Dynamic Resource Management in IaaS Cloud

Study Summary / Project Report

by

PANKAJ MALHOTRA
www.pankajmalhotra.com

Abstract

Infrastructure as a Service (IaaS) is one of the three fundamental service models of cloud computing alongside Platform as a Service (PaaS) and Software as a Service (SaaS). Many organizations are increasingly moving their IT infrastructure to the cloud and this has led to an abrupt increase in number of companies emerging as cloud infrastructure provider which indirectly leads to a very tough competition for providing resources at lowest cost possible while maintaining the performance levels of applications deployed on this kind of infrastructure.

The workload patterns of applications vary depending on the kind of application, typically there are four categories, predictable burst, unpredictable burst, on and off and growing fast. These workload patterns determine the amount of resources that the application needs from infrastructure provider and predicting the amount of resources required can considerably decrease the operating cost of such machines.

There have been numerous solutions proposed for predicting such bursts and take actions accordingly, these solutions depend on the infrastructure provider, for eg: Elastic Load Balancing provided by Amazon Web Services aims to distribute load over virtual machines by redirecting the amount of traffic and distributing it equally and adding more machines to the cluster, which is also called horizontal scaling. These providers usually have API integrations and hooks which constantly monitor data and let the user take actions based on the manipulation of data using machine learning models.

Main problem with such kind of horizontal scaling solutions is that for providing an additional VM it takes about 50-800 seconds, for each VM operating system is required which limits the additional capacity usable by the application, this can also sometime cause over provisioning and always requires a load balancer to maintain the resource distribution.

Very few infrastructure providers provide option of vertical scaling which is much costlier than horizontal scaling and requires a considerable amount of resources as compared to horizontal scaling.

Ideally the models for autoscaling infrastructure should combine both techniques and scale infrastructure accordingly, but there still are limitations for providing and maintaining large virtual machines especially in case of vertical scaling, also the distribution of resources across data centers provide a network overhead and sending

across the state and images of virtual machines, which leads to increased response time and delays in scaling infrastructure.

In this project I have tried to explore various such techniques possible, starting from simple horizontal scaling triggers on AWS using the Cloud Watch API from AWS itself. The results we got show that horizontal scaling is easily achievable and there exist many academic and research papers tackling the problem of generating triggers for autoscaling, such as various machine learning models based on kNN predictions and much more.

We gradually move towards achieving vertical scaling, and the main challenge we tackle there is the hot plugging of resources from hypervisor to virtual machines. Taking data and experimenting with a lot of hypervisors and Operating Systems we narrow down to using a XEN hypervisor, which also is the backbone for AWS host machines.

XEN provides a wide range of CPU schedulers among which credit based scheduling is the most commonly used, we target the same to increase and decrease the number of virtual cpu's (vcpu's) for a live virtual machine.

A virtual CPU (vCPU) also known as a virtual processor, is a physical central processing unit (CPU) that is assigned to a virtual machine (VM). By default, virtual machines are allocated one vCPU each. If the physical host has multiple CPU cores at its disposal, however, then a CPU scheduler assigns execution contexts and the vCPU essentially becomes a series of time slots on logical processors. We use XL - Xen management tool, based on LibXenlight to set the vcpu parameter for a DomainU, keeping in mind the maximum number of vcpu's that can be added cannot exceed the maximum declared value when creating the DomU and ideally can never be 0.

We lay a concrete model for CPU hot plugging, similar approach can be used to achieve scale up and down of memory and disk (Block Device) on a virtual machine running on a Xen hypervisor also called a Xen Domain.

Using such techniques this project can further be extended and integrated in Open Source cloud computing softwares like OpenStack and more experimentation can be done for different hypervisors and Operating Systems which can later be modeled as a hybrid tool to achieve both horizontal and vertical scaling based on triggers or alerts.

Contents

Chapter Page

Chapter 1

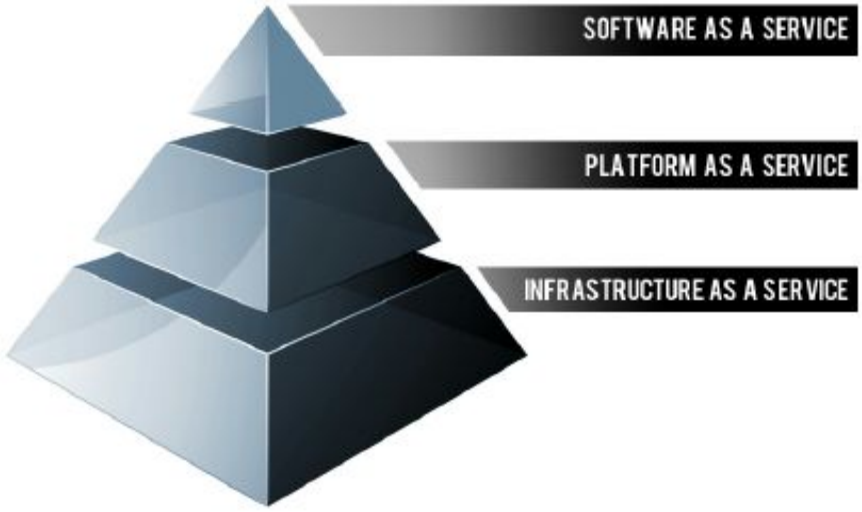
Introduction

Cloud Computing Services provide information technology (IT) as a service over the Internet or dedicated network, with delivery on demand, and payment based on usage. Cloud Computing is often described as a stack, as a response to the broad range of services built on top of one another under the moniker “Cloud”. The generally accepted definition of Cloud Computing comes from the National Institute of Standards and Technology (NIST). The NIST definition runs to several hundred words but essentially says that:

Cloud computing is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.
What this means in plain terms is the ability for end users to utilize parts of bulk resources and that these resources can be acquired quickly and easily.
NIST also offers up several characteristics that it sees as essential for a service to be considered “Cloud”. These characteristics include:

• On-demand self-service. The ability for an end user to sign up and receive services without the long delays that have characterized traditional IT
• Broad network access. Ability to access the service via standard platforms (desktop, laptop, mobile etc)
• Resource pooling. Resources are pooled across multiple customers
• Rapid elasticity. Capability can scale to cope with demand peaks
• Measured Service. Billing is metered and delivered as a utility service

Among the many types of cloud computing services delivered internally or by third party service providers, the most common are:

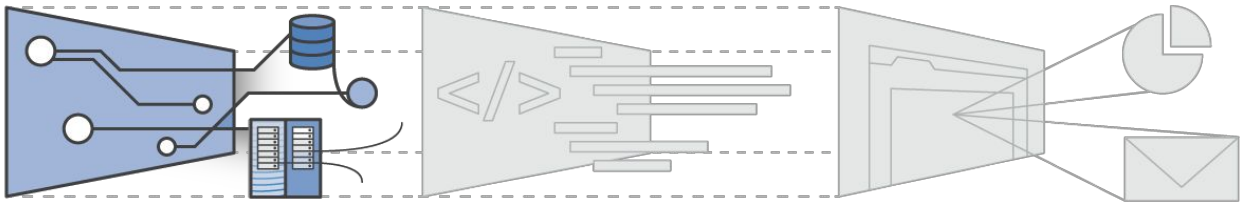


- Software as a Service (SaaS) – software runs on computers owned and managed by the SaaS provider, versus installed and managed on user computers. The software is accessed over the public Internet and generally offered on a monthly or yearly subscription.

- Infrastructure as a Service (IaaS) – compute, storage, networking, and other elements (security, tools) are provided by the IaaS provider via public Internet, VPN, or dedicated network connection. Users own and manage operating systems, applications, and information running on the infrastructure and pay by usage.

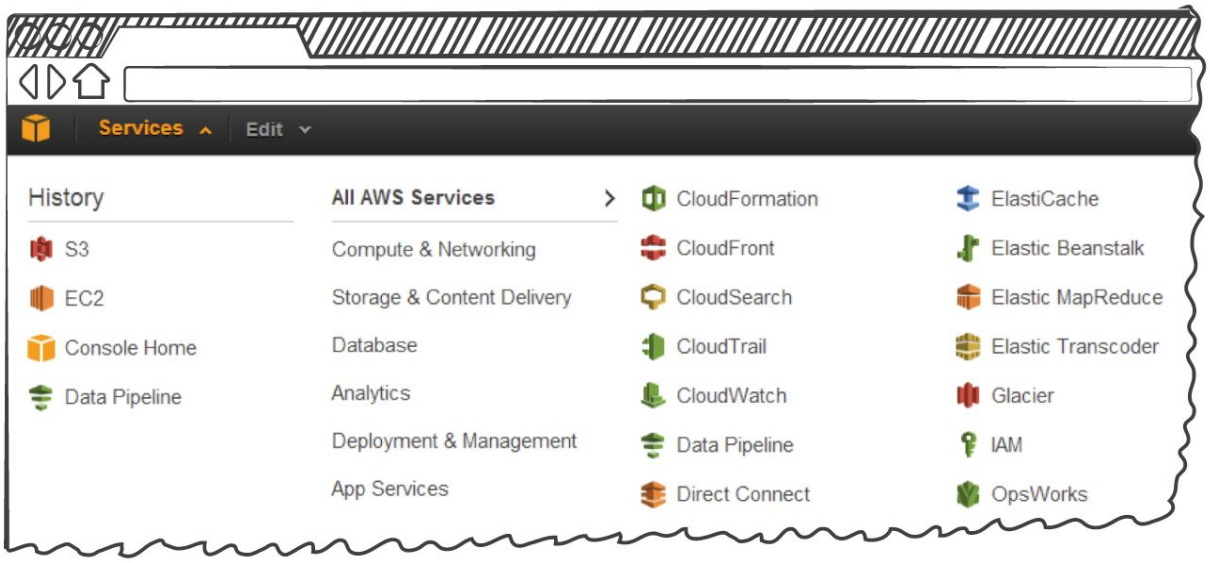
- Platform as a Service (PaaS) – All software and hardware required to build and operate cloud-based applications are provided by the PaaS provider via public Internet, VPN, or dedicated network connection. Users pay by use of the platform and control how applications are utilized throughout their lifecycle.

1.1 Infrastructure as a Service



Infrastructure as a Service, sometimes abbreviated as IaaS, contains the basic building blocks for cloud IT and typically provide access to networking features, computers (virtual or on dedicated hardware), and data storage space. Infrastructure as a Service provides you with the highest level of flexibility and management control over your IT resources and is most similar to existing IT resources that many IT departments and developers are familiar with today.

1.2 Cloud Providers and Services

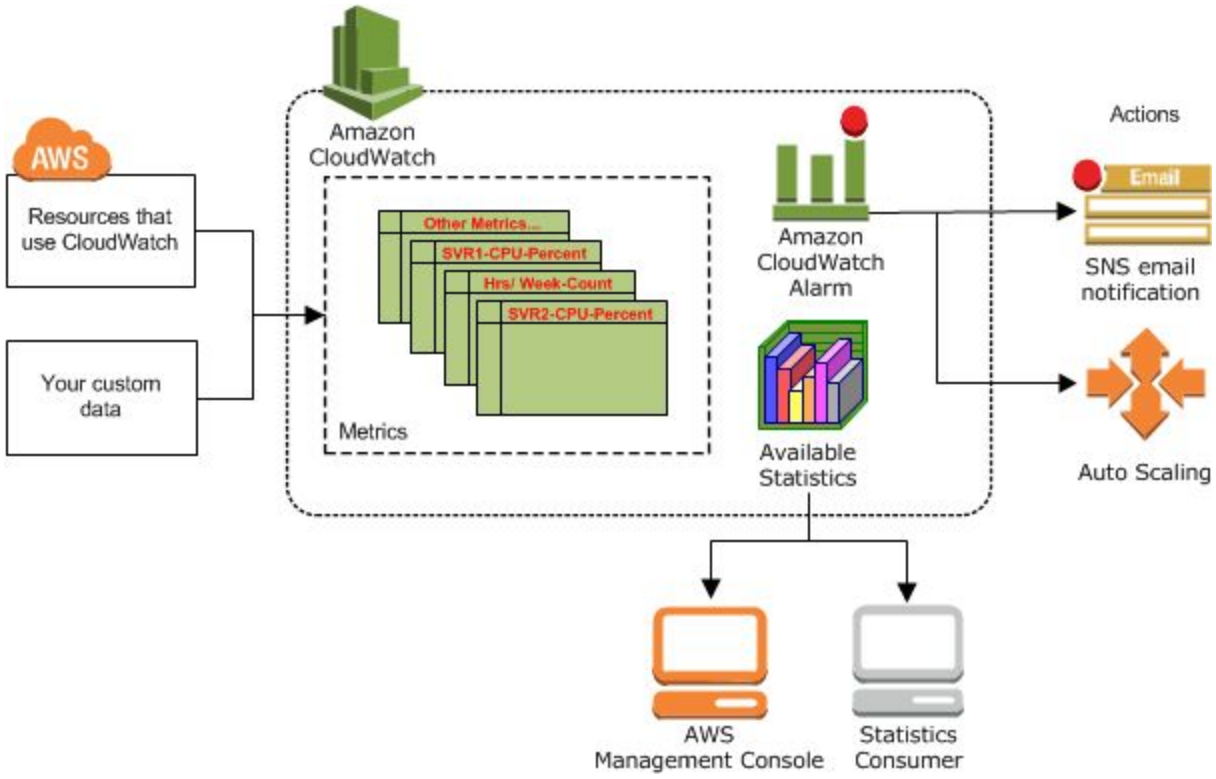


Apart from providing software, platform and infrastructural services cloud providers also provide metrics to monitor them and set alarms and triggers to scale them depending on application and load requirements. The exact implementation of such services and API's depend on the service provider.

1.2.1 AWS Cloud Watch API

Amazon CloudWatch is a monitoring service for AWS cloud resources and the applications running on AWS. CloudWatch collects and tracks metrics, collects and monitors log files, and set alarms. It can monitor AWS resources such as Amazon EC2 instances, Amazon DynamoDB tables, and Amazon RDS DB instances, as well as custom metrics generated by applications and services, and any log files that the applications generate. CloudWatch can be used to gain system-wide visibility into resource utilization, application performance, and operational health. Such insights can be used to react and keep application running smoothly and optimize the resource being utilized by application which benefits both the infrastructure provider and application owner.

Architecture



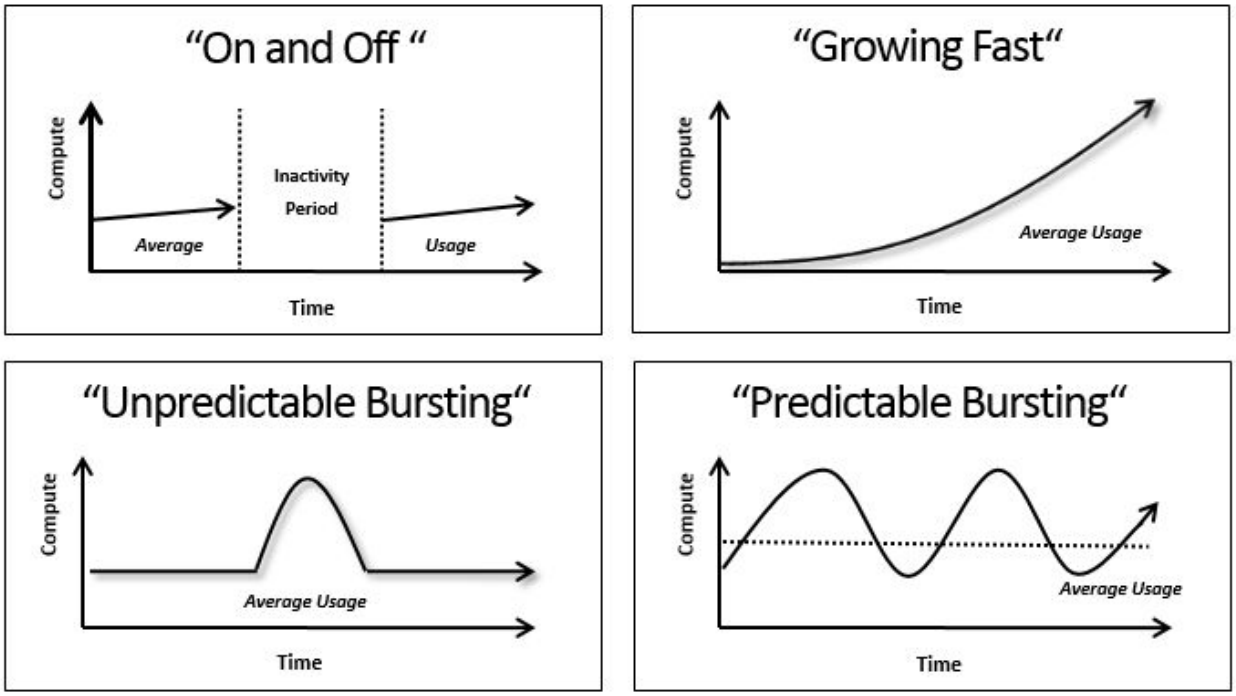
1.3 Cloud Workload Patterns

Not all workloads are the same, and not all Clouds are the same!
Different applications have different set of requirements and characteristics. Some Clouds (i.e. GAE or Heroku) are natural fits for certain class of workloads (i.e. WebApps) whereas for other types of workloads (i.e. batch), other Cloud services (i.e. AWS) are more appropriate. In some cases, the business operation and/or legal requirements may require a completely different deployment (i.e. private Cloud).

There are a set of general characteristics or attributes to consider when analyzing the applications:

- Workload Type: In general, there are two types of workloads: Batch or Online. It is important to consider this differentiation for the following reasons:
    - There are different resource requirements and considerations. As an example, batch workloads may require specific capacity in terms of storage and compute resources (i.e. vCPU, memory) to finish the job in a timely fashion whereas for online workloads network bandwidth may be more critical…
    - There are differences in programming models. As an example, some batch jobs may be implemented over a framework like Hadoop whereas for some online workloads a PaaS like Force.com may be the best choice.

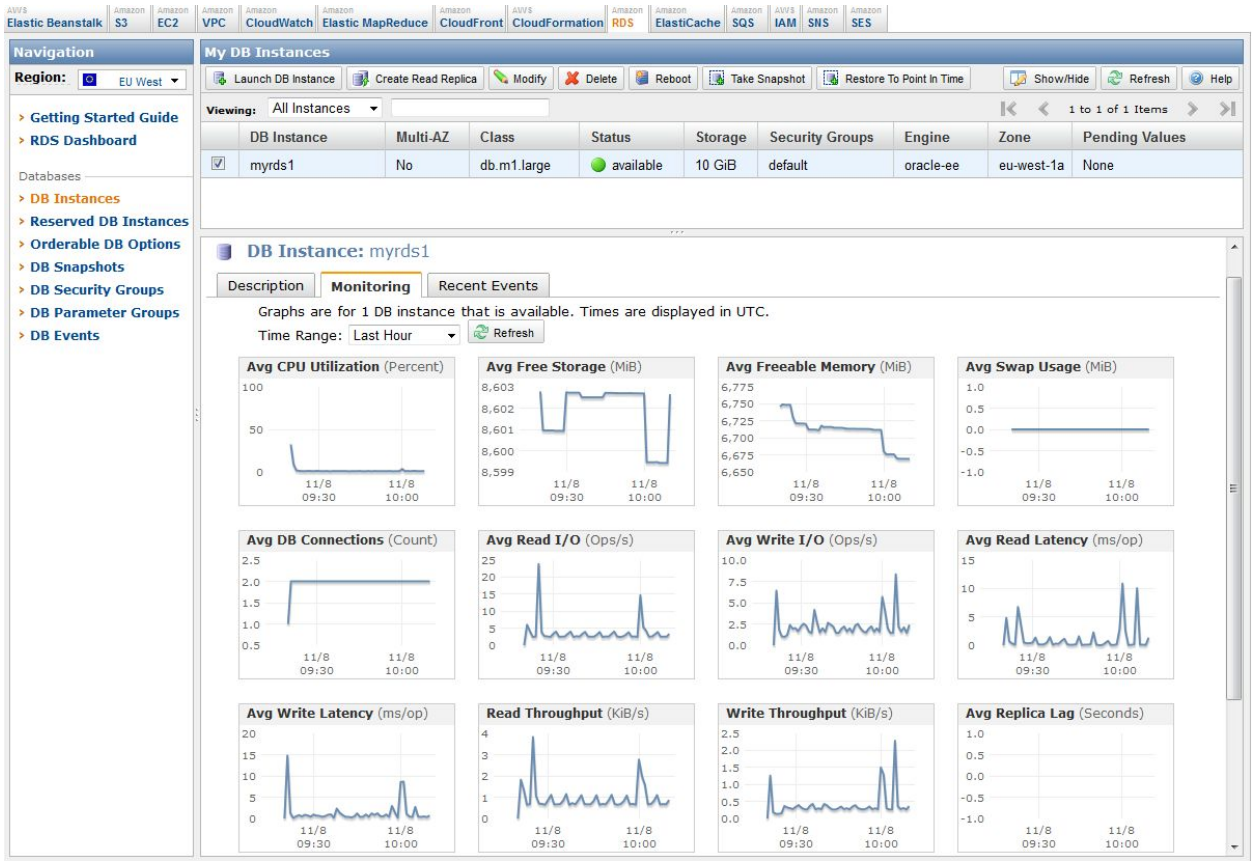
- Workload Frequency:



- Workload Cost: It is important to capture the total cost of workload including hardware, software, application maintenance and support, etc. It would be even more useful to develop a cost allocation model reflecting percentages in infrastructure, software, application development, support and maintenance. This information is useful in Cloud service selection.

1.4 Parameters and Measurements

Most infrastructure providers have API's to fetch information, monitor infrastructure and trigger alarms to scale infrastructure up/down or perform relevant actions on load balancer. Depending on current state on infrastructure and predicted state from various machine learning models or simple algorithms the infrastructure needs are adjusted which minimize the resources being over utilized and help bring down the overall cost of deployed infrastructure.



Chapter 2

Infrastructure Scaling

Infrastructure scalability in general refers to the actions taken by infrastructure providers to adjust to need of deployed application, it might result in increase or decrease of resources, in most cases cpu's, memory, network. This can actually be done in two ways, either to keep adding resources to existing virtual machines and build more powerful machines or other way is to add more virtual machines to autoscaling group itself.
Most of them treat individual virtual servers as immutable. There are some complex resizing rules, and local storage cannot be resized at any time. Resizing a VM image also results in all new public and private IP addresses. They really build a new server when resizing.

While adding CPU/memory capacity to a running virtual machine or trying to resize to an instance type of a different virtualization type – it must first be stopped. Many times instances between different virtualization types cannot be resized, so the user has to carefully plan for this. Also most of the times stopping a VM means that anything on the ephemeral storage is destroyed. Infrastructure providers also strongly encourage customers to build horizontally-scalable apps, and their rich Auto Scaling services supports that. It works by adding (or removing) virtual resources from a pool based on policies.

2.1 Vertical Scaling

Vertical scaling can essentially resize your server with no change to your code. It is the ability to increase the capacity of existing hardware or software by adding resources. Vertical scaling is limited by the fact that you can only get as big as the size of the server.

Pros:
>Simpler to code against
> Less machines to maintain

Cons:
> Downtime needed for upgrades
> Capacity has to be increased in large 'chunks'

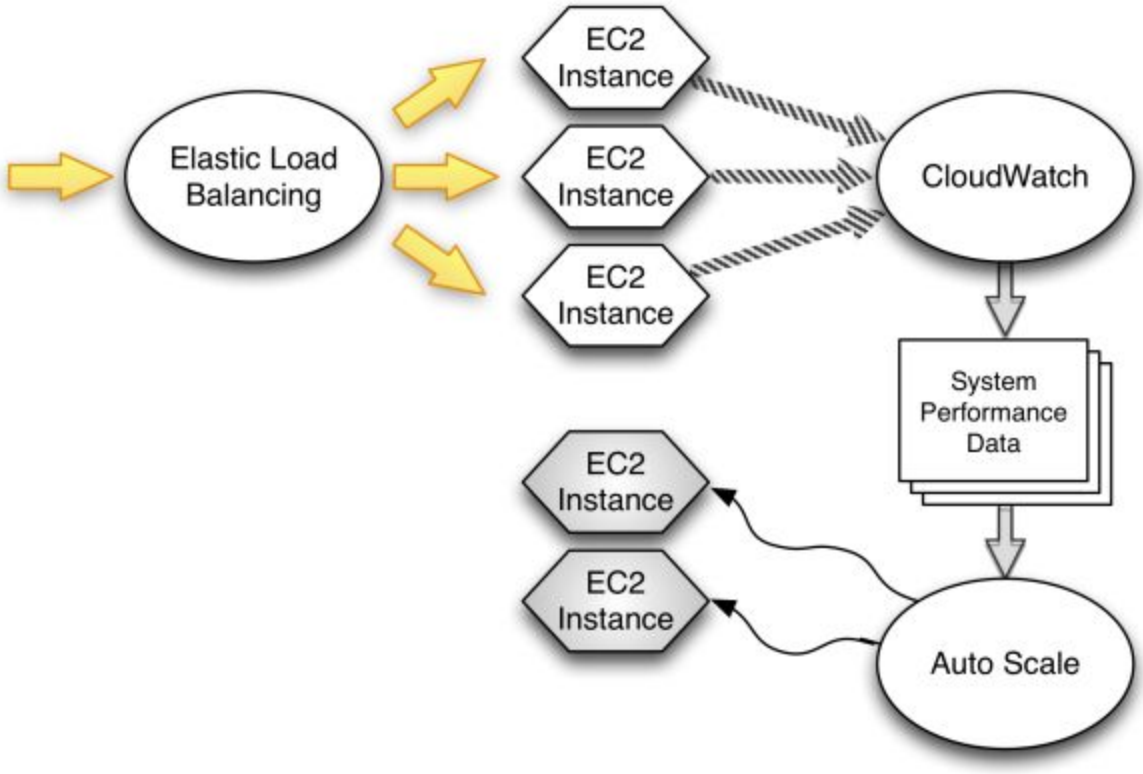
2.2 Horizontal Scaling

Horizontal scaling affords the ability to scale wider to deal with traffic. It is the ability to connect multiple hardware or software entities, such as servers, so that they work as a single logical unit. This kind of scale cannot be implemented at a moment's notice.

Pros:
> Commodity hardware
> Smooth expansion as needed

Cons:
> Code complexity increases
> Individual response performance can be limited

Example Architecture for auto scaling on AWS infrastructure:



2.3 Predicting Load Bursts

Much of the workload patterns show that the change in state of infrastructure can be predicted based on application behaviour and the nature of application. Databases show different kind of workload patterns as compared to web applications, which are themselves different from batch processing distributed systems.

2.3.1 Machine Learning Models

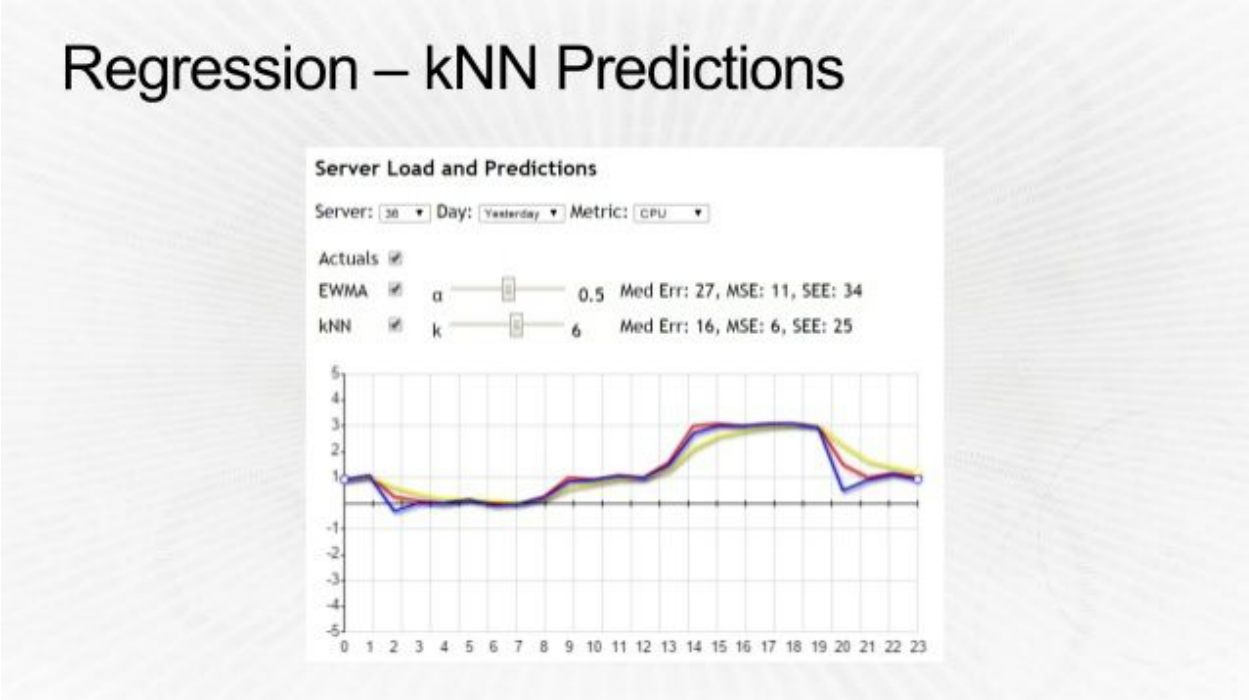
Sometimes the application behaviour is not predictable and there are no standard workload patterns that can predict the possible state in which the application will be after a particular time. In such cases it's a necessary requirement to train the auto-scaling models using machine learning techniques which can predict the state of application. Many techniques have been proposed for this purpose, I'll discuss one here that we experimented on:

2.3.2 Regression kNN Predictions

Starting with AWS Cloud Watch API data following steps need to be taken:

> Time Series Smoothing on data from Cloud Watch API
> EWMA (Exponential Weighted Moving Average)
> k-NN approach for Classification
> Prediction Performance using:
> Mean Squared Error, Median Error and Standard Error

Results



Red - Actual Load
Yellow - EWMA Trend
Blue - k-NN Predictions

Chapter 3

Experiments on Xen

Xen Project is a hypervisor using a microkernel design, providing services that allow multiple computer operating systems to execute on the same computer hardware concurrently.

3.1 Hypervisors

A hypervisor, also called a virtual machine manager, is a program that allows multiple operating systems to share a single hardware host. Each operating system appears to have the host's processor, memory, and other resources all to itself. However, the hypervisor is actually controlling the host processor and resources, allocating what is needed to each operating system in turn and making sure that the guest operating systems (called virtual machines) cannot disrupt each other.
Hypervisors use a thin layer of code in software or firmware to allocate resources in real-time. Hypervisor can be thought of as the traffic cop that controls I/O and memory management.

There are two types of hypervisors: Type 1 and Type 2.

Type 1:
Hypervisors run directly on the system hardware. They are often referred to as a "native" or "bare metal" or "embedded" hypervisors in vendor literature.

Type 2:
Hypervisors run on a host operating system. When the virtualization movement first began to take off, Type 2 hypervisors were most popular. Administrators could buy the software and install it on a server they already had.

Type 1 hypervisors are gaining popularity because building the hypervisor into the firmware is proving to be more efficient. According to IBM, Type 1 hypervisors provide higher performance, availability, and security than Type 2 hypervisors. (IBM recommends that Type 2 hypervisors be used mainly on client systems where efficiency is less critical or on systems where support for a broad range of I/O devices is important and can be provided by the host operating system.) Experts predict that shipping hypervisors on bare metal will impact how organizations purchase servers in the future. Instead of selecting an OS, they will simply have to order a server with an embedded hypervisor and run whatever OS they want.

3.2 Managing Xen Resources

Xen contains a number of features that help management of system resources. Functionality in this group covers CPU pools, pinning, NUMA and Schedulers and is often related to performance tuning.

3.3 Xen Scheduler

The Xen Project Hypervisor supports several different schedulers with different properties. Different schedulers can be assigned to

- an entire host
- a pool of physical CPU's on a host (VMs need to be assigned to a pool or pinned to a CPU)

Scheduler parameters can be modified per

- an entire host
- a CPU pool
- A Virtual Machine

3.3.1 Credit Scheduling

The credit scheduler is a proportional fair share CPU scheduler built from the ground up to be work conserving on SMP hosts.

Each domain (including Host OS) is assigned a weight and a cap.

Weight

A domain with a weight of 512 will get twice as much CPU as a domain with a weight of 256 on a contended host. Legal weights range from 1 to 65535 and the default is 256.

Cap

The cap optionally fixes the maximum amount of CPU a domain will be able to consume, even if the host system has idle CPU cycles. The cap is expressed in percentage of one physical CPU: 100 is 1 physical CPU, 50 is half a CPU, 400 is 4 CPUs, etc... The default, 0, means there is no upper cap.
NB: Many systems have features that will scale down the computing power of a CPU that is not 100% utilized. This can be in the operating system, but can also sometimes be below the operating system, in the BIOS. If you set a cap such that individual cores are running at less than 100%, this may have an impact on the performance of the workload over and above the impact of the cap. For example, if the processor runs at 2GHz, and you cap a vm at 50%, the power management system may also reduce the clock speed to 1GHz; the effect will be that the VM gets 25% of the available power (50% of 1GHz) rather than 50% (50% of 2GHz).

SMP load balancing

The credit scheduler automatically load balances guest VCPUs across all available physical CPUs on an SMP host. The administrator does not need to manually pin VCPUs to load balance the system. However, he can restrict which CPUs a particular VCPU may run on using the generic vcpu-pin interface.

Algorithm

Each CPU manages a local run queue of runnable VCPUs. This queue is sorted by VCPU priority. A VCPU's priority can be one of two value: over or under representing whether this VCPU has or hasn't yet exceeded its fair share of CPU resource in the ongoing accounting period. When inserting a VCPU onto a run queue, it is put after all other VCPUs of equal priority to it.
As a VCPU runs, it consumes credits. Every so often, a system-wide accounting thread recomputes how many credits each active VM has earned and bumps the credits. Negative credits imply a priority of over. Until a VCPU consumes its allotted credits, it priority is under.
On each CPU, at every scheduling decision (when a VCPU blocks, yields, completes its time slice, or is awaken), the next VCPU to run is picked off the head of the run queue. The scheduling decision is the common path of the scheduler and is therefore designed to be lightweight and efficient. No accounting takes place in this code path.
When a CPU doesn't find a VCPU of priority under on its local run queue, it will look on other CPUs for one. This load balancing guarantees each VM receives its fair share of CPU resources system-wide. Before a CPU goes idle, it will look on other CPUs to find any runnable VCPU. This guarantees that no CPU idles when there is runnable work in the system.

3.4 Hot Plugging Xen DomU

One of the huge improvements found in virtual vs. physical infrastructure is the ease with which additional computing resources can be added to virtual machines. In the physical world, adding resources such as RAM and processing power required a hardware order, downtime planning, installation and then hope that everything would continue to operate as it did before the upgrade, except with additional resources.

With the ability for virtual machines to hot add additional RAM and processors, the addition of these resources can be accomplished with exactly zero downtime, as long as appropriate steps are taken and guest machine is running an operating system that supports this feature.

3.4.1 CPU Plugging

Xen supports vcpu hot add and remove, which allows to add and remove CPUs from a running system without downtime.

How it works

Inside domU config, setting “maxvcpus” to the maximum number of VCPUs that domU will be allowed to have. If not defined this way, it defaults to the value of “vcpus”, so it’ll always be able to hot remove, but wouldn’t be able to hot-add anything more than what it was booted with.

```
#vcpus - number of VCPUs to boot the system with.
vcpus = 2;


#maxvcpus - maximum number of VCPUs (total) that can be hot added later.
maxvcpus = 8;
```

VCPU Hot Add Example

If we have a virtual machine named foo which is given 1 VCPU. One day we notice that the system is struggling to keep up with a CPU heavy load. So, we want to add another VCPU to the VM, but we can’t afford a downtime. No problem (as long as we configured the maxvcpus value above).
Here’s the system with one 1 VCPU:

```
# xm list
Name                        ID Mem(MiB) VCPUs State   Time(s)
foo                         11  4096    1    -b----    23.7
```

To resize we use the ‘xm vcpu-set’ command. For example, to re-size our 1 VCPU domain to 2 VCPUs, execute the following.

```
# xm vcpu-set foo 2

# xm list
Name                                ID Mem(MiB) VCPUs State   Time(s)
foo                                 11  4096     2    -b----    31.6
```

VCPU Hot Remove Example

Similarly, we can hot remove VCPUs from a domain using the ‘xm vcpu-set’ command. Here’s the system with one 2 VCPUs:

```
# xm list
Name                                ID Mem(MiB) VCPUs State   Time(s)
foo                                 11  4096     2    -b----    52.5
```

To hot remove VPCUs simply execute ‘xm vcpu-set’, specifying a lower number than what is currently assigned.

```
# xm vcpu-set foo 1

# xm list
Name                                ID Mem(MiB) VCPUs State   Time(s)
comlag                               11  4096     1    -b----    56.7
```

And that’s it. As long as we’ve taken the time to set your ‘maxvcpus’ setting in the domU config before booting the machine we’ll be able to adjust our VCPU assignments as load varies.

3.4.2 Memory Plugging

Xen 3+ supports memory “ballooning” which allows you to hot add and remove memory from a running system. It is a nice feature and has come in handy for me on many occasions.

Memory Hot Remove Example

Let’s say we have a virtual machine named foo which I’ve given 1024 megs of ram. One day we notice that the system is using only 25% of this memory. We want to resize it to 512 megs and free the rest of the memory for use by other domains but we can’t afford a downtime. No problem, Xen lets you resize memory on-the-fly. Before we adjust the memory allocations let’s see what the system looks like

```
# xm list
Name                        ID Mem(MiB) VCPUs State   Time(s)
Domain-0                     0     256    2 r-----  49678.8
foo                          2    1024    1 -b----   2160.2
```

To resize we use the ‘xm mem-set’ command. For example, to re-size our 1024M domain to 512M execute the following.

```
# xm mem-set foo 512
# xm list
Name                        ID Mem(MiB) VCPUs State   Time(s)
Domain-0                     0     256    2 r-----  49681.0
foo                          2     512    1 -b----   2160.3
```

Memory Hot Add Example

We can also use the xm-mem set command to hot-add memory to a running domain, there is one catch, however. Xen can only add memory up to the amount specified by the ‘maxmem’ configuration directive in your domU config file. If no ‘maxmem’ option is specified then the maximum memory value is set to the size of the ‘memory’ option.

This is an example config file for a domain named ‘foo’ that will boot with 1024M RAM and can grow up to 4096M.

```
#/etc/xen/configs/foo

name   = "foo"
memory = 1024
maxmem = 4096
vcpus  = 1
```

When we boot this domain we’ll see that its initial memory size is 1024M.

```
# xm create /etc/xen/configs/foo
Using config file "/etc/xen/configs/foo".
Started domain foo

# xm list
Name                                ID Mem(MiB) VCPUs State   Time(s)
Domain-0                             0      256     2 r-----  49723.1
foo                                 11     1024     1 r-----      5.1
```

Now we can mem-set the domain up to the value of maxmem, in our case 4096M.

```
xm mem-set foo 4096
```

```
# xm list
Name                                        ID Mem(MiB) VCPUs State   Time(s)
Domain-0                                     0      256     2 r-----  49725.6
foo                                         11     4096     1 -b----      8.8
```

Chapter 4

Integrations and Future Scope

The above experiments and study can be used to explore vertical scaling and hot plugging on more hypervisors and operating systems. Data from some such experiments has been covered in this report.

4.1 Exploring Hypervisors

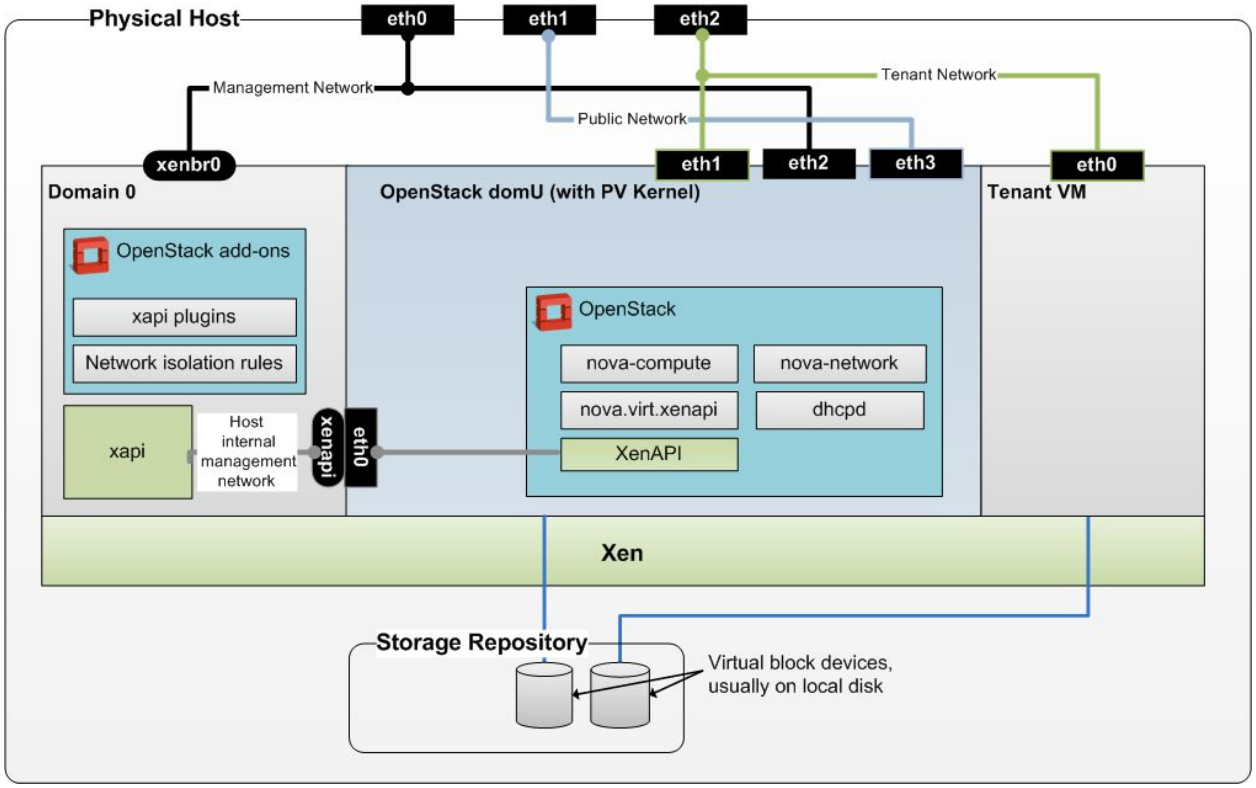
		KVM	VMware	Xen	Hyper-V
CPU core	Add/ Remove	Yes[30]/ No[30]	Yes[65]/ No[65]	Yes[26]/ Yes[26]	No[34]/ No[34]
Memory	Add/ Remove	Yes[31,33]/ Yes[31,33]	Yes[65]/ No[65]	Yes[25]/ Yes[25]	Yes[42]/ No[42]
Disk	Add/ Remove	Partly[32]/ ?	Partly[4]/ Yes[4]	Yes[27]/ Yes[27]	Partly[54]/ Partly[54]
	Extend/ Shrink	No[56]/ No[56]	Yes[65]/ Partly[65]	Partly[40]/ No[40]	Partly[43]/ Partly[43]
Network	Add/ Remove	Yes[32]/ ?	Yes[65]/ Yes[63]	Yes[1]/ Yes[1]	No[34]/ No[54]

4.2 Supporting Operating Systems

		CentOS	SUSE	Debian	Ubuntu	Windows Server
CPU core	Add/ Remove	Yes[52]/ Yes[52]	No[59]/ No[59][12]	Yes[3]/ Yes[3]	Yes[3]/ Yes[3]	Yes[64]/ ?
Memory	Add/ Remove	Yes[51]/ Yes[51]	Yes[59]/ Yes[59]	Yes[64]/ Yes[2]	Yes[64]/ Yes[2]	Yes[64]/ ?
Disk	Add/ Remove	Yes[50]/ Yes[50]	Yes[59]/ Yes[59]	Yes[6]/ Yes[20]	Yes[7]/ Yes[7]	Partly[45]/ Yes[45]
	Extend/ Shrink	Partly[53]/ Partly[53]	Partly[60]/ Partly[60]	Partly[5]/ Partly[5]	Yes[15]/ Partly[15]	Yes[43]/ Partly[43]
Network	Add/ Remove	Yes[50]/ Yes[50]	Yes[59]/ Yes[59]	Yes[9]/ Yes[9]	Yes[58]/ Yes[14]	Yes[44]/ Yes[44]

4.3 Integration with OpenStack

Such an auto scalable system can be tested on Open Source Infrastructure Management Tools like OpenStack to adjust to needing demands of application and virtual machines.


Physical Host
eth0
eth1
eth2
Management Network
Tenant Network
Public Network
xenbr0
eth1
eth2
eth3
eth0
Domain 0
OpenStack domU (with PV Kernel)
Tenant VM
OpenStack add-ons
xapi plugins
Network isolation rules
OpenStack
nova-compute
nova-network
nova.virt.xenapi
dhcpd
XenAPI
xapi
Host internal management network
xenapi
eth0
Xen
Storage Repository
Virtual block devices, usually on local disk

Conclusions

The experiments and observations in this report along with existing research show that it's possible to create a hybrid model combining both horizontal and vertical scaling which can meet the demands of application based on policies and machine learning models which collaboratively predict workload patterns. The resources provided to application can optimizely be utilized, it decreases overestimations and latency in scaling the entire infrastructure.

We prove that if proper workload prediction algorithms are applied it's possible to scale up entire infrastructure by adding resources to existing pools of auto scaling groups, but we discovered that along with many of its advantages it has many downsides one of which mainly is that each virtual machine has an overhead cost of operating system and in many cases it needs careful implementation of application so that it can be horizontally scalable.

Much of the workload patterns show that the change in state of infrastructure can be predicted based on application behaviour and the nature of application. Databases show different kind of workload patterns as compared to web applications, which are themselves different from batch processing distributed systems.

We explore a purely vertically scalable model in which resources can be added on the go. We try to add vcpu's and memory to a live virtual machine so that it can be scaled up without any down time. The hot plugging hypervisor in this case is a Xen hypervisor. The deployed virtual machines can fetch resources from host machine on demand. But, the additional operations cost of powering and cooling and the large footprint they will occupy in the data center makes pure vertical scaling costly.
Most of the infrastructure providers treat individual virtual servers as immutable. There are some complex resizing rules, and local storage cannot be resized at any time. Resizing a VM image also results in all new public and private IP addresses. They really build a new server when resizing.

We experiment with a set of hypervisors and operating systems on which this kind of a hybrid model can be set up and finally discuss how the complete system can be deployed in integration with open source cloud management software like OpenStack. The experiments performed target limited infrastructure providers and infrastructure management tools but in general it can be extended to a vide range of similar cloud resource providers and leverage the power of cloud computing optimally.

References

[1]http://www.rackspace.com/knowledge_center/whitepaper/understanding-the-cloud-computing-stack-saas-paas-iaas

[2] http://soa-biz.blogspot.in/2009/11/workload-analysis-in-cloud-computing.html

[3] https://seroter.wordpress.com/2014/11/20/comparing-clouds-iaas-scalability-options/

[4] http://www.thoughtsoncloud.com/2014/04/explain-vertical-horizontal-scaling-cloud/

[5] http://www.cs.cornell.edu/~zshen/papers/11726-icac13_nguyen.pdf

[6] http://www.slideshare.net/harishganesan/auto-scaling-using-amazon-web-services-aws

[7] http://www.cardinalpath.com/autoscaling-your-website-with-amazon-web-services-part-2/

[8] http://blog.celingest.com/en/2013/12/19/auto-scaling-with-the-aws-management-console/

[9] http://resources.intenseschool.com/amazon-aws-understanding-auto-scaling-part-i/

[10] https://aws.amazon.com/cloudwatch/

[11]http://www.slideshare.net/AmazonWebServices/pfc307-auto-scaling-a-machine-learning-approach-aws-reinvent-2014

[12] http://backdrift.org/how-to-hot-addremove-vcpus-from-a-xen-domain

[13] http://backdrift.org/xen-memory-hot-add-and-remove

[14] http://wiki.xen.org/wiki/Credit_Scheduler

[15] https://wiki.openstack.org/wiki/XenServer/XenAndXenServer