Loewen *et al. Journal of Cloud Computing: Advances, Systems and Applications* 2013, **2**:24
http://www.journalofcloudcomputing.com/content/2/1/24

Journal of Cloud Computing
a SpringerOpen Journal

RESEARCH **Open Access**

THUNDER: helping underfunded NPO's distribute electronic resources

Gabriel Loewen*, Jeffrey Galloway, Jeffrey Robinson, Xiaoyan Hong and Susan Vrbsky

Abstract

As federal funding in many public non-profit organizations (NPO's) seems to be dwindling, it is of the utmost importance that efforts are focused on reducing operating costs of needy organizations, such as public schools. Our approach for reducing organizational costs is through the combined benefits of a high performance cloud architecture and low-power, thin-client devices. However, general-purpose private cloud architectures are not easily deployable by average users, or even those with some computing knowledge. For this reason, we propose a new vertical cloud architecture, which is focused on ease of deployment and management, as well as providing organizations with cost-efficient virtualization and storage, and other organization-specific utilities. We postulate that if organizations are provided with on-demand access to electronic resources in a way that is cost-efficient, then the operating costs may be reduced, such that the user experience and organizational efficiency may be increased. In this paper we discuss our private vertical cloud architecture called THUNDER. Additionally, we introduce a number of methodologies that could enable needy non-profit organizations to decrease costs and also provide many additional benefits for the users. Specifically, this paper introduces our current implementation of THUNDER, details about the architecture, and the software system that we have designed to specifically target the needs of underfunded organizations.

Introduction

Within the past several years there has been a lot of work in the area of cloud computing. Some may see this as a trend, whereas the term "cloud" is used simply as a buzzword. However, if viewed as a serious contender for managing services offered within an organization, or a specific market, cloud computing is a conglomerate of several very desirable qualities. Cloud computing is known for being scalable, which means that resource availability scales up or down based on need. Additionally, cloud computing represents highly available and on-demand services, which allow users to easily satisfy their computational needs, as well as access any other required services, such as storage and even complete software systems. Although there is no formal definition for cloud computing, we define cloud computing as a set of service-oriented architectures, which allow users to access a number of resources in a way that is elastic, cost-efficient, and on-demand. General cloud computing can be separated into three categories: Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS), and Software-as-a-Service (SaaS). Infrastructure-as-a-Service provides access to virtual hardware and is considered the lowest service layer in the typical cloud stack. An example of Infrastructure-as-a-Service is the highly regarded Amazon EC2, which is subsystem of Amazon Web Services [1]. At the highest layer is Software-as-a-Service, which provides complete software solutions. An example software solution, which exists as a cloud service is Google Docs. Google Docs is a SaaS which gives users access to document editing tools, which may be used from a web browser. In between SaaS and IaaS is Platform-as-a-Service, which allows users to access programming tools and complete API's for development. An example of a PaaS is Google AppEngine, which gives developers access to robust API's and tools for software development in a number of different languages. We are beginning to see many software services being offered by a number of public cloud providers, including image editing software, email clients, development tools, and even language translation tools. However, these tools are all offered by different providers and are not necessarily free for general use.

*Correspondence: gloewen@crimson.ua.edu
Department of Computer Science, The University of Alabama, Tuscaloosa, AL, USA

Considering that non-profit organizations cannot always afford to purchase access to software, we propose that these organizations should simply maintain their own private cloud, which could decrease the costs associated with software licensing. There are several freely available cloud architectures that may be considered. However, general-purpose cloud architectures are not suitable for organization that do not have highly trained professionals to manage such a system. This downfall of most general-purpose architectures is due to the lack of an easy to use user interface and somewhat complicated deployment process. Many architectures, such as Eucalyptus [2] and OpenStack [3], rely heavily on the command line for interfacing with the system, which isn't desirable for markets that do not have experts readily available for troubleshooting. A cloud architecture designed for these specific markets must have the following attributes: ease of deployment, user friendly interface, energy efficiency, and cost effectiveness. In consideration of these qualities we have designed a new IaaS cloud architecture, which we call THUNDER (THUNDER Helps Underfunded NPO's Distribute Electronic Resources). THUNDER utilizes the notion of simplicity at all levels in order to ensure that all users, regardless of their technical experience, will be able to use the system or redeploy the architecture if necessary.

Most IaaS cloud architectures rely upon the general case model. In the general case, an IaaS cloud architecture supports low-level aspects of the cloud stack, such as hardware virtualization, load balancing of virtual machine instances, elastic storage, and modularity of physical hardware. Vertical clouds, on the other hand, are defined by a specific market, and therefore, are able to abstract the general case IaaS cloud model to provide features that are tailored for a specific set of uses. We see vertical clouds predominantly in the healthcare sector with the e-health cloud architecture. The THUNDER architecture is an abstraction of the general case model by taking care of the low-level details of hardware virtualization, load balancing, and storage in a way that is considerate of the technical maturity of the users, as well as the level of expertise expected from the administrators. This abstraction is possible in a vertical cloud designed for the non-profit sector because we can make an assumption about the maximum number of virtual machines, the type of software required, and the expected level of experience of the users. We assume the number of virtual machine instances is congruent to the number of client devices in an office or computer lab. Additionally, the software available on the cloud is defined by a set of use cases specific to the organization. For example, THUNDER deployed to a school may be used in conjunction with a mathematics course, which would be associated with a virtual machine image containing mathematics software, such as Matlab or Maple. Additionally, we assume that the technical experience of administrators and instructors in a school setting is low. Therefore, by deviating from the general case model of an IaaS cloud architecture, and by considering the special needs of the market, we can minimize the complexity of deployment by removing the necessity of a fine-tuned configuration.

In the following sections we discuss related background work in private vertical cloud architectures, our proposed architecture, future work, and we end with a summary and conclusion.

Background and motivation

There has been much discussion on the topic of cloud computing for various administrative purposes at educational institutions. However, cloud computing is a topic that until recently has not been widely considered for the high school grade bracket. Due to the nature of cloud computing, being a service oriented architecture, there is a lot of potential in adopting a cloud architecture that can be used in a classroom [4]. Cloud computing in the classroom could be used to provide valuable educational tools and resources in a way that is scalable, and supportive of the ever-changing environment of the classroom. Production of knowledgeable students is not a trivial task. Researchers in education are focused on providing young students with the tools necessary to be productive members of society [4]. The past decade has seen, in some cases, a dramatic decrease in state and local funding for public secondary education. This reduction in funding indicates that a paradigm shift in how technology is utilized in the classroom is necessary in order to continue to provide high quality education. The authors of [4-7] believe that cloud computing may be a viable solution to recapture students' interests and improve student success.

Education

Researchers at North Carolina State University (NCSU) have developed a cloud architecture, which is designed to provide young students with tools that help to engage students in the field of mathematics [4]. This cloud architecture, known as "Virtual Computing Lab" or "VCL", has been provided as a public service to rural North Carolina 9th and 10th grade algebra and geometry classes. The goal of this study is to broaden the education of STEM related topics using the VCL in these schools, and two applications were selected to be used in the course curriculums: Geometer's Sketchpad 5, and Fathom 2. The authors describe a set of key challenges that were encountered during the study, including: diversity of software, software licensing, security, network availability, life expectancy of hardware, affordability, as well as technical barriers. Software availability is a prime concern when it comes to provisioning educational tools for academic use.

The specific needs of the classroom, in many cases, require specific software packages. When deploying software to a cloud architecture, it is not always possible to provide certain software packages as cloud services. For this reason, it is common to bundle software with virtual machine images, which are spawned on an IaaS cloud. A virtual machine image is a single file that contains a filesystem along with a guest operating system and software packages. Additionally, software packages may have some conflicts with one another that can create an issue with the logistics of the system [4]. Another software concern is related to software-specific licensing, and how it affects the cloud. Many software packages require licensing fees to be paid per user of the system, or as a volume license, which may or may not impose a maximum number of users allowed access to the software. Therefore, depending on the specific requirements of the school and course, software licensing fees must be paid for accordingly. For example, when geometers sketchpad was deployed to the VCL, the authors made sure that the software licensing fees were paid for in accordance with the software publishers' license agreement. The necessity for licensing does affect the cost effectiveness of using a cloud in this setting, however it is no different than licensing software for traditional workstations [4].

The authors of [8] have created a private cloud architecture, called CloudIA, which supports e-Learning services at every layer of the cloud stack. At the IaaS layer, the CloudIA architecture supports an automated virtual machine image generator, which utilizes a web interface for creating custom virtual machine images with predefined software packages installed. At the PaaS layer, the CloudIA architecture supports computer science students with a robust API for writing software that utilizes cloud services. At the SaaS layer, the CloudIA architecture supports collaborative software for students to utilize for projects and discussion.

The authors of [9] describe the benefits of cloud computing for education. The main point that the authors make is that cloud computing provides a flexible and cost effective way to utilize hardware for improving the way information is presented to students. Additionally, the authors describe details about the ability of cloud computing to shift the traditional expenses from a distributed IT infrastructure model to a more pay-as-you-go model, where services are paid for based on specific needs.

Authors of [5] discuss "Seattle", which is a cloud application framework and architecture, enabling users to interact with the cloud using a robust API. By using this platform students can execute experiments for learning about cloud computing, networking, and other STEM topics. The authors also describe a complimentary programming language built upon Python, which gives students easy access to the Seattle platform.

The authors of [10] discuss a new model for SaaS, which they have named ESaaS. ESaaS is defined as a Software-as-a-Service cloud architecture with a focus on providing educational resources. The authors discuss the need for a managed digital library and a global repository for educational content, which is easily accessible through a web interface. The proposed architecture is meant to integrate into existing secondary and post-secondary institutions as a supplementary resource to their existing programs.

LTSP

One approach is the use of thin client devices, which have been used in other educational endeavors, such as the Linux Terminal Server Project (LTSP) [11]. Thin client solutions, when paired with an IaaS cloud, offer low power alternatives to traditional computing infrastructures. The authors of [12] analyze energy savings opportunities in the thin-client computing paradigm.

Authors of [13] discuss design considerations for a low power and modular cloud solution. In this study the LTSP [11] architecture is reviewed and compared to the authors cloud architecture design. LTSP is a popular low power thin client solution for accessing free and open source Linux environments using a cluster of server machines and thin client devices. The LTSP architecture provides services, which are very similar to an IaaS cloud architecture with a few notable limitations. Firstly, LTSP only offers Linux environments, which differs from an IaaS cloud in that the cloud can host Linux, Windows, and in some instances Apple OSX virtual machine instances. Additionally, LTSP does not utilize virtualization technology, rather it provides several minimal Linux and X windows environments on the same host computer. Interfacing with an LTSP instance also differs from an IaaS cloud in that an LTSP terminal will boot directly from the host machine using PXE or NetBoot, which is a remote booting protocol. A client connected to an IaaS cloud will typically rely upon the Remote Desktop Protocol (RDP) for accessing Windows instances, or the Virtual Network Computing (VNC) protocol for Linux instances.

Other work

All of the previous work relate to educational resources and services in the cloud. However, most of the related work is integrated using public cloud vendors and is specific towards one particular subject, as is presented in [4] and [5]. The authors of [14] present their solution, SQRT-C, which is a light-weight and scalable resource monitoring and dissemination solution using the publisher/subscribe model, similar to what is described in this manuscript. The approach considers three major design implementations as top priority: Accessing physical resource usage in a virtualized environment, managing data distribution service (DDS) entities, and shielding cloud users from complex DDS QoS configurations. SQRT-C can be deployed seamlessly in other cloud platforms, such as Eucalyptus and OpenStack, since it relies on the libvirt library for information on resource management in the IaaS cloud.

In [15], the authors propose a middleware for enterprise cloud computing architectures that can automatically manage the resource allocation of services, platforms, and infrastructures. The middleware API set used in their cloud is built around servicing specific entities using the cloud resources. For end users, the API toolkit provides interaction for requesting services. These requests are submitted through a web interface. Internal interface APIs communicate between physical and virtual cloud resources to construct interfaces for users and determine resource allocation. A service directory API is provided for users based on user privileges. A monitoring API is used to monitor and calculate the use of cloud system resources. This relates to the middleware introduced in this manuscript; however, it addresses architectures more suitable for large enterprises.

The authors of [16] propose a resource manager that handles user requests for virtual machines in a cloud environment. Their architecture deploys a resource manager and a policy enforcer module. First, the resource manager decides if the user has the rights to request a certain virtual machine. If the decision is made to deploy the virtual machine, the policy enforcer module communicates with the cloud front-end and executes an RPC procedure for creating the virtual machine.

Authors of [17] describe how cloud platforms should provide services on-demand that helps the user complete their job quickly. Also mentioned is the cloud's responsibility of hiding low-level technical issues, such as hardware configuration, network management, and maintenance of guest and host operating systems. The cloud should also reduce costs by using dynamic provisioning of resources, consuming less power to complete jobs (within the job constraints), and by keeping human interaction to cloud maintenance to a minimum.

Development of cloud APIs is discussed in [18]. The author mentions three goals of a good cloud API: Consistency, Performance, and Dependencies. Consistency implies the guarantees that the cloud API can provide. Performance is relatively considered in forms of decreasing latency while performing actions. Cloud dependencies are other processes that must be handled, other than spawning virtual machines and querying cloud resource and user states. These three issues are considered in the development process of our own IaaS cloud architecture.

Proposed architecture

Our focus is to provide underfunded non-profit organizations with the means to facilitate the computing needs of their users in a cost-effective manner. The THUNDER architecture is composed of a special purpose private cloud stack, and an array of low power embedded systems, such as Raspberry Pi's [19] or other low-power devices. The THUNDER stack differs from the general-purpose private cloud model in a number of ways. General purpose cloud stacks, such as Eucalyptus [2] and OpenStack [3], are focused on providing users with many different options as to how the cloud can be configured. These general-purpose solutions are great for large organizations because the architecture is flexible enough to be useful for diverse markets. However, non-profit organizations do not typically have the resources to construct a general-purpose cloud architecture. Therefore, a special-purpose or vertical cloud architecture is desirable because it circumvents the typical cloud deployment process by making assumptions about the use of the architecture. THUNDER may be utilized by various NPO's and for various purposes, but a secondary focus of THUNDER is focused on the education market. Research in cloud computing for education has shown that educational services in high school settings are successful in motivating students to learn and achieve greater success in the classroom [4].

The THUNDER cloud stack utilizes a number of commodity compute nodes, in addition to persistent storage nodes with a redundant backup, as well as a custom DHCP, MySQL, and system administration server. Each compute node is capable of accommodating four Windows virtual machines or twelve Linux virtual machines. The lab consists of low-power client devices with a keyboard, mouse, and monitor connected to a gigabit network. A custom web-based interface allows users to login, select their desired virtual machine from a list of predefined images, and then launch the virtual machine image. For example, students taking a course in Python programming might be required to use a GNU/Linux based computer for development. However, a receptionist in an office setting might be required to use a Microsoft Windows system. Therefore, regardless of the user requirements, THUNDER will be able to provide all necessary software components to each user independently. Figure 1 illustrates the THUNDER network topology. The THUNDER network topology resembles a typical cloud topology, where the compute cluster is connected to a single shared LAN switch, and support nodes share a separate LAN switch. Additionally, the topology shows the client devices and how they interface with the rest of the system. Table 1 shows a power cost comparison between THUNDER and a typical 20 PC lab, and shows a possible savings of 50% when compared to a traditional computer lab.

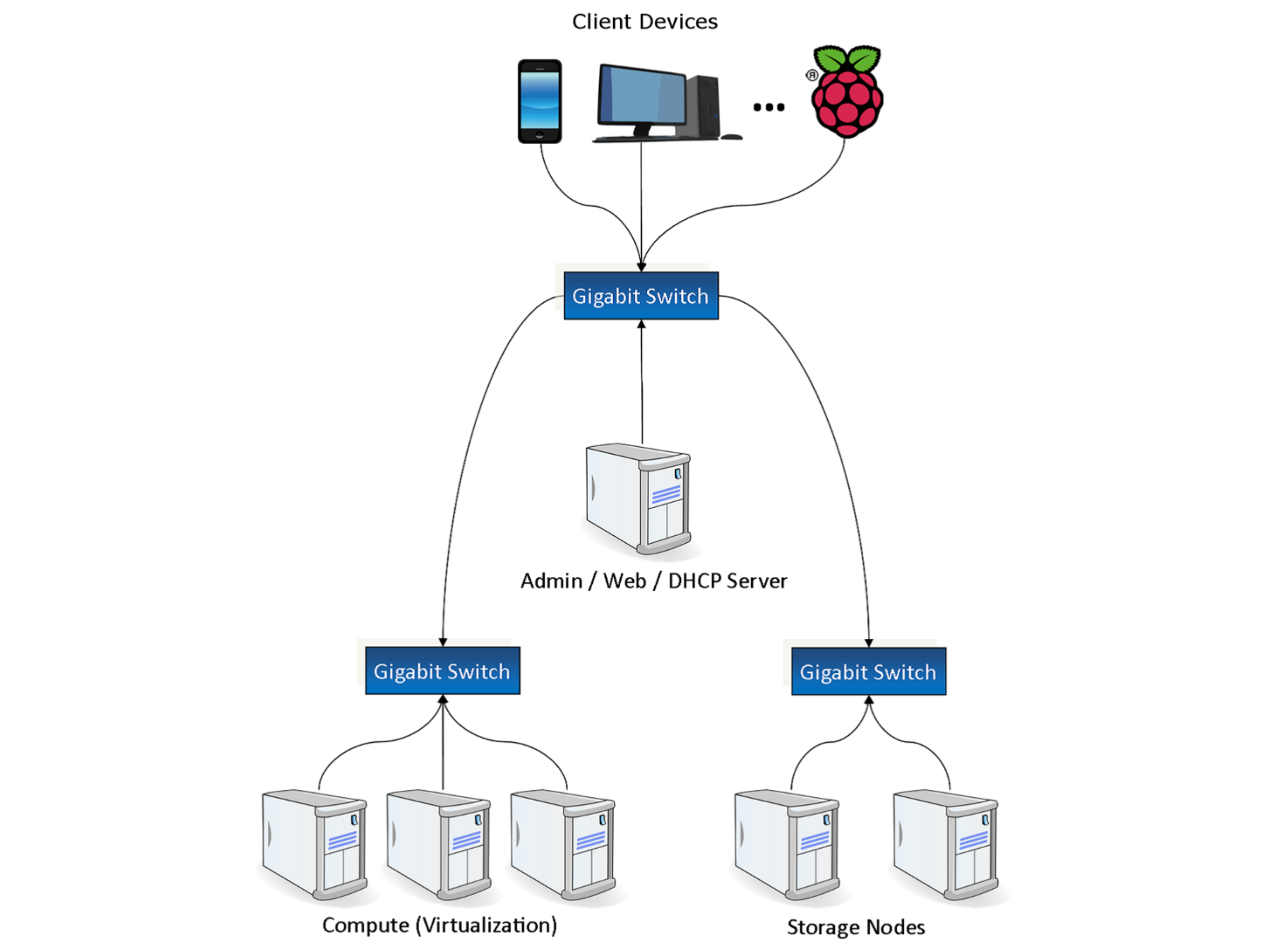


Figure 1 Networking topology for the THUNDER cloud architecture.

Network topology

The THUNDER network topology in Figure 1 is most cost efficient when combined with low-power or thi–client devices, but can also be paired with regular desktop and laptop computers. One of the common advancements in wired Ethernet technology is the use of switches.

Table 1 Power cost comparison between THUNDER and a typical lab

Typical lab			THUNDER		
#	Hardware	Watts	#	Hardware	Watts
20	PC Desktops	6,000	20	Thin client	60
20	Display	2,000	20	Display	2,000
			3	Compute node	1,200
			2	Storage node	500
			1	Admin/Web	250
Total		8,000	Total		4,010
Monthly bill: $152			Monthly bill: $77		

Ethernet switches allow for adjacent nodes connected to the switch to communicate simultaneously without causing collisions. The network interface cards used in all of the devices of THUNDER support full duplex operation, which further allows nodes to send and receive data over the network at the same time. Ethernet is a Link-Layer protocol, which determines how physical devices on the network communicate. The clients communicate with THUNDER through simple socket commands and a virtual desktop viewing client, such as VNC or RDP viewers.

Compute and store resources

THUNDER compute and storage resources will consume a considerable amount of network bandwidth. The compute nodes are responsible for hosting virtual machines that are accessed by the clients. These compute nodes will mount the user's persistent data as the virtual machine is booting. Each compute node will communicate with the THUNDER cloud resources and the client devices using a 1 Gbps network interface. The client devices should be equipped with a 10/100/1000 Mbps network adapter, and considering the limited number of cloud servers, it is unlikely that the 1 Gbps network switch will become completely saturated with traffic. Userspace storage nodes are connected to the same switch as the compute nodes, which allows for tighter coupling of storage nodes and compute nodes, decreasing the potential delay for persistent file access. Backup storage nodes are connected to a separate network switch, and are used to backup the cloud system in case of a system failure.

Administrative resources

The THUNDER administrative resources include a MySQL database, Web interface, and Networking services. These services are hosted on a single physical machine, with a backup machine isolated to the same network switch. There will be no need for a high amount of resources in the administrative node since the numbers of compute and storage nodes determine the amount of clients that can be connected to THUNDER. The THUNDER cloud, when accessed from devices external to the organization's private network, can be routed to a secondary administrative node, such that the on-site users will not experience any quality of service issues.

Network provisioning for low latency

Using a top down approach, the amount of bandwidth (maximum) needed for a twenty node THUNDER lab can be described. If we assume that each client device requires a sustained 1 Mbps network throughput, we would need to accommodate for sustaining 20 Mbps within the network switch used by the clients. Since the network switch is isolated to communicating with the resources of THUNDER, this throughput needs to be sustainable on the uplink port. This is relatively easy, given the costs of gigabit switches on the market today. The specification that needs close attention is the total bandwidth of the switch backplane. Making the assumption that this bandwidth is the number of ports multiplied by the switch speed is not always true. In our case, the bandwidth needed, 20 Mbps, is much lower than the maximum throughput of a twenty-four port gigabit switch.

There is little to no communication between THUNDER compute node resources. These devices are used to host virtual machines that are interfaced to the clients directly. Given a THUNDER lab size of twenty clients, the network bandwidth needed on the isolated network containing the compute nodes should be above 20 Mbps, assuming each client consumes 1 Mbps of bandwidth.

The THUNDER storage node resources are also isolated to the same gigabit network switch as the compute node resources. When the user logs into THUNDER and requests a virtual machine, their persistent storage is mounted inside the virtual machine for them to use. The data created by the users has to be accessed while they are using a virtual machine.

Middleware design and implementation

One of the core components in building a cloud architecture is the development of a middleware solution, allowing for ease in resource management. Additionally, in order to improve the quality of service (QOS) an emphasis on minimizing resource utilization and increasing system reliability is desirable. Our reasoning for developing a new cloud middleware API is to address issues that we have encountered in current cloud middleware solutions, which are centered upon ease of deployment and ease of interfacing with the system. Additionally, we have utilized our API to build a novel cloud middleware solution for use in THUNDER. Specifically, this middleware solution is designed for management of compute resources, including instantiation of virtual machine images, construction and mounting of storage volumes, metadata aggregation, and other management tasks. We present the design and implementation for our cloud middleware solution and we introduce preliminary results from our study into the construction of THUNDER, which is our lightweight private vertical IaaS cloud architecture.

Management of resources is a key challenge in the development of a cloud architecture. Moreover, there is a necessity for minimizing the complexity and overhead in management solutions in addition to facilitating attributes of cloud computing, such as scalability and elasticity. Another desirable quality of a cloud management solution is modularity. We define modularity as the ability to painlessly add or remove components on-the-fly without the necessity to reconfigure any services or systems. The field of cloud management exists within several overlapping domains, which include service management, system deployment, access control management, and others. We address the requirements of a cloud management middleware API, which is intended to support the implementation of the private cloud architecture currently in development. Additionally, we compare our cloud management solution to solutions provided by freely available private IaaS cloud architectures.

When examining the current state of the art in cloud management, there are few options. We are confined to free and open source (FOSS) cloud implementations, such as Eucalyptus [2] and Openstack [3]. Cloud management solutions used in closed-source, and often more popular cloud architectures, such as Amazon EC2, are out of reach from an academic and research perspective due to their closed nature. However, there has been an effort to make Eucalyptus and Openstack compatible with Amazon EC2 by implementing a compatible API and command line tools, such as eucatools [20] and Nova [21], respectively. The compatibility of API's makes it easy to form a basis of comparison between different architectures. Although, this compatibility may also serve as a downfall because if one API suffers from a bug, it may also be present in other API's.

Eucalyptus discussion

The methodology for management of resources in Eucalyptus is predominantly reliant upon establishing a control structure between nodes, such that one cluster is managed by one second-tier controller, which is managed by a centralized cloud controller. In the case of Eucalyptus, there are five controller types: cloud controller, cluster controller, block-based storage controller (EBS), bucket-based storage controller (S3), and node controller. The cloud controller is responsible for managing attributes of the cloud, such as the registration of controllers, access control management, as well as facilitating user interaction through command-line and, in some cases, web-based interfacing. The cluster controller is responsible for managing a cluster of node controllers, which entails transmission of control messages for instantiation of virtual machine images and other necessities required for compute nodes. Block-based storage controllers provide an abstract interface for creation of storage blocks, which are dynamically allocated virtual storage devices that can be utilized as persistent storage. Bucket-based storage controllers are not allocated as block-level devices, but instead are treated as containers by which files, namely virtual machine images, may be stored. Node controllers are responsible for hosting virtual machine instances and for facilitating remote access via RDP [22], SSH [23], VNC [24], and other remote access protocols.

OpenStack discussion

Similar to the methodology used by Eucalyptus, OpenStack also maintains a control structure based on the elements present in the Amazon EC2 cloud. OpenStack maintains five controllers: compute controller (Nova), object-level storage (Swift), block-level storage (Cinder), networking controller (Quantum), and dashboard (Horizon). There are many parallels between the controller of OpenStack and the controllers of Eucalyptus. The Nova controller of OpenStack is similar to the node controller of Eucalyptus. Similarly we see parallels between Swift in OpenStack with the bucket-based controller in Eucalyptus, and Cinder in Openstack with the block-based storage of Eucalyptus. There seems to be a discretion in implementation between the highest-level controller in each architecture. OpenStack maintains different controllers for interfacing and network management, while Eucalyptus maintains a single cloud controller combining these functionalities. Additionally, OpenStack does not maintain a higher-level control structure for managing compute components, which is a deviation from the cluster controller mechanism present in Eucalyptus.

Middleware interfacing, communication, and authentication

In developing our middleware solution we encountered challenges regarding the method by which it would interface with the various resources in the cloud. Many different methodologies were considered. However, we decided to use an event-driven mechanism, which is similar to remote procedure calls (RPC).

One of the prime differences in the way Eucalyptus and OpenStack perform management tasks is in the means of communication. Eucalyptus utilizes non-persistent SSH connections between controllers and nodes in order to remotely execute tasks. OpenStack, on the other hand utilizes remote procedure call, or RPC's. In keeping with the methodology introduced by OpenStack and its current momentum in the open source cloud computing community, we utilize an event driven model, which presents a very similar mechanism to that of RPC. However, these two architectures share a common component. They both utilize the libvirt [25] library, which is the same library that we utilize in our architecture.

Additionally, authentication was a challenge because in reducing the complexity of authentication we introduce new possible security threats. Although, we believe the security threats posed by our authentication model are minimal, additional threats could be uncovered during system testing. We believe that this solution is important because we address concerns regarding the overall usage of the cloud architecture, and our initial performance results in Figure 2 show that our middleware performs well when compared to Eucalyptus [2].

Node-to-node communication scheme

In contrast to the methodologies used by Eucalyptus, OpenStack, and presumably Amazon, our cloud middleware API addresses resource management in a simplified and more direct manner. The hierarchy of controllers used in Eucalyptus introduces extra complexity that we have deemed unnecessary. For this reason, our solution utilizes a simple publisher/subscriber model by which compute, storage, and image repository nodes may construct a closed network. The publisher/subscriber system operates in conjunction with event driven programming, which allows events to be triggered over the private network to groups of nodes subscribed to the controller node. Figure 3 shows the logical topology and lines of communication constructed using this model.

In constructing the communication in this manner we are able to broadcast messages to logical groups in order to gather metadata about the nodes subscribed to that group. Message passing is useful for retrieving the status

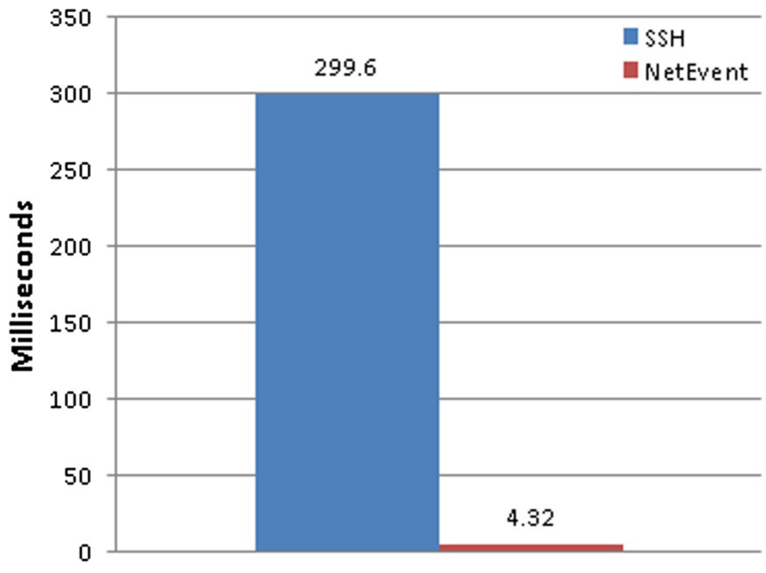


Figure 2 **Performance comparison of NetEvent and SSH authentication protocols.**

of nodes, including virtual machine utilization, CPU and memory utilization, and other details pertaining to each logical group. Additionally, we are able to transmit messages to individual nodes in order to facilitate virtual machine instantiation, storage allocation, image transfer, and other functions that pertain to individual nodes.

Registration of nodes

Communication between nodes utilizes non-persistent socket connections, such that the controller node maintains a static pre-determined port for receiving messages, while other nodes may use any available port on the system. Thus, each node in the cloud, excluding the controller node, automatically selects an available port at boot time. Initial communication between nodes is done during boot time to establish a connection to the controller node. We utilize a methodology for automatically finding and connecting to the controller node via linear search over the fourth octet of the private IP range (xxx.xxx.xxx.0 to xxx.xxx.xxx.255). Our assumption in this case is that the controller node will exist on a predefined subnet that allows us to easily establish lines of communication without having to manually register nodes. Additionally, we can guarantee sequential ordering of IP addresses with our privately managed DHCP server. Once a communication link is established between a node and the controller node, the node will request membership within a specific logical group, after which communication between the controller node and that logical group will contain the node in question.

The registration methodology used in our middleware solution differs from the methodology used by Eucalyptus and OpenStack. For example, Eucalyptus relies upon command line tools to perform RSA keysharing and for

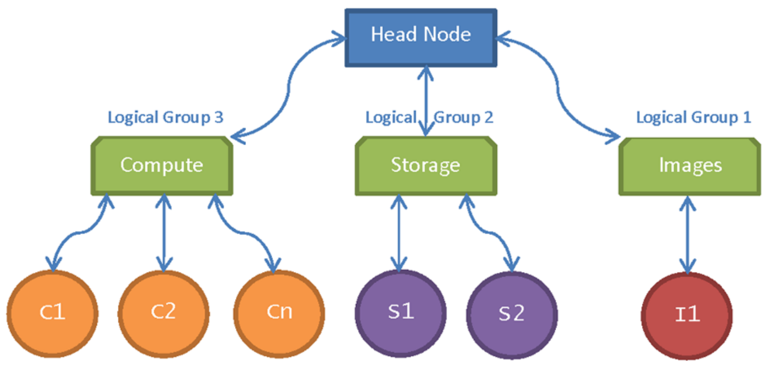


Figure 3 **Logical topology - logical groups represent group-wise membership in publisher/subscriber model.**

establishing membership with a particular controller. We do not perform key sharing, and instead rely upon a pre-shared secret key and generated nonce values. This approach is commonly known as challenge-response [26], and it ensures that nodes requesting admission into the cluster are authentic before communication is allowed. When a node wishes to be registered as a valid and authentic node within a cluster, a nonce value is sent to the originating node. The node will then encrypt the nonce with the pre-shared key and transmit the value back to the controller. We validate the message by comparing the decrypted nonce produced by the receiver and the nonce produced by the sender. Thus, we do not rely upon manual sharing of RSA keys beforehand, and instead we eliminate the need for RSA keys altogether and utilize a more dynamic approach for validation of communication during the registration process. Figure 4 presents the registration protocol.

Middleware API

As stated in the introduction, our methodology for constructing a middleware API for cloud resource management centers around the decreasing overhead when compared to general-purpose solutions. In order to facilitate a simple middleware solution, our API was designed to provide a powerful interface for cloud management while not introducing excessive code overhead. We have titled our API "NetEvent", which is indicative of its intended purpose as an API for triggering events over a network. This API is utilized within our private IaaS cloud architecture as a means for communication, management of resources, and interaction with our cloud interface. Figures 5 and 6 illustrate the manner in which the API is accessed. Although, the code examples presented here are incomplete, they illustrate the simplicity of creating events to be triggered by the system for management of resources.

In Figure 5 we present sample code for the creation of a controller node, which is responsible for relaying commands from the web interface to the cloud servers. In Figure 6 we present a skeleton for the creation of a compute node with events written for instantiation of virtual machine images and for retrieving the status of the node. Although, we do not present code for the implementation of storage or image repository nodes, the implementations are similar to that of the compute node. In addition, the code examples presented in this paper show only a subset of the functionality contained within the production code.

The API presented here provides a powerful interface for implementing private cloud architectures. By means of event triggering over a private network we are able to instantiate virtual machine images, mount storage volumes, retrieve node status data, transfer virtual machine images, monitor activity, and more. The implementation of the system is completely dependent upon the developer's needs and may even be used in distributed systems, which may or may not be implemented as a

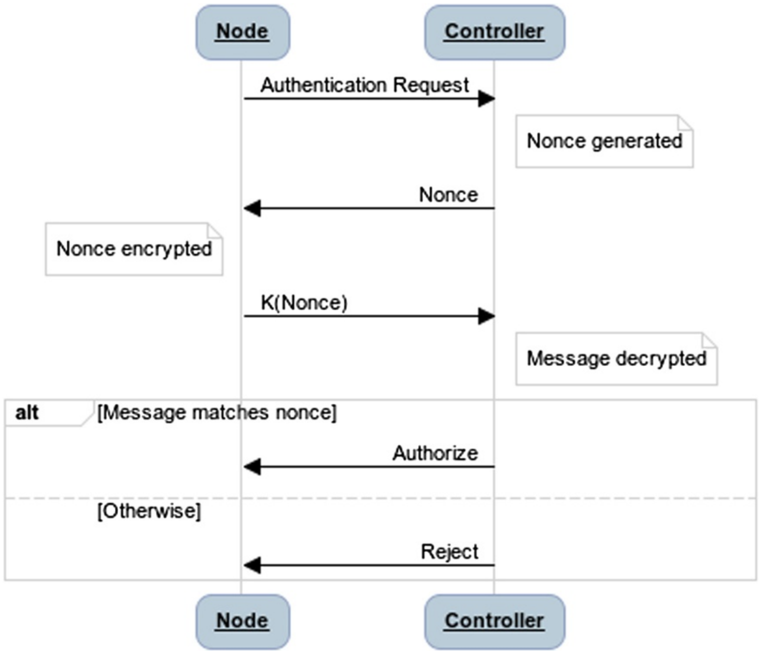


Figure 4 Node registration protocol.

```
function main
  // Instantiate NetEvent object
  // Listen on port 6667 with a
  // role of ADMIN
  netEvent = NetEvent(6667, 'ADMIN')

  // Register an event for invoking commands
  netEvent.registerEvent('INVOKE', invoke)
end

// Invoke a command within the system.
// Allows for the retrieval of information
// and triggering of events in other nodes.
function invoke(params)
  if (params[0] == 'GET_CLIENT_LIST')
    return netEvent.getClientList()
  elseif (params[0] == 'GROUP')
    groupName = params[1]
    event = params[2]
    res = netEvent.publishToGroup(groupName, event)
    return res
  else
    ...
  endif
end

main()
```

Figure 5 Example controller node service written in pseudocode.

cloud architecture. This approach is different than the more traditional approach of remote execution of tasks by means of SSH tunneling.

User interfacing

In the previous section we introduced our middleware API for managing cloud resources. However, another important component is a reasonable way to interface with the middelware solution. Although, the middleware API solution is completely independent from the interface, we have chosen to use a message passing approach that is different from that of general-purpose architectures. In this approach our web interface, which is written in PHP, connects to the controller node in order to trigger the "INVOKE" event. By interfacing with the controller node we are able to pass messages to groups or individual nodes in order to manage the resources of that node and receive responses. The ability to interface in this manner allows our interface to remain decoupled from the logical implementation, while allowing for flexibility in the interface and user experience. Figure 7 shows an example PHP script for interfacing with the resources in the manner described in this section.

```
function main
  // Create NetEvent object
  netEvent = NetEvent()

  // Register events
  netEvent.registerEvent('INSTANTIATE', instantiate)
  netEvent.registerEvent('STATUS', status)

  // Set the logical group name
  netEvent.associateGroup('COMPUTE')

  // Find and connect to controller node
  netEvent.findController()
end

// Instantiate virtual machine, return
// private IP address of instance
function instantiate(params)
  // params[0] -> name of image
  ...
  return ip_addr
end

main()
```

Figure 6 Skeleton for compute node service written in pseudocode.

```
class Communication {
  var $sock;

  /* Insert socket connect/send/receive code here */

  function getStatus($group) {
    $this->connect();
    $this->send('INVOKE GROUP ' .
                $group .
                ' STATUS');
    $response = $this->receive();
    $this->close();
    if ($response == '') {return array();}
    return explode('\;',$response);
  }

  function getClusterList() {
    $this->connect();
    $this->send('INVOKE CONTROL GET_CLUSTER_LIST');
    $response = $this->receive();
    $this->close();
    if ($response == '') {return array();}
    return explode('\;',$response);
  }
}
```

Figure 7 Example communication interface in PHP.

The PHP interface presented in Figure 7 illustrates the methodology behind how we may capture and display data about the nodes, as well as provide a means for user interaction in resource allocation and management. Although, we do not present the full source code in this paper, additional functions could be written. For example, a function could be written that instructs compute nodes to instantiate a particular virtual machine image. One important aspect of this system is that the mode of communication remains consistent at every level of the cloud stack. Every message sent is implemented via non-persistent socket connections. This allows for greater data consistency without modifying the semantics of messages between the different systems. Figure 8 shows an example interface for metadata aggregation of a logical compute group. Figure 9 presents a sequence diagram for the VM selection interface.

Supporting storage services

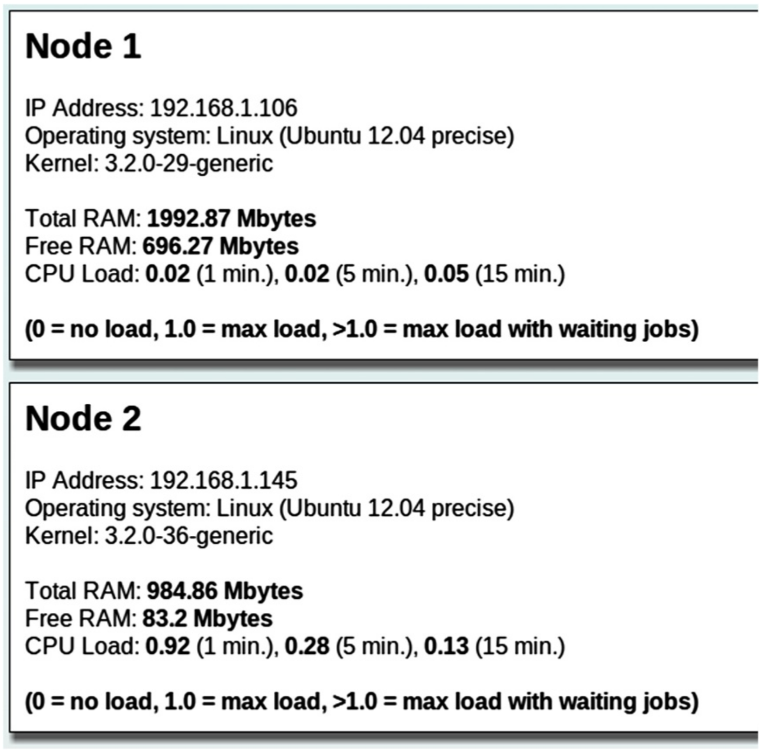
Pinnacle to the development of a complete cloud architecture, and a pre-requisite to supporting compute services is the ability for a cloud middleware to support the mounting and construction of persistent storage volumes. Storage service support is a pre-requisite of compute services because it is common for virtual machine images to reside on a separate image repository or network attached storage device. Therefore, before compute services can be fully realized it is necessary to be able to mount the image repository, such that the local hypervisor may have access to the virtual machine images. We can support storage services using the storage driver provided by libvirt. Figure 10 shows the XML specification provided to libvirt, which is required by the storage driver.

Once the storage pool has been mounted, then the user of the cloud may be provided access to storage space, if it is persistent userspace. Alternatively, if the share is a image repository, then the compute node will be given access to the virtual machine images provided by the storage pool.

Supporting compute services

The NetEvent API allows for services to be written and distributed to nodes within a private cluster. These services utilize the NetEvent API as a means for triggering events remotely. Within cloud architectures there are a few important events that must be supported. Firstly, the instantiation of virtual machine images must be supported by all cloud architectures. Compute services may be supported by combining the flexibility of the NetEvent API and a hypervisor, such as KVM. A proper compute service should maintain an image instantiation event which invokes the hypervisor and instructs it to instantiate a specific virtual machine image.

The steps involved in supporting compute services start with mounting the storage share containing the virtual machine images. This is made possible with the function, *mountVMPool*, which constructs a storage pool located in the directory "*/var/lib/iibvirt/images*", and is the default location by which libvirt may locate the available domains or virtual machine images available to the system. Once the virtual machine pool is mounted then a specific virtual machine may be instantiated, which is made possible with the function, *instantiateVM*. This function looks up the virtual machine, and if it exists in the storage pool, it will be instantiated. Once the VM is instantiated, a domain object will be returned to the node, which provides the methods for managing the virtual machine instantiation.



Node 1

IP Address: 192.168.1.106
Operating system: Linux (Ubuntu 12.04 precise)
Kernel: 3.2.0-29-generic

Total RAM: **1992.87 Mbytes**
Free RAM: **696.27 Mbytes**
CPU Load: **0.02** (1 min.), **0.02** (5 min.), **0.05** (15 min.)

(0 = no load, 1.0 = max load, >1.0 = max load with waiting jobs)

Node 2

IP Address: 192.168.1.145
Operating system: Linux (Ubuntu 12.04 precise)
Kernel: 3.2.0-36-generic

Total RAM: **984.86 Mbytes**
Free RAM: **83.2 Mbytes**
CPU Load: **0.92** (1 min.), **0.28** (5 min.), **0.13** (15 min.)

(0 = no load, 1.0 = max load, >1.0 = max load with waiting jobs)

Figure 8 Example interface for metadata aggregation with two nodes being polled for data.

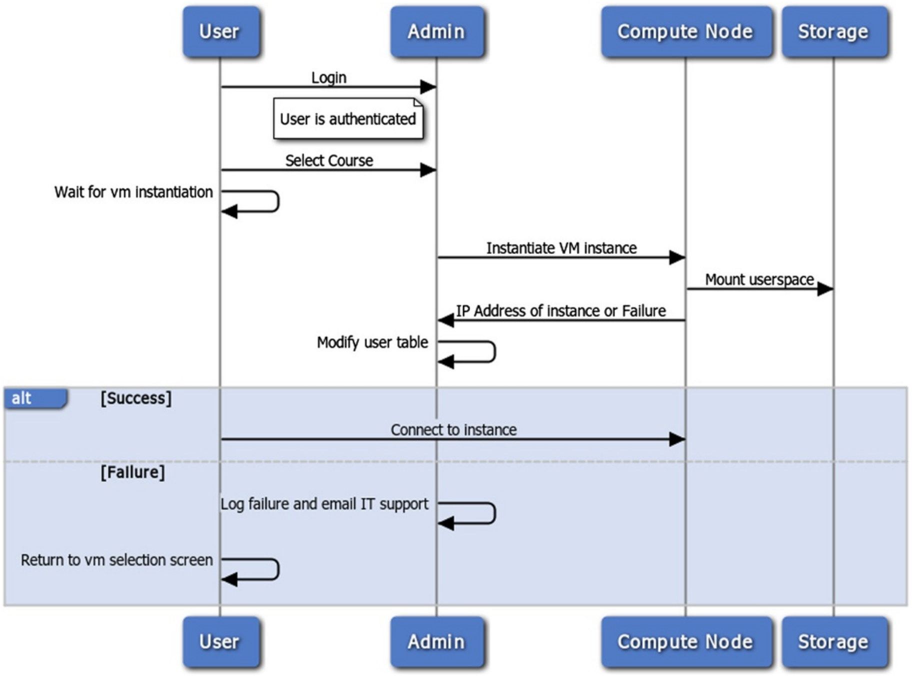


Figure 9 Sequence diagram for virtual machine image selection process.

```
<pool type="netfs">
  <name>Name_libvirt</name>
  <source>
    <host name="Hostname"/>
    <dir path="Name_share"/>
    <format type='Format'/>
  </source>
  <target>
    <path>Mountpoint</path>
  </target>
</pool>
```

Figure 10 **Storage pool XML specification required by libvirt.**

Supporting metadata aggregation

Metadata aggregation refers to the ability to retrieve data about each node within a specific group. This data may be used for informative purposes, or for more complex management tasks. Example metadata includes the nodes IP address, operating system, and kernel. Additionally, dynamic data may be aggregated as well, including RAM availability and CPU load. We can support metadata aggregation in each service by introducing events that retrieve the data and transmit it to the controller node.

Performance results

One of the many reasons for not using SSH, which seems to be the industry standard approach for inter-node communication in general-purpose cloud architectures, is that SSH produces excessive overhead. The communication approach used by NetEvent is very simplified and does not introduce data encryption or a lengthy handshake protocol. The downfall of simplifying the communication structure is that the system becomes at risk for loss of sensitive data being transmitted between nodes. However, in the case of this system no sensitive data is ever transmitted, and instead only simple commands are ever sent between nodes. For this reason encryption is unnecessary. However, authentication is still required in order to determine if nodes are legitimate. In testing the performance of NetEvent we compared the elapsed time for authenticating a node with the controller and establishing a connection with the elapsed time for SSH to authenticate and establish a connection. We gathered data over five trials, which is presented in Table 2. Additionally, Figure 2 presents the average latencies between SSH and NetEvent.

From the performance comparison we draw the conclusion that general-purpose cloud architectures that utilize SSH connections, such as Eucalyptus, sacrifice up to a 99% loss in performance when compared to traditional sockets. However, this comparison is being made at optimal conditions, because the servers are under minimal load. More data needs to be gathered to determine how much the performance is affected when the servers are overloaded.

Supporting software services

The preceding sections discussed our implementation of the software system necessary for supporting IaaS cloud services, namely hardware virtualization and persistent storage. Building upon virtualization of hardware, we are able to provide software services as custom virtual machine instances. The approach that THUNDER takes is instantiation of server virtual machine images, which deploy web services for user collaboration, research, and other tools and utilities. By implementing services in this fashion, no modifications are required to the infrastructure of the cloud, and administrators may easily start services by allocating hardware resources and stop services by deallocating resources. This approach differs from typical SaaS architectures in that no additional configuration is necessary outside of what is required for regular instantiation of virtual machines. The only difference is that regular users do not have access to connecting to service instances directly using VNC.

Future work and conclusion

We would like to introduce this architecture in a select number of organizations in order to determine the effectiveness and usability of the architecture from both the user's and administrator's perspectives. Based on the results of the study, we will alleviate any possible concerns from users or administrators. We plan to form an incremental process, such that various aspects of the system are studied in different organizational environments, then small changes will be made to the system

Table 2 Performance results comparing NetEvent to traditional SSH-based authentication

Trial #	SSH (ms)	NetEvent (ms)
1	298	3.1
2	301	3.2
3	302	9.2
4	298	3.1
5	299	3.0

before running another study. In this fashion, we will have more control over which features are beneficial to organizations, and which features are least significant. Currently, the THUNDER cloud is comprised of a set of standalone servers which are not organized in a shared structure, such as a rack-style chassis. We would like to build a complete prototype that is presentable and able to be easily taken to organizations for demonstration purposes. One of the key challenges in building a cloud infrastructure is the development of a middleware solution, which allows for ease in resource management. The work presented in this paper demonstrates that a middleware solution does not have to be as complex as those found in the popular cloud architectures, Eucalyptus and Openstack. We also introduced the model by which our middleware API offers communication between nodes, namely utilizing event driven programming and socket communication. We have developed our API to be efficient, light weight, and easily adaptable for the development of vertical cloud architectures. Additionally, we showed the manner in which a web interface may interact with the middleware API in order to send messages and receive responses from nodes within the cloud. For future work we would like to investigate approaches for fault tolerance in this architecture. Additionally, we would like to perform an overall system performance benchmark and make comparisons between other cloud architectures. We would also like to implement a method for obfuscation of management traffic such that the system may not be as susceptible to malicious users.

We have presented our work in designing and implementing a new private cloud architecture, called THUNDER. This architecture is implemented as a vertical cloud, which is designed for use in non-profit organizations, such as publicly funded schools. We leverage a number of technologies, such as Apache2 web server, and MySQL for implementation of the architecture. Additionally, we introduce socket programming and RPC as a viable alternative to the more common SSH based solution for inter-node communication. We have established that the primary goal of THUNDER is not to replace traditional private cloud architectures, but to serve as an alternative, which is custom tailored for reducing complexity, costs, and overhead in underprivileged and underfunded markets. We also demonstrate that if an organization were to adopt the THUNDER architecture they could benefit by reducing up to 50% of their power bill due to the low power usage when compared to a traditional computer lab. We believe that the power and cost savings, when combined with the features and qualities of THUNDER as presented in this paper, make THUNDER a desirable architecture for school computer labs and other organizations. Further studies and analysis will validate the effectiveness of the architecture.

Competing interests
The authors declare that they have no competing interests.

Authors' contributions
GL performed the research, design, and development of the THUNDER architecture. JG and JR revised the manuscript and contributed to the background work. XH provided insight and guidance in developing the networking model for THUNDER. SV edited and revised the final manuscript. All authors read and approved the final manuscript.

Received: 11 August 2013 Accepted: 14 December 2013
Published: 21 December 2013

References

1. Amazon Web Services. [http://aws.amazon.com/]. Accessed: 12/18/2012
2. Eucalyptus Enterprise Cloud. [http://eucalyptus.com/]
3. OpenStack. [http://openstack.org/]
4. Stein S, Ware J, Laboy J, Schaffer HE (2012) Improving K-12 pedagogy via a Cloud designed for education. Int J Informat Manage. [http://linkinghub.elsevier.com/retrieve/pii/S0268401212000977]
5. Cappos J, Beschastnikh I (2009) Seattle: a platform for educational cloud computing. ACM SIGCSE 111–115. [http://dl.acm.org/citation.cfm?id=1508905]
6. Donathan K, Ericson B (2011) Successful K-12 outreach strategies. In: Proceedings of the 42nd ACM technical symposium on Computer science education, pp 159–160. [http://dl.acm.org/citation.cfm?id=1953211]
7. Ercan T (2010) Effective use of cloud computing in educational institutions. Procedia - Social and Behavioral Sci 2(2):938–942. [http://linkinghub.elsevier.com/retrieve/pii/S1877042810001709]
8. Doelitzscher F, Sulistio A, Reich C, Kuijs H, Wolf D (2010) Private cloud for collaboration and e-Learning services: from IaaS to SaaS. Comput 91: 23–42. [http://www.springerlink.com/index/10.1007/s00607-010-0106-z]
9. Sultan N (2010) Cloud computing for education: A new dawn? Int J Inform Manage 30(2):109–116. [http://linkinghub.elsevier.com/retrieve/pii/S0268401209001170]
10. Masud M, Huang X (2011) ESaaS: A new education software model in E-learning systems. Inform Manage Eng 468–475. [http://www.springerlink.com/index/H5547506220H73K1.pdf]
11. Linux Terminal Server Project. http://ltsp.org/. [Accessed: 12/23/2012]
12. Willem Vereecken LD (2010) Energy efficiency in thin client solutions. GridNets 25:109–116
13. Cardellini V, Iannucci S (2012) Designing a flexible and modular architecture for a private cloud: a case study. In: Proceedings of the 6th international workshop on Virtualization Technologies in Distributed Computing Date, VTDC '12. ACM, New York, NY, USA, pp 37–44. [http://doi.acm.org/10.1145/2287056.2287067]
14. An K, Pradhan S, Caglar F (2012) Gokhale AA publish/subscribe middleware for dependable and real-time resource monitoring in the cloud. In: Proceedings of the Workshop on Secure and Dependable Middleware for Cloud Monitoring and Management, SDMCMM '12. ACM, New York, NY, USA, pp 1–3:6. [http://doi.acm.org/10.1145/2405186.2405189]
15. Lee SY, Tang D, Chen T, Chu WC (2012) A QoS Assurance middleware model for enterprise cloud computing. In: IEEE 36th Annual Computer Software and Applications Conference Workshops (COMPSACW), 2012, pp 322–327
16. Apostol E, Baluta I, Gorgoi A, Cristea V (2011) Efficient manager for virtualized resource provisioning in cloud systems. In: IEEE International Conference on Intelligent Computer Communication and Processing (ICCP), 2011, pp 511–517
17. Khalidi Y (2011) Building a cloud computing platform for new possibilities. Computer 44(3):29–34
18. Pallis G (2010) Cloud computing: the new frontier of internet computing. IEEE Int Comput 14(5):70–74
19. Raspberry Pi Foundation (2013). http://www.raspberrypi.org/ [Accessed: 11/15/2012]
20. EC2 Tools (2013). [http://www.eucalyptus.com/eucalyptus-cloud/tools/ec2]
21. OpenStack Nova (2013). [http://nova.openstack.org/]

22. Surhone L, Timpledon M, Marseken S (2010) Remote desktop protocol. VDM Verlag Dr. Mueller AG & Company Kg, Saarbruecken, Germany
23. Barrett DJ, Silverman RE, Byrnes RG (2005) SSH, The secure shell: the definitive guide. O'Reilly Media, Sebastopol, CA, USA
24. VNC - Virtual network computing (2013). [http://www.hep.phy.cam.ac.uk/vnc_docs/index.html]. [Accessed: 12/18/2012]
25. libvirt - The virtualization API (2013). http://libvirt.org/. [Accessed: 7/21/2013]
26. M'Raihi D, Rydell J, Bajaj S, Machani S, Naccache D "OCRA: OATH Challenge-Response Algorithm", RFC 6287. June 2011

doi:10.1186/2192-113X-2-24
Cite this article as: Loewen *et al.*: **THUNDER: helping underfunded NPO's distribute electronic resources.** *Journal of Cloud Computing: Advances, Systems and Applications* 2013 **2**:24.