Practical Compute Capacity Management for Virtualized Datacenters

Mukil Kesavan, Irfan Ahmad, Orran Krieger, Ravi Soundararajan,
Ada Gavrilovska, Karsten Schwan

Abstract—We present CCM (Cloud Capacity Manager) – a prototype system and its methods for dynamically multiplexing the compute capacity of virtualized datacenters at scales of thousands of machines, for diverse workloads with variable demands. Extending prior studies primarily concerned with accurate capacity allocation and ensuring acceptable application performance, CCM also sheds light on the tradeoffs due to two unavoidable issues in large scale commodity datacenters: (i) maintaining low operational overhead given variable cost of performing management operations necessary to allocate resources, and (ii) coping with the increased incidences of these operations' failures. CCM is implemented in an industry-strength cloud infrastructure built on top of the VMware vSphere virtualization platform and is currently deployed in a 700 physical host datacenter. Its experimental evaluation uses production workload traces and a suite of representative cloud applications to generate dynamic scenarios. Results indicate that the pragmatic cloud-wide nature of CCM provides up to 25% more resources for workloads and improves datacenter utilization by up to 20%, compared to the common alternative approach of multiplexing capacity within multiple independent smaller datacenter partitions.

Index Terms—Cloud computing, virtualization, performance evaluation, fault-tolerance, metrics/measurement, resource management

1 Introduction

Multi-tenant cloud computing datacenters run diverse workloads with time varying resource demands inside virtual machines (VMs). This continuously leads to areas of high and low resource utilization throughout the cloud infrastructure. By dynamically multiplexing available compute capacity between workloads and correcting resource imbalances, significant improvements in application resource availability, and, as a consequence, overall datacenter utilization can be achieved. Such capacity multiplexing is typically implemented using a combination of per-server resource schedulers to address short-term needs and through VM live migrations to address long-term changes in resource needs. However, architecting a realistic multiplexing solution, at the scale of thousands of machines, across an entire datacenter, is a challenging proposition.

- *M. Kesavan, A. Gavrilovska and K. Schwan are with the College of Computing, Georgia Institute of Technology, Atlanta, GA, 30332. E-mail: mukil@cc.gatech.edu, schwan@cc.gatech.edu, ada@cc.gatech.edu.*
- *Irfan Ahmad is with CloudPhysics, Inc., 201 San Antonio Rd, Suite 135, Mountain View, CA, 94040. E-mail: irfan@cloudphysics.com.*
- *Orran Krieger is with the Department of Computer Science, Boston University, 1 Silber Way, Boston, MA, 02215. E-mail: okrieg@bu.edu.*
- *Ravi Soundararajan is with VMware, Inc., 3401 Hillview Ave, Palo Alto, CA, 94304. E-mail: ravi@vmware.com.*

Despite years of research and development, commercially available infrastructure resource allocation software such as Microsoft PRO, VMware DRS, and Citrix XenServer currently only support dynamic allocation for a set of 32 or fewer hosts [1], [2], [3]. Gulati et. al. summarize the potential reasons underlying such scale restrictions, including overhead, machine heterogeneity, resource partitions, fault-tolerance, etc. [4]. Consequently, datacenter administrators are forced to use dynamic allocation within multiple smaller independent partitions in order to cover their entire infrastructure[1] [6], [7], while relying on some intelligent or random assignment of workloads to these partitions to somewhat balance resource usage [8]. But such partitioning foregoes opportunities to multiplex capacity across the entire datacenter, leading to decreased utilization levels. It also limits responses to sudden increases in workload demand to the capacity available within each partition, even when there is unused capacity elsewhere, thereby negatively impacting application performance, as well.

The focus of much of the research done in this area has been on the development of accurate methods for workload demand prediction and allocation, and, application service level agreement (SLA) compliance [9], [10], [11], [12], [13], [14]. Most of these systems are developed for specific applications models (e.g. map-reduce,

1. Popular infrastructures like Amazon EC2 do not seem to employ live migration of VMs at all, a feature required to dynamically allocate capacity across many servers [5].

Digital Object Indentifier 10.1109/TCC.2013.8

N-tier web applications, etc.) and small-scale environments where the overhead of exhaustive, but accurate, allocation methods can be tolerated. However, given the steadily increasing number and variety of cloud applications [15], datacenter administrators and infrastructure-as-a-service (IaaS) providers need *application-model agnostic, low-level capacity management solutions that provide quality of service (QoS) abstractions across the entire cloud*, to facilitate the construction of higher level application specific SLA enforcement services, as required.

In this paper, we present such a system – CCM (Cloud Capacity Manager) – an on-demand compute capacity management system that combines various low-overhead techniques, motivated by practical on-field observations, to achieve scalable capacity allocation for thousands of machines. CCM achieves this scale by employing a 3-level hierarchical management architecture. The capacity managers at each level continuously monitor and aggregate *black-box* VM CPU and memory usage information and then use this aggregated data to make *independent* and localized capacity allocation decisions.

The use of black-box monitoring and allocation enables CCM to perform capacity allocation for a broad class of applications including those that don't typically scale out horizontally but yet vary dramatically in the amount of capacity they use over time [16]. In addition to allocating capacity purely based on VMs' runtime resource demands, CCM also offers absolute minimum, maximum and proportional resource allocation QoS controls for each VM, interpreted across the entire cloud. These abstractions enable the safe multiplexing of capacity while still meeting workload performance obligations.

The core concept embodied in CCM's capacity multiplexing is the on-demand balancing of load, across logical host-groups, at each level in the hierarchy. Reductions in management cost are obtained by having monitoring and resource changes occur at progressively less frequent intervals when moving up the hierarchy, i.e. at lower vs. higher level managers, the reasons for which are explained in Section 3.

The cost of reconfiguration, i.e., performing VM migrations to re-allocate resources, at large scales is fairly expensive [17] and non-deterministic [18]. Further, as we show in this work, large-scale capacity multiplexing systems are acutely limited in the number of reconfiguration operations they can successfully carry out at any given time, which influences the effectiveness of any potential allocation method used in this environment.

CCM takes this into account in its design and, hence, uses simple greedy hill-climbing algorithms to iteratively reduce load imbalances across the hierarchy in lieu of more complex approaches. It also uses management operation throttling and liberal timeouts, to select cheaper management actions, and, minimize the incidence of management failures due to resource insufficiency as explained in Section 4. Such simple methods are easier to scale, develop and debug, and, their effectiveness is shown through detailed large scale experiments.

In order to properly characterize CCM's scalability and resilience to the aforementioned practical issues in real-life environments, we deploy and evaluate CCM on a 700 physical host datacenter, virtualized with the VMware vSphere [19] virtualization platform. We generate realistic datacenter load scenarios using a combination of a distributed load generator [20] and additional application workloads. The load generator is capable of accurately replaying production CPU and memory usage traces, and simulating resource usage spikes at varying scales. The application workloads include Cloudstone [21] – a 3 tier web 2.0 application, Nutch [22] – a mapreduce based crawler, Voldemort [23] – a key-value store and the Linpack high performance computing benchmark [24].

In the remainder of the paper, Section 2 presents background on the vSphere QoS abstractions that can be used to achieve desired isolation and sharing objectives at smaller scales. These abstractions are adopted and extended by CCM in the cloud context. Section 3 presents CCM's black-box metrics computed for dynamically allocating capacity to application VMs at cloud scale, the assumptions we make, and its load balancing algorithms. Section 4 outlines CCM's implementation, with a focus on methods for dealing with management failures. Section 5 presents a detailed experimental evaluation. Related work in Section 6 is followed by a description of conclusions and future work.

2 vSphere Basics

VMware vSphere [19] is a datacenter virtualization platform that enables infrastructure provisioning and virtual machine lifecycle management. CCM relies on VMware vSphere platform features that include (i) Distributed Resource Scheduler (DRS) [25] (ii) Distributed Power Management (DPM) [26]. These features are employed at the lower levels in the CCM hierarchy for management of smaller scale clusters of hosts. Hosts are arranged into logical clusters, each a collection of individual physical hosts. The overall resource capacity of a cluster is the sum of all individual host capacities. With vSphere, a central server offers these features for a cluster of at most 32 hosts. We refer the reader to [27] for a detailed discussion of the design of DRS and DPM.

DRS Capacity Allocation. DRS automates the initial and continuing placement of VMs on hosts based on the following resource QoS controls.

Reservation (R): a per resource minimum absolute value (in MHz for CPU and MB for memory) that must always be available for the corresponding VM. To meet this requirement, VM admission control rejects VMs with reservations when the sum of powered on VMs' reservations would exceed total available cluster capacity.
Limit (L): a per resource absolute value denoting the maximum possible allocation of a resource for the VM; strictly enforced even in the presence of unused capacity.
Share (S): a proportional weight that determines the fraction of resources a VM would receive with respect to the total available cluster capacity and other resident VMs' shares; allows graceful degradation of capacity allocation under contention.

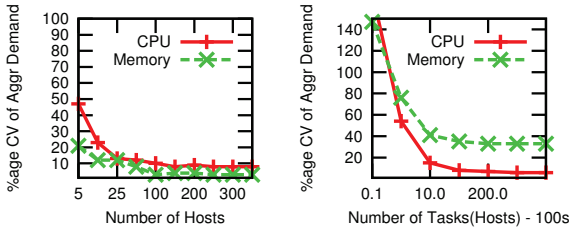
Details on how to choose appropriate values for these controls, based on application QoS needs, appear in [28]. For each VM, DRS computes a per-resource *entitlement* value as a function of its own and other VMs' resource controls and demand, and, also the total available cluster resource capacity. VM demand is estimated based on both its present and anticipated future resource needs: computed as its average resource usage over a few minutes plus two standard deviations. Using the entitlement values, DRS computes a set of VM to host associations and performs migrations as necessary.

The CCM algorithms, described in the following section, extend the interpretation and enforcement of these resource control abstractions to cloud scales of thousands of hosts.

DPM Power Management. Whenever a cluster host's utilization value falls below a specified threshold, DPM performs a cost benefit analysis of all host power downs in the entire cluster, so as to raise the host utilization value above the threshold by accepting the VMs from the powered down hosts [27]. The outcome of this analysis is a per-host numeric value called *DPMScore*. A higher DPMScore denotes greater ease with which a host can be removed from operation in a cluster. CCM uses this value to aid in host reassociation between clusters and superclusters, as explained next.

3 CCM: System Design

It is well-known that clustering and hierarchies can help with scalability [29] by reducing the overheads of system operation. More importantly, in the capacity allocation context, as resource consumption of workloads is aggregated across larger and larger numbers of physical machines, as one moves upwards from lower levels in the hierarchy, there is the possibility of decreased degrees of variation in this aggregate consumption. The intuition here is that across a large set of servers in a typical multi-tenant cloud, the individual customer workload resource consumption patterns are not likely temporally correlated, i.e., their peaks and valleys don't coincide. As a result, there could be substantial opportunity for workload multiplexing, which increases with the size and the diversity of the workload [11]. *In terms of its system design implication, this permits monitoring and management operations to achieve low overhead by running at progressively larger time scales, when moving up the hierarchy, without substantive loss in the ability to meet changing workload resource demands.*



(a) VMware vSphere Cluster (b) Google Production Cluster

Fig. 1: Variation in Aggregate Demand over increasing scales from production traces.

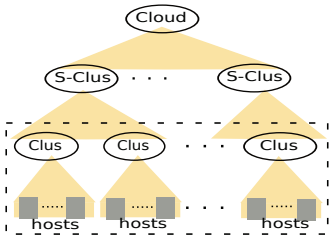


Fig. 2: Hosts are grouped into a logical cluster with a cluster-level capacity manager (VMware DRS), clusters are grouped into a supercluster with a corresponding capacity manager and a cloud is composed as a collection of superclusters under a cloud level capacity manager.

We reinforce this idea by analyzing the aggregate resource consumption in two production datacenter traces from: (a) an enterprise application cluster running the VMware vSphere [19] datacenter virtualization platform and (b) a publicly available trace from Google's production cluster [30] (version 1). Figure 1 presents the coefficient of variation (as percentage) in aggregate CPU and memory consumption across larger and larger groupings of physical hosts, in each trace. The graphs show that the coefficient of variation, for both CPU and memory resources, decreases roughly for each order of magnitude increase in the number of hosts considered for aggregation.

The above factors, therefore, motivate the natural fit of a hierarchical management architecture to capacity allocation. Hierarchies can also be defined so as to a) match underlying structures in the datacenter in terms of rack boundaries, performance or availability zones, different machine types, etc., or to b) capture structure in the application such as co-locating frequently communicating VMs in the same part of the infrastructure.

We plan on exploring this in our future work.

The CCM architecture, shown in Figure 2, is organized as a hierarchy of a top-level cloud manager, mid-level supercluster managers, and finally cluster managers at the lowest level. Per-VM management takes place only at cluster level. At this level, VMware DRS is used to make independent and localized decisions to balance loads as briefly explained in Section 2. At higher levels, i.e., supercluster and cloud, CCM computes the total estimated demand of a cluster or supercluster, respectively, as an *aggregation* of per-VM resource usage, i.e., the total demand of all VMs running under the given lower-level management entity in the hierarchy. This total demand estimate is then used to determine the amount of capacity required by a cluster or supercluster and to perform coarse grain capacity changes, as necessary. The capacity changes are realized by *logically reassociating* hosts between cluster and supercluster managers. A logical host reassociation or *a host-move operation* is a two step process where the VMs running on a chosen host are first live migrated to other hosts in the *original source cluster* and then the source and destination managers' inventories are updated to reflect the change. This operation, however, preserves VM-to-capacity-manager associations across the hierarchy. The benefit of this property and the rationale behind its choice are explained in detail in Section 3.1. Such coarse-grained higher level capacity changes, then, automatically trigger the cluster-level re-balancing actions of DRS. Finally, for reasons cited above, both the monitoring and resource changes in our system occur at progressively less frequent intervals when moving up the hierarchy.

Metrics. The following key metrics are computed and used by CCM algorithms at the supercluster and cloud levels of the management hierarchy to determine the capacity to be allocated to these entities, based on their aggregate demand and R, L, S constraints.

Entitlement(E): For each VM, per-resource entitlements are computed by DRS (see Section 2). We extend the notion of entitlement to entire clusters and superclusters, to express their aggregate resource demand. There are no R, L, S parameters for clusters and superclusters. This entitlement calculation is explained in detail in Section 3.3.

For the CCM hierarchy, an aggregate notion of demand is computed by successively aggregating VM-level entitlements for clusters and superclusters. Upper-level capacity managers, then, allocate capacity between clusters and superclusters using such aggregate information instead of considering individual VMs, while lower-level cluster managers continue to independently decide and perform detailed allocations.

We note that when aggregating entitlements, some additional head room is added, worth the last two standard deviations in resource usage, so as to absorb minor transient usage spikes and also give VMs the ability to indirectly request more resources from the system, by consuming the extra head-room over longer periods. This results in the successive addition of some amount of excess resource to the computed entitlement as aggregation proceeds up the hierarchy. The effect is that even with coarser sample/estimation intervals, there is a high likelihood of being able to cover a cluster's or supercluster's varying resource demands. The resource volume used for this head-room is directly proportional to the amount of variability in resource demands – i.e., when demands are stable, less resources are added, and vice versa.

Normalized Resource Entitlement(NE): is a measure of the utilization of available capacity. For a given entity, e.g., cluster or supercluster, it is defined as the sum of the total amount of a resource entitled to that entity, divided by the entity's total resource capacity. Thus, NE captures resource utilization, ranging from 0 to 1.

Resource Imbalance(I): captures the skew in resource utilizations across a set of similar entities at the same level in the resource management hierarchy. It is defined as the standard deviation of individual normalized resource entitlements (NE) of the entities in consideration. High I and NE values suggest the need for capacity reallocation between entities to better back observed VM demand.

DPMRank: is a cluster-level aggregation of the individual host DPMScore values. A high DPMRank metric for a cluster indicates its suitability to cheaply donate hosts to other overloaded clusters, or superclusters in the event of a higher layer load imbalance. It is computed as the squared sum of individual host DPMScores, so that the metric favors clusters with a small number of hosts with very low resource utilization vs. those that have large numbers of relatively modestly utilized hosts. For example, a cluster of 3 hosts with a DPMScore of 10 each will be favored over a cluster of 10 hosts with DPMScores of 3. The reason, of course, is that the cost of reassociating a host with DPMScore of 10 is less than that of a host with DPMScore of 3.

3.1 Logical Capacity Reassociation

As mentioned before, CCM logically reassociates capacity between cluster and supercluster managers. We chose such reassociation based on several practical considerations. First, in any given large scale system, hosts fail and recover over time, causing these systems to incorporate capacity changes in their design, thereby making dynamic host addition and removal a standard mechanism. Second, VM migration would also require moving substantial state information about each such

VM accumulated at the corresponding capacity managers. This includes statistics computed about the VM's resource usage over the course of its existence, runtime snapshots for fail-over, configuration information, etc. Migrating this state along with the VM to the new capacity manager, would be costly, and it would potentially seriously prolong the delay experienced until the VM and capacity management once again become fully operational. While we do have mechanisms to migrate VMs across clusters, we found host reassociation to be a simpler solution from a design and implementation standpoint.

For the reasons above, CCM preserves the association of VMs to clusters and superclusters, and it manages capacity by logically reassociating *evacuated hosts* between them. Host reassociation uses the DPMRank value to identify clusters with hosts for which evacuation costs are low, and once a host has been moved to its new location, the lower level resource managers (DRS/ESX) notice and automatically take advantage of increased cluster capacity by migrating load (i.e., existing VMs) onto the new host. In this fashion, CCM achieves a multi-level load balancing solution in ways that are transparent to operating systems, middleware, and applications.

3.2 Assumptions

In the construction of CCM, we make two basic assumptions about datacenters and virtualization technologies. (1) Hosts are assumed uniformly compatible for VM migration; this assumption could be removed by including host metadata in decision making. (2) Storage and networking must be universally accessible across the cloud, which we justify with the fact that there already exist several instances of large scale NFS deployments, and VLANs that are designed specifically to facilitate VM migration across a large pool of hosts. Further assistance can come from recent technology developments in networking [31], [32], [33], along with the presence of dynamically reconfigurable resources like storage virtualization solutions, VLAN remapping, switch reconfiguration at per VM level, etc. Finally, these assumptions also let us carefully study the impact of management operation failures and costs on system design.

3.3 Capacity Management Algorithms

Table 1 summarizes the metrics described previously. CCM's cloud-scale capacity management solution has three primary allocation phases: (i) Initial Allocation, (ii) Periodic Balancing, and (iii) Reactive Allocation. The initial allocation recursively distributes total available cloud capacity among superclusters and clusters based solely on the underlying VMs' static resource allocation constraints: Reservation, Limit and Shares. Once this phase completes, the periodic balancing phase is activated across the hierarchy; it continually monitors and rectifies resource utilization imbalances, i.e., the presence of high and low areas of resource contention. Finally, in order to deal with unexpected spikes in resource usage, CCM uses an additional reactive allocation phase, which is triggered whenever the resource utilization of an entity exceeds some high maximum threshold (e.g., 80%). Reactive allocation quickly allocates capacity using a tunable fraction of idle resources set aside specifically for this purpose. In this manner, the three phases described allow for the complete automation of runtime managment of datacenter capacity.

L	Entitlement	Norm. Ent	Imbalance
C	$E_{VM} \leftarrow f(R,L,S,Cap(CL))$ $E_C = \sum E_{VM}$	$NE_H = \frac{\sum_H E_{VM}}{Cap(H)}$	$I_C = \sigma(NE_H), \forall H$
SC	$E_C^{SC} = E_C + 2 * \sigma(E_C)$ $E_{SC} = \sum E_C^{SC}$	$NE_C = \frac{E_C^{SC}}{Cap(C)}$	$I_{SC} = \sigma(NE_C), \forall C$
CL	$E_{SC}^{CL} = E_{SC} + 2 * \sigma(E_{SC})$	$NE_{SC} = \frac{E_{SC}^{CL}}{Cap(SC)}$	$I_{CL} = \sigma(NE_{SC}), \forall SC$
Cluster	$DPMRank_C = \sum (DPMScore_H)^2$		

TABLE 1: Resource Management Metrics Across All Levels in the Hierarchy. Key: R = Reservation, L = Limit, S = Share, Cap = Capacity, H = Host, C = Cluster, SC = SuperCluster and CL = Cloud. $Metric_X^Y$ denotes value of "Metric" for an entity at level "X" computed at an entity at the next higher level "Y". $Metric_X$ implies "Metric" for an entity at level "X" computed at the same level. I_X denotes the Imbalance metric computed across all sub-entities of level "X".

(i) Initial Allocation. The amount of a resource to be allocated to a cluster or supercluster is captured by the *entitlement* metric. For an initial allocation that does not yet have runtime resource demand information, the entitlement value is computed using only static allocation constraints. As capacity allocation proceeds across the cloud, three types of VMs, in terms of resource allocation, will emerge: (a) Reservation-Based VMs (R-VMs), (b) Limit-Based VMs (L-VMs), and (c) Share-Based VMs (S-VMs). A R-VM requires more of a given resource due to its specified resource reservation being larger than the amount of resources it would have been allocated based only on its proportional shares, i.e., $E_{VM} = R_{VM}$. Similarly, a L-VM requires less of a given resource due to its resource limit being lower than the amount of resources it would have been allocated based on its proportional shares alone, i.e., $E_{VM} = L_{VM}$. The VMs that don't fall into either category have their capacity allocations determined by their Shares value – S-VMs.

For example, consider four VMs A, B, C, and D

contending for a resource, all with equal shares. The amount of resources they would receive based on their shares alone is 25% each. Now, if A has a reservation of 40% for the resource and B has a limit of 20%, the resulting allocation for VMs A and B would be 40% and 20%, respectively. VM A, then, is a R-VM, whereas VM B is a L-VM. VMs C and D are S-VMs, and they share the remaining resources (after A's and B's constraints have been enforced) at 20% each.

Therefore, R-VMs reduce the total amount of resources available to be proportionally shared between S-VMs, and L-VMs have the opposite effect. The process of initial allocation, then, involves the identification of the three classes of VMs and then computing the final entitlement values. Across the cloud, R-VMs would be entitled to their reservation values, L-VMs would be entitled to their limit values, and the remaining capacity is used to compute the entitlement for S-VMs.

In order to explain the resource entitlement calculation for S-VMs, we first define a metric for these VMs, called *entitlement-per-share (EPS)*, as ρ such that, $\rho = E_{VM}/S_{VM}$. Since a VM's resource proportion is computed using its share with respect to the share values of *all* of the other VMs in the cloud, the EPS value stays the same for every Share-Based VM. Now, a given S-VM's entitlement can be computed as $E_{VM} = \rho * S_{VM}$. Thus, the complete equation to compute the entitlement of a specific resource of a VM is given by Equation 1.

The goal of initial capacity allocation is to ensure that *all of the available capacity* in the cloud is allocated to VMs in lieu of their allocation constraints as in Equation 2. We simplify the determination of capacity to be allocated to VMs, i.e., entitlement, via a recursive allocation algorithm. The algorithm performs a binary search over all possible assignments for ρ such that Equation 2 is satisfied throughout the process of computing entitlements using Equation 1. VM-level entitlements are aggregated across levels of the hierarchy. There can only be one unique value for ρ that ensures that this sum exactly equals the total available resource capacity of the cloud [34].

$$E_{VM} = MIN(MAX(R_{VM}, S_{VM} * \rho), L_{VM}) \quad (1)$$

$$\sum E_{VM} = Capacity(Cloud) \quad (2)$$

$$E_{VM} = MIN(MAX(R_{VM}, MIN(S_{VM} * \rho, D_{VM}))), L_{VM}) \quad (3)$$

(ii) Periodic Balancing. As mentioned before, the periodic balancing algorithm typically runs increasingly infrequently at lower vs. higher levels. Since the granularity of this interval impacts the overhead and accuracy of the CCM capacity balancing solution, administrators are given the ability to configure the resource monitoring and algorithmic intervals to individual deployment scenarios. The VM entitlement calculation now uses estimated runtime resource demand (D_{VM}) information together with the resource controls as shown in Equation 3. Load balancing between different hosts in a cluster is provided by the DRS software.

For a given set of clusters or superclusters, when the maximum value of the normalized resource entitlement of the set, for a given resource, exceeds an administrator-set threshold and, simultaneously, the resource imbalance across the set exceeds a second administrator-specified threshold, the CCM capacity balancing algorithm incrementally computes a series of host reassociations across the set to try to rectify the imbalance, up to a configured upper limit (explained in Section 4). The first condition ensures that CCM does not shuffle resources unnecessarily between entities of the same level in the hierarchy when the overall utilization is low although the imbalance is high. Once it has been decided to rectify the imbalance, hosts are moved from entities with lowest normalized resource entitlements (and higher DPMRanks) [2] to those with highest normalized resource entitlements. This results in the greatest reduction in resource imbalance. When removing hosts from an entitity, we always ensure that its capacity never goes below the amount needed to satisfy running VMs' sum of reservations (R). When selecting particular hosts to be moved from a cluster or supercluster, hosts that have the least number of running VMs are preferred.

(iii) Reactive Allocation. In order to deal with unexpected spikes in resource usage, CCM uses an additional reactive allocation phase, which is triggered whenever the resource utilization of an entity exceeds some high maximum threshold (e.g. 80%). It is typically invoked in the space of a few minutes, albeit progressively infrequently (up to an hour at the cloud level), as one moves upwards in the management hierarchy. But, as seen in Section 3, macro-level spikes across larger and larger groupings of physical hosts are increasingly unlikely to occur, especially at small time scales. To aid in the quick allocation of hosts to resource starved clusters, we maintain a per supercluster *central free host pool* that holds a tunable fraction of DPM module recommended, pre-evacuated hosts from across clusters that belong to the supercluster. This also removes the need for the host selection algorithm to be run. CCM currently only holds otherwise idle resources in the central free host pool, but simple modifications to the scheme could allow holding hosts even when the system is being moderately utilized. If there are no hosts in the free host pool or if the hosts currently present are insufficient for absorbing the usage spike, the periodic algorithm has to perform the remaining allocation in its next round.

2. In the actual realization, we sort in descending order of the sum $0.5 * (1 - NE) + 0.5 * NormalizedDPMRank$. Both normalized resource entitlement and NormalizedDPMRank are given equal weights.

There are some notable differences between the reactive algorithm running at the supercluster manager vs. the cloud manager. The cloud manager pulls from and deposits to the per supercluster central free host pool as opposed to having to pick a specific sub-cluster. This optimization serves to reduce overhead and will not affect the operation of the supercluster manager. The supercluster manager only maintains the administrator specified fraction of hosts in the free host pool and the rest are automatically distributed among clusters.

4 Prototype Implementation

CCM is implemented in Java, using the vSphere Java API [35] to collect metrics and enforce management actions in the vSphere provisioning layer. DRS is used in its standard vSphere server form, but for DPM, we modify the vSphere server codebase to compute the DPMRank metric and export it via the vSphere Java API. For simplicity of prototyping and algorithm evaluation, both the cloud manager and the supercluster manager are implemented as part of a single multithreaded application run on a single host.

4.1 Management Fault-tolerance and Cost Amortization

In order to improve the efficacy of CCM, an important element of CCM's implementation is the need to deal with non-determinism in operation completion times, as well as to handle failures of management operations. Regarding completion times, consider the host-move operation. In addition to performing VM migrations within a cluster, a *host-move* is one of the basic management operations performed by CCM at the supercluster and cloud levels, the purpose being to elastically allocate resources across the datacenter.

There are 2 major kind of host-moves: (i) host-move between clusters, and (ii) host-move between superclusters. The host evacuation step, common to both move cases, places a host into a "maintenance" state in which no VMs currently use it, so that this host can then be moved from one cluster or supercluster to another. This step is potentially the most resource intensive part of a host-move operation due to the need to evict all VMs currently running on the host. The resource intensity and thus, the duration of the operation is governed by factors that include VM size and active memory dirtying rate [36]. For example, Figure 3 shows the average time taken for a single host-move operation for a particular run of the experiment in Scenario I in Section 5. It can be seen that the duration varies between a wide range of 44 seconds to almost 7 minutes. This fact complicates the amortization of management overhead subject to the workload benefits being derived.

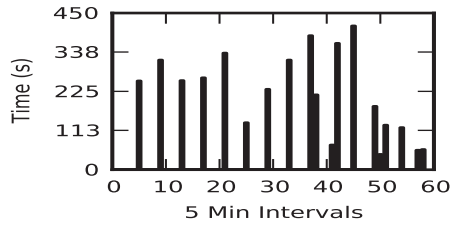


Fig. 3: Avg Single Host Move Latency.

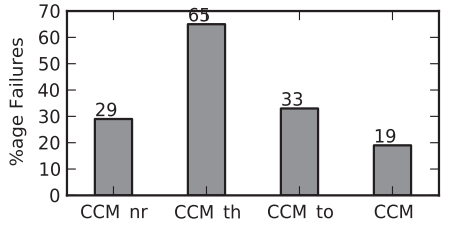


Fig. 4: Migration failures.

In addition to the variable cost of management operations, we've also observed significant number of VM live migration failures, between 19% to 65%, at scale, for different experimental runs with different configurations of CCM (explained below), as shown in Figure 4. Note that failure to migrate away VMs would also result in failure of the compound host-move operation leading to sub-par capacity multiplexing. Error logs gathered in our environment show diverse factors causing such failures. However, a significant portion of the error reports suggested a link between failures and higher live migration load subject to infrastructure resource availability (e.g., network bandwidth, CPU at source, etc.) i.e. *higher management resource pressure*.

CCM uses several methods to address the above issues, with a focus on *simplicity*, so as to be practicable at large scales. In order to contain management resource pressure at any given time, and to avoid the failures seen in Figure 4, we offer administrators the ability to tailor the system's management action enforcement aggressiveness using: (a) explicit throttling of management enactions, i.e., the number and parallelism of host-moves during each periodic balancing round, and (b) automatically aborting long running actions using timeouts, with partial state-rollbacks for consistency. Aborting long running management operations helps to avoid inflating the management overhead and taking away useful resources from applications. It also permits the potential exploration of alternative, cheaper actions at a later time. This also leads to effective management, as indicated by the results shown in this paper, without demanding expensive fine-grained monitoring of workload VMs at large scales, otherwise required if trying to proactively predict their migration costs [37].

Finally, the management algorithms at each manager

Metric	NR	TH	TO	All
Total Host-move Mins	4577	703	426	355
Successful Host-moves	14	24	30	37
emat (moves/hr)	0.18	2.05	4.23	6.25

TABLE 2: Management effectiveness. Key: NR - No Restrictions, TH - Only Host-move Thresholds, TO - Only Host-move Timeouts, All - TH + TO.

are not employed when there is insufficient monitoring information. Transient monitoring data "black-outs", although rare, happen in our environment due to temporary network connectivity issues, monitoring module crashes, etc. Also, failed multi-step operations use asynchronous re-tries for some fixed number of times before declaring them to have failed.

We now quantify the effectiveness of the simple host-move timeout and thresholding abstractions by turning them on one-by-one in the CCM prototype, resulting in four different system configurations. Table 2 briefs these configurations and shows the total amount of time spent in making all host-moves (including failed ones, counting parallel moves in serial order) and the number of successful host-moves, in each case. The workload and configuration parameters used in the experiment are described in detail in Scenario I in Section 5.

It can be seen that using a combination of limiting host-moves and using abort/retry, CCM exhibits higher management action success rates and lower resource costs (directly proportional to the total enforcement time) in our datacenter environment. This leads us to a direct formulation of a new management goodput metric – *effective management action throughput (emat)* – with which it is possible to more objectively compare the different configurations:

$$emat = \frac{num\_successful\_host\_moves}{total\_time\_spent\_in\_making\_ALL\_moves} \quad (4)$$

The emat metric, shown in the final row of Table 2, sets a two- to six-fold advantage of using both thresholding and abort/retry, over the other configurations. This result underscores the limitations imposed by practical issues in large scale setups and how simple methods that *explicitly design for these restrictions* can lead to better overall capacity allocation outcomes.

5 Experimental Evaluation

The experimental evaluations described in this section show: (i) that CCM is effective in keeping a cloud load balanced, by shuffling capacity based on demand; (ii) this reduces resource consumption hotspots and increases the resources available to workloads, and as a consequence, improves overall datacenter utilization; and (iii) CCM incurs operational overheads commensurate with the dynamism in the total workload. We measure system overhead as the number of VM migrations performed. Migrations are the primary and biggest contributor, in terms of resource usage and time, to the cost of capacity management. The numbers reported in this section are averaged over 3 runs of each experiment for the first two scenarios and 4 runs for the final scenario. We furnish information on the variability in performance where it is non-trivial (i.e. $> 1\%$).

In the absence of a datacenter-wide capacity multiplexing solution, administrators typically resort to statically partitioning their servers and employing traditional capacity multiplexing solutions within each partition [6], [7]. This strategy, which we refer to as partitioned management (PM), forms the basis for comparing CCM's performance and overheads. We emulate such a strategy by using VMware DRS inside each partition to continuously migrate VMs amongst the machines in the partition in response to load changes. CCM, then, naturally builds upon and extends this strategy with techniques explicitly designed to deal with issues at scale. This makes it easier to adopt in existing deployments.

Testbed and Setup: CCM operates on a private cloud in a 700 host datacenter. Each host has 2 dual core AMD Opteron 270 processors, a total memory of 4GB and runs the VMware vSphere Hypervisor (ESXi) v4.1. The hosts are all connected to each other and 4 shared storage arrays of 4.2TB total capacity via a Force 10 E1200 switch over a flat topology. The common shared storage is exported as NFS stores to facilitate migrating VMs across the datacenter machines. The open-source Cobbler installation server uses a dedicated host for serving DNS, DHCP, and PXE booting needs. VMware's vSphere platform server and client are used to provision, monitor, and partially manage the cloud.

Two important infrastructure limitations in this testbed influence our experiment design: (i) each server has a fairly low amount of memory compared to current datacenter standards, so that over-subscribing memory has non-negligible overhead [38], and (ii) each server has bandwidth availability of approximately 366 Mbps during heavy use, due to a relatively flat but somewhat underprovisioned network architecture, able to efficiently support VM migration only for VMs configured with moderate amounts of memory. As a consequence, most experiments are designed with higher CPU resource vs. memory requirements. Capacity allocation with respect to one resource proceeds as long as the imbalance or maximum utilization of other resource(s) do not cross their respective thresholds.

The private cloud used in experiments is organized as follows. The total number of hosts is divided into 16 partitions in the PM case, with an instance of VMware DRS

managing each partition. The DRS instances themselves run external to the partitions, on four individual hosts, as part of corresponding vSphere server instances. CCM builds on these base partitions or clusters by managing each set of 4 clusters via a supercluster manager (4 total). All of the supercluster managers, in turn, come under the purview of a single cloud level manager. The cloud level and supercluster level managers of CCM are deployed on a dedicated physical machine running Ubuntu Linux 9.10. Figure 2 from Section 3 shows the overall logical organization of both PM and CCM. The organization for PM appears inside the dashed rectangle.

In terms of the cloud-wide QoS controls settings exposed by CCM, we predominantly use the same proportional Shares for all the VMs in our environment with no Reservation and Limit, unless otherwise noted in specific circumstances in Scenario III. The use of high Reservation and Limit settings on the VMs would reduce the flexibility of dynamically multiplexing capacity for both the CCM and PM cases. Since our goal is to study the multiplexing aspect of system design, we use only proportional Shares.

5.1 Workloads

Trace-driven Simulation: the Xerxes distributed load generator [20] produces global, datacenter-wide CPU and memory usage patterns. It is organized as a collection of individually operating, per-VM load generators, supervised by a master node. The master takes in a global load specification, translates it to per-VM load specification, and sets up the simulation. Once the simulation starts, the individual VM generators need no further intervention or coordination. We use this tool to replay real-life datacenter traces and also generate global resource usage volume spikes, as will be explained in detail later.

Cloud Application Suite: four distributed applications represent the most commonly deployed classes of cloud codes: (i) Nutch (data analytics) [22], (ii) Voldemort (data serving) [23] with the YCSB [39] load generator, (iii) Cloudstone (web serving), and (iv) HPL Linpack (high performance computing) [24]. The Nutch instance crawls and indexes a local internal mirrored deployment of the Wikipedia.org website, so that we avoid any skews in results due to WAN delays. The crawl job is set to process the top 200 pages at each level up to a depth of 4. The Cloudstone Faban workload generator is modified to only generate read-only requests, in order to avoid known mysql data-tier scalability bottlenecks [40]. We set Faban to simulate a total of 12k concurrent users. For YCSB, we use 4MB records and a workload profile that consists of 95% read operations and 5% update operations with zipfian request popularity. Finally, Linpack solves 8 problems of pre-determined sizes over the course of each experiment, measuring the average throughput achieved in each case.

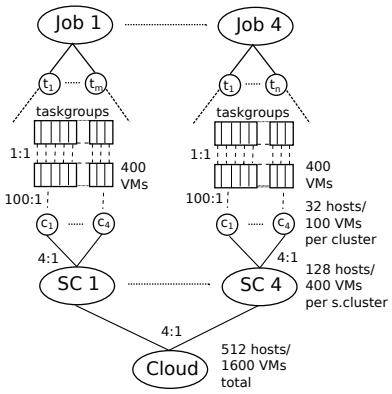


Fig. 5: Google trace replay.

5.2 Experimental Scenarios

I. Rightsizing the Cloud: recently, Google Inc. released a large scale, anonymized production workload resource usage trace from one of its clusters [30], containing data worth over 6 hours with samples taken once every five minutes. The workload consists of 4 large jobs that each contain a varying number of sub-tasks at each timestamp. The trace reports the normalized CPU and memory usage for each of the tasks at a given timestamp. Using Xerxes, we replay the resource usage pattern of the four major jobs in the trace on 1600 VMs, 400 per job, running on 512 of the 700 servers in our private cloud. All the VMs have equal Shares with no Reservation and Limit control values. For the sake of convenience, we replay only the first 5 hours worth of data from the traces.

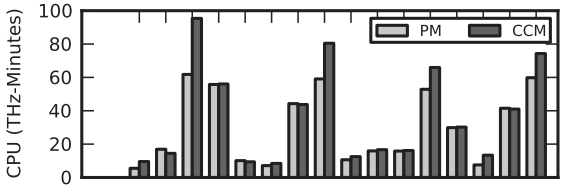
Figure 5 illustrates how the trace is mapped to the CCM datacenter: the tasks of a particular job are evenly partitioned, at a given timestamp, into 400 taskgroups, with one taskgroup mapped to a single VM. For example, if there are 2000 tasks for Job 1 at a given time, then each taskgroup contains 5 tasks, and a single VM replays the aggregate CPU and memory usage of this taskgroup[3]. This produces a different resource usage footprint in each VM at a given simulation step, but total resource usage of all 400 VMs together reflects the job's relative resource usage from the trace at that step. Across multiple runs of the scenario, for both PM and CCM, the same taskgroup to VM mapping is used.

The placement of VMs on the physical hosts works as follows. Each set of 400 VMs representing a particular job is further sub-divided into sets of 100 VMs and initially distributed evenly amongst the 32 hosts of a cluster or partition. Each VM is configured with 4 virtual

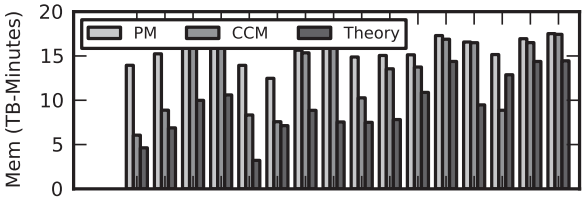
3. The CPU and memory usages of each taskgroup are re-normalized to a percentage value representing the fraction of the VM's configured capacity.

Param (row-wise)	Clus/Part	S-clus		Cloud	
Scenario (col-wise)	I, II, III	I, III	II	I, III	II
Mon Int. (secs)	20	120		600	
Bal Int. (mins)	5	20		60	
I_{CPU}	$\leq Prio3$	0.15	0.1	0.15	0.1
I_{MEM}	$\leq Prio3$	0.2		0.2	
$Max(Moves_{host})$	n/a	6	8	6	8
$Moves_{host}^{parallel}$	n/a	3	4	4	8
$Timeout_{host}^{move}$ (mins)	3	16		16	
Reac. Int (mins)	n/a	n/a	10	n/a	30

TABLE 3: Scenario-wise DRS and CCM Parameters. Parameter settings common for all scenrios span the entire entity column.



(a) CPU



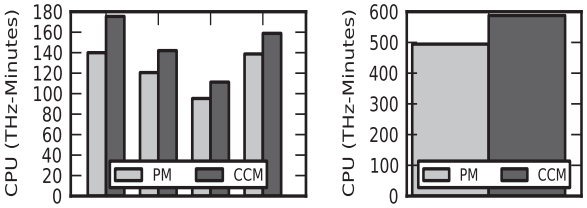
(b) Memory

Fig. 6: Cluster-wise/Partition-wise Resource Utilization.

CPUs (vCPU) and 2GB of memory. In the case of PM, VMware DRS dynamically multiplexes the load of the 100VMs on the 32 hosts of each partition using VM migrations. In the case of CCM, each set of 4 clusters are further composed into a single supercluster, and the 4 superclusters in turn report to a single cloud manager, with capacity multiplexing being performed at each of those levels.

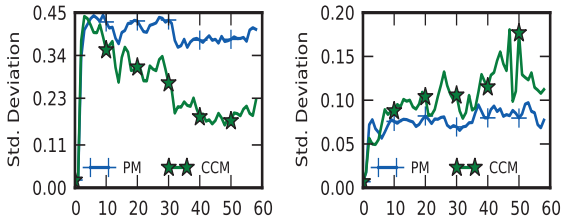
The first experiment evaluates CCM's ability to continuously rightsize the allocation for each job based on its actual runtime demand. Jobs become underprovisioned when the VMs' total resource demand during any time period is more than the available capacity in their respective cluster or partition, and similarly, become overprovisioned when the total demand is less than capacity. Table 3 shows the configurable parameter settings for DRS and CCM used for this experiment. We've derived these parameters experimentally. Our future work will address how to automate this process for different deployment scenarios.

All cluster, job, and cloud utilization numbers reported here and in the other scenarios are aggregates of the



(a) Jobs (b) Cloud

Fig. 7: Aggregate CPU utilization.



(a) Clusters of Supercluster 0 (b) Superclusters

Fig. 8: CPU Imbalance in Hierarchy.

individual VM utilizations, and they do not include the resource utilization due to the management operations or virtualization overhead. As a result, any improvement in utilization is due to the workloads inside VMs being able to do extra work as more resources are made available to them. Figure 6a shows the cluster-wise or partition-wise CPU usage computed as the Riemann sum of the corresponding aggregated VM usage curve over time, with individual samples obtained once every 5 minutes (average value over interval). Overall, there are a few clusters that require a large amount of CPU resource while the majority require a much smaller amount, over the course of the experiment.

As seen in the figure, statically sizing partitions and managing resources within the partition leads to sub-optimal use of available cloud capacity. In contrast, CCM is able to improve the total CPU utilization of the VMs in high demand clusters by up to 58%, by adding capacity from other unused clusters that are mostly part of the same supercluster. Figure 7a shows the job-wise CPU usage as the sum of their corresponding VM CPU usages. Here again, CCM is able to increase every jobs's utilization, between 14% to 25%, as its multiplexing removes resource consumption hotspots. The additional effect of all of these actions is that CCM is able to improve the overall datacenter VMs' CPU utilization by around 20%, as well (see Figure 7b), compared to a partitioned management approach. Also, as seen in Figure 6b, while CCM has a lower memory utilization than PM for VMs of each cluster, it is still more than what is theoretically required by the workload as computed from the trace. The memory utilization values collected for each VM at the virtualization infrastructure

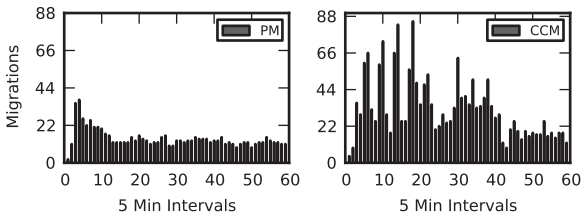


Fig. 9: Total VM Migrations every 5 Minutes.

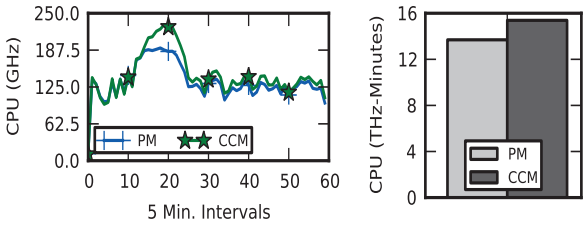
level include both the workload and the guest OS base memory usage.

Figures 8a[4] and 8b provide more information on how CCM operates with respect to reducing imbalance. Across the clusters of each supercluster, the CPU imbalance metric starts off high but CCM soon reduces it to the configured threshold of 0.15. Clusters are dynamically grown and shrunk in terms of capacity backing the workload VMs in such a way that their normalized utilization (NE) values are similar. Interestingly, in Figure 8b, CPU imbalance between superclusters for CCM grows higher than for the PM approach. This expected behavior is due to the fact that CCM does not take any action to reduce imbalance as long as it is within the configured threshold of 0.15. In addition to varying workload demands, another factor affecting how quickly CCM corrects resource usage imbalances in the datacenter, is the aggressiveness of its management actions, controlled via active throttling.

The number of VM migrations performed by PM and CCM across the entire datacenter, every 5 minutes, over the course of the experiment, is shown in Figure 9. Given the increased imbalance inherent in this workload, CCM's successful balancing of capacity uses twice as many migrations as PM on average – 32 vs. 14, per 5 minute interval, but it is apparent from Figure 7a that the implied management overhead is sufficiently well amortized, ultimately leading to notable improvement in the resources made available to jobs. When considered in the context of the entire cloud, this figure reflects as only 0.9% and 2%, respectively, of all the VMs, being migrated every 5 minutes. One reason for this modest overhead is that CCM explicitly manages management enforcement, by using timeouts on the overall host-move operation to abort long running migrations, and by promoting the selection of cheaper operations at a later time.

II. Workload Volume Spikes: the previous experiment featured jobs with variable demands but with a gradual rate of change. To test CCM's ability to allocate capacity to workloads experiencing a sudden volume spike in resource usage, as described by Bodik et.al.[41], we use the parameters identified by the authors (duration of spike onset, peak utilization value, duration of peak utilization and duration of return-to-normal) to add two volume spikes to a base resource usage trace. We generate the baseline resource (CPU only) usage trace from enterprise datacenter resource usage statistics reported by Gmach et.al.[10]. We use the statistics about the mean and 95th percentile values of the 139 workloads described in the paper to generate per workload resource usage trace for 5 hours assuming that the usage follows a normal distribution. Groups of 12 VMs in the datacenter replay each single workload's resource usage, letting us cover a total of 1600 VMs on 512 hosts. The cluster and supercluster organizations are similar to the previous scenario. The first volume spike quickly triples the overall workload volume over the space of 20 minutes across a subset of 100 VMs, mapping to a single cluster/partition, during the intial part of the experiment. The second spike triples the overall workload volume somewhat gradually across a larger scale of 400 VMs, mapping to a single supercluster, over the space of an hour, during the latter part of the experiment.

Table 3 shows the configurable parameter settings for this scenario. The central free host pool is set to hold DPM recommended hosts of up to 10% of the total supercluster capacity. The appropriate hosts are harvested from the clusters of each supercluster at the end of every periodic balancing round. Note also that in this scenario the reactive balancing operation does not have any limits to the number of hosts that can be moved from the central free host pool to a resource starved cluster or supercluster. This is due to the fact that these hosts are already pre-evacuated, resulting in the corresponding management action having lower complexity and cost. The overall behavior of the system in this scenario would be such that the majority of capacity movements happen due to reactive balancing, with periodic balancing only correcting any major imbalances in sub-entity utilizations.

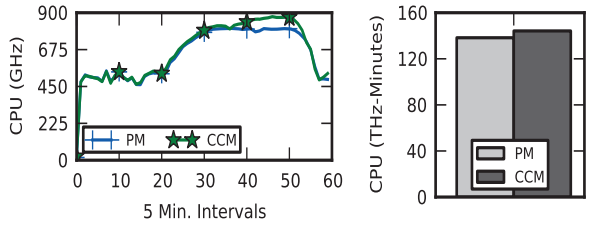


(a) Util. over Time (b) Util. Area of Spike

Fig. 10: Cluster Spike: Aggregate CPU utilization. Spike Duration = Samples 10 to 25 (75 mins).

Figure 10 shows the total VM CPU utilization achieved, for both PM and CCM, in the cluster replaying the sharper spike across 100 VMs. CCM is able to achieve a 26% higher peak value compared to PM (see Figure 10a). Overall, this translates to a 15%

4. Due to space limitations, imbalance across clusters is only shown for Supercluster 0; other superclusters follow a similar trend.



(a) Util. over Time (b) Util. Area of Spike

Fig. 11: Supercluster Spike: Aggregate CPU utilization. Spike Duration = Samples 20 to 56 (180 mins).

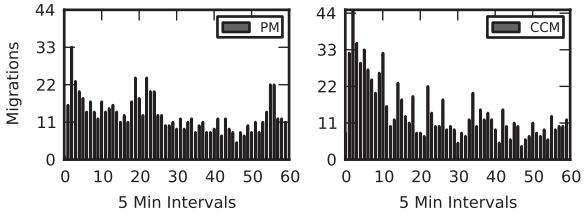


Fig. 12: Total VM Migrations every 5 Minutes.

improvement in total CPU utilization over the duration of the spike (see Figure 10b). With the larger magnitude supercluster level spike though, CCM's improvements are a modest 13% for the peak CPU utilization value (see Figure 11a) and 4% for the overall CPU utilization over the duration of the spike (see Figure 11b). Given the larger scale of the spike, a higher improvement would require the quick movement of a correspondingly larger number of hosts.

Our ability to accomplish this is constrained by the capacity of the central free host pool at each supercluster and also the rate at which the free host pool can be replenished with hosts. In addition, the host harvesting operation at the end of each periodic balancing round still has a limitation on the number of hosts that can be moved in order to reduce move failures and management cost. Depending on the importance of dealing with, and, the relative incidence of such aggressive volume spikes, the datacenter operator can choose to tailor the configurable parameters to trade between the potential loss of performance due to holding too many hosts in "hot-standby" vs. the improved ability to deal with volume spikes.

In terms of total migration costs, as seen in Figure 12, CCM has only a slightly higher migration rate compared to PM (14 vs. 13 on average, per 5 minute interval). The extra migrations are mostly due to the host harvest operations that happen at the superclusters. The DPM module is fairly conservative in choosing candidate hosts to be powered down; sometimes even choosing none. This reduces the number of hosts harvested during each invocation and the number of migrations being performed.

III. Application Performance: CCM's multiplexing should improve the performance of applications experiencing high usage or increased resource constraints due to co-location, without significantly degrading the remaining workloads. To evaluate this capability, we construct a scenario in which the cloud runs a workload consisting of the 4 jobs from Scenario I and an instance each of Nutch, Olio, Linpack and Voldemort. This arrangement allows us to capture each application's behavior in a datacenter environment with a realistic background load pattern. We run the workloads using 1024 VMs running on 256 servers in the datacenter. Almost all the VMs have equal Shares with no Reservation and Limit control values except for the Nutch, Linpack and Voldemort master nodes that have a memory Reservation of 1GB each given their relative importance. The overall physical and workload VM organization is shown in Figure 13.

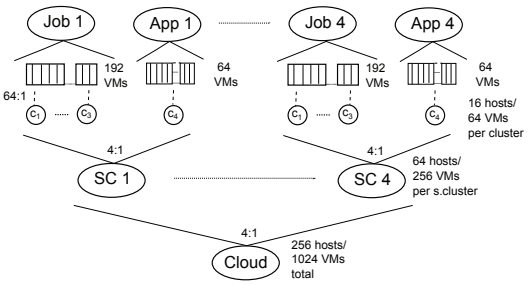


Fig. 13: Application Performance Scenario Architecture.

App.	Metric	PM		CCM		% diff
		Avg	*CV*	*Avg*	*CV*	
Nutch	Jobtime (mins)	134	0.55	106	0.28	21
Linpack	MFLOPS	392	0.04	468	0.11	20
Voldemort	Ops/s	10.81	0.42	9.8	0.5	-9
Olio	Reqs/s	33	0.65	274	0.6	730

TABLE 4: Application Performance Metrics.

Table 4 shows the raw average application performance metrics and the coefficient of variation (CV) of each metric, across 4 runs each of PM and CCM. The last column shows the percentage change in performance using CCM compared to PM. It is apparent that by dynamically allocating capacity to back the actual resource demand, CCM leads to noticeable performance improvements for Nutch, Olio and Linpack with a mild performance penalty for Voldemort. The Olio application is both CPU and memory intensive during its operation. The reason for the extremely high improvement seen with Olio is due to the fact that, unlike high CPU pressure, an extremely high memory pressure causes the Olio VM guest operating system to invoke its local memory management functions such as swapping and OOM killer, during the course of the experiment. As a result, request throughput suffers more than linearly, as in the case of PM, when such memory pressure isn't alleviated with additional capacity.

An important factor to point out in these results is the non-trivial variability in performance observed for the applications (CV values in Table 4). Some of this variability is due to the fact that management operations in both PM and CCM (i.e. VM migrations and host-moves) have different success and failure rates across multiple runs. This leads to a different resource availability footprint for different runs causing variability in performance. In addition, the current version of CCM also has limitations due to the fact that it does not explicitly take into account datacenter network hierarchy, rack boundaries etc. in its VM migration decisions. We believe that upcoming fully provisioned datacenter networking technologies [32], [31] will obviate some of these concerns. We are also currently creating interfaces between CCM and application frameworks to better manage end performance and reduce variability.

6 Related Work

Broadly, resource management solutions for cloud datacenters can be classified as application-driven, infrastructure-driven, or a hybrid of both. Among the infrastructure level solutions, Meng et.al. present a system to identify collections of VMs with complimentary resource demand patterns and allocate resources for the collection so as to take advantage of statistical multiplexing opportunities [11]. Such opportunities are inherent in cloud datacenter VM collections given the many customers and workloads they host. Wood et.al. present a system that uses black-box and gray-box information from individual VMs to detect and alleviate resource hotspots using VM migrations [13]. Chen et.al. provide and O(1) approximation algorithm to consolidate VMs with specific demand patterns onto a minimal set of hosts [42]. These solution approaches are designed to inspect individual resource signatures to allocate and extract good multiplexing gains in the datacenter.

There also exist systems that use explicit application feedback to tune resource allocation[43], [9], [44], [45], [46] at fine granularity (e.g. MHz for CPU) and coarse granularity (e.g. add/remove servers) in order to maintain SLAs. Such systems can be built on top of low-level capacity management systems like CCM to better achieve the objectives of both the cloud provider and customers. CloudScale [14] is an elastic resource allocation system that monitors both application and system parameters, primarily focusing on the accuracy of allocation methods in order to balance workload consolidation with SLA achievement. As the authors point out, this system is designed for a single datacenter node and can complement a large scale system like CCM. Zhu et.al. construct a resource allocation solution that integrates controllers that operate at varying scopes and time scales to ensure application SLA conformance [12]. CCM shares a similar architecture. The differences lie in the fact that CCM's mechanisms are designed for scalability (e.g. moving capacity vs. VMs), tackling management operation failures and cost variability. Further, we envision CCM as a low level datacenter capacity management system that exports interfaces through which application level solutions interact to maintain SLA conformance.

It is rather well known that large scale management systems must be designed to work in the presence of machine and component failures[47], [48]. However, the presence and the need to design for failure of *management operations* has received much less attention, if any. In addition, management operations also have bursty and highly variable resource consumption patterns [18], [17] that if not kept in check, may lead to unacceptable overheads that degrade application performance. CCM shares the design philosophy of using conservative but reasonably effective methods in this regard, over complex ones, with other large scale datacenter systems such as Ganglia [47] and Autopilot [48].

7 Conclusions

This paper describes a composite set of low-overhead management methods for managing cloud infrastructure capacity. Most real life datacenters have varying hardware and software limitations that prevent aggressively carrying out management operations. We have demonstrated, through data from an experimental evaluation on a fairly large infrastructure, that to achieve better capacity multiplexing, the focus needs to not only be on the accurate prediction of workload demand and aggressive optimization of the allocation algorithms, but also on dealing with the practical limitations of real-life infrastructures. While in this work, we stress the need for, and, use simple methods to overcome the detrimental effects and achieve good performance, in the future, we plan on further analyzing our data on these overheads and failures, and, develop systematic approaches to tackling them.

References

[1] "Hyper-V: Using Hyper-V and Failover Clustering," http://technet.microsoft.com/en-us/library/cc732181(v=ws.10).aspx.
[2] "Configuration Maximums - VMware vSphere 5.1," http://pubs.vmware.com/vsphere-51/index.jsp.
[3] "Citrix Workload Balancing 2.1 Administrator's Guide," 2011.
[4] A. Gulati, G. Shanmuganathan, A. Holler, and I. Ahmad, "Cloud-scale resource management: challenges and techniques," in *HotCloud'11*.
[5] T. Ristenpart *et al.*, "Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds," in *CCS '09*.
[6] "Cloud Infrastructure Architecture Case Study," 2012, http://www.vmware.com/resources/techresources/10255.
[7] "Xen Cloud Platform Administrator's Guide - Release 0.1," 2009.
[8] A. Tumanov, J. Cipar, M. A. Kozuch, and G. R. Ganger, "alsched: algebraic scheduling of mixed workloads in heterogeneous clouds," in *SOCC '12*.

[9] P. Padala *et al.*, "Adaptive control of virtualized resources in utility computing environments," in *EuroSys '07*.
[10] D. Gmach, J. Rolia, L. Cherkasova, and A. Kemper, "Workload analysis and demand prediction of enterprise data center applications," in *IISWC '07*.
[11] X. Meng, C. Isci, J. Kephart, L. Zhang, E. Bouillet, and D. Pendarakis, "Efficient resource provisioning in compute clouds via vm multiplexing," in *ICAC '10*.
[12] X. Zhu *et al.*, "1000 islands: Integrated capacity and workload management for the next generation data center," in *ICAC '08*.
[13] T. Wood *et al.*, "Black-box and gray-box strategies for virtual machine migration," in *NSDI'07*.
[14] Z. Shen *et al.*, "CloudScale: Elastic Resource Scaling for Multi-Tenant Cloud Systems," in *SoCC '11*.
[15] "Amazon Web Services - Case Studies," http://aws.amazon.com/solutions/case-studies/.
[16] O. Krieger, P. McGachey, and A. Kanevsky, "Enabling a marketplace of clouds: Vmware's vcloud director," *SIGOPS Oper. Syst. Rev.*, vol. 44, pp. 103–114, December 2010.
[17] A. Verma *et al.*, "The cost of reconfiguration in a cloud," in *Middleware Industrial Track '10*.
[18] V. Soundararajan and J. M. Anderson, "The impact of management operations on the virtualized datacenter," in *ISCA '10*.
[19] "VMware vSphere," http://www.vmware.com/products/vsphere/.
[20] M. Kesavan *et al.*, "Xerxes: Distributed load generator for cloud-scale experimentation," in *Open Cirrus Summit*, 2012.
[21] W. Sobel *et al.*, "Cloudstone: Multi-platform, multi-language benchmark and measurement tools for web 2.0," in *CCA '08*.
[22] "Apache Nutch," http://nutch.apache.org/.
[23] "Project Voldemort," http://project-voldemort.com/.
[24] "HPL - a portable implementation of the high-performance linpack benchmark for distributed-memory computers," http://www.netlib.org/benchmark/hpl/.
[25] "VMware DRS," http://www.vmware.com/products/DRS.
[26] "VMware distributed power management concepts and use," http://www.vmware.com/files/pdf/DPM.pdf.
[27] A. Gulati *et al.*, "VMware Distributed Resource Management: Design, implementation and lessons learned," *VMware Technical Journal*, vol. 1, no. 1, pp. 45–64, Apr 2012.
[28] "DRS performance and best practices," www.vmware.com/files/pdf/drs_performance_best_practices_wp.pdf.
[29] V. Kumar, B. F. Cooper, G. Eisenhauer, and K. Schwan, "imanage: policy-driven self-management for enterprise-scale systems," in *Middleware '07*.
[30] "googleclusterdata - Traces of Google tasks running in a production cluster," http://code.google.com/p/googleclusterdata/.
[31] R. Niranjan Mysore, A. Pamboris, N. Farrington, N. Huang, P. Miri, S. Radhakrishnan, V. Subramanya, and A. Vahdat, "Portland: a scalable fault-tolerant layer 2 data center network fabric," in *SIGCOMM '09*.
[32] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta, "Vl2: a scalable and flexible data center network," in *SIGCOMM '09*.
[33] "Unified Computing System," http://www.cisco.com/en/US/netsol/ns944/index.html.
[34] A. Gulati *et al.*, "Decentralized management of virtualized hosts," 12 2012, patent No. US20120324441 A1.
[35] "VMware VI (vSphere) Java API," http://vijava.sourceforge.net/.
[36] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield, "Live migration of virtual machines," in *NSDI '05*.
[37] H. Liu *et al.*, "Performance and energy modeling for live migration of virtual machines," in *HPDC '11*.
[38] C. A. Waldspurger, "Memory resource management in VMware ESX server," in *OSDI '02*.
[39] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, "Benchmarking cloud serving systems with ycsb," in *SoCC '10*.
[40] R. Cattell, "Scalable SQL and NoSQL data stores," *SIGMOD Rec.*, vol. 39, no. 4, pp. 12–27, May 2011.
[41] P. Bodik *et al.*, "Characterizing, modeling, and generating workload spikes for stateful services," in *SoCC '10*.
[42] M. Chen *et al.*, "Effective vm sizing in virtualized data centers," in *Integrated Network Management*, 2011.
[43] H. C. Lim, S. Babu, J. S. Chase, and S. S. Parekh, "Automated control in cloud computing: challenges and opportunities," in *Proceedings of the 1st workshop on Automated control for datacenters and clouds*, ser. ACDC '09.
[44] Q. Zhu and G. Agrawal, "Resource provisioning with budget constraints for adaptive applications in cloud environments," in *HPDC '10*.
[45] R. Singh *et al.*, "Autonomic mix-aware provisioning for non-stationary data center workloads," in *ICAC '10*.
[46] R. Nathuji, A. Kansal, and A. Ghaffarkhah, "Q-clouds: managing performance interference effects for qos-aware clouds," in *EuroSys '10*.
[47] M. L. Massie, B. N. Chun, and D. E. Culler, "The ganglia distributed monitoring system: Design, implementation and experience," *Parallel Computing*, vol. 30, p. 2004, 2003.
[48] M. Isard, "Autopilot: automatic data center management," *SIGOPS Oper. Syst. Rev.*, vol. 41, no. 2, pp. 60–67, Apr. 2007.

Mukil Kesavan received the M.S. in Computer Science degree at Georgia Institute of Technology in 2008. Currently, he is a PhD candidate (Computer Science) at Georgia Institute of Technology, focusing on scalable, fault-tolerant datacenter capacity allocation methods and on I/O performance isolation solutions for virtualized servers.

Irfan Ahmad spent 9 years at VMware where he was R&D lead for flagship products including Storage DRS and Storage I/O Control. Before VMware, he worked on a software microprocessor at Transmeta. Irfan's research has been published at ACM SOCC (best paper), USENIX ATC, FAST, and IEEE IISWC. He was honored to have chaired HotStorage '11 and VMware's R&D Innovation Conference (RADIO). Irfan earned his pink tie from the University of Waterloo.

Orran Krieger is the Founding Director for the Center for Cloud Innovation (CCI) and a research Professor at the Department of Computer Science Boston University. Before coming to BU, he spent five years at VMware starting and working on vCloud. Prior to that he was a researcher and manager at IBM T. J. Watson, leading the Advanced Operating System Research Department. Orran did his PhD and MASc in Electrical Engineering at the University of Toronto.

Ravi Soundararajan works in the Performance Group at VMware, specializing in the performance and scalability of virtualization management infrastructure. His research interests include simplifying performance monitoring and troubleshooting in virtualized systems. He received his SB from MIT, and his MS and PhD degrees from Stanford University, all in Electrical Engineering

Ada Gavrilovska is a senior research faculty at the College of Computing and the Center for Experimental Research in Computer Systems (CERCS) at Georgia Tech. She has published numerous book chapters, journal and conference publications, and edited a book High Performance Communications: A Vertical Approach (CRC Press, 2009). She has a BS degree in Computer Engineering from University Sts. Cyril and Methodius in Macedonia ('98), and a MS ('99) and PhD ('04) degrees in Computer Science from Georgia Tech.

Karsten Schwan is a professor in the College of Computing at the Georgia Institute of Technology. He is also the Director of the Center for Experimental Research in Computer Systems (CERCS), with co-directors from both GT's College of Computing and School of Electrical and Computer Engineering. Prof. Schwan's M.S. and Ph.D. degrees are from Carnegie-Mellon University in Pittsburgh, Pennsylvania. He is a senior member of the IEEE.