International Journal of Automation and Computing 8(3), August 2011, 269-279
DOI: 10.1007/s11633-011-0582-4

IC Cloud: Enabling Compositional Cloud

Yi-Ke Guo Li Guo
Department of Computing, Imperial College London, London SW7 2BZ, UK

Abstract: Cloud computing has attracted great interest from both academic and industrial communities. Different paradigms, architectures and applications based on the concept of cloud have emerged. Although many of them have been quite successful, efforts are mainly focusing on the study and implementation of particular setups. However, a generic and more flexible solution for cloud construction is missing. In this paper, we present a composition-based approach for cloud computing (compositional cloud) using Imperial College Cloud (IC Cloud) as a demonstration example. Instead of studying a specific cloud computing system, our approach aims to enable a generic framework where various cloud computing architectures and implementation strategies can be systematically studied. With our approach, cloud computing providers/adopters are able to design and compose their own systems in a quick and flexible manner. Cloud computing systems will no longer be in fixed shapes but will be dynamic and adjustable according to the requirements of different application domains.

Keywords: Cloud computing, compositional cloud, infrastructure as a service (IaaS), cloud service, cloud elasticity.

1 Introduction

Cloud computing has been widely studied in the past few years as it demonstrated the potential to shape the way of development, maintenance and deployment of both computing hardware and software resources. Cloud computing is not a pure technical development, but a commercial reality. It has a profound reason for being adopted and rapidly developed. The technical development in the last 10 years lays down a solid foundation for cloud computing. The increased degree of connectivity, the hugely improved communication bandwidth, and the rapidly expanding amount of data have led many computational service providers and data centers to employ larger infrastructures with dynamic load and access balancing mechanism. By distributing and replicating data across servers on demand, resource utilization has been significantly improved. Such an improvement of scalability via dynamic load balancing was also realized in large scale web server hosting scenario with respect to the accessing scaling. All these technologies have made the offering of resources to the general public in a managed and elastic fashion to become possible. Cloud computing adopts various technologies, including virtualization, service oriented architecture (SOA), grid computing, utility computing, etc., to realise a completely new industrial paradigm where computing is uniformly provided by a cloud which is "an elastic execution environment of resource involving multiple stakeholders and providing a metered service at multiple granularities for a specified level of quality (of services)"[1].

Both academic and industrial people have shown great interest in cloud computing. Different paradigms, architectures, and implementations based on the concept of cloud have been proposed. Cloud computing, based on the content of services provided, is divided into three conceptual categories, namely, infrastructure as a service (IaaS), platform as a service (PaaS) and software as a service (SaaS). IaaS focuses on how to provide hardware resources as services, while PaaS delivers application development platforms as services and SaaS focuses on deploying software as services.

Developing a cloud computing system includes many design decisions regardless of the genre of the system. So far, efforts have been devoted into developing particular cloud systems, which mainly focuses on providing easy access services to the end users within certain focused contexts. Although many efforts have been successful, a generic and more flexible design space, where particular cloud system development can be studied and built up, is missing from the current literature. Not much work or many systems have been provided to help cloud service providers construct new cloud platforms and technologies. Transferring such knowledge from one system to another is also very difficult.

The work presented in this paper focuses on a rather different approach-enabling compositional cloud computing technology. We present Imperial College Cloud (IC Cloud) as a design space for cloud system development. Instead of providing a specific cloud computing system, IC Cloud aims to enable a generic design space where various cloud computing architectures and implementation strategies can be systematically studied. With our approach, cloud computing providers/adopters are able to design and implement their own systems in a very quick and flexible manner by systematically composing a set of well-developed services based on a service level agreement (SLA) oriented requirement specification. Cloud computing systems built in such manner will no longer be in fixed shapes but will be adjustable according to the requirements of different applications.

The IC Cloud design strictly follows the SOA principle and adopts a component-based implementation. Especially we impose minimal architecture specificity during the IC Cloud construction and unify every component as web services. Furthermore, we use a coordination language as the systematical composition mechanism of the system which also provides the capacity of dynamically adjusting the shape of the system based on different requirements.

This article is organised as follows. Some related work and technologies for the development of cloud computing

Manuscript received August 3, 2010; revised March 7, 2011

systems are presented in Section 2. In Section 3, we explain in detail the design rationale behind our compositional cloud framework as well as all the essential components required for the same. We further extend our discussion on how a cloud system can be composed in Section 4 using a simple example. In Section 5, we summarise our work and discuss the direction of possible future research and development.

2 Background and related work

One of the core ideas of cloud computing is virtualisation. Many projects have been producing virtualisation software (hypervisors)[2−5] to enable the implementation of the basic infrastructure of cloud computing. The Xen hypervisor[4] was developed in Cambridge University. It can run on a wide variety of computing systems. It currently supports Linux, NetBSD, FreeBSD, Solaris, Windows, and other common operating systems as guest operating systems (OSs). It is well-known for its near-native performance and its use of paravirtualization. The kernel-based virtual machine (KVM)[6] relies on CPU support for virtualization and leverages existing Linux kernel infrastructure to provide an integrated hypervisor approach (as opposed to Xen′s stand-alone hypervisor approach). KVM is known for its rapid inclusion into the mainline Linux kernel.

An interesting and a very useful piece of work is libvirt, the virtualisation application programming interface (API)[7]. It provides an abstract layer on top of many different hypervisors such as Xen and KVM in a programmable way. The key advantage of using such an abstract layer for the IaaS development is that the upper level development such as the control of VMs′ lifecycle, storage, and network management is completely separated from the underlying hypervisor′s implementation. With libvirt, IaaS cloud platform development only needs to concentrate on the upper level without considering the low level hypervisors adopted.

With respect to cloud computing platforms, there are quite a few services available in the market such as the Amazon EC2/S3/AWS, GoGrid, [8] and RackSpace[9]. Academic efforts include Virtual Workspaces, OpenNebula[10], and Reservoir[11]. Among them, Amazon web services (AWS) is no doubt the leading and the most successful IaaS solution in the commercial world. It enables its users to control the entire software stack on top of the hardware instances (VM images) provided. Hardware such as CPUs, memory, disk volumes are provided to users on a pay-as-you-go basis. Users are either able to create a new Amazon machine image (AMI), which replicates their local working environment, or select from a library of globally available AMIs that are provided by others. They then need to upload the created or selected AMIs to Amazon simple storage service (S3) before he can start, stop, and monitor instances of the uploaded AMIs.

Google App Engine[12] provides a PaaS based cloud platform which enables its users to run their applications written by composing functions provided by the platform using the Python programming language. Thus, in addition to supporting the Python standard library, Google App Engine also supports APIs for various Google provided services. Google App Engine also provides a web-based administration console for users to easily manage their running web applications.

Microsoft Azure[13] targets on providing an integrated hosting, development and manageable cloud computing environment so that application developers can easily create, host, manage, and distribute both Web and standalone applications through Microsoft data centres. In order to achieve this goal, Microsoft Azure supports a comprehensive collection of proprietary development tools and protocols which includes Live Services, Microsoft .NET Services, Microsoft SQL Services, Microsoft SharePoint Services, and Microsoft Dynamics CRM Services. It also supports web service standards like SOAP and REST to allow applications to interact non-Microsoft solutions.

Sun network.com (Sun grid)[14] allows its users to run Solaris OS, Java, C, C++, and FORTRAN based applications. A user has to build and debug his applications and runtime scripts in a local development environment that is configured to be similar to that on the Sun environment. The user then needs to create a bundled zip archive (containing all the related scripts, libraries, executable binaries and input data) and upload it to Sun grid. At last, he/she can execute and monitor the application using the Sun grid web portal or API. After the completion of the application, the user will need to download the execution results to his local development environment for viewing.

The open source project, Eucalyptus[15], is built to allow administrators and researchers to deploy a cloud computing infrastructure. The whole system is designed and implemented in a hierarchical fashion and with an interface compatible AWS. Eucalyptus claims to be highly modular with each module represented by a well-defined API and therefore enabling users to replace components for experimentation with new cloud-computing solutions. Also, the system exposes its features through a common set of Amazon EC2 and S3 interfaces and expects users who are familiar with EC2 and S3 to transition seamlessly to an Eucalyptus deployment. It now enables businesses of any size to leverage their own IT resources to get the benefits of cloud computing and it eliminates lock-in, security ambiguity, and unexpected storage costs that could associate with public clouds. The newest version of Eucalyptus makes available a variety of relevant information which can be used in open source tools like Nagios and Ganglia, providing cloud administrators with insights on various aspects of the cloud as well as the status of specific Eucalyptus components.

OpenNebula focuses on the resource management issues in cloud computing. To manage virtual infrastructures, it provides an abstract view of virtual resources regardless of the underlying virtualisation platform. It manages the full lifecycle of the VMs and supports configurable resource allocation policies. It is believed that cloud resources may be limited. In the circumstances where the resource demands cannot be met, the requests for resources will have to be prioritised, queued, pre-reserved, deployed to external clouds, or even rejected. This can be solved with resource lease managers such as Haizea, which is alike to futures market for cloud computing resources. It pre-empts resource usage and puts in place advance resource reservations so that highly prioritised demand can be served promptly. Haizea acts as a scheduling backend for OpenNebula, and together they overwhelm other virtual infrastructure managers by giving the functionality to scale out to external clouds, and providing support for scheduling groups of VMs so that either the entire group of VMs are provided with resources or no member of the group is.

All these works provide rich technical mechanisms for building cloud systems. It is therefore becoming desirable, as well as possible, that we can build up a design space by making proper abstraction over these works so that a cloud system can be systematically studied, designed and constructed. We will present our attempt to this goal by describing our development of IC Cloud.

3 Enabling compositional cloud

3.1 Rationale behind compositional cloud

Acknowledging that "one model fits all" is unlikely to be appropriate for the development of cloud computing.

1) Cloud computing is an aggregation of many existing computing technologies. Different service providers and vendors have their own understandings about cloud computing so are their designs and system implementations. It seems impossible or improper to debate which one is the best design, and it is difficult for early cloud computing developers and/or adopters to try out all the possibilities.

2) The second argument is that cloud computing platforms are still fast evolving. New features and functions will emerge while new requirements are being proposed. It is unrealistic to assume that a pre-designed system will fit all the future requirements.

3) The last but not the least, cloud computing system only emerged gradually from the existing computing system as technology evolves. Therefore, it is impossible to clearly define the boundary between these two. In other words, the new and cloud based systems will incorporate many features, components and data of the old systems, and it is hard to pre-define rigidly what the new system′s shape would be beforehand.

All these observations suggest that we require a framework where technical design options and system construction approaches can be systematically studied. The need for such a systematical study of cloud system becomes even more important when the dynamical adaptive resource provision becomes desirable in a cloud system. That is, to be scalable and sustainable, a cloud computing system needs a form of meta-scalability which enables the system to evolve along the complex environment in which it operates. A system is said to be meta-level scalable when it has the ability to adapt to any change in an evolving environment. In the case of cloud computing, a cloud platform is said to be meta-level scalable when it is designed in such a way that it can be reconfigured by its users or by the system itself dynamically to satisfy specific needs, emerging activities, means of coordination and rules. This means that cloud platform designer should adopt a different approach. They should separately focus on the development of atomic services of cloud computing systems and the composition mechanisms of them.

In Fig. 1, we illustrate a simple framework of compositional cloud. Instead of constructing any specific cloud infrastructure in advance, cloud infrastructure providers submit their requirements to the compositional cloud which maintains a set of atomic services and resource pools. A set of coordination descriptions are then generated along with a list of selected services and VMs, which together form a specific cloud setup (cloud one or cloud two as shown in the diagram) for a particular cloud construction requirement. IC Cloud is developed as a compositional cloud system based on this simple design principle.

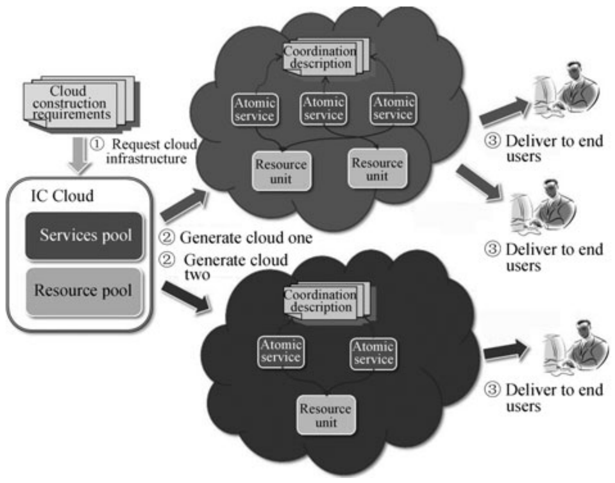


Fig. 1 Compositional cloud

3.2 Components for IC Cloud

There are five main components defined for IC Cloud, namely, VM, service and VM pool, atomic service, coordination description, and cloud requirement description.

3.2.1 VM

VM is the most fundamental and manageable computing element in IC Cloud. Each VM, based on users′ requirements, can have number of CPUs, memories, storage volumes and an OS defined. It is not functional but only acts a "box" that carries resources serving for users′ applications. A user can have as many as possible VMs in his/her cloud setup. Fig. 2 shows all the properties that are defined for a VM.

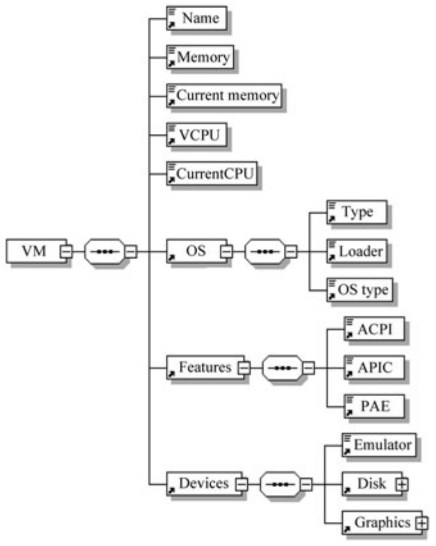


Fig. 2 Virtual machine properties

In Fig. 2, ACPI denotes advanced configuration and power interface, APIC denotes advanced programmable interrupt controller, PAE denotes physical address extention, and OS denotes operating system.

3.2.2 Service and VM pool

Service pool acts as a service repository which enables publishing, searching, and match making of all available atomic services. It is a direct implementation of the service registry concept of SOA. VM pool is a resource container from which VMs are created. It usually contains a large amount of computing resources which are sliced into smaller resource units based on the user requirements. Many VM pools can be deployed in parallel in a data centre in order to provide sufficient computing resources. At implementation level, the concept of VM pool may be mapped to a physical server or a cluster of servers.

3.2.3 Atomic service

Atomic services refer to the fundamental functions that are required in almost every cloud computing system. They provide computational operations for cloud systems. Every user composed cloud can have any number of atomic services based on the features of their applications.

We summarised and implemented twelve atomic services in IC Cloud. They are also described in a semantic manner by specifying their functional properties (inputs, outputs, precondition, and post-condition) and non-functional properties (cost, pricing model, etc.) as shown in Fig. 2.

1) Requirements processing service. It converts high level cloud construction requirements to detailed and executable requirements. Users are not always able to specify their requirements in technical details but can only explain what their applications are trying to achieve. For such a case, requirements processing service comes into play.

2) Contract service. This service deals with all contract related issues such as what is the price plan; when is the bill made and when the payment needs to be made. It ensures the agreement between cloud providers and users.

3) Billing service. This service periodically reviews the resource usage of a user and generates bills based on the contract that the user signed.

4) Payment service. It handles all payment related issues such as how to deal with different payment methods, whether there is a pending payment.

5) Monitoring service. Monitoring service is one of the most important components in our system. It keeps monitoring the VMs′ health status periodically. It also acts as a utility metre that keeps recording all the VMs resource usages in a VM pool including CPU time, memory, I/O operations, networking traffic, etc.

6) Security service. It is designed to provide security functions such as authentication, authorization, credential conversion, auditing, and delegation. It handles the details of processing and validating authentication tokens, and it evaluates policy rules regarding the decision to allow the attempted actions based on information about the requestor (identity, attributes, etc.), the target (identity, policy, attributes, etc.), and details of the request. Also, it securely logs relevant information about events.

7) Communication control service. It handles the networking issues of all VMs in the system. It is responsible for initialising the local sub-net of a VM pool and allocates MAC addresses to newly created VMs. It keeps tracking of the allocation of IP addresses in the sub-net. If more IP addresses are needed, it will generate more items in the IP address and MAC address mapping table and refresh the dynamic host configuration protocol (DHCP) server on each VM pool. In addition, it controls how network packages are sent, routed, and forwarded between VMs that cross different sub-net to perform virtual LANs management. In this way, VM users are unaware of the underlying network infrastructure and can be managed based on the sub-network basis.

8) Storage service. The storage controller provides a basic storage service which is similar to S3 to the IC Cloud users. Users are able to upload their data, image files through the interface provided to the IC Cloud storage space.

9) SLA service. It ensures that the signed SLA between a cloud service provider and a consumer is met. In IC Cloud, SLA issue is mainly resolved at application level. A performance repository is maintained in order to provide past performance data of each system component to the SLA service.

10) VM management service. It controls all the VM lifecycle related operations such as creating VMs, booting up VMs, shutting down VMs, pausing VMs, resuming VMs, destroying VMs and deleting VMs.

11) VM pool management service. As a cloud system is normally made of many VM pools, how to allocate resources from those VM pools becomes an issue. The VM pool management service is designed exactly for this purpose. For requests that ask for new VM instances, it fetches resource information (CPUs, memories, disk′s size, etc.) of available VM pools and decides to which VM pool the new VM instance should go to by applying different scheduling algorithm.

12) Elastic scheduling service. One of the key merits of cloud computing paradigm is its elastic resources provision feature. Elastic computing service is the enabling force for this. Based on the information fed by the monitor service, it is able to dynamically extend/shrink the resources supplied to an application.

Besides the atomic services, we also developed four extra integrated services which offer ready-to-use computing infrastructure for quick cloud solutions.

3.2.3.1 IC Cloud giant storage service

The IC Cloud giant storage service extends the atomic storage service to provided unlimited storage space. It designed for storing very big data files (next generation sequencing experiment files for example). As shown in Fig. 3, it is implemented and deployed in a distributed manner and all the components of the service are deployed on VMs to ensure its scalability. It supports automatic data backup for high data availability and is accessible both via APIs and standard ftp clients. Large files are split into smaller chunks when they are stored and are merged back when requested by the scheduling controller. The unified name space maintains the meta-data of those files, and the balancing controller automatically adjusts the workloads amongst the available data nodes. When there is not enough storage space, the balancing controller also requests more VMs.

3.2.3.2 IC Cloud high I/O performance service

The IC Cloud high performance I/O service provides extremely high speed file processing power. It is built on top of giant storage services using memory objects. The basic idea behind it is that besides storing large files in the giant storage service, we maintain a memory cache which stores the most used file objects. It benefits from the IC Cloud elastic resource infrastructure as the memory it requests to use can be provided in an on-demand manner. It is designed in an extendable way in order to adopt different persistent strategies for different I/O operations. Fig. 4 shows IC Cloud high I/O performance service.

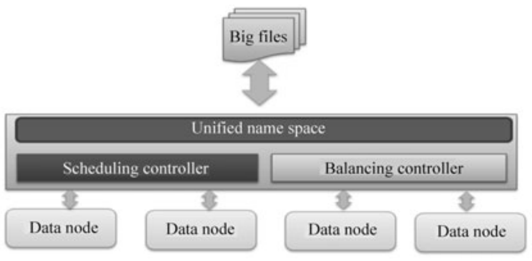


Fig. 3 IC Cloud giant storage service

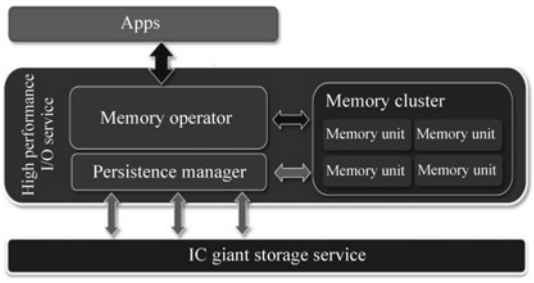


Fig. 4 IC Cloud high I/O performance service

The main components of the service are:

1) Memory cluster. It is a pool that holds available memories for storing file objects. The size of the pool, as explained earlier, is elastic as the provision of memories is in the similar way of VMs provision and we can always add or remove VMs to/from the pool.

2) Memory operator. It maintains and adjusts size of the memory cluster based on the file processing throughout, and current capacity of the memory cluster.

3) Persistence manager. It persists the file objects in the memory cluster to the IC Cloud giant storage services regularly. At the moment, we only adopt a simple algorithm for the purpose. File objects that are not required for more than five minutes are taken out from the memory cluster and are put into the giant storage service.

3.2.3.3 IC Cloud pattern service

The IC Cloud pattern service provides a list of preconfigured and ready-to-use computing architectures derived from frequently used computing patterns. The patterns that IC Cloud provides are:

1) Flat cluster pattern. It provides users with a cluster that is made up of several VMs which are able to communicate with each other. This infrastructure can be used by applications that require many computing nodes which maintain their data, system states locally and are only able to share information through message passing. Fig. 5 shows flat cluster pattern.

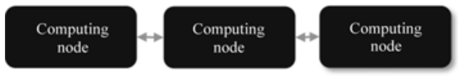


Fig. 5 Flat cluster pattern

2) Pyramid cluster pattern. Pyramid cluster has a control VM which talks to the rest of the computing VMs in the cluster. Only the networks between the control VM and each of the computing VMs are enabled. This pattern suits those applications that require a central node which maintains the whole system states and leave concrete computation to computing nodes. Fig. 6 shows pyramid cluster.

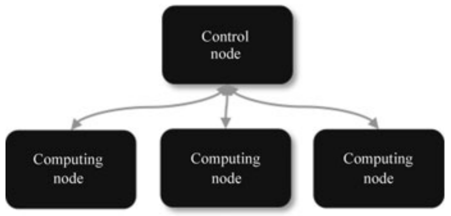


Fig. 6 Pyramid cluster

3) Diamond cluster. It is usual for applications to maintain a shared storage point. This can be achieved in two ways in IC Cloud. Applications can either use the storage services provided or use a diamond cluster. In a diamond cluster, a special VM is provided. It has limited computing power but has a volume attached, which can be accessed by all the computing VMs in the cluster. The size of volume, of course, is provided by end users when the cluster is requested. The storage node can be replaced by a database node if it is desired. Fig. 7 shows diamond clusters.

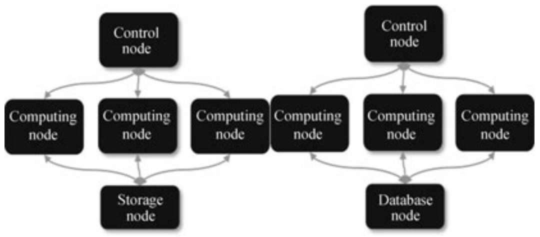


Fig. 7 Diamond clusters

4) Star cluster. It is a hybrid infrastructure of the two different diamond clusters for more complex applications on the cloud.

It should be noticed that, in all the above patterns, although the number of VMs in the clusters is given by the end users at their creation time, new VMs can be dynamically added in based on the run time application status. Fig. 8 shows star cluster.

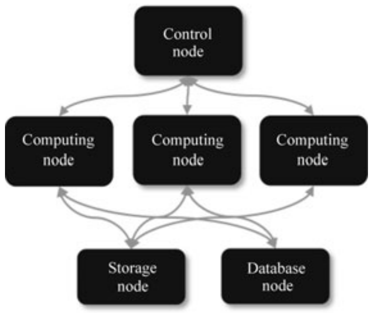


Fig. 8 Star cluster

3.2.3.4 IC Cloud giant relational database (RDB) service

The IC Cloud giant DB service provides unlimited DB space using a bunch of MySQL clusters. As it supports standard SQL interface, it can be adopted quickly by any application that requires a giant, scalable and reliable DB solution. Similar to the giant storage services, the whole giant RDB service is deployed on the VMs for its elasticity. Fig. 9 shows IC Cloud giant DB service.

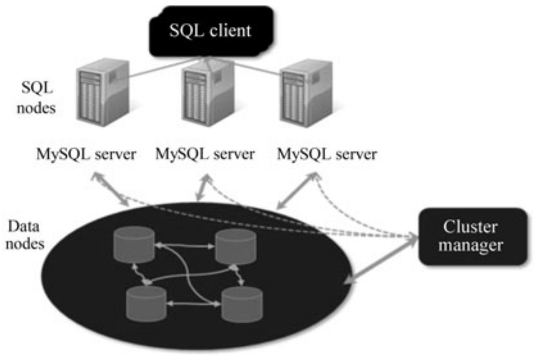


Fig. 9 IC Cloud giant DB service

3.2.4 Coordination description

As we have argued, there is no standard way to put together all the atomic services to make up a "perfect" system. Atomic services will have to be composed in a coordinated and dynamical fashion. We adopt the notion of coordination description to specify how atomic services are composed and coordinated for different cloud setups. Each user composed cloud has to be made by using coordination description in IC Cloud.

The coordination mechanism that we adopted follows the similar concept of workflow management system. Although there have been many approaches[16, 17] for composing services, we use a revised version of coordination description language-lightweight coordination calculus (LCC)[18] for atomic service composition for its simplicity.

In most of the existing coordination systems, such as conventional workflow systems, workflows need to be pre-deployed before getting executed and every new incoming request will trigger a new workflow instance based on the deployed template. To change the shape of the system, new workflows need to be deployed. With LCC, the coordination description, although usually pre-defined, can be changed at run time without affecting any running instance in the system as there is no pre-deployment process. Each submitted coordination description will directly trigger a new coordination.

3.2.4.1 Lightweight coordination calculus

LCC is a language used to represent coordination between distributed components. In a distributed system, the coordination that conveys information between components is performed only by sending and receiving messages. For example, suppose an interaction allows a component $a(r_1, a_1)$ to send a message m_1 to component $a(r_2, a_2)$ while component $a(r_2, a_2)$ is expected to reply with a message m_2. Assuming that each component operates sequentially, the sets of possible interaction sequences that can be allowed to these two components are as given below (with $M_1 \Rightarrow A_1$ denoting a message M_1 sent to A_1 and $M_2 \Rightarrow A_2$ denoting a message M_2 received from A_2.

$$a(r_1, a_1) :: (m_1 \Rightarrow a(r_2, a_2) \text{ then } m_2 \Leftarrow a(r_2, a_2))$$

$$a(r_2, a_2) :: (m_1 \Rightarrow a(r_1, a_1) \text{ then } m_2 \Leftarrow a(r_1, a_1)).$$

This definition of the message passing behaviour of the interaction is referred to as the interaction framework. An interaction framework defines a space of possible interactions determined by message passing, so the descriptions allow constraints to be specified based on the circumstance under which messages are sent or received. Two forms of constraints are permitted:

1) Constraints under which message M is allowed to be sent to component A. We write $M \Rightarrow A \leftarrow C$ to attach a constraint C to an output message.

2) Constraints under which message M is allowed to be received by component A. We write $M \Leftarrow A \leftarrow C$ to attach a constraint C to an input message.

Its complete syntax is shown below:

$$\begin{aligned}
Framework &= \{Clause, ...\}\\
Clause &= Agent :: Def\\
Agent &= a(Type, id)\\
Def &= Agent|Message|Def \text{ then } Def\\
&\quad |Def \text{ or } Def|Def \; par \; Def\\
Message &= M \Rightarrow Agent|M \Rightarrow Agent \leftarrow C\\
&\quad |M \Leftarrow Agent|M \Leftarrow Agent \leftarrow C\\
C &= Term|C \wedge C|C \vee C\\
type &= Term\\
id &= Constant\\
Constant &= Term.
\end{aligned}$$

One of the purposes of this language is to eliminate centralised engines during the coordination process, which largely improves system scalability and reduces performance bottleneck that is usually caused by the existence of the centralised server. Due to the limited paper space, we are not going to give more language details of LCC as they can be found in paper. In order to use the LCC for describing the composition and coordination of component services in building a cloud system, the LCC calculus is extended to support cloud specific features.

In a cloud system, it is very common to see operations that take long time to accomplish, such as VM image transferring, upload bundle image, etc. This requires asynchronous communication between different services (components) during the coordination process. The original LCC syntax does not support this directly. In order to express asynchronous message passing, we introduce a new term:

$$\left\{\begin{array}{l} M_1 \Rightarrow A \leftarrow C \\ \text{then} \\ M_2 \Leftarrow A \leftarrow C \end{array}\right\}_{\text{Token}}.$$

The above syntax indicates that after a message M_1 is sent to A, message M_2 will be received in an asynchronous manner. "Token" specifies the unique ID that is used to identify the asynchronous message pair.

In addition, as LCC is designed for generic coordination purpose, it does not specify message semantic for any specific application domain. In order to apply it to compositional cloud, we define a set of cloud message semantics as illustrated in Table 1. These messages, we believe, are independent of any concrete underlying implementation and are required for almost all the cloud system setups.

Coordination of different atomic services using LCC is illustrated in Fig. 10.

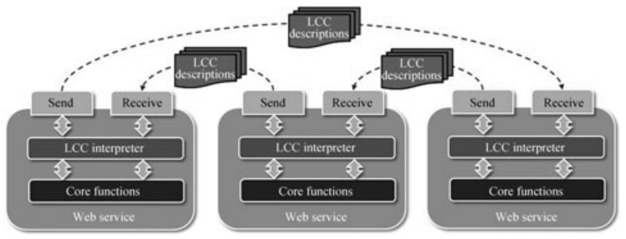


Fig. 10 LCC based atomic service coordination

In the diagram, each atomic service is implemented as a web service. Unlike most of the standard web services, each atomic service only has two public interfaces, namely, send and receive, through with which they communicate by passing the LCC coordination descriptions. An LCC interpreter is implemented and is embedded into the atomic service. It resolves the constraints specified in the LCC description by calling the methods provided by the atomic services and keeps the whole coordination process continuous.

3.2.5 Cloud requirement description

To compose a new cloud platform in IC Cloud, the first step is to submit cloud construction requirement. The document specifies both infrastructure level and functional level requirements of the potential cloud platform. The cloud requirement description can be regarded as an abstraction of service requirement specification in an SLA. Two sorts of requirements are supported by IC Cloud, i.e., explicit requirement and implicit requirement.

3.2.5.1 Explicit requirement

Users who know precisely how to compose their clouds using provided components can submit explicit requirement documents to the IC Cloud. They have to conduct system planning studies of their potential cloud setup prior to deployment. Fig. 11 shows all the properties of an explicit requirement.

1) VMPools. It specifies a list of physical servers that can be turned into VM pools as well as the access points of them for the potential system.

2) Network setup of potential system. In an explicit requirement, users also need to specify how the network of their potential system should be set up. Such information includes whether the system needs to provide internet access to its end users; whether the system allows its users to communicate with each other; whether the end users of the system need to publish their services to the public, etc.

3) Atomic services and the coordination descriptions for them. Based on the functions that the potential system needs to provide, the system composer has to select atomic services from the service pool. Also, he/she needs to produce the coordination descriptions for those services to provide various functions.

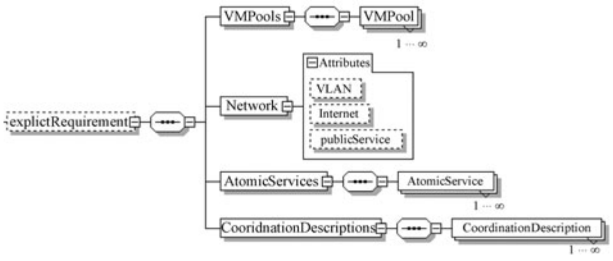


Fig. 11 Properties of explicit requirement

3.2.5.2 Implicit requirement

Users are sometimes less knowledgeable about how to build a cloud system or may not have that knowledge at all. Therefore, instead of submitting explicit requirements, they can submit high-level requirement descriptions that describe what the systems should be rather than how they can be implemented. A concrete cloud composition solution is automatically generated based on the given requirements. Fig. 12 shows the properties that are defined for an implicit requirement document.

1) Type and UserPopulation. These two attributes indicate whether the potential cloud system is a public cloud or a private cloud and how many users the providers intend to serve. Using the combination of them, we could roughly infer the amount of computing resources that are required.

Table 1 Cloud specific LCC messages

Messages	Categories	Semantics	Parameters
Upload	Storage service	Upload item to storage service	Image bundle information
Retrieve	Storage service	Retrieve item from storage service	Image bundle ID
createStorage	Storage service	Create a virtual storage space	Storage requirement description
attachVD	Storage service	Attach a virtual disk to a VM	Virtual disk information
Bundle	Storage service	Image bundle copied to destination	Image bundle ID
Unblock	Communication control service	Open firewall block for VM	VM ID
Routing	Communication control service	Route network packages to all given VM list	VM IDs
Create	VM management service	Create a new VM	VM creation requirement
setResource	VM management service	Adjust resource setup	New resource requirements
Delete	VM management service	Delete an existing VM	VM ID
Email	General	Send email to an user	Textual information

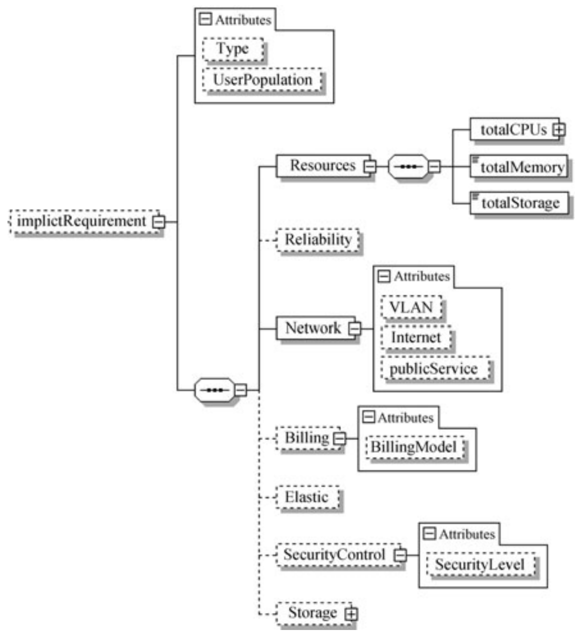


Fig. 12 Properties of implicit requirement

2) A total amount of computing resources that the potential system requires. Once receiving this information, the IC Cloud system allocates a number of available VM pools which, in total, meet the resource requirement of the requestor. This information, if provided, will overwrite the estimated resource information derived from the type and user population properties.

3) Billing. Providers are able to specify how they want to charge their users for the cloud service consumption. Once this property is ticked, IC Cloud is able to generate a full billing and payment solution as a part of the final system, of course, through the generation of coordination descriptions.

4) Elastic. It specifies whether a cloud provider needs the elastic feature in his/her system. This feature affects how VMs are provided once the system is composed. VMs are treated and created differently if they are used to scale an existing VM.

5) SecurityControl. There are two levels of security control defined in IC Cloud, namely, causal and intensive. The former only requires user account based verification and the latter plugs in certificate based verification while interacting with the security service.

6) Storage. This property specifies how much storage space a cloud provider may need and how storage space is distributed and used in the generated cloud system.

4 Composing a cloud

Composing a cloud requires four main steps in IC Cloud as shown in Fig. 13.

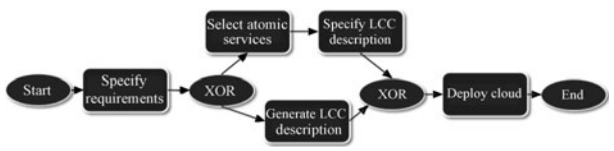


Fig. 13 Workflow for composing a cloud

A cloud provider needs to specify his requirements for the potential cloud system at the beginning of the composing process. The requirement, as discussed earlier, can be either explicit or implicit. If the output requirement of the first step is an explicit one, the following steps are "select atomic services" and "specify LCC descriptions". Otherwise, it "generates LCC descriptions". Once the LCC descriptions are generated, they will be executed at run time to provide cloud services. We will work through a concrete example to demonstrate how a cloud can be composed.

Department of Computing of Imperial College London (DoC) tries to implement an experimental and Amazon like cloud infrastructure. However, the aim is to maximise the use of computing resources rather than making profit. The basic services that they need for the initial stage are:

1) On-demand VM provision,

2) Network management (mainly virtual local area network (VLAN) management),

3) Attaching virtual disk that provides on-demand storage for each VM.

After carrying careful study of the current resource usage pattern, technicians from the computing support group give their explicit cloud construction requirement shown below.

```
<Requirement>
<explictRequirement>
<vmpools>
<vmpool>
http://barking01.doc.ic.ac.uk:/ICCloud/vmsService?WSDL
</vmpool>
<vmpool>
http://barking02.doc.ic.ac.uk:/ICCloud/vmsService?WSDL
</vmpool>
<vmpool>
http://barking03.doc.ic.ac.uk:/ICCloud/vmsService?WSDL
</vmpool>
<vmpool>
http://barking04.doc.ic.ac.uk:/ICCloud/vmsService?WSDL
</vmpool>
</vmpools>
<network publishService="yes" internet="yes" vlan="yes"/>
<AutomicServices>
<VMPoolManagementService/>
<StorageService/>
<VMManagementService/>
<CommunicationControlService/>
</AutomicServices>
<CoordinationDescriptions>
<CoordinationDescription>
http://iccloud.ic.ac.uk/doc/VMProvision/vmp1.xml
</CoordinationDescription>
<CoordinationDescription>
http://iccloud.ic.ac.uk/doc/vlan/vlan1.xml
</CoordinationDescription>
<CoordinationDescription>
http://iccloud.ic.ac.uk/doc/storage/vdisk1.xml
</CoordinationDescription>
</CoordinationDescriptions>
</explictRequirement>
</Requirement>
```

The <VMPools> element in the above requirement specifies the available servers that can be used in the potential cloud and their access points. <Network> element indicates that the control of the internet access and virtual Lan management are needed as well as the ability to publish services to the public. The selected atomic services are "VM pool management service", "storage service", "VM management service", and "communication control service". Three LCC descriptions are attached to the requirement document, which realise the basic functions that DoC cloud requires.

Fig. 14 shows how VM instance provisions should be performed for the DoC cloud. A user has to upload his own image bundle (normally contains all his applications, data, and OS) to the cloud before deploying it as a VM. He will then make a VM provision request to the VM pool management service. On receiving this request, the VM pool management service selects a VM pool and informs the VM management service that is deployed on it. The VM management service will retrieve the user uploaded image bundle using the storage service and create a VM. After the whole process is completed, it sends the VM access point to the requesting user.

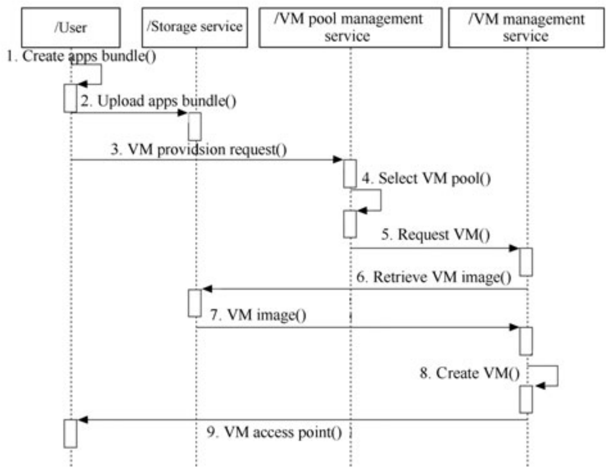


Fig. 14 VM instance provision

The corresponding LCC description is given below. For the sake of readability, all the LCC descriptions will be presented in plain text rather than their original XML format.

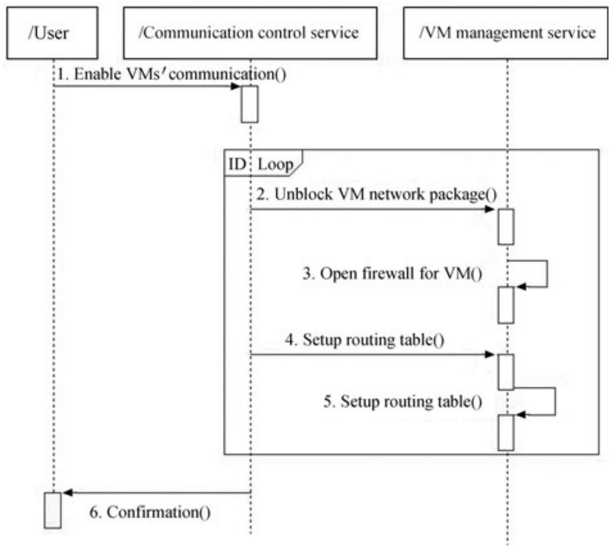
$$
\begin{array}{l}
a(user, U_1) :: \\
\left\{\begin{array}{l}
request(\text{“}create\text{”}, VMDes, BundleID) \Rightarrow a(vps, A_2) \\
\qquad \leftarrow exist(BundleID) \\
then \\
email(VMAccessInfor) \Leftarrow a(vms, A_3)
\end{array}\right\}_T \\
or \\
\left\{\begin{array}{l}
upload(IB_1) \Rightarrow a(ss, S_1) \leftarrow createBundle(IB_1) \\
then \\
confirm(BundleID) \Leftarrow a(ss, S_1)
\end{array}\right\}_{T1}
\end{array}
$$

$$
\begin{array}{l}
a(ss, S_1) :: \\
\left\{\begin{array}{l}
upload(IB_1) \Leftarrow a(user, U_1) \\
then \\
confirm(BundleID) \Rightarrow a(user, U_1) \\
\qquad \leftarrow store(IB_1, BundleID)
\end{array}\right\}_{T2} \\
or \\
\left\{\begin{array}{l}
request(\text{“}retrieve\text{”}, BundleID) \Leftarrow a(vms, A_3) \\
then \\
reply(\text{“}bundle\text{”}, IB_1) \Rightarrow a(vms, A_3) \leftarrow \\
fetch(Bundle, IB_1)
\end{array}\right\}_{T3}
\end{array}
$$

$$
\begin{array}{l}
a(vps, A_2) :: \\
request(\text{“}create\text{”}, VMDes, BundleID) \Leftarrow a(user, U_1) \\
then \\
request(\text{“}create\text{”}, VMDes, BundleID) \Rightarrow a(vms, A_3) \\
\qquad \leftarrow selectVMPool(VMDes, VMPoolList, A_2)
\end{array}
$$

$$
\begin{array}{l}
a(vms, A_3) :: \\
request(\text{“}create\text{”}, VMDes, BundleID) \Leftarrow a(vps, A_2) \\
then \\
\left\{\begin{array}{l}
request(\text{“}retrieve\text{”}, BundleID) \Rightarrow a(ss, S_1) \\
then \\
reply(\text{“}bundle\text{”}, IB_1) \Leftarrow a(ss, S_1)
\end{array}\right\}_{T4} \\
then \\
email(VMAccessInfor) \Rightarrow a(user, U_1) \\
\qquad \leftarrow createVM(VMDes, IB_1).
\end{array}
$$

Based on the requirement, the underlying network of DoC cloud needs to be set up in a way that VMs are distributed across multiple subnets. Fig. 15 shows the diagrammatic representation of VLAN management pattern followed by its LCC description. To enable the communication of a group of VMs, the communication control service will inform the involved VM management services to open the firewall restrictions for the requesting VMs. It then will set the routing table on each VM pool to make sure all VMs know where to find the others.



/User
/Communication control service
/VM management service
1. Enable VMs' communication()
ID Loop
2. Unblock VM network package()
3. Open firewall for VM()
4. Setup routing table()
5. Setup routing table()
6. Confirmation()

Fig. 15 VLAN management

$$
\begin{array}{l}
a(user, A_1) :: \\
request(\text{“}enableCommunication\text{”}, [VM|VMList]) \Rightarrow a(cms, A_2) \\
then \\
confirmation(\text{“}communicationEnabled\text{”}) \Leftarrow a(cms, A_2)
\end{array}
$$

$$
\begin{array}{l}
a(cms, A_2) :: \\
request(\text{“}enableCommunication\text{”}, [VM|VMList]) \Leftarrow a(user, A_1) \\
then \\
\left(\begin{array}{l}
a(cmsL([VM|VMList]), A_2) \\
or \\
confirmation(\text{“}communicationEnabled\text{”}) \Rightarrow a(user, A_1)
\end{array}\right)
\end{array}
$$

$$
\begin{array}{l}
a(cmsL([VM|VMList]), A_2) :: \\
request(\text{“}unblock\text{”}, VM) \Rightarrow a(vms, A_3) selectVMPool(VM, A_3) \\
then \\
request(\text{“}routing\text{”}, VMList) \Rightarrow a(vms, A_3) \\
\qquad \leftarrow selectVMPool(VM, A_3) \\
then \\
a(cmsL(VMList), A_2)
\end{array}
$$

$$
\begin{array}{l}
a(vms, A_3) :: \\
request(\text{“}unblock\text{”}, VM) \Leftarrow a(cmsL, A_2) \leftarrow unblock(VM) \\
or \\
request(\text{“}routing\text{”}, VMList) \Leftarrow a(cmsL, A_2) \leftarrow routing(VMList).
\end{array}
$$

The third LCC description specifies the service that attaches a virtual disk to a VM in order to provide unlimited and on-demand storage space for the VM.

Fig. 16 shows attach virtual disk to VM.

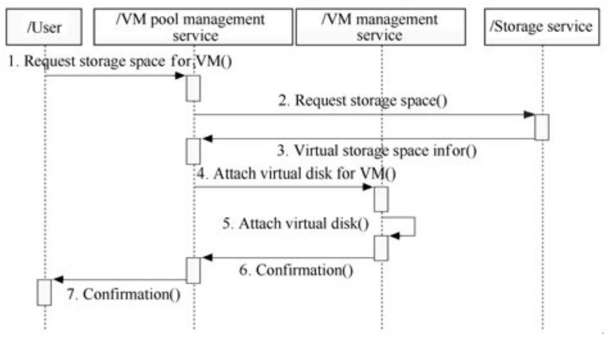

/User
/VM pool management service
/VM management service
/Storage service
1. Request storage space for VM()
2. Request storage space()
3. Virtual storage space infor()
4. Attach virtual disk for VM()
5. Attach virtual disk()
6. Confirmation()
7. Confirmation()

Fig. 16 Attach virtual disk to VM

```
$a(user, A_1)$ ::
  $request(\text{"}attachVD\text{"}, VDReq, VM) \Rightarrow a(vps, A_2)$
  $then$
  $confirmation(\text{"}VDAttched\text{"}, VM) \Leftarrow a(vps, A_2)$

$a(vps, A_2)$ ::
  $request(\text{"}attachVD\text{"}, VDReq, VM) \Leftarrow a(user, A_1)$
  $then$
  $request(\text{"}createStorage\text{"}, VDReq) \Rightarrow a(ss, S_1)$
  $then$
  $confirmation(StorageInfor, VDReq) \Leftarrow a(ss, S_1)$
  $then$
  $request(\text{"}attachVD\text{"}, StorageInfor, VM) \Rightarrow a(rms, A_3)$
      $\leftarrow fetchVMPool(VM, A_3)$
  $then$
  $confirmation(\text{"}VDAttached\text{"}, VM) \Leftarrow a(rms, A_3)$
  $then$
  $confirmation(\text{"}VDAttached\text{"}, VM) \Rightarrow a(user, A_1)$

$a(vms, A_3)$ ::
  $request(\text{"}attachVD\text{"}, StorageInfor, VM) \Leftarrow a(vps, A_2)$
  $then$
  $confirmation(\text{"}VDAttached\text{"}, VM) \Rightarrow a(vps, A_2)$
      $\leftarrow attachVD(StorageInfor, VM)$

$a(ss, S_1)$ ::
  $request(\text{"}createStorage\text{"}, VDReq) \Leftarrow a(vps, A_2)$
  $then$
  $confirmation(StorageInfor, VDReq) \Rightarrow a(vps, A_2).$
```

Fig. 17 shows the hierarchical system architecture of the composed DoC cloud.

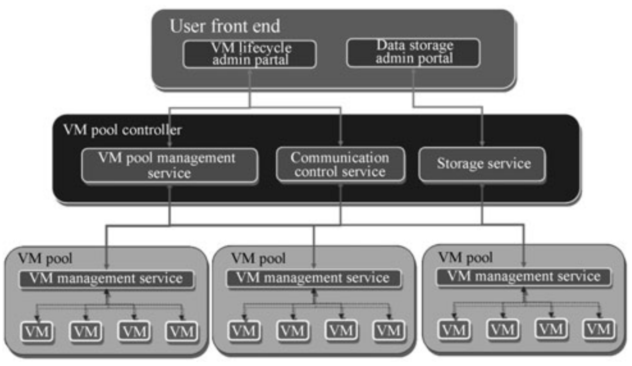


Fig. 17 Architecture of the composed DoC cloud

With the DoC cloud implementation, the three selected atomic services in Section 4 (VM pool management service, communication control service, and storage service) are deployed on one physical server. Several VM pools are set up for resource provision with each having a VM management service deployed.

Users of the DoC cloud interact with the system through two portals which are VM lifecycle admin portal and data storage admin portal. Once a user submits a VM creation request to the DoC cloud, a copy of VM instance provision description is generated and passed to the VM pool management service. The whole system then acts following the instructions specified in the LCC description. To enable the communication of different VMs in the system, users only need to submit a VLAN management description through the user front end. By using the data storage admin portal, users are able to attach virtual disks on their VMs and these virtual disks provide them with unlimited storage through the storage service.

The DoC cloud can be easily scaled up. If there are not enough resources, we can put a new VM pool into the system simply by deploying VM management services on a new server. There could be more than one VM pool controller too in order to improve the overall system performance.

5 Conclusions

In this paper, we declared that due to the evolving speed and the complexity of cloud computing, cloud providers should be very careful when they use the pre-designed approach for cloud system construction. We proposed a composition based approach for building cloud computing systems through the detailed demonstration of IC Cloud.

We developed a set of atomic services which are generic and provide basic functions for composing a cloud system. We then introduced a coordination mechanism and extended it to pull all these services together and specify how they should work as an integrated system. We also gave a concrete example showing how a cloud system can be composed. Comparing with the others, the main contribution of our approach is that it adopts a fully flexible and agile system architecture and sets up a well-formed design space for carrying out systematic studies and experiments for cloud computing.

The next step that we are going to take is to deploy different applications on the IC Cloud to further test the system performance. We, especially, aim to deploy it for the academic usage such as physical/chemical experiments. The immediately following work afterwards is to build up a customised system performance tuning mechanism based on the quality of service (QoS) information that is gathered for each user through his/her use of the system. With such a mechanism, the system should be able to dynamically detect and predicate the upcoming system bottlenecks and adjust them accordingly. In addition, we plan to experiment and adopt more scheduling algorithms as it is still unclear at the moment which one is the best or is more appropriate for academic cloud.

References

[1] Future of Cloud Computing, EU 2010.

[2] Virtualbox Home Page, [Online], Available: http://www.

virtualbox.org/, June 4, 2011.

[3] Vmware, [Online], Available: http://www.vmware.com/, June 4, 2011.

[4] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, A. Warfield. Xen and the art of virtualization. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles*, ACM, New York, USA, pp. 164–177, 2003.

[5] T. Deshane, M. Ben, A. Shah, B. Rao. Quantitative comparison of Xen and KVM. In *Proceedings of the Xen Submit*, 2008. [Online], Available: http://wiki.xensource.com/xenwiki/Open_Topics_For_Discussion?action=AttachFile&do=get&target=Quantitative+Comparison+of+Xen+and+KVM.pdf, June 13, 2011.

[6] A. Kivity, Y. Kamay, D. Laor, U. Lublin, A. Liguori. KVM: The linux virtual machine monitor. In *Proceedings of the Linux Symposium*, pp. 225–230, 2007. [Online], Available: http://www.kernel.org/doc/ols/2007/ols2007v1-pages-225-230.pdf, June 13, 2011.

[7] Libvirt virtualization API, [Online], Available: http://libvirt.org/ index.html, June 4, 2011.

[8] Gogrid: Build the most complete infrastructure in the cloud, [Online], Available: http://www.gogrid.com/, June 4, 2011.

[9] The rackspace cloudhosting, [Online], Available: http://www.rackspace.co.uk/rackspace-home/, June 4, 2011.

[10] OpenNebula project, [Online], Available: http://www.opennebula.org/, June 4, 2011.

[11] Reservoir project, [Online], Available: http://www.reservoir-fp7.eu/, June 4, 2011.

[12] Google App Engine, [Online], Available: http://appengine.google.com/, June 4, 2011.

[13] Microsoft Azure, [Online], Available: http://www.microsoft.com/azure/, June 4, 2011.

[14] ORACLE, [Online], Available: http://www.oracle.com/us/sun/index.htm.

[15] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, D. Zagorodnov. The eucalyptus open-source cloud-computing system. In *Proceedings of the 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, ACM, Washington, USA, pp. 124–131, 2009.

[16] T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic, S. Weerawarana. Business process execution language for web services version 1.1, [Online], Available: http://public.dhe.ibm.com/software/dw/webservices/ws-bpelsubproc/ws-bpelsubproc.pdf, May 2003.

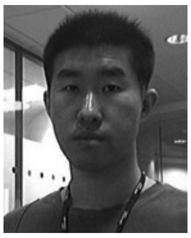
[17] D. Martin, M. Burstein, J. Hobbs, O. Lassila. Owl-S Technical Report, [Online], Available: http://www.w3.org/Submission/OWL-S/.

[18] D. Roberston. A lightweight method for coordination of agent oriented web services. In *Proceedings of AAAI Spring Symposium on Sematic Web Services*, 2004. [Online], Available: http://www.aaai.org/Papers/Symposia/Spring/2004/SS-04-06/SS04-06-011.pdf, June 13, 2011.

Yi-Ke Guo graduated in computer science from Tsinghua University, PRC and received Ph. D. degree in computational logic and declarative programming at Imperial College London. He has been working in the area of data intensive analytical computing since 1995 when he was the technical director of Imperial College Parallel Computing Centre. He is currently a professor in computing science in the Department of Computing, Imperial College London, UK. During the last 10 years, he has been leading the data mining group of the department to carry out many research projects, including some major UK e-science projects such as discovery net on grid based data analysis for scientific discovery, MESSAGE on wireless mobile sensor network for environment monitoring, Biological Atlas of Insulin Resistance (BAIR) on system biology for diabetes study. He has been focusing on applying data mining technology to scientific data analysis in the fields of life science and healthcare, environment science and security.

His research interests include large scale scientific data analysis, data mining algorithms and applications, parallel algorithms, and cloud computing.

E-mail: yg@doc.ic.ac.uk (Corresponding author)



Li Guo received Ph. D. degree in artificial intelligence at the University of Edinburgh, UK. He has been working in the area of grid computing and cloud computing since 2006. He is currently a research associate in computing science in the Department of Computing, Imperial College London, UK. During the last 5 years, he has been involved in major grid and cloud related EU and UK projects. He is the chief architect of Imperial College Cloud platform.

His research interests include large scale distributed system, intelligent applications, and cloud computing.

E-mail: liguo@doc.ic.ac.uk