and future work are finally given in Section V.

II. Related Work

Cloud providers such as Amazon [2], Google [4, 5], Microsoft [13], and Salesforce [12] have built up the public clouds for hosting a variety of cloud services and applications. However, these cloud systems are mainly established for the commercial usage with high prices. In addition, it is difficult to clone these clouds on our on-premised server farm for testing, trial run or even academic education and research. With the rapid development of open-source cloud platforms, some open and free solutions could help us to build our clouds. A comparative study of current open-source cloud platforms could be referred to [16, 19, 20].

The previous work [14] presents the InterCloud for the producer-consumer-based cloud federation. In the InterCloud, the Cloud Exchange acts as a market maker between the service consumers represented by users and the service providers represented by Cloud Coordinators. The goal is to supply the dynamic expansion or contraction of resource capabilities to handle the sudden variations of demands. An extended work [15] of the InterCloud is proposed to detail the design of a Cloud Coordinator. These works conduct a simulation-based experiment using the CloudSim and a small-scale evaluation with the Eucalyptus-based cluster to demonstrate the feasibility of cloud federation. However, their works neither focus on constructing a cross-cloud virtual cluster nor a federated storage system across multiple clouds.

Our work not only pays attentions on both federated computing resources and storage ones but also implements a preliminary prototype across clouds in different geographic locations. The Taiwan UniCloud adopts the viewpoint on top of multiple self-managing university clouds to leverage cloud resources. Moreover, we enable the cross-cloud virtual cluster with a virtual network over Internet, and the VM live migration across clouds. A federated storage system across different storage providers is also supplied in the current implementation. To realize our design, we adopt the OpenStack [9] as the basis of our platform. OpenStack could support high compatible cloud services with the public Amazon cloud in recent releases. Therefore, some preprocesses to realize the community-driven hybrid cloud could be simplified. The current prototype of Taiwan UniCloud can also provision cloud resources from different campus clouds and the public Amazon EC2 [1].

III. Architecture and Implementation

This section describes the proposed framework and presents the implementation details of our current prototype.

A. System Overview

As shown in Fig. 1, the current prototype focuses on the following issues: web portal, federated compute, federated storage, meta-cloud interface, and SLA-based resource provisioning. A uniform UniCloud Portal is developed to leverage and manipulate different clouds. The federated compute addresses the distributed computing services such as to support the virtual cluster computing for parallel programming. On the other hand, the federated storage focuses on leveraging storage resources across multiple storage providers to supply an object-based storage service. We further conduct the benchmarking on our platform and try to model the performance assurance for the fulfillment of SLA-based resource provisioning.

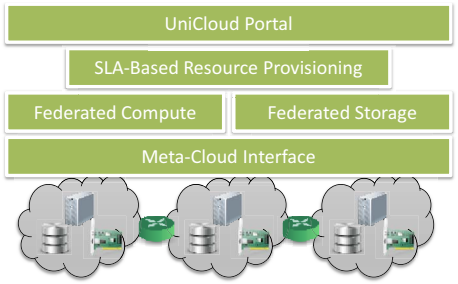

Fig. 1. Framework of current prototype

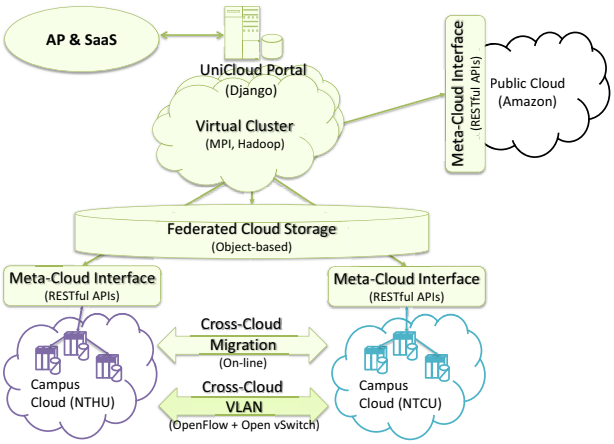

Fig. 2. Overview of the architectural design

Fig. 2 depicts an overview of our current implementation. A UniCloud portal is developed using the web framework. The restful APIs are adopted as the meta-cloud interface to facilitate the usage of cloud resources residing in different organizations or campuses. With the meta-cloud interface, our platform is able to supply the virtual clusters in which virtual resources may be obtained from multiple clouds. The virtual cluster is built upon a cross-cloud VLAN (Virtual Local Area Network) to interconnect with each virtual resource. The VLAN is constructed by exploiting the OpenStack quantum plugin and the Open vSwitch to manage the OpenFlow rules. Besides, the online VM migration across different clouds also adopts the GRE (Generic Routing Encapsulation) tunnel to keep the running service within a virtual cluster.

B. UniCloud Portal

The UniCloud portal aims at leveraging multiple geographic clouds. The major design consideration is to provide a uniform and simplified user interface. In the current implementation, the portal supports the manipulation of

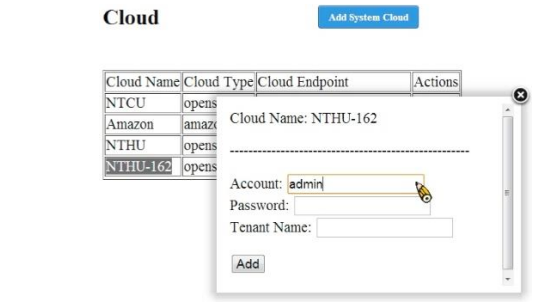

Fig. 3. Access authorization for multiple clouds

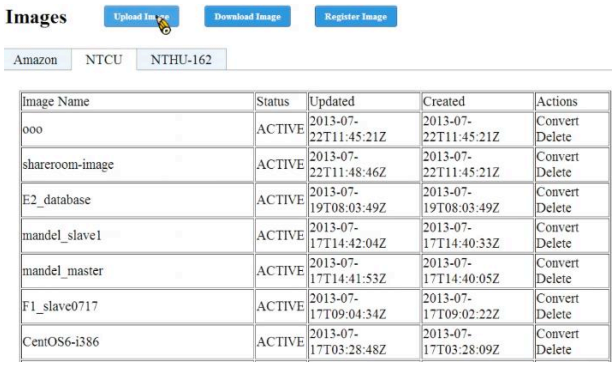

Fig. 4. Image management among multiple clouds

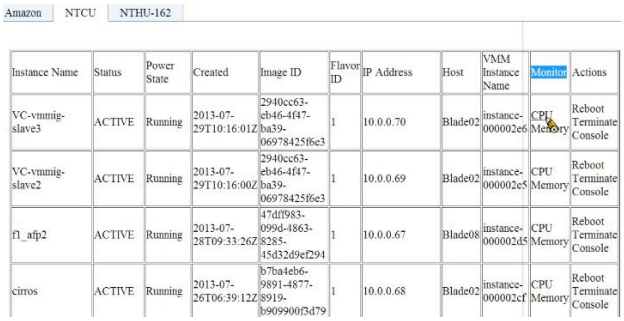

Fig. 5. Resource information about VM instances

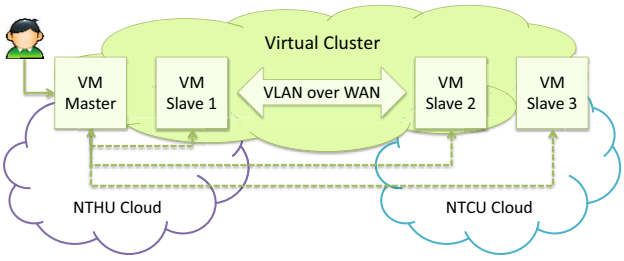

Fig. 6. Illustration of a cross-cloud virtual cluster

OpenStack clouds and the VM operations of Amazon EC2. Fig. 3 presents that users can register their authorized information and provide the corresponding endpoints for each candidate cloud. By using the restful APIs, the authenticated users could create VMs or use those cloud resources they leased from multiple clouds. These manipulations are all accomplished on the same UniCloud portal.

For most of users, they can install necessary packages after launching VMs. Users may also create VMs with their own images for the sake of deploying the prepared cloud applications. Accordingly, our UniCloud portal allows users to upload the compatible and customized VM images, as shown in Fig. 4. The portal can register these VM images to all candidate clouds for future VM launch.

Moreover, the Ganglia monitoring system [17] is adopted to monitor the resource utilization of the UniCloud platform. The portal provides a visualized GUI (Graphical User Interface) to report the status of cloud resources. Users can browse the static or dynamic information about their leased resources. Fig. 5 reveals the monitoring information about VMs hosted by Amazon EC2 and two campus clouds. In our current implementation, not only the resource information about virtual machines, but also the status of physical resources are all aggregated on the same UniCloud portal.

C. Federated Compute

The federated compute aims at supplying the construction of parallel and distributed computing environments across multiple clouds. The construction of a cross-cloud virtual cluster is presented first, following by the key techniques to enable the cross-cloud collaborative computing: the VLAN over WAN and the cross-cloud live migration.

1) Cross-Cloud Virtual Cluster

A virtual cluster is composed by a set of VMs which may be supplied from a single cloud or multiple clouds. Fig. 6 illustrates an environment of a virtual cluster across two campus clouds, in which two VMs are launched in each cloud. One of the VMs serves as a master of the virtual cluster and the others are slaves. The master is randomly chosen and associated with a public IP so that users outside the cloud can connect to the virtual cluster. The inter-connections among VMs within a virtual cluster are constructed with a cross-cloud network to form a VLAN.

To easy the construction of a virtual cluster, the one-click provisioning is supplied on the web portal. All necessary software packages are automatically installed via the post-script functionality supplied by OpenStack. The default packages in a virtual cluster include NIS, NFS, and SSH keys. In addition, the Torque is also adopted for the job scheduling and allocation. Fig. 7 depicts a workflow to construct a typical cluster computing environment. While launching a virtual cluster, a VM is configured as a master or a slave according to the pre-defined virtual cluster configuration. During the construction, the master collects the information of all slaves such as to update the mapping between IP and hostname of slaves and exchange the SSH keys. After the successful installation, users can login to the master via the associated public IP and login to other slaves from the master without typing passwords. The virtual cluster would not be created when any one step in the process fails. To make the system be more reliable, the runtime failures would be handled in the future work.

Users can request any number of VMs to form a virtual cluster. How to make an optimal decision for VM deployment

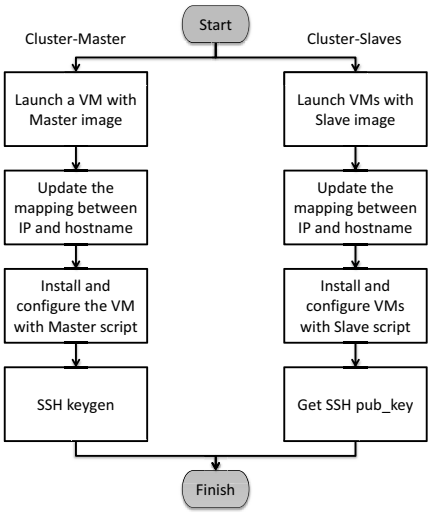

Fig. 7. Workflow of constructing a typical cluster environment

on multiple clouds is an interesting issue, but beyond the scope of this paper. Instead, we focus on the fundamental features to enable the construction of a cross-cloud virtual cluster. In the current implementation, all VMs in a virtual cluster are equally distributed to multiple clouds. VM images and installation scripts for both master and slave are reconfigurable. As a result, the MPI or Hadoop packages could be installed to provision a parallel and distributed computing environment.

2) VLAN over WAN

The VLAN over WAN mechanism highly depends on the network configuration of each OpenStack cloud. In OpenStack Grizzly, a recommended network configuration is to use the quantum plugin to manage the network and the GRE tunnel to interconnect all the nodes, includes both network node and compute node [10]. Fig. 8 depicts a sample network architecture across two OpenStack clouds. Each node has two Open vSwitches: br-int and br-tun. The br-int is used for the intra-node communication and the br-tun is used for the inter-node communication. The network node is equipped with an additional br-ex vSwitch for the communication in/out the cloud. Note that the name of a vSwitch is changeable and therefore could be distinct in different clouds.

The VLAN inside an OpenStack cloud is accomplished using the 802.1Q VLAN tag (*vid* in short) accompanied with the GRE tunnel ID (*tid* in short). Each VM is connected to the br-int of a compute node with a *vid* via the corresponding Open vSwitch port. For VMs residing in the same VLAN, if they are hosted by the same compute node, each VM can communicate with others through the same *vid*. If they are hosted by different compute nodes, the *tid* is used as a unique identifier of VLAN across multiple nodes. That is, when a packet is sent from br-int to br-tun, the *vid* is converted to a *tid* according to which VLAN it belongs to, and will be converted back (from *tid* to *vid*) in reversed course. In short, VLAN in OpenStack cloud is uniquely identified by the *tid*, which may be mapped to different *vids* in different nodes. This mechanism enables a cross-node VLAN and allows the flexible assignment of *vid* in different nodes.

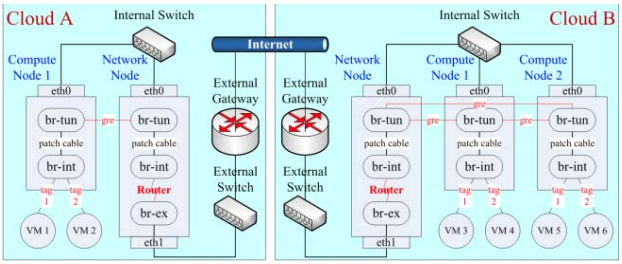

Fig. 8. Sample of the network architecture across two OpenStack clouds with the Open vSwitch plugin

Table 1: Example of the mapping table for VLAN ID using the dynamic VLAN mapping mechanism

	Cloud A	Cloud B	Cloud C
Federated VLAN 1	*tid*=1	*tid*=2	*tid*=1
Federated VLAN 2	*tid*=2	*tid*=1	*tid*=3
Federated VLAN 3	*tid*=4	*tid*=5	

To enable the cross-cloud VLAN, the UniCloud should consider four issues:

- How to form the federated VLAN?
- How to identify the VLAN information?
- How to comply with OpenStack networking?
- How to connect the subnet of each VLAN?

To solve the first issue, a dynamic VLAN mapping mechanism is proposed. The mechanism allows a dynamic connection of VLANs in different clouds to form a federated VLAN. Once a VLAN of each single cloud is chosen, its *tid* is recorded in the mapping table as shown in Table 1. The proposed mechanism has some benefits. First, the assignment of *tid* to the VLAN would not destroy the original design of OpenStack. Second, the mechanism is applicable to both existing and new creating VLANs, even for a new joined cloud. Third, not all clouds have to participate in the federated VLAN, e.g., the VLAN 3 in Table 1, which presents a more flexibility for the constitution of a federated VLAN.

For the second issue, the cross-cloud GRE tunnel is established pairwisely between the br-tun of a network node in each cloud. Since all the cross-cloud packets pass through the network node, the br-tun preserves the *tid* information to identify the source VLAN of a packet. The sender-initiate approach is used to correct the mapping of *tid* according to the VLAN ID mapping table. That is, a cloud that sends the packet to another cloud is responsible for correcting the *tid* before propagating the packet through br-tun. This procedure can be automatically configured by setting openflow rules in the br-tun vSwitch, as described next following.

The third issue is to deal with the openflow rules in an OpenStack cloud. The OpenStack has some protection schemes to prevent the broadcast storm and to protect the network by dropping illegal packets, such as the packets coming with unregistered *vid*, *tid*, or MAC address. However, with the default rules, multicast packets will not be forwarded to the cross-cloud GRE tunnel. To solve this issue, we aim at forwarding the packet via the federated VLAN while not compromising the original networking schemes used in OpenStack.

Therefore, two openflow rules are proposed as add-ons to the original rules:

- For all packets coming from another cloud (this can be determined by the used port of Open vSwitch), one rule directly forwards them to all br-tun switches of compute nodes in this cloud.
- For all incoming packets with the *tid* registered in a federated VLAN, one rule converts the *tid* according to the VLAN ID mapping table and sends it to the corresponding cloud.

Note that these add-on rules do not violate the design principle of OpenStack networking. First, the broadcast storm will not appear because all multicast packets coming from other cloud are confined to the local cloud only. The original rules in OpenStack takes over the prevention of the broadcast storm in a local cloud. Second, only the packets with *vid* registered in a federated VLAN will be forwarded to other clouds, so that other illegal usage of the network is still protected.

Regarding the fourth issue, all VMs in the same VLAN should be put into the same subnet so that VMs can communicate with each other using the assigned private IP, even those VMs are hosted by multiple clouds. A feasible network configuration is proposed: (1) all the federated VLANs are assigned within the same private IP subnet, e.g., 192.168.5.0/24; (2) in order to avoid the IP conflict, the gateway IP address and the DHCP range of each VLAN in the same federation are exclusive, e.g., one VLAN is assigned with the IP range from 192.168.5.1 to 192.168.5.50 while the other one is assigned from 192.168.5.51 to 192.168.5.100. In the current prototype system, as a proof of concept, the assignments of a subnet and the IP ranges are pre-configured. Nevertheless, sophisticated policies or rules can be easily developed to automate the decision making.

3) Cross-Cloud Live Migration

The goal of cross-cloud live migration is to migrate a VM residing in the same federated VLAN from one cloud to another one. Meanwhile, all the network state are preserved during the migration process, i.e., all original TCP connections will still be alive before and after the migration process. This goal is difficult for the cross-cloud paradigm due to two essential issues: the functionality issue and the IPv4 mobility issue.

Regarding the first issue, none of open-source cloud platforms inherently provides the live migration feature across two clouds. Therefore, as a proof of concept, we exploit the libvirt API to perform the cross-cloud live migration on OpenStack VMs. Before presenting the solution, several basic requirements for the live migration are highlighted in our current prototype: 1) all clouds adopt the hypervisor with the same version; 2) all clouds use similar or compatible hardware in the hypervisor point of view; 3) each cloud has a shared file system (like NFS) to store all VM instances for the candidate compute nodes.

The detail procedure of a cross-cloud live migration is given as follow:

- Retrieve the domain information of a candidate VM.
- Configure the shared instance directory for the candidate VM in both source and destination clouds.

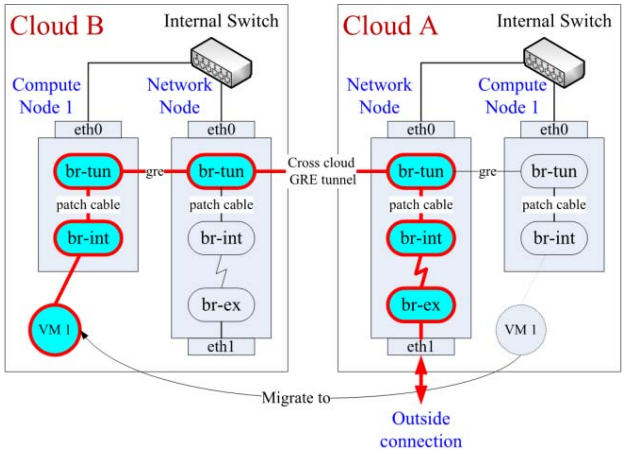

Fig. 9. Packet flow after the VM migration

- Configure DNAT to expose the TCP ports used in live migration if a compute node is behind the NAT.
- Configure openflow rules on the destination compute node (in br-int) to accept the MAC address of the VM.
- Call libvirt APIs to perform the cross-cloud live migration.
- Correct the *vid* of the VM according to which VLAN it belongs to in the destination compute node.

The second issue is that different clouds own a different range of public IP addresses. Once a VM is migrated from one cloud to another cloud, the public IP address associated for the connection to the Internet should be varied. Consequently, the original TCP connection will be broken while the public IP is changed after the migration. To solve this issue, we apply the GRE tunnel and openflow rules to hold the original public IP and redirect those packets sent to this IP address to the migrated VM. As shown in Fig. 9, with the proposed mechanism, a VM associated with the floating IP can keep receiving the connection from that IP address even if the VM is migrated to another cloud.

D. Federated Storage

The goal of our federated storage service is to integrate the storage services provided by different clouds participating in the UniCloud environment. Since each cloud is managed by its own cloud service provider or third-party organization, the storage services may be varied significantly in many ways, in terms of API interface, I/O performance, geographic location, level of data availability, reliability or security, etc. Therefore, our proposed service not only allows users to specify their requirements for their data, but also designs a matchmaking strategy to select the proper storage providers for storing the data. To achieve our goal, some key research questions are addressed in our approach.

- How to provide a unified storage service API?
- How to select a proper storage provider for servicing each storage request?
- Can we achieve better I/O performance through a federated storage environment?

Our federated cloud storage system consists of three main components as shown in Fig. 10. The actual data is stored in

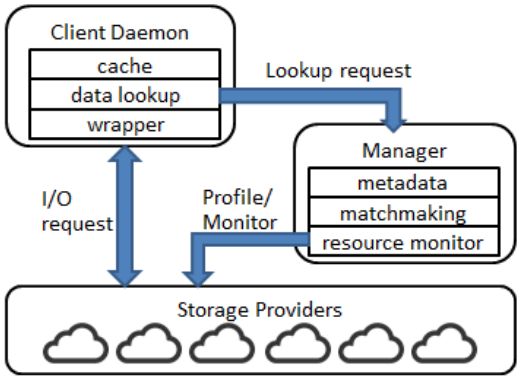

Fig. 10. Architecture of the federated storage system

multiple storage providers, which may be provided by different participants in UniCloud. The federated cloud storage manager is responsible for three important tasks: (1) collecting the information of all storage providers; (2) deciding where to store the data through a matchmaking policy; (3) resolving data lookup requests from the storage client. At the client side, we deploy a client daemon of the federated cloud storage to serve storage requests. The client daemon is mainly composed of two sub-components. One is the wrapper which defines a unified I/O interface based on the S3-like restful protocol which is useful to negotiate with various storage providers. Accordingly, arbitrary storage providers can be plug-in or register into our storage system. The other subcomponent is the lookup cache which prevents redundant lookup operations between I/O requests. When a lookup request of the data location is sent to the storage manager, the data transfer is directly performed between the storage provider and the user while the desired data is located.

In our current implementation, we support a basic object storage service to demonstrate our approach and benefit. The user APIs are summarized in Table 2. Through the APIs, users could upload/download files, or create, delete, and list folders. Instead of selecting the storage location for each file or data object, we select the data location at the per-folder basis. In other words, all the files immediately under a folder must be stored in the same storage provider, but a sub-folder can be stored separately from its parent folder. For example, all the files under "/folder1" can be stored in storage provider1, but the files under "/folder1/subfolder" can be stored in another storage provider.

We made this design decision for many reasons. First, all the files under the same folder are likely to have similar storage requirements and file properties. Therefore, the same storage provider can satisfy all their needs. Second, the amount of metadata we need to store by the manager is reducible such as to improve the scalability of our system. Last but not least, our client side cache can effectively minimize the number of data lookup requests when users access multiple files under the same folder. Therefore, our storage selection decision happens when users call "*create_folder()*" request along with a given description. The

Table 2: APIs of the federated storage system

Type	Command	Parameter
File	*upload_file*	filename, parent folder fullpath
	download_file	filename, parent folder fullpath
Folder	*list_folder*	folder fullpath
	create_folder	folder fullpath, description
	delete_folder	folder fullpath

attribute={BW, Availability, Reliability}
level = {HIGH, MED, LOW}
type = {LOG, BACKUP, ACTIVE, TEMP}
description = {constraint}*; {preference}*; {hint}*;
constraint = {attribute} {"<" || ">" || "="} {values}
preference = {attribute} : {level}
hint ={type}

Fig. 11. Syntax of the requirement description for storage

description allows users to specify the desired storage requirements and preferences. Our matchmaking algorithm can select the best storage provider to store the files. Details of the description are explained next.

Each storage provider in our system is characterized by a list of attributes, such as "availability", "reliability", and "performance", etc. Assume that the value of these attributes is normalized between 0 and 1, and a greater value means the higher quality resource. The values is either measured by our storage manager or given by the storage providers. Based on these attributes, users may specify their storage requirements using three types of rules: *constraint*, *preference*, and *hint*. The complete syntax of our resource description is defined in Fig. 11. The *constraint* is a more restrict rule that allows users to specify a threshold value on a certain attribute, so that the selected candidates must satisfy the given requirement. However, users have to require extensive knowledge to set the proper threshold values for their data. If the value is too high, the storage system might not be able to find any candidates; if too low, the results may fail to meet user's expectation.

Therefore, our second rule, *preference*, only asks users to specify the level of importance of an attribute to their data, in terms of HIGH, MED or LOW. The level of importance implies two things in our matchmaking algorithm. First, it represents the requirement of resource quality relative to all other storage providers. For example, the HIGH means the attribute value of the selected candidates should be ranked in the top 33% from our resource pool, while the LOW only requires the selected candidates to be ranked within 66%. Second, it indicates the importance of an attribute relative to other attributes. In other words, when multiple candidates are satisfied with all the given requirements from users, our matchmaking algorithm selects the one that has higher values from more important attributes.

Moreover, we consider that most of users are only familiar with the type of data from their applications, but not familiar with the meaning of system-defined attributes. Therefore, our third requirement rule is a *hint* that automatically generates the proper preference based on the type usage of data given from users. Users may also overwrite the preference on a specific attribute according to the second rule.

Table 3: Example of pre-defined system parameters in the federated cloud storage

Type	Availability	Reliability	Perfor-mance	File Size	R/W Ratio
LOG	HIGH	MED	LOW	Small	0
BACKUP	LOW	HIGH	MED	Large	0
ACTIVE	HIGH	HIGH	HIGH	Small	0.5
CACHE	MED	LOW	LOW	Small	1
BIGDATA	HIGH	HIGH	MED	Large	1

Table 4: Measured virtualization overhead and consolidation overhead of UniCloud

Overhead		CPU	Memory	Disk
Virtualization		0.01	0.03	0.30
Consoli-dation	CPU	0.01	0.01	0
	Memory	0.13	0.52	0.03
	Disk	-0.07	-0.25	0.54

Table 3 lists a couple of data types, default preferences, and file characteristics, which are commonly used in storage applications. Note that, all the settings in the table are configurable in our system based on user experience, resource condition, and other factors. The discussion on how to configure these system parameters is out of the scope of this paper. Instead, we only present the design and mechanism of the system to provide the usability and flexibility of our storage service.

Based on the given resource requirement above, our matchmaking algorithm is divided into four phases. The first phase is to filter the candidates that do not satisfy with the threshold according to the constraint rule. The second phase is to filter the candidates that are not ranked within a preferred percentile among all resource providers according to the level of preference. The third phase is to calculate the total value of a candidate by aggregating all the attribute values and weighted by the attribute preference. Finally, our system selects the candidate with the highest total value to be the final decision. As shown by our evaluation results, our matchmaking algorithm can select more proper storage provider for users with different requirements and preferences, and maximize their I/O performance.

E. SLA-Based Resource Provisioning

The goal of SLA-based resource provisioning is to assign desired resources to requests such the service level agreement, companied with requests, can be satisfied. In cloud computing, there are different types of SLAs, such as the availability of services. In the UniCloud, the SLA-based resource provisioning focuses on performance assurance, whose objective is to ensure the provisioned virtual machines can have similar performance as physical machines. Moreover, the utilization of physical machines is expected to be as high as possible, which means the provisioning algorithm will consolidate virtual machines if such an action does not affect the performance of SLA.

Our approach consists of three steps: machine benchmarking, performance modeling, and dynamic provisioning. In the first step, we analyze the major performance degradation on cloud platforms. In the second step, we build the performance models, based on the analysis in the first step, to predict the performance degradation. In the last step, we develop a dynamic allocation mechanism to adjust the resource allocation according to the VM behaviors and our performance models.

In our analysis, the major performance degradation of cloud platforms comes from two factors: virtualization and consolidation. To quantify their influences, two overhead metrics are defined: the virtualization overhead and the consolidation overhead. Given a program p, let t_1 be the execution time of p on a physical machine; t_2 be the execution time of p on a virtual machine that is running solely on a physical machine; and t_3 be the execution time of p on a virtual machine that is running with other virtual machines. The virtualization overhead of p is defined as $(t_2\text{-}t_1)/t_1$; and the consolidation overhead of p is $(t_3\text{-}t_1)/t_1$.

The virtualization overhead and the consolidation overhead are varied on different cloud platforms owing to the techniques and machine specification in use. Therefore, a benchmarking tool is designed to measure those metrics. We further categorize the computing resources as CPU, memory, and disk IO according to the difference in performance degradation under virtualization and consolidation. For instance, with hardware support virtualization, the virtualization and consolidation overhead of CPU benchmarks are almost zero. However, for disk IO, the virtualization overhead can be nearly 50% for write and 40% for read. Table 4 lists the benchmarking results on UniCloud.

The performance model is built based on the profiles of programs and the measured metrics of performance degradation. We use monitoring tools to obtain the profile of programs, or more specifically, the running virtual machines. In UniCloud, the launched virtual machine is managed by KVM (Kernel-based Virtual Machine) [6] hypervisor, which is a process running on physical OS. Performance monitoring and profiling tools, such as PAPI (Performance API) [11], can generate the desired profiles of VMs. Most modern machines have hardware counters for instructions and hardware information, such as cache miss rate. PAPI can read the information online and report them in real time. We use the linear model to estimate the total performance degradation of the VM. For instance, if a_i is the percentage of instructions using computing resource i, and the performance degradation of computing resource i is b_i, the total performance degradation of the VM is estimated as $\sum_{i=1}^{K} a_i b_i$, where K is the number of computing resources.

The provisioning for VM requests follows the OpenStack procedure, which consists of two parts: filtering and weighting. In the filtering stage, the qualified computing resources (PM) are selected; in the weighting stage, one of the qualified PM is chosen according to some cost functions. Initially, since there is no programming execution information, we use the requested resources as the filtering criteria in the filtering stage. That is, the system finds a physical machine that fits the specification to host the request virtual machine. After the VM is running, the scheduler polls the monitoring subsystem to obtain the runtime information of the virtual machines, such as memory usage, disk IO frequency, etc. When some

profiling information is gathered, we use the performance modeling as the filtering conditions. If the current provisioning cannot guarantee the SLA performance, the action of migration is taken. On the other hand, if there are some VMs that can be consolidated without scarifying their performance, the consolidation process will be activated.

IV. EVALUATION

To illustrate the achievement of our UniCloud platform, we design some experiments to explore the feasibility in terms of federated compute and federated storage.

A. Federated Compute

To evaluate the federated computation, two geographically different campus clouds are deployed. One cloud is located in NTHU at HsihChu and the other one is located in NTCU at TaiChung. Each cloud is composed of three servers with the installation of OpenStack Grizzly, in which one server acts as both the controller node and the network node while two servers act as compute nodes for the VM hosting. Different campus clouds may have heterogamous hardware equipment. Detail information about the testing environment is listed in Table 5.

The construction of a virtual cluster is equally to deploy VMs on two clouds with the same instance flavors. The NASA NPB [7] is exploited to estimate the feasibility of collaborative cloud services over the UniCloud prototype. We adopt the EP (Embarrassingly Parallel) benchmark to evaluate the performance of a cross-cloud virtual cluster. The problem size of Class B and Class C [8] are both assigned to the experiments with a varied number of VMs. Experimental results are also averaged in several rounds.

Fig. 12 and Fig. 13 reveal that the EP problem with the larger size takes higher average time to complete the computation. The average execution time of Class C is higher than that of the Class B. In addition, the more the VMs are launched in a virtual cluster, the less the average time will be. Particularly, the average execution time of NTHU cloud outperforms that of the NTCU cloud due to the computing capability of hardware equipment.

On the other hand, the performance of a cross-cloud virtual cluster is moderate. The average execution time of a cross-cloud virtual cluster is longer and shorter than that of single NTHU cloud and that of single NTCU cloud, respectively. That is because some VMs in a cross-cloud virtual cluster are allocated in the NTHU cloud and the others are resided in the NTCU cloud. In addition, each single cloud cannot fulfill a large demand while more VMs are required. The experiment demonstrates that our UniCloud platform can supply the large demand of a virtual cluster across multiple clouds.

B. Federated Storage

We conduct real experiments to show the performance benefit from using our federated cloud storage approach in UniCloud. In our testing environment, we setup four storage providers and each is based on different file systems and configurations as shown in Table 6 and Table 7.

For simplicity, here we consider performance as the only resource property in our setting. As a result, we observe these

Table 5: Hardware/Software information for the experiment

Site	Hardware			Software	
	Node	CPU	Ram	OS	Hyper-visor
NTHU Cloud	Controller	Intel X5670	24GB	Ubuntu 12.04	KVM
	Compute1				
	Compute2				
NTCU Cloud	Controller	Intel E5520	18GB	Ubuntu 12.04	KVM
	Compute1				
	Compute2	Intel 5130	2GB		

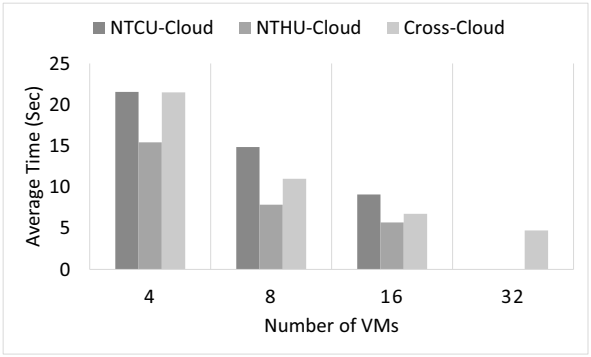

Fig. 12. Results of a virtual cluster running with Class B of the EP

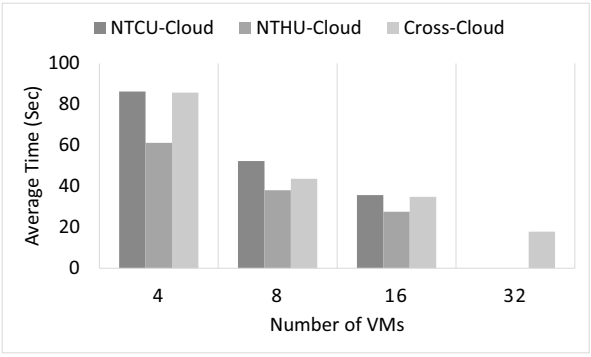

Fig. 13. Results of a virtual cluster running with Class C of the EP

storage providers differ in I/O performance when servicing various types of I/O requests. For example, SP1 and SP4 has better I/O performance for writing large file, but SP2 and SP3 has better I/O performance for reading large file. Therefore, our objective is to maximize the overall I/O performance by placing files onto the storage providers that could achieve the highest throughput according to their access pattern.

Our evaluation is based on a real file access trace of a FTP server during a ten-day period [18]. We extract the file upload and download operations, and observe three important workload characteristics from the trace: (1) the file access requests are distributed over folders following the 80/20 rule. In other words, the 80 percent of file upload or download requests occurs in 20 percent of the folders; (2) the file size distribution can be classified into small and large files, and each of them can be characterize by a normal distribution

Table 6: Configurations of four storage providers in our federated storage environment

Storage Provider	File System	Replication Factor	Number of Nodes
SP1	Swift	3	5
SP2	Ceph	3	18
SP3	Ceph	2	4
SP4	HDFS	2	4

Table 7: Measured I/O throughputs for various I/O patterns in our federated storage environment

Storage Provider	Small file (1MB)		Large file (100MB)	
	Read	Write	Read	Write
SP1	4.60 MB/s	0.68 MB/s	6.50 MB/s	20.17 MB/s
SP2	15.01 MB/s	0.65 MB/s	29.49 MB/s	14.50 MB/s
SP3	11.49 MB/s	0.44 MB/s	30.70 MB/s	11.92 MB/s
SP4	8.07 MB/s	0.67 MB/s	11.53 MB/s	22.40 MB/s

function as shown in Fig. 14; (3) the size of files within the same folder is similar. As shown in Fig. 15, all the large files are located in three folders.

The experiments firstly create folders that content the files for I/O requests. According to the expected I/O pattern in that folder, we classify the folders into five types: LOG, BIGDATA, CACHE, BACKUP, ACTIVE. The file size and read/write ratio of each type are listed in Table 3. Hence, when we use the federated storage API to create these folders in our system, we also specify the type as a hint in the folder description. Our matchmaking algorithm will place those folders to a proper storage provider based on the performance measured in our system as shown in Table 7. For example, the BACKUP data means writing large files. Our matchmaking algorithm will sort candidates based on the performance for writing a large file, and place the file to SP4. In contrast, the BIGDATA means reading large files, so that the file will be allocated to SP3. As shown in Fig. 15, our random generated files are distributed across all four storage providers due to each of them can achieve the best I/O performance for a certain type of files. For example, SP1 is selected for small file writes; SP2 is selected for small file reads; SP3 is selected for large file reads; and SP4 is selected for large file writes.

Finally, we exploit FABAN [3] to generate the workload request for file upload and download according to our observed workload characteristics. FABAN has been widely used for evaluating Cloud services, and it can measure the response time of each service request (i.e. a file upload or download request in our experiments). In each run of our experiments, we continue sending I/O requests into our system for over 30 minutes, and we report the performance results by taking the average over five runs.

As shown in Fig. 16, our federated storage, named as FCSS in the plot, significantly outperforms any other single storage providers by at least 35%, and reaches almost 20MB/s throughput in average. The second best storage provider, SP3,

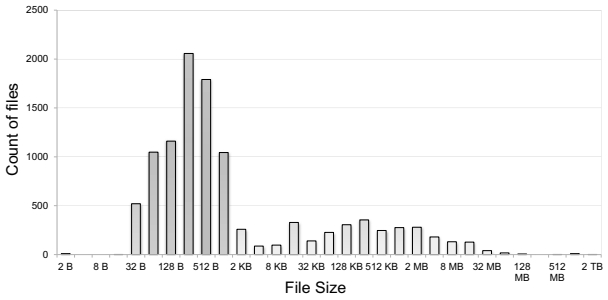

Fig. 14. File size distribution from the FTP trace log

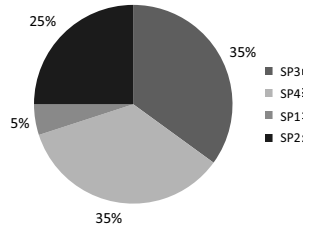

Fig. 15. File location distribution in federated storage according to the matchmaking results

only achieves 15 MB/s. In Fig. 17, we further breakdown the performance comparison with respect to each type of files. As shown, our federated storage may not achieve the better performance for individual type of data access. However, our matchmaking algorithm can explore multiple storage options, and optimize for each type of data. Therefore, we can still significantly improve the overall I/O performance, and satisfy their individual requirement for availability, reliability or any other resource properties.

V. CONCLUSIONS

This paper introduces a prototype of Taiwan UniCloud that is a community-driven hybrid cloud platform for academics to support education, research, and application development in cloud computing. Each self-managing cloud can join the UniCloud to share its resources and simultaneously leverage the resources and scale-out capabilities of other clouds. The architectural of UniCloud is presented and corresponding issues we tackled are also discussed in terms of a cross-cloud virtual cluster, VLAN over WAN, live migration across clouds, and a federated storage across different providers. The feasibility of our current implementation is also demonstrated through various experiments. Based on the platform, a large resource demand is affordable based on the federated computation and the collaborative cloud services and applications could be further sustained. In addition, the federated storage gains the overall performance of matchmaking between disparate data types and different access properties among multiple storage providers.

The future work will include adding more campus clouds to the UniCloud platform. We will also continue investigating methods for improving the performance, as well as enriching functionalities of our platform, for instance a self-adaptive framework for multi-cloud resource provisioning, integrating with more APIs to improve the interoperability with other clouds services, and investigating security concerns. A trial run of cloud applications for academic education and research

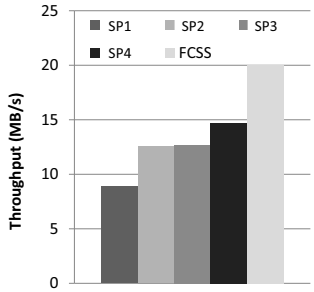

Fig. 16. Average IO throughput of all requests

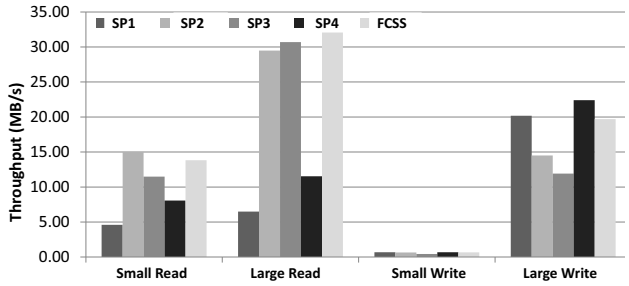

Fig. 17. Performance comparisons with respect to each type of files

is also considerable to improve the reliability of handling runtime failures. The long-term vision of our UniCloud is to provide an open and self-sustained cloud ecosystem to deliver infrastructure resources, runtime platform, and software application as a service for users.

ACKNOWLEDGMENTS

The authors would like to thank You-Fu Yu, Meng-Ru Hsieh and Chien-Yu Liu for their assistance in implementing and evaluating, and thank Jatinder Singh and anonymous reviewers for their insightful comments to polish this paper.

REFERENCES

[1] Amazon Elastic Compute Cloud (Amazon EC2). Available: http://aws.amazon.com/ec2/.

[2] Amazon Web Service (AWS). Available: http://aws.amazon.com/.

[3] FABAN. Available: http://faban.org/.

[4] Google App Engine (GAE). Available: http://appengine.google.com.

[5] Google Compute Engine (GCE). Available: http://cloud.google.com/products/compute-engine.

[6] KVM (Kernel-based Virtual Machine). Available: http://www.linux-kvm.org/page/Main_Page.

[7] NAS Parallel Benchmarks. Available: http://www.nas.nasa.gov/publications/npb.html.

[8] NPB Problem Size. Available: http://www.nas.nasa.gov/publications/npb_problem_sizes.html.

[9] OpenStack Cloud Software. Available: http://www.openstack.org/.

[10] OpenStack Networking. Available: http://docs.openstack.org/grizzly/basic-install/apt/content/basic-install_network.html.

[11] Performance Application Programming Interface (PAPI). Available: http://icl.cs.utk.edu/papi/.

[12] SalesForce.com. Available: http://www.salesforce.com/.

[13] Windows Azure. Available: http://www.windowsazure.com/.

[14] R. Buyya, R. Ranjan, and R. N. Calheiros, "InterCloud: Utility-Oriented Federation of Cloud Computing Environments for Scaling of Application Services," in *Proceedings of the 10th International Conference on Algorithms and Architectures for Parallel Processing* (*ICA3PP*), Springer-Verlag, Busan, Korea, 2010, pp. 13-31.

[15] R. N. Calheiros, A. N. Toosi, C. Vecchiola, and R. Buyya, "A Coordinator for Scaling Elastic Applications Across Multiple Clouds," *Future Generation Computer Systems,* vol. 28, no. 8, 2012, pp. 1350-1362.

[16] M. Mahjoub, A. Mdhaffar, R. B. Halima, and M. Jmaiel, "A Comparative Study of the Current Cloud Computing Technologies and Offers," in *The First International Symposium on Network Cloud Computing and Applications* (*NCCA*), IEEE, Toulouse, France, 2011, pp. 131-134.

[17] M. L. Massie, B. N. Chun, and D. E. Culler, "The Ganglia Distributed Monitoring System: Design, Implementation, and Experience," *Parallel Computing,* vol. 30, no. 7, 2004, pp. 817-840.

[18] R. Pang and V. Paxson, "A high-level programming environment for packet trace anonymization and transformation," in *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications* (*SIGCOMM*), ACM, Karlsruhe, Germany, 2003, pp. 339-351.

[19] I. Voras, B. Mihaljevic, M. Orlic, M. Pletikosa, M. Zagar, T. Pavic, K. Zimmer, I. Cavrak, V. Paunovic, I. Bosnic, and S. Tomic, "Evaluating open-source cloud computing solutions," in *Proceedings of the 34th International Convention on Information and Communication Technology, Electronics and Microelectronics* (*MIPRO*), IEEE, Opatija, Croatia, 2011, pp. 209-214.

[20] X. Wen, G. Gu, Q. Li, Y. Gao, and X. Zhang, "Comparison of open-source cloud management platforms: OpenStack and OpenNebula," in *The 9th International Conference on Fuzzy Systems and Knowledge Discovery* (*FSKD*), IEEE, Sichuan, China, 2012, pp. 2457-2461.