International Journal of
Emerging Research in Management &Technology
ISSN: 2278-9359 (Volume-3, Issue-5)

Research Article | **May 2014**

Improving the Efficiency of Cloud Infrastructures with Elastic Tandem Machines

P. Radha Madhavi [1]
Pursuing MCA,
Department of MCA,
Tirumala Engineering College,
Narasaraopet, Andhra Pradesh, India

Srikanth Yadav. M, [2]
Associate Professor
Department of CSE & IT
Tirumala Engineering College,
Narasaraopet, Andhra Pradesh, India

Abstract—

In this paper, we propose a concept for improving the energy efficiency and resource utilization of cloud infrastructures by combining the benefits of heterogeneous machine instances. The basic idea is to integrate low-power system on a chip (SoC) machines and high-power virtual machine instances into so-called Elastic Tandem Machine Instances (ETMI). The low-power machine serves low load and is always running to ensure the availability of the ETMI. When load rises, the ETMI scales up automatically by starting the high-power instance and handing over traffic to it. For the non-disruptive transition from low-power to high-power machines and vice versa, we present a handover mechanism based on softwaredefined networking technologies. Our evaluations show the applicability of low-power SoC machines to serve low load efficiently as well as the desired scalability properties of ETMIs.

Keywords— cloud computing, infrastructure as a service, efficiency, energy, elasticity, scaling, system on a chip, software defined Networking, green computing

I. INTRODUCTION

In modern virtualization based compute clouds, applications share the underlying hardware by running in isolated Virtual Machines (VMs). Each VM, during its initial cre- Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. ation, is configured with a certain amount of computing resources (such as CPU, memory and I/O). A key factor for achieving economies of scale in a compute cloud is resource provisioning, which refers to allocating resources to VMs to match their workload. Typically, efficient provisioning is achieved by two operations: (1) static resource provisioning. VMs are created with specified size and then consolidated onto a set of physical servers. The VM capacity does not change; and (2) dynamic resource provisioning [20, 18, 15]. VM capacity is dynamically adjusted to match workload fluctuations. Static provisioning often applies to the initial stage of capacity planning. It is usually conducted in offline and occurs on monthly or seasonal timescales [7, 32]. Such provisioning functionality has been included in many commercial cloud management softwares [27, 28, 11, 19, 16]. In both static and dynamic provisioning, VM sizing is perhaps the most vital step.

VM sizing refers to the estimation of the amount of resources that should be allocated to a VM. The objective of VM sizing is to ensure that VM capacity is commensurate with the workload. While over-provisioning wastes costly resources, under-provisioning degrades application performance and may lose customers. Traditionally, VM sizing is done on a VM-by-VM basis, i.e., each VM has an estimated size based on its workload pattern. In a significant departure from such an individual-VM based approach, we advocate a *joint-VM provisioning* approach in which multiple VMs are consolidated and provisioned based on an estimate of their aggregate capacity needs. Conceptually, joint-VM provisioning exploits statistical multiplexing among the dynamic VM demand characteristics, i.e., the peaks and valleys in one VM's demand do not necessarily coincide with the other VMs. The unused resources of a low utilized VM, can then be directed to the other co-located VMs at their peak utilization. Thus, VM multiplexing potentially leads to significant capacity saving compared to individual-VM based provisioning. The savings achieved by multiplexing are realized by packing VMs more densely into hardware resources without sacrificing application performance commitment. While this increases the overall consolidation ratio, the additional virtualization overheads associated with scheduling somewhat higher number of VMs is generally minimal as long as the VM footprints fit in the provisioned capacity [31]. The savings with our joint-sizing approach are up to 40% according to our analysis on the utilization data from a production data center. The above seemingly simple concept poses several research challenges. For example, given a set of VMs to be consolidated and provisioned, how to estimate their total capacity needs that neither break application performance commitments nor waste resources? Since any combination of VMs can be potentially provisioned together, how to find combinations that saves the most capacity?

What are the potential scenarios for applying this technique in compute clouds?

In this work, we address these questions in detail. Specifically, the primary contributions of this work are:

- We introduce a Service-level-agreement (SLA) model that map application performance requirements to resource demand requirement. We propose a systematic method to estimate the total amount of capacity for provisioning multiplexed VMs.
  The estimated aggregate capacity ensures that the SLAs for individual VMs are still preserved.
- We present a VM selection algorithm that seeks to find those VMs with the most compatible demand patterns. The identified VM combinations lead to high capacity savings if they are multiplexed and provisioned together.
- We illustrate effective and feasible applications of the proposed technique for capacity planning and for providing resource guarantees via VM reservations. Both applications can be easily employed in existing cloud and virtualization management infrastructures with minimal intrusion and substantial benefits in return.

We conduct simulations to evaluate the proposed methods by using a massive dataset collected from commercial data centers. The dataset includes 159-thousand VMs and spans three months. In the capacity planning application, joint provisioning uses 45% less physical machines for hosting the same set of VMs. In the VM reservation application, joint provisioning improves the ratio of admitted VMs by 16% on average, and up to 75% with more stringent SLA requirements. These results demonstrate the significant potential by leveraging VM multiplexing. The rest of the paper is organized as follows. Section 2, surveys prior work related to resource provisioning and VM multiplexing. Section 3, motivates the use of VM multiplexing to improve resource utilization efficiency. It uses data from commercial global data centers to demonstrate the potential capacity gains with joint-VM based provisioning. Section 4, provides an overview of our methodology. Section 5, describes the underlying SLA model for resource provisioning. Section 6 describes the algorithm for joint-VM sizing. Section 7 presents a method for selecting compatible VMs for multiplexing. Section 8 discusses the use cases for VM multiplexing and provides the experimental evaluations. Last, Section 9 offers our conclusions.

II. RELATEDWORK

As resource provisioning is a common management task in modern virtualization-based compute clouds, it has become an important ingredient of commercial cloud management products including VMware Capacity Planner [27] and CapacityIQ [28], BM WebSphere CloudBurst [11], Novell lateSpin Recon [19] and Lanamark Suite [16]. Meanwhile, the topic has been widely studied in the research community. Some existing work apply application/VM profiling and statistical modeling for long-term resource provisioning [1, 7, 8, 18]. Other work focus on short-term, dynamic provisioning techniques [20, 18, 15]. While all these products and research work provide valid solutions, they stand on a VM-by-VM basis, that is, they consider each VM's resource need separately. Such an approach provides a reasonably effective solution to the provisioning problem, yet it generally leads to low resource utilization [13, 22, 25]. In contrast, our work exploit the workload multiplexing among multiple VMs. We demonstrate that how multiple VMs' capacity requirement can be satisfied collectively with a much lower total resource consumption.

A few prior arts consider concepts similar to VM multiplexing for improving resource utilization. Specifically, Sonnek and Chandra [24] identify VMs that are most suitable for being consolidated on a single host. They propose to multiplex VMs based on their CPU and I/O boundedness, and to co-locate VMs with higher potential of memory sharing. Wood et al. [33] present a method for co-locating VMs with similar memory content on the same hosts for higher memory sharing. Gupta et al. [10] further progress this memory sharing method by limiting memory sharing within page boundaries.

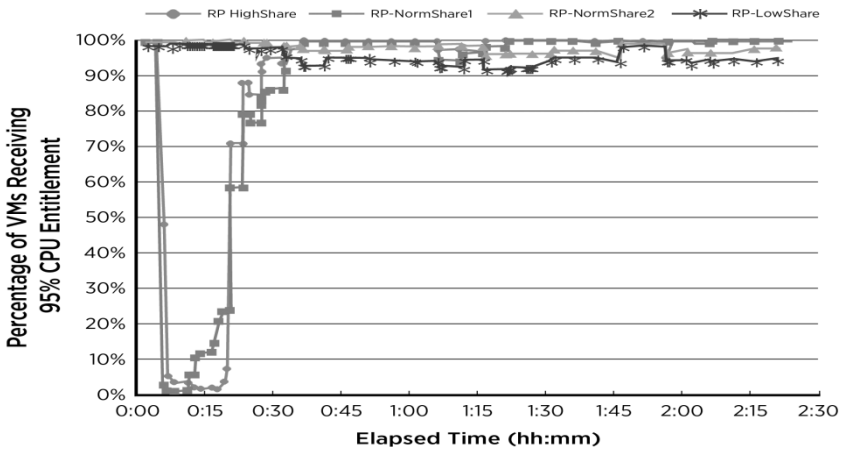

Figure 1-Individaul VM requirements

Govindan et al. [9] propose to consolidate VMs based on their communication patterns. Govindan et al. [5, 8] use statistical multiplexing of applications to identify applications that fit into given power budgets. In comparison to all these studies, our work also apply VM multiplexing to consolidate VMs more densely on hosts. Nevertheless, our work

provide a general performance model and enabling techniques that ensure the application performance is not degraded by VM multiplexing

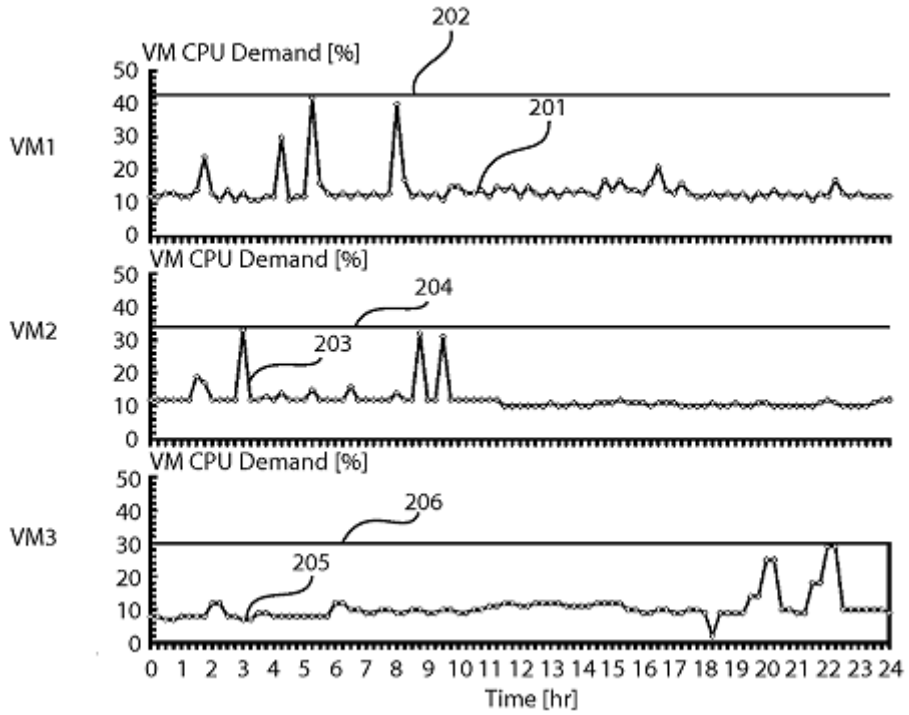

Figure 2- Aggeregate Perofrmance of VM's

III. MOTIVATION

Our proposed *joint-VM provisioning* approach stems from an observation on the VM resource demand in actual data centers. It is well known that the applications enclosed by VMs - and thus the VMs themselves - exhibit timevarying resource demand patterns with bursts of high demand periods, intermixed with low-utilization regions [3, 23, 26]. Furthermore, our measurement on a large set of VMs shows that many VMs, even in the same data center, exhibit demand patterns with different, unaligned distributions of these *peaks* and *valleys*. Therefore, while a capacity planner that operates over singleton VMs is bound by the peaks of *each* individual VM, a joint-VM approach can potentially exploit the *multiplexing* among the demand patterns of multiple VMs to reach an aggregated capacity measure that is only bound by the aggregate peak behavior.

Figures 1 and 2 illustrate an example with three VMs from a production data center. Figure 1 depicts the monitored CPU demand of each VM over a 24-hour period. Each VM exhibits a time-varying demand pattern with interspersed peaks. However, the peaks for each VM occur at different time. The figure also depicts a simple capacity bound required by each VM that is based on conservatively satisfying each instantaneous peak, i.e., the capacity required for each VM is set as the maximum demand ever observed in the history time period. Based on this capacity model, the total capacity required for the three VMs is 104%. In contrast, Figure 2 depicts the *multiplexed* behavior of the three VMs when they are jointly provisioned. The figure shows the aggregated CPU demand. Here we see the increased number of peaks in the aggregated demand. The total capacity required to satisfy the demand of all the three VMs is only 67%, a dramatic improvement compared to the separate provisioning scenario. To assess the potential capacity savings with multiplexing in VM capacity planning at the enterprise scale, we extend the above comparison to a massive dataset collected from a set of commercial data centers. The dataset involves 15,897 VMs that reside on 1325 physical hosts, managed by tens of regional hosting operators and used by hundreds of enterprise customers. The dataset includes configurations of each host, and the CPU and memory utilization ratio of each VM for up to three months. All the evaluation in the rest of this work is based on this dataset.

IV. IMPLEMENTATION

We extend the open source Nimbus cloud computig toolkit, which provides on-demand access to resources (in the form of Ms), to support the deployment of preemptible leases on idle cloud nodes, also referred to as backfill. We make a number of simplifying assumptions in our current implementation. First, the Nimbus administrator must configure backfill. On-demand cloud users cannot elect to deploy backfill VMs. Second, the current implementation is capable of using only a single backfill VM image per VMM node. Different backfill VM images could be deployed on different VMM nodes, allowing multiple backfill VM images to operate within the same IaaS cloud, each performing different functions. Unless the user specifies the max number of backfill instances, backfill automatically attempts to deploy as

many backfill VMs as possible when it is enabled. Initially, we support two termination policies for selecting backfill VMs for termination in order to fulfill an on-demand lease. The first policy simply selects a random backfill VM. The second policy, and the default, terminates the most recently deployed backfill VM in an attempt to minimize the amount of work "lost" by the premature termination of backfill VMs. In future work we hope to add additional backfill termination policies. Finally, our backfill implementation cleanly shuts down the backfill VM. Clean shutdown requires additional time over trashing the VM, however, performing a clean shutdown notifies services and applications running inside the backfill VM that the VM will be terminated, allowing them to respond appropriately (e.g. notify a central manager to reschedule currently running jobs).

A. Backfill Configuration Options

Only the Nimbus cloud administrator can configure Backfill. The main configuration options are specified in a backfill.conf file on the Nimbus service node, allowing administrators to easily configure and deploy backfill VMs on Nimbus clouds. The backfill.conf options include:

- Backfill.disabled: This option specifies whether backfill is enabled or disabled for the specific cloud. The default is disabled.
- Max.instances: This option specifies the maximum number of backfill VMs to launch (assuming there are enough idle nodes). The default, 0, launches as many as possible.
- Disk.image: This option specifies the full path to the backfill VM image on the VMM nodes. This option assumes that the backfill VM image has already been pushed out to the VMM node, the Nimbus service does not automatically transfer it. The image must be in the same location on every VMM node.
- Memory.MB: This option specifies the amount of memory (RAM) to use for backfill VMs. The default is 64 MB.
- VCPUs: This option specifies the number of VCPUs to use for backfill VMs. The default is 1.
- Duration.seconds: This option specifies the amount of time (in seconds) backfill VMs should run before being terminated (currently Nimbus doesn't support "infinite" length VM deployments). The default is one week.
- Termination.policy: This option specifies the termination policy to use when terminating backfill VMs. The policies currently supported include a "most recent" policy that first terminates backfill VMs running for the least amount of time and an "any" policy that simply terminates a random backfill VM. The default is the most recent policy.
- Retry.period: This option specifies the duration (in seconds) that the backfill timer waits in between attempts to deploy backfill VMs on idle VMM nodes. The default is 300 seconds.
- Network: This option allows the IaaS cloud administrator to specify whether the backfill VMs should use the public network or private.

B. Extensions to the Nimbus Workspace Service

We modified the Nimbus workspace service [16] to support backfill VM deployments. The workspace service is responsible for managing the VMM nodes and servicing on demand user requests for VMs. In particular, we added a backfill Java class that contains the majority of the backfill implementation code. The backfill configuration file is read when the Nimbus workspace service is started; if backfill is enabled then the service attempts to launch backfill VMs until the request for resources is denied or the maximum number of backfill instances is reached (as specified in backfill.conf). The service also starts a backfill timer that continually loops, sleeping for the duration specified by duration.seconds in backfill.conf, and attempts to launch backfill VMs when it wakes (until the request for resources is denied or the maximum number of backfill instances have been launched).

In general this is an acceptable approach, however, there is a design flaw in this initial implementation. It is possible that all backfill VMs could be terminated and yet the on-demand request could still be rejected. In this case the ideal solution would be to recognize, upfront, that the IaaS cloud is unable to fulfill the on-demand request and, therefore, the on-demand request should be rejected immediately before terminating any backfill VMs. However, recognizing this upfront requires a complete picture of the VM placement and the location of individual VM deployments on VMM nodes.

V. RELATED WORK

In this section, we give an overview of related approaches. As already described in Sec. I, approaches based on resource overbooking such as [1], [2], [3] target the same goals as our approach, namely, to increase energy efficiency and utilization of physical resources. Due to the problems of heavy overbooking (risk of overloading) and on-demand booting or restoring of VMs (reduced availability due to latencies), we investigated a different approach in this paper utilizing heterogeneous hardware to efficiently support idle and weakly loaded VMs without sacrificing availability.

From a technical point of view, our approach shares similarities with load balancing mechanisms. A common principle is to first forwarded requests to a load balancer, which then directs the requests to machines from a pool of servers. Similar to our approach, this redirection can be performed by rewriting addresses, either IP addresses (network address translation (NAT), e.g., [8]) or MAC addresses (e.g., [9]). In contrast to most of these approaches, we avoid a dedicated

load balancer and rather utilize available (core) switches together with an SDN controller to implement request redirection.

A load balancing approach also utilizing SDN was proposed in [8]. Although this approach is based on NAT instead of MAC address rewriting, the problem of keeping established TCP connections alive is the same as in our approach. The solution proposed in [8] has two drawbacks: Either it redirects packets to the controller to distinguish between old and new connections, which might lead to a bottleneck at the (software) controller. Or it uses a heuristic (60 s timeout) to detect closed connections on the switch. This heuristic might either lead to broken connections if packets are sent after 60 s of inactivity. Or it might even prevent the handover since also new requests are forwarded to the old instance during the timeout period, which restarts the timer. We avoid these problems using readily available connection state from the end systems (HPI, LPMI), a feature enabled by the application-level SDN controller.

VI. SUMMARY

In this paper, we presented a concept to implement Elastic Tandem Machine Instances, which combine the benefits of low-power SoC hardware (low energy consumption) and high-power virtual machine instances (large resources). While the low-power instance serves low load and ensures constant availability, the high-power instance serves high load. We presented a concept to scale up ETMIs by switching adaptively from the low-power instance to the high-power instance. Using a handover protocol based on software-defined networking technologies, the transition is made seamlessly without disrupting existing connections. Moreover, we demonstrated the applicability of low-power SoC hardware to serve low load in realistic three-tier system settings, as well as a proof of concept of ETMIs.

REFERENCES

[1] X. Meng, C. Isci, J. Kephart, L. Zhang, E. Bouillet, and D. Pendarakis, “Efficient resource provisioning in compute clouds via VM multiplexing,” in Proc. of the 7th International Conference on Autonomic Computing (ICAC ’10), Washington, DC, Jun. 2010, pp. 11–20.

[2] B. Uranokar, P. Shenoy, and T. Roscoe, “Resource overbooking and application profiling in shared hosting platforms,” ACM SIGOPS Operating Systems Review, vol. 36, no. SI Winter 2002, pp. 239–254, Dec. 2002.

[3] D. Breitgand and A. Epstein, “Improving consolidation of virtual machines with risk-aware bandwidth oversubscription in compute clouds,” IBM Haifa Research Laboratory, Tech. Rep. H-0312 (H1201-006), Jan. 2012.

[4] W. Voorsluys, J. Broberg, S. Venugopal, and R. Buyya, “Cost of virtual machine live migration in clouds: A performanceevaluation,” in Proceedings of the 1st International Conference on Cloud Computing (CloudCom ’09), Beijing, China, Dec. 2009, pp. 254–265.

[5] M. Mao and M. Humphrey, “A performance study on the vm startup time in the cloud,” in Proc. of the 5th IEEE International Coonference on Cloud Computing (Cloud ’12), Honolulu, Hawaii, Jun. 2012, pp. 423–430.

[6] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, “OpenFlow: Enabling innovations in campus networks,” ACM SIGCOMM Computer Communication Review, vol. 38, no. 2, pp. 69–74, Apr. 2008.

[7] B.-G. Chun, G. Iannaccone, G. Iannaccone, R. Katz, G. Lee, and L. Niccolini, “An energy case for hybrid datacenters,” ACM SIGOPS Operating Systems Review, vol. 44, no. 1, pp. 76–80, Jan. 2010.

[8] R. Wang, D. Butnariu, and J. Rexford, “OpenFlow-based server load balancing gone wild,” in Proc. of the 11th Workshop on Hot Topis in Management of Internet, Cloud, and Enterprise Networks and Services (Hot-ICE ’11), Boston, MA, Mar. 2011, pp. 12–17.

[9] G. Goldszmidt and G. Hunt, “NetDispatcher: A TCP connection router,” IBM T.J. Watson Research Center, Tech. Rep. RC20853, May 1997.