Take a Penny, Leave a Penny
Scaling out to Off-premise Unused Cloud Resources

Matthias Steinbauer, Gabriele Kotsis

Department of Telecooperation
Johannes Kepler University Linz
Altenbergerstrae 69
4040 Linz
matthias.steinbauer@jku.at
gabriele.kotsis@jku.at

Abstract: This work describes a system that enables sharing of cloud resources through a middle-ware layer we call market. It is designed with the requirements: minimal configuration, easy access and sharing in mind. This effort resulted in a platform that provides a common interface to cloud computing resources, hiding implementation details from the users and allowing them to tap into cloud resources from different vendors.The system supports extensible mechanisms to configure virtual machines dispensing the user from error-prone tasks, such as setting up and configuring compute clusters. A simple web based user interface is provided which allows browsing, configuring, launching and managing of virtual machines and virtual clusters. Integrated remote desktop facilities make the virtual computing resources accessible directly from the user interface.

1 Introduction

In the last decade, models for computing infrastructure are under drastic change. In many application areas the use of cloud computing [RCL09] is gaining ground. The easy provisioning and deallocation of computing resources combined with the fact that one gains better utilization of the underlying hardware, have lead to a situation where many organizations consider switching to cloud computing or are already in the progress of doing so.

Cloud computing as a discipline of computer science is considered mature for practical applications. These applications can usually be assigned to one of the three service levels given in the NIST definition of cloud computing [MG11]. The first level according to this definition is Infrastructure as a Service (IaaS) where a cloud provides virtual computing instances that can be provisioned via software interfaces. On top of that often Platform as a Service (PaaS) systems run that provide more complex computing platforms to their users. These platforms are able to use the underlying IaaS cloud in order to scale dynamically to current compute demands. Finally, on top of the PaaS systems, or even directly on IaaS virtual instances, Software as a Service (SaaS) applications are living. They provide a very

specific software application to their user; typically with zero setup and configuration at the users side. This work is mainly contributing to IaaS clouds as as in making them easier accessible and their resources shareable. Our user interface, the middle-ware and example applications, are situated in the SaaS level and help users in managing resources at the IaaS level.

NIST also defines deployment models of cloud computing. (1) A public cloud is a computing, storage and networking infrastructure open for public use. Customers are able to lease computing resources and pay per use. The cloud is open to public use. (2) A private cloud is located on premise within an organization and is used only for their own needs. Practice shows that there are hybrid approaches, for instance a private cloud installation is used for a companies main computing demands, but during peak seasons public cloud offerings are used to handle the higher demand.

One of the main advantages in cloud computing is the pay-per-use availability of computing resources from commercial providers. This allows projects to start off quickly and adapt their computing resources to the current demand. Further, this computing model allows testing of new systems without the need to have any on-premise hardware investment.

However, commercial public cloud offerings also come with some disadvantages. In general they promote vendor lock-in. Naturally each commercial cloud vendor will want their customers to use their systems over the offerings of competitors. This is one reason why there is not too much effort in cross system compatibility of computing resources between different public cloud vendors.

Further, different public cloud providers and vendors support different APIs for accessing their cloud. This means if a customer created an application that automatically scales to current demand, it is not necessarily portable. Moving from one cloud provider to another will always involve software adaptation.

Finally, in demanding applications as in Big Data research, public cloud offerings provide little to no insight to the inner workings of the system. It is hardly possible for researchers to investigate effects on the infrastructure of a public cloud since access to these parts of a public cloud is restricted.

When we observe the field of non-commercial cloud labs, but also commercially operating organizations, we see that many organizations are already running virtual computing clusters, or a private cloud or there is work in progress in installing such shortly.

Private clouds might fit better to some application areas, especially when it comes to tune the inner workings of a cloud to the requirements of an application. Still they come with the problem of differences in APIs between cloud stack vendors, making it difficult to switch the cloud vendor without code changes.

Private clouds introduce further problems, the cloud owner and their operators need to take care about optimal resource usage on their own. Cloud stacks typically provide mechanisms to balance the workload within an installation. This balancing is done with many different scheduling algorithms [RLTB10, HGSZ, WYLW]. However, a private cloud installation may hit resource limits that make it necessary to scale out some tasks to a public

cloud. On the other hand, it might also be the case that a cloud installation is too large for the baseline workload of an organization, leading to a situation where resources are not used efficiently; the private cloud is under-utilized.

Virtualization is widely used to solve the problem of under-utilization of single machines. In fact virtualization and over-provisioning of hardware are key enabling technologies of cloud computing. However, this does not automatically mean that there is no under-utilization in private cloud clusters. The problem has often just moved to a higher level in the big picture. We whiteness a shift from under-utilization of machines to under-utilization of private cloud installations. This development has already been investigated by Osterman Research [Ano11]. Their survey clearly shows that 52.2% of the surveyed companies, running private cloud installations state that their private cloud is vastly under-utilized.

A very similar problem is called cloud sprawl or virtual machine sprawl [LMMR11]. In scenarios where virtual machines are acquired for short term use i.e., for a temporary project or a test-run, these virtual machines tend to sprawl. Often the provisioned resources are never released and continue to consume resources although the machines are unused.

In order to use underutilized clouds more efficiently, one can apply the same concepts that were used for single machines. The first option that feels natural is to turn off unused resources in order to save energy. On the other hand the same resources could be shared by several tasks. While there is much work that elaborates the advantages in turning off unused hardware in a cloud in order to increase energy efficiency it is sometimes impractical or does not justify the investment in hardware.

Cloud computing is all about a simple and easy way of obtaining, using and redressing computing resources. Costs for computing resources are shifted from one-time hardware and setup costs accompanied by maintenance costs to a more linear model where consumers pay per use. Vendors and providers are using many different concepts in order to market their computing resources. The concept of cloud computing makes it inherent that there is some instance like a market that provides the interlink platform between resource providers and resource consumers. Markets feel natural in this scenario, since the cloud computing concept is designed with a pay as you go model in mind. This is why we call our application middle-ware *market* from now on. We believe that by providing a market that allows for sharing and selling of idle cloud resources could motivate private cloud owners to put their underutilized resources to good use.

Since cloud computing should simplify computing we believe that in cloud markets need to be as simple as possible. Consequentially one should try to learn from one of the simplest, markets available. One of these are the *Take a Penny, Leave a Penny* trays available in many shops and restaurants all over the world. They form the ideal reference for a resource sharing market. There are three inherent properties in these trays that make them so popular. (1) They require only *minimal configuration*. Basically just the act of putting up a tray with the sign "Take a Penny, Leave a Penny" is sufficient to make the system work. (2) The trays are overly *easy to access*. There are no barriers at all between the user and the market. Everybody is allowed to access the tray with almost no restrictions. Over consumption takes place seldom since the shop keepers have an eye on the tray. (3) Finally

these trays are all about *sharing*. People that can spare a penny just drop their change to the tray, those in the need of a penny are free to take one without asking further permission. In this work we will present how these properties can be applied in the creation of a cloud computing market.

The remainder of this paper is structured as follows: In section 2, we briefly describe related work. Section 3 describes the concept of our market application. In section 4, we explain our current prototype implementation. Lessons learned and an outlook on future work are given in section 5, and section 6 concludes the paper.

2 Related Work

The problem of accessing different cloud stacks with one general API was already solved in related work, see JSClouds [jsc]. However, this library lacks support for all available stacks. Further, its APIs works on a reduced feature set that is compatible to all supported clouds. In our work we do not rely on JSClouds since it does not support advanced configuration features like contextualization in OpenNebula. Further, it does not support the OpenNebula API at all at the time of writing this.

Allowing easy access to computing resources is a common goal for many cloud computing sites and projects. One example that focuses on creating a test-bed for research is OpenCirrus [CGH+09]. It is a middle-ware that is capable of providing access to four different cloud computing sites and their resources. The available resources are virtual machines and bare metal in heterogeneous clusters. This system provides computing resources, but does not provide preconfigured virtual machine images or clusters.

In contrast, the myHadoop project focuses on the provisioning of Hadoop clusters. It provides tools and mechanisms in order to deploy Hadoop clusters on traditional high performance computing sites. This way the system leverages tools from high performance computing in order to deploy the Hadoop cluster. In [KTB11] it is also briefly discussed that mileage may vary since in high performance computing, compute nodes with minimal local storage are usually used whereas Hadoop or other programming models, similar to MapReduce, are designed for shared-nothing architectures.

The goal of Future Grid and its educational virtual clusters [FWC11] is to provide easy to use virtual appliances that can be configured as clusters. The focus of this system is entirely on training. The users should have very short turn around times on setting up their system. This is achieved by providing a very basic Linux based appliance that can be provisioned on demand. Once the user has access to the virtual machine, he is able to install and start more complex cluster systems by executing batch scripts inside the appliance. This way, single cluster nodes are configured automatically.

Of course, the process could be further simplified by also automating the provisioning of complete clusters, i.e. not only automating the setup and basic configuration of software, but automating the whole process. This would be achieved by creating the virtual instances, installing the necessary software and, finally, configuring each node according to its task. Thus, enabling one-click provisioning of complete cluster systems. As explained

in [KPP09], this process is often automated, however, mostly only the provisioning of one type of cluster system is supported. For instance, there exist may different ways for one-click provisioning of Hadoop systems [ama, clo, hadb]. Our system, however, is able to harness several different cluster systems and provides access to them with one generic user interface.

3 Concept

We propose a market application that consists of two major components, illustrated in figure 1. First, a middle-ware layer situated at the PaaS level according to NIST, is handling the access to different cloud stacks, virtual machine images and their storage location, and manages user authentication and sharing ratios. It accesses underlying cloud resources completely transparent to applications and users.

The middle-ware contains a component called virtual machine repository which manages and hosts virtual machine images. This repository also provides tools that transparently convert images between different file formats.

On top of this middle-ware we provide a user interface that is a SaaS application itself. It provides users with a one stop facility to manage all aspects of their cloud computing needs, such as, creating and destroying virtual machines, accessing virtual machines via virtual desktop protocols, or even managing and re-sizing complete compute clusters.

The middle-ware exposes its methods as an API effectively allowing user applications to use the middle-ware for more complex applications. For instance in cluster setups it would be possible to automatically scale the cluster to current demands, without the need for the cluster to know the exact cloud stack implementation it runs on.

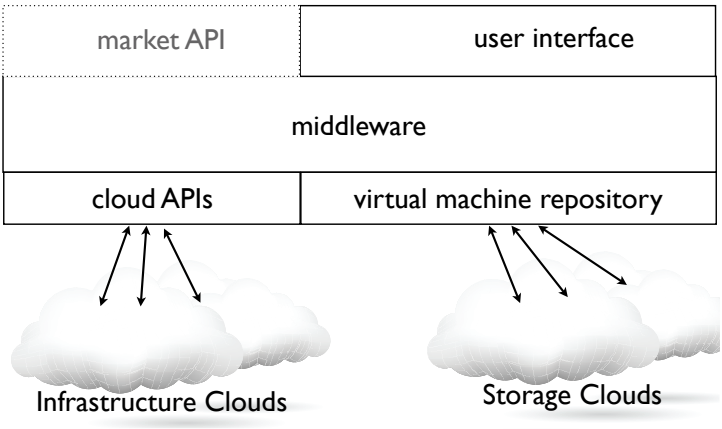

Figure 1: Overall system architecture

As mentioned in section 1 we try to learn (1) minimal configuration, (2) easy access and

(3) sharing from the Take a Penny, Leave a Penny trays. The following will elaborate on what these properties mean in the cloud computing space.

3.1 Minimal Configuration

Minimal configuration affects two aspects of a cloud market: the market application and the available virtual machine images. For the market application itself no further complexity should be introduced, instead the market should simplify cloud computing by providing a common interface that hides differences in cloud stacks, virtual machine hypervisors and operating systems from the user.

In infrastructure clouds complexity does not end with the cloud access interface. Also the systems and services that are running inside virtual machine images need configuration. The configuration and setup effort is already reduced due to the introduction of cloud computing and the possibility of sharing virtual machine images. However, many services running inside a cloud need further configuration besides simply booting a virtual machine image. The configuration of the services is often more troublesome than expected, especially for users new to a certain topic. This hinders the curious from experimenting with different systems without going through the hassles of configuration their systems.

With our market we provide virtual machine configurations that can be injected into virtual machines during boot. These configuration parameters and scripts are used to configure individual virtual machine images to the users need, for example, users could enter default log-in credentials that are used to configure the virtual machines default user-names and passwords. Here the same information can be used for many different virtual machines.

If users are interested in larger compute clusters, such as MPI [GFB+04], Hadoop [hada] or Storm [sto] clusters, the configuration of the systems gets even more cumbersome. This usually involves configuring many machines that run the same software stack in such a way that they are able to work together in a cluster. From a users perspective this configuration task can be reduced to a single configuration parameter, the size of the cluster.

Some clustered software systems allow dynamic scaling of the cluster size depending on the current demand. The market also needs to reflect this dynamicity by allowing its users to add and remove compute nodes from cluster. However, the market needs to enforce a lower bound on the cluster size to ensure the stability of the system.

3.2 Easy Access

The field of cloud computing is diverse in terms of access APIs, authentication APIs, virtual machine hypervisors and remote access protocols. A common interface that hides this diversity and allows uniform access to any resource can ease the work-flow of cloud users.

Most private and public cloud stacks provide API access to their system which can be used

create scaleable applications. The field of APIs is very diverse and projects like JSClouds try to create a uniform access layer. Some of the protocols used in this ecosystem became de facto standard since they are so widely used in the industry. This is the case, for, Amazons EC2 cloud API, which has been adopted by other systems like OpenNebula or Eucalyptus [NWG+09, ST10].

The Open Grid Forum, on the other hand, is publishing the specification of the Open Cloud Computing Interface (OCCI) [NEPM11]. OCCI is designed to provide interoperability between cloud based applications and different IaaS providers, enabling the creator of applications to scale out on different clouds from different vendors. The market and its middle-ware will need to support many different cloud access protocols, starting with EC2 and OCCI.

The diversity does not stop at accessing cloud infrastructures but also comprises a variety of authentication mechanisms. We whiteness many different authentication mechanisms, such as user-name and password, X.509 certificates, and several authentication as a service providers, in use. A market application needs to be able to authenticate to infrastructure clouds by proxy. The market can either use its own credentials or use those provided by the user.

Remote desktop and remote shell access protocols introduce more diversity to the cloud ecosystem. The virtual machines running on a cloud stack may run different operating systems coming with different mechanisms of remote access. Windows machines are typically configured with RDP or VNC access, Linux and Unix machines are often using VNC or SSH. Further, there are alternative software packages that allow multi platform access like TeamViewer, LogMeIn or Splashtop to name only a few. In this heterogeneous area of remote access options a unified remote desktop system needs to be integrated into cloud markets. This remote desktop system needs to allow seamless access to any cloud based virtual machine.

3.3 Sharing

Sharing expands into three dimensions in the context of this work. Users need to be able to share virtual machine images, complex configuration and most obviously computing resources. Easy sharing of virtual machine images has to cover two aspects: the conversion between different virtual machine image formats and the storage of the images independently of a certain cloud installation.

Virtual machine images are the digital building blocks of cloud based computing services. They contain operating systems and previously configured software to run a service. Depending on the hypervisor used in a infrastructure cloud these images may vary in their storage format. Typically hypervisors like Virtual Box or VMWare use their proprietary virtual machine image format. The sharing component of our market application needs to be able to convert between different virtual machine image formats on demand. The conversion of virtual machine images is situated at the virtual disk image repository which is responsible for storing disk images it is able to integrate disk images from several back-

ends into one repository. A backend is any cloud storage service as they are provided from Amazon S3, Hadoop DFS or the CEPH distributed file system. If a virtual disk image gets deployed to an infrastructure cloud the repository converts to the correct hypervisor format on demand and copies the resulting target image to the infrastructure cloud.

Finally, an integral aspect of this market is the ability to share computing resources amongst different organizations. Each organization participating in this pool of resources needs to be able to connect their cloud resources to the market. The market application needs to provide a simple administrative user interface to allow organization representatives to connect and disconnect their resources to and from the cloud.

4 Prototype

Along the architecture and the requirements that were elaborated in section 3, we created a prototype that implements the explained concepts. In the following we will highlight some facets of the resulting implementation.

Our prototype application was implemented as a Java Enterprise application using Spring [spr] service bindings and Spring Web Flow [web] . All important access layers, such as cloud, authentication, and image repository, were implemented as Spring services to make them exchangeable. Further, this allows us to separate the user interface from the middle-ware layer; the middle-ware and the image repository can be deployed on different application servers. Our application was tested on top of Tomcat 7 installations and is available as a virtual machine disk image that runs the Debian Linux 6.0 operating system.

We are currently running our prototype on top of an OpenNebula installation at our department. The market is connected to a second OpenNebula instance located in Marrakesh and an OpenStack installation located in Hagenberg. We implemented an access API that uses the OCCI interface to access OpenNebula clouds and EC2 to access the OpenStack installation.

Our virtual machine image repository implements a backend to access OpenNebula disk images, Amazons S3 and arbitrary HTTP URLs that point to a disk image. The list of possible backends is subject to expansion in the future through the use of an storage abstraction layer like CSAL [HH10].

As indicated earlier, on top of the middle-ware we run a user interface that is based on Spring Web Flow and PrimeFaces [pri]. This user interface caters to users and administrators; users are enabled to create, access and delete virtual computing instances, administrators are able to add and remove cloud data-centers and image storage repositories.

4.1 Configuration

We assume that the market is rather an access terminal to virtual computing resources, then a configuration and administration tool. That is why the virtual machine images need

to be prepared on some cloud stack or virtual computing tool prior their usage. These images are then deployed to the image storage repository and can then be accessed from the market. The images should require only very little configuration which is collected by a wizard during virtual machine instantiation.

We have tested the market with many different images two of which are running the Windows 7 and the Debian 6.0 operating systems. Both images can be instantiated on our cloud and get further configuration injected during boot up. We setup the virtual machines host-name according to the users input and create a user with a user-name and password pair from the users preferences in the market. The configuration parameters and scripts are injected into the virtual machines using OpenNebulas contextualization feature [KF08] that allows complex setup routines during virtual machine boot time.

In our implementation we make use of the Guacamole remote desktop package, which is implemented in pure HTML5 and JavaScript. This package enables the market to provide remote desktop access to any virtual machine that supports either the VNC, RDP or SSH protocol.

We also implemented more complex virtual machine images that demonstrate certain aspects of our market in further detail. One is running an instance of Tomcat 7 [tom] that deploys web applications during boot up, another one is running the distributed batch processing system Hadoop [hada] , and one is running the streaming big data processing platform Storm [sto].

Our Tomcat 7 image contains a plain Tomcat 7 installation that can be configured to run a certain web application during boot. This image makes use of Maven [mav] and its possibility of deploying artifacts to Maven repositories. When a user instantiates our Tomcat 7 image the market asks for a Maven group id and artifact id, it then tries to resolve these ids and retrieves the web application WAR file from the repository. This WAR file then gets deployed to the virtual machine accordingly.

In cluster setups the market supports to start many virtual machines at once, logically linking them together as a cluster. The user is able to manage single nodes or the cluster en bloc. It is possible to add or remove virtual machines to a cluster or to start and stop complete clusters respectively.

To configure single virtual machines or multi-node clusters we use OpenNebulas' contextualization feature. This feature uses a virtual CD-Rom image that is created on demand by the cloud infrastructure. The image contains a configuration file and user scripts that actually perform the setup process. The configuration scripts are provided by the creators of the virtual machine disk image. However, the configuration file is setup by the computing infrastructure. In the default case this file contains variables, also called context parameters, such as the IP address of the virtual machine and its host-name. On virtual machine instantiation more user defined variables can be added to this configuration file. We leverage this feature to inject further information like user-names, passwords or complex cluster setups into the virtual machine.

In the case of our Hadoop and Storm cluster setups, we needed to distinguish between the setup of master or slave node. That is why we introduced a context parameter called *MASTER*. If this variable is set the configuration scripts configure the affected machine

as master node. If the variable is not available and the virtual machine image is a cluster image, then the node is configured as a worker node. In the worker node setup, the configuration scripts look for a parameter called *IP_MASTER* which specifies the clusters master node.

The virtual machine images for Hadoop and Storm clusters are preloaded with the software needed. We use the same image for all node types; whether a virtual machine is to become a master or worker node is decided during boot time [1].

The market application is using a two phase instantiation strategy to correctly configure clusters and hosts.First the master node with *MASTER* parameter set is started. The system waits for this machine to be deployed on the cloud to retrieve its public IP address. In the second phase this IP address is injected into the slaves configuration (*IP_MASTER* parameter) and all slave nodes are started in a batch.

First, the virtual image is modified in such a way that during boot time the virtual CD drive is mounted and the start-up scripts are executed. For Linux systems, this can usually be achieved by adapting the */etc/rc.local* script. For Windows based virtual machines, the context scripts need to be registered as PowerShell start-up scripts in the local Group Policy Editor.

In the case of Linux we execute a script on the virtual CD-Rom, which is called *init.sh* it is reading the *context.sh* file that contains the configuration parameters. Then, depending on the use case, further scripts are executed. To illustrate this, an example, the case of our Hadoop cluster images, is explained here.

Our Hadoop images are configured in two phases. First we configure networking then the Hadoop system. The *net.sh* script which is responsible for network configuration. It modifies Debian / Ubuntu specific network configuration files in order to use the IP address provided by the context and to setup */etc/hosts* and */etc/hostname* correctly to the host-name provided.

When the network is configured correctly, the Hadoop setup script (*hadoop.sh*) is run, which first installs SSH keys for the machines to communicate via SSH without the need for passwords. Then, the script decides if it is executed on a master or on a slave node and changes configuration accordingly. After each machine has completed its local setup, it continues to complete the cluster setup. For the master nodes, this means that the master node is waiting for all worker nodes to finish their configuration. The worker nodes are signalling this to the master node via SSH. After all worker nodes have registered with the master node, the master starts up the Hadoop services on all machines and then formats the Hadoop distributed file system.

[1] Please find the virtual machine images and the configuration scripts ready for download at `http://www.steinbauer.org/publish/vm-context-hadoop-storm.zip`

4.2 Hadoop Test-case

Hadoop is a PaaS level cloud computing platform designed for Big Data processing. It is an open source implementation of Google's distributed file-system and MapReduce programming model [DG08]. This allows for the creation of very data intensive applications that are using the distributed file system as their storage. MapReduce applications usually run as batch processing jobs, reading and writing to the distributed file system.

The system is very elastic or scaleable and consists of several components, which may run on different machines. Particularly important is the NameNode, which provides the index of the distributed file-system. Usually, one also runs a MapReduce TaskTracker on the same node. The TaskTracker is able to schedule MapReduce jobs and sends them to worker nodes, called JobTracker, in Hadoop. Nodes running the JobTracker usually also run the DataNode component that is used to manage chunks of the distributed file system.

With our images, the user is able to run the following two configurations: (1) a development setup and a (2) a production setup. During development, one could run all four components on one single machine. One would do that during development to debug MapReduce jobs or to investigate the Hadoop distributed file system. A user is able to achieve this setup by simply starting a one-node Hadoop cluster in our market.

During production use of Hadoop, the system is usually deployed in a cluster setup. The most common setup is to have one dedicated machine that is running the NameNode and TaskTracker components of Hadoop. This is the master node of the cluster and a single point of failure for the system. This master node is accompanied with many slave nodes, that run the DataNode and JobTracker components. Thus, every slave node is able to store parts of the distributed file system and execute MapReduce jobs on its local data. In this setup, replication is also enabled such that slave nodes may fail without influencing the stability of the complete system. We are able to create that type of clusters on demand within our market by instantiating clusters of any size. If more than one node is assigned to a Hadoop cluster, the first node is a designated NameNode and TaskTracker - all other nodes become configured as DataNodes and JobTrackers.

In very large setups, having the NameNode and the TaskTracker component on the same host may be a performance issue. To overcome this, it would also be possible to run the NameNode and the TaskTracker on different machines, however, this is not supported for automatic provisioning by our system right now. Nonetheless, the user has access to each individual computing node and, thus, is able to reconfigure his cluster manually for such a setup.

4.3 Storm Test-case

As a second test case, we use Twitter Storm, which is also a PaaS cloud computing platform. In contrast to Hadoop, this Big Data system is focused on streaming data. It allows the creation of topologies of computing elements, which get distributed on a Strom cluster. The cluster management system takes care of load balancing and routing data correctly

between the individual processing elements.

Since Storm clusters are designed to not save a persistent state to a file system, the system does not need any tools to govern a distributed file system. This leads to a system where only two types of nodes exist: The Nimbus node, which is the master node of a Storm cluster and an arbitrary number of worker nodes, which are running a process called Supervisor.

5 Limitations and Future Work

In our prototype implementation we have shown the technical feasibility of our approach. However, we could not address certain aspects that might hinder successful application of this concept in real world scenarios. First of all participants in this kind of market are eligible to disconnect their cloud from the market at any time. This means that users need to design their applications in a way that their stability remains unaffected if virtual resources located at a sharing partner stop functioning. Moreover we did not plan and implement the scenario of organizations opting out, this means that if an organization disconnects their cloud from the market, the virtual resources keep running. In future work we will address this disconnection scenario by setting up processes that either terminate virtual resources on disconnection or try to migrate virtual resources to a different cloud installation.

Further, the configuration of services inside of virtual machine images relies on the contextualization feature of OpenNebula. If other cloud stacks are used with our market the configuration features supported by the market are drastically reduced, this means that in future work we will create a more general approach of configuring virtual machines. We are planning to use configuration bundles located at network locations to support cloud stacks that do not support contextualization or similar configuration features.

Finally our prototype was not put to thorough evaluation yet. We are planning to setup the market in real world scenarios and measure the usage of the market.

6 Conclusion

In this paper we have elaborated on our work in progress in establishing a market for cloud computing resources. This market inherits the properties *minimal configuration*, *easy access*, and *sharing* from the all famous *Take a Penny, Leave a Penny* trays found in many shops and restaurants all over the world. We argued to remove some of the complexity in federated clouds to unfold their full potential.

With our guiding properties in mind we have developed a first prototype implementation that consists of a middle-ware which acts as a broker between users and infrastructure clouds. Our prototype features a virtual machine image repository that hosts virtual machine images and is able to convert to other formats on demand. The market provides

access to cloud resources via a web based user interface or by direct API access.

References

[ama] Amazon Elastic MapReduce. http://aws.amazon.com/elasticmapreduce/.

[Ano11] Results of a Survey Conducted for Electric Cloud. Technical report, 2011.

[CGH+09] Roy Campbell, Indranil Gupta, Michael Heath, Steven Y. Ko, Michael Kozuch, Marcel Kunze, Thomas Kwan, Levin Lai, Hing Yan Lee, Martha Lyons, Dejan Milojicic, David O'Hallaron, and Yeng Chai Soh. OpenCirrus Cloud Computing Testbed: Federated Data Centers for Open Source Systems and Services Research. 2009.

[clo] Cloudera Manager. http://www.cloudera.com/products-services/tools/.

[DG08] Jeffrey Dean and Sanjay Ghemawat. MapReduce: simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, January 2008.

[FWC11] Renato J. Figueiredo, David I. Wolinsky, and Panoat Chuchuaisri. Educational Virtual Clusters for On-demand MPI/Hadoop/Condor in FutureGrid. 2011.

[GFB+04] Edgar Gabriel, Graham E. Fagg, George Bosilca, Thara Angskun, Jack J. Dongarra, Jeffrey M. Squyres, Vishal Sahay, Prabhanjan Kambadur, Brian Barrett, Andrew Lumsdaine, Ralph H. Castain, David J. Daniel, Richard L. Graham, and Timothy S. Woodall. Open MPI: Goals, Concept, and Design of a Next Generation MPI Implementation. In *Proceedings, 11th European PVM/MPI Users' Group Meeting*, pages 97–104, Budapest, Hungary, September 2004.

[hada] Apache Hadoop. http://hadoop.apache.org.

[hadb] Hadoop on Demand. http://hadoop.apache.org/common/docs/r0.17.0/hod.html.

[HGSZ] Jinhua Hu, Jianhua Gu, Guofei Sun, and Tianhai Zhao. A Scheduling Strategy on Load Balancing of Virtual Machine Resources in Cloud Computing Environment. In *Third International Symposium on Parallel Architectures, Algorithms and Programming (PAAP 2010)*, pages 89–96. IEEE.

[HH10] Zach Hill and Marty Humphrey. CSAL: A Cloud Storage Abstraction Layer to Enable Portable Cloud Applications. In *2010 IEEE 2nd International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 504–511. IEEE, 2010.

[jsc] JSClouds, Cloud interfaces, simplified. http://www.jclouds.org.

[KF08] K Keahey and T Freeman. Contextualization: Providing One-Click Virtual Clusters. In *IEEE Fourth International Conference on eScience*, pages 301–308, 2008.

[KPP09] Karthik Kambatla, Abhinav Pathak, and Himabindu Pucha. Towards Optimizing Hadoop Provisioning in the Cloud. *Workshop on Hot Topics in Cloud Computing*, 2009.

[KTB11] Sriram Krishnan, Mahidhar Tatineni, and Chaitanya Baru. myHadoop - Hadoop-on-Demand on Traditional HPC Resources. 2011.

[LMMR11] Maik Lindner, Fiona McDonald, Barry McLarnon, and Philip Robinson. Towards automated business-driven indication and mitigation of VM sprawl in Cloud supply chains. In *2011 IFIP/IEEE International Symposium on Integrated Network Management (IM 2011)*, pages 1062–1065. IEEE, 2011.

[mav] Apache Maven Project. `http://maven.apache.org`.

[MG11] P Mell and T Grance. The NIST definition of cloud computing (draft). *NIST special publication*, 800:145, 2011.

[NEPM11] Ralf Nyren, Andy Edmonds, Alexander Papaspyrou, and Thijs Metsch. Open Cloud Computing Interface - Core. Technical report, 2011.

[NWG+09] Daniel Nurmi, Rich Wolski, Chris Grzegorczyk, Graziano Obertelli, Sunil Soman, Lamia Youseff, and Dmitrii Zagorodnov. The Eucalyptus Open-source Cloud-computing System. In *9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, pages 124–131. IEEE, 2009.

[pri] PrimeFaces. `http://www.primefaces.org`.

[RCL09] Bhaskar Prasad Rimal, Eunmi Choi, and Ian Lumb. A Taxonomoy and Survey of Cloud Computing Systems. In *Fifth International Joint Conference on INC, IMS and IDC*, pages 44–51, 2009.

[RLTB10] Martin Randles, David Lamb, and A Taleb-Bendiab. A comparative study into distributed load balancing algorithms for cloud computing. In *24th International Conference on Advanced Information Networking and Applications Workshops*, pages 551–556. IEEE, 2010.

[spr] Spring Framework. `http://www.springsource.org/spring-framework`.

[ST10] Peter Sempolinski and Douglas Thain. A Comparison and Critique of Eucalyptus, OpenNebula and Nimbus. In *2nd International Conference on Cloud Computing Technology and Science*, pages 417–426. IEEE, 2010.

[sto] Storm: Distributed and fault-tolerant realtime computation. `http://storm-project.net`.

[tom] Apache Tomcat Project. `http://tomcat.apache.org`.

[web] Spring Web Flow. `http://www.springsource.org/spring-web-flow`.

[WYLW] Shu-Ching Wang, Kuo-Qin Yan, Wen-Pin Liao, and Shun-Sheng Wang. Towards a Load Balancing in a three-level cloud computing network. In *2010 3rd IEEE International Conference on Computer Science and Information Technology (ICCSIT 2010)*, pages 108–113. IEEE.