Elastic Resource Allocation in Datacenters: Gremlins in the Management Plane

Mukil Kesavan
Center for Experimental
Research in Computer
Systems (CERCS)
Georgia Institute of Technology
mukil@cc.gatech.edu

Ada Gavrilovska
Center for Experimental
Research in Computer
Systems (CERCS)
Georgia Institute of Technology
ada@cc.gatech.edu

Karsten Schwan
Center for Experimental
Research in Computer
Systems (CERCS)
Georgia Institute of Technology
schwan@cc.gatech.edu

Abstract

Virtualization has simplified the management of datacenter infrastructures and enabled new services that can benefit both customers and providers. From a provider perspective, one of the key services in a virtualized datacenter is elastic allocation of resources to work-loads, using a combination of virtual machine migration and per-server work-conserving scheduling. Known challenges to elastic resource allocation include scalability, hardware heterogeneity, hard and soft virtual machine placement constraints, resource partitions, and others. This paper describes an additional challenge, which is the need for IT management to consider two design constraints that are systemic to large-scale deployments: failures in management operations and high variability in cost. The paper first illustrates these challenges, using data collected from a 700-server datacenter running a hierarchical resource management system built on the VMware vSphere platform. Next, it articulates and demonstrates methods for dealing with cost variability and failures, with a goal of improving management effectiveness. The methods make dynamic tradeoffs between management accuracy compared to overheads, within constraints imposed by observed failure behavior and cost variability.

1. Introduction

Virtualization of physical datacenter resources enables a fluid mapping in which resource allocations can be varied elastically in response to changes in workload and resource availability. This is critical to realizing the benefits of utility computing environments like cloud computing systems, which can dynamically grow and shrink the resources allocated to customer workloads based on actual and current demands. Such elasticity of resources results in operational efficiency for cloud providers and in potential cost savings for customers.

IT managers face a number of challenges when implementing elastic resource allocation in current-generation virtualized datacenters that are often populated with tens of thousands of machines [10]. These challenges include scalability, hard-ware heterogeneity, hard and soft virtual machine (VM) placement constraints, resource partitions, and others, to which the research community has responded with novel techniques and associated system support [5, 8, 14, 13, 11].

This paper highlights the importance of two additional factors posing challenges to elastic resource management for large-scale datacenter and cloud computing systems. First, management operations may fail because the majority of these higher-level services are implemented in a best effort management plane. Second, there can be large variations in the costs of these management operations. For example, consider these three elastic resource allocation scenarios: 1) dynamic virtual machine placement to address long-term virtual machine demands, 2) live virtual machine migration or offline placement during power-on, and 3) using per-server resource schedulers for finer-grained allocation [2]. Prior work has shown that these management plane operations, including live virtual machine migration, can fail and that they exhibit varying resource costs [12]. These failures decrease the effectiveness of elastic resource allocation and variable costs complicate dealing with management overhead, relative to the benefits derived from elastic resource management. These facts, then, contribute to a 'glass ceiling' in the management plane that limits the improvements achievable by elastic resource allocation services [7, 9].

This paper illustrates the effects and importance of understanding management plane operations and their behavior, including empirical evidence of the operation failure rates and cost variability, observed in a 700-server datacenter running VMware vSphere. In this system, the base functionality of the vSphere platform has been extended with Cloud Capacity Manager (CCM), a scalable, hierarchical, elastic resource allocation system that is built on top of VMware's DRS. CCM consists of three hierarchical levels: (i) clusters (small groups of hosts as defined by DRS), (ii) superclusters (groups of clusters), and (iii) cloud (a group of superclusters). In addition to the load balancing and re-source allocation performed for virtual machines of a single cluster by DRS, CCM dynamically shuffles *capacity* be-tween clusters and superclusters in response to aggregate changes in demand.

This paper quantifies the impact of management operation failures and cost variability on CCM, and presents simple methods for coping with these issues. It concludes with a brief discussion of the broader implications this poses when designing and constructing large-scale datacenter infrastructure services. Future research points out that design for management operation failures and cost variability, explored in the context of elastic resource allocation, is more

broadly applicable to higher-level services pertaining to high availability, power management, virtual machine backups/disaster recovery, virtual machine environment replication, and more generally, to adaptive systems and control.

2. CCM Overview

The overall architecture of CCM is shown in Figure 1. Demand-aware load balancing is periodically per-formed by capacity managers at the cluster and supercluster levels, and at the overall cloud level. Capacity managers operate independently, but share with the level above (if present) the combined resource demand information of all virtual machines on the hosts they manage. Sharing, as well as load balancing, operates at progressively larger time scales when moving up the hierarchy. Based on the combined virtual machine resource demand information (including some additional headroom), an *imbalance* metric is computed at each manager, as the standard deviation of the normalized demand of sub-entities. Load balancing is triggered when this imbalance is above an administrator-specified threshold during an invocation of the algorithm, and capacity is moved from entities with low-normalized demand to those with high-normalized demand.

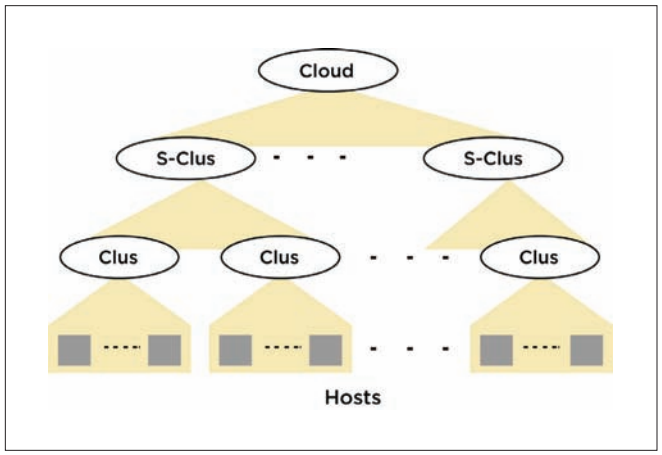

Figure 1. Hierarchical resource management architecture.

At the cluster level, VMware DRS is used to make independent and localized decisions to balance loads by migrating virtual machines using per-virtual machine demand estimates. At the supercluster and cloud levels, coarse-grained allocation changes are carried out by *logically re-associating capacity*. This process migrates individual evacuated hosts, rather than individual virtual machines, across clusters and superclusters. All virtual machines running on a host to be re-associated are migrated to other hosts that are part of the same cluster to which the host currently belongs. This is done to seamlessly integrate with DRS and to minimize the amount of state that must be moved between capacity managers during each migration. DRS automatically adapts to increased and decreased capacity in a cluster without requiring any changes.

CCM is implemented in Java, using the vSphere Java API [3] to collect metrics and enforce management actions in the vSphere provisioning layer. DRS is used in its standard vSphere server form. Both the cloud manager and supercluster managers are implemented as part of a single, multithreaded application running on a single host to make it simple to prototype and evaluate.

The remainder of this paper treats CCM as a black-box system that ingests monitoring information and emits management actions. The paper studies the management actions carried out in the management plane, or management enactment, and focus on enactment failures and variation in enactment cost.

Host-move action: A *host-move* is one of the basic actions performed by CCM at the supercluster and cloud levels, the purpose being to elastically allocate resources in the datacenter. There are two significant types of moves: host-move between clusters, and host-move between super-clusters. Each host-move is composed of a series of *macro* operations in the management plane that must be executed in order for an inter-cluster move, as shown in Figure 2. Each macro operation may be implemented by one or more lower level *micro* management plane operations.

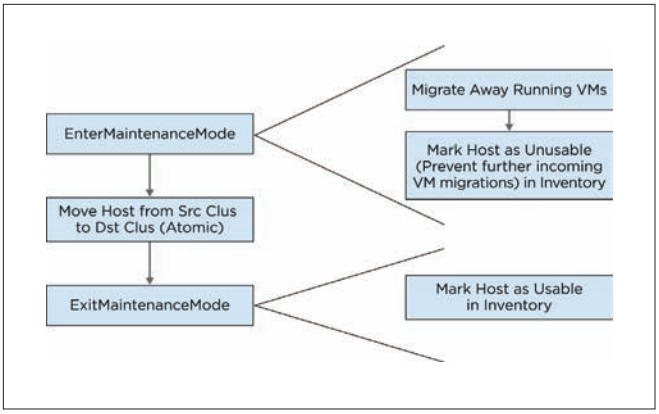

Figure 2. Inter-cluster host-move.

The "EnterMaintenanceMode" operation, for instance, places a host into a *maintenance* state, in which no virtual machines currently use it, so that this host can be moved from one cluster to another. This is potentially the most resource-intensive of these management plane operations because all virtual machines currently running on the host must be evicted. The time to complete this operation depends on factors like virtual machine size and active virtual machine memory footprints [4]. DRS selects other hosts in the source cluster and moves evicted virtual machines to those hosts.

The important point to note about these composite host-move operations is that the failure of a macro operation always results in complete failure of the whole host-move, whereas a failure of a micro-operation may or may not result in total failure depending on whether it is a pre-requisite for future operations (e.g., "Getting un-collected stats" need not result in total failure). As discussed further below, this classification of management plane operations helps when devising ways to cope with failures.

3. Failure and Cost Variability Analysis

This section outlines the experimental setup and presents data on virtual machine migration and host-move operation failures for representative large runs across 256 hosts of the 700-host datacenter. It also shows how these failures impact the performance of CCM, by defining a 'goodput' metric termed *effective management action throughput (emat)*.

Testbed: Each datacenter host has two dual-core AMD Opteron 270 processors, 4 GB of memory, and runs the VMware vSphere Hypervisor (ESXi) v4.1. The hosts are all connected to each other and four shared storage arrays (4.2 TB total capacity) via a Force 10 E1200 switch over a flat IP space. The common shared storage is exported as NFS stores to make it easy to migrate virtual machines across the datacenter machines. The open source Cobbler installation server runs on a dedicated host for DNS, DHCP, and PXE booting needs. The VMware vSphere server and client are used to provision, monitor, and partially manage the cloud.

Workload and setup: Realistic datacenter-wide load patterns are generated by replaying the resource usage traces released by Google Inc. [1]. The first four hours of the resource usage pattern of the jobs in the trace are replayed on 1024 virtual machines, 256 per job. There are a total of 16 clusters covering the 256 hosts, with each cluster initially containing 16 hosts and 64 virtual machines. A supercluster is composed of a set of 4 clusters, separate from other superclusters, with a total of 64 hosts and 256 virtual machines. This results in a total of four superclusters in the cloud. A given job's virtual machines fit within a single supercluster. A more detailed explanation of the load generation framework and trace replay appears in [6].

The load-balancing algorithm at the cluster level runs once every 5 minutes, at the supercluster level once every 20 minutes, and at the cloud level once every hour. Results for four different configurations of CCM are shown. The first configuration attempts to carry out *all* of the host-move actions recommended by the balancing algorithms during their respective invocations. Further, all of the moves are also carried out in parallel, with the intent to reduce the amount of time the system is in a state of flux. This sort of a configuration is not uncommon in practice where system developers assume a low and stable cost for enforcing management. This configuration is named CCM_nr, short for CCM with "no restrictions". In CCM_nr and the rest of the configurations, DRS is set to carry out priority 1, 2, and 3 virtual machine migration recommendations, with at most eight virtual machines per host in parallel.

Figure 3 shows the number of successful host-move operations. All of the attempted host-moves are inter-cluster moves, which are determined by the nature of the work-load. The results showed a 38% host-move failure rate and low number of successful host-moves that result in little to no effect on work-load performance (data not shown here). In addition to outright failures, there were also a large number of extremely slow operations that did not complete during the course of the experiment. In the figure, these are also counted as failures. Figure 4 presents the proportion of failures due to a failure of each of the three macro management plane operations in the inter-cluster move. It can be seen that more than 90% of the failures are due to a failure of the "EnterMaintenanceMode" operation, or in other words, a failure to evict (by migrating them away) all of the running virtual ma-chines on the hosts in question.

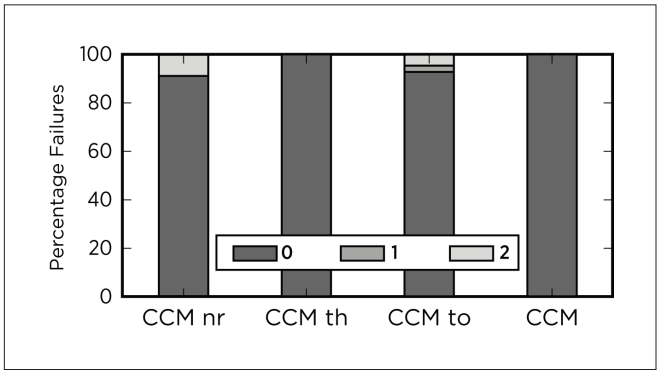

Figure 4. Fraction of inter-cluster host-move failure due to failure of each macro operation. Key: 0=EnterMaintenance-Mode, 1=Move-Host, 2=ExitMaintenanceMode.

In Figure 5, CCM_nr shows a noticeable 29% virtual machine migration failure over the course of the experiment. Note that the migration failures reported here include those due to host-moves and those performed by DRS during its load balancing. However, it still gives a general link between high migration failure rates and host-move failures given that a failure to evacuate a single running virtual machine would result in the failure of the entire operation.

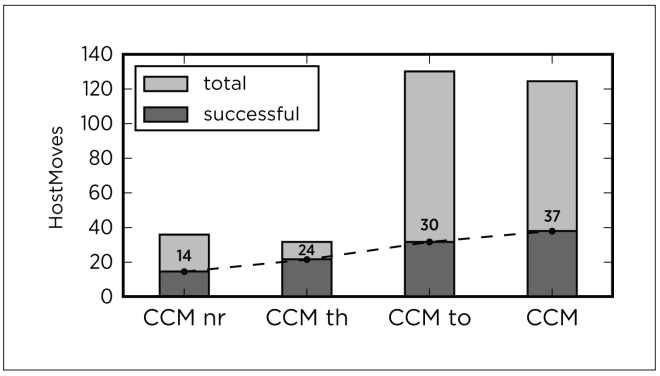

Figure 3. Number of successful host-moves.

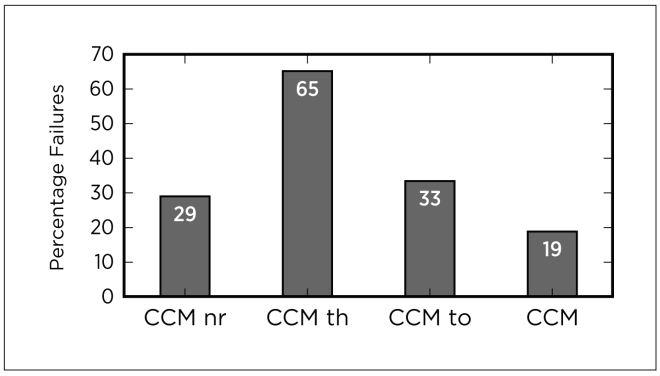

Figure 5. Migration failures.

Table 1 shows some of the major causes of VM migration failures collected from the vSphere layer and their percentage contribution to the overall number of migration failures. Some of the causes appear to be random or transient in nature, possibly due to software bugs or configuration issues, whereas others point to the fact that migrations may have failed directly or indirectly due to high resource pressure given the way CCM_nr enforces actions (e.g., "Operation timed out").

ABBV. CAUSE	nr	th	to	CCM
General system error	31	22	10	21
Failed to create journal file	45	7	63	54
Operation timed out	4	49	18	13
Operation not allowed in cur state	12	22	4	12
Insuf. host resources for VM reserv.	0	0	3	0
Changing mem greater than net BW	0	0	1	0
Data copy fail: already disconnected	8	0	0	1
Error comm. w/ dest host	0	0	1	0

Table 1: VM migration failure causes breakdown for each con-figuration. Values denote percentages.

Given this intuition, three different configurations of CCM are presented in which the degree of resource pressure imposed by management actions is controlled. Pressure is changed by: (1) explicit throttling of management enactions, i.e., the number and parallelism of host-moves (CCM_th), (2) automatically aborting long-running actions using timeouts (CCM_to) and, (3) a combination of both action throttling and timeouts (CCM). Configurations differ in their use of values for throttling host-moves, i.e., by limiting the maximum number of host-moves per balancing period to eight and reducing the parallelism in moves to no more than four at a time for each supercluster. In addition, a static timeout setting of 16 minutes is used for a single host-move operation for CCM_to and CCM. Experimental results with these configurations test the hypothesis that the resource pressure induced by management causes the observed failures.

The CCM_th, CCM_to, and CCM configurations exhibit higher success rates (41%, 53%, and 62%, respectively, compared to CCM_nr) in the host-move operations performed, as shown in Figure 3. This success points to the fact that there is a quantifiable benefit in managing the resource pressure of management action enforcement. Note that the monitoring and load-balancing algorithms, and the workload, remain the same for all configurations. In the case of CCM_to and CCM, both configurations achieve a much higher success rate while also attempting almost three times as many host moves as CCM_nr and CCM_th.

This is because a timeout allows stopping long running host-moves where the cost-to-workload benefit ratio is unfavorable. If the load imbalance in the workload continues to persist, the balancing algorithms recommend a fresh set of moves during the next round that may be more cost effective. In this fashion, the higher-level service could explore more efficient host-move alternatives than those available at a prior point in time.

To better quantify the behavior exhibited in the experiments discussed above required a new to more objectively compare the different configurations: *effective management action throughput (emat)*—the ratio of the number of successful host-moves to the total amount of time spent in making all moves (successful and failed). Table 2 shows the total minutes spent performing the host-moves and the emat metric for each of the four configurations. The total time is summed over all host-moves attempted by each configuration, counting parallel moves in serial order. Even though CCM and CCM_to perform nearly three times as many moves (see Figure 3) as the other two configurations, the total time spent enforcing the moves was significantly lower. This, combined with the fact that these configurations also delivered a higher host-move success rate, results in a two-to-six fold advantage in terms of the *emat* metric.

METRIC	CCM_nr	CCM_th	CCM_to	CCM
Total Host-move Mins	4577	703	426	355
emat (moves/hr)	0.18	2.05	4.23	6.25

Table 2: Management Metrics

Note that for the CCM_nr and CCM_th configurations, the balancing algorithms cannot start recommending and enforcing new host-moves while the previous moves are still in progress and the system is in an unstable state. This is the reason for the rather smaller number of host-move attempts in both of these cases. The large proportion of host-move failures for CCM_to and CCM are due to a combination of explicit aborts and the fact that the configurations are still afflicted by a non-negligible fraction of virtual machine migration failures as shown in Figure 5.

In addition to the host-move failures, it is also important to consider the cost-to-benefit ratio of the enforced management actions. Figure 6 depicts the average, minimum, and maximum values of successful host-move action times for all four configurations. Given that the resource cost of an action is directly proportional to

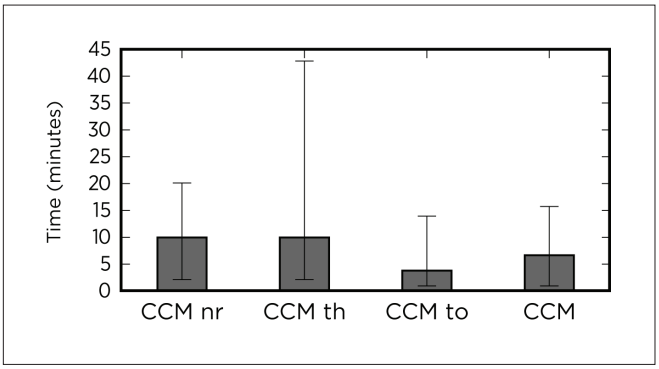

Figure 6. Host-move times.

the amount of time the action takes, it is important to abort overly long-running host-moves, which is illustrated with the more stable host-move times observed with CCM_to and CCM, each of which

are more likely to deliver a favorable cost-to-benefit ratio. Further, since the most resource-intensive stage of a host-move operation is the "EnterMaintenanceMode" operation, with its need to evict all of the current virtual ma-chines running on a host, this operation is particularly affected by high variations in virtual machine migration times. Variability in these times is evident from the fact that the 90th percentile values of virtual machine migration times computed for CCM_nr, CCM_th, CCM_to, and CCM show high values of 377s, 320s, 294s, and 335s, respectively. These variations have two causes: those attributable to management pressure and natural causes due to differences in virtual behavior, leading to runtime variations in the active memory footprints. The latter are beyond the control of the management layer, but demonstrate that host-move times can vary widely, and as a result, the abort/search method of exploring more efficient moves is still relevant. Alternatively, the system can also explicitly track and predict management costs and make moves when the moves will produce the most favorable cost-to-benefit ratio.

4. Conclusions

This paper draws attention to the importance of designing for failures, not only for the applications running in large-scale datacenter systems, but also and perhaps even more importantly, for the management actions that are in-tended to improve application performance. Intuitively, this is because management failures can have a substantial effect on the efficiency of datacenter operation. First, because failed actions consume resources without contributing to the desired improvements and second, because the resource pressure induced by such actions can lead to action failures or undue delays. Therefore, it is important to design a data-center's management plane that considers management failures as well as variability in the costs of enforcing actions in the management plane.

The experimental results shown in this paper illustrate the efficacy of simple methods for improving otherwise cost-variable and/or failure-intolerant management action. Methods include explicit action throttling to reduce the resource pressure imposed by such actions, and aborts that prevent undue resource consumption by actions experiencing delays. Results shown in the paper use static settings to test the usefulness of the action throttling and early aborts. Future work will develop techniques to dynamically derive these parameters. Additional experiments are in process with alternative strategies to hedge against failures that cannot be controlled, in order to minimize overall management cost, reduce failure rates, and maximize the bene-fits to application workloads.

References

1 googleclusterdata: traces of google tasks running in a production cluster, http://code.google.com/p/googleclusterdata/

2 VMware DRS, http://www.vmware.com/products/DRS

3 VMware VI (vSphere) Java API, http://vijava.sourceforge.net/

4 C. Clark et al. Live migration of virtual machines. In NSDI'05.

5 D. Gmach et al. Workload analysis and demand prediction of enterprise datacenter applications. In IISWC '07.

6 M. Kesavan et al. Xerxes: Distributed load generator for cloud-scale experimentation. Open Cirrus Summit, 2012.

7 S. Kumar et al. vManage: loosely coupled platform and virtualization management in datacenters. ICAC '09.

8 X. Meng et al. Efficient resource provisioning in compute clouds via virtual machine multiplexing. In ICAC '10.

9 R. Moreno-Vozmediano et al. Elastic management of cluster-based services in the cloud. ACDC '09.

10 D. Schneider et al. Under the hood at Google and Facebook. Spectrum, IEEE, 48(6):63 –67, June 2011.

11 Z. Shen et al. Cloudscale: Elastic resource scaling for multi-tenant cloud systems. In SoCC '11.

12 V. Soundararajan et al. The impact of management operations on the virtualized datacenter. In ISCA '10.

13 T. Wood et al. Black-box and gray-box strategies for virtual machine migration. In NSDI'07.

14 X. Zhu et al. 1000 islands: Integrated capacity and workload management for the next generation datacenter. In ICAC '08.