# Cloud-Scale Resource Management: Challenges and Techniques

*Ajay Gulati*
*VMware, Inc.*
*agulati@vmware.com*

*Ganesha Shanmuganathan*
*VMware, Inc.*
*sganesh@vmware.com*

*Anne Holler*
*VMware, Inc.*
*anne@vmware.com*

*Irfan Ahmad*
*VMware, Inc.*
*irfan@vmware.com*

## Abstract

Managing resources at large scale while providing performance isolation and efficient use of underlying hardware is a key challenge for any cloud management software. Most virtual machine (VM) resource management systems like VMware DRS clusters, Microsoft PRO and Eucalyptus, do not currently scale to the number of hosts and VMs supported by cloud service providers. In addition to *scale*, other challenges include heterogeneity of systems, compatibility constraints between virtual machines and underlying hardware, islands of resources created due to storage and network connectivity and limited scale of storage resources.

In this paper, we shed light on some of the key issues in building cloud-scale resource management systems, based on five years of research and shipping cluster resource management products. Furthermore, we discuss various techniques to provide large scale resource management, along with the pros and cons of each technique. We hope to motivate future research in this area to develop practical solutions to these issues.

## 1 Introduction

Managing compute and IO resources at large scale in both public and private clouds is quite challenging. The success of any cloud management software critically depends on the flexibility, scale and efficiency with which it can utilize the underlying hardware resources while providing necessary performance isolation [5, 8, 11]. Customers expect cloud service providers to deliver quality of service (QoS) controls for tenant VMs. Thus, resource management at cloud scale requires the management platform to provide a rich set of resource controls that balance the QoS of tenants with overall resource efficiencies of datacenters.

For public clouds, some systems (e.g., Amazon EC2) provide largely a one-to-one mapping between virtual and physical CPU and memory resources. This leads to poor consolidation ratios and customers are unable to exploit the benefits from statistical multiplexing that they enjoy in private clouds. Some recent studies [4] have done a cost comparison of public vs. private clouds and have come up with a mantra of *don't move to the cloud, but virtualize*.

For private clouds, resource management solutions like VMware DRS [2] and Microsoft PRO [1] have led to better performance isolation, higher utilization of underlying hardware resources via over-commitment and overall lower cost of ownership. Unfortunately, the rich set of controls provided by these solutions does not easily scale to cloud environments. As part of a team, we have worked on VMware's shipping Distributed Resource Scheduler (DRS) product and are in the process of prototyping and evaluating a scaled version of DRS. In doing so, we have faced a set of challenges and have explored various alternatives for solving them.

In this position paper, we first describe a resource model that provides a rich but needed set of resource controls for supporting high consolidation ratios, service guarantees, and performance isolation. We also outline how these controls are enforced while providing efficient utilization of underlying resources in our current small scale implementation. Then, we discuss some of the key issues and challenges in scaling and designing a resource management solution for a cloud environment. These issues include scale, heterogeneity of resources, handling compatibility constraints and resource islands due to storage and network connectivity.

Finally, we discuss three different approaches to designing a scalable solution along with pros and cons of each. These approaches include hierarchical-scaling, flat-scaling and statistical-scaling. Our hope is that this paper will motivate future research in this area to tackle some of these relevant issues in designing practical resource management solutions.

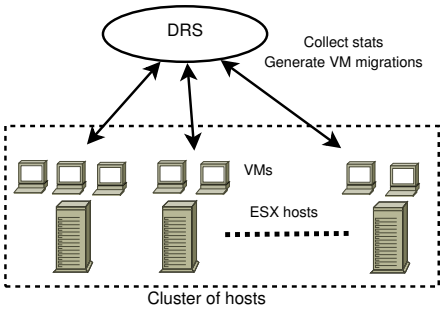


Figure 1: DRS Cluster Architecture

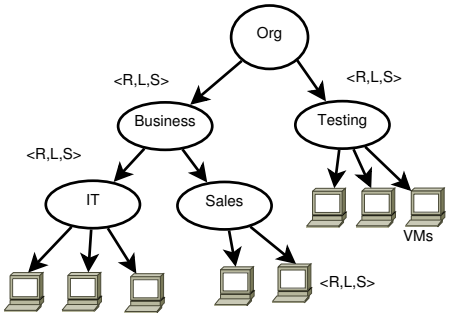


Figure 2: Resource pool tree. R,L and S denote reservation, limit and share values respectively.

## 2 Resource Management Operations

There are many ways to provide resource management controls in a cloud environment. We pick VMware's Distributed Resource Scheduler (DRS) as an example because it has a rich set of controls that are needed for efficient multi-resource management while providing differentiated QoS to groups of VMs, albeit at a small scale compared to typical cloud deployments. Currently DRS supports 32 hosts and roughly 3000 VMs in a management domain called a cluster, as shown in Figure 1. In this section, we first describe the resource management services provided by DRS, then discuss the importance of such services in the cloud setting, and finally provide a brief description of load balancing approach used in DRS.

In a large scale environment, having rich resource management controls alleviates the noisy-neighbor problem for tenants if, like DRS, the underlying management infrastructure natively supports automated enforcement and guarantees. At the same time, such controls support the cloud service provider over-committing the hardware resources safely, allowing better efficiency from statistical multiplexing of resources without sacrificing the exposed guarantees.

### 2.1 Basic Resource Controls in DRS

VMware ESX and DRS provide resource controls which allow administrators and users to express allocations in terms of either absolute VM *service rates* or relative VM *importance*. The same control knobs are provided for cpu and memory allocations at both the host and cluster levels. Similar controls are under development for I/O resources and have been validated by a research prototype [7]. Note that VMware's Distributed Power Management product (DPM) powers on/off hosts while respecting these controls.

**Reservation:** A reservation is used to specify a minimum guaranteed amount of resources, i.e., a lower bound that applies even when a system is heavily over-committed. Reservations are expressed in absolute units, such as megahertz (MHz) for cpu, and megabytes (MB) for memory. Admission control during VM power on ensures that the sum of reservations for a resource does not exceed total capacity.

**Limit:** A limit is used to specify an upper bound on consumption, even when a system is under-committed. A VM is prevented from consuming more than its limit, even if that leaves some resources idle. Like reservations, limits are expressed in concrete absolute units, such as MHz and MB.

**Shares:** Shares are used to specify relative importance, and are expressed as abstract numeric values. A VM is entitled to consume resources proportional to its share allocation; it is guaranteed a minimum resource fraction equal to its fraction of the total shares in the system. Shares represent relative resource rights that depend on the total number of shares contending for a resource.

Reservations and limits play an important role in the cloud. Without these guarantees, users would suffer from performance unpredictability, unless the cloud provider takes the non-work-conserving approach of simply statically partitioning the physical hardware, which leads to the inefficiency of over-provisioning. This was the main reason for providing these controls for enterprise workloads running on top of VMware ESX and VMware DRS.

### 2.2 Resource Pools

In addition to the basic resource controls presented earlier, administrators and users can specify flexible resource management policies for groups of VMs. This is facilitated by introducing the concept of a logical *resource pool* – a container that can be used to specify an aggregate resource allocation for a set of VMs. A resource pool is a named object with associated settings for each managed resource – the same familiar shares, reservation, and limit controls used for VMs. Admission control is performed at the pool level; the sum of the reservations for a pool's

children must not exceed the pool's own reservation.

Separate, per-pool allocations provide both isolation between pools, and sharing within pools. For example, if some VMs within a pool are idle, their unused allocation will be reallocated preferentially to other VMs within the same pool. Resource pools may be configured in a flexible hierarchical organization as shown in Figure 2; each pool has an enclosing parent pool, and children that may be VMs or sub-pools. Resource pools are useful in dividing large capacity into logically grouped users. Organizational administrators can use resource pool hierarchies to mirror human organizational structures, and to support delegated administration.

The resource pool construct is well-suited to the cloud setting as organizational administrators typically buy capacity in bulk from providers and run several VMs. It is missing in several large scale cloud providers although it has been useful for thousands of enterprise customers using VMware DRS and VMware ESX, each managing the resource needs of thousands of VMs.

## 2.3 DRS Load Balancing

DRS performs three key resource-related operations: (1) It computes the amount of resources that each VM should get based on the reservation, limit and shares settings for VMs as well as resource pool nodes, (2) It does initial placement of VMs on to hosts, so that a user doesn't have to make a manual placement decision, and (3) It recommends and performs live VM migrations to do load balancing across hosts in a dynamic environment where the VMs' resource demands may change over a period of time.

DRS manages a cluster of distributed hosts, providing the illusion that the entire cluster is a single huge "uber-host" with the aggregate capacity of all individual hosts. To implement this illusion, DRS breaks up the user-specified resource pool hierarchy into per host resource pool hierarchies with appropriate host-level resource pool settings. Once the VMs are placed on a host, the local schedulers on each ESX host allocate resources to VMs fairly based on host-level resource pool and VM resource settings. DRS is invoked every 5 minutes by default, but can also be invoked on demand.

To describe DRS load-balancing, it is important to first clarify what DRS uses as its load metric. In particular, it does not use host utilization. In DRS, load reflects VM importance, as captured by the concept of *dynamic entitlement*. Dynamic entitlement is computed based on the resource controls and actual demand for CPU and memory resources for each VM. The entitlement is higher than the reservation and lower than the limit; its actual value depends on the cluster capacity and total demand. Dynamic entitlement is equivalent to demand when the demands of all the VMs in the cluster can be met, else it is a scaled-down demand value with the scaling dependent on cluster capacity, the demands of other VMs, the VM's place in the resource pool hierarchy, and its shares, reservation and limit. Dynamic entitlement is computed using a pass over the resource pool hierarchy tree to allocate to all VMs and resource pools their cpu and memory reservations and to constrain their demand by their limits, followed by another pass over the tree to allocate spare resources to address limit-constrained demand above reservation in accordance with the associated share values.

DRS currently uses *normalized entitlement* as its core per-host load metric. For a host $h$, normalized entitlement $N_h$ is defined as the sum of the per-VM entitlements $E_i$ for all VMs running on $h$, divided by the host capacity $C_h$ available to VMs: $N_h = \frac{\sum E_i}{C_h}$. If $N_h \leq 1$, then all VMs on host $h$ would receive their entitlements, assuming that the host-level scheduler is operating properly. If $N_h > 1$, then host $h$ is deemed to have insufficient resources to meet the entitlements of all its VMs, and as a result, some VMs would be treated unfairly. After calculating $N_h$ for each host, the centralized load balancer computes the cluster-wide imbalance, $I_c$, which is defined as the standard deviation over all $N_h$.

The DRS load-balancing algorithm (presented as Algorithm 1) uses a greedy hill-climbing technique. This approach, as opposed to an exhaustive [say offline] approach that would try to find the best target balance, is driven by the practical considerations that the VMotion operations needed to improve load-balancing have a cost and that VM demand is changing over time so highly optimizing for a particular dynamic situation is not worthwhile. DRS aims to minimize $I_c$ by evaluating all possible migrations, many filtered quickly in practice, and selecting the move that would reduce $I_c$ the most. The selected move is applied to the algorithm's current internal cluster snapshot so that it then reflects the state that would result when the migration completes. (The actual migration execution engine runs subsequent to the algorithm.) This move-selection step is repeated until no additional beneficial moves remain or there are enough moves for this pass or the cluster imbalance is at or below the threshold $T$ specified by the DRS administrator.

Please note that this overview of the DRS algorithm is greatly simplified to focus on its core load-balancing metric. The actual load-balancing algorithm considers many other factors, including the risk-adjusted benefit of each move given the range and stability of VMs' dynamic demand over the last hour, as well as the cost of the migration and any potential impact of the migration on the workload running in the VM. A more detailed description of DRS, including both its load-balancing and other cluster management aspects, can be found in [6].

**Algorithm 1**: DRS Load Balancing

```
Input: Snapshot of entire cluster (hosts and VMs)
I_c ⟵ σ(N_h)   /*standard deviation over all hosts*/
NumMigrations ⟵ 0
while I_c > T and NumMigrations < MaxMigrations do
    BestMigration ⟵ NULL
    Max_δ ⟵ 0
    foreach VM v in the cluster do
        foreach compatible destination host h do
            δ ⟵ improvement in imbalance I_c
            when v is migrated to h
            if benefit of migration > cost then
                if δ > Max_δ then
                    BestMigration ⟵ migrate v
                    to h
                    Max_δ ⟵ δ
    if M is NULL then
        break
    Apply M to the algorithm's internal cluster
    snapshot and update I_c
    NumMigrations++
```

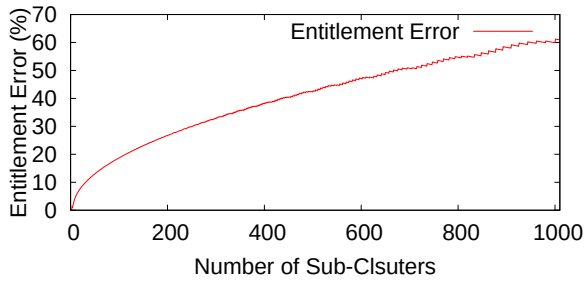


Figure 3: Entitlement error as %age of total entitlement for a cluster as the number of sub-clusters increase

# 3 Resource Management Challenges

As previously described, a successful resource management solution for cloud environments, needs to provide a rich set of resource controls for better isolation, while doing initial placement and load balancing for efficient utilization of underlying resources. However getting all these at large scale is quite challenging. The problems with scaling the existing solutions can be broadly classified as the following:

## 3.1 Inventory Management

As clusters increase in size, the overhead in collecting, analyzing, and acting upon the associated data grows. One needs to be able to collect host and VM level data periodically and act upon that to do load balancing. Similarly the number of concurrent VM power-on operations may need to update some common data structures in a short span of time. Large scale inventory management needs a good decomposition of tasks and fine grained locking of various objects in the inventory. Note that management of elasticity also becomes more important, so that approaches that support rapid increase and decrease of available resources are attractive.

## 3.2 Heterogeneity of clusters

As cluster scale increases, the amount of heterogeneity in the cluster tends to increase. It is difficult for cloud providers to procure a large number of machines that are identical. Also, cloud providers typically add machines over time; there could be several generations of hardware co-existing in the cloud, possibly impacting the ability to live-migrate between hosts. In addition, the storage, networking, and other hardware needs of the VM may limit the hosts to which it may live-migrate. Current virtual machine disk storage systems don't scale to thousands of machines, implying that only a subset of hosts are co-connected to storage devices. As cluster scale increases, the hosts to which a VM can be migrated becomes a small percentage of the cluster.

It is important for any load balancing or initial placement solutions to be aware of the load-balancing islands and their impact. As overlapping islands increase, cluster-wide metrics (e.g. $I_c$) become more and more inaccurate. We did a simulation to study how the global entitlement might be misleading in the extreme case as the number of sub-clusters increases. Figure 3 shows the increase in entitlement error, when calculated globally as the number of overlapping sub-clusters increase. The error in global entitlement is the sum of the absolute difference between the entitlements of the VMs in the cluster with and without considering the sub-cluster (represented as %age of the sum of entitlement) in a cluster with 5000 VMs.

Resource pool operations also become more challenging as the heterogeneity of the cluster increases. Users may specify reservation for sets of VMs by specifying reservation at the resource pool level. But the VMs may be compatible only with few hosts that do not have enough capacity to satisfy all the reservations.

Another issue is the difference in MHz of various processors in a heterogeneous environment. For example, 500 MHz on a Intel's Nehalam chip may be more valuable than that of older generation processors. Newer chips tend to have more L2, L3 caches and better micro-architecture. Thus one may need to consider various architectures while doing placement and load balancing decisions. One way to deal with such issues is to use a

normalizing factor for various CPUs based on a set of benchmarks.

### 3.3 Frequency of operations

As the scale increases, the number of users and the frequency of management operations increase. It is important for the system to keep providing low latency as the cluster size increases.

### 3.4 Resistance to failures

As scale increases, the impact of failure of the resource management component increases; its failure would mean users not being able to power on more VMs or change resource pool settings, resources not flowing between resource pools when the VMs need them, and VMs not being moved out of a host when it becomes overloaded. Also, as scale increases, so does the likelihood of hardware failures. This layer needs to be increasingly robust and should gracefully handle any component failures.

## 4 Techniques

In this section we discuss various techniques to handle the challenges mentioned above while supporting the resource management model discussed in Section 2. The goal of this section is not to advocate a single approach but to promote debate and research projects in different directions. We discuss three approaches here: (1) Hierarchical-scaling, (2) Flat-scaling and (3) Statistical-scaling.

### 4.1 Hierarchical Scaling

In this approach, resource management systems are built on top of each other in order to reach scale. For instance, current tools like DRS provide cluster level resource management using a cluster of up to 32 hosts and 3000 VMs. Similar solutions have been proposed by other companies like Microsoft PRO [1]. In order to scale these solutions, one can build a layer on top of such solutions that interacts with these solutions and does operations like initial placement and load balancing across clusters. This hierarchy can be built within a datacenter and even across datacenters. Some of the issues in building such a hierarchy are:

*Issue: What are the good cluster level metrics to use?*

**Reason:** Existing solutions use host level metrics such as CPU and memory utilization to do resource management. These metrics do not aggregate well for cluster. For example a cluster consisting of 32 hosts, with 2 GHz available CPU per host, will have total of 64 GHz CPU available, but such a cluster may not be able to host a VM with 4 GHz CPU requirement.

*Issue: How to do load balancing across Clusters?*

**Reason:** The layer built on top of small clusters will be expected to do resource management operations across clusters, while keeping track of resource fragmentation within each cluster. Since many common metrics don't aggregate well, this layer may not scale well. The benefit of a hierarchical solution is unclear if the problem doesn't decompose easily in a hierarchical manner.

*Issue: How to handle resource pools?*

**Reason:** The entitlement computation does not decompose in a hierarchical manner. The higher level layers in this case will have to do the resource pool computations by going to the VM level settings. This makes it harder to scale in large environments. Ideally, we would like to decompose entitlement computations across various layers.

### 4.2 Flat Scaling

In this approach, a completely distributed and decentralized resource management layer is built, creating a single domain for resource management. Decisions are made using data collected and aggregated over the large number of hosts and VMs. One way to do this is using structured peer-to-peer techniques where all hosts are part of a P2P network. An aggregation layer similar to SDIMS [12] or Astrolab [10] can collect and aggregate stats across all hosts. A query infrastructure on top of this P2P substrate can find hosts that are over-loaded and under-loaded using techniques like anycast and multicast (Scribe [3]). This approach can do optimizations at a global scale rather than limiting itself to local optimizations as done by the hierarchical solution. For example, power management may yield more benefit in this case because hosts' spare capacity is not binned into smaller-sized clusters. However, this technique also has some issues.

*Issue: How to do compatibility checks efficiently?*

**Reason:** Doing compatibility checks for a VM is equivalent to doing either multi-dimensional queries or taking a join of multiple queries. These are known to be challenging problems for P2P environments.

*Issue: Too many failure modes*

**Reason:** Because of the fully distributed nature, the number of failure modes will be high. The overall code would be much harder to debug and reason about.

*Issues: Hard to get the consistent view of the system?*

**Reason:** Operations may fail due to the lack of a consistent view. For example, group power on of hundreds of VMs may fail several times before succeeding if too many VMs try to power-on on the same host.

*Issue: How to handle resource pools?*

**Reason:** In this case we were able to design a decen-

tralized entitlement computation mechanism. The mechanism works by aggregating various stats and sending them to all hosts using multicast primitives in Scribe. This solution require multiple rounds of communication to converge to the correct set of values.

### 4.3 Statistical Scaling

In this approach, large scale resource management is achieved by doing smarter operations at a small scale. The idea is to create dynamic clusters based on the current operation by querying a set of relevant hosts that are needed. For example, if the operation is VM power on, a query service will ask for top $k$ lightly loaded hosts that also satisfy other constraints for the VM. It will then create a dynamic cluster of those $k$ hosts and invoke DRS-like placement algorithm on that cluster.

One key property we need to show for the success of this approach is that one can attain large scale load balancing and optimal placement by doing repeated small scale optimizations. This intuitive property is supported by the well known research field related to the *power of two choices* [9].

This main implication of the key result in this area is that having a small amount of information in the form of one more choice improves the overall load balancing by a large factor. This result has been used extensively in many other areas for load balancing. In our case, we are selecting hosts based on a greedy criteria and not randomly. In addition, we are running a centralized algorithm on a larger number of hosts (32 or 64). So we expect to have similar load balancing as compared to both the centralized solution (that would considered all hosts) or the flat scaling approach. Some of the issues with this approach include:

*Issue: How to handle resource pools?*

**Reason:** The entitlement computation needs to be done while considering the entire resource pool hierarchy. This may limit the scalability of a single resource pool.

## 5 Conclusions

Efficient management of resources at cloud scale while providing proper performance isolation, higher consolidation and elastic use of underlying hardware resources is key to a successful cloud deployment. Existing approaches either provide poor management controls, or low consolidation ratios, or do not scale well. Based on years of experience shipping the VMware DRS resource management solution and prototypes to increase its scale, we have presented some use cases for powerful controls, key challenges in providing those controls at large scale, and an initial taxonomy of techniques available to do so. We hope that our experience will help motivate future research in this critical area to solve these practical issues.

**Acknowledgments**: Thanks to Carl Waldspurger, John Zedlewski and Minwen Ji who developed the original DRS load balancing algorithm. The authors, along with the rest of the DRS team, work on the current versions of DRS/DPM algorithms. We are indebted to all the members of the Resource Management and DRS teams who have contributed to the development and maintenance of DRS.

## References

[1] Microsoft Performance and Resource Optimization (PRO), 2011. http://technet.microsoft.com/en-us/library/cc917965.aspx.

[2] VMware Distributed Resource Scheduler - Dynamic Resource Balancing, 2011. http://vmware.com/products/drs.

[3] M. Castro, P. Druschel, A.-M. Kermarrec, and A. Rowstron. Scribe: A large-scale and decentralized application-level multicast infrastructure. *IEEE JSAC*, 20(8), Oct. 2002.

[4] G. Clarke. Mckinsey: Adopt the cloud, lose money: Virtualize your datacenter instead. `http://www.theregister.co.uk/2009/04/15/mckinsey_cloud_report`.

[5] U. Drepper. The Cost of Virtualization. *ACM Queue*, Feb. 2008.

[6] D. Epping and F. Denneman. *VMware vSphere 4.1 HA and DRS technical deepdive*. CreateSpace, USA, 2010.

[7] A. Gulati, A. Merchant, and P. Varman. mClock: Handling Throughput Variability for Hypervisor IO Scheduling. In *9th USENIX OSDI*, October 2010.

[8] E. Kotsovinos. Virtualization: Blessing or Curse? *ACM Queue*, Jan. 2011.

[9] M. Mitzenmacher. The power of two choices in randomized load balancing. *IEEE Transactions on Parallel and Distributed Systems*, 12:1094–1104, 2001.

[10] R. V. Renesse and K. Birman. Scalable management and data mining using astrolabe. In *IPTPS*, 2002.

[11] W. Vogels. Beyond Server Consolidation. *ACM Queue*, Feb. 2008.

[12] P. Yalagandula and M. Dahlin. A scalable distributed information management system. SIGCOMM '04, pages 379–390, New York, NY, USA, 2004. ACM.