Characterizing Co-located Datacenter Workloads: An Alibaba Case Study

Yue Cheng, Zheng Chai, Ali Anwar†

George Mason University †*IBM Research–Almaden*

1 INTRODUCTION

To improve resource utilization and thereby reduce costs, leading cloud infrastructure operators such as Google and Alibaba *co-locate* transient batch jobs with long-running, latency-sensitive jobs [9, 18] on the same cluster. Workload co-location resembles hypervisor-based server consolidation [8] but at datacenter scale. At its core, the driving force is what is called a DC/OS [19], managing job scheduling, resource allocation, and so on. As one example, Google's Borg [18] adopts the workload co-location technique by leveraging resource isolation provided by Linux containers [3].

Workload co-location has become a common practice [9, 12, 13, 15–18, 20]. To better facilitate the understanding of interactions among the co-located workloads and their real-world operational demands Alibaba recently released a cluster usage and co-located workload dataset [1], which was collected from Alibaba's production cluster in a 24-hour period.

We perform a characterization case study targeting Alibaba's co-located long-running and batch job workloads across several dimensions. We analyze the resource request and reservation patterns, resource usage, workload dynamicity, straggler issues, interaction and interference of co-located workloads, among other aspects. We make several unique insightful findings. Some of them may be specific to the Alibaba infrastructure, but we believe the generality is critical and applicable to designers, administrators, and users of co-located resource management systems.

Overbooking, over-provisioning, and over-commitment. Overbooking happens at long-running container deployment phase but just sparsely, only for few jobs that may not have strict overbooking requirements (e.g., CPU core sharing). Over-provisioning mainly happens for long-running containerized jobs for accommodating potential load spikes; but most time long-running jobs are resource-inactive, leaving co-located batch jobs ample space for elastic resource over-commitment for improved cluster utilization.

Co-location implications. High resource sharing means intricate resource contentions at different levels of the software stack, and potentially high performance interference. Our analysis also reveals evidences implying that Alibaba's workload-specific schedulers for long-running and batch jobs may not be as cohesively coordinated as they should, stressing the need for a more integrated, co-location-optimized solution.

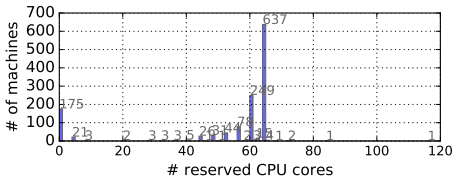


Figure 1: Histogram of reserved CPU cores.

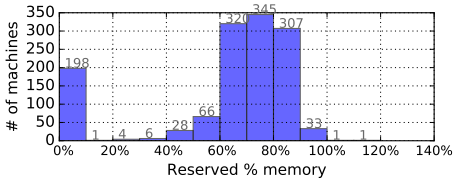


Figure 2: Histogram of reserved memory.

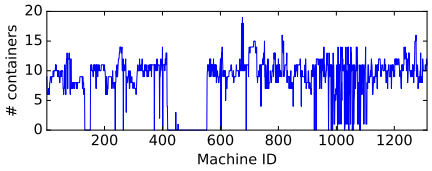


Figure 3: Distribution of containers across the cluster.

2 LONG-RUNNING JOB WORKLOADS

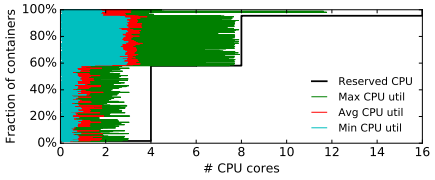
Resource reservation. We first calculate the per-machine resource amount requested and reserved by containerized long-running jobs from the container request trace (`container_event.csv`). Figure 1 and Figure 2 plot the same trend: for both CPU and memory allocation, overbooking is rare but does exist; about half of the machines (637) get all 64 cores reserved for containerized long-running applications; 60–80% of memory on 74% machines are reserved for containers. In fact, container management systems such as Sigma [4] need to consider a lot of other constraints when performing scheduling for long-running containers, including affinity and anti-affinity constraints (e.g., co-locating applications that belong to the same services for reducing network cost, or co-locating applications with complementary runtime behaviors), application priorities, whether or not the co-located applications tolerate resource overbooking of the same host machine [14]. A side effect of such multi-constraint multi-objective optimization is that the number of containers is unevenly distributed across the cluster as shown in Figure 3.

Resource Over-provisioning and Usage Dynamicity. To better understand the observed resource usage imbalance, we compare the reserved CPU and memory capacity with the actual usage, as shown in Figure 4. We make the following observations: (1) CPU resources are over-provisioned by all containers (Figure 4(a)), while

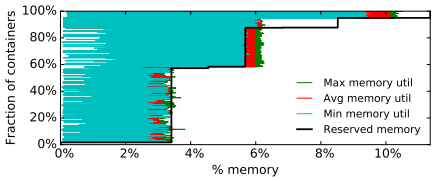
APSys '18, August 27–28, 2018, Jeju Island, Republic of Korea

ACM ISBN 978-1-4503-6006-7/18/08…$15.00
https://doi.org/10.1145/3265723.3265742

Yue Cheng, Zheng Chai, Ali Anwar†
George Mason University †*IBM Research–Almaden*

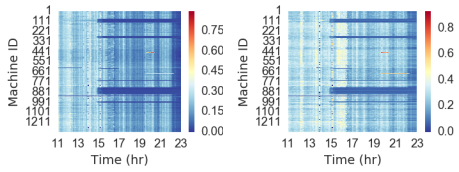


(a) CPU.



(b) Memory.

Figure 4: Reserved CPU and memory vs. their actual usage. We break down each container's 12-hour usage changes into {Min, Avg, Max} to show the dynamicity. The X-axis is sorted by the reserved resource amount.



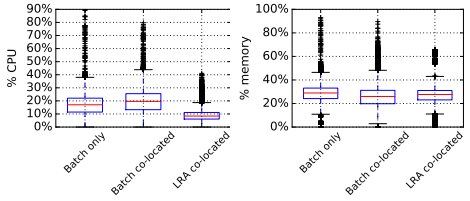
(a) CPU usage. (b) Memory usage.

Figure 5: Batch job resource usage (hour 11–23).

memory resources are over-committed by a large majority of containers (Figure 4(b)). (2) The CPU and memory request patterns are clearly visible—CPU requests have 4 distinguishable patterns while memory requests have 6. (3) ∼ 60% of containers are inactive in terms of CPU usage, having less than 1% average CPU utilization with the maximum percentile capped at 3%; average resource usage correlates to temporal stability—the more resource consumed on average, the higher the temporal variation tends to be—this is especially true for CPU; memory usage dynamicity is not as high as that of CPU. (4) Most containers have a higher average memory usage above 2%, which is as expected since containers need a minimum amount of memory to keep online services functional. This is consistent with the well-studied behaviors of web-scale distributed storage (memory and storage bound instead of CPU bound) workloads [5–7, 10, 11].

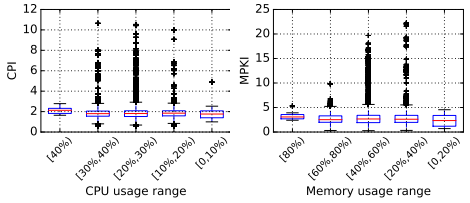
3 BATCH JOB WORKLOADS

The batch job workloads exhibit different resource patterns compared to that of containerized long-running job workloads. The vertical stripes (batch job waves) in Figure 5 is due to the dynamic nature of batch job workloads—task instances are transient and most of them finish in seconds. We can also observe that, within a single wave, the CPU and memory resource usage are roughly balanced across the cluster (later quantitatively demonstrated in §4), except for some regions with no batch jobs scheduled (i.e., the horizontal, dark blue stripes). This is because: (1) Fuxi is not constrained by data locality thanks to compute and storage disaggregated infrastructure at Alibaba (for batch jobs all data are stored and accessed



(a) CPU usage. (b) Memory usage.

Figure 6: Box-and-whisker plots showing CPU and memory usage distributions. `Batch only`: the machine region hosting batch jobs only; `Batch co-located`: batch jobs' resource usage in region hosting both batch and long-running applications (LRA); `LRA co-located`: LRA's resource usage in region hosting both.



(a) CPI. (b) MPKI.

Figure 7: Box-and-whisker plots showing maximum CPI and MPKI distribution as a function of machine's resource usage range (for both CPI and MPKI the lower the better).

remotely [2]), hence task instances can be scheduled anywhere there is enough resource; (2) Fuxi adopts an incremental scheduling heuristic that incrementally fulfills the resource demands at per-machine level [21].

4 WORKLOAD CO-LOCATION

We are particularly interested in how Fuxi [21] allocates resources in such `Batch only` machine region. We thus partition the cluster into a `Batch only` region where only batch jobs are running, and a `Co-located` region where both long-running and batch jobs are sharing the resources. Figure 6 depicts the CPU and memory resource usage distribution as a function of workload type and partitioned machine region. We observe that in `Batch only` region the average resource utilization is almost the same as that of batch jobs in `Co-located` region. This implies that: (1) `Batch only` region's resource utilization is significantly lower than that of the co-located region; and (2) Fuxi–the batch scheduler–does not take into account the resource usage heterogeneity caused by co-located long-running job workloads. The container trace records the runtime performance metrics including mean/maximum CPI (cycles per instruction) and MPKI (memory accesses per kilo-instructions). Figure 7 plots the maximum CPI and MPKI distributions at different resource usage ranges at per machine level. Statistically, both CPI and MPKI (the major percentiles e.g. medium) reaches the highest at highest resource utilization: 40%+ for CPU usage, and 80%+ for memory usage. Outliers at other usage ranges do exist and account for only a negligible set of data points. Note that the resource usage here accounts for both containerized long-running applications and non-containerized batch job workloads. This also implies that co-location tends to introduce performance interference when resource contention (i.e., resource usage) increases.

REFERENCES

[1] Alibaba production cluster data. https://github.com/alibaba/clusterdata.

[2] Alibaba Vendor BoF. https://www.usenix.org/conference/fast18/bofs#alibaba.

[3] Linux Containers. https://linuxcontainers.org/.

[4] Maximizing CPU Resource Utilization on Alibaba's Servers. https://102.alibaba.com/detail/?id=61.

[5] Anwar, A., Cheng, Y., Gupta, A., and Butt, A. R. Taming the cloud object storage with mos. In *Proceedings of the 10th Parallel Data Storage Workshop* (New York, NY, USA, 2015), PDSW '15, ACM, pp. 7–12.

[6] Anwar, A., Cheng, Y., Gupta, A., and Butt, A. R. Mos: Workload-aware elasticity for cloud object stores. In *Proceedings of the 25th ACM International Symposium on High-Performance Parallel and Distributed Computing* (New York, NY, USA, 2016), HPDC '16, ACM, pp. 177–188.

[7] Anwar, A., Mohamed, M., Tarasov, V., Littley, M., Rupprecht, L., Cheng, Y., Zhao, N., Skourtis, D., Warke, A. S., Ludwig, H., Hildebrand, D., and Butt, A. R. Improving docker registry design based on production workload analysis. In *16th USENIX Conference on File and Storage Technologies (FAST 18)* (Oakland, CA, 2018), USENIX Association, pp. 265–278.

[8] Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., and Warfield, A. Xen and the art of virtualization. In *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles* (New York, NY, USA, 2003), SOSP '03, ACM, pp. 164–177.

[9] Burns, B., Grant, B., Oppenheimer, D., Brewer, E., and Wilkes, J. Borg, omega, and kubernetes. *Queue 14*, 1 (Jan. 2016), 10:70–10:93.

[10] Cheng, Y., Gupta, A., and Butt, A. R. An in-memory object caching framework with adaptive load balancing. In *Proceedings of the Tenth European Conference on Computer Systems* (New York, NY, USA, 2015), EuroSys '15, ACM, pp. 4:1–4:16.

[11] Cidon, A., Rushton, D., Rumble, S. M., and Stutsman, R. Memshare: a dynamic multi-tenant key-value cache. In *2017 USENIX Annual Technical Conference (USENIX ATC 17)* (Santa Clara, CA, 2017), USENIX Association, pp. 321–334.

[12] Delimitrou, C., and Kozyrakis, C. Paragon: Qos-aware scheduling for heterogeneous datacenters. In *Proceedings of the Eighteenth International Conference on Architectural Support for Programming Languages and Operating Systems* (New York, NY, USA, 2013), ASPLOS '13, ACM, pp. 77–88.

[13] Delimitrou, C., and Kozyrakis, C. Quasar: Resource-efficient and qos-aware cluster management. In *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems* (New York, NY, USA, 2014), ASPLOS '14, ACM, pp. 127–144.

[14] Garefalakis, P., Karanasos, K., Pietzuch, P., Suresh, A., and Rao, S. Medea: Scheduling of long running applications in shared production clusters. In *Proceedings of the Thirteenth EuroSys Conference* (New York, NY, USA, 2018), EuroSys '18, ACM, pp. 4:1–4:13.

[15] Janus, P., and Rzadca, K. Slo-aware colocation of data center tasks based on instantaneous processor requirements. In *Proceedings of the 2017 Symposium on Cloud Computing* (New York, NY, USA, 2017), SoCC '17, ACM, pp. 256–268.

[16] Mars, J., Tang, L., Hundt, R., Skadron, K., and Soffa, M. L. Bubble-up: Increasing utilization in modern warehouse scale computers via sensible co-locations. In *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture* (New York, NY, USA, 2011), MICRO-44, ACM, pp. 248–259.

[17] Schwarzkopf, M., Konwinski, A., Abd-El-Malek, M., and Wilkes, J. Omega: Flexible, scalable schedulers for large compute clusters. In *Proceedings of the 8th ACM European Conference on Computer Systems* (New York, NY, USA, 2013), EuroSys '13, ACM, pp. 351–364.

[18] Verma, A., Pedrosa, L., Korupolu, M., Oppenheimer, D., Tune, E., and Wilkes, J. Large-scale cluster management at google with borg. In *Proceedings of the Tenth European Conference on Computer Systems* (New York, NY, USA, 2015), EuroSys '15, ACM, pp. 18:1–18:17.

[19] Zaharia, M., Hindman, B., Konwinski, A., Ghodsi, A., Joesph, A. D., Katz, R., Shenker, S., and Stoica, I. The datacenter needs an operating system. In *Proceedings of the 3rd USENIX Conference on Hot Topics in Cloud Computing* (Berkeley, CA, USA, 2011), HotCloud'11, USENIX Association, pp. 17–17.

[20] Zhang, X., Tune, E., Hagmann, R., Jnagal, R., Gokhale, V., and Wilkes, J. Cpi2: Cpu performance isolation for shared compute clusters. In *Proceedings of the 8th ACM European Conference on Computer Systems* (New York, NY, USA, 2013), EuroSys '13, ACM, pp. 379–391.

[21] Zhang, Z., Li, C., Tao, Y., Yang, R., Tang, H., and Xu, J. Fuxi: A fault-tolerant resource management and job scheduling system at internet scale. *Proc. VLDB Endow. 7*, 13 (Aug. 2014), 1393–1404.