VMware Distributed Resource Management: Design, Implementation, and Lessons Learned

Ajay Gulati
VMware, Inc.
agulati@vmware.com

Anne Holler
VMware, Inc.
anne@vmware.com

Minwen Ji
Facebook, Inc.
mji@fb.com

Ganesha Shanmuganathan
VMware, Inc.
sganesh@vmware.com

Carl Waldspurger
carl@waldspurger.org

Xiaoyun Zhu
VMware, Inc.
xzhu@vmware.com

Abstract

Automated management of physical resources is critical for reducing the operational costs of virtualized environments. An effective resource-management solution must provide performance isolation among virtual machines (VMs), handle resource fragmentation across physical hosts and optimize scheduling for multiple resources. It must also utilize the underlying hardware infrastructure efficiently. In this paper, we present the design and implementation of two such management solutions: DRS and DPM. We also highlight some key lessons learned from production customer deployments over a period of more than five years.

VMware's Distributed Resource Scheduler (DRS) manages the allocation of physical resources to a set of virtual machines deployed in a cluster of hosts, each running the VMware ESX hypervisor. DRS maps VMs to hosts and performs intelligent load balancing in order to improve performance and to enforce both user-specified policies and system-level constraints. Using a variety of experiments, augmented with simulation results, we show that DRS significantly improves the overall performance of VMs running in a cluster. DRS also supports a "what-if" mode, making it possible to evaluate the impact of changes in workloads or cluster configuration.

VMware's Distributed Power Management (DPM) extends DRS with the ability to reduce power consumption by consolidating VMs onto fewer hosts. DPM recommends evacuating and powering off hosts when CPU and memory resources are lightly utilized. It recommends powering on hosts appropriately as demand increases, or as required to satisfy resource-management policies and constraints. Our extensive evaluation shows that in clusters with non-trivial periods of lowered demand, DPM reduces server power consumption significantly.

Categories and Subject Descriptors

C.4 **[Performance of Systems]**: Modeling techniques;
C.4 **[Performance of Systems]**: Measurement techniques;
C.4 **[Performance of Systems]**: Performance attributes;
D.4.8 **[Operating Systems]**: Performance—*Modeling and prediction*;
D.4.8 **[Operating Systems]**: Performance—*Measurements*;
D.4.8 **[Operating Systems]**: Performance—*Operational analysis*

General Terms

Algorithms, Design, Experimentation, Management, Measurement, Performance

Keywords

VM, Virtualization, Resource Management, Scheduling, Cluster, Hosts, Load Balancing, Power Management

1 Introduction

Initially, the rapid adoption of virtualization was fueled by significant cost savings resulting from server consolidation. Running several virtual machines (VMs) on a single physical host improved hardware utilization, allowing administrators to "do more with less" and reduce capital expenses. Later, more advanced VM capabilities such as cloning, template-based deployment, checkpointing, and live migration [43] of running VMs led to more agile IT infrastructures. As a result, it became much easier to create and manage virtual machines.

The ease of deploying workloads in VMs is leading to increasingly large VM installations. Moreover, hardware technology trends continue to produce more powerful servers with higher core counts and increased memory density, causing consolidation ratios to rise. However, the operational expense of managing VMs now represents a significant fraction of overall costs for datacenters using virtualization. Ideally, the complexity of managing a virtualized environment should also benefit from consolidation, scaling with the number of hosts, rather than the number of VMs. Otherwise, managing a virtual infrastructure would be as hard — or arguably harder, due to sharing and contention — as managing a physical environment, where each application runs on its own dedicated hardware.

In practice, we observed that a large fraction of the operational costs in a virtualized environment were related to the inherent complexity of determining good VM-to-host mappings, and deciding when to use vMotion [8], VMware's live migration technology, to rebalance load by changing those mappings. The difficulty of this problem is exacerbated by the fragmentation of resources across many physical hosts and the need to balance the utilization of multiple resources (including CPU and memory) simultaneously.

We also found that administrators needed a reliable way to specify resource-management policies. In consolidated environments, aggregate demand can often exceed the supply of physical resources. Administrators need expressive resource controls to prioritize VMs of varying importance, in order to isolate and control the performance of diverse workloads competing for the same physical hardware.

We designed *DRS* (for *distributed resource scheduler*), to help reduce the operational complexity of running a virtualized datacenter. DRS enables managing a cluster containing many potentially-heterogeneous hosts as if it were a single pool of resources. In particular, DRS provides several key capabilities:

- A *cluster* abstraction for managing a collection of hosts as a single aggregate entity, with the combined processing and memory resources of its constituent hosts.
- A powerful *resource pool* abstraction, which supports hierarchical resource management among both VMs and groups of VMs. At each level in the hierarchy, DRS provides a rich set of controls for flexibly expressing policies that manage resource contention.
- Automatic *initial placement*, assigning a VM to a specific host within the cluster when it is powered on.
- Dynamic *load balancing* of both CPU and memory resources across hosts in response to dynamic fluctuations in VM demands, as well as changes to the physical infrastructure.
- Custom *rules* to constrain VM placement, including affinity and anti-affinity rules both among VMs and between VMs and hosts.
- A host *maintenance mode* for hardware changes or software upgrades, which evacuates running VMs from one host to other hosts in the cluster.

In this paper, we discuss the design and implementation of DRS and a related technology called *DPM* (for *dynamic power management*). DRS provides automated resource-management capabilities for a cluster of hosts. DPM extends DRS with automated power management, powering off hosts (placing them in standby) during periods of low utilization, and powering them back on when needed [9].

Our experimental evaluation demonstrates that DRS is able to meet resource-management goals while improving the utilization of underlying hardware resources. We also show that DPM can save significant power in large configurations with many hosts and VMs. Both of these features have been shipping as VMware products for more than five years. They have been used by thousands of customers to manage hundreds of thousands of hosts and millions of VMs world-wide.

The remainder of the paper is organized as follows. We discuss the resource model supported by DRS in Section 2. Section 3 explains the details of the DRS algorithm, and Section 4 discusses DPM. Section 5 contains an extensive evaluation of DRS and DPM based on both a real testbed and an internal simulator used to explore alternative solutions. We discuss lessons learned from our deployment experience in Section 6. Section 7 highlights opportunities for future work. A survey of related literature is presented in Section 8, followed by a summary of our conclusions in Section 9.

2. Resource Model

In this section, we discuss the DRS resource model and its associated resource controls. A resource model explains the capabilities and goals of a resource-management solution. DRS offers a powerful resource model and provides a flexible set of resource controls. A wide range of resource-management policies can be specified by using these controls to provide differentiated QoS to groups of VMs.

2.1 Basic Resource Controls

VMware's resource controls allow administrators and users to express allocations in terms of either absolute VM allocation or relative VM importance. Control knobs for processor and memory allocations are provided at the individual host level by the VMware ESX hypervisor. DRS provides exactly the same controls for a distributed cluster consisting of multiple ESX hosts, allowing them to be managed as a single entity. The basic VMware resource controls are:

- **Reservation**: A *reservation* species a minimum guaranteed amount of a certain resource, *i.e.*, a lower bound that applies even when this resource is over-committed heavily. Reservations are expressed in absolute units, such as megahertz (MHz) for CPU, and megabytes (MB) for memory. Admission control during VMpower-on ensures that the sum of the reservations for a resource does not exceed its total capacity.
- **Limit**: A *limit* species an upper bound on the consumption of a certain resource, even when this resource 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* specify relative importance, and are expressed using 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 when there is contention for a resource. In the literature, this control is sometimes referred to as a *weight*.

The ability to express hard bounds on allocations using reservations and limits is extremely important in a virtualized environment. Without such guarantees, it is easy for VMs to suer from unacceptable or unpredictable performance. Meeting performance objectives would require resorting to crude methods such as static partitioning or over-provisioning of physical hardware, negating the advantages of server consolidation. This motivated the original implementation of reservations and limits in ESX, as well as their inclusion in DRS.

Although DRS focuses primarily on CPU and memory resources, similar controls for I/O resources have been validated by a research prototype [30]. VMware also offers shares and limit controls for network and storage bandwidth [10, 28]. A new *Storage DRS* feature was introduced in vSphere 5.0, providing a subset of DRS functionality for virtual disk placement and load-balancing across storage devices [16, 29, 31].

2.2 Resource Pools

In addition to basic, per-VM resource controls, 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 species 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.

Resource pools may be configured in a flexible hierarchical organization; each pool has an enclosing parent pool, and children that may be VMs or sub-pools. Resource pools are useful for dividing or sharing aggregate capacity among groups of users or VMs. For example, administrators often use resource-pool hierarchies to mirror human organizational structures. Resource pools also provide direct support for delegated administration; an administrator can allocate resources in bulk to sub-administrators using sub-pools.

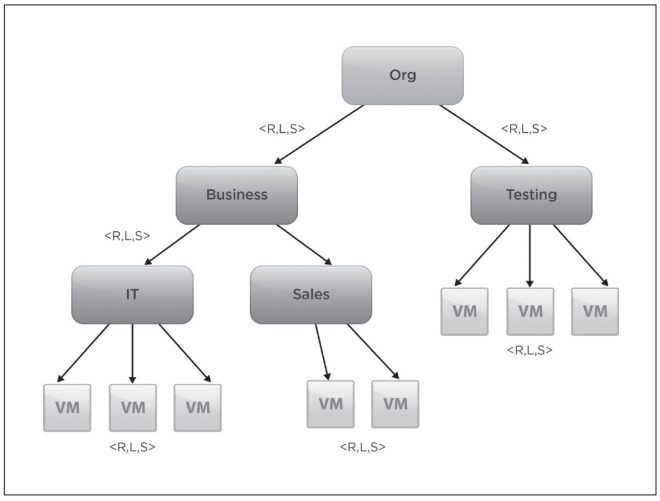

Figure 1. Resource pool tree. R, L, and S denote reservation, limit, and share values, respectively, and are specified for each internal node (pool) and leaf node (VM).

Figure 1 shows a resource pool structure defined by an example organization. Here the resources are first split across two groups, *Business and Testing*. *Business* is further sub-divided into *IT* and *Sales* groups, while *Testing* is a flat collection of VMs. Separate, per-pool allocations provide both isolation between pools, and sharing within pools.

For example, if some VMs within the *Sales* pool are idle, their unused allocation will be reallocated preferentially to other VMs within the same pool. Any remaining spare allocation flows preferentially to other VMs within *Business*, its enclosing parent pool, then to its ancestor, *Org*. Note that in a resource-pool hierarchy, shares are meaningful only with respect to siblings; each pool effectively defines a scope (similar to a currency [50]) within which share values are interpreted.

A distinguished *root resource pool* for a cluster represents the physical capacity of the entire cluster, which is divvied up among its children. All resource pools and VMs in a cluster are descendants of the root resource pool.

2.3 Resource Pool Divvy

A resource pool represents an aggregate resource allocation that may be consumed by its children. We refer to the process of computing the entitled reservation, limit and shares of its child sub-pools and VMs as *divvying*.

Divvying is performed in a hierarchical manner, dividing the resources associated with a parent pool among its children. Divvying starts from the root of the resource pool hierarchy, and ultimately ends with the VMs at its leaves. DRS uses the resulting entitlements to update host-level settings properly, ensuring that controls such as reservations and limits are enforced strictly by the ESX hypervisor. DRS also uses host-level entitlement imbalances to inform VM migration decisions.

During divvying, DRS computes resource entitlements for individual pools and VMs. Divvying computations incorporate user-specified resource controls settings, as well as per-VM and aggregate workload demands. Since pool-level resource allocations reflect VM demands, DRS allows resources to flow among VMs as demands change. This enables convenient multiplexing of resources among groups of VMs, without the need to set any per-VM resource controls.

Divvying must handle several cases in order to maintain the resource pool abstraction. For example, it is possible for the total reservation of a parent pool to be greater than the sum of its childrens' reservations. Similarly, the limit at a parent pool can be smaller than the sum of its childrens' limit values. Finally, a parent's shares need to be distributed among its children in proportion to each child's shares value. In all such cases, the parent values need to be divided among the children based on the user-set values of reservation, limit, shares and the actual runtime demands of the children.

Three divvy operations are defined: *reservation-divvy, limit-divvy,* and *share-divvy*, named for their respective resource controls. DRS carries out these divvy operations periodically (by default, every 5 minutes), reflecting current VM demands. DRS also initiates divvy operations in response to changes in resource allocation settings.

The divvy algorithm works in two phases. In the first, bottom-up phase, divvying starts with the demand values of the individual VMs, which are the leaf nodes. It then accumulates aggregate demands up the resource pool tree.

VM demands are computed by the ESX hypervisor for both CPU and memory. A VM's CPU demand is computed as its actual CPU consumption, CPU_{used}, plus a scaled portion of CPU_{ready}, the time it was ready to execute, but queued due to contention:

$$CPU_{demand} = CPU_{used} + \frac{CPU_{run}}{(CPU_{run} + CPU_{sleep})} \times CPU_{ready} \quad (1)$$

A VM's memory demand is computed by tracking a set of randomly-selected pages in the VM's physical address space, and computing how many of them are touched within a certain time interval [49]. For example, if 40% of the sampled pages are touched for a VM with 16 GB memory allocation, its *active memory* is estimated to be 0.4 X 16 = 6.4 GB.

Once the per-VM demand values have been computed, they are aggregated up the tree. Demand values are updated to always be no less than the reservation and no more than the limit value. Thus, at each node, the following adjustments are made:

$$demand = MAX(demand, reservation) \quad (2)$$

$$demand = MIN(demand, limit) \quad (3)$$

The second divvying phase proceeds in a top-down manner. Reservation and limit values at each parent are used to compute the respective resource settings for its children such that the following constraints are met:

1. Child allocations are in proportion to their shares.

2. Each child is allocated at least its own reservation.

3. No child is allocated more than its own limit.

In order to incorporate demand information, if the sum of the childrens' demands is larger than the quantity (reservation or limit) that is being parceled out, we replace the limit values of the children by their demand values:

$$limit = MIN(demand, limit) \quad (4)$$

This allows reservation and limit settings at a parent to flow among its children based on their actual demands.

Conceptually, the divvy algorithm at a single level in the tree can be implemented by first giving the reservation to every child. Then the extra reservation or limit, depending on what is being divvied out, can be allocated in small chunks to the children, by giving a chunk to the child with minimum value of current *allocation/shares*. If a child's current allocation hits its limit, that child can be taken out from further consideration and capped at the limit.

The share-divvy is performed simply by dividing a parent's shares among its children, in proportion to the shares of each child. Demands do not play any role in the share-divvy operation.

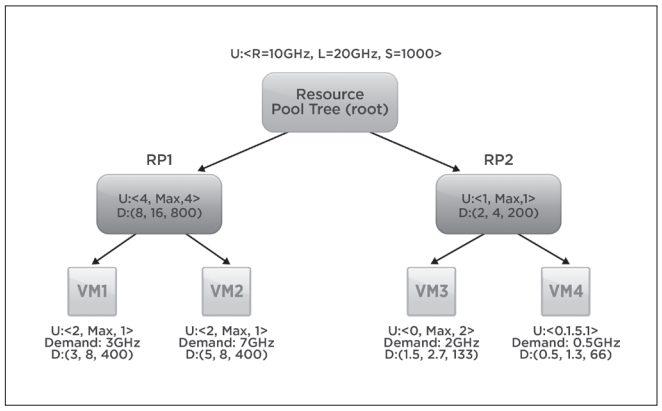

Figure 2. Resource pool divvy example. U denotes the user-set values and D denotes the divvied values. The special Max limit value indicates the allocation is not limited.

We illustrate the divvy operations using a simple example with a two-level resource pool tree. Figure 2 shows a resource pool tree with CPU reservation R = 10 GHz, limit L = 20 GHz and shares S = 1000 at the root node. Shares are unitless and the other two settings are in absolute CPU units (GHz). The user-set resource control values are denoted by U, and the final divvied values are denoted by *D*. Although this example shows only CPU divvying, a similar computation is performed for memory.

In Figure 2, there are two resource pools under the root, *RP*1 and *RP*2, each with two child VMs. First, the bottom-up divvying phase aggregates demands up the tree, starting with individual VM demands. Next, the top-down divvying phase starts at the root node, doing reservation and limit divvy recursively until reaching the leaf nodes.

The 10 GHz reservation at the root is greater than the 5 GHz sum of its childrens' reservations. Thus, the reservation is divvied based on shares, while meeting all the divvying constraints. In this case, the shares-based allocation of 8 GHz and 2 GHz satisfies the reservation and limit values of the two resource pools. Next, the reservation of 8 GHz at RP1 is divvied among VM1 and VM2. Since 8 GHz is smaller than the aggregate demand of 10 GHz from VM1 and VM2, we replace the limit values of VM1 and VM2 by their demand values of 3 GHz and 7 GHz, respectively. As a result, even though the shares of these two VMs are equal, the divvied reservations are 3 GHz and 5 GHz (instead of 4 GHz each). This illustrates how demands are considered while allocating a parent's reservation.

Note that VM demands do not act as an upper bound while divvying limit values here, since the sum of the children's demands is smaller than the limit at the parent. As a result, the actual limit values of the children are used in the divvying. Shares are simply distributed in proportion to the shares of the children. Although we have not walked through every computation in this resource pool tree, one can verify that both reservation and limit values can be divvied in a top-down pass starting from root, while considering user-set values and current demands.

3. DRS Overview and Design

DRS is designed to enforce resource-management policies accurately, delivering physical resources to each VM based on the resource model described in the previous section.

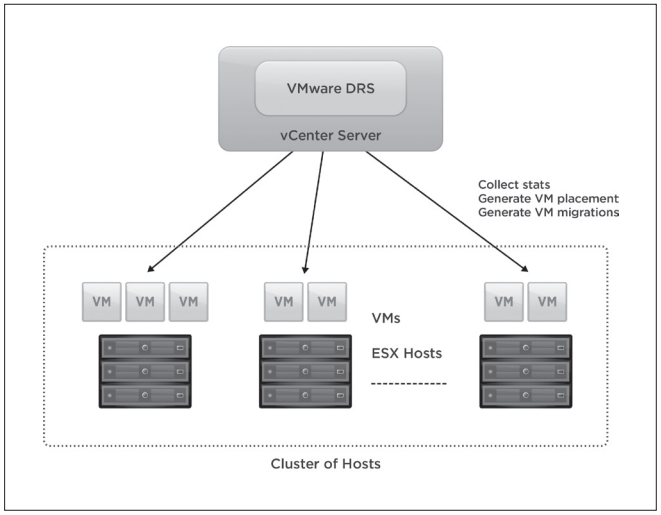

Figure 3. VMware DRS runs as part of the VMware vCenter server management software. VMware DRS collects statistics about hosts and VMs and runs periodically or on demand.

As illustrated in Figure 3, DRS runs as part of the vCenter Server centralized management software [11]. It manages resources for a cluster of ESX hosts as well as the VMs running on them. More specifically, DRS performs four key resource-management operations: (1) It computes the amount of resources to which each VM is entitled based on the reservation, limit and shares values and the runtime demands for all related VMs and resource pools, (2) It recommends and performs migration of powered-on VMs to balance load across hosts in a dynamic environment where VMs' resource demands may change over time, (3) It optionally saves power by invoking DPM, and (4) It performs initial placement of VMs onto hosts, so that a user does not have to make manual placement decisions.

DRS load balancing is invoked periodically (by default, every 5 minutes) to satisfy cluster constraints and ensure delivery of entitled resources. It is also invoked on demand when the user makes cluster configuration changes, *e.g.*, adding a host to the cluster, or requesting that a host enter maintenance mode. When DRS is invoked, it performs the first three resource-management operations listed above, along with a pass to correct cluster constraint violations. For example, constraint correction evacuates VMs from hosts that the user has requested to enter maintenance or standby mode.

DRS initial placement assigns a VM to a host within a cluster when the VM is powered-on, resumed from a suspended state, or migrated into the cluster manually. DRS initial placement shares code with DRS load balancing to ensure that placement recommendations respect constraints and resource entitlements.

In this section, we first discuss DRS load balancing, since it forms the core of the functionality that supports the resource model discussed in the Section 2. We next outline how DRS initial placement works, highlighting how it builds on the DRS load-balancing model. We conclude by presenting the kinds of constraints DRS respects, and how they are handled. DPM is discussed in Section 4.

3.1 Load Balancing

We first examine the DRS load-balancing metric and algorithm. We then consider in more detail how DRS analyzes possible load-balancing moves in terms of their impact on addressing imbalance, their costs and benefits, and their interaction with pending and dependent actions.

3.1.1 Load Balancing Metric

The DRS load-balancing metric is *dynamic entitlement*, which differs from the more commonly-used metric of host utilization in that it reflects resource delivery in accordance with both the needs and importance of the VMs. Dynamic entitlement is computed based on the overall cluster capacity, resource controls, and the actual demand for CPU and memory resources from each VM.

A VM's entitlement for a resource is higher than its reservation and lower than its limit; the 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; otherwise, 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 by running the divvy algorithm over the resource pool hierarchy tree. For entitlement computation, we use the cluster capacity at the root as the quantity that is divvied out. This is done for both CPU and memory resources separately.

DRS currently uses *normalized entitlement* as its core per-host load metric, reflecting host capacity as well as the entitlements of the running VMs. 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. 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, the VMs on that host would be treated unfairly as compared to VMs running on hosts whose normalized entitlements were not above 1.

After calculating N_h for each host, DRS computes the cluster-wide imbalance, I_c, which is defined as the standard deviation over all N_h values. The cluster-wide imbalance considers both CPU and memory imbalance using a weighted sum, in which the weights depend on resource contention. If memory is highly contended, *i.e.*, its max normalized entitlement on any host is above 1, then it is weighted more heavily than CPU. If CPU is highly contended, then it is weighted more heavily than memory; equal weights are used if neither resource is highly contended. The ratio 3:1, derived by experimentation, is used when one resource is weighted more heavily than the other.

3.1.2 Load Balancing Algorithm

The DRS load-balancing algorithm, described in Algorithm 1, uses a greedy hill-climbing technique. This approach, as opposed to an exhaustive, offline approach that would try to find the best target balance, is driven by practical considerations. The live migration operations used to improve load-balancing have a cost, and VM demand is changing over time, so optimizing for a particular dynamic situation is not worthwhile.

DRS aims to minimize cluster-wide imbalance, I_c, by evaluating all possible single-VMmigrations, 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 state so that it reflects the state that would result when the migration completes. This move-selection step is repeated until no additional beneficial moves remain, there are enough moves for this pass, or the cluster imbalance is at or below the threshold T specified by the DRS administrator. After the algorithm completes, an execution engine performs the recommended migrations, optionally requiring user-approval.

```
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
            δ̂ ← improvement in imbalance I_c when v is migrated to h
            if benefit of migration > cost then
                if δ̂ > Max_δ then
                    BestMigration ← migrate v to h
                    Max_δ ← 0
    if BestMigration is NULL then
        break
    Apply BestMigration to the algorithm's internal cluster state and update I_c
    NumMigrations++
```

Algorithm 1: DRS Load Balancing

This overview of the DRS algorithm has been greatly simplified to focus on its core load-balancing metric. The actual load-balancing algorithm considers many other factors, including the impact of the move on imbalance and the risk-adjusted benefit of each move. The next sections describe these checks and other factors in more detail.

3.1.3 Minimum Goodness

DRS load balancing rejects a move if it does not produce enough benefit in terms of improvement in the standard deviation value representing imbalance. The threshold used for this filtering is computed dynamically based on the number of hosts and VMs. The threshold is reduced significantly when imbalance is very high and moves to correct it are filtered by the normal threshold, so that many low-impact moves can be used to correct high imbalance.

3.1.4 Cost-Benefit Analysis

The main check DRS uses to filter unstable moves is cost benefit analysis. Cost-benefit analysis considers the various costs involved in the move by modeling the vMotion costs as well as the cost to the other VMs on the destination host which will have an additional VM competing for resources. The benefit is computed as how much the VMs on the source and the migrating VM will benefit from the move. Cost-Benefit analysis also considers the risk of the move by predicting how the workload might change on both the source and destination and if this move still makes sense when the workload changes.

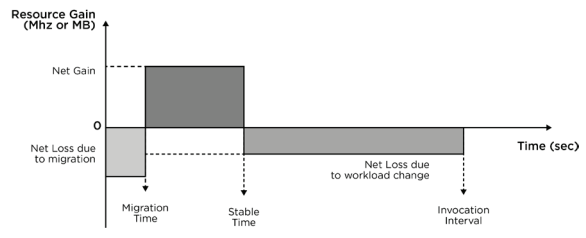

Figure 4: Overview of DRS Cost-Benet analysis

The cost is modeled by estimating how long this VM would take to migrate. The migration time mainly depends on the memory size of the VM as well as how actively the VM modifies its pages during migration. If the VM dirties its pages frequently during vMotion, they must be copied to the destination host multiple times.

DRS keeps track of the transfer rate based on the history of migration times for each VM and host. Based on the transfer rate and current memory size, it calculates the time required for single round of copying. During this time the cost is computed as the resources required for the migrating VM to exist on the destination. The vMotion cost is in the unit of resources (MHz or MB) over time.

The vMotion process itself consumes some resources on both the source and destination hosts and this is also added as a cost. DRS then computes how much the workload inside the VM would suer due to migration. This is done by measuring how actively the VM writes to its memory pages and how much time each transfer takes. DRS approximates this by measuring the number of times the VM had to be descheduled by the CPU scheduler during past vMotions and how active the VM was at that time. The performance degradation due to the increased cost of modifying memory during migration is computed. This value is used to extrapolate how much the VM would suer by taking into account its current consumption and added to the cost. This cost is also measured in units of resources over time, such as CPU seconds.

The vMotion benefit is also measured in resources over time. The benefit is computed by calculating how much of the demand for the candidate VM is being clipped on the source versus the increased amount of demand that is expected to be satisfied on the destination. The benefit also includes the amount of unsatisfied demand for other VMs on the source that will be satisfied after the VM moves off. To represent possible demand changes, demand is predicted for all VMs on the source and destination. Based on the workloads of all these VMs, a *stable time* is predicted after which we assume the VM workloads will change to the worst configuration for this move, given recent demand history (by default, for the previous hour). The use of this worst-case value is intended to make the algorithm more conservative when recommending migrations.

With respect to the source host, the worst possible situation is computed as one in which, after the stable time, the workloads of the VMs on the source exhibit the minimum demand observed during the previous hour. With respect to the destination host, the worst possible situation is computed as one in which the demands for the VMs on the destination including the VM being moved are the maximum values observed in the last one hour. By assuming this worst-case behavior, moves with benefits that depend on VMs with unstable workloads are filtered out.

3.1.5 Pending Recommendations

DRS considers the recommendations in flight and any pending recommendations not yet started, so that it does not correct the same constraint violation or imbalance several times. VMs that are currently migrating are treated as if they exist on both source and destination. New recommendations that would conflict with pending recommendations are not generated.

3.1.6 Move Dependency

When DRS generates a sequence of moves, some VM migrations in the sequence may depend on capacity freed up on a host by a VM migration off that host earlier in the sequence. For example, during the constraint violation correction step, DRS may generate a recommendation to move VM *x* off of host A to correct a rule violation, and then during the load-balancing step, it may generate a recommendation to move VM *y* to host A that depends on the resources freed up by the migration of VM *x*. For such sequences, the second move should be executed only after the first move succeeds. DRS can designate dependencies within its output recommendations, and these dependencies are respected by the DRS recommendation execution engine. DRS does not issue sequences of moves whose dependencies cannot be satisfied.

3.1.7 Host-level Resource Settings

As mentioned in section 2.3, in maintaining the illusion that the cluster is a single large host with the aggregate capacity of its constituent hosts, DRS breaks up the user-specified resource-pool hierarchy into per-host resource pool hierarchies with appropriate host-level resource pool settings. Thus, while the VMs are running on a host, the local schedulers on each ESX host allocate resources to VMs fairly, based on VM resource settings and on the host-level resource pool tree provided to the host by DRS.

At the beginning of each balancing invocation, DRS runs the reservation-divvy and limit-divvy algorithm discussed in Section 2.3, to capture the impact of any changes in VM demand over time. Based on any differences between the updated divvy results and those the algorithm last produced, DRS generates recommendations to adjust the resource trees and settings on hosts in the cluster, in accordance with the VMs running on that host.

3.2 VM Initial Placement

DRS VM initial placement is invoked to perform admission control and host selection for VM power-on, for resumption of a suspended VM, or for manual migration of a running VM into a cluster. DRS can be asked to place a single VM, for which it attempts to generate one or more alternative host placement recommendations, or it can be asked to place a set of VMs, for which it attempts to generate a single coordinated recommendation comprised of a placement action for each VM. The latter handles the placement of multiple VMs more efficiently and effectively than a series of individual VM placements, because it builds a single representation of the cluster to place the set of VMs and because the algorithm can order its placement of the VMs to facilitate bin-packing. Specific errors are issued for any VMs DRS cannot place.

During placement, DRS does not have an estimate of the VM's current CPU and memory demand. It makes the conservative assumption that the VM being placed will consume its maximum possible load, *i.e.*, that its memory demand will match its configured memory size and its CPU demand will be such that each of its virtual CPUs (vCPUs) will consume a physical core. DRS placement code leverages the DRS load-balancing code to evaluate the relative goodness of each possible placement.

However, one way that placement differs from load balancing is that placement considers any prerequisite moves that may be necessary. If a VM cannot be placed without a constraint violation on any powered-on host in the cluster, DRS next considers placing the VM on each of the hosts, while attempting to correct the resulting constraint violations via prerequisite moves of other VMs on the host. And if no such placement on powered-on hosts is possible, DRS considers powering-on standby hosts for the VM via DPM.

DRS also handles the initial placement of fault tolerant (FT) VMs [6]. An FT VM consists of both primary and secondary VMs, with the secondary being a replica of the primary that runs in lock-step to allow immediate failover. The primary and secondary VMs must be placed on different hosts that are vMotion compatible. Since it is computationally very expensive to consider all possible pairs of hosts, DRS places the primary VM on the best possible host in terms of goodness. Then, from the subset of hosts that are vMotion-compatible with the primary, it picks the best host for the secondary. If no secondary host can be found, it considers the second-best host for the primary and tries again. This process is repeated until a host is found for both primary and secondary or there are no more hosts left to consider for the primary VM.

3.3 Constraints

The fundamental constraint respected by DRS is VM-to-Host compatibility, i.e., the ability of a host to satisfy a VM's execution requirements. VM-to-Host compatibility information is passed to DRS from the compatibility-checking module of the vCenter management software, and DRS respects this constraint during load balancing and VM placement.

In addition, DRS provides support for enforcing a set of constraints to handle various use cases such as co-location of VMs for performance, placement of VMs on separate hardware for availability and fault isolation, and affinity to particular hardware to handle licensing issues. It also handles evacuation of hosts the user has requested to enter maintenance or standby mode, preservation of spare resources for failover, and the role of special VMs that provide services to other VMs. DRS respects constraints during VM initial placement and runs a pass prior to load balancing to correct violations of DRS-enforced constraints, reporting errors for any of those constraints that cannot be corrected. We discuss several constraints and their specific use cases next.

3.3.1 Affinity Rules

DRS supports VM-to-VM or VM-to-Host rules, which are used for a variety of common business scenarios. VM-to-VM *anti-affinity* rules define a set of VMs that are to be kept on separate hosts. These rules are typically used for availability and are mandatory, *i.e.,* DRS will not make any recommendation that would violate them. For example, avoiding a single point of failure due to running two VMs on the same host. VM-to-VM *affinity* rules define a set of VMs that are to kept on the same host and are used to enhance performance of communicating VMs, because intra-host VM-to-VM networking is optimized to perform in-memory packet transfers, without using NIC hardware. These rules are mandatory for load-balancing, but

DRS will violate them if necessary to place VM(s) during power-on. VM-to-VM rule violations are corrected during the initial phase of a load-balancing run and that corrected state is maintained during load balancing. For VM-to-VM anti-affinity, potential balancing moves introducing violations are filtered; for VM-to-VM affinity, potential balancing moves are formed by treating each set of affine VMs on a host as if it were a single large VM.

VM-to-Host rules define a set of VMs being affine or anti-affine with a set of hosts. These rules can be specified as mandatory or preferred, with the latter meaning that they are enforced unless they engender additional constraint violations or cannot be respected without causing host overutilization. VM-to-Host rules are useful for a variety of reasons. Mandatory VM-to-Host affinity rules are often used to enforce licensing, to associate VMs requiring a host-based software license with the hosts having that license. Preferred VM-to-Host affinity or anti-affinity rules are often used to manage availability and/or site locality.

Mandatory VM-to-Host rules are represented in the VM-to-Host compatibility information passed to DRS and hence are handled as a fundamental constraint. Preferred VM-to-Host rules are represented as alternative VM-to-Host compatibility information. DRS handles preferred VM-to-Host rules by running a what-if pass with the preferred rules treated as mandatory. If the resulting cluster state has no constraint violations or overutilized hosts, the result of the pass is accepted. Otherwise, DRS retries the what-if pass with the preferred rules dropped and accepts the latter result if it has fewer constraint violations or overutilized hosts, else it accepts the former result.

3.3.2 High Availability

vSphere High Availability [7] (HA) is a cluster service that handles host failures, and restarts failed VMs on remaining healthy hosts. HA runs as a decentralized service on the ESX hosts and keeps track of liveness information through heart-beat mechanisms. DRS supports HA functionality in two ways: by preserving powered-on idle resources to be used for VM restart in accordance with the HA policy and by defragmenting resources in the cluster when HA restart cannot find sufficient resources.

For DRS to preserve enough powered-on idle resources to be used for VM restart, HA needs to express to DRS the resources required based on HA policy settings. HA supports three methods by which users can express the resources they need preserved for failover: (1) the number of host failures they would like to tolerate, (2) the percentage of resources they would like to keep as spare, and (3) the designation of particular host(s) as failover hosts, which are not used for running VMs except during failover.

HA expresses the implication of these policies to DRS using two kinds of constraints: the minimum amount of CPU and memory resources to be kept powered-on and the size and location of unfragmented chunks of resources to be preserved for use during failover. The latter is represented in DRS as *spare VMs*, which act like special VMs with reservations used to ensure DRS maintains spare resource slots into which VMs can be failed over and whose VM-to-Host compatibility information is used to create those slots on appropriate hosts.

Spare VMs are also used in the unusual event that more resources are needed for failure restart than expected due to the failure of more hosts than configured by the HA policy. For any VMs that HA cannot restart, DRS is invoked with spare VMs representing their configuration, with the idea that it may be able to recommend host power-ons and migrations to provide the needed resources.

3.3.3 ESX Agent VMs

Some services that an ESX host provides for VM use (*e.g.*, vShield [15]) are encapsulated in VMs; these VMs are called ESX Agent VMs. An agent VM needs to be powered on and ready to serve prior to any non-agent VM being powered on or migrated to that host, and if the host is placed in maintenance or standby mode, the agent VM needs to be powered off after the non-agent VMs are evacuated from the host.

To support agent VMs, DRS respects their role as part of the host platform, which has a number of implications. The DRS algorithm does not produce recommendations to migrate or place non-agent VMs on hosts on which required agent VMs are not configured. On hosts with configured agent VMs, the DRS algorithm respects the agent VMs' reserved resources even when they are not in a ready-to-serve state. The DRS execution engine understands that non-agent VMs need to wait for required agent VMs to be powered on and ready to serve on the target host. And the DRS load-balancing code understands that agent VMs do not need to have evacuation recommendations produced for them when a host is entering maintenance or standby mode; the agent VMs are automatically powered off by the agent VM framework after the non-agent VMs are evacuated.

4. DPM Overview and Design

Distributed Power Management (DPM) is a feature of DRS that opportunistically saves power by dynamically right-sizing cluster capacity to match workload demands, while respecting all cluster constraints. DPM recommends VM evacuation and powering off ESX hosts when the cluster contains sufficient spare CPU and memory resources. It recommends powering ESX hosts back on when either CPU or memory resource utilization increases appropriately or additional host resources are needed to meet cluster constraints, with DRS itself recommending host power-ons for the latter reason. DPM runs as part of the DRS balancing invocation as an optional final phase.

Note that in addition to DPM, each ESX host performs Host Power Management (HPM), which uses ACPI P-states and C-states [12] on ESX hosts to reduce host power consumption while respecting VM demand. HPM works synergistically with DPM in reducing overall cluster power consumption, with DPM reducing the number of running hosts and HPM reducing the power consumed by those hosts.

DPM decouples the detection of the need to consider host power-on or power-off from the selection of the host, which allows host selection to be independent of how the hosts are currently utilized. This means that DPM host selection for power-off does not depend on finding hosts on which both CPU and memory utilization are currently low; DPM can migrate VMs to exploit low utilization of these resources

currently not manifest on the same host. And more importantly, this decoupling means that DPM host selection for power-off can consider criteria that are more critical in the longer term than current utilization, such as headroom to handle demand burst, power efficiency, or temperature.

For example, host A may currently have lower CPU and memory utilization than host B, but host B may be a better candidate for power-off, because it is smaller than host A and hence is less able to handle VM demand burst than host A, or it is less power-efficient than host A, or it is in a hotter part of the datacenter than host A.

In this section, we first discuss how DPM evaluates the utilization of the powered-on hosts to determine if it should suggest host power-off or power-on recommendations. Next, we examine how DPM evaluates possible recommendations. Finally, we explain how DPM sorts hosts for power-on or power-off consideration. Note that this section includes a number of specific factors whose defaults were chosen based on experimental data; the default values are included to illustrate how the system is typically configured, but all of these values can be tuned by the user if desired.

4.1 Utilization Evaluation

To determine whether the currently powered-on cluster capacity is appropriate for the resource demands of the running VMs, DPM evaluates the CPU and memory resource utilization (VM demand over capacity) of powered-on hosts in the cluster. Utilization can exceed 100% since demand is an estimate of VM resource consumption assuming no contention on the host. As mentioned in Section 2.3, a VM's CPU demand includes both its CPU usage and a portion of its ready time, if any. A VM's memory demand is an estimate of its working set size.

DPM considers demand over an extended time period and it uses the mean demand over that period plus two standard deviations for the utilization calculation. This makes DPM conservative with respect to host power-off, which is the bias customers prefer. In order to be relatively quick to power-on a host in response to demand increases and relatively slow to power-off a host in response to demand decreases, the time period used for host power-off is the last 40 minutes and for host power-on is the last 5 minutes.

Utilization is characterized with respect to a target range of 45-81% (*i.e.*, 63 +/-18), with utilization above 81% considered high and utilization below 45% considered low. DPM evaluates cluster utilization using a measure that incorporates CPU and memory utilization on a per-host basis in such a manner that higher utilizations on certain hosts are not offset by lower utilizations on other hosts. This is done because DRS cannot always equalize VM load across hosts due to constraints.

Use of this metric allows DPM to consider host power-on to address high utilization on a few hosts in the cluster (*e.g.*, ones licensed to run Oracle VMs) when the overall utilization of the powered-on hosts in the cluster is not high, and to consider host power-off to address low utilization on a few hosts in the cluster (e.g., ones not connected to a popular shared storage device) when the overall utilization of the powered-on hosts in the cluster is not low. Note that DPM executes on a representation of the cluster produced by a complete DRS load balancing pass.

DPM computes a cluster-wide high and a low score for each resource. Considering the hosts whose utilization for the power-on demand period is above the high-utilization threshold (default 81%):

$$High\ UtilizationScore = \sqrt{\sum DistanceAboveThreshold^2} \quad (5)$$

Considering the hosts whose utilization for the power-off demand period is below the low utilization threshold (default 45%):

$$LowUtilizationScore = \sqrt{\sum DistanceBelowThreshold^2} \quad (6)$$

If the high score is greater than zero for *either CPU or memory*, DPM considers producing host power-on recommendations. Otherwise, if the low score is greater than zero for *both CPU and memory* (possibly on different hosts), DPM considers producing host power-off recommendations. The key intuition behind this scoring is that we want to identify cases where a subset of hosts may have very high or very low utilizations that can be acted upon by suggesting power-on and power-off operations respectively.

4.2 Host Power-off Recommendations

DPM considers host power-off in the current invocation if two criteria are met: non-zero low scores for both CPU and memory and zero high scores for both. To evaluate various alternatives, DPM considers each candidate host in a sorted order. If a candidate host power-off passes both the cluster utilization and cost-benefit evaluations described below, the recommendation to power it off along with its prerequisite VM evacuation recommendations is added to the list of the recommendations that will be issued by this invocation of DRS. The internal representation of the cluster is then updated to reflect the effects of this recommendation. DPM continues to consider host power-off until either the cluster utilization scores no longer show both low CPU and memory utilization or there are no more candidate hosts to be considered.

For a candidate host, DPM runs DRS in a what-if mode to correct constraints and load balance the cluster assuming that the host were entering standby. If what-if DRS can fully evacuate the host, DPM evaluates the resulting cluster state using the utilization scores given in the previous section and compares the scores with those of the cluster state before the host power-off was considered. If the cluster low utilization score is lowered and if the cluster high utilization score is not increased, then the candidate host power-off is next subjected to DPM power-off cost-benefit analysis.

DPM cost-benefit analysis considers the risk-adjusted costs and benefits of the power-off. The costs include the CPU and memory resources associated with migrating VMs off the powering-off host and the corresponding resources associated with repopulating the host when it powers back on in the future. The costs also include the risk-adjusted resource shortfall with respect to satisfying VM demand if the host's CPU and/or memory resources are needed to meet that demand while the host is entering standby, powered-off, or rebooting. The risk-adjusted resource shortfall is computed

assuming the current demand persists for the expected stable time of that demand given an analysis of recent historical stable time, after which demand spikes to a "worst-case" value. This value is defined to be the mean of the demand over the last 60 minutes plus three standard deviations.

The benefit is the power saved during the time the host is expected to be down, which is translated into CPU and memory resource currency (the host's MHz or MB multiplied by time in standby) to be weighed against cost. By default, the benefit must outweigh the cost by 40x to have the power-off pass DPM's cost benefit filter; this value was chosen based on experimentation to match customer preference of not impacting performance to save power.

4.3 Host Power-on Recommendations

DPM evaluates host power-on given high CPU or memory utilization of any powered-on host. DPM considers each candidate host in a sorted order. If a candidate host power-on passes the cluster utilization evaluation described below, the recommendation to power it on is added to the list of recommendations generated by DRS. The internal representation of the cluster is also updated to reflect the effects of this recommendation. DPM continues to consider host power-on until either the cluster utilization scores no longer show either high CPU or memory utilization or there are no more candidate hosts to be considered.

For a candidate host, DPM runs DRS in a what-if mode to correct constraints and load balance the cluster assuming that host were powered-on. If the host power-on reduces the high utilization score, then the stability of that improvement is evaluated. Stability of the improvement involves checking that the greedy rebalancing performed by DRS in the presence of that host was better because of the addition of that host. This is checked by doing a what-if power-off of the same candidate host and checking that the improvement in the high-utilization score obtained by considering its power-on does not remain. If the host power-on benefit is stable, then the host is selected for power-on. Some extra evaluation is performed for the last host needed to address all remaining high utilization in the cluster. In that case, DPM evaluates multiple hosts until it finds one that decreases the high score and provides minimum or zero increase in the low utilization score.

4.4 Sorting Hosts for Power-on/off

As mentioned earlier, DPM considers the hosts for evaluation in a sorted order. DPM currently prefers to keep larger servers powered-on, motivated by the fact that they provide more resource headroom and that they typically are more efficient at amortizing their idle power consumption, which for datacenter-class servers often exceeds 60% of their peak power. For servers of the same size, cost of evacuation is a secondary criteria for power-off consideration and randomization for wear-leveling is a secondary criteria for power-on consideration.

As long as some host in the order satisfies the necessary conditions, it is selected and the corresponding operation is applied to that host in the internal snapshot. The rest of the DPM evaluation is conducted with that host in the new state. Thus DPM doesn't select a host that leads to maximum improvement in the scores, but uses this sorted list as the greedy order.

5. Experimental Evaluation

In this section, we present the results of an extensive evaluation of DRS and DPM using an in-house simulator and experiments on a real testbed. In the first subsection, we present the results using a cluster simulator to show how DRS load-balancing improves the amount of CPU and memory resources delivered to a set of VMs as compared to those delivered without DRS load-balancing; we also highlight the need and impact of DRS cost-benefit analysis and of DPM.

In the second subsection, we present results gathered on a real cluster to illustrate the effectiveness of the DRS algorithm using a variety of metrics. And finally, we will present the results gathered on a real cluster that includes power measurements to show the performance of DPM and to demonstrate the benefit of using DPM in addition to host power management [13].

5.1 Experimental Results from Simulation

5.1.1 Simulator Design

To develop and evaluate various algorithms, we implemented a simulator that simulates a cluster of ESX hosts and VMs. The simulator allows us to create different VM and host profiles in order to experiment with different configurations. For example, a VM can be defined in terms of a number of virtual CPUs (vCPUs), configured CPU (in MHz) and configured memory size (in MB). Similarly a host can be defined using parameters such as number of physical cores, CPU (MHz) per core, total memory size, power consumption when idle etc. In terms of workload, the simulator supports arbitrary workload specifications for each VM over time and generates CPU and memory demand for that VM based on the specification.

Based on the physical characteristics of the host, VM resource demands and specifications, the simulator mimics ESX CPU and memory schedulers and allocates resources to the VMs in a manner consistent with the behavior of ESX hosts in a real DRS cluster. The simulator supports all the resource controls supported by the real ESX hosts, including reservation, limit and shares for each VM along with the resource pools.

The simulator generates the allocation each VM receives, whenever there is any change in demand by any of the VMs on the host. The simulator supports vMotion (live migration) of VMs, models the cost of vMotion and the impact on the workload running in the VM, based on a model of how vMotion works in a physical ESX host. The simulator also takes into account the resource settings for the resource pool trees on the host when resources are divvyed out, similar to how the real ESX host divvies out the host resources based on the host-level resource pool hierarchy.

The simulator was used to study configurations of VMs and hosts and how the algorithm performs in terms of improving the resource allocation to the VMs by placing and moving the VMs at the right time. If the algorithm causes too many migrations, it will affect the workload negatively as the migrations may require a non-trivial amount of time and the workload performance will be degraded during migration.

The production DRS algorithm code is compiled into this simulator. This allows the quality of the moves can be evaluated using all the modeling done by the simulator and decisions made by DRS or DPM.

Since the simulator is deterministic, we can run a test with different algorithms and compute the overall resources allocated to the VMs. Even though the simulator supports random distributions as input for workload generations, the random numbers use a configurable seed so that the VMs would have the same load value at the same time in the simulation across different runs.

The main metric used to study how effectively resources are being allocated is a *cumulative payload metric* that can be defined as follows:

$$\sum_{t=1}^{T} \sum_{k=1}^{All\ VMs} \frac{(Satisfied\ Demand\ for\ VM_k\ at\ t)}{Cluster\ capacity\ at\ t} \times 100 \quad (7)$$

Here *T* denotes the total simulation time. This metric is calculated separately for both CPU and memory. The CPU payload is the area under the curve when the sum of the CPU utilization for all the VMs is plotted in time. If all the VMs have equal shares and have no reservation or limit, a higher number is better. Furthermore, this value is normalized by dividing the area by sum of capacity of all the hosts times length of the experiment. The CPU playload thus captures the total number of CPU cycles in the cluster (on all the hosts) and how many of those cycles were usefully spent.

The memory payload is computed similarly. If VMs have different shares, then this metric must be calculated for each share class to ensure the DRS algorithm is fair when it improves the overall utilization. The simulator also estimates power consumption of the ESX hosts based on a simple power model that involves base power and additional power for the workload it currently runs.

EXPERIMENT	MIGRATIONS	CPU	MEMORY
Without VMware DRS	0	55.68	65.94
With VMware DRS	166	73.74	94.99

Table 1: Benefits of VMware DRS. VMware DRS improved CPU and memory utilization significantly.

5.1.2 Simulator-Based Evaluation

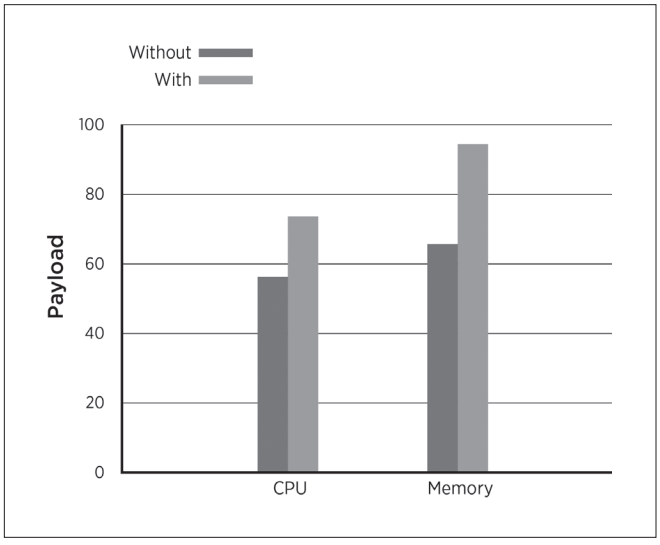

Figure 5: Difference in CPU and Memory payload when VMware DRS is enabled

Using the simulator we first show the benefits of DRS load balancing. We simulated an environment consisting of 30 hosts and 400 VMs. The simulated hosts were each configured with 8 cores and 3 GB of RAM. The VMs were all configured as uniprocessors (with a single vCPU) and had a High-Low workload pattern inducing a demand of either 600 or 1800 MHz randomly for 600 seconds, followed by an inactive period where they would demand 100 MHz for another 600 seconds. The VMs had constant memory demand of 220 MB including overhead. The VMs were randomly placed in the beginning of the experiments.

In the first experiment, DRS was disabled and the VMs could not be moved around so that active VMs could take advantage of the hosts with many inactive VMs. When the same experiment was performed with DRS, the VMs were moved when some hosts had many inactive VMs and some hosts were overcommited. The results are shown in Table 1. The CPU and memory metrics were as computed using Equation 7. The results show the percentage of the total cluster resources (in the units of cycles for CPU and memory-seconds for memory) that was used to satisfy demand.

In the next set of experiments, we illustrate the impact of DRS cost-benefit analysis. The environment consisted of 15 hosts and 25 VMs. The hosts were configured with three different types — a small host with 2 cores, 2 GB memory, a medium host containing 4 cores and 2 GB of memory and a large host containing 4 cores and 4 GB of memory. The workloads inside the VMs mimicked a diurnal pattern. For 600 seconds the VMs were busy. For experiment 1, The values were normally distributed around 1000 MHz with a variance of 800 MHz. For the next 600 seconds it was distributed around 600 MHz with a variance of 500 MHz. For the other two experiments the high value was distributed uniformly between 500 and 1500 MHz and the low value was distributed between 0 and 500 MHz. Figure 7 shows a typical workload inside VM from experiment 1. DPM was turned on for these set of experiments. The three experiments show that cost benefit significantly reduces the

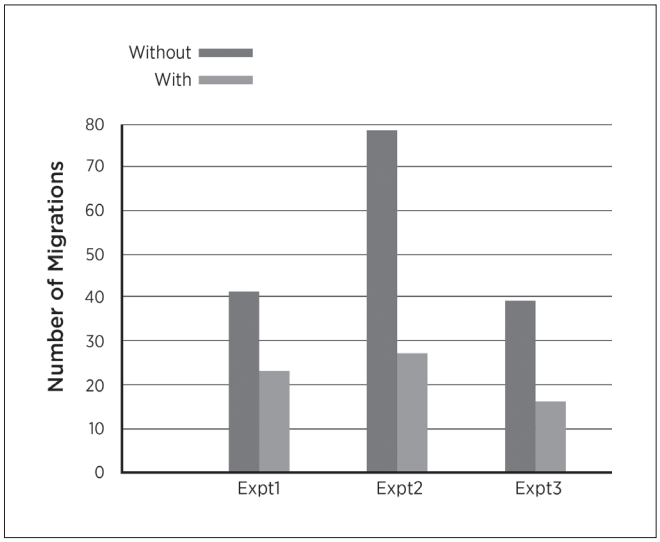

Figure 6: Difference in the number of migrations when cost-benefit analysis is enabled.

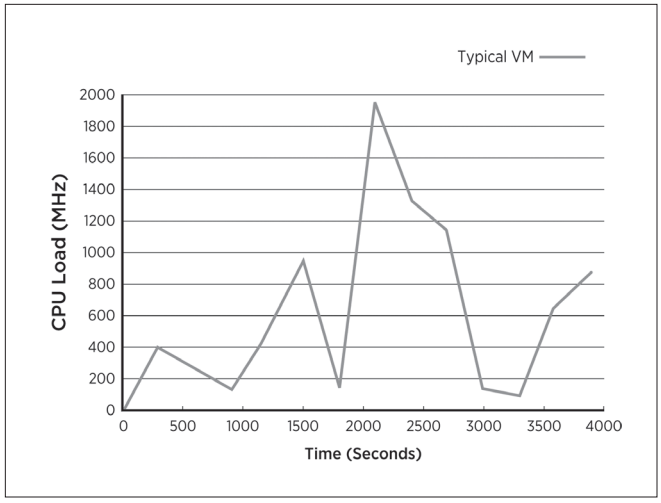

Figure 7: Typical VM workload used in the cost-benefit experiments.

number of migrations while keeping CPU and memory utilizations of the cluster the same or improving it. Table 2 presents the results of these experiments.

EXPERIMENT	MIGRATIONS	CPU	MEMORY
Expt1	41	19.16	15.84
Expt1-CB	25	19.25	15.86
Expt2	78	18.92	15.84
Expt2-CB	27	19.20	15.84
Expt3	39	6.67	42.90
Expt3-CB	16	6.67	43.05

Table 2: Impact of cost-benefit analysis.

The simulator also models the power consumption of each host and the whole cluster. We used a cluster with 32 hosts and 200 VMs. Similar to the cost benefit experiment, the VMs had a day-night workload where all the VMs were active (between 1200 and 2200 MHz) during the day and would go partially idle (workload normally distributed between 500 to 1000 MHz) during the night. DPM would shut down the hosts when the VMs became idle. Table 3 shows the results with and without DPM. Power is in the units of Kilowatt hours. The CPU and memory metrics are as described above.

EXPERIMENT	MIGRATIONS	CPU	MEMORY	POWER
Without VMware DPM	0	2.77	2.31	31.65
With VMware DPM	273	2.78	2.32	11.97

Table 3: DPM achieves signicant power savings without impacting performance.

5.2 Real Testbed: DRS Performance

We now present the results of an experiment that shows the effectiveness of the DRS algorithm on a real testbed. The experiment was run on a DRS cluster consisting of 32 ESX hosts and 1280 Red Hat Enterprise Linux (RHEL) VMs. vSphere 5.0 [4] was used to manage the cluster. The VMs belonged to 4 different resource pools (RP-HighShare, RP-NormShare-1, RP-NormShare-2, RP-LowShare) with different resource shares, with 320 VMs in each resource pool. Each VM was configured with 1 vCPU and 1 GB memory. Notice that we are using only uniprocessor VMs in our evaluation because the number of vCPUs in a VM does not affect DRS behavior in a significant way. The hardware and software setup of the testbed follows:

- **vCenter Server 5.0**: Intel Xeon E5420, 2 x 4 cores @ 2.50 GHz, 16 GB RAM, runs 64-bit Windows 2008 Server SP1.
- **vCenter Database**: AMD Opteron 2212 HE, 2 x 2 cores @ 2 GHz, 12 GB RAM, runs 64-bit Windows 2008 Server SP1 and Microsoft SQL Server 2005.
- **ESX 5.0 hosts**: Dell PoweEdge 1950II servers, Intel Xeon E5420, 2 x 4 cores @ 2.5 GHz, 32 GB RAM.
- **Network**: 1 Gb vMotion network configured on a private VLAN.
- **Storage**: 10 LUNs on an EMC Clariion CX3-80 array shared by all the hosts.

In this experiment, DRS was enabled with default settings, and DPM was disabled, so all 32 hosts remained powered on. We started with all 1280 VMs idling in the cluster. These VMs were evenly distributed across all 32 hosts, due to DRS load balancing. We then injected a CPU load of roughly 20% of a core into each of the 640 VMs in the first two resource pools (RP-HighShare and RP-NormShare-1). Since these 640 VMs were running on 16 hosts of the cluster, the VM load spike led to an overload on these 16 hosts and a significant load imbalance in the cluster.

Even though such an extreme scenario may not occur commonly in a datacenter, it serves as a stress test for evaluating how DRS handles such a significant imbalance in the cluster. We also use this example to illustrate the following key metrics for characterizing the effectiveness of the DRS algorithm.

- **Responsiveness**: How quickly can DRS respond to VM load changes?
- **Host Utilization**: Total resource entitlement from all the VMs running on a host, normalized by the host's available capacity.
- **Cluster Imbalance**: The standard deviation of the current host load across all hosts in the cluster.
- **Number of vMotions**: How many vMotions are recommended by DRS to balance the cluster?
- **VM Happiness**: Percentage of resource entitlement received by each VM.

Next we discuss the experimental results using these five metrics.

Responsiveness: In this experiment, DRS responded in the first invocation after the VM load spikes that led to a severe imbalance in the cluster. The DRS algorithm took 20 seconds to run, and gave recommendations to migrate VMs away from the overloaded hosts. This behavior demonstrates the responsiveness of the DRS algorithm when faced with significant VM load changes.

Host Utilization: Figures 8 and 9 show the normalized total CPU/memory entitlement of all hosts based on the VMs running on each host.[1] The larger number is worse in terms of load on the host. A value above one indicates that the host is overloaded. As we can see, due to the increased VM loads, 16 of the hosts had a spike in the normalized CPU utilization and reached an overload state, whereas the other 16 hosts remained lightly loaded.

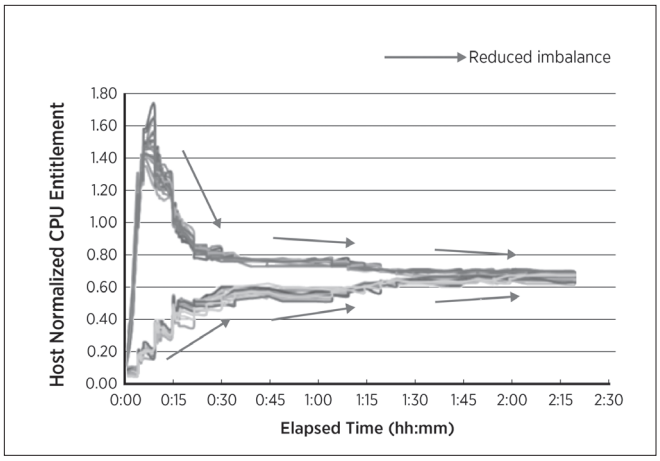

Figure 8: Per-host normalized CPU entitlement. Each line represents a host in the 32-host cluster.

After DRS started moving VMs out of the busy hosts, within 30 minutes the CPU utilization of the most-loaded hosts was reduced to around 0.8. In the remaining time, DRS continued to balance the CPU load in the cluster and reached a steady state in 2 hours and 20 minutes. As a result of balancing the host CPU loads, the host memory loads became slightly imbalanced, while staying within a range of 0:14-0:32. This is acceptable because for most applications, memory does not become a bottleneck at such low utilizations.

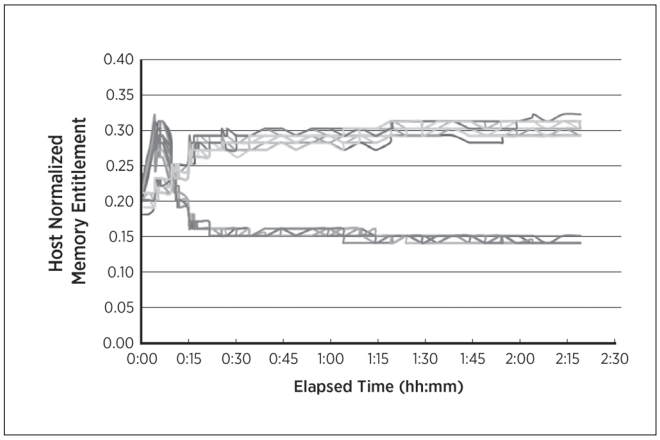

Figure 9: Per-host normalized memory entitlement. Each line represents a host in the 32-host cluster.

Cluster Imbalance: Figure 10 plots the cluster imbalance over time. The target imbalance value is 0.05 in this case. As a result of the VM load increase, the cluster imbalance initially reached a high value of 0.5. DRS quickly brought the imbalance to below 0.1 within 30 minutes, and within 90 minutes the cluster was fairly close to being "balanced." In the remaining 50 minutes, DRS further reduced the imbalance gradually and finally brought the cluster imbalance down below its target.

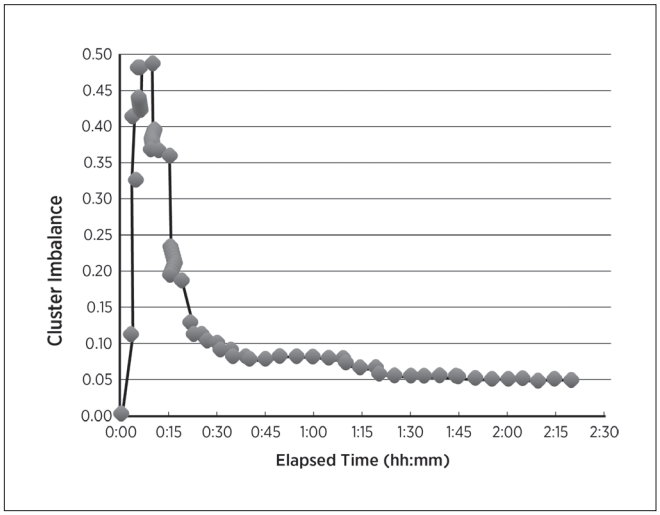

Figure 10: Cluster imbalance.

Number of vMotions: Figure 11 reports the number of vMotions recommended during each DRS invocation. A total of 267 vMotions were executed, and among these, 186 vMotions occurred in the first three rounds of DRS invocations (> 40 in each round), due to the severe imbalance in the cluster. This caused the cluster imbalance to drop quickly. In the remaining 25 rounds of invocations, the imbalance became more subtle and DRS behaved relatively conservatively, taking into account the migration cost in the cost-benefit analysis.

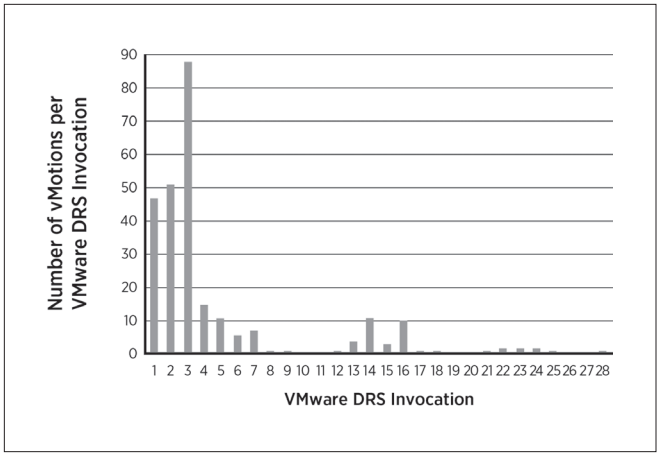

Figure 11: Number of vMotion operations per VMware DRS invocation.

VM Happiness: Finally, we summarize the overall VM happiness during the course of the experiment. Since memory never became a bottleneck in this experiment, we focus on CPU and define *happiness* for an individual VM as the percentage of CPU entitlement received by this VM. For a resource pool, we characterize its overall happiness using two metrics — *average happiness and percentage of happy VMs*. The former is the individual VM happiness averaged across all the VMs in the resource pool, and the latter is the percentage of VMs in the resource pool receiving at least 95% of entitled CPU.

[1] In most of the figures in this subsection, the x-axis represents the elapsed time T in the format of hh:mm, where T=0:00 indicates when the VM load increase occurred.

The values of these two metrics for the 4 resource pools are displayed in Figure 12 and Figure 13, respectively. In the first 30 minutes, the happiness of the VMs in the first two resource pools (RP-HighShare and RP-NormShare-1) suffered, because the VMs in these two resource pools experienced a load spike and started contending for CPU time on the 16 overloaded hosts. However, it is worth noting that under such resource contention, the average happiness of the first resource pool (RP-HighShare) is much higher than that of the second resource pool (RP-NormShare-1), because the former has a higher shares value than the latter ("High" vs. "Normal").

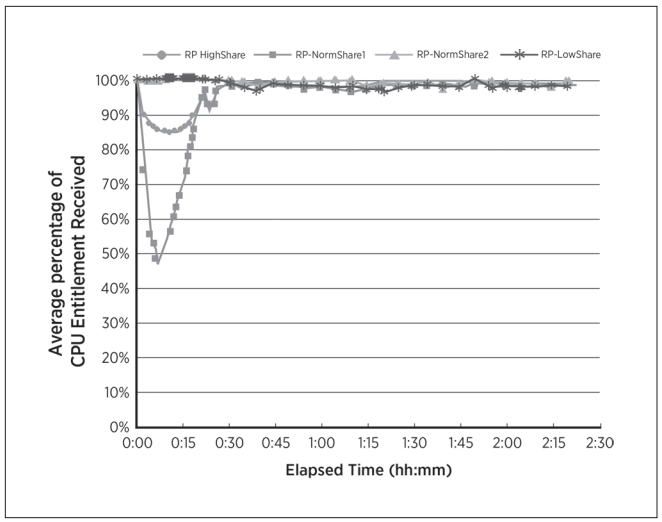

Figure 12: Average happiness (percentage of CPU entitlement received) for each of the 4 resource pools.

This result also validates the fairness of the DRS algorithm with respect to resource shares. Over time, DRS was able to improve the happiness of all the VMs, resulting in an average happiness of close to 100% for all the resource pools. Similarly, the percentage of happy VMs reached 100% for the first two resource pools (RP-HighShare and RP-NormShare-1), and was above 90% for the other two resource pools.

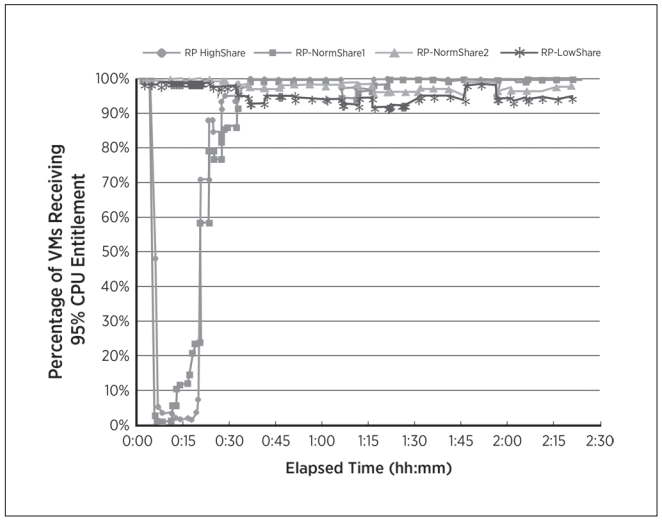

Figure 13: Percentage of happy VMs (receiving at least 95% of CPU entitlement) for each of the 4 resource pools.

5.3 Real Testbed: DPM Performance

We present the results of an experiment that evaluates the performance of DPM and compares DPM with host power management (HPM) [13]. These results demonstrate that DPM is able to provide a fairly significant reduction in power consumption for clusters with sufficiently long periods of low utilization. In addition, combining DPM with host power management provides the most power savings as compared with using either policy alone.

The experiment was run on a DRS cluster consisting of 8 ESX 4.1 hosts and 400 RHEL VMs. vSphere 4.1 [4] was used to manage the cluster. Each VM was configured with 1 vCPU and 1 GB memory. The hardware setup for the testbed is similar to that of the DRS experiment, except that we used a later generation of Dell servers (PowerEdge R610) for the ESX hosts, so that we could obtain host power measurements from the iDRAC controller [2].

The experiment was run for 90 minutes. In the first 30 minutes, all 400 VMs were idle. After the 30th minute, each VM had a surge in the CPU demand resulting in higher cluster utilization. The increased load lasted until the 60th minute, after which all the VMs became idle again. The experiment was repeated three times using three power management policies: *HPM only*, *DPM only*, and *DPM+HPM combined*.

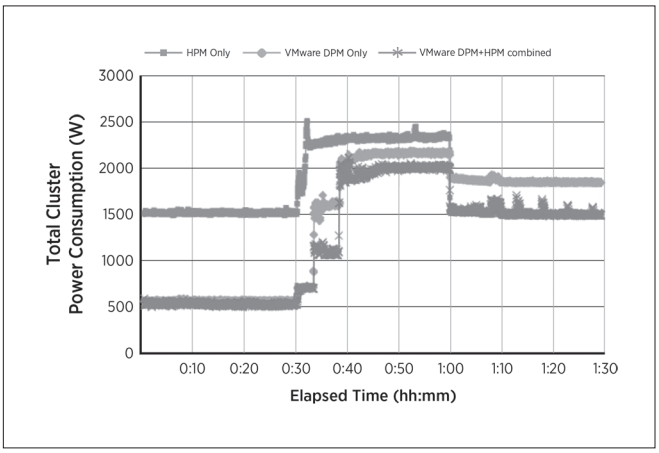

Figure 14: Total power consumption of the 8-host cluster.

Figure 14 shows the total power consumption from the eight hosts in the cluster as a function of time for the three different policies. Next, we describe the result from each policy.

HPM only: In this case, the HPM "Balanced" policy [13] was used. Since DPM was disabled, all eight hosts remained powered on. When all the VMs were idle (first and last 30 minutes), each host had a low (about 25%) CPU utilization. HPM was able to reduce the per-host power consumption by leveraging the lower processor P-states, resulting in 1500W of total cluster power. When the VMs were busy (second 30 minutes), host CPU utilization became high enough (> 60%) such that all the processor cores were running at the highest P-state (P0) most of the time. As a result, the total cluster power was between 2200-2300W.

DPM only: In this case, the HPM policy was set to "High Performance", effectively disabling HPM. Since DPM had been enabled, six of the eight hosts remained in the powered-off state during the initial 30 minutes of the idle period. This led to a total cluster power consumption of about 566W, a 60% reduction as compared with the 1500W in the HPM-only case. Notice the power saving is not exactly 75% here because in the DPM case, the two hosts that remained powered on had a much higher CPU utilization, resulting in higher per-host power consumption. After the VM load increase, DPM first powered on four of the standby hosts in the 33rd minute, and then powered on two additional hosts in the 38th minute, bringing the cluster back to full capacity. The total cluster power consumption increased to approximately 2100W after all the hosts were powered on. When the VMs became idle again, DPM (by design) kept all the hosts powered on for more than 30 minutes, resulting in a total cluster power consumption of approximately 1900W.

DPM+HPM combined: This case is similar to the DPM-only case, except for the following three observations. First, after the VM load increase, DPM initially powered on two of the standby hosts in the 33rd minute, and then powered on the four remaining hosts in the 38th minute. Second, the total cluster power after all the hosts were powered on was roughly 2000W, 100W lower compared to the DPM-only case. This was because the newly powered-on hosts had fewer VMs running on them, resulting in lower host utilization and providing HPM with an opportunity to reduce the power consumption on these hosts. Third, in the last 30 minutes when the cluster was idle, the DPM+HPM combined policy provided an additional 400W of power reduction (1500W vs. 1900W) compared to the DPM-only case. Overall, this combined policy provides the maximum power savings among the three power management policies we tested.

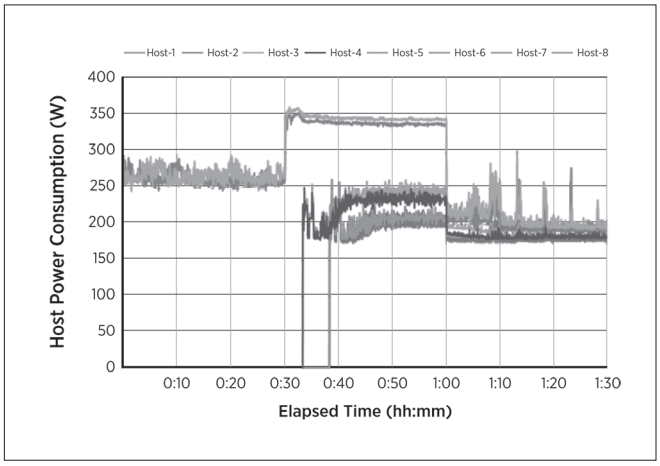

Figure 15: Per-host power consumption for the 8 hosts in the cluster.

Figure 15 shows the per-host power consumption as a function of time for the DPM+HPM combined case. We can clearly see how each host's power consumption varied as the host power state or the VM load level changed. In the last 30 minutes, periodic spikes are visible in the host power consumption, due to vMotion-induced higher CPU utilization on these hosts. These vMotions were recommended by DRS every five minutes to balance the load in the cluster.

6. DRS and DPM in the Field

When DRS was introduced in early 2006, vMotion was just beginning to gain widespread adoption, but customers were wary of automated migration of virtual machines. In the early days, one request from customers was support for manual-move recommendations. A human administrator would inspect the recommendations and apply the moves only if they made sense. Some administrators would run DRS in manual mode and if the same move was recommended over a substantial number of DRS invocations, then the administrator would apply the move. The use of DRS manual mode diminished over time as the DRS algorithm became more mature and as administrators became more comfortable with automated VM migration; as of vSphere 5.0, the use of DRS manual mode is very low.

The first version of DRS did not have cost-benefit analysis turned on, as the code was considered experimental. This led to the problem that DRS could make recommendations that moved VMs back and forth in response to frequently changing demand. In the very next release, cost-benefit feature was enabled by default, leading to higher-quality moves and fewer migrations.

The DRS algorithm tries to get the biggest bang for its vMotion buck, *i.e.*, to minimize the total number of moves needed for load-balancing. Moving the largest, most-active VMs can have the highest impact on correcting imbalance, and hence DRS would favor such moves. While choosing such VMs for vMotion in order to issue fewer moves seemed good in theory, some customers did not like this selection since their largest, most active VMs were also their most important and performance-sensitive VMs, and vMotioning those VMs could adversely impact their performance during the migration.

To address this issue, DRS cost-benefit analysis was changed to take into account the impact of vMotion on the workload which it had not done previously. As vSphere's vMotion continued to be improved, the cost modeling of that impact required updating as well. Over time we learned that the modeling aspects of the algorithm should be separated from the parts of the algorithm that use the model, to ease the maintenance of the algorithm code as the technology changes. For example, we moved to having the algorithm consider the vMotion time, with the details of the parameters relevant to that generation of vMotion technology handled in modeling-specific code.

Earlier versions of DRS did not support affinity between VMs and hosts and it was thought that affinities between VMs should be sufficient. We also wanted administrators to think less about individual hosts and more about aggregate clusters. While VM-to-VM affinity was sufficient for most technical use-cases, there were other requirements such as software licensing that made administrators want to isolate VMs onto a set of hosts. Administrators started rolling out their own solutions to pinning VMs to a set of hosts, such as adding dummy networks to the VMs and adding the networks only to a subset of hosts, making the other hosts incompatible.

VM-to-Host rules were added to DRS in vSphere 4.1 and have been used for licensing and other use cases such as supporting availability zones. Similarly, partners asked for DRS to support ESX agent VMs, which did not need to be migrated off hosts entering maintenance or standby mode and which needed to be ready to serve on an active host before non-agent VMs could placed or migrated to that host; DRS support for agent VMs was introduced in vSphere 5.0.

Another area where administrators wanted improvement was in reporting and tracking the reasons for DRS recommendations. We added reason descriptions to the recommendations and descriptive faults for situations that prevented recommendations from being issued. Users also wanted to know how DRS evaluates the cluster as the cluster-wide entitlement of a particular VM is not always apparent. Since VM entitlement depends on the demands and resource settings of all the VMs and resource pools in the cluster, we added graphical displays of metrics to the DRS user interface, showing demand and entitlement for VMs and resource pools, as well as cluster-wide metrics including overall imbalance and the percentage of a VM's entitled resources being delivered to it.

When DPM was first introduced, the only technology it supported to bring hosts out of standby was Wake On Lan (WoL). This made some administrators uncomfortable, since WoL packets were required to be sent over the vMotion network (on the same subnet) from another host in the cluster, hence requiring at least one connected powered-on host in the cluster to power the others on. Subsequent versions of DPM supported IPMI and iLO, allowing vCenter to power-on a host by talking directly with its baseboard controller, improving adoption.

Administrators were also uncomfortable when DPM shut down many hosts even when they were idle. Based on this concern, multiple options were added to DPM to make it more conservative as well as take into account the intentions of administrators regarding how many hosts can be powered down.

A third area that was improved over several releases was the interoperability between DPM and the VMware high availability (HA) product. Since DPM reduced the number of powered-on hosts HA could choose from for restarting failing-over VMs, HA needed to convey to DPM the user's configured VM failover coverage. Spare VMs were introduced to express this information, and they prevented consolidation beyond their reservation requirements.

7. Future Directions

In this section, we highlight several of the future directions that we are actively exploring for DRS and DPM.

7.1 DRS

One important area for improvement is to expand DRS scaling for cloud-scale resource management. In vSphere 5.0, the supported maximum DRS cluster size is 32 hosts and 3000 VMs, which satisfies enterprise department deployments but falls short of cloud scale. In [32], we proposed three techniques for increasing DRS scale: hierarchical scaling (build a meta-load-balancer on top of DRS clusters), at scaling (build an overlay network on ESX hosts and run DRS as a decentralized algorithm), and statistical scaling (run DRS on a selected subset of the hosts in the cluster), and we argued for statistical scaling as being the most robust.

Another future direction is to expand the computing resources that DRS manages beyond CPU and memory. In the area of network resource management, vSphere currently supports Network I/O control [10], which runs on ESX hosts to provide shares and limits for traffic types, but there is the additional opportunity for DRS to provide cross-host network management, including vMotion to respect NIC bandwidth reservations, avoid NIC saturation, or locate communicating VMs closer together.

In the area of storage resource management, vSphere recently introduced SIOC [28] (vSphere 4.1) to manage data-store I/O contention and Storage DRS [16, 29, 31] (vSphere 5.0) to place and redistribute virtual machine disks using storage vMotion across a cluster of datastores for out-of-space avoidance and I/O load balancing, but there is the additional opportunity for DRS to support cross-host storage management, including storage vMotion perhaps coupled with vMotion to allow VM power-on placement to respect I/O bandwidth reservations.

A third future direction is to support proactive operation of DRS and DPM. Currently DRS and DPM operate reactively, with the last one hour's worth of behavior used in making its calculations of current demand more conservative. The reactive model works well in ensuring that the recommendations made by DRS and DPM are worthwhile.

However, when there is a sudden steep increase in demand, a reactive operation can result in undesirably-high latency to obtain resources (*e.g.*, time to power on a host or to reclaim memory resources from other VMs) or difficulty in obtaining the resources needed to respond while those resources are being highly contended. When such sudden steep increases in demand are predictable (*e.g.*, an 8am spike in VM usage), proactive operation can allow preparation for the demand spike to occur, to hide the latency of obtaining the resources and to avoid competing for resources with the demand spike itself.

7.2 DPM

One future direction for DPM is to revise DPM's host sorting criteria for choosing host power-on/-off candidates. The host sorting currently does not use host power efficiency, which is becoming more practical as a wider variety of hosts report server power consumption in comparable industry-standard ways. Interestingly, this has not been a frequent request from DPM users for two reasons: 1) many use hosts from the same hardware generation in a cluster, so the hosts' efficiency is similar, and 2) in clusters with mixed server generations, the newer servers are both larger and more efficient, so the existing sorting based on host size happens to coincide with host efficiency.

DPM's sorting could also consider host temperature; powering off hosts with measurably higher ambient temperature reduces the usage of hosts that could be more likely to fail.

Another future direction is to revise DPM (and DRS) to understand host and cluster power caps such as those in HP Dynamic Power Capping [1] and Intel Node Manager [3]. Power caps are a mechanism

to allow provisioning of power with respect to a specified peak value, which can limit the effective MHz capacity of the hosts governed by the cap. Given knowledge of the power cap and the power efficiency of the hosts, DPM and DRS can decide how to distribute VMs on hosts with the goal of meeting the VM's CPU resource needs in accordance with the caps.

Another area for investigation is to allow the user to tune how much consolidation is appropriate for DPM. For some users, consolidation based on CPU and memory demand is too aggressive, and they would prefer that the utilization metric be based on values derived for the VM configuration.

8. Related Work

Virtualization has introduced many interesting and challenging resource-management problems due to sharing of underlying resources and extra layers of indirection (see [26, 40, 48]). Success of a highly consolidated environment depends critically on the ability to isolate and provide predictable performance to various tenants or virtual machines. A recent study [22] performed a cost comparison of public vs. private clouds and distilled its results into the mantra *don't move to the cloud, but virtualize*. Part of the reason is the ability to better consolidate VMs in a private virtual datacenter as compared to a public cloud, yet a higher consolidation ratio requires better management of the shared infrastructure. This has led to a flurry of recent research by both industry and academia in building more efficient and scalable resource management solutions.

Many of the resource-management tasks have their roots in well-known, fundamental problems from the extensive literature on bin packing and job-scheduling. We classify the related literature in this area in two broad categories: *bin packing & job-scheduling literature* and *VM-based resource optimizations*. The first category covers classical techniques that are widely applicable in many scenarios and the second covers the algorithms and techniques specific to virtual machine based environments.

8.1 Bin Packing and Job Scheduling Algorithms

Bin packing is a well-known combinatorial NP-hard problem. Many heuristics and greedy algorithms have been proposed for this problem [17, 18, 24, 37, 47]. The simple greedy algorithm of placing the item in a bin that can take it or using a new bin if none is found, provides an approximation factor of 2. Heuristics such as first-fit decreasing (FFD) [54, 33], best-fit decreasing and MFFD [36] have led to competitive bounds of (11=9 *OPT* + 6=9) and (71=60 *OPT* + 1), where *OPT* is the number of bins given by the optimal solution.

The DRS problem is closer to multidimensional bin packing without rotation or multidimensional vector bin packing. This problem is even harder than the one-dimensional bin packing and many bounds have been proved in the literature [19, 20, 35, 38]. These bounds are less tight in most cases. Bansal and Sviridenko [20] showed that an asymptotic polynomial time approximation scheme does not exist for the two-dimensional case. Similar result has been proven for two-dimensional vector bin packing by Woeginger [51].

The problem faced by DRS and DPM is more complicated since many of the proposed techniques and lower bounds make several assumptions in terms of bin sizes and object dimensions. DRS cannot directly use such techniques because there is a cost of migration and optimizations need to be solved in an online manner. For online bin packing again, one can use algorithms such as Next-Fit, First-Fit [23] and Harmonic [41]. These algorithms are shown to have a worst-case performance ratio of 2, 1.7 and 1.636 respectively. DRS does not map directly to any such algorithm because it tries all possible one-step migrations and compares them using cost and benefit analysis. The cost-benefit analysis is very dependent on the underlying virtual machine migration technology and hypervisor overheads. But one can classify DRS as an online, greedy hill-climbing algorithm that evaluates all possible one-step moves, chooses the best one, and repeats this process after applying that move.

Fair job scheduling and fair queuing [21, 25, 55] techniques are also relevant to the DRS solution. Most of these techniques provide fairness across a single dimension or resource. In DRS, we use standard deviation as a measure of imbalance and combine standard deviation across multiple resources based on dynamic weights. These weights are again determined based on the actual resource utilization and we favor the balancing of the resource with higher utilization.

8.2 VM-based Resource Optimizations

More recently there has been considerable interest in automated resource scaling for individual virtual machines. For example, both [53] and [44] proposed a two-layered control architecture for adaptive, runtime optimization of resource allocations to co-hosted VMs in order to meet application-level performance goals. The former employs fuzzy logic and the latter adopts a feedback-control based approach.

These approaches are complementary to methods that utilize live VM migration in other work to alleviate runtime overload conditions of virtualized hosts. For example, in [39], the dynamic VM migration problem was solved using a heuristic bin packing algorithm, evaluated on a VMware-based testbed. In [52], it was discovered that using information from OS and application logs in addition to resource utilization helps the migration controller make more effective decisions.

Workload migration has also been utilized in other work to consolidate workloads onto fewer hosts in order to save power. In [27], a trace-based workload placement controller and a utilization-based migration controller were combined to minimize the number of hosts needed while meeting application quality of service requirements. In [45], a coordination framework was proposed to integrate five resource and power controllers from the processor level up to the data center level to minimize overall power consumption while preventing individual servers from being overloaded or exceeding power caps.

For private clouds, resource-management solutions like Microsoft PRO [14] provide a similar high-level functionality, but the details of their approach are not known. Similar approaches such as LBVM [5] have been proposed for Xen and OpenVZ-based virtualized environments. LBVM consists of a number of scripts that are configurable and has a very different architecture. It allows a

configurable load balancing rule per VM. It is unclear how well LBVM works in terms of overall cluster-level metrics. Neither Microsoft PRO nor LBVM support a rich resource model like DRS.

In an IaaS-based cloud many approaches have been proposed for elastic scaling and provisioning of VMs based on demand [32, 42, 46]. These techniques are complementary to DRS and can run as a service on top. Based on actual application monitoring one can either power-on more VMs (*i.e., horizontal scaling*) or change the settings of an individual VM (*i.e., vertical scaling*). DRS can then be used to place the new VMs more efficiently or to migrate VMs in order to respect new resource control settings.

9. Conclusions

In this paper, we presented the design and implementation of DRS (Distributed Resource Scheduler) along with our experience of improving it over the last five years. DRS provides automated management for a group of virtualized hosts by presenting them as a single cluster. DRS provides a very rich resource model in terms of controls such as reservations, limits, shares and resource pools in order to specify user intent in terms of performance isolation and prioritization. Various user operations like initial placement of virtual machines, load balancing based on dynamic load, power management and meeting business rules are carried out without any human intervention. We also presented the design of DPM (Distributed Power Management), which provides power savings without impacting application workloads.

Our extensive performance evaluation based on an in-house simulator and real experiments shows that DRS is able to increase CPU and memory consumption of VMs by performing automatic load balancing. Similarly, DPM is able to save power without negatively impacting running workloads. DRS has been shipping as a feature for the last six years and has been extensively deployed by our user base. Our own experiments and user feedback have led to several enhancements over the initial design. We highlighted some of these optimizations and demonstrated their benefit in improving the overall accuracy and effectiveness of DRS.

Acknowledgments

We would like to thank John Zedlewski, Ramesh Dharan, Andy Tucker, Shakari Kalyanaraman, Tahir Mobashir, Limin Wang, Tim Mann, Irfan Ahmad, Aashish Parikh, Chirag Bhatt, Ravi Soundararajan, Canturk Isci, Rajasekar Shanmugam and others for their contributions to DRS and related components in the overall system and for valuable discussions over all these years.

We are also grateful to Haoqiang Zheng, Rajesh Venkatasubramanian, Puneet Zaroo, Anil Rao, Andrei Dorofeev and others who have always listened patiently to our requests and happily provided the necessary support needed from ESX CPU and memory scheduling layers.

We are also indebted to Mike Nelson, Kit Colbert, Ali Mashtizadeh, Gabriel Tarasuk-Levin and Min Cai, for providing us with the core primitive of *vmotion* that DRS uses and enhancing it further over all these years. Without the help, support and guidance of all these people, DRS would not have been such a successful, fun and exciting project to work on.

10. References

[1] HP Power Capping and HP Dynamic Power Capping for ProLiant Servers. http://h20000.www2.hp.com/bc/docs/support/SupportManual/c01549455/c01549455.pdf.

[2] Integrated Dell Remote Access Controller (iDRAC). http://support.dell.com/support/edocs/software/smdrac3/idrac/.

[3] Node Manager – A Dynamic Approach To Managing Power In The Data Center. http://communities.intel.com/docs/DOC-4766.

[4] VMware vSphere. http://www.vmware.com/products/vsphere/.

[5] Load Balancing of Virtual Machines, 2008. http://lbvm.sourceforge.net/.

[6] VMware Fault Tolerance – Deliver 24 X 7 Availability for Critical Applications, 2009. http://www.vmware.com/files/pdf/VMware-Fault-Tolerance-FT-DS-EN.pdf.

[7] VMware High Availability: Easily Deliver High Availability for All of Your Virtual Machines, 2009. http://www.vmware.com/files/pdf/VMware-High-Availability-DS-EN.pdf.

[8] VMware vMotion: Live Migration for Virtual Machines Without Service Interruption, 2009. http://www.vmware.com/files/pdf/VMware-VMotion-DS-EN.pdf.

[9] VMware Distributed Power Management Concepts and Use, 2010. http://www.vmware.com/resources/techresources/1080.

[10] VMware Network I/O Control: Architecture, Performance, and Best Practices, 2010. http://www.vmware.com/resources/techresources/10119.

[11] VMware vCenter Server Performance and Best Practices, 2010. http://www.vmware.com/resources/techresources/10145.

[12] Advanced Configuration and Power Interface Specification, 2011. http://www.acpi.info/DOWNLOADS/ACPIspec50.pdf.

[13] Host Power Management in vSphere 5, 2011. http://www.vmware.com/resources/techresources/10205.

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

[15] VMware vShield Product Family, 2011. http://vmware.com/products/vshield/.

[16] VMware vSphere Storage Distributed Resource Scheduler, 2011. http://www.vmware.com/products/datacenter-virtualization/vsphere/vsphere-storage-drs/features.html.

[17] S. Albers and M. Mitzenmacher. Average-case analyses of first fit and random fit bin packing. In *In Proc. of the Ninth Annual ACM SIAM Symposium on Discrete Algorithms*, pages 290—299, 1998.

[18] B. S. Baker and J. E. G. Coman. A tight asymptotic bound for next-fit-decreasing bin-packing. *In SIAM J. Alg. Disc. Meth.*, pages 147—152, 1981.

[19] N. Bansal, A. Caprara, and M. Sviridenko. Improved approximation algorithms for multidimensional bin packing problems. In *IEEE Symposium on Foundations of Computer Science (FOCS '06)*, pages 697—708, 2006.

[20] N. Bansal and M. Sviridenko. New approximability and inapproximability results for 2-dimensional bin packing. In *In Proc. of the ACM SIAM Symposium on Discrete Algorithms,* pages 196{203, 2004.

[21] J. C. R. Bennett and H. Zhang. WF2Q: Worst-case fair weighted fair queueing. In *Proc. of INFOCOM '96*, pages 120—128, March 1996.

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

[23] E. G. Coman, M. R. Garey, and D. S. Johnson. Approximation algorithms for bin packing: A survey. In *Approximation Algorithms for NP-Hard Problems, D. Hochbaum (editor), PWS Publishing, Boston (1997), 46-93. A survey of worst- and average-case results for the classical one-dimensional bin packing problem,* 1997.

[24] W. F. de la Vega and G. S. Lueker. Bin packing can be solved within 1+epsilon in linear time. *Combinatorica*, 1(4):349—355, 1981.

[25] A. Demers, S. Keshav, and S. Shenker. Analysis and simulation of a fair queuing algorithm. Journal of *Internetworking Research and Experience*, 1(1):3{26, September 1990.

[26] U. Drepper. The cost of virtualization. *ACM Queue*, Feb. 2008.

[27] D. Gmach, J. Rolia, L. Cherkasova, G. Belrose, T. Turicchi, and A. Kemper. An integrated approach to resource pool management: Policies, efficiency and quality metrics. In *DSN'08*, June 2008.

[28] A. Gulati, I. Ahmad, and C. Waldspurger. PARDA: Proportionate allocation of resources for distributed storage access. In *Proc. Conference on File and Storage Technology (FAST '09)*, Feb. 2009.

[29] A. Gulati, C. Kumar, I. Ahmad, and K. Kumar. BASIL: Automated IO load balancing across storage devices. In *Proc. Conference on File and Storage Technology (FAST '10)*, Feb. 2010.

[30] A. Gulati, A. Merchant, and P. Varman. mClock: Handling throughput variability for hypervisor IO scheduling. In *9th Symposium on Operating Systems Design and Implementation (OSDI '10)*, October 2010.

[31] A. Gulati, G. Shanmuganathan, I. Ahmad, C. Waldspurger, and M. Uysal. Pesto: Online storage performance management in virtualized datacenters. In *Proceedings of the 2nd ACM Symposium on Cloud Computing (SOCC '11)*, 2011.

[32] A. Gulati, G. Shanmuganathan, A. Holler, and I. Ahmad. Cloud scale resource management: Challenges and techniques. In *USENIX HotCloud'11*, June 2011.

[33] Győrgy Dósa. The tight bound of First Fit Decreasing bin-packing algorithm is FFD(I)=(11/9)OPT(I)+6/9. In *ESCAPE*, 2007.

[34] H. Herodotou, F. Dong, and S. Babu. No one (cluster) size fits all: Automatic cluster sizing for data-intensive analytics. In *Proceedings of the 2nd ACM Symposium on Cloud Computing (SOCC '11)*, October 2011.

[35] M. L. J. Csirik, J. B. G. Frenk and S. Zhang. On the multidimensional vector bin packing. In *Acta Cybernetica*, pages 361—369, 1990.

[36] D. S. Johnson and M. R. Garey. A 71/60 theorem for bin packing. *J. Complexity*, 1(1):65—106, 1985.

[37] N. Karmarkar and R. M. Karp. An efficient approximation scheme for the one-dimensional bin-packing problem. *In IEEE Symposium on Foundations of Computer Science (FOCS '82)*, pages 312—320, 1982.

[38] R. M. Karp, M. Luby, and A. Marchetti-Spaccamela. A probabilistic analysis of multidimensional bin packing problems. In *ACM Symposium on Theory of Computing (STOC '84)*, pages 289{298, 1984.

[39] G. Khana, K. Beaty, G. Kar, and A. Kochut. Application performance management in virtualized server environments. In I*EEE/IFIP NOMS'06*, April 2006.

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

[41] C. C. Lee and D. T. Lee. A simple on-line bin-packing algorithm. *J. ACM*, 32:562—572, July 1985.

[42] C. Liu, B. T. Loo, and Y. Mao. Declarative automated cloud resource orchestration. In *Proceedings of the 2nd ACM Symposium on Cloud Computing (SOCC '11)*, October 2011.

[43] M. Nelson, B.-H. Lim, and G. Hutchins. Fast transparent migration for virtual machines. In *Usenix Annual Technical Conference (Usenix ATC '05)*, April 2005.

[44] P. Padala, K. Hou, X. Zhu, M. Uysal, Z. Wang, S. Singhal, A. Merchant, and K. Shin. Automated control of multiple virtualized resources. In *EuroSys '09*, April 2009.

[45] R. Raghavendra, P. Ranganathan, V. Talwar, Z. Wang, and X. Zhu. No power struggles: Coordinated multi-level power management for the data center. In I*nternational Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '08)*, March 2008.

[46] Z. Shen, S. Subbiah, X. Gu, and J. Wilkes. Cloudscale: Elastic resource scaling for multi-tenant cloud systems. In *Proceedings of the 2nd ACM Symposium on Cloud Computing (SOCC '11)*, October 2011.

[47] R. R. T. Batu and P. White. Fast approximate PCPs for multidimensional bin-packing problem. In *LNCS*, pages 245—256, 1999.

[48] W. Vogels. Beyond server consolidation. *ACM Queue*, Feb. 2008.

[49] C. A. Waldspurger. Memory resource management in VMware ESX Server. In *Proceedings of the Fifth Symposium on Operating Systems Design and Implementation (OSDI '02)*, Dec. 2002.

[50] C. A. Waldspurger and W. E. Weihl. Lottery scheduling: Flexible proportional-share resource management. In *1st Symposium on Operating Systems Design and Implementation (OSDI '94)*, 1994.

[51] G. J. Woeginger. There is no asymptotic ptas for two-dimensional vector packing. *Inf. Process. Lett.*, 64(6):293—297, 1997.

[52] T. Wood, P. J. Shenoy, A. Venkataramani, and M. S. Yousif. Black-Box and gray-box strategies for virtual machine migration. In *Symposium on Networked Systems Design and Implementation (NSDI '07)*, April 2007.

[53] J. Xu, M. Zhao, J. Fortes, R. Carpenter, and M. S. Yousif. Autonomic resource management in virtualized data centers using fuzzy logic-based approaches. *Cluster Computing Journal*, 11:213—227, 2008.

[54] M. Yue. A simple proof of the inequality FFD (L) 11/9 OPT (L) + 1, for the FFD bin-packing algorithm. In *ACTA MATHEMATICAE APPLICATAE SINICA, Volume 7, Number 4*, pages 321—331, October 1991.

[55] L. Zhang. VirtualClock: A new traffic control algorithm for packet-switched networks. *ACM Trans. Comput. Syst.*, 9(2):101{124.