

**Cork Institute of Technology**

Department of Computing

**Virtual machine provisioning in OpenStack and its potential  
for improvement**

**By**

**John V Brennan**

**Student ID: R00104987**

**Research Project – MSc. in Cloud Computing**

**Supervisor: Dr. John Creagh**

**Submitted: March 2015**

**Declaration**

This report is submitted in partial fulfilment of the requirements for the Degree of Master of Science in Cloud Computing. It represents substantially my own work except where explicitly indicated in the text.

## Abstract

This paper explores the area of resource management in an Infrastructure as a Service (IaaS) solution. We examine the core components of a resource management system and their importance in the successful operation of a cloud environment. We look at management objectives defined by the cloud service provider that dictate how network, compute and storage resources should be managed so that they meet performance and efficiency requirements. The resource management component of OpenStack, a real world open-source IaaS solution is explored. We discover how a virtual machine is scheduled through detailed inspection of the OpenStack source code. We look at the role that the scheduling component plays within OpenStack in the placement of new virtual machine instances. We also examine third party alternatives to the built-in scheduling algorithms and how they fill gaps in the built-in offering. Finally, we look at the process of creating and deploying a custom scheduler in an OpenStack cloud environment. This allows the scheduler behaviour in OpenStack to be tailored to suit a specific business need.

## Acknowledgements

The researcher would like to thank Dr. John Creagh at Cork Institute of Technology for his time, insight and valuable feedback over the course of this research project. A sincere thank you to Colm Gillard at Version1 for his assistance over the past two years and without whose support I would not have reached this stage. Finally, a sincere thank you to my wife Brigitte for her continuous support throughout this project.

## Contents

Abstract.....	i
Acknowledgements.....	ii
1 Introduction .....	1
2 Cloud Services .....	2
2.1 Cloud Service Characteristics .....	2
2.2 Deployment Models.....	3
2.3 Cloud Types .....	4
3 OpenStack - Open Source Cloud .....	5
3.1 The OpenStack Project .....	5
3.1.1 The OpenStack Foundation.....	6
3.2 OpenStack Components.....	7
3.2.1 Identity Service [Keystone] .....	8
3.2.2 Compute Service [Nova] .....	8
3.2.3 Image Service [Glance] .....	8
3.2.4 Networking Service [Neutron] .....	8
3.2.5 Block Storage Service [Cinder] .....	8
3.2.6 Object Storage [Swift] .....	9
3.2.7 Dashboard [Horizon].....	9
3.3 Sentiment .....	10
4 Virtual Machine Scheduling .....	12
4.1 Management Objectives .....	12
4.2 Virtual Machine Placement.....	12
4.2.1 The Bin Packing Problem .....	13
4.2.2 Placement Strategies .....	14
5 Virtual Machine Scheduling in OpenStack.....	17
5.1.1 Controller Node .....	17
5.1.2 Compute Node.....	17
5.1.3 Storage and Network Nodes.....	17
5.2 Nova Compute Service .....	17
5.3 OpenStack Scheduling Algorithms .....	18
5.3.1 Filter and Weight Scheduler .....	18

5.3.2	Chance Scheduler.....	22
6	Code Walkthrough: Provisioning a new Virtual Machine in OpenStack .....	23
6.1	Virtual Machine Request Attributes .....	23
6.2	Processing New Virtual Machine Request .....	24
6.2.1	Compute API create() method .....	25
6.2.2	_create_instance() .....	25
6.2.3	build_instances() .....	28
7	Third Party OpenStack Schedulers.....	31
7.1	FairShare Scheduler .....	31
7.2	IBM – Platform Resource Scheduler .....	32
7.2.1	The Goldilocks Dilemma .....	32
7.2.2	Policy based Scheduling.....	33
8	Building a Custom OpenStack Scheduler .....	35
8.1	Scheduler Base Classes.....	35
8.2	Tenant Scheduler .....	36
8.2.1	Scheduler Algorithm .....	36
8.2.2	Tenant Scheduler Configuration .....	36
8.2.3	Source Code .....	37
8.3	Deployment.....	38
8.3.1	Location.....	39
8.3.2	Service Configuration.....	39
8.3.3	Restart Nova Scheduler Service .....	39
8.3.4	Testing.....	40
9	Summary and Conclusions.....	41
10	Cited References .....	43
	Appendix 1: TenantScheduler Source Code .....	47
	Appendix 2: Issues Encountered during this Project .....	48
	Installing OpenStack on Multiple Nodes.....	48
	Deployment of Custom Scheduler .....	49

# 1 Introduction

In recent years there has been a significant uptake in cloud computing with most companies now using cloud services in some shape or form as part of their IT strategy. Cloud services are changing the way business use IT to achieve their strategic objectives by enabling them to move from a CAPEX (Capital Expenditure) to an OPEX (Operational Expenditure) model. In the 2015 State of the Cloud Report, 93 percent of organisations surveyed were running applications or experimenting with infrastructure as a service. [1] As the market matures, many organisations are adopting a hybrid approach that uses a mixture of public and private cloud services. This hybrid approach is driven by issues such as compliance and concerns over performance and data security.

Operating a private cloud infrastructure presents challenges and responsibilities for an organisation, in particular the task of efficiently managing network, compute and storage resources. This research project will look at the concepts of resource management in general and examine a concrete implementation of resource management in a real-world cloud platform. A major component of resource management is the system that governs the scheduling and placement of new virtual machine requests on to physical hosts within the datacenter. This project will examine the scheduling of virtual machines and the role that scheduling plays in achieving the management objectives of the cloud service provider. These management objectives must balance the task of keeping operating costs low while at the same time adhering to specific service-level agreements (SLA).

The project will examine the scheduling component within OpenStack, a popular Infrastructure as a Service (IaaS) solution used by many organisations implementing a hybrid cloud strategy. We will examine its functionality, its shortcomings and its potential for improvement. OpenStack is an open-source project that supports different scheduling strategies “out of the box”. We will examine the different scheduling algorithms provided by OpenStack. The open-source nature of OpenStack provides an excellent opportunity to explore the inner workings of how virtual machines are provisioned in a real world cloud environment. As part of this research a code walkthrough of the OpenStack source code will be performed. This will trace and document in detail the sequence of events that occur from the point at which the end user submits a new virtual machine request through to its eventual fulfilment.

Alternatives to the built in OpenStack scheduling algorithms will also be examined to determine what approach they have taken to the process of virtual machine scheduling. Finally, a custom scheduling algorithm will be implemented, deployed and tested on a multi-node OpenStack cloud. This will demonstrate the process of altering the behaviour of the built-in scheduler in an OpenStack cloud to use our own custom logic.

## 2 Cloud Services

The National Institute of Standards in Technology defines Cloud Computing as: “a model for enabling ubiquitous, 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.”[2]

Today, Cloud Computing represents an evolution in the way IT resources are managed, delivered and consumed. As our world has become more connected, the demands for network, compute and storage resources have grown rapidly. Accurately forecasting demand for IT resources is difficult. Recent trends such as Big Data and the “consumerisation of IT” [3] have forced the modern datacenter to become more agile and responsive to massive fluctuations in demand.

### 2.1 Cloud Service Characteristics

The latest generation of datacenters are using new technologies and processes to deliver highly responsive, adaptive services known as cloud services. These services typically exhibit the following characteristics [4]:

- **Elasticity:** services scale up and down based on user demand. For the end user, new services can be quickly spawned giving the illusion of unlimited network, storage and computing resources. Such rapid provisioning is made possible through the use of technologies like virtualization which allows for resources to be provisioned in a short space of time using programmable APIs.
- **Availability:** cloud services must be highly available and highly reliable. For standard cloud services this is generally 99.95%. For mission critical applications it is normal to see 99.999% (five 9's) availability which corresponds to 6.05 seconds of downtime per year. Cloud service providers will specify the guaranteed levels of availability in the Service Level Agreement (SLA) and are liable to pay financial penalties to the customer if the SLA is violated.
- **Pay per use:** cloud services typically follow a different business model to the traditional shrink wrapped software licensing model. Customers generally don't pay a big cost upfront for cloud services. Instead, cloud service are metered and customers are charged based on measured usage. This makes it particularly appealing to businesses as it allows them to eliminate large capital IT expenditure (CAPEX) and replace this with lower cost operational expenditure (OPEX). The overhead of managing and paying for capital intensive hardware and software is eliminated as large amounts of computing resources can be spun up when needed and just as quickly be released when no longer required.
- **Self Service:** cloud based services are manageable via a web based interface or a programmable API. New services can be spawned without the need for manual human intervention.

- **Multi-tenancy:** resource utilisation is managed through the use of technologies like virtualization. Cloud consumers (tenants) share the same physical hardware but are logically separated from other tenants in the cloud environment.

Companies such as Microsoft, Amazon and Google have made big bets on the success of this technology paradigm, committing billions of dollars to build next generation datacenters capable of meeting the demands of a global market for cloud services. As of 2013, Google had spent \$21 billion dollars building out the infrastructure required to provide cloud computing services [5]. Likewise, Microsoft has invested over \$15 billion to date on its datacenter construction program [6]. The scale of the investment by these companies is a clear indication of the potential marketplace for cloud services.

Cloud technology provides tremendous opportunities for service providers and customers alike. For the service provider, it opens the door to a truly global audience for their product or service. This has been very evident in the mobile gaming market where the barrier to entry for small companies looking to take a foothold in the market have been lowered thanks to cloud computing. Previously, companies in this space faced large upfront costs to purchase hardware and software. With cloud based services, the barrier has been greatly reduced. Companies can now buy the cloud based resources needed to get started and if the project is successful they can rapidly scale up the infrastructure to handle demand. This can be seen in the success of companies such as Rovio (producer of Angry Birds) and King Games (producer of Candy Crush Saga).

For the customer, there are benefits too. They get the use of the cloud service without the financial outlay of purchasing new hardware, installing software and maintaining the underlying infrastructure. The significant burden that comes with managing and maintaining an IT environment for this service becomes the responsibility of the service provider.

## 2.2 Deployment Models

Cloud Service Providers offer different types of cloud services:

- **Infrastructure as a Service (IaaS):** is the most fundamental level of cloud service. Service providers own, manage and maintain dedicated data centers that provide customers with on-demand access to pools of storage, compute and network resources. Customers can normally provision these resources via a web based management portal.
- **Platform as a Service (PaaS):** these generally build on top of IaaS enabling customers to create applications that run on the cloud infrastructure. These applications depend on operating systems, database server technologies and specific programming environments to be present in order to function. Examples of PaaS include Microsoft Azure and Google App Engine.
- **Software as a Service (SaaS):** A cloud service provider can use IaaS and PaaS to build software applications that are typically provided on a pay per use basis. Popular SaaS applications include email (Gmail), scheduling (Google Calendar) and time tracking systems (Innotas). SaaS applications can be multi-tenant based. In this case, a single



version of the software is hosted centrally but is used by multiple users. The service provider must ensure that data from each user/organisation is safely segregated.

## 2.3 Cloud Types

There are several different types of cloud provider:

- **Public Cloud:** this is the most common deployment model whereby cloud services are made available to the general public. Examples of public cloud are Amazon EC2 and Microsoft Azure.
- **Private Cloud:** companies may have concerns about placing sensitive data in a public cloud where they do not have the same level of control over how that data is protected and accessed. However, these companies still want to take advantage of the benefits that cloud services can provide for their organisation such as rapid provisioning, greater hardware utilisation and so forth. To do so, companies must build and manage their own private cloud, i.e. a cloud service used exclusively by their organisation. There are some drawbacks to private cloud as unlike public cloud the organisation itself is responsible for funding, managing and maintaining the hardware and software used to operate it.
- **Hybrid Cloud:** for many organisations, it is necessary to use a combination of public and private cloud to meet their business needs. Hybrid cloud refers to the process of enabling interoperability between these different cloud environments. The ultimate goal with hybrid cloud is to make it easy to move workloads from one cloud to another. The proprietary nature of public cloud solutions and a lack of well-defined standards among competing cloud vendors currently makes this difficult to achieve.

### 3 OpenStack - Open Source Cloud

Building an end to end cloud service is challenging for an organisation as there are many complex features that must be considered. [7] The first requirement is to be able to efficiently manage large pools of compute, storage and network resources. Virtualisation technologies play an important role in this regard as they allow efficiencies to be realised through greater hardware utilisation as well as making it possible to rapidly provision new virtual machines. Management tools are also required to monitor the cloud environment, to ensure that problems are quickly identified and resolved and that the levels of service comply with the Service Level Agreement (SLA) agreed with customers. Automation of key tasks and processes is essential in order to keep operating costs down and to allow the environment to scale to handle a large volume of users and requests. Key among these tasks is the provisioning of new virtual machines and the on-boarding process for new users. End users require a user-friendly way to manage this cloud environment. This is normally implemented as a web based portal. Finally, we need to consider how to measure each user's resource usage so that they (or their department) can potentially be billed for the resources they have consumed. [8]

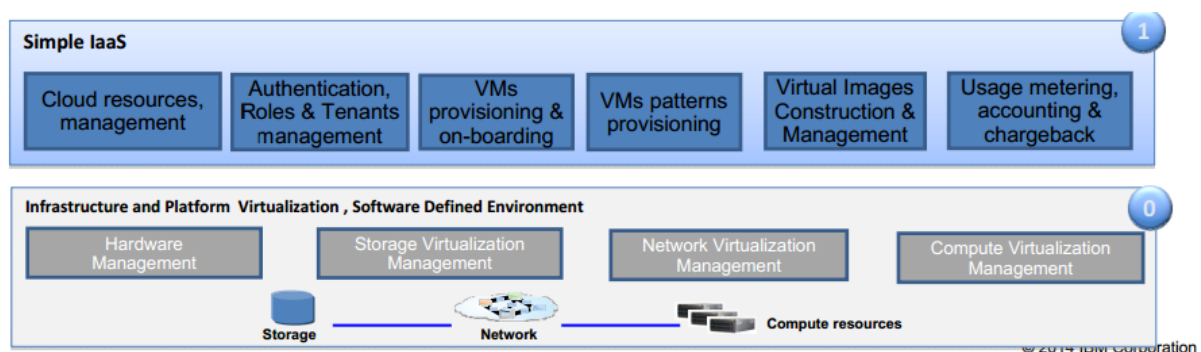


Figure 1 Functional areas for a simple IaaS implementation (source: IBM CCRA 4.0)

#### 3.1 The OpenStack Project

There are a number of open-source IaaS solutions available today, among the best known being Eucalyptus, CloudStack and OpenStack. Of these OpenStack has gained the greatest momentum in recent times with technology enthusiasts and industry leaders alike. OpenStack is an open-source platform that enables organisations of any size to create and manage a highly scalable, highly available Infrastructure as a Service (IaaS) service. The primary benefits of open-source include greater transparency, reduced risk of vendor lock-in and support from a vibrant community of developers and technology enthusiasts. The platform enables cloud service providers to manage the resources of a private cloud while enabling cloud consumers to provision resources through an easy to use management portal. [9] The first release of OpenStack appeared in 2010 as part of a joint collaboration between Rackspace and NASA. The project has since been open-sourced and is supported by an active community of 10,000+ individual members from across the globe. These members all have an interest in furthering OpenStack and freely contribute their time towards supporting, developing and documenting different aspects of the project.

### 3.1.1 The OpenStack Foundation

As of 2012 the strategic direction of the OpenStack project has been overseen by the OpenStack Foundation (OSF) whose mission is to “protect, empower and promote” OpenStack.[10] The OSF consists of two bodies; the Board of Directors and the Technical Committee. The Board of Directors is responsible for the strategic and legal oversight of the Foundation [11] and consists of a maximum of 24 members. It is important to note that there are different categories of member within the project, each with different levels of power and representation on the Board:

- **Platinum:** these are companies that have committed substantial financial and human resources towards the OpenStack Project. These companies have aligned their strategic objectives with the OpenStack Project. Currently, there are eight “Platinum Members” (AT&T, Canonical, HP, IBM, Intel, Rackspace, Red Hat & SUSE) with each being guaranteed a single seat on the Board. Most of these companies have significant experience managing large open-source projects as well as an extensive knowledge of the technology marketplace. [12]
- **Gold:** these are also companies that contribute financial and human resources to OpenStack but at a reduced level than that put forward by Platinum Members. There is a maximum of 24 companies allowed within the Gold Member level at one time. 8 seats are eligible to be filled on the Board of Directors from the overall pool of Gold Members.
- **Individual:** these are individuals who are interested in the OpenStack project but may not belong to a specific company. There can be an unlimited number of individual members and they are eligible to elect 8 members to the Board of Directors. Those elected have usually demonstrated strong commitment to OpenStack through source code contributions or through activities in the OpenStack community.

The governance structure of the OpenStack Foundation is such that no one member can seize control thus ensuring that decisions are taken collectively and in the best interests of all members. For a global open-source project of this scale, strong governance is essential to steer the project in the right direction and to deliver a product that meets the needs of end users. This governance structure brings credibility and experience to the project thus ensuring that decisions taken are in the best interests of the project rather than a select few.

The Technical Committee represents contributors to the project and oversees the technical goals and objectives for upcoming OpenStack releases. This committee oversees all of the various programs that make up a release and they have the final say on all matters technical. New releases are typically shipped every 6 months. The Technical Committee has 13 members, elected annually by the Active Technical Contributors, individuals who have contributed source code to the project over the last 2 release cycles.[13]

### 3.2 OpenStack Components

When thinking about the job that OpenStack must perform within the datacenter, it is helpful to think of OpenStack as being analogous to a traditional desktop operating system except that instead of managing network, storage and compute resources for a single computer it must manage pools of resources at a datacenter level. OpenStack is not one monolithic program but instead consists of a collection of services, each fulfilling a distinct role.

Architecturally, the project is made up of several core components, each of which follows a service oriented, shared nothing, message based approach [14]. As a distributed system, each component is designed with highly availability and fault tolerance in mind. Each component must be capable of scaling up to handle potentially very large workloads. Components are designed to be resilient and in the event of failure should not cause a ripple effect which could potentially take down other components.

Each component has a distinct purpose within the system. Communication between components is facilitated via Advanced Message Queuing Protocols (AMQP). Message queues enable reliable asynchronous processing which results in faster response times and helps eliminate situations where one process is waiting on another process before it can complete. Each component has a RESTful API endpoint that exposes specific functionality from within that component to the outside world. [15] Interaction among components in the OpenStack cloud all takes place using their individual APIs. For the purposes of this research we will examine only the Core components within OpenStack.

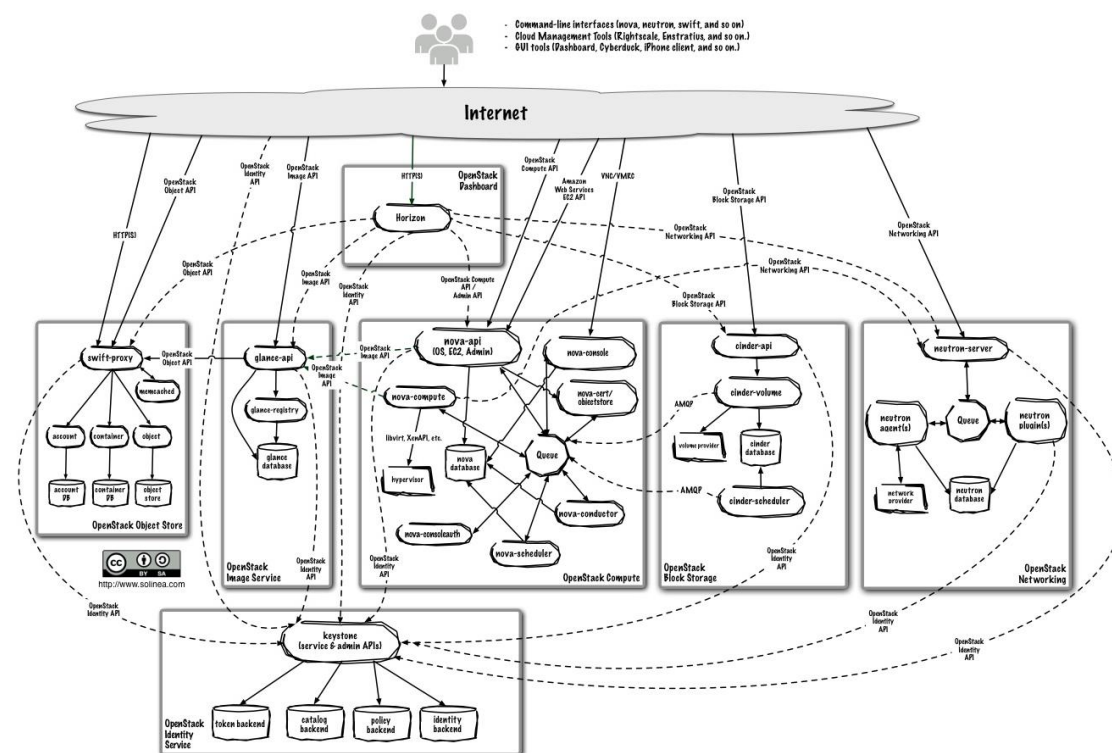


Figure 2 OpenStack Logical Architecture Overview Showing Interaction between Services (Source: OpenStack.org)

### 3.2.1 Identity Service [Keystone]

This service is an identity component that maintains a centralised store for user authentication, user authorisation and access control [16]. In addition, it maintains, a Service Catalogue that maintains a register of the API endpoints of all other services within the OpenStack cloud installation. The Service Catalogue acts as a discovery service that can be queried by any OpenStack service to discover what other services are available along with their contact details. The identity service runs on an Authentication Node.

### 3.2.2 Compute Service [Nova]

Nova is a cloud computing fabric controller that interfaces with hypervisors to run virtual machine instances. Nova is hypervisor agnostic and uses standard interfaces that allow it to communicate with many commonly available hypervisors. Out of the box Nova supports KVM and Libvirt which are two of the most popular Linux based hypervisors. Drivers are also available for other popular hypervisors such as Xen, VMware and Hyper-V.

### 3.2.3 Image Service [Glance]

Virtual machines are defined and configured using virtual machine images. The Image Services provides a central repository for storing and managing virtual machine image metadata [17]. Such metadata includes details such of the physical location where the image is stored as well as details of when the image was created among other things. In OpenStack images can reside on a file system or within the Object Storage service.

Each image contains a virtual disk on to which a bootable operating system is installed. Images are used as the basis for virtual machine instances. When an end user requests a new virtual machine, the Image Service is queried and then returns the requested image to the hypervisor which in turn instantiates a concrete virtual machine instance.

### 3.2.4 Networking Service [Neutron]

The Network Service provides granular control over networks and IP addressing within an OpenStack cloud. In a cloud environment networking can quickly become complex especially when there are multiple tenants, each with potentially different networking requirements [18]. At its simplest level, the OpenStack Networking service enables cloud consumers to manage connectivity between virtual devices in the cloud and the levels of network access these devices should have to the outside world. The networking service also addresses a key concern of multi-tenant cloud, namely that tenants can easily create and manage networks where traffic is securely isolated from the networks of other tenants. The Networking Service supports different network models include flat networks and VLANs.

### 3.2.5 Block Storage Service [Cinder]

The Block Storage service provides an abstraction layer for managing pools of block level storage devices within an OpenStack cloud. Block level storage is suitable for workloads that are performance sensitive and have intensive I/O requirements e.g. databases and

virtual machines. Only storage devices that have an appropriate Block Storage driver can be used by the service. The Block Storage driver is required to enable direct access to the storage device and thus ensures high performance I/O.

The Cinder API allows end users and services to request block based resources without knowing about how the underlying storage is actually implemented or where it resides [19]. Once a storage device has been added, storage volumes can be created and made available to other OpenStack services such as OpenStack Compute (Nova) for use with virtual machines. In addition to creating new volumes, the Cinder API provides services for volume management including the ability to delete volumes and take point in time snapshots of a volume.

### 3.2.6 Object Storage [Swift]

Object level storage devices are managed using the Object Storage service. Its primary role is to provide a secure, highly available, cost effective storage for massive quantities of data. The type of data that is most suited to object storage is data that changes infrequently and is not retrieved regularly. This makes it ideal for long term data archival such as that required by banks and government institutions.

### 3.2.7 Dashboard [Horizon]

The Dashboard is a web based user interface for administrators and users alike. For administrators, it provides a bird's eye view of the health of the compute, network and storage resources that make up the cloud. Administrators can also manage tenants, users and quotas for resource usage. For administrators that want to use the command terminal, the OpenStack Command Line Interface (CLI) allows commands to be issued to any of the API's on each of the respective OpenStack services. The CLI is very powerful as it can be used to develop reusable scripts using bash or python.

For end users, they get an easy to use self-service portal that allows them to provision new resources in an OpenStack cloud. The Dashboard is designed to be extensible so that third parties can create add-on components that further enhance its functionality. In addition the Dashboard user interface can be easily customised to match the corporate branding of commercial cloud service providers.

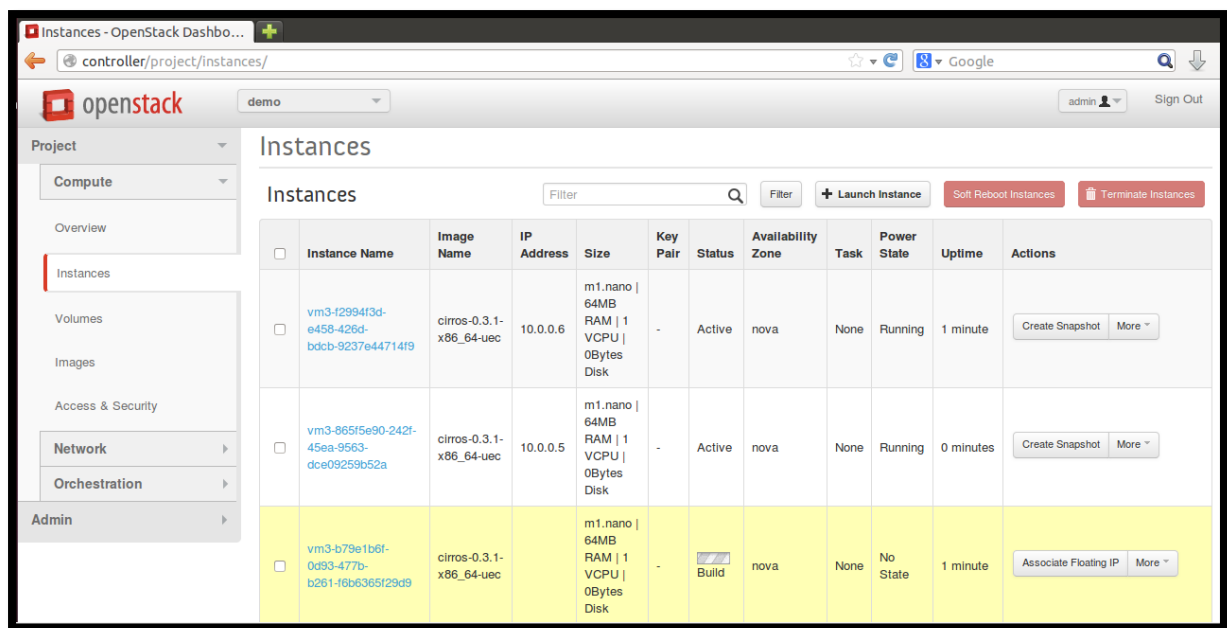


Figure 3 OpenStack Horizon dashboard

OpenStack is an evolving project whose scope continues to expand. There are many other non-core projects within OpenStack but these are not of relevance to this research paper.

### 3.3 Sentiment

The 2015 RightScale State of the Cloud Report [1] shows that 13% of respondents are currently using OpenStack for private cloud and 30% are evaluating or planning to use it in the future.

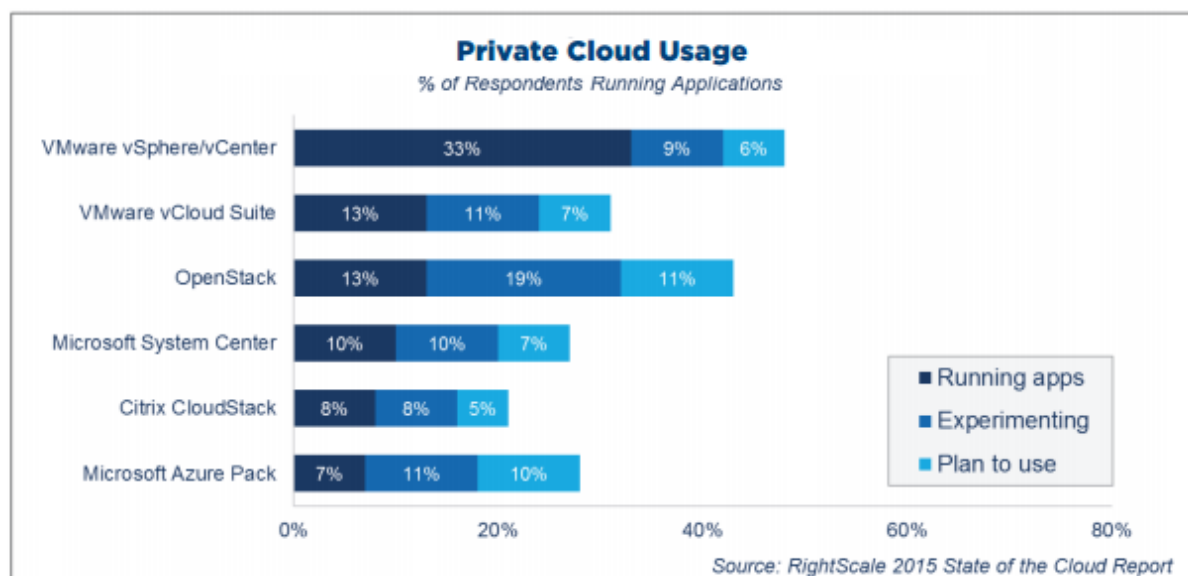


Figure 4 Private Cloud Usage 2015 (Source: RightScale 2015 State of the Cloud Report)

OpenStack has captured the imagination of developers and in particular those in the open-source community who see OpenStack as being in a similar position to where Linux was in

the mid-nineties. The success achieved by Linux in the server market demonstrates that large open-source solutions can deliver at the enterprise level. Fortune 500 companies like IBM and HP are also betting heavily on OpenStack being around for the long term and have committed significant time and money to its future development. There have been some notable wins for OpenStack. Walmart, one of world's largest retailers has an OpenStack installation with 100k CPU Cores. US production traffic to Walmart.com has been running successfully on OpenStack Compute since the end of 2014. [20]

Not all commentary on OpenStack has been positive however. Gartner have expressed their concerns about the quality of core functionality and the need to improve both stability and usability. They have also called out OpenStack marketing for over exaggerating its success at enterprise level. In 2013 Gartner research director Allesandro Perilli criticised the project in this regard; "Don't believe the hype generated by press and vendor marketing: OpenStack penetration in the large enterprise market is minimal."



## 4 Virtual Machine Scheduling

The primary function of a cloud service is the efficient management of resources, in particular network, storage and compute resources. One of the key functions within resource management is the scheduling of new virtual machines and their placement onto a physical host within the datacenter.

### 4.1 Management Objectives

Cloud service providers define “management objectives” [21] that outline a strategy for how resources are allocated. These objectives generally seek to maximise hardware utilisation while keeping operating costs low. Some common management objectives include:

- *Balanced load* – ensuring that CPU and memory are equally balanced across servers.
- *Fault tolerance* – workloads are provisioned so that they can withstand the failure of one or more servers.
- *Reduced power consumption* – workloads are consolidated on to the minimal number of servers possible while idle servers are placed into standby mode to conserve power. Power and cooling alone account for 82% of datacenter infrastructure costs. [22]

These management objectives are further complicated by the need to continuously conform to the Service Level Agreements (SLAs) agreed with cloud consumers. In addition, the cloud service provider may also offer different levels of service such as Platinum, Gold or Bronze to different customers.

In addition, there are many constraints that need to be considered. High availability constraints prevent specific virtual machines from being placed on the same physical host (a concept known as anti-colocation[23]). There are security constraints that may require virtual machines from different tenants to be physically isolated so that virtual machines belonging to either tenant never reside on the same physical host. Resource constraints [24] also need to be borne in mind so that the virtual machine is assigned to a host that has sufficient CPU and memory to run it. These are just some of the constraints that must be borne in mind when determining how resources are allocated in a cloud environment. At times, constraints may conflict with the interests of the cloud consumer and cloud service provider. It is the job of the resource management system to satisfy the needs of the cloud consumer from a service level perspective and the needs of the cloud service provider from a performance and efficiency point of view.

### 4.2 Virtual Machine Placement

Virtual machine placement is the process of determining the physical location for newly provisioned virtual machines. The placement of virtual machines plays a key role in allowing management objectives to be realised. Within the datacenter there exists many physical machines, each running a hypervisor capable of hosting multiple virtual machines. Ideally all physical machines will have the same hardware specification (homogeneity). However, in a

large datacenter where hardware is being continuously added, a heterogeneous environment is very likely to exist.

#### 4.2.1 The Bin Packing Problem

Different algorithms have been developed for this problem with many being modelled as variations of the “Bin Packing” problem [25]. The “Bin Packing” problem asks what is the minimum number of bins (of equal size) required to store a collection of objects (of variable size) such that no bin contains objects whose sum exceeds the bin’s capacity. In the context of virtual machine placement, the physical machines correspond to the bins while the virtual machines correspond to the objects that get placed inside the bins. The “Bin Packing” problem is categorised as an NP-hard problem which means that calculating the optimal solution is computationally expensive. Instead, heuristic algorithms [25] such as “best fit” and “worst fit” are used to quickly find an approximate solution though not necessarily an optimal solution to the problem.

For best fit, items are placed into the next available bin that has free capacity to accept the item and leaves the least room left over. For worst fit, the items are placed into the next available bin that has free capacity to accept the item and leaves the most room left over.

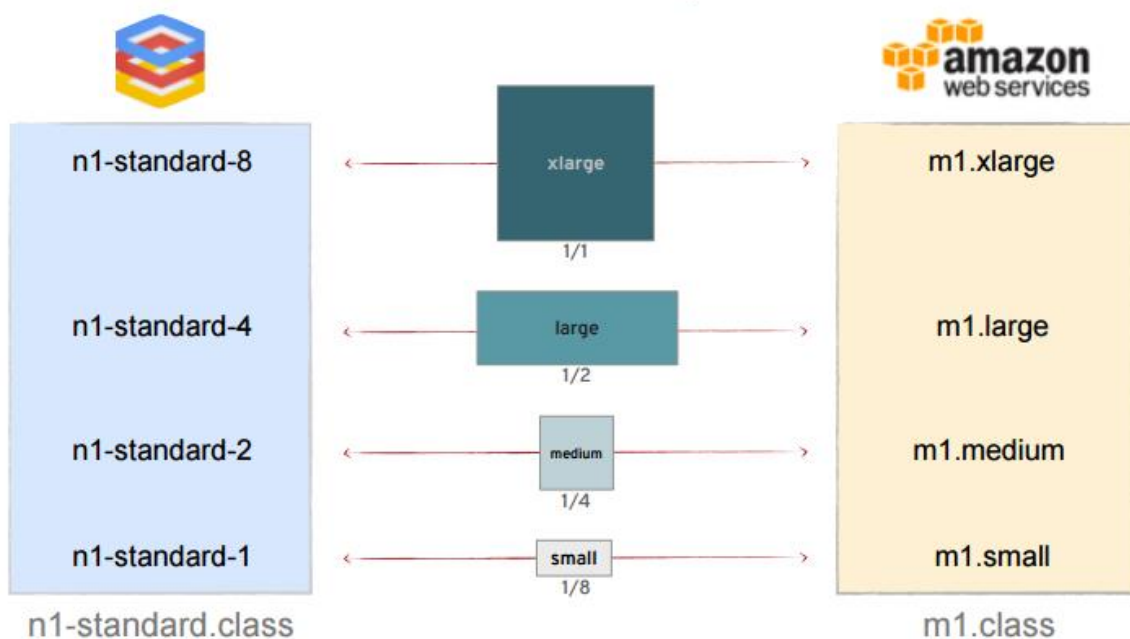


Figure 5 Public Cloud Virtual Machine Instances Sizes (Source: 2014 Red Hat Summit)

From a capacity planning perspective, the size of each virtual machine needs careful consideration to enable efficient hardware use and scheduling. Figure 5 shows the different virtual machine types provided by Google and Amazon, two of the largest public cloud service providers. Users do not submit arbitrary virtual machine sizes for provisioning. Instead, the service provider carefully controls the virtual machine flavours made available for selection. It is up to the end user to pick a flavour from the menu of available virtual machines that best suits their business needs. The size of each virtual machine flavour is

such that starting with the largest flavour, each subsequent flavour is exactly half the capacity of the preceding one, a concept described by Keith Basil of Red Hat as “fractional proportions”. [26]

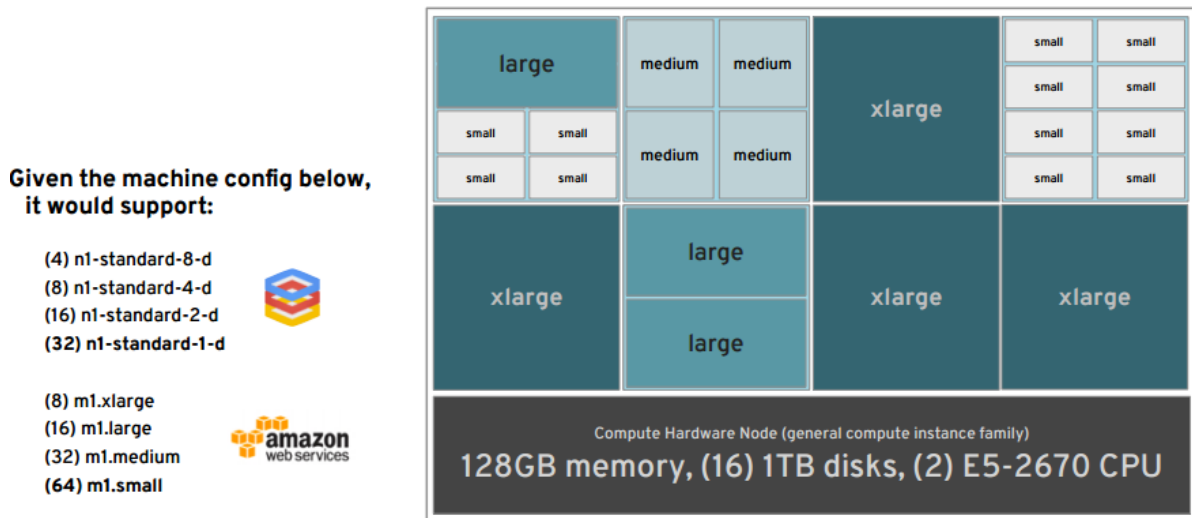


Figure 6 Efficient bin packing using fractional proportions (Source: 2014 Red Hat Summit)

Taking this approach provides a very clean solution to the bin packing problem for virtual machines. Once a bin (host) can be found for the largest virtual machine size then all other flavours are immediately solved since they are in fractional proportion to the largest. The size of the largest virtual machine flavour will dictate the specification of the hardware for the compute nodes that will host virtual machine instances. An example of this can be seen in Figure 6.

#### 4.2.2 Placement Strategies

Placement of virtual machines is a tricky problem and an area of significant research for both industry and academia. [27] Placement of virtual machines matters for many reasons include performance, security, availability and cost. From an availability standpoint, virtual machines need to be placed in different “failure domains” so that the failure of a single virtual machine does not affect other virtual machines.

When provisioning a multi-virtual machine three-tier application it is necessary to consider the relationship between different virtual machines. [28] Scheduling the virtual machines independently and hoping for the best can result in a single point of failure which is to be avoided at all costs when building a distributed system. An example of this can be seen in Table 1 below.

Table 1 Random placement of virtual machines in a 3-Tier application

Host 1	Host 2	Host 3
--------	--------	--------

Database Server VM1	Web Server VM1	Load Balancer VM1
Database Server VM2	Web Server VM2	Load Balancer VM2
	Web Server VM3	

In Table 1 there are 3 host servers available on to which virtual machines can be placed. The three tier application consists of two database server virtual machines, three web server virtual machines and two load balancer virtual machines. The application requires a minimum of a single Database Server, Web Server and Load Balancer to function properly. However, with this distribution of virtual machines, the failure of a single host will result in the entire application becoming unavailable.

The placement strategy needs to apply anti-affinity rules when making placement decisions to prevent specific virtual machines from residing on the same host. Applying anti-affinity rules would yield the following placement which is more fault tolerant since it allows for at least a single host to fail while allowing the three-tier application to continue operating:

*Table 2 Placement of virtual machines in a 3-Tier application using anti-affinity rules*

Host 1	Host 2	Host 3
Load Balancer VM1	Web Server VM2	Web Server VM3
Database Server VM1	Database Server VM2	Load Balancer VM2
Web Server VM1		

For the resource management software to make a decision, there are many factors to be considered. For example, it needs to know the characteristics of the virtual machine being requested. It needs to be aware of the available free resources on each physical machine in the datacenter. Finally, it needs to be aware of the constraints that apply to each customer and the constraints of the cloud provider. All of these inputs need to be processed in order to make the right decision.

#### 4.2.2.1 Revisiting Placement Decisions

After the initial placement of the virtual machine has been made, there are scenarios where that virtual machine may subsequently be moved to a new host. Such scenarios can occur due to failure of the host or due to changes in load patterns across the datacenter. These situations illustrate the dynamic nature of resource usage in a cloud environment and the need to be able to quickly respond by repositioning virtual machines on other hosts in the

datacenter. Real world solutions to this problem include VMware's Dynamic Resource Scheduler (DRS) which automatically moves virtual machines from one physical host to another. This can be of benefit when one server comes under load within the cluster as it allows virtual machines to be moved to a server with more available resources. Movements are made based on the need to fulfil objectives such as balancing system load, reducing power consumption, maximising resource utilisation etc. DRS uses VMware's vMotion technology which enables virtual machines to be transferred from one physical host to another without any interruption in service.

## 5 Virtual Machine Scheduling in OpenStack

An OpenStack cloud is made up of physical machines known as “nodes.” These nodes typically run a UNIX based operating system and have one or more OpenStack services installed on them which determines the type of the node. Four different types of node can be configured; controller, compute, storage and network.

### 5.1.1 Controller Node

The controller node is the management node for the OpenStack cloud. It runs services such as the identity service (Keystone), dashboard (Horizon) and the compute service (Nova). The controller node is an intelligent node that makes operational decisions. The Nova-Scheduler service runs on the controller node and it is this service that makes decisions about where new virtual machines will be launched. When the Nova-Scheduler has made a placement decision, it issues a request to the target compute node instructing it to host the new virtual machine.

### 5.1.2 Compute Node

The compute node runs the compute service (Nova) but it does not make scheduling decisions. Instead the compute node receives instructions from the controller node about virtual machine instances that it should host.

### 5.1.3 Storage and Network Nodes

The storage node is configured with the Object Storage (Glance) and Block Storage (Cinder) services which provide object and block storage capabilities to virtual machines. Network nodes run the Network service (Neutron) which provides network connectivity between virtual machines.

## 5.2 Nova Compute Service

Requests for new virtual machines are made via the Nova-API which places them in a message queue for processing by the nova-scheduler component. Nova-scheduler has a modular architecture which allows different scheduling algorithms to be configured for use. The default OpenStack scheduler is the filter scheduler which uses a 2 step process of filters and weights when choosing a suitable compute node.

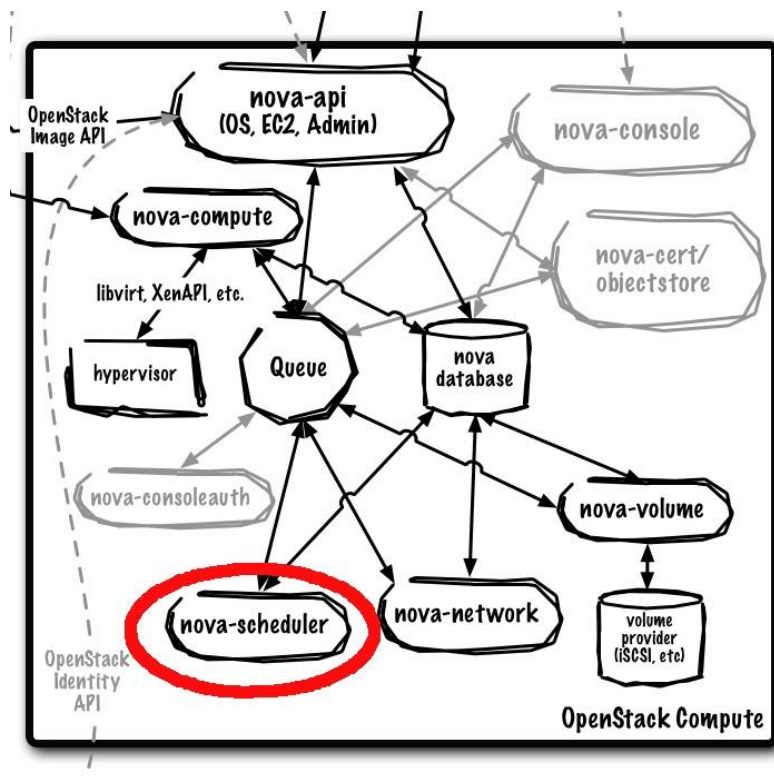


Figure 7 Nova Components (Source: OpenStack.org)

### 5.3 OpenStack Scheduling Algorithms

The nova-schedule component is a daemon that runs inside the Nova Compute Service and is responsible for scheduling new virtual machine placements within an OpenStack cloud. It implements a pluggable architecture that enables the active scheduling algorithm to be to be altered by modifying the scheduler\_driver property in the nova.conf configuration file. There are two different scheduling algorithms that ship with OpenStack; FilterScheduler and ChanceScheduler.

#### 5.3.1 Filter and Weight Scheduler

The Filter Scheduler is the default scheduler used for the placement of virtual machines in an OpenStack environment. The Filter Scheduler is enabled by editing the scheduler\_driver settings in the nova.conf configuration file on the controller node. The nova.conf file is the configuration file for the Nova Compute Service and resides in the “/etc/nova” directory.

```

openstack@controller: ~/devstack
GNU nano 2.2.6 File: /etc/nova/nova.conf

scheduler_driver = nova.scheduler.filter_scheduler.FilterScheduler

```

Figure 8 Enabling the FilterScheduler via nova.conf

The algorithm operates a two-step approach to virtual machine placement. Firstly, the filtering process applies a set of binary filters to the list of available compute nodes. The result of the filtering process is zero or more compute nodes that are deemed as eligible to host the virtual machine instance. After the filtering process has completed a weighting algorithm selects the most suitable node from the filtered list of compute nodes.

#### 5.3.1.1 Filters

Filters are applied to the list of active hosts in the OpenStack environment. A filter is a binary condition that a candidate either satisfies or does not satisfy. Candidates that do not satisfy a filter are excluded from consideration. Some of the commonly applied filters are `RamFilter` which checks if the compute node has sufficient available memory and `CpuFilter` which checks if the compute node has the required number of available CPU's to satisfy the virtual machine request.

```
scheduler_driver = nova.scheduler.filter_scheduler.FilterScheduler
scheduler_default_filters=RamFilter,ComputeFilter
```

*Figure 9 Configuring the filters to be used by FilterScheduler via nova.conf*

The `scheduler_default_filters` attribute within the `nova.conf` configuration file defines the list of filters that should be used by `FilterScheduler` when filtering a list of suitable hosts. As you will see from Figure 9 a comma separated list of filters can be specified. A host must pass each of the applied filters before it is considered a successful candidate. This chaining together of different filters allows complex filtering conditions to be realised. After the filtering process has completed, a list of hosts that fulfil the minimum requirements of the new virtual machine specification remains. In the context of the bin packing problem, filtering returns a collection of bins that have the capacity to accept new objects.

##### 5.3.1.1.1 RamFilter

One of the most commonly used filters is the `RamFilter`. This enables a list of hosts to be filtered so that only those that satisfy the minimum memory requirements required to run the virtual machine remain.



```

--
37 class BaseRamFilter(filters.BaseHostFilter):
38
39     def _get_ram_allocation_ratio(self, host_state, filter_properties):
40         raise NotImplementedError
41
42     def host_passes(self, host_state, filter_properties):
43         """Only return hosts with sufficient available RAM."""
44         instance_type = filter_properties.get('instance_type')
45         requested_ram = instance_type['memory_mb']
46         free_ram_mb = host_state.free_ram_mb
47         total_usable_ram_mb = host_state.total_usable_ram_mb
48
49         ram_allocation_ratio = self._get_ram_allocation_ratio(host_state,
50                                                                 filter_properties)
51
52         memory_mb_limit = total_usable_ram_mb * ram_allocation_ratio
53         used_ram_mb = total_usable_ram_mb - free_ram_mb
54         usable_ram = memory_mb_limit - used_ram_mb
55         if not usable_ram >= requested_ram:
56             LOG.debug("%(host_state)s does not have %(requested_ram)s MB "
57                       "usable ram, it only has %(usable_ram)s MB usable ram.",
58                       {'host_state': host_state,
59                        'requested_ram': requested_ram,
60                        'usable_ram': usable_ram})
61             return False
62
63         # save oversubscription limit for compute node to test against:
64         host_state.limits['memory_mb'] = memory_mb_limit
65         return True
--

```

**Returns False when insufficient RAM to run new VM**

**Returns True if host has sufficient RAM to run new VM**

Figure 10 Source code for RamFilter (Source: OpenStack.org)

Examining the source code in Figure 10, we can see that the RamFilter works by inspecting the specification of the newly requested virtual machine. This data is extracted from the request by reading the “memory\_db” property of the specification (Line 45), which is the amount of RAM the new virtual machine instance requires. Next, the filter inspects the “free\_ram\_mb” property on the host which indicates the amount of free memory (in megabytes) available on the host.

The algorithm allows for the over-allocation of physical memory for virtual machines through the use of the “ram\_allocation\_ration” property (Line 49). Over committing memory is a hypervisor feature which allows more memory than is physically available on the host to be assigned to virtual machines. This is possible because virtual machines rarely require all of the memory assigned to them in order to function. The benefit of over allocation for the cloud service provider is that it enables greater resource utilisation in the datacenter by allowing virtual machines to be packed more densely on the physical machines. Within OpenStack over committing can be enabled by configuring the “ram\_allocation\_ratio” to a number greater than 1.

The function returns true if the amount of available memory on the host is greater than or equal to the amount of memory being requested. Otherwise the function returns false.

#### 5.3.1.2 Weighting

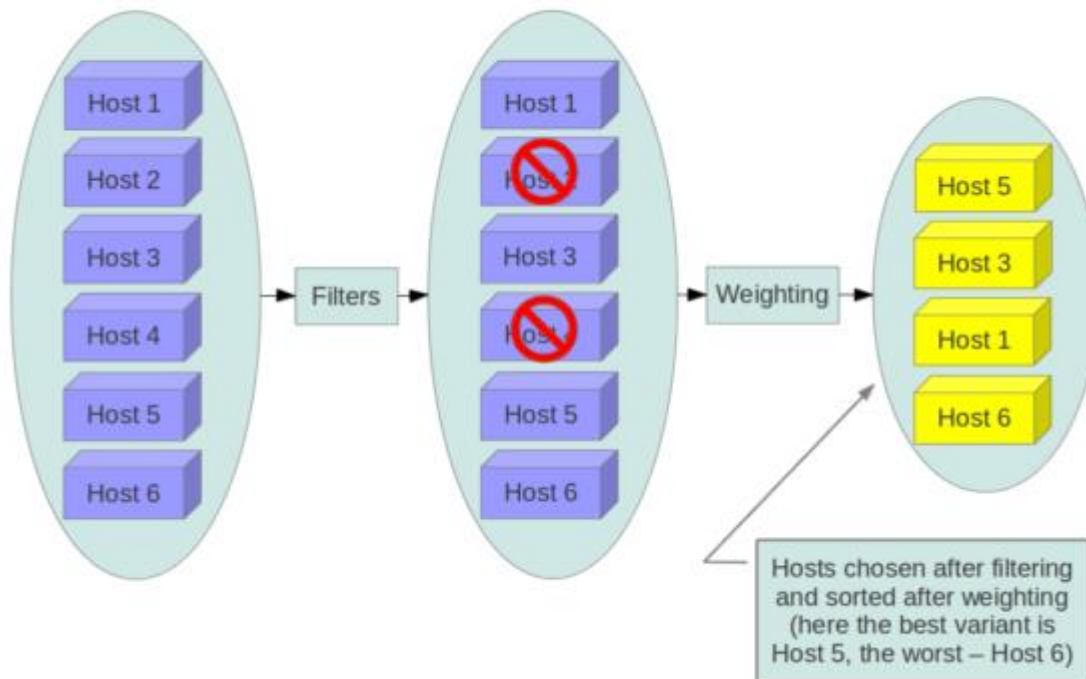


Figure 11 Filtering & Weighting of hosts (Source: OpenStack.org)

The weighting process is used to select the most suitable host from the list of filtered hosts. All hosts returned by the filtering process are capable of running the virtual machine that the scheduler is attempting to schedule. The purpose of the weighting process is to pick the host that is best suited to run the virtual machine. Within OpenStack there are two available cost functions[29]:

1. **Fill first:** this cost function looks at the amount of memory available on each host and multiplies this figure by the weighting value. The host with the lowest score is then selected as the best candidate by the scheduler. The weighting value influences how hosts will be filled. Specifying a value of -1 will result in the host with the most available memory being selected as the best candidate. However, specifying a value of 1 for the weighting will result in the host with the least available memory being selected. The weight setting can influence how hosts are filled. A positive value results in hosts being stacked more densely while a negative value results in hosts being filled more sparsely.
2. **Retry host:** this cost function penalises hosts previously picked by the scheduler as a suitable host for hosting a virtual machine instance but where the scheduling process did not complete successfully. Hosts that consistently fail will appear lower on the list despite passing the initial filtering process.

In Figure 11, the process begins by considering all available compute nodes in the OpenStack cloud. The filtering process applies one or more filters to each host to create a filtered list of compute nodes that meet the minimum requirements for the virtual machine request. In this case, host 1, 3, 5 and 6 meeting the filtering criteria. Next, the configured weighting algorithms are applied to each host on the list of filtered hosts. The results of each algorithm are summed together to produce a weighting/cost for that host. After the weighting process has completed, the compute node with the lowest overall weight is selected as the best candidate on which to host the virtual machine.

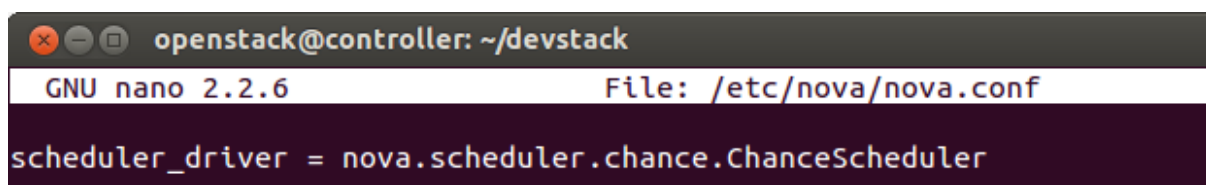
#### 5.3.1.3 *Least Fit vs Worst Fit*

The default weight in OpenStack is based on the amount of free RAM available on the compute node. This is implemented by the `Ram_Weight_Multiplier` class. The weight configuration influences how compute nodes are filled. In the context of bin packing, the weight can be configured so that the best or worst fit is found. For best-fit, the management objective is focused on optimising resource utilisation by densely stacking virtual machines on to the fewest number of compute nodes. For worst-fit, the focus is on maximising performance by placing virtual machines on the compute node with the most available resources. The worst fit approach results in virtual machines being spread across a greater number of hosts. [24]

The behaviour of the memory weighting algorithm is configured via the “`ram_weight_multiplier`” property in the `/etc/nova/nova.conf` configuration file. Setting a value of 1.0 for this property results in a “best fit” approach while setting a value of -1.0 results in a “worst fit” approach to virtual machine placement.

#### 5.3.2 *Chance Scheduler*

The Chance Scheduler is a simplistic scheduler. Similar to the Filter and Weight scheduler, it consists of two stages. Firstly, it performs the same filtering process to eliminate hosts that do not meet the criteria for the new virtual machine request. Following that, it selects a host at random from a list of filtered hosts. The Chance Scheduler is enabled by editing the `scheduler_driver` setting in the `/etc/nova/nova.conf` file.



```
openstack@controller: ~/devstack
GNU nano 2.2.6                               File: /etc/nova/nova.conf
scheduler_driver = nova.scheduler.chance.ChanceScheduler
```

Figure 12 Enabling the ChanceScheduler via `nova.conf`

## 6 Code Walkthrough: Provisioning a new Virtual Machine in OpenStack

To understand where the scheduler fits into the overall process of provisioning a new virtual machine, it's necessary to take a step back and look at the sequence of steps involved in the provisioning process within OpenStack. For this, we analyse the OpenStack Nova Compute service source code and trace the sequence of events that occur after the end user clicks the "Launch" button in the Horizon dashboard to create a new virtual machine instance. The source code for the Nova Compute service is publicly available on the GitHub source code repository at <https://github.com/openstack/nova>.

### 6.1 Virtual Machine Request Attributes

Virtual machines in OpenStack can be launched via the Dashboard (Horizon) or programmatically using the Nova Compute API. Regardless of how the virtual machine is instantiated, there are specific attributes of the new virtual machine that must be supplied.

These attributes include:

Attribute	Description
<b>Instance Name</b>	The friendly name for the virtual machine instance.
<b>Flavour</b>	The flavour is a template that defines the memory, CPU and disk capacity of a virtual machine. A flavour also has an optional property called <code>extra_specs</code> that influences placement decisions by the scheduler. The property contains a list of key-value pairs and any compute node that is to be considered eligible to host a virtual machine of this flavour must also have these key-value pairs. The <code>extra_specs</code> property is intended for scenarios where a virtual machine needs to be run on a compute node with specific hardware capabilities.
<b>Instance Count</b>	This is the number of virtual machine instances that OpenStack should create.
<b>Instance Boot Source</b>	This property indicates if the new virtual machine should be created using an image stored in the Glance image service or from a volume in the Cinder block storage service.
<b>Security Group</b>	Security groups control which virtual machines are permitted to communicate with one another. The request may contain a list of security groups that the virtual machine will be added to upon creation. If no security group is specified in the request then the

default security group is used instead. The default security group allows all virtual machines within this group to communicate with one another and excludes network traffic from all other sources. [31]

### Launch Instance

Details \* Access & Security \* Networking \* Post-Creation Advanced Options

Availability Zone:  
nova

Instance Name: \*  
NewInstance1

Flavor: \*  
m1.nano

Instance Count: \*  
1

Instance Boot Source: \*  
Boot from image

Image Name:  
cirros-0.3.1-x86\_64-uec (24.0 MB)

Specify the details for launching an instance.  
The chart below shows the resources used by this project in relation to the project's quotas.

#### Flavor Details

Name	m1.nano
VCPUs	1
Root Disk	0 GB
Ephemeral Disk	0 GB
Total Disk	0 GB
RAM	64 MB

#### Project Limits

Number of Instances0 of 10 Used

Number of VCPUs0 of 20 Used

Total RAM0 of 51,200 MB Used

Cancel

Launch

Figure 13 Launch Instance Panel in Horizon Dashboard Portal

## 6.2 Processing New Virtual Machine Request

After the “Launch” button is clicked, the request for a new virtual machine instance is constructed. Attributes in the request such as memory and CPU are used as inputs by the scheduler to make placement decisions for the virtual machine instance.

In the source file `nova/api/openstack/compute/servers.py` an action method “`create()`”, on the Controller class, handles the “Launch” button click event on the page. This method extracts the contents of the request, performs some simple high level validation such as checking that the server name provided does not exceeds maximum permitted length. After extracting each attribute of the virtual machine request, it makes a call to the “`create()`” method on the Nova Compute API.

### 6.2.1 Compute API create() method

In the source code file `"/nova/compute/api.py"` the Compute API `create()` method orchestrates the process of creating a new virtual machine instance. Firstly, a security check is performed to ensure that the user who submitted the virtual machine request is authorised to do so. This check is carried out by the method `_check_create_policies()`. OpenStack implements role based access control policies that restrict method calls on the API to users that are members of a specific security role. The security policy for the Nova service is defined via the file `/etc/nova/policy.json`. This file enables granular control over what functionality on the API is available to administrative and non-administrative users.

The following security policy checks are performed:

1. Checks the 'compute:create' policy to verify that the user has permission to create a virtual machine.
2. Checks the 'create:attach\_network' policy to verify that the user has permission to attach a virtual machine to a network.
3. Checks the 'create:attach\_volume' policy to verify that the user can attach a storage volume to a virtual machine.

If any of the above security checks fail, a `PolicyNotAuthorized` exception is raised. Otherwise, the method continues with network related checks for the virtual machine request. If the request specifies that multiple virtual machine instances be created and that each should be added to a network then a check is performed to ensure that an attempt is not made to assign the same IP address to multiple virtual machine instances. Next the code checks if Neutron, a Software Defined Networking (SDN) component, is enabled on the OpenStack cloud. If Neutron is enabled, a check is made to ensure that no attempt is made to launch multiple virtual machines with the same port number. If this network check fails, a `MultiplePortsNotApplicable` exception is raised.

In the event that all checks pass, a call is made to the method `_create_instance()`

### 6.2.2 `_create_instance()`

A reservation id, a unique 8 character identifier is generated and associated with the request. Attributes on the request are checked and in some cases are assigned a default value before the instance(s) are scheduled for creation.

Property	Default Value
<b>security_groups</b>	If no security group is specified in the request then the virtual machine is assigned to the "default" security group.
<b>min_count/max_count</b>	min_count and max_count control the number of virtual machine instances that should be provisioned. If no value for

	these properties is supplied then both values are initialised to one.
<b>instance_type</b>	If a virtual machine flavor is not specified in the request, the default flavor “m1.small” is used instead.
<b>availability_zone</b>	An Availability Zone enables compute hosts that share common attributes to be assigned into logical groupings such as hosts that share the same power supply could be placed in their own availability zone. When creating a virtual machine instance it is possible to specify the availability zone that the virtual machine should be created in. If no availability zone is specified then the virtual machine is assigned to the default availability zone.

If booting from a specific image, the details of this image are loaded. Otherwise, the virtual machine is booting from a volume and a call is made to `_get_bdm_image_metadata()` to load volume metadata from Cinder block storage service.

Next, the `_validate_and_build_base_options()` method is called to validate data in the request:

- Checks that the `availability_zone` specified on the request exists and is available for use. An `InvalidRequest` exception is raised if this check fails.
- If the virtual machine flavor is in a “disabled” state, a `FlavorNotFound` exception is raised.
- Checks that if user data is supplied in the request that it does not exceed the maximum valid length (65535) for storage in the database. An `InstanceUserDataTooLarge` exception is raised if the data exceeds the maximum permitted size. An attempt is made to read this base64 encoded data. An `InstanceUserDataMalformed` exception is raised in the event that the data could not be read.
- The helper method `_check_metadata_properties_quota()` validates metadata if supplied, by ensuring that the number of metadata items does not exceed the permitted quota.
- The helper method `_check_injected_file_quota()` checks that the number of injected files does not exceed the permitted quota. The file path and file content are checked to ensure they do not exceed permitted maximum lengths.
- The `_check_requested_image()` helper method checks that the image used by this virtual machine has a status = active and that the minimum memory and disk requirements are satisfied for the selected image flavor.
- The `_check_requested_security_groups()` ensures that the security groups specified in the request exist. If a security group is found not to exist then a `SecurityGroupNotFoundForProject` exception is raised.

- The `_check_requested_networks()` checks if the networks in the request are valid, that the fixed IP address belongs to the network and that the IP address is valid and not already in use. A number of exceptions can potentially be raised by this section including `FixedIpInvalid`, `FixedIpNotFoundForNetwork` and `FixedIpAlreadyInUse`. The method also returns the maximum number of virtual machine instances that can be created in this request adjusted for network quota constraints. This can potentially mean that the number of virtual machines instances created will be lower than what was originally requested.

The `validate_and_build_base_options()` method returns a data structure that contains a validated list of options for the virtual machine request.

```
base_options = {
    'reservation_id': reservation_id,
    'image_ref': image_href,
    'kernel_id': kernel_id or '',
    'ramdisk_id': ramdisk_id or '',
    'power_state': power_state.NOSTATE,
    'vm_state': vm_states.BUILDING,
    'config_drive': config_drive,
    'user_id': context.user_id,
    'project_id': context.project_id,
    'instance_type_id': instance_type['id'],
    'memory_mb': instance_type['memory_mb'],
    'vcpus': instance_type['vcpus'],
    'root_gb': instance_type['root_gb'],
    'ephemeral_gb': instance_type['ephemeral_gb'],
    'display_name': display_name,
    'display_description': display_description or '',
    'user_data': user_data,
    'key_name': key_name,
    'key_data': key_data,
    'locked': False,
    'metadata': metadata or {},
    'access_ip_v4': access_ip_v4,
    'access_ip_v6': access_ip_v6,
    'availability_zone': availability_zone,
    'root_device_name': root_device_name,
    'progress': 0,
    'pci_requests': pci_request_info,
    'numa_topology': numa_topology,
    'system_metadata': system_metadata}
```

Figure 14 Validate list of virtual machine options from `_validate_and_build_base_options()`

The `_check_and_transform_bdm()` method is invoked to check rules around block level device mappings for the virtual machine request.



The `_get_requested_instance_group()` method is invoked to extract any server group specific scheduler hints specified on the request (if any).

Next, the `_provision_instances()` method first makes a call to `_check_num_instances_quota()` which calculates the number of CPU cores, memory and video memory required to complete this request. An attempt is then made to place a reservation on these resources for this request. If the reservation violates the remaining resources quota then an `OverQuota` exception is raised. If the reservation is successful, a call is made to `create_db_entry_for_new_instances()` which records details of each new virtual machine instance in the Nova MySQL database. The virtual machine is recorded with an initial state set to “Building”. A call is made to the method `_validate_bdm()` to verify that storage volumes for the virtual machine instance are valid and that connections between them and the virtual machine can be established without error.

`_build_filter_properties()` – When launching a virtual machine via the “nova boot” command it is possible to specify scheduler hints which override scheduler behaviour and influences placement decisions. This helper method looks for and extracts scheduler hints into a dictionary that is returned to the caller and later passed through to the scheduler. In particular, this method is interested in the scheduler hints “force\_hosts”, “force\_nodes”, “pci\_requests” and “instance\_type”. These filter properties allows the request to override the normal filtering process and force the virtual machine instance to be placed on a specific host.

Finally a Remote Procedure Call (RPC) is made to the `build_instances()` method on the `compute_task_api`.

### 6.2.3 `build_instances()`

In the code file `/nova/conductor/rpcapi.py`. The `build_instances()` method inspects the request and builds a makes a call to appropriate version of the `build_instances()` method on the API.

In the code file `/nova/conductor/api.py`: The `build_instances()` method passes the incoming request off to the `build_instances()` method on the `ConductorManager`. It uses `utils.spawn_n` to invoke this method.

The code file `/nova/conductor/manager.py` is where scheduling decisions are made. Firstly, a call is made to `build_request_spec()` which is a helper method in the scheduler utilities. This method creates an object of type `request_spec` that contains the information the scheduling algorithm requires in order to make a placement decision. This `request_spec` stores details of the image, the number of instances to be scheduled and properties set on the virtual machine.

Next, the `setup_instance_group()` method checks to see if filters that apply affinity/anti-affinity rules (`ServerGroupAffinityFilter` and `ServerGroupAntiAffinityFilter`) have been

enabled via `/etc/nova/nova.conf` configuration file. If either of these filters is enabled, then `groups_hosts` and `group_policies` fields are added to the `filter_properties` dictionary.

```
532         try:
533             scheduler_utils.setup_instance_group(context, request_spec,
534                                                  filter_properties)
535             scheduler_utils.populate_retry(filter_properties, instance['uuid'])
536             hosts = self.scheduler_client.select_destinations(
537                 context, request_spec, filter_properties)
538             host_state = hosts[0]
539         except exception.NoValidHost as ex:
540             vm_state = instance.vm_state
541             if not vm_state:
542                 vm_state = vm_states.ACTIVE
543             updates = {'vm_state': vm_state, 'task_state': None}
544             self._set_vm_state_and_notify(context, instance.uuid,
545                                          'migrate_server',
546                                          updates, ex, request_spec)
547         quotas.rollback()
```

Figure 15 Extract from `/nova/conductor/manager.py` showing call to `select_destinations` method on scheduler driver

Finally, a call is made to the `scheduler_client.select_destinations()` method, passing the `request_spec` and `filter_properties` that will be used as the input for the scheduling decision. The `scheduler_client` object is initialised in the constructor of the `The SchedulerManager` is responsible for loading the appropriate scheduler driver instance that is configured via the “`scheduler_driver`” attribute in the Nova configuration file in `/etc/nova/nova.conf`. The call to `select_destinations()` results in the scheduler making a placement decision that will return the details of a suitable host (compute node) for the virtual machine image. In the event that the scheduler is unable to determine a suitable host for the request, a `NoValidHost` exception is raised and the process is aborted.

```
class SchedulerManager(manager.Manager):
    """Chooses a host to run instances on."""

    target = messaging.Target(version='4.0')

    def __init__(self, scheduler_driver=None, *args, **kwargs):
        if not scheduler_driver:
            scheduler_driver = CONF.scheduler_driver
            self.driver = importutils.import_object(scheduler_driver)
            super(SchedulerManager, self).__init__(service_name='scheduler',
                                                  *args, **kwargs)
            self.additional_endpoints.append(_SchedulerManagerV3Proxy(self))

    @periodic_task.periodic task
```

Figure 16 Highlighted lines in `SchedulerManager` showing the configured scheduler being loaded.

Once the scheduler has determined the host for the virtual machine instance, a call is made to `_build_and_run_instance()` which is responsible for building the virtual machine instance and running it on the target host.

## 7 Third Party OpenStack Schedulers

The pluggable design of the OpenStack architecture allows developers to create their own scheduling component. This can be tailored to suit their operating environment and overcome the limitations of the standard Nova scheduler drivers. Alternatively, the existing scheduler can be plugged out and swapped for drivers purchased from a third party supplier.

### 7.1 FairShare Scheduler

CERN (European Center for Nuclear Research) is a major proponent of OpenStack and uses it to enable their researchers to rapidly provision virtual machines for a variety of different uses cases. [32] Tim Bell, the infrastructure service manager for CERN current sits on the board of directors of the OpenStack Foundation and CERN operates 4 OpenStack (Icehouse release) Clouds, the largest of which has over 75,000 CPU cores across 3,000 physical servers. RackSpace and CERN have recently collaborated to implement Federated identity of OpenStack clouds within the OpenStack cloud project. [26] Federated cloud is a concept that enables an OpenStack user in one cloud to authenticate themselves and use resources in another cloud providers OpenStack cloud.

Lisa Zangrando of CERN has outlined some of the short comings of the traditional OpenStack scheduler. The main shortcomings identified by Zangrando of the default OpenStack scheduler were:

1. Schedule requests that could not be satisfied are allowed to simply fail. They are not queued up and retried again later.
2. There is no priority mechanism for requests and as a result all requests are processed on a first come first served basis.

CERN has developed their own custom scheduler known as FairShareScheduler which can plug into an existing OpenStack cloud to perform scheduling activities. The concept of a fair share scheduler (FSS) is one that has been present in operating system schedulers and batch schedulers for many years. Operating systems such as Oracle Solaris [34] have different scheduling policies that control the allocation of CPU resources. One of these policies is the fair share policy that assigns a priority to each workload. This contrasts with equally dividing resources among the number of competing processes. Instead, workloads are assigned a number of shares and resources are allocated proportional to the number of shares held by each workload.

In OpenStack, the FairShare Scheduler applies a similar policy so that a specific tenant can be assigned a share of the available pool of resources. For example, if there are two tenants defined on the OpenStack cloud it is possible to assign tenant 'A' 65% of available resources with the remaining 35% being assigned to tenant 'B'. Configuring such a scenario is possible by setting the project\_shares attribute via the nova.conf file:

```
project_shares={'A':65, 'B':35}
```

The scheduler allows tenants to exceed their designated resource usage in the event that other tenants have unused capacity. In the scenario where tenant 'A' is not using the full 65% share allocated to it then tenant 'B' is permitted to use these available resources. Likewise when tenant 'B' is not using its full allocation, tenant 'A' is permitted to exceed its 65% share and use any spare capacity available from tenant 'B'. This approach maximises the utilisation of system resources.

Within each tenant, the FairShare scheduler allows resources to be prioritised among users. The default behaviour is that each user is allocated a 1% share of resources. This can be altered so that specific users are given a larger share than others. This is configured via the `user_shares` attribute in the `nova.conf` file. For example, to allocate 'user\_id\_1' a 20% share of resources with tenant 'A' it would be as follows:

```
user_shares={'A':{'user_id_1':20}}
```

## 7.2 IBM – Platform Resource Scheduler

IBM have many years of experience in the area of scheduling and resource management, most notably in the area of mainframe technology which to this day underpins the mission critical systems used by many of the world's largest businesses. Since 1999, IBM's business strategy has been built around open-source and the development of open standards. [27] This is in marked contrast to their position before that time where they operated a strategy based on proprietary software.

### 7.2.1 The Goldilocks Dilemma

Gord Sissons from the IBM's Platform Computing division has identified two key limitations in how the current OpenStack scheduler is implemented[28]:

1. It makes its decisions based on static data stored in the Nova SQL Database. The data stored in this database tracks the resource usage (CPU, Memory) for each physical node in the OpenStack cloud.
2. Scheduling decisions about the initial placement of the virtual machine. However, there is no mechanism that allows OpenStack scheduler to review placement decisions periodically to ensure that the virtual machines are optimally placed based on current resource usage.

Demand for computing resources can quickly go from low to high and the scheduler needs to be able to react to these sudden shifts in resource usage patterns. The fact that resource usage is based on figures recorded in the Nova SQL database and not based on actual performance metrics from the virtual machine means that the scheduler may not be getting a clear and accurate picture of resource usage which may lead to poor placement decisions. This can manifest itself in two ways. The physical host can become overloaded with virtual machines because the measurements in the Nova database understate the actual usage. Conversely the physical host can be underutilised because the measurements in the Nova database overstate the actual usage. The first instance Sisson refers to the hosts operating "too hot" as they are overloaded with virtual machines. This can lead to a scenario where

virtual machines are starved of the minimum resources they require thus resulting in poor performance for end users. The second instance Sisson refers to the hosts operating “too cold” as they are underutilised. This increases the operating costs of the datacenter as physical hosts are not operating to their maximum capacity or in a worst case scenario are sitting idle while still consuming power and cooling resources. IBM describes the “too hot” and “too cold” scenarios as the “Goldilocks dilemma” and have put forward the Platform Resource Scheduler as the solution to making scheduling decisions that are “just right”.

### 7.2.2 Policy based Scheduling

IBM’s solution to overcoming the limitations of the OpenStack schedule is the Platform Resource Scheduler (PRS). This implements a much more “realtime” approach to resource management as well continuous monitoring of the cloud environment. Unlike the default OpenStack schedulers, with PRS the story doesn’t end after a virtual machine placement decision has been made. The PRS continuously monitors resources on both virtual machines and physical compute nodes and using this information can identify situations where it makes sense to move a virtual machine from one physical host to another. This rebalancing is made possible using a feature in the hypervisor known as “live migration” which enables a virtual machine to be transferred from one host to another without any interruption in service. [29] This characteristic can be used to fulfil management objectives in the areas of performance, availability and maximisation of resources. It also assists with host maintenance by making it possible to move virtual machines to another host before applying software updates and performing host restarts.

PRS supports different placement policies that guide its decision making when placing and rebalancing virtual machines within the cloud environment [30]:

Policy name	Description
CPU Load Balance	CPU usage is evenly distributed across compute nodes.
Memory Load Balance	Memory is evenly distributed across compute nodes.
Striping	Virtual machines are evenly distributed across available compute nodes.
Packing	Virtual machines are stacked as densely as possible within compute nodes.

A single policy can be applied globally across all physical hosts in the cloud (Global Mode). It is also possible in OpenStack to carve out groups of compute nodes known as Host Aggregates and apply a different policy at the Host Aggregate level. The Host Aggregate feature allows advanced scheduling scenarios to be configured by making it possible to it logically group compute nodes together based on a specific capability of the host (e.g. isolate hosts that have solid state drives).

When a policy is selected, the PRS will automated check on a regular interval (by default every 1 minute) to ensure the cloud environment is adhering to the goals of the policy.

Policy configuration is performed via an XML based configuration file, `rtpolicy_conf.xml` which resides under the `/etc/nova` directory on the OpenStack controller node[31].

```
<placement_policies>
  <placement_policy>
    <id>1</id>
    <name>CPU Load Balance</name>
    <description>This policy's goal is to balance the CPU load as evenly as possible
      among the host systems on a best-effort basis.</description>
    <state>disabled</state>
    <run_interval>1</run_interval>
    <stabilization>3</stabilization>
    <goal>(vcpuConsumed+vcpuRequest)*100/vcpu</goal>
    <threshold>70</threshold>
    <max_parallel>25</max_parallel>
    <action>migrate_vm</action>
  </placement_policy>
</placement_policies>
```

*Figure 17 Placement policy configuration file (source: ibm.com)*

There are three attributes on the policy that instruct the PRS on how it should enforce the policy:

**Threshold:** the limit at which the policy is considered to have been violated.

**Stabilization:** the number of times that a policy can be violated before rebalancing is attempted.

**Maximum Migrations:** this is a hard limit on the number of live migrations that can take place within a cluster.

The continuous monitoring and rebalancing provide by IBM Platform Resource Scheduler greatly reduces the management overhead for the cloud administrator and is delivers the enterprise features that are required for large private and public cloud installations.

## 8 Building a Custom OpenStack Scheduler

The default OpenStack schedulers may not always satisfy the management objectives of the cloud service provider. The driver architecture in OpenStack makes it possible to purchase and deploy a scheduling solution from a third party. Alternatively, there is the option of creating and deploying your own custom scheduler.

### 8.1 Scheduler Base Classes

OpenStack drivers and plugins are written using the Python programming language. All custom schedulers must inherit from the `nova.scheduler.driver.Scheduler` abstract base class. An abstract base class is an object oriented programming construct [32] and in this case is used to enforce a pattern that all schedulers must inherit. This ensures that while the internal workings of each scheduler may be different, externally they will all conform to a standard interface/contract. The requirement that all schedulers implement a standard interface is part of the driver architecture within OpenStack. It is this standard interface that makes it possible to swap in different scheduler implementations simply by editing the `nova.conf` configuration file on the controller node.

The scheduler base class has the following public methods which form the contract that every OpenStack scheduler must implement.

Method name	Description
<code>run_periodic_tasks</code>	Used by internal manager classes within OpenStack
<code>hosts_up</code>	Returns a list of hosts that have a specific service running
<code>select_destinations</code>	Returns the list of hosts that are suitable candidates for placing a new virtual machine on.

Figure 18 Methods on the scheduler base class

Of the three methods listed in Figure 18, the custom scheduler must provide an implementation for the “`select_destinations`” method. This method contains the custom logic that decides on the most suitable compute node (host) for a new virtual machine instance. It is this logic that differentiates the behaviour of the custom scheduler from the default OpenStack schedulers.

```
def select_destinations(self, context, request_spec, filter_properties):
    """Must override select_destinations method.

    :return: A list of dicts with 'host', 'nodename' and 'limits' as keys
            that satisfies the request_spec and filter_properties.
    """
    msg = _("Driver must implement select_destinations")
    raise NotImplementedError(msg)
```

Figure 19 The `select_destinations()` method definition in the abstract base class `nova.scheduler.driver.Scheduler`



## 8.2 Tenant Scheduler

The OpenStack Operations Guide provides a good reference for creating and deploying a custom scheduler. As part of this research, a custom scheduler, TenantScheduler, was created. TenantScheduler enables the service provider to allocate specific compute nodes to a tenant. A tenant, is a container that is used by role based access security to enforce a logical separation of one OpenStack project/customer from another. Every user in OpenStack must be assigned to a tenant. Tenant management is performed using the Keystone authentication service.

### 8.2.1 Scheduler Algorithm

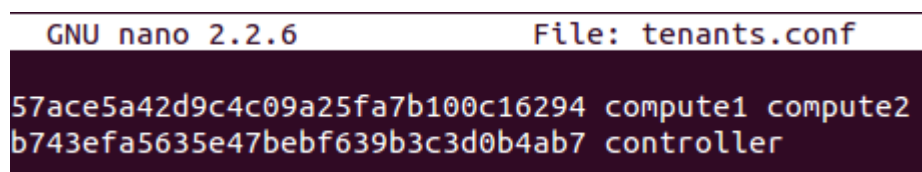
When a request to schedule a new virtual machine is submitted the tenant scheduler performs the following steps:

1. Determine the unique id of the tenant that submitted the virtual machine request.
2. Read the configuration file (tenants.conf) and retrieves the list of compute nodes (hosts) configured for this tenant.
3. If no hosts are found for this tenant, raise a critical exception.
4. From the list of all available hosts, filter out those that are not listed on the configuration file as being associated with this tenant.
5. Next, filter out any hosts that may have been specified on the (optional) ignore list.
6. From the hosts that remain after filtering in steps 4 and 5, pick one host at random.

### 8.2.2 Tenant Scheduler Configuration

The configuration file for the TenantScheduler has the following expected format:

<Tenant ID> <Hostname\_1> <Hostname\_2> ... <Hostname\_N>



```
GNU nano 2.2.6 File: tenants.conf
57ace5a42d9c4c09a25fa7b100c16294 compute1 compute2
b743efa5635e47bebf639b3c3d0b4ab7 controller
```

Figure 20 Sample tenants.conf file

There is a single line in the tenants.conf file for each tenant in the system. The first entry on the line is the Tenant ID, a unique identifier for the tenant. Since it is possible for tenants to share the same name, the unique identifier for the tenant was used instead. The tenant\_id can be obtained by querying the Projects table in the MySQL Keystone database on the controller node. Alternatively, the “keystone tenant-list” command can be issued to retrieve a list of all tenants and their id’s.

```
mysql> select id, name, enabled from project WHERE name IN ('demo','admin');
+-----+-----+-----+
| id | name | enabled |
+-----+-----+-----+
| 946b77f95513491386e38638ac79fbd0 | demo | 1 |
| fe751af561844aa6b492f00df0c0a6ce | admin | 1 |
+-----+-----+-----+
2 rows in set (0.00 sec)
```

Figure 21 retrieving the tenant\_id of the demo and admin tenants from the Keystone database

After the tenant id has been specified, the list of compute nodes (hosts) that are available for the tenant are provided. The list of all currently available compute nodes can be obtained by running the “nova hypervisor-list” command.

```
openstack@controller:~/devstack$ nova hypervisor-list
+-----+-----+
| ID | Hypervisor hostname |
+-----+-----+
| 1 | controller |
| 2 | compute1 |
| 3 | compute2 |
+-----+-----+
```

Figure 22 List of available hypervisors/hosts via hypervisor-list command

### 8.2.3 Source Code

The code for the custom scheduler requires the following helper methods:

1. A method to read the list of compute nodes from the tenants.conf configuration file.
2. A method to select a target destination/host for the virtual machine request.

A new file, tenant\_scheduler.py was created in the “/nova/scheduler” directory on the controller node. Inside this file, a new class called TenantScheduler was created which inherited from the nova.scheduler.driver.Scheduler base class.

#### 8.2.3.1 Get Hosts from Configuration File

A method `_get_tenant_hosts()` was created to fetch the list of compute node hosts configured for each tenant. The method reads the configuration file tenants.conf and parses the contents into a dictionary object. The dictionary consists of a set of key value pairs, with the key being the unique identifier of the tenant and the values a list of compute node names. If no entry is found the tenants.conf file matching the unique identifier of the tenant that owns the virtual machine placement request, an exception is raised.

```

def _get_tenant_hosts(tenantName):
    with open('c:\\temp\\tenants.conf') as configFile:
        line = [line.strip().split() for line in configFile.readlines()]
        tenantDictionary = {d[0]: d[1:] for d in line}

        hosts = tenantDictionary.get(tenantName, None)

    if hosts is None:
        raise Exception('No entry found in the TenantScheduler config file for this tenant.')

    return hosts

```

Figure 23 Python Code that reads and parses tenants.conf

### 8.2.3.2 Select Target Destination for a Virtual Machine

The private method `_filter_hosts()` is implemented so that it calls the `_get_tenant_hosts()` method to retrieve the list of compute nodes configured for the tenant that owns the virtual machine request. The hosts are then filtered so that only those host that are in an active state are considered.

The `select_destinations()` method calls `_filter_hosts()` to retrieve the list of active hosts and then selects one of these hosts at random as the destination for the virtual machine. The `select_destinations()` method includes code that logs debug information to the `scheduler.log` file. This logging information can be used to verify that the TenantScheduler scheduler is configured and activated correctly.

## 8.3 Deployment

Deployment of the scheduler required setting up a multi-node OpenStack cloud environment. For this DevStack[40], a developer oriented version of OpenStack was used. DevStack provides a scripted install of OpenStack services using the latest source code from the OpenStack Git repository. A typical deployment using DevStack installs all OpenStack services on to a single machine. In order to observe the scheduler in action making meaningful placement decisions, it was necessary to create an environment with multiple compute nodes. Using Oracle VirtualBox it was possible to create a virtual environment that could be run on a standard laptop with 16 GB of memory. The environment contained the following virtual machines:

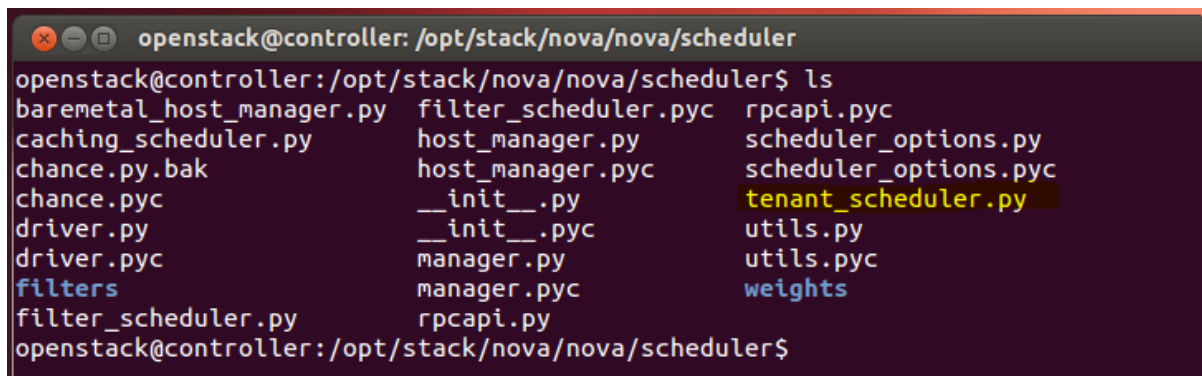
Virtual Machine Name	Description
<b>controller</b>	The controller node that runs the following services: Compute, Neutron, Glance, Nova and Keystone
<b>compute1</b>	A compute node that runs the Nova compute service.
<b>compute2</b>	A compute node that runs the Nova compute service.
<b>compute3</b>	A compute node that runs the Nova compute service.

Deploying the scheduler to the OpenStack environment is a three-step process:

1. Deploy the tenant\_scheduler.py file.
2. Change the scheduler\_driver property on the nova.conf file
3. Restart Nova Compute service which loads the new driver.

### 8.3.1 Location

The tenant\_scheduler.py file is copied to the “/opt/stack/nova/scheduler” directory. This is the location that Nova expects to find all available scheduler implementations. As can be seen from Figure 24 this directory stores the implementation for the default OpenStack schedulers, ChanceScheduler (chance.py) and FilterScheduler (filter.py).

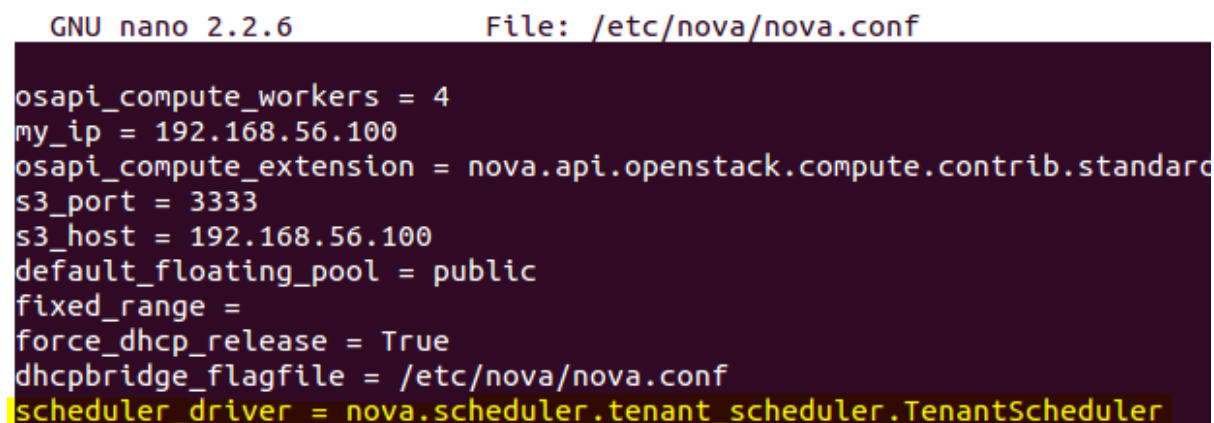


```
openstack@controller: /opt/stack/nova/nova/scheduler
openstack@controller:/opt/stack/nova/nova/scheduler$ ls
baremetal_host_manager.py  filter_scheduler.pyc  rpcapi.pyc
caching_scheduler.py      host_manager.py      scheduler_options.py
chance.py.bak             host_manager.pyc     scheduler_options.pyc
chance.pyc                __init__.py          tenant_scheduler.py
driver.py                  __init__.pyc         utils.py
driver.pyc                 manager.py            utils.pyc
filters                    manager.pyc           weights
filter_scheduler.py        rpcapi.py
```

Figure 24 Deployment location of TenantScheduler.py

### 8.3.2 Service Configuration

The Nova compute service is configured via the “/etc/nova/nova.conf” configuration file. When the Nova Compute service starts it uses the setting for scheduler\_driver in nova.conf to load the appropriate scheduler implementation. In this case, the scheduler implementation is TenantScheduler. The scheduler\_driver property is set to scheduler\_driver = nova.scheduler.tenant\_scheduler.TenantScheduler.



```
GNU nano 2.2.6      File: /etc/nova/nova.conf
osapi_compute_workers = 4
my_ip = 192.168.56.100
osapi_compute_extension = nova.api.openstack.compute.contrib.standard
s3_port = 3333
s3_host = 192.168.56.100
default_floating_pool = public
fixed_range =
force_dhcp_release = True
dhcpbridge_flagfile = /etc/nova/nova.conf
scheduler_driver = nova.scheduler.tenant_scheduler.TenantScheduler
```

Figure 25 Changing nova.conf to use TenantScheduler

### 8.3.3 Restart Nova Scheduler Service

The Nova Scheduler service needs to be restarted to force the service to reinitialize the scheduler component.

We can restart the service using the following steps:

- screen -r stack
- Press **Ctrl+A** followed by **N** until you reach the `n-sch` screen.
- Press **Ctrl+C** to kill the service.
- Press Up Arrow to bring up the last command.
- Press Enter to run it.

### 8.3.4 Testing

To verify that the scheduler instance being used by the OpenStack environment is the TenantScheduler, logging output from Nova scheduler process was monitored as new virtual machine instance requests were submitted for processing. In Figure 26, the custom logging messages that are specific to the TenantScheduler are highlighted. This illustrates that the TenantScheduler implementation is used by the Nova scheduler to schedule new virtual machine instances. It is also possible to tell from the logging output that in this instance the “controller” node is the only hosts being considered for the tenant with id = ‘b743efa5635e47bebf639b3c3d0b4ab7’. The other nodes ‘compute1’ and ‘compute2’ are excluded from consideration. This corresponds to the behaviour we expect based on the tenants.conf configuration file defined in Figure 20.

```

nova/nova/drivers/db.py:71
2015-03-01 17:14:07.879 DEBUG nova.scheduler.tenant_scheduler [req-8ff6ebc-e095-4a44-ba9e-117bd3804a7
8 admin demo] Tenant ID: b743efa5635e47bebf639b3c3d0b4ab7 _filter_hosts /opt/stack/nova/nova/scheduler
/tenant_scheduler.py:44
2015-03-01 17:14:07.880 DEBUG nova.scheduler.tenant_scheduler [req-8ff6ebc-e095-4a44-ba9e-117bd3804a7
8 admin demo] All available hosts: [u'controller', u'compute1', u'compute2'] _filter_hosts /opt/stack/
nova/nova/scheduler/tenant_scheduler.py:45
2015-03-01 17:14:07.881 DEBUG nova.scheduler.tenant_scheduler [req-8ff6ebc-e095-4a44-ba9e-117bd3804a7
8 admin demo] Configured hosts for this tenant: ['controller'] _filter_hosts /opt/stack/nova/nova/sche
duler/tenant_scheduler.py:48
2015-03-01 17:14:07.882 DEBUG nova.scheduler.tenant_scheduler [req-8ff6ebc-e095-4a44-ba9e-117bd3804a7
8 admin demo] Filter out tenant specific hosts: [u'controller'] _filter_hosts /opt/stack/nova/nova/sch
eduler/tenant_scheduler.py:51
2015-03-01 17:14:07.891 DEBUG nova.scheduler.tenant_scheduler [req-8ff6ebc-e095-4a44-ba9e-117bd3804a7
8 admin demo] Filter out ignored hosts: [u'controller'] _filter_hosts /opt/stack/nova/nova/scheduler/t
enant_scheduler.py:55
2015-03-01 17:14:08.904 29476 INFO oslo.messaging._drivers.impl_rabbit [-] Connected to AMQP server on
192.168.56.100:5672
2015-03-01 17:14:10.285 DEBUG nova.openstack.common.periodic_task [req-6a5a94d6-14ea-40b8-b9bc-fb21cb6
a34d9 None None] Running periodic task SchedulerManager._expire_reservations run_periodic_tasks /opt/s
tack/nova/nova/openstack/common/periodic_task.py:178
2015-03-01 17:14:10.325 DEBUG nova.openstack.common.periodic_task [req-6a5a94d6-14ea-40b8-b9bc-fb21cb6
a34d9 None None] Running periodic task SchedulerManager._run_periodic_tasks run_periodic_tasks /opt/st
ack/nova/nova/openstack/common/periodic_task.py:178
2015-03-01 17:14:10.326 29476 DEBUG nova.openstack.common.loopingcall [-] Dynamic looping call sleepin
g for 60.00 seconds _inner /opt/stack/nova/nova/openstack/common/loopingcall.py:132

14$(L) n-sch* 15$(L) n-novnc 16$(L) n-xvnc 17$(L) n-cauth 18$(L) n-obj 19$(L) c-api 20$(L) c-sc

```

Figure 26 logging from Nova Scheduler Service (TenantScheduler specific log items highlighted)

## 9 Summary and Conclusions

One of the primary functions of an IaaS solution is the efficient management of resources. While the scheduler is a relatively small component within the overall resource management system, it plays a significant role in the smooth and efficient operation of a cloud environment. The scheduling component is of strategic importance in fulfilling management objectives set out by the cloud service provider. The decisions made by the scheduler directly impact on areas such as power and cooling, two major costs in operating a modern datacenter.

Not all scheduling algorithms are the same. As can be seen from this report, the default OpenStack schedulers have significant shortcomings and there are question marks over their ability to meet the expectations of a modern-day cloud service provider. The static behaviour of the built in OpenStack schedulers stands in stark contrast to the dynamic nature of a cloud environment where resources are expected to quickly scale up and down in response to fluctuations in user demand.

There are other areas in the context of resource management that impact on performance and efficiency. One of these areas is the sizing of virtual machines. As can be seen in this report, the sizing decisions for each virtual machine flavour should be made with careful consideration to the physical machine that will host the virtual machine instance. Virtual machine flavours should be sized proportionally in order to pack the optimal number of virtual machines on to each physical host. The cloud service provider needs to examine closely their strategy for sizing virtual machine flavours and the corresponding specifications of these physical machines.

Public cloud service providers will continue to squeeze greater efficiencies from their operations and to raise the bar in terms of reliability and availability. Competition from the big industry players such as Google, Microsoft and Amazon and their ability to use economies of scale will make public cloud offerings more attractive from a pricing perspective. Private cloud is and will continue to be a more expensive proposition than public cloud which is all the more reason why private cloud should strive to be as efficient as possible. Private cloud platforms need to be conscious of current trends such as energy efficiency and the “greening” of the datacenter and incorporate these into their resource management systems to further drive down operating costs. Expanding the levels of intelligence within the resource management system could play a major role in this regard.

OpenStack is a relatively young project and the jury is still out on its potential to be a major force in the cloud arena. The OpenStack marketing team has done an excellent job of generating an awareness of the OpenStack brand and in creating momentum within the developer community. However, analysts like Gartner believe that the quality of the product still has some way to go to meet market expectations. In the author’s opinion, installation of OpenStack is still a complicated task and diagnosing problems can be a frustrating experience when the process goes wrong.

On a positive note, the extensible architecture of OpenStack provides customers with an alternative when a built in component like the scheduling algorithm does not meet their expectations. This architectural decision is quite clever because it creates benefits for OpenStack, for third party developers and for users of OpenStack itself. For organisations interested in creating new drivers for OpenStack, it can potentially unlock new revenue streams. For OpenStack it provides its end-users with options to fill gaps in functionality. Overall, this architectural decision makes the OpenStack platform stronger because it allows difficult problems like scheduling to be tackled by experts in that domain. A company like IBM with over 60 years of main frame experience has in-depth knowledge of problems in the area of scheduling that are still relevant in cloud environments today. The IBM Platform Resource Scheduler (PRS) is an excellent example of how IBM is using its vast knowledge in scheduling to create a new market for its services while enhancing the capabilities of OpenStack. The very existence of the PRS product is in itself evidence that IBM views the capabilities of the built in OpenStack scheduling algorithms as not being enterprise ready.

The only constant is change and in a cloud datacenter this is certainly the case. The current OpenStack scheduler is deeply lacking in intelligence and its inability to revisit placement decisions is a major shortcoming. The virtual machine provisioning process would benefit from a more cohesive view of overall resource usage within the OpenStack cloud. This, in conjunction with an agent with similar functionality to VMWare's Dynamic Resource Scheduler (DRS) and a policy based approach to resource management would enable system load to be monitored and adjusted to meet management objectives. A modern scheduling component requires this capability to revisit placement decisions thus ensuring that management objectives are being upheld and that the cloud environment is balancing efficiency with performance. Managing resources in a large scale public or private cloud is challenging. [41] The OpenStack Foundation needs to review their current scheduler offering if they are to achieve their goal to "produce the ubiquitous Open Source cloud computing platform that will meet the needs of public and private cloud providers regardless of size, by being simple to implement and massively scalable."

## 10 Cited References

- [1] RightScale, “2015 State of the Cloud Report,” 2015.
- [2] P. Mell and T. Grance, “The NIST Definition of Cloud Computing Recommendations of the National Institute of Standards and Technology.”
- [3] “Consumerization - Gartner IT Glossary.” [Online]. Available: <http://www.gartner.com/it-glossary/consumerization>. [Accessed: 02-Mar-2015].
- [4] Information Systems Audit and Control Association (ISACA), “Essential characteristics of Cloud Computing.” [Online]. Available: [http://www.isaca.org/Groups/Professional-English/cloud-computing/GroupDocuments/Essential characteristics of Cloud Computing.pdf](http://www.isaca.org/Groups/Professional-English/cloud-computing/GroupDocuments/Essential%20characteristics%20of%20Cloud%20Computing.pdf).
- [5] R. Miller, “Google Has Spent \$21 Billion on Data Centers | Data Center Knowledge,” 2013. [Online]. Available: <http://www.datacenterknowledge.com/archives/2013/09/17/google-has-spent-21-billion-on-data-centers/>. [Accessed: 26-Jan-2015].
- [6] Microsoft Corporation, “Microsoft Cloud Infrastructure Datacenter and Network Factsheet.” [Online]. Available: [file:///C:/Users/jbrennan/Downloads/Microsoft\\_Cloud\\_Infrastructure\\_Datacenter\\_and\\_Network\\_Fact\\_Sheet.pdf](file:///C:/Users/jbrennan/Downloads/Microsoft_Cloud_Infrastructure_Datacenter_and_Network_Fact_Sheet.pdf). [Accessed: 02-Mar-2015].
- [7] Cloud Standards Customer Council (CSCC), “Cloud Computing Use Cases v.1.0,” vol. 0, no. c, p. 27, 2011.
- [8] M. Behrendt, B. Glasner, P. Kopp, R. Dieckmann, G. Breiter, S. Pappe, H. Kreger, and a Arsanjani, “Cloud Computing Reference Architecture v2.0,” *Draft Version V*, vol. 1, 2011.
- [9] OpenStack, “OpenStack Open Source Cloud Computing Software,” 2015. [Online]. Available: <http://www.openstack.org/software/>.
- [10] OpenStack Foundation, “Governance Foundation Mission,” 2015. [Online]. Available: <https://wiki.openstack.org/wiki/Governance/Foundation/Mission>.
- [11] J. Artymiak and L. Namphy, “OpenStack Technology Breaking the Enterprise Barrier.”
- [12] The OpenStack Foundation, “Companies Supporting the OpenStack Foundation,” 2015. [Online]. Available: [https://wiki.openstack.org/wiki/Governance/Foundation/Structure#Platinum\\_Members](https://wiki.openstack.org/wiki/Governance/Foundation/Structure#Platinum_Members).



- [13] "OpenStack Technical Committee Charter — OpenStack Governance 2014." [Online]. Available: <http://governance.openstack.org/reference/charter.html#election-for-tc-seats>. [Accessed: 22-Jan-2015].
- [14] J. Lifflander, A. McDonald, and O. Pilskalns, "Clustering Versus Shared Nothing: A Case Study," *Proc. - Int. Comput. Softw. Appl. Conf.*, vol. 2, pp. 116–121, 2009.
- [15] A. Rodriguez, "Restful web services: The basics," *Online Artic. IBM Dev. Tech. Libr.*, pp. 1–11, 2008.
- [16] The OpenStack Foundation, "Welcome to Keystone, the OpenStack Identity Service! — keystone." [Online]. Available: <http://docs.openstack.org/developer/keystone/>. [Accessed: 22-Jan-2015].
- [17] The OpenStack Foundation, "OpenStack Glance documentation," 2015. [Online]. Available: <http://docs.openstack.org/developer/glance/>. [Accessed: 22-Jan-2015].
- [18] OpenStack.org, "OpenStack Networking » OpenStack Open Source Cloud Computing Software." [Online]. Available: <http://www.openstack.org/software/openstack-networking/>. [Accessed: 23-Jan-2015].
- [19] "Cinder - OpenStack." [Online]. Available: <https://wiki.openstack.org/wiki/Cinder#Description>. [Accessed: 23-Jan-2015].
- [20] A. Juneja, "@WalmartLabs » Why we chose OpenStack for Walmart Global eCommerce," 2015. [Online]. Available: [http://www.walmartlabs.com/2015/02/17/why-we-chose-openstack-for-walmart-global-e-commerce/?awesm=awe.sm\\_jM31y](http://www.walmartlabs.com/2015/02/17/why-we-chose-openstack-for-walmart-global-e-commerce/?awesm=awe.sm_jM31y). [Accessed: 25-Feb-2015].
- [21] F. Wuhib, R. Stadler, and H. Lindgren, "Dynamic Resource Allocation with Management Objectives — Implementation for an OpenStack Cloud," pp. 309–315, 2012.
- [22] J. Hamilton, "Cooperative Expendable Micro-Slice Servers ( CEMS ): Low Cost , Low Power Servers for Internet-Scale Services," *Power*, pp. 1–8, 2009.
- [23] L. Shi and B. Butler, "Provisioning of requests for virtual machine sets with placement constraints in IaaS clouds," in *IFIP/IEEE International Symposium on Integrated Network Management (IM 2013)*, 2013, pp. 499–505.
- [24] E. Bin, O. Biran, O. Boni, E. Hadad, E. K. Kolodner, Y. Moatti, and D. H. Lorenz, "Guaranteeing high availability goals for virtual machine placement," in *Proceedings - International Conference on Distributed Computing Systems*, 2011, pp. 700–709.
- [25] Wikipedia, "Bin Packing Problem," 2015. [Online]. Available: [http://en.wikipedia.org/wiki/Bin\\_packing\\_problem](http://en.wikipedia.org/wiki/Bin_packing_problem).

- [26] K. Basil and S. Cohen, "Deterministic capacity planning for OpenStack - Red Hat Summit," 2014. .
- [27] B. Jennings and R. Stadler, "Resource Management in Clouds : Survey and Research Challenges," *J. Netw. Syst. Manag.*, 2013.
- [28] D. Bonde, "Techniques for Virtual Machine Placement in Clouds," *Design*, 2010.
- [29] Rackspace, "OpenStack Nova-Scheduler And vSphere DRS," 2013. [Online]. Available: <http://www.rackspace.com/blog/openstack-nova-scheduler-and-vsphere-drs/>. [Accessed: 05-Mar-2015].
- [30] American Mathematical Society, "Bin Packing." [Online]. Available: <http://www.ams.org/samplings/feature-column/fcarc-bins2>. [Accessed: 05-Mar-2015].
- [31] The OpenStack Foundation, "Nova Concepts and Introduction — nova 2012.1.2-dev documentation." [Online]. Available: <http://docs.openstack.org/developer/nova/nova.concepts.html>. [Accessed: 22-Feb-2015].
- [32] M. C. dos P Andrade, T Bell, J van Eldik, G McCance, B Panzer-Steindel and S. T. and U. S. Santos, "Review of CERN Data Centre Infrastructure," *European Organization for Nuclear Research (CERN)*, 2012. [Online]. Available: [http://cds.cern.ch/record/1457989/files/chep 2012 CERN infrastructure final.pdf?version=1](http://cds.cern.ch/record/1457989/files/chep%202012%20CERN%20infrastructure%20final.pdf?version=1). [Accessed: 12-Mar-2015].
- [33] "Rackspace and Cern update OpenStack with federated cloud support - IT News from V3.co.uk." [Online]. Available: <http://www.v3.co.uk/v3-uk/news/2353097/rackspace-and-cern-update-openstack-with-federated-cloud-support>. [Accessed: 08-Feb-2015].
- [34] Oracle Corporation, "Introduction to the Scheduler (System Administration Guide: Oracle Solaris Containers-Resource Management and Oracle Solaris Zones)." [Online]. Available: <http://docs.oracle.com/cd/E19455-01/817-1592/rmfss-2/index.html>. [Accessed: 11-Mar-2015].
- [35] P. G. Capek, S. P. Frank, S. Gerd, and D. Shields, "A history of IBM's open-source involvement and strategy," *IBM Syst. J.*, vol. 44, no. 2, pp. 249–257, 2005.
- [36] G. Sissons, "Addressing OpenStack's Goldilocks Dilemma | Gord Sissons." [Online]. Available: <http://sissons.ca/?p=454>. [Accessed: 26-Jan-2015].
- [37] Linux-KVM.org, "Migration - KVM." [Online]. Available: <http://www.linux-kvm.org/page/Migration>. [Accessed: 11-Feb-2015].
- [38] "IBM Knowledge Center - Resource optimization global based policy." 01-Jan-2013.

- [39] D. Bader, "Abstract Base Classes in Python – dbader.org." [Online]. Available: <http://dbader.org/blog/abstract-base-classes-in-python>. [Accessed: 17-Feb-2015].
- [40] OpenStack.org, "DevStack - OpenStack." [Online]. Available: <https://wiki.openstack.org/wiki/DevStack>. [Accessed: 09-Mar-2015].
- [41] A. Gulati, I. Ahmad, and A. Holler, "Cloud-Scale Resource Management : Challenges and Techniques," in *Cloud-Scale Resource Management : Challenges and Techniques*, 2011.

## Appendix 1: TenantScheduler Source Code

```
# File: TenantScheduler.py
# Author: John Brennan
# Date: 02-MAR-2015
#
# Description: TenantScheduler reads a configuration file that defines the OpenStack compute nodes
#              that are assigned to a specific tenant. This scheduler reuses some of the ChanceScheduler
#              implementation. The main differences are the _get_tenant_hosts() and the _filter_hosts()
#              methods.
#
"""
Tenant Scheduler implementation
"""
import random

from oslo.config import cfg
from nova.compute import rpcapi as compute_rpcapi
from nova import exception
from nova.openstack.common.gettextutils import _
from nova.scheduler import driver
from nova.openstack.common import log as logging

CONF = cfg.CONF
CONF.import_opt('compute_topic', 'nova.compute.rpcapi')
LOG = logging.getLogger(__name__)

class TenantScheduler(driver.Scheduler):
    """Implements Scheduler as a random node selector."""

    def __init__(self, *args, **kwargs):
        super(TenantScheduler, self).__init__(*args, **kwargs)
        self.compute_rpcapi = compute_rpcapi.ComputeAPI()

    def _get_tenant_hosts(self, tenantId):
        """Returns list of hosts associated with a specific tenant as per tenants.conf file."""
        with open('/home/openstack/devstack/tenants.conf') as configFile:
            line = [line.strip().split() for line in configFile.readlines()]
            tenantDictionary = {d[0]: d[1:] for d in line}

            hosts = tenantDictionary.get(tenantId, None)

            if hosts is None:
                raise Exception('No entry found in the TenantScheduler config file for this tenant.')

            return hosts

    def _schedule(self, context, topic, request_spec, filter_properties):
        """Picks a host at random from list of available hosts configured for this tenant."""

        elevated = context.elevated()
        hosts = self.hosts_up(elevated, topic)
        if not hosts:
            msg = _("Is the appropriate service running?")
            raise exception.NoValidHost(reason=msg)

        hosts = self._filter_hosts(request_spec, hosts, filter_properties)
        if not hosts:
            msg = _("Could not find another compute")
            raise exception.NoValidHost(reason=msg)

        return random.choice(hosts)

    def select_destinations(self, context, request_spec, filter_properties):
        """Selects random destinations."""
        num_instances = request_spec['num_instances']
        dests = []
        for i in range(num_instances):
            host = self._schedule(context, CONF.compute_topic,
                                  request_spec, filter_properties)
            host_state = dict(host=host, nodename=None, limits=None)
            dests.append(host_state)

        if len(dests) < num_instances:
            raise exception.NoValidHost(reason='')
        return dests

    def schedule_run_instance(self, context, request_spec,
                              admin_password, injected_files,
                              requested_networks, is_first_time,
                              filter_properties, legacy_bdm_in_spec):
        """Create and run an instance or instances."""
        instance_uuids = request_spec.get('instance_uuids')
        for num, instance_uuid in enumerate(instance_uuids):
            request_spec['instance_properties']['launch_index'] = num
            try:
                host = self._schedule(context, CONF.compute_topic,
                                      request_spec, filter_properties)
                updated_instance = driver.instance_update_db(context,
                                                              instance_uuid)
                self.compute_rpcapi.run_instance(context,
                                                  instance=updated_instance, host=host,
                                                  requested_networks=requested_networks,
                                                  injected_files=injected_files,
                                                  admin_password=admin_password,
                                                  is_first_time=is_first_time,
                                                  request_spec=request_spec,
                                                  filter_properties=filter_properties,
                                                  legacy_bdm_in_spec=legacy_bdm_in_spec)
            except Exception as ex:
                driver.handle_schedule_error(context, ex, instance_uuid, request_spec)
```

## Appendix 2: Issues Encountered during this Project

The number one issue on this research project was the installation of OpenStack itself. While this may sound like a trivial task, it was far from it and consumed a significant amount of the researcher's time. Having a fully functioning OpenStack environment was essential from a learning perspective so that the various features of the product could be explored. It was also required to enable the development and testing of the custom scheduler component, TenantScheduler.

### Installing OpenStack on Multiple Nodes

It is relatively straightforward to configure OpenStack to run all services on a single physical machine. Books like the OpenStack Cloud Computing Cookbook provide detailed documentation on the steps involved. To adequately inspect the scheduling components, it was necessary to configure an OpenStack environment with multiple compute nodes. This is known as a multi-node install. At least one instance of the Nova Compute service is necessary to host a virtual machine instance. However, configuring a single node installation of OpenStack environment would not provide a very interesting environment in which to observe and configure scheduler behaviour. After all, with only a single compute node available the scheduler would end up choosing the same candidate to host virtual machines for every new virtual machine request.

Configuring a real-world multi-node OpenStack installation required a significant investment in hardware but the budget for this did not exist. The researcher instead upgraded a laptop and used Oracle VirtualBox virtualisation software to create a virtual environment that would host multiple virtual machines. Each of these virtual machines ran Ubuntu Linux and were configured to run various OpenStack services. Four virtual machines were created in total, a single OpenStack controller and three separate compute nodes. The installation process was very time consuming as services need to be installed and manually configured on each node. By far, the biggest problem was getting the compute nodes and the controller node to communicate with each other. The only way to know if the installation process had worked was to try to provision a new virtual machine instance via the Horizon dashboard. Getting to the stage where this provisioning test could be attempted usually took several hours. Invariably, the request to create the virtual machine would be received by the controller node, processed but no virtual machine would be provisioned on one of the available compute nodes. The problem was that the controller was unable to see the compute nodes that had been configured for the multi-node installation.

Many developers provided scripts via GitHub.com that claimed to script a multi-node OpenStack install. These scripts typically used popular cloud provisioning and deployment tools like Puppet and Vagrant. Again, these never quite worked and there was usually a failure in a step that caused the overall installation to be unsuccessful. Attempts to find solutions to these failures consumed many hours of the researchers time.

In the end, the researcher discovered DevStack which is a scripted install of OpenStack provided by OpenStack.org. DevStack is not intended for commercial use and is instead targeted at software developers and testers. It allows an OpenStack development environment to be setup by executing a single script “stack.sh” from the command line. The OpenStack documentation provides a guide to creating a multi-node DevStack installation (<http://docs.openstack.org/developer/devstack/guides/multinode-lab.html>) which finally enabled the researcher to install OpenStack on a controller node and three compute nodes.

## Multi-Node Lab

---

Here is OpenStack in a realistic test configuration with multiple physical servers.

### Prerequisites Linux & Network

---

#### Minimal Install

---

You need to have a system with a fresh install of Linux. You can download the [Minimal CD](#) for Ubuntu releases since DevStack will download & install all the additional dependencies. The netinstall ISO is available for [Fedora](#) and [CentOS/RHEL](#).

Install a couple of packages to bootstrap configuration:

```
apt-get install -y git sudo || yum install -y git sudo
```

Figure 27 Screen shot of Multinode DevStack Install

### Deployment of Custom Scheduler

During the process of creating the custom scheduler for OpenStack there was a question over how the scheduler would be deployed. The researcher is used to working with code that is compiled into a binary and is subsequently deployed to a pre-determined location on a server. In this case the Python code used to implement the scheduler does not need to be compiled and is interpreted at runtime. Determining the location to deploy the python file to was not immediately obvious. In the end the research reached out to the OpenStack community via <https://ask.openstack.org/en/question/52506/how-do-i-deploy-a-custom-scheduler-in-openstack/> to get an answer to this question.

It turns out that the deployment locations vary between DevStack (developer version of OpenStack) and a regular OpenStack install (production version).

For DevStack the python file must be deployed in:

“/opt/stack/nova/nova/scheduler”

For OpenStack, the python file must be deployed in:

“/usr/lib/python2.7/sites-packages/nova/scheduler”



## How do I deploy a custom scheduler in Openstack

nova

scheduler

customize

deployment



I've read Chapter 15 (Customization) in the OpenStack Operations Guide which walks through the process of creating a customer scheduler called IPScheduler and deploying it in DevStack. I managed to follow the instructions in this document successfully.

asked Nov 3 '14



Newberry

1 1 1 2

In DevStack it was a case of:

1. Creating the python file `ip_scheduler.py` that contains implementation of customer scheduler in `/opt/stack/nova/nova/scheduler`.
2. Editing the `/etc/nova/nova.conf` file to use the new customer scheduler
3. Restart the nova scheduler service to enable new configuration to take effect.

What I would like to do now is to deploy the scheduler in my test Openstack environment but I'm not sure how to do this as there is no `/opt/stack/nova/nova/scheduler` directory to deploy my custom scheduler file too. Also I'm not sure if the custom scheduler needs to be compiled into a binary or if it can be copied as is.

If anyone has any tips, advice or articles they can point me to on this I'd be really grateful.

Regards, John

### Comments

- 1 The document was assumed that you use devstack to deploy openstack cloud. Did you use devstack for your test environment setup?

Mzoorikh (Nov 3 '14)

Hi Mzoorikh, I have 2 environments configured, the first is a devstack environment which was created using the Multinode Lab guide from <http://devstack.org/guides/multinode-lab.html> (<http://devstack.org/guides/multinode-lab.html>). My second test environment is using Ubuntu Openstack, not devstack.

Newberry (Nov 3 '14)

Figure 28 Extract of the researcher's question posted on OpenStack.org