Poster: Improving Server Utilization via Resource-adaptive Batch VMs

Seyyed Ahmad Javadi, Piyush Shyam Banginwar, Vaishali Chanana,
Rashmi Narvekar, Mitesh Kumar Savita, Anshul Gandhi
Department of Computer Science, Stony Brook University
{sjavadi,pbanginwar,vchanana,rnarvekar,msavita,anshul}@cs.stonybrook.edu

CCS Concepts • **Computer systems organization** → **Cloud computing**; • **Software and its engineering** → **Virtual machines**;

Keywords Server utilization, Resource management, VMs

ACM Reference format:
Seyyed Ahmad Javadi, Piyush Shyam Banginwar, Vaishali Chanana, Rashmi Narvekar, Mitesh Kumar Savita, Anshul Gandhi. 2017. Poster: Improving Server Utilization via Resource-adaptive Batch VMs. In *Proceedings of Middleware Posters and Demos '17: Proceedings of the Posters and Demos Session of the 18th International Middleware Conference, Las Vegas, NV, USA, December 11–15, 2017 (Middleware Posters and Demos '17),* 2 pages.
DOI: 10.1145/3155016.3155025

1 Introduction

Public cloud data centers often suffer from low resource utilization [1]. To increase utilization, recent works have proposed running batch workloads next to customer VMs to leverage idle resources [6]. While effective, the key challenge here is *interference* – the performance degradation of the colocated customer VMs due to resource contention with batch workload VMs at the underlying host server [5].

Prior work in this area typically resorts to terminating the batch workload VMs when the customer VMs' performance starts to degrade [3]. Public cloud providers also offer such services, e.g., Azure Batch VMs, that can be used to run batch workloads but can be preempted by the provider at any time to accommodate higher-priority VMs.

We propose a new class of batch VMs whose resource utilization can be dynamically regulated by the provider based on the usage of colocated VMs to minimize interference. To achieve this goal, we first analyze the resource consumption of Spark workloads and then investigate hypervisor capabilities that enable precise resource regulation of batch VMs. Experimental results on KVM hosts show that our batch VMs can increase resource utilization without affecting the performance of colocated latency-sensitive web workloads.

2 Experimental Setup

Figure 1 illustrates our experimental setup. We use several servers, referred to as PMs (Physical Machines), each with 8-core CPUs and 64 GB memory; we use KVM to deploy VMs

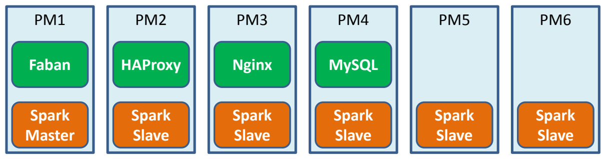


Figure 1. Illustration of our cloud setup.

on these PMs. To mimic a customer deployment, we set up the CloudSuite web serving benchmark on three VMs, each running in a separate PM, hosting the load balancer (HAProxy), web server (Nginx), and database server (MySQL), respectively; we refer to these as the *primary VMs*. For batch VMs, we deploy a Spark cluster running workloads from the spark-bench suite. We colocate a Spark slave VM with each of the three primary VMs, as depicted by PM2, PM3, and PM4 in Figure 1. We also make use of two PMs, PM5 and PM6, to host additional Spark slave VMs in specific experiments (see Section 3). To drive the primary application, we leverage the Faban workload generator, which is colocated with the Spark Master on a different PM.

3 Analyzing Resource Contention

The baseline plot (solid black line) in Figure 2 shows the 90%ile Response Times (RT) of CloudSuite when the colocated Spark batch VMs are idle. The blue dashed line shows the performance of CloudSuite when we run the KMeans workload on Spark; clearly, the colocated batch VMs severely impact the performance of the primary VMs.

To further analyze the interference due to Spark VMs, we also investigate TeraSort and LinearRegression workloads. Figure 3(a) shows the 90%ile RTs of CloudSuite when only its HAProxy load balancer VM is colocated with Spark (on PM2); we move the Spark Slave VMs initially colocated with the web server and database VMs of CloudSuite to PM5 and PM6. We see that, regardless of the Spark workload, CloudSuite performance is not affected significantly. Figures 3(b) and 3(c) highlight the impact of different Spark workloads

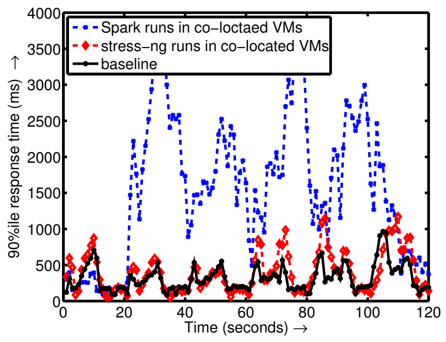
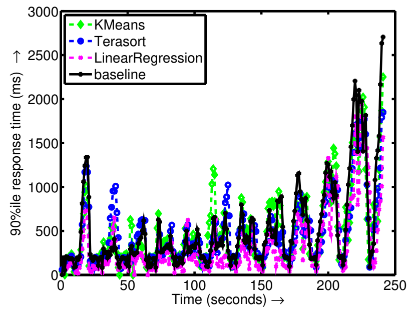
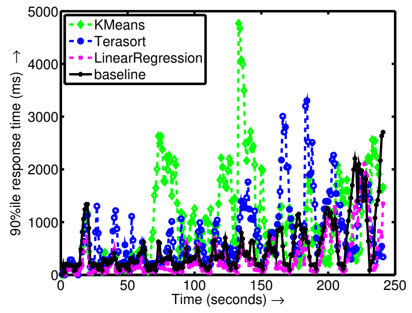


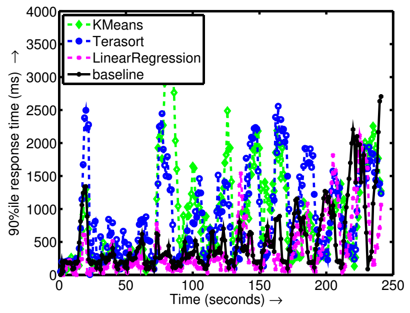
Figure 2. Performance of primary workload (CloudSuite).



(a) Spark colocated with load balancer.



(b) Spark colocated with web server.



(c) Spark colocated with database.

Figure 3. Impact of Spark workloads on different components of CloudSuite.

when colocated with only the Nginx web server and only the MySQL database of CloudSuite, respectively. This time, we see that the performance of CloudSuite is affected. However, on careful observation, we see that only KMeans and TeraSort impact CloudSuite performance significantly, and not LinearRegression. This is because KMeans and TeraSort jobs create high disk and network load when writing the data to HDFS during the shuffle stage; we verified this claim by monitoring the disk and network I/O traffic. As such, prior works that only focus on CPU interference mitigation [6] are ineffective in this case. The above results highlight the need for dynamically regulating the Spark resource usage based on the specific primary and batch workloads.

4 Dynamic Resource-Adaptive Batch VMs

To precisely regulate the resource consumption of batch VMs, we exploit existing capabilities of hypervisors and OSes.

Figure 4 shows the 90%ile RTs of CloudSuite under various resource regulation schemes for the background Spark batch VMs running KMeans (colocated only with MySQL). As before, baseline (solid black line) illustrates the performance of CloudSuite when colocated Spark VMs are idle. The dashed blue line shows the performance when Spark is running in colocated VMs and the CPU resources are fairly shared (1:1 shares) among colocated VMs using virsh [4] to control the resource allocation of KVM guests. Clearly, fair allocation results in significant performance interference. If we prioritize CPU for CloudSuite by giving it a 1000:1 shares ratio, the resulting performance (dashed magenta line), while better than 1:1 shares, is still quite variable; this suggests that CPU is not the only resource under contention. If we also limit the disk I/O usage (to 15MBps write bandwidth) for colocated Spark VMs using virsh, the resulting performance (dashed green line) is almost similar to the baseline performance, suggesting effective interference mitigation. Further, the resulting CPU usage of the physical host server increases by 116% when compared to the baseline, suggesting a substantial increase in resource efficiency. Thus, by carefully regulating the resource usage of batch VMs, we can increase resource efficiency while maintaining acceptable performance.

5 Related Work

Heracles [2] combines software and hardware isolation mechanisms to run batch jobs next to latency sensitive jobs. However, Heracles focuses on bare metal environments whereas we focus on virtualized environments and propose a new class

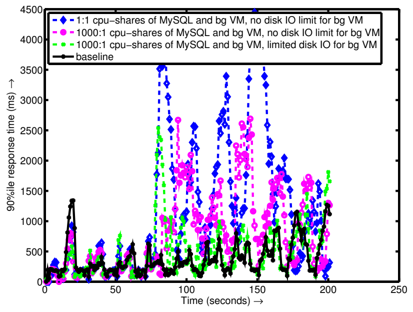


Figure 4. Experiment for cpu-shares and limited disk IO.

of VMs that adjust their resource usage based on colocated VMs. Zhang et al. [7] use historical data to determine the resource consumption treads of primary jobs and accordingly colocate long, medium, or short length batch tasks next to them. However, the batch tasks are killed when the primary job requires more resources, whereas our approach aims to throttle the batch VMs without killing them, thus minimizing the loss of batch workload progress.

6 Conclusion and Future Work

This paper makes the case for a new class of provider-controlled resource-adaptive VMs that can be launched next to customer VMs in public cloud data centers to increase resource efficiency. As part of future work, we will focus on regulating other resources, including memory and cache, and design an automated system that dynamically regulates resource consumption on batch VMs to minimize interference.

Acknowledgements

This work was partially supported by NSF grant CNS-1617046.

References

[1] C. Delimitrou et al. Quasar: Resource-efficient and QoS-aware Cluster Management. In *ASPLOS*, pages 127–144, 2014.
[2] D. Lo et al. Heracles: Improving Resource Efficiency at Scale. In *ACM SIGARCH Computer Architecture News*, volume 43, pages 450–462, 2015.
[3] Y. Yan et al. TR-Spark: Transient Computing for Big Data Analytics. In *SoCC*, pages 484–496, 2016.
[4] Linux man page. virsh(1). https://linux.die.net/man/1/virsh.
[5] S. A. Javadi et al. DIAL: Reducing Tail Latencies for Cloud Applications via Dynamic Interference-aware Load Balancing. In *Proceeding of ICAC 2017*, pages 135–144. IEEE, 2017.
[6] W. Zhang et al. Minimizing Interference and Maximizing Progress for Hadoop Virtual Machines. *SIGMETRICS Performance Evaluation Review*, 42(4):62–71, 2015.
[7] Y. Zhang et al. History-based Harvesting of Spare Cycles and Storage in Large-scale Datacenters. In *Proceedings of OSDI 2016*, pages 755–770, 2016.