OpenPEX: An Open Provisioning and EXecution System for Virtual Machines

Srikumar Venugopal[1] James Broberg[2] Rajkumar Buyya[2]

[1] School of Computer Science and Engineering,
University of New South Wales, Australia
srikumarv@cse.unsw.edu.au
[2] Department of Computer Science and Software Engineering,
The University of Melbourne, Australia
{brobergj, raj}@csse.unimelb.edu.au

Technical Report
UNSW-CSE-TR-0918
October 2009

School of Computer Science and Engineering
The University of New South Wales
Sydney 2052, Australia

Abstract

Virtual machines (VMs) have become capable enough to emulate full-featured physical machines in all aspects. Therefore, they have become the foundation not only for flexible data center infrastructure but also for commercial Infrastructure-as-a-Service (IaaS) solutions. However, current providers of virtual infrastructure offer simple mechanisms through which users can ask for immediate allocation of VMs. More sophisticated economic and allocation mechanisms are required so that users can plan ahead and IaaS providers can improve their revenue. This paper introduces OpenPEX, a system that allows users to provision resources ahead of time through advance reservations. OpenPEX also incorporates a bilateral negotiation protocol that allows users and providers to come to an agreement by exchanging offers and counter-offers. These functions are made available to users through a web portal and a REST-based Web service interface.

1 Introduction

In the past, many networked services (such as web sites, databases and computational services) were hosted on dedicated physical hardware, which were configured exclusively to suit application dependent requirements. However, recent hardware and software advances have made it possible to host these services within Virtual Machines (VMs) on 'commodity x86 hardware with minimal overhead. Virtualisation solutions such as Xen [2] and VMWare [1] create a virtual representation of a complete physical machine, enabling operating systems (and any running applications) to be de-coupled from the physical hardware. This allows improved utilisation and consolidation of computing infrastructure by multiplexing many VMs onto one physical host.

These developments provide an interesting framework where a lightweight VM image can be a unit of execution (i.e. instead of a task or process on a shared system) and migration. Migration of VMs on the same subnet can be achieved in a totally transparent, work-conserving fashion without the running applications, nor any dependant clients or external resources being aware that it has occurred [9, 5]. Also, VMs enable workload isolation and can be shutdown without adverse effects on other concurrent VMs.

The ability to fashion VMs into expendable resource units has led to their use for ad-hoc deployment of computational infrastructure in order to meet sudden spikes in resource demand. VMs also enable users to create computing environments customised to suit the requirements of their specific e-Science or e-Business applications. These capabilities have led to the advent of infrastructure provisioning services both private (within an enterprise or organisation), or commercial. Providers and consumers of such services negotiate Service Level Agreements (SLAs) that encapsulate the user requirements in terms of Service Level Objectives (SLOs), and the rewards and penalties for meeting and violating them respectively. Therefore, the provider's aim is to maximise its own return of investment by maximising resource utilisation while avoiding or minimising penalties caused by SLA violations.

VMs have also become the 'enabling technology' behind the recent emergence of Cloud Computing [4]. Infrastructure as a Service (Iaas) Providers such as Amazon EC2, GoGrid or Mosso Cloud Servers have emerged that offer virtualised machines (or resource slices) that can be obtained under a pay-per-use arrangement (i.e. no commitment required, utility style pricing). Users can take advantage of elastic capacity, where they can scale up and scale down on demand using a self-service interface, such as a Web Service or Web Portal. The resources themselves (such as compute and storage) are highly abstracted or virtualised.

However, IaaS service models are evolving and currently, most providers operate on a lease model – the users pay for the time the VM was active. Also, the choices available to the user in terms of specifying requirements are limited. Such models do not allow for more flexible strategies where the user can reduce both his costs and risk by booking his resources in advance. An advance reservation

provides a guaranteed allocation of the resources at the needed time to the consumer and helps the provider plan capacity requirements better. However, advance reservations induce new challenges in resource management and require new architectures for realisation. In this paper, we introduce OpenPEX, a utility-based virtual infrastructure manager that enables users to reserve VM instances in advance. OpenPEX also offers a bilateral negotiation protocol that allows users and providers to exchange offers and counter-offers, and come to an agreement that is mutually beneficial. In the next section, we distinguish the contributions of OpenPEX from the state-of-the-art. Section 3 discusses the design and implementation of OpenPEX at length. Section 4 discusses the Web Service interface to OpenPEX and finally, we conclude the paper with details on our future plans for the system.

2 Related Work

Virtual Machine technology has become an essential enabling technology of Cloud Computing environments. Cloud Computing is a style of computing where resources can be obtained in a pay-per-use manner (no commitment, utility pricing). Such resources have elastic capacity, where they can be scaled up and down on demand. Resources are highly abstracted and virtualised, and can be obtained via a self-service interface.

VMs are highly attractive to manage resources in such environments as they improve utilisation by multiplexing many VMs on one physical host (consolidation), allow agile deployment and management of services, provide on demand cloning, (live) migration and checkpoint which improves reliability. Furthermore, a VM can be a self-contained unit of execution and migration. As such, effective management of VMs and Virtual Machine infrastructure is critical for any Cloud Computing Infrastructure as a Service (IaaS) provider.

Clouds are often defined as public (i.e. external), private (i.e. internal) or hybrid (a combination thereof). A Public Cloud typically has a publicly accessible, self-service interface that is made available via well-defined and published Web Services (i.e. SOAP or REST). Most public clouds at present are pay-for-use but there is no ongoing contract or commitment required, i.e., the user pays only for the capacity provisioned at a particular time.

A Private Cloud aims to emulate a public cloud on private / internal resources while maintaining control over an organisation's resources and data. It therefore aims to provide the benefits of clouds (elasticity, dynamic provisioning, multiplexing) while meeting security and governance requirements in the organisation. Private clouds also allow an organisation to scale out by employing resources from public clouds when required and acceptable.

A Hybrid Cloud is a combination of private/internal and external cloud resources that enables outsourcing of non-critical functions whilst keeping the remainder internal. The key functionality is the ability to use and release resources from public clouds as and when required. This is used to handle sudden demand

surges ('flash crowds') and is called as 'cloudbursting'.

2.1 Public IaaS Cloud Services

Amazon Elastic Compute Cloud (EC2) is an IaaS service that provides resizable compute capacity in the cloud. These services can be leveraged via Web Services (SOAP or REST),a web-based AWS Management Console or the EC2 Command Line Tools. The Amazon service provides hundreds of pre-made AMIs (Amazon Machine Image), giving users a wide choice of operating systems (i.e. Windows or Linux) and pre-loaded software. Instances come in different sizes, from Standard Instances (S, L, XL), which have proportionally more RAM than CPU, to High CPU Instances (M, XL), which have proportionally more CPU than RAM. A user can deploy these instances in two different regions, US-East and EU-West Regions, with EU instances costing more per hour than their US counterparts.

Amazon EC2 provides an alternative to it's on-demand instances, known as a *reserved instance*. This facility offers a number of benefits over simply requesting instances on demand, as it provides a lower per-hour rate, and provides assurances that any reserved instance you launch is guaranteed to succeed (provided you have booked them in advance). That is, users of such instances should not be affected by any transient limitations in EC2 capacity.

GoGrid Cloud Hosting provides pre-made Windows and Linux images, with a range of 'Value-added' stacks on top. GoGrid offers a range of (fixed) instance sizes which can be procured on demand. GoGrid is notable in that it offers hybrid dedicated / cloud server infrastructure, where cloud resources can be mixed with existing physical resources. Free hardware load balancing and auto-scaling is provided as part of the service.

Mosso Cloud Servers fixed size Cloud Servers, where server size measured by RAM, not CPU Mosso offers range of pre-made Linux instances which can be procured on demand. Like GoGrid, Mosso provides a hybrid dedicated / cloud server infrastructure, where cloud resources can be mixed with existing physical resources.

2.2 Private IaaS Cloud Platforms

Many different platforms exist to assist with the deployment and management of virtual machines on a virtualised cluster (i.e. a cluster running Virtual Machine software). Such platforms are often referred to as 'Private Clouds', as they can bring the benefits of Cloud Computing (such as elasticity, dynamic provisioning and multiplexing workloads onto fewer machines) into local clusters.

Eucalyptus [11, 10] is an open-source (BSD-licensed) software infrastructure for implementing Infrastructure as a Service (IaaS) Compute Cloud on commodity hardware. Eucalyptus is notable by offering a Web Service interface that is fully Amazon Web Services (AWS) API compliant. Specifically, it emulates Amazon's Elastic Compute Cloud (EC2), Simple Storage Service (S3) and Elastic Block

Store (EBS) services at the API level. However, as the implementation details of Amazon's services are not published, Eucalyptus' internal implementation would differ.

OpenNebula [13] is an open-source Virtual Infrastructure management software that supports dynamic resizing, partitioning and scaling of computing resources. OpenNebula can be deployed in a private, public or hybrid Cloud models. The OpenNebula software turns an existing cluster into private cloud, which can be used privately or can expose service to public via XML-RPC Web Services. The integration of Cloud plugins (EC2, GoGrid) enable hybrid model, where you can mix and match private and public resources. Haizea [13] has extended OpenNebula further, allowing resource providers to lease their resources using sophisticated leasing arrangements, instead of only providing on-demand VMs like most other IaaS services.

Nimbus [7] (formerly known as Globus Virtual Workspaces [8]) is another open-source framework to turn an existing cluster into an IaaS cloud. Nimbus offers two Web Service interfaces; Amazon EC2 WSDLs and Grid community WSRF. There are several test 'Science Clouds' deployed at Universities of Chicago, Florida, Purdue and Marsarky University.

2.3 Issues with existing solutions

With the exception of OpenNebula (when used in conjunction with the Haizea extension) and Amazon EC2 (when used with Reserved Instances), none of the above public or private platforms offer the ability to perform an Advanced Reservation of computing resources, rather they only supply on-demand capacity on a best-effort basis (i.e. if adequate resources are available). Furthermore, none of these platforms provide an alternate offer (that is, a modified offering that can satisfy a users request that may differ from their initial request) in the event that the system cannot satisfy a user's specific request for resources.

Whilst Haizea supports a form of Advanced Reservation (which it denotes as *advanced reservation leases*), if the request cannot be satisfied there is no recourse - the request will be rejected. Under the same circumstances, the OpenPEX system enacts a bilateral negotiation protocol that allows users and providers to come to an agreement by exchanging offers and counter-offers, so a user's advanced reservation request can be satisfied.

Amazon EC2 offers its own variation on the notion of Advanced Reservation with its Reserved Instances product. However, you need to purchase a Reserved Instance for every instance you wish to guarantee to be available at some point in the future. This essentially requires the end user to forecast exactly how many they will require in advance. Acquisition of Reserved Instance is not instantaneous either, in the authors' experience a request for an Reserved Instance has taken more than an hour on previous occasions.

2.4 Utility Computing Platforms

With increasing popularity and usage, large Grid installations are facing new problems, such as excessive spikes in demand for resources coupled with strategic and adversarial behaviour by users. Traditional Grid resource management techniques did not ensure fair and equitable access to resources in many systems. Traditional metrics (throughput, waiting time, slowdown) failed to capture the more subtle requirements of users. There were no incentives for users to be flexible about resource requirements or job deadlines, nor provisions to accommodate users with urgent work.

In such systems, users assign a "utility" value to their jobs, where utility is a fixed or time-varying valuation that captures various QoS constraints (deadline, importance, satisfaction). The valuation is amount they are willing to pay a service provider to satisfy demands. Service providers attempt to maximise their own utility, where utility may directly correlate with their profit. Providers can prioritise high yield (i.e. profit per unit of resource) user jobs, and shared Grid systems are then viewed as a marketplace, where users compete for resources based on the perceived utility / value of their jobs. Further information and comparison of these utility computing environments are available in extensive survey of these platforms [3].

3 OpenPEX

OpenPEX was constructed around the notion of using advance reservations as the primary method for allocating VM instances. The use case followed here is of a user who, either through a web portal or through the web service makes a reservation for any number of instances of a Virtual Machine that have to be started at a specific time and have to last for a specific duration. The VM is described by a template that is already registered in the system. If the request can be satisfied for the price asked for by the user, then OpenPEX creates the reservation, else it creates a counter-offer with an alternate time interval where the request can be accommodated. The counter offer may also instead specify a different price for the original time interval. Once the reservations have been finalised, the user can chose to activate the reservation or have OpenPEX automatically start the instances when required.

These requirements motivate a resource management system that is able to manage physical nodes in such a manner that it maintains the capacity to satisfy as many advance reservation requests as possible. Such adaptive management may be enabled by a variety of techniques including load forecasting, migrating existing VMs to other resources, or/and suspending some of the VMs in order to increase the available resource share.

Figure 3.1 shows the architecture of the OpenPEX Resource Manager. The Resource Manager has the following components:

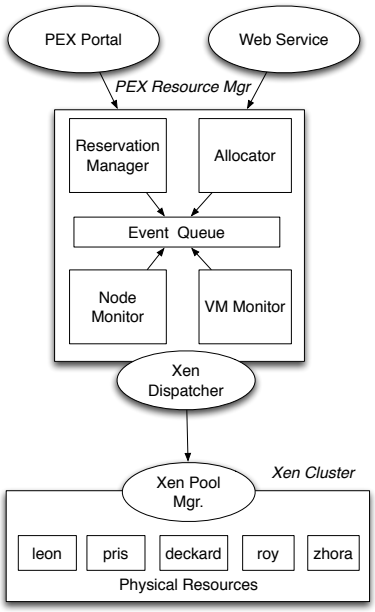


Figure 3.1: The OpenPEX Resource Manager.

Reservation Manager - This component interacts with the users through the portal or the web service, and receives incoming requests. It examines them to check whether they are feasible according to the reservation policy employed, and creates counter-offers when required.

Scheduler - Manages the allocation of VMs to physical nodes. The scheduler's roles are to: create capacity for new VMs by triggering migration of existing VMs to other nodes, or by suspending long running VMs; to identify physical nodes on which reservations can be activated; and to react to events such as the loss of a physical node.

Node Monitor - Monitors the health and load of the physical nodes.

VM Monitor - Monitors the health of the VMs that have been started in OpenPEX. It detects events such as a VM shut down by a user from the inside, a VM suddenly crashing, or a VM being unresponsive.

Dispatcher - Interacts with the virtual machine manager or the hypervisor on the physical nodes. It relays commands such as create, start or shutdown a VM to the virtual machine manager. The dispatcher is the only component that is specific to the underlying virtual machine manager, the rest of OpenPEX is designed to be independent of the underlying infrastructure.

All these components are connected to an **Event Queue** that acts as a simple message bus for the system. The Event Queue also enables scheduling of future

tasks by allowing delayed events. The entire system is backed by a **Persistence** layer that saves the current state to a database.

3.1 Negotiating Advance Reservations

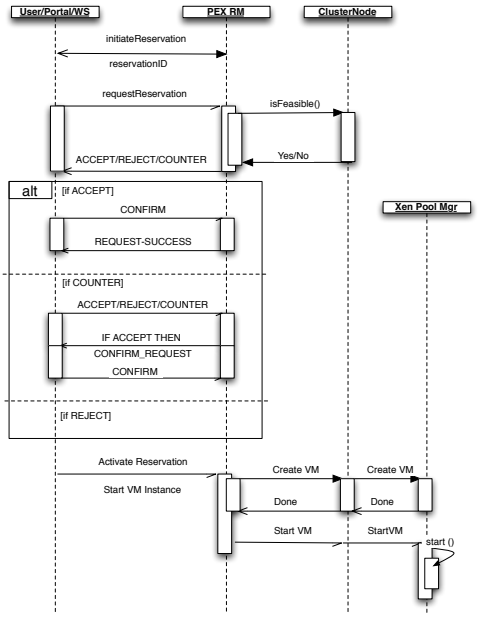


Figure 3.2: Alternate Offers Negotiation for Advance Reservations.

As described previously, OpenPEX allows advance reservations to be negotiated bilaterally between the producers and consumers. As described previously, OpenPEX allows advance reservations to be negotiated bilaterally between the producers and consumers. For this, we have employed a protocol based on the Alternate Offers mechanism [12] which was previously used for negotiation of SLAs in an enterprise Grid framework [14]. The implementation of this protocol in OpenPEX is shown in Figure 3.2. The user opens the interaction by sending an `initiateReservation` request in reply to which OpenPEX returns a unique `reservationID` identifier. This identifier acts as a handle for the session and if the reservation goes through, then until its life-cycle is complete. The user then submits a proposal through a `requestReservation` call. The proposal contains a description of the VM being requested (e.g. instance size), the number of instances required, the start time for activating the reservation and the duration for which the reservation is required. The instance sizes are detailed in the next section and examples of these descriptions are given in Section 4. In return, OpenPEX can respond with: `ACCEPT`, if the proposal is acceptable; `REJECT`, if the proposal cannot be satisfied in any manner; and `COUNTER`, if the reservation required cannot be fulfilled with the parameters given in the proposal but an alternative can be gen-

Table 3.1: List of available VM configurations in OpenPEX.

Size	Configuration
SMALL	1 CPU, 768 MB RAM, 10 GB HDD
MEDIUM	2 CPU, 1.5 GB RAM, 20 GB HDD
LARGE	3 CPU, 2.5 GB RAM, 40 GB HDD
XLARGE	4 CPU, 3.5 GB RAM, 60 GB HDD

erated instead. With the last option, OpenPEX returns an alternative (or counter) proposal generated by replacing terms of the user's original proposal with those acceptable to it. For example, a user could ask for 5 instances of a Virtual Machine of small instance size (refer Section 3.2) and with Red Hat Enterprise Linux operating system. These instances have to be started at 10:00 a.m. on August 21, 2009 for six days after which they can be shut down. However, OpenPEX may not be able to provision these instances on August 21, but may have free nodes for six days starting August 23. In this case, it will generate a counter proposal that replaces only the start time with the new start time in the user's original proposal. If OpenPEX is not able to provision the VMs in any case (e.g. number of instances requested exceeds its capacity), then the proposal is rejected.

When the user receives an `ACCEPT`, he can then reply with `CONFIRM` to confirm the reservation. In reply to a `COUNTER`, the user has the same three reply options. In case the user accepts the counter-proposal (through the reply `ACCEPT`), OpenPEX sends back a `CONFIRM-REQUEST` so that the user can reply back with a `CONFIRM` to confirm the reservation. This extra step is necessary as, even though the protocol is bilateral, only OpenPEX can confirm a reservation.

Once the reservation is confirmed, the user can activate it by instantiating the VMs in the reservation after the agreed-upon start time. The user can start or shutdown VMs at any time during the course of the reservation. Once the duration is over, the reservation expires, and all active VMs on that reservation are shutdown.

3.2 Resource Management in OpenPEX

Users are only able to ask for standardised configurations of virtual machines from the OpenPEX Resource Manager. These configurations depict the "sizes" of the virtual machines as given in Table 3.1. The virtual machines can be paired with an operating environment chosen from the templates available in the OpenPEX database.

When a user request arrives at the Reservation Manager, the individual nodes are polled to determine which of them are free at the requested time. If more than one nodes are free, then the Manager choses the most loaded of them to host the request. In case, there are no free nodes available, an alternate time slot is requested from each of the nodes. The node that provides a starting time closest to the original request is temporarily locked, and the new starting time is sent as an alternate offer to the user via a COUNTER reply in the negotiation protocol.

3.3 Implementation Details

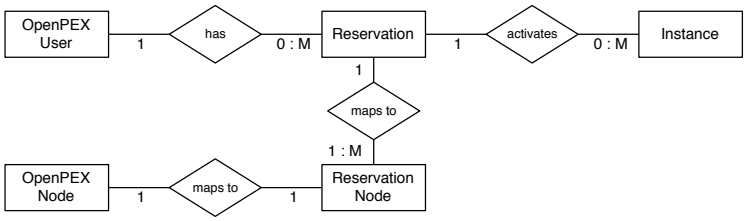


Figure 3.3: OpenPEX Entity Relationship Diagram.

We have a cluster with 5 computing nodes, each with 2 dual-core Intel Xeon processors and 4 GB RAM, and a storage node with 1 TB of disk space. We use Xen Enterprise v5.0 to manage this cluster, and therefore, the Xen hypervisor is installed on each computing node. The computing nodes are organised into a pool and one of the machines is designated as the pool manager with which external programs can communicate using the Xen Enterprise API. The Xen API is full-fledged allowing developers to manipulate every aspect of a VM's life-cycle (creation, starting, migration, shutdown, and deletion).

OpenPEX is developed completely in Java and is deployed as a service in an application container on the cluster head node (or the storage node). It communicates with the pool manager using the Xen API and uses the Java Persistence API (JPA) for the persistence back-end. The database structure for PEX is depicted in Figure 3.3 and follows Object-Relational Mapping (ORM) for easy development and extensibility.

3.4 Web Portal Interface

The OpenPEX system provides an easy to use Web Portal interface, enabling the user to access all the functionality of the OpenPEX. Users can access the system via a web browser, register for an account and login to the system. Upon logging in they will be greeted by a simple Welcome Screen depicted in Figure 3.4, which shows what functions are available to the end user.

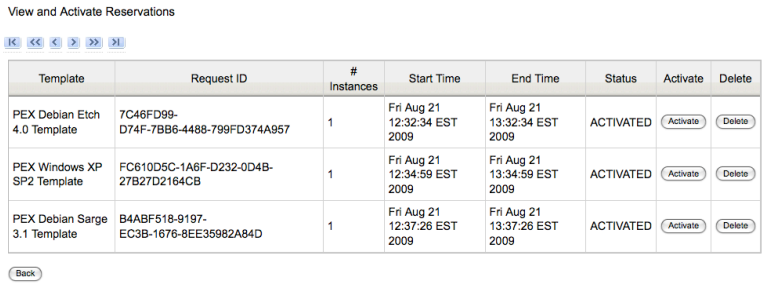
Figure 3.4: OpenPEX Welcome Screen.

The user can choose to make a new reservation, where they can choose the size of the reservation they wish to make (from the choices listed in Table 3.1), the start and end time, the template (i.e. Operating System) they wish to use and the

number of instances they require. Their request can be accepted or they can enter into a negotiation until they come to an agreement with the OpenPEX cluster.

Once this process has occurred, they can view their existing reservations and activate any unclaimed reservations via the Reservations screen. Figure 3.5 shows the Reservations screen with three reservations. If a reservation was not yet activated, a user can choose to delete it (if they no longer required it) or activate it, so the associated instances can start at the appropriate start time.

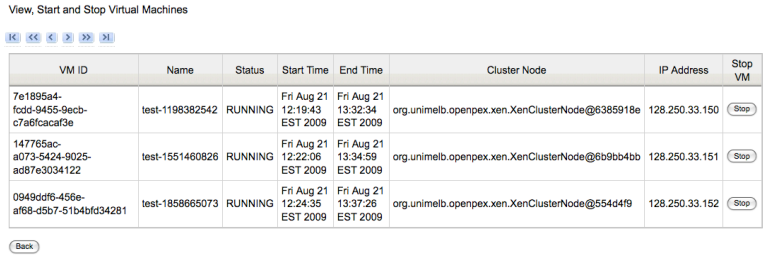
Virtual Machine instances can be viewed and manipulated via the Instances screen depicted in Figure 3.6. Here the user can view salient information regarding their VM instance, such as it's machine name, status (e.g. HALTED, RUNNING, PAUSED, SUSPENDED). start time, end time, and IP address. An instance can also be stopped early (i.e. before it's designated end time) if desired.


View and Activate Reservations

Template	Request ID	# Instances	Start Time	End Time	Status	Activate	Delete
PEX Debian Etch 4.0 Template	7C46FD99-D74F-7BB6-4488-799FD374A957	1	Fri Aug 21 12:32:34 EST 2009	Fri Aug 21 13:32:34 EST 2009	ACTIVATED	Activate	Delete
PEX Windows XP SP2 Template	FC610D5C-1A6F-D232-0D4B-27B27D2164CB	1	Fri Aug 21 12:34:59 EST 2009	Fri Aug 21 13:34:59 EST 2009	ACTIVATED	Activate	Delete
PEX Debian Sarge 3.1 Template	B4ABF518-9197-EC3B-1676-8EE35982A84D	1	Fri Aug 21 12:37:26 EST 2009	Fri Aug 21 13:37:26 EST 2009	ACTIVATED	Activate	Delete

Back

Figure 3.5: OpenPEX Reservations Screen.


View, Start and Stop Virtual Machines

VM ID	Name	Status	Start Time	End Time	Cluster Node	IP Address	Stop VM
7e1895a4-fcdd-9455-9ecb-c7a6fcacaf3e	test-1198382542	RUNNING	Fri Aug 21 12:19:43 EST 2009	Fri Aug 21 13:32:34 EST 2009	org.unimelb.openpex.xen.XenClusterNode@6385918e	128.250.33.150	Stop
147765ac-a073-5424-9025-ad87e3034122	test-1551460826	RUNNING	Fri Aug 21 12:22:06 EST 2009	Fri Aug 21 13:34:59 EST 2009	org.unimelb.openpex.xen.XenClusterNode@6b9bb4bb	128.250.33.151	Stop
0949ddf6-456e-af68-d5b7-51b4bfd34281	test-1858665073	RUNNING	Fri Aug 21 12:24:35 EST 2009	Fri Aug 21 13:37:26 EST 2009	org.unimelb.openpex.xen.XenClusterNode@554d4f9	128.250.33.152	Stop

Back

Figure 3.6: OpenPEX Instances Screen.

4 RESTful Web Service Interface

It is essential to provide programmatic access to the functions and capabilities of an OpenPEX cluster, in order for users to be able to dynamically request reservations from the system (i.e. scaling out during periods of peak load), or even to integrate an OpenPEX system into a wider pool of computing resources. As such, the full functionality of the OpenPEX system is exposed via Web Services, which are implemented in a RESTful (REpresentational State Transfer) style [6].

Table 4.1: OpenPEX RESTful Endpoints.

OpenPEX Operation	HTTP	Endpoint	Params	Return type
Create reservation	POST	/OpenPEX/reservations	JSON	JSON
Update reservation	PUT	/OpenPEX/reservations/requestId	JSON	JSON
Delete reservation	DELETE	/OpenPEX/reservations/requestId	None	HTTP 200 (OK)
Activate reservation	PUT	/OpenPEX/reservations/requestId/activate	None	HTTP 200 (OK)
Get reservation information	GET	/OpenPEX/reservations/requestId	None	JSON
List reservations	GET	/OpenPEX/reservations	None	JSON
Get instance information	GET	/OpenPEX/instances/vm_id	None	JSON
List instances	GET	/OpenPEX/instances	None	JSON
Stop instance	PUT	/OpenPEX/instances/vm_id/stop	None	HTTP 200 (OK)
Reboot instance	PUT	/OpenPEX/instances/vm_id/reboot	None	HTTP 200 (OK)
Delete instance	DELETE	/OpenPEX/instances/vm_id	None	HTTP 200 (OK)

```
{
   "duration": 3600000,
   "numInstancesFixed": 1,
   "numInstancesOption": 0,
   "startTime": "Mon, 17 Aug 2009 04:49:03 GMT",
   "template": "PEX Debian Etch 4.0 Template",
   "type": "XLARGE"
}
```

Figure 4.1: Create reservation JSON body

The REST-style architecture provides a clear and clean delineation between the functions of the client and the server. A client performs operations on resources (such as *reservations* and *instances*) which are identified through standard URIs. The server returns a JSON[1] *representation* of the resource back to the client to indicate the current state of that resource. Clients can modify and delete these resources by altering and returning their representations as required. A client could be a Java or Python program, or a Web Portal management interface for the OpenPEX system.

Table 4 lists the functions exposed by the Web Service interface, along with their corresponding HTTP methods and endpoints. Some calls require a JSON body whilst others trigger their functionality by simply being accessed. The calls typically return a JSON object or array, or simply a HTTP code denoting whether an operation was a success or failure. All the methods listed require HTTP basic authentication. From this table we can see that the full OpenPEX life-cycle is exposed via the Web Services interface. Customers can create a new reservation, engage in the bilaterally negotiation (via the Alternate Offers protocol described earlier in this paper) via the `/OpenPEX/reservations` endpoint, and finally activate their reservation. Once a reservation has been activated, the corresponding Virtual Machine instances are started at their designated start time. A user has

[1]The application/json Media Type for JavaScript Object Notation (JSON) - http://tools.ietf.org/html/rfc4627

```
{
   "proposal":    {
      "duration": 3600000,
      "id": "5D0FA0EB-90A8-F4E6-1DFF-61B2CEC6AD91",
      "numInstancesFixed": 1,
      "numInstancesOption": 0,
      "startTime": "Mon, 17 Aug 2009 04:49:03 GMT",
      "template": "PEX Debian Etch 4.0 Template",
      "type": "XLARGE",
      "userid": 1
   },
   "reply": "ACCEPT"
}
```


Figure 4.2: Reply to reservation request


```
{
   "proposal":    {
      "duration": 3600000,
      "id": "F07640D4-32BC-DDB6-457E-32B5595BA066",
      "numInstancesFixed": 1,
      "numInstancesOption": 0,
      "startTime": "Mon, 17 Aug 2009 05:52:31 GMT",
      "template": "PEX Debian Etch 4.0 Template",
      "type": "XLARGE",
      "userid": 1
   },
   "reply": "COUNTER"
}
```


Figure 4.3: Counter Reply to reservation request

control over these instances via the `/OpenPEX/instances` endpoint, where they can stop, reboot or delete these instances.

Figure 4.1 depicts the JSON body for a new reservation call. A user specifies the duration of the reservation, the number of instances required, the start time, the desired template (e.g. Operating System) required and the type (Table 3.1) of instance required. These preferences are expressed in the JSON body of the call. OpenPEX will then respond with a JSON reply that indicates the outcome of the request, which could be an acceptance of the proposed reservation (shown in Figure 4.2), a counter offer indicated an alternate reservation that could satisfy the user (shown in Figure 4.3), or an outright rejection of the proposed reservation.

Upon successfully obtaining a reservation in the system, a user can get the reservation record and activate the reservation. Once the reservation has been activated the user can then operate on the instances themselves, obtaining the instance record, and control the state of the Virtual Machine instance itself by stopping, rebooting or deleting it.

5 Conclusion and Future Work

In this paper we introduced OpenPEX, a system that allows users to provision resources ahead of time through advance reservations, instead of being limited to on-demand, best-effort resource acquisition. OpenPEX also incorporates a novel bilateral negotiation protocol that allows users and providers to come to an agreement by exchanging offers and counter-offers, in the event that a user's original request cannot be precisely satisfied.

The fundamental aim of OpenPEX was to harness virtual machines for adaptive provisioning of services on shared computing resources. Adaptive provisioning may involve a combination of: 1) creating of new VMs to meet increase in demand; 2) migrating existing VMs to other available resources; and/or 3) suspending execution of some VMs in order to increase the resource share available to others. These techniques are, however, governed by negotiated agreements between the users and the resource providers, and between providers that encapsulate costs and guarantees for deployment and maintenance of VMs for services. Demand and supply for services in such an environment is, therefore, mediated by market-driven resource management mechanisms, thereby leading to a so-called utility computing environment.

As such, we are endeavouring to implement provisional market-based resource management techniques in OpenPEX system to collect pricing and utilisation data, and introduce policies and strategies for managing virtual machines in a market-driven utility computing environment. We intend to achieve this by:

1. Soliciting a wide range of users (from other faculties and other collaborating Universities) to run VM encapsulated workloads on the test-bed.
2. Measuring crucial pricing (using simulated currency mechanism) and usage data from users of the system, which is difficult to obtain from commercial computing centres and largely absent from the literature.
3. Formulating market-driven policies for scheduling and migration in VM platforms based on data collected.
4. Integrating market-driven scheduling and migration policies into OpenPEX.
5. Evaluating strategies for negotiating among multiple VM providers and users based on market conditions in conjunction with the proposed market-drive policies.

We also intend to evaluate known distributed scheduling policies for allocating and migrating VMs. We intend to achieve this by:

1. Evaluate the performance of existing distributed scheduling algorithms for allocating and migrating VMs. These include Random/Round Robin, Least-Loaded-First (e.g. Shortest Queue, Least-Work-Remaining) and size based partitioning approaches.

2. Comparing the cost versus benefit of work-conserving (saving state) and non-work-conserving (discarding state) migration policies in a VM setting for different workloads.

Bibliography

[1] K. Adams and O. Agesen. A comparison of software and hardware techniques for x86 virtualization. In *Proceedings of the 12th international conference on Architectural support for programming languages and operating systems*, pages 2–13, San Jose, California, USA, 2006. ACM.

[2] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the art of virtualization. *ACM SIGOPS Operating Systems Review*, 37(5):164–177, 2003.

[3] J. Broberg, S. Venugopal, and R. Buyya. Market-oriented Grids and Utility Computing: The state-of-the-art and future directions. *Journal of Grid Computing*, 6(3):255–276, 2008.

[4] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic. Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation Computer Systems*, 25(6):599–616, June 2009.

[5] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield. Live migration of virtual machines. In *Proceedings of the 2nd conference on Symposium on Networked Systems Design \& Implementation - Volume 2*, pages 273–286. USENIX Association, 2005.

[6] R. T. Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, 2000.

[7] K. Keahey, R. Figueiredo, J. Fortes, T. Freeman, and M. Tsugawa. Science Clouds: Early Experiences in Cloud Computing for Scientific Applications. *Cloud Computing and Applications*, 2008, 2008.

[8] K. Keahey, I. Foster, T. Freeman, and X. Zhang. Virtual workspaces: Achieving quality of service and quality of life in the grid. *Scientific Programming*, 13(4):265–275, 2005.

[9] M. Nelson, B. Lim, and G. Hutchins. Fast transparent migration for virtual machines. In *Proceedings of the annual conference on USENIX Annual Technical Conference*, pages 25–25, Anaheim, CA, 2005. USENIX Association.

[10] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov. Eucalyptus: A Technical Report on an Elastic Utility Computing Archietcture Linking Your Programs to Useful Systems UCSB Computer Science Technical Report Number 2008-10.

[11] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov. The Eucalyptus Open-source Cloud-computing System. *Proceedings of Cloud Computing and Its Applications*, 2008.

[12] A. Rubinstein. Perfect equilibrium in a bargaining model. *Econometrica*, 50(1):97–109, January 1982.

[13] B. Sotomayor, R. Montero, I. Llorente, I. Foster, and F. de Informatica. Capacity Leasing in Cloud Systems using the OpenNebula Engine. *Cloud Computing and Applications*, 2008, 2008.

[14] S. Venugopal, X. Chu, and R. Buyya. A negotiation mechanism for advance resource reservations using the alternate offers protocol. In *Proceedings of the 16th International Workshop on Quality of Service (IWQoS 2008)*, pages 40–49. IEEE Computer Society Press, Los Alamitos, CA, USA, June 2008.