CONMAN: Mitigating Shared Memory Interference to Maximize Cloud Host Utilization

Junsung Yook[0000-0001-8068-4884], Jorn Altmann[0000-0002-8880-9546] 1,2, and Bernhard Egger ✉[0000-0002-6645-6161] 1,2

[1] Seoul National University, Seoul, Republic of Korea
[2] Lucerne University of Applied Sciences and Arts, Lucerne, Switzerland
`{junsung,bernhard}@csap.snu.ac.kr, jorn.altmann@acm.org`

Abstract. Cost efficiency in cloud computing requires efficient resource utilization while meeting application performance objectives. A promising approach to improving cost efficiency is to consolidate lightly utilized services onto fewer physical servers, thereby increasing overall resource utilization. However, achieving high consolidation levels remains challenging due to resource interference among co-located workloads under dynamic load conditions. In particular, shared-memory interference can significantly degrade the performance of latency-critical (LC) services when multiple services are co-located on the same machine. Existing approaches commonly use memory bandwidth as an indicator of memory contention, but bandwidth alone does not accurately reflect the degree of shared-memory interference.

To address this limitation, we present *CONMAN*, a feedback-driven memory contention controller that maximizes the throughput of LC services while satisfying their required quality-of-service (QoS) targets. To accurately quantify shared-memory contention, CONMAN continuously profiles hardware performance counters to measure memory access latency as a direct indicator of interference, and adjusts resource allocation in real-time based on observed contention levels. Across highly consolidated environments with bursty memory-access patterns, CONMAN reduces tail latency by 53% on average compared to *Parties* and by 14% on average (up to 47%) compared to *Caladan*. As a result, CONMAN enables a larger number of compatible LC services to safely share a physical machine than prior systems, yielding a measurable improvement in server consolidation. The evaluation shows that, averaged over a variety of workloads and compared to a state-of-the-art approach, CONMAN can sustain 58% more co-located services while achieving the same QoS, translating into estimated annual savings of $3 million per 100,000 service instances.

Keywords: Cloud Computing, Server Consolidation, Shared-Memory Interference, Resource Contention, Quality of Service, Cost Efficiency

1 Introduction

As online cloud services continue to grow in scale and popularity, cost efficiency has become a primary concern for cloud providers. Cost efficiency means maximizing resource utilization while maintaining the desired performance objectives of hosted applications. A promising approach to improving cost efficiency is to consolidate lightly utilized services onto fewer physical servers, thereby reducing the number of machines required to operate a data center [4, 24, 27, 16]. The widespread adoption of microservice architectures further facilitates server consolidation by decomposing applications into modular services that can be flexibly co-located while maintaining scalability and reliability.

However, achieving high consolidation levels remains challenging due to resource interference among co-located workloads. When multiple services share the same physical machine, contention for shared resources can significantly degrade application performance under dynamic load conditions. In particular, latency-critical (LC) services are highly sensitive to shared-memory interference and often experience substantial performance degradation when co-located with other workloads. Best-effort workloads can disrupt latency-critical interactive services, resulting in increased tail latency and QoS violations. Moreover, modern cloud workloads exhibit considerable fluctuations in request rates and service times, making effective interference management essential for maintaining both service quality and overall system throughput [9].

Prior work has proposed various mechanisms to mitigate resource interference in consolidated environments. *Caladan* [7] dynamically reallocates resources at microsecond timescales, enabling rapid reactions to changes in workload demand and supporting fine-grained per-request resource management. To mitigate the performance impact of shared-memory contention, Caladan's memory bandwidth controller relies on bandwidth saturation thresholds to detect memory contention and trigger corrective actions. *Parties* [2] has adapted to the shift towards a microservice architecture where many latency-critical and interactive services need to be scheduled on one physical host to maximize utilization.

Despite their effectiveness, existing approaches leave an important question unanswered: what is the most accurate indicator of shared-memory interference? Most prior systems rely on memory bandwidth utilization as an indicator of memory contention [2, 7]. However, this alone does not directly capture the performance impact of shared-memory interference and often fails to accurately reflect the degree of contention experienced by latency-critical services. As a result, bandwidth-based control can make suboptimal resource-management decisions and fail to satisfy strict latency objectives under dynamic workloads.

To address this limitation, we present *CONMAN*, a feedback-driven memory contention controller that maximizes the throughput of latency-critical services while satisfying their required QoS targets. Rather than relying on memory bandwidth as an indirect signal, CONMAN profiles hardware performance counters (PMUs) to measure memory access latency as a direct indicator of interference, and adjusts resource allocation in real time based on observed contention levels. By directly measuring the performance impact of contention, CONMAN enables more accurate and responsive memory-interference management.

The key insight behind CONMAN is that memory-access latency captures shared-memory contention more accurately than bandwidth utilization. Leveraging this observation, CONMAN maximizes the number of latency-critical services that can be safely consolidated on a single physical host machine while maintaining their performance objectives.

Our contributions are as follows:

- We present CONMAN, a feedback-driven memory contention controller that uses memory access latency to directly quantify shared-memory interference.
- We demonstrate that CONMAN reduces tail latency by 53% on average compared with Parties and by 14% on average (up to 47%) compared with Caladan, enabling higher consolidation of latency-critical services and improving cost efficiency.

2 Economic Background and Motivation

2.1 Total Cost of Ownership in Modern Datacenters

Capital expenditures (CapEx) and operational expenditures (OpEx) associated with physical servers constitute a significant fraction of the total cost of ownership (TCO) of modern data centers [25, 8]. Server costs typically break down across three major categories: hardware acquisition and amortization, energy consumption (including cooling), and software licensing. Eliminating a single physical server saves approximately $500 in annual energy costs, $500 in software licensing fees, and $1,500 in hardware maintenance expenses [10], for a combined saving of $2,500 per server per year. At datacenter scale where fleets commonly number in the tens of thousands of servers, even modest improvements in consolidation efficiency translate into substantial annual savings.

2.2 The Consolidation–Interference Trade-off

Despite these incentives, most cloud servers operate substantially below their peak capacity. Studies report average CPU utilization in production data centers in the range of 10–50% [25], leaving large fractions of provisioned capacity idle. Server consolidation, that is, co-locating multiple services on fewer physical hosts, is the primary mechanism by which cloud providers reclaim this headroom. However, consolidation introduces *resource interference*: when co-located workloads compete for shared hardware resources (memory bandwidth, last-level cache, I/O), the resulting contention degrades the performance of latency-critical (LC) services, leading to QoS violations.

QoS violations carry direct economic consequences. Cloud providers typically offer service-level agreements (SLAs) that impose financial penalties when tail-latency targets are breached. Even in the absence of explicit SLA penalties, performance degradation causes customer attrition: controlled experiments at Google and Microsoft show that server-side delays of 500 ms reduce revenue per user by approximately 1% [23], and that these effects scale roughly linearly with delay magnitude. Consequently, operators must provision conservative consolidation ratios that leave substantial headroom for interference, effectively limiting the economic benefit of consolidation.

2.3 Cost Model for Consolidation Efficiency

Let S denote the number of LC service instances hosted on a single physical server, and let C_{server} denote the annualized cost of operating one physical server. The *cost per service instance* is:

$$c = \frac{C_{\text{server}}}{S}$$

Improving the achievable consolidation ratio from S to $S' > S$ (while preserving QoS) reduces the cost per instance by a factor of S/S', and reduces the total number of servers required to host a fixed fleet of N service instances from $\lceil N/S \rceil$ to $\lceil N/S' \rceil$. The annualized savings are therefore:

$$\Delta_{\text{cost}} = \left(\left\lceil \frac{N}{S} \right\rceil - \left\lceil \frac{N}{S'} \right\rceil \right) \times C_{\text{server}}$$

For a fleet of $N = 1{,}000$ service instances with $C_{\text{server}} = \$2{,}500/\text{year}$, increasing the consolidation ratio by even 10%—from $S = 10$ to $S' = 11$ services per host—eliminates approximately 91 servers and saves roughly $228,000 annually. At hyperscale (millions of instances), the savings scale proportionally.

This cost model establishes a direct link between system-level interference management and cloud economic efficiency: the primary lever is the maximum QoS-safe consolidation ratio S^*, which is determined by how accurately and responsively a system can detect and mitigate memory contention.

3 Related Work

Prior work has extensively studied mechanisms for mitigating memory interference in consolidated systems. *Heracles* [13] and Parties [2] use DRAM bandwidth utilization as a proxy for memory contention and regulate resource allocation when bandwidth consumption approaches saturation. A large body of work has proposed memory scheduling mechanisms that improve throughput and fairness by managing contention among competing memory requests [11, 15, 26, 22, 19]. For example, *Thread Cluster Memory Scheduling* [12] improves system performance by separating latency-sensitive and bandwidth-sensitive threads and applying different scheduling policies to each cluster. *Twig* [18] dynamically adjusts resource allocations to satisfy the QoS requirements of latency-critical services while improving overall resource utilization. Other studies have addressed memory interference through bandwidth-aware scheduling and CPU-allocation techniques that regulate workload execution according to observed bandwidth demand [28, 14, 3, 21, 20, 5]. Collectively, these approaches share the common assumption that memory bandwidth utilization serves as an effective indicator of memory contention.

4 Challenges

In this section, we examine two representative memory-access patterns and demonstrate the limitations of existing memory-contention indicators for characterizing shared-memory interference. In particular, we investigate the following research question: *Can memory bandwidth accurately indicate the performance impact of memory interference on latency-critical services?*

4.1 Experimental Setup

Hardware Setup We conduct our experiments on a dual-socket server equipped with two 16-core Intel Xeon Silver 4215R processors running at 3.2 GHz and 128GiB of DRAM. The system runs Ubuntu 24.04 with Linux kernel 6.8.0-39. One socket is dedicated to service execution, while the other is used to generate memory interference. To minimize measurement noise, hyper-threading, Turbo Boost, C-states, P-states, and CPU frequency scaling are disabled.

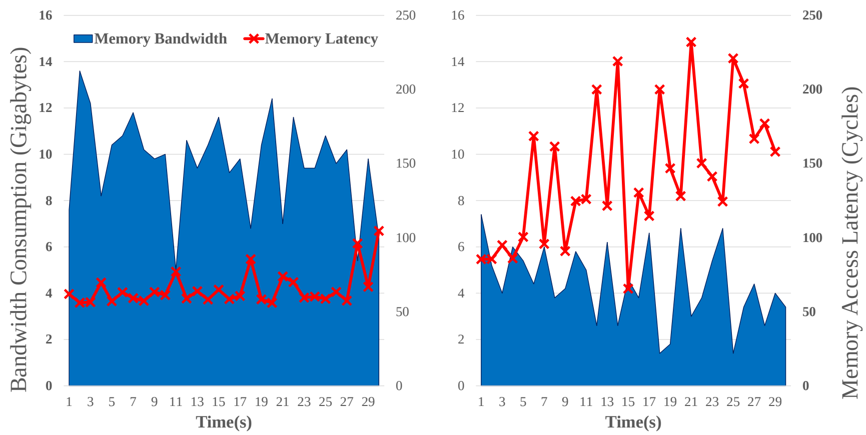
Workloads To characterize different memory-access behaviors, we develop microservice benchmarks that generate either constant or burst memory-access patterns. Each benchmark consists of up to eight concurrent sessions that continuously issue memory requests. Requests are generated periodically according to either a constant-rate or bursty request pattern.

Empirical Observation Figure 1 reports memory bandwidth consumption and memory access latency measured using hardware performance monitoring units (PMUs) over time. Despite consuming less memory bandwidth, the burst memory-access pattern exhibits substantially higher memory access latency than the constant memory-access pattern. This result indicates that memory bandwidth and memory access latency are not necessarily correlated and that low bandwidth consumption can still result in severe memory interference.

4.2 Design Rationale

Our goal is to maximize the throughput of latency-critical services while satisfying their performance requirements, thereby enabling a larger number of services to be consolidated on a single physical machine. Achieving this goal requires accurately mitigating shared-memory interference. A key challenge is that the performance impact of memory interference is primarily manifested as CPU stall cycles caused by outstanding memory accesses.

Limitations of Bandwidth-Based Indicators. Existing systems commonly use memory bandwidth consumption as an indicator of memory contention. However, bandwidth-aware contention indicators, which rely on memory access throughput, are often insufficient to capture transient bursts in memory demand. Because bandwidth is fundamentally a throughput-oriented metric, accurately interpreting bandwidth consumption requires knowledge of request arrival patterns and memory-access behavior and thus cannot directly reveal the degree



(a) Constant Memory Access Pattern **(b) Burst Memory Access Pattern**

Fig. 1. Burst memory accesses incur higher latency despite lower bandwidth consumption. Memory bandwidth consumption and memory access latency over time. Despite lower memory bandwidth consumption, the burst memory-access pattern exhibits substantially higher memory access latency than the constant memory-access pattern.

of memory contention. As shown in Figure 1, workloads with lower bandwidth consumption can experience significantly higher memory access latency. This mismatch indicates that bandwidth utilization does not reliably reflect the degree of shared-memory interference, leading to inaccurate contention detection and ineffective resource-management decisions.

5 CONMAN

To address these limitations, we present CONMAN, a feedback-driven memory contention controller that maximizes the throughput of latency-critical services while satisfying their performance requirements under bursty memory-access behaviors. Rather than relying on memory bandwidth as an indirect signal of contention, CONMAN exploits memory access latency as a direct indicator of shared-memory interference and continuously adjusts resource allocation based on observed contention levels. By directly measuring the performance impact of interference, CONMAN enables more accurate and responsive memory-interference management.

5.1 Overview

Figure 2 presents an overview of the CONMAN architecture. CONMAN periodically measures memory access latency every 100 μs using hardware performance

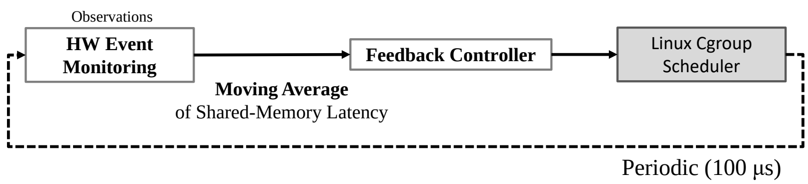


Fig. 2. Controller architecture

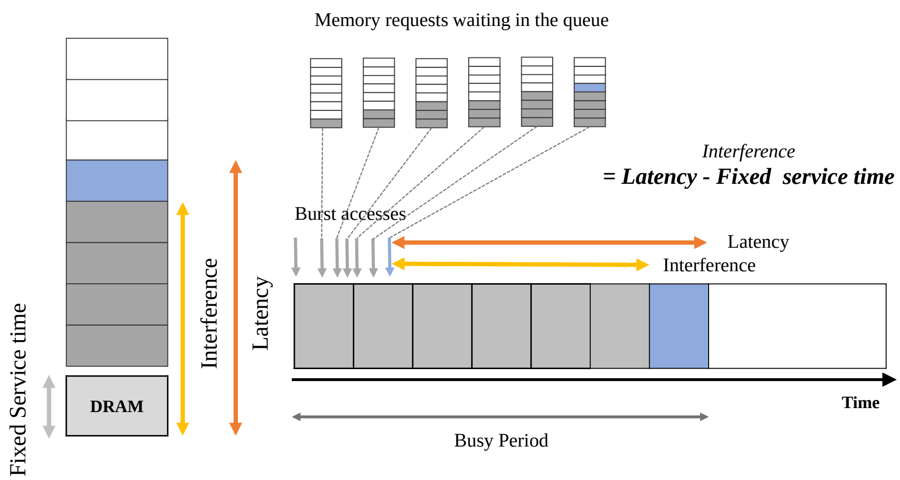


Fig. 3. Memory latency as a memory-interference indicator

counters. Based on the measured latency, a feedback controller determines the appropriate level of CPU provisioning for best-effort (BE) applications. The resulting control decisions are enforced through Linux cgroup CPU controllers, which dynamically adjust the CPU allocation of BE applications. By regulating BE execution intensity, CONMAN mitigates shared-memory interference and preserves the performance of latency-critical services.

5.2 Latency as a Memory-Interference Indicator

Memory access latency is an accurate and low-overhead indicator of shared-memory interference. Unlike memory bandwidth consumption, memory access latency directly reflects the performance impact of contention in the memory hierarchy. As illustrated in Figure 3, the observed memory latency consists of a fixed memory service time and a variable waiting time caused by contention for shared memory resources. Because the fixed service time remains largely unchanged, variations in memory access latency directly capture changes in the degree of memory interference.

In contrast, interpreting memory bandwidth consumption requires knowledge of the underlying memory-request arrival process. Since memory requests are highly dependent and exhibit bursty behavior, bandwidth utilization alone does not reliably indicate contention intensity. As demonstrated in Section 4.1, workloads with lower bandwidth consumption can experience substantially higher memory access latency. Consequently, bandwidth-based approaches can produce false positives and false negatives when identifying memory contention.

5.3 Measuring Memory Access Latency

CONMAN estimates memory access latency using Little's Law:

$$L = \frac{Q}{\lambda}$$

where L denotes the average memory access latency, Q denotes the average queue occupancy, and λ denotes the average memory request arrival rate.

Little's Law is independent of workload characteristics, arrival distributions, service-time distributions, and scheduling policies. By applying Little's Law to processor CHA (Caching and Home Agent) PMU events, CONMAN efficiently estimates the average memory access latency with minimal runtime overhead.

Algorithm 1 Feedback Control Algorithm

1: $setpoint \leftarrow get\_moving\_average(WIDTH)$;
2: $latency \leftarrow memory\_latency$;
3: **if** $setpoint > latency$ **then**
4: $BE_{allocation} \leftarrow 1.1 \times BE_{allocation}$ ▷ Increase BE CPU allocation by 10%
5: **else**
6: $BE_{allocation} \leftarrow 0.9 \times BE_{allocation}$ ▷ Decrease BE CPU allocation by 10%
7: **end if**

5.4 Feedback Controller

The objective of the feedback controller is to mitigate shared-memory interference by reducing CPU stall cycles caused by memory-access latency. Since memory interference is amplified when memory requests arrive in bursts, the controller seeks to smooth memory demand over time by regulating the execution intensity of BE applications.

CONMAN uses the moving average of memory access latency as the target operating point. At each control interval, the controller compares the current average memory access latency with the moving-average latency and uses the resulting error as a control signal. The controller then adjusts BE resource allocations to drive the current latency toward the target operating point.

Algorithm 1 summarizes the feedback-control procedure used by CONMAN to manage the memory intensiveness of BE applications and reduce shared-memory interference. Because memory access latency directly reflects contention-induced waiting time, the controller can accurately compare interference levels across different execution periods and react accordingly. CPU allocations of BE applications are adjusted through Linux cgroup CPU controllers. Specifically, CONMAN dynamically increases or decreases the CPU allocation of BE applications by modifying their `cpu.max` configuration, thereby regulating memory demand and mitigating interference.

Table 1. Latency-critical application services

Server	Domain	Mean Service Time	Memory Load
Memcached	Key-value store	60µs	non-intensive
NGINX	Web server	5ms	non-intensive
MariaDB	OLAP	50ms	intensive

6 Evaluation

We evaluate CONMAN by answering the following question: **How does CONMAN compare with prior approaches as memory contention increases?**

6.1 Experimental Setup

The hardware configuration is identical to that described in Section 4.1.

Workloads To generate memory contention, we deploy multiple latency-critical (LC) services and issue requests according to a Poisson arrival process. Table 1 summarizes the mean service time and memory load characteristics of each LC service. For all experiments, the expected inter-arrival time is configured to be ten times the mean service time of the corresponding LC service.

- NGINX is a high-performance HTTP server [17]. We use NGINX 1.27.0 as a front-end server for static content. The workload consists of multiple HTML files, each 32 KB in size.
- Memcached is a high-performance distributed memory object caching system [6].
- MariaDB is an open-source relational database management system widely used for its flexibility and cost efficiency.
- For BE applications, we use the NAS Parallel Benchmarks (NPB) [1], a widely adopted benchmark suite for evaluating parallel computing systems.

Each LC service is allocated 12 CPU cores. This configuration ensures that CPU interference remains negligible even when up to 60 LC services are co-located on the same machine. The remaining four CPU cores are dedicated to BE applications.

Baselines We compare CONMAN against two state-of-the-art cloud resource-management systems: Parties [2] and Caladan [7].

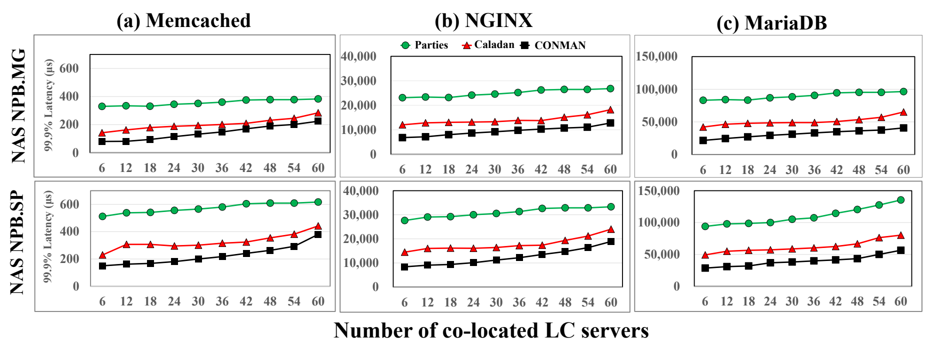


Fig. 4. 99.9th-percentile tail latency as the number of co-located LC services increases. Tail latency is reduced by up to 47% over Caladan.

6.2 Comparison of 99.9th-Percentile Tail Latency

Figure 4 shows the 99.9th-percentile tail latency as the number of co-located LC services increases. Increasing the number of co-located services intensifies shared-memory contention and consequently increases memory interference. The results demonstrate how accurately each system detects and mitigates contention-induced interference under increasing memory pressure.

Across all workloads, CONMAN consistently achieves the lowest 99.9th-percentile tail latency. On average, CONMAN reduces tail latency by 53% over Parties and by 14% over Caladan. The maximum improvement over Caladan reaches 47% in terms of 99.9th-percentile tail latency reduction. These results demonstrate that memory-access latency is a more effective indicator of shared-memory interference than bandwidth utilization, enabling more accurate contention management and better preservation of application performance requirements in highly consolidated environments.

Bandwidth-based contention indicators often fail to capture transient bursts in memory demand because they primarily reflect memory-access throughput. In contrast, memory-access latency directly exposes contention-induced delays and provides a more sensitive signal of bursty access behavior. As a result, latency-aware contention management enables timely contention detection and mitigates tail-latency degradation under bursty memory-access patterns.

6.3 Economic Impact of Improved Consolidation

To translate tail-latency improvements into economic terms, we determine for each system the maximum QoS-safe consolidation ratio S^*, defined as the largest number of co-located LC services whose 99.9th-percentile tail latency remains within the threshold achieved by CONMAN at 60 co-located services. Table 2 reports S^* for each workload, averaged across both BE benchmarks (NAS NPB.MG and NAS NPB.SP), along with the estimated annual cost savings relative to Caladan when operating a fleet of 100,000 service instances (C_{server} = \$2,500/year). Parties is excluded from this analysis as it fails to satisfy the QoS threshold across all evaluated workloads.

Table 2. Maximum QoS-safe consolidation ratio S^* and estimated annual savings vs. Caladan (fleet of 100,000 instances, \$2,500/server/year).

Workload	Caladan	CONMAN	Savings vs. Caladan
Memcached	51	60	$750,000
NGINX	45	60	$1,400,000
MariaDB	18	60	$9,725,000

Across all workloads, CONMAN achieves $S^* = 60$, the maximum evaluated, while Caladan falls short to varying degrees depending on workload memory intensity. The gains are most pronounced for MariaDB, the most memory-intensive workload: Caladan can only safely consolidate 18 services per host, whereas CONMAN sustains all 60, reducing the number of servers required to host 100,000 instances from 5,556 to 1,667 and saving an estimated $9,725,000 per year. For the less memory-intensive Memcached and NGINX workloads, Caladan comes closer but still requires 18–25% more servers than CONMAN, translating into $750,000–$1,400,000 in annual savings per 100,000 instances.

These results confirm that accurate interference detection is not merely a performance engineering concern but has direct and quantifiable economic implications for cloud infrastructure costs.

7 Discussion and Conclusion

This paper presented CONMAN, a feedback-driven memory contention controller for mitigating shared-memory interference in consolidated cloud environments. CONMAN introduces a memory latency-based metric that directly captures the performance impact of shared-memory contention and enables more accurate interference detection than bandwidth-based approaches. Using memory access latency as a feedback signal, CONMAN dynamically regulates the resource allocation of BE applications through Linux cgroup CPU controllers.

Our evaluation demonstrates that CONMAN consistently outperforms prior state-of-the-art systems and achieves a higher QoS-safe consolidation ratio than

Caladan across all evaluated workloads. Translating these gains into economic terms, we estimate annual infrastructure savings of $750,000–$9,725,000 per 100,000 service instances depending on workload memory intensity, scaling to tens of millions of dollars at hyperscale. These findings position accurate interference detection as a key lever for improving cloud cost efficiency, complementing existing work on pricing models, capacity planning, and SLA design.

Acknowledgments. We thank the anonymous reviewers for their helpful feedback. This work was funded, in part, by the Korean National Research Foundation (grant no. 21A20151113068 – BK21 Plus for Pioneers in Innovative Computing - Dept. of Computer Science & Engineering, SNU), by the Ministry of Trade, Industry and Energy (MOTIE) and the Korea Evaluation Institute of Industrial Technology (KEIT) (grant no. 10077609), by the Ministry of Science and ICT (MSIT) (grant no. RS-2023-00302083), and by the EU Horizon program within the f-inference project (grant no. 101298393). ICT at Seoul National University provided research facilities for this study.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. Bailey, D., Harris, T., Saphir, W., Van Der Wijngaart, R., Woo, A., Yarrow, M.: The nas parallel benchmarks 2.0. Tech. rep., Technical Report NAS-95-020, NASA Ames Research Center (1995)
2. Chen, S., Delimitrou, C., Martínez, J.F.: Parties: Qos-aware resource partitioning for multiple interactive services. In: Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems. pp. 107–120 (2019)
3. Cho, Y., Guzman, C.A.C., Egger, B.: Maximizing system utilization via parallelism management for co-located parallel applications. In: Proceedings of the 2018 International Conference on Parallel Architectures and Compilation Techniques (PACT). pp. 35–48. ACM (2018). https://doi.org/10.1145/3243176.3243191, https://doi.org/10.1145/3243176.3243191
4. Dragoni, N., Giallorenzo, S., Lafuente, A.L., Mazzara, M., Montesi, F., Mustafin, R., Safina, L.: Microservices: yesterday, today, and tomorrow. Present and ulterior software engineering pp. 195–216 (2017)
5. Feliu, J., Petit, S., Sahuquillo, J., Duato, J.: Cache-hierarchy contention-aware scheduling in cmps. IEEE Transactions on Parallel and Distributed Systems **25**(3), 581–590 (2013)
6. Fitzpatrick, B.: Distributed caching with memcached. Linux journal **2004**(124), 5
7. Fried, J., Ruan, Z., Ousterhout, A., Belay, A.: Caladan: Mitigating interference at microsecond timescales. In: 14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20). pp. 281–297 (2020)
8. Furht, B., Escalante, A., et al.: Handbook of cloud computing, vol. 3. Springer (2010)
9. Gong, C., Liu, J., Zhang, Q., Chen, H., Gong, Z.: The characteristics of cloud computing. In: 2010 39th International Conference on Parallel Processing Workshops. pp. 275–279. IEEE (2010)

10. Heslin, K.: Decommissioning as a discipline: Server Roundup winners share success. Uptime Institute Journal (Aug 2014), https://journal.uptimeinstitute.com/decommissioning-discipline-server-roundup-winners-share-success/
11. Kim, S., Genc, H., Nikiforov, V.V., Asanović, K., Nikolić, B., Shao, Y.S.: Moca: Memory-centric, adaptive execution for multi-tenant deep neural networks. In: 2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA). pp. 828–841 (2023). https://doi.org/10.1109/HPCA56546.2023.10071035
12. Kim, Y., Papamichael, M., Mutlu, O., Harchol-Balter, M.: Thread cluster memory scheduling: Exploiting differences in memory access behavior. In: 2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture. pp. 65–76. IEEE (2010)
13. Lo, D., Cheng, L., Govindaraju, R., Ranganathan, P., Kozyrakis, C.: Heracles: Improving resource efficiency at scale. In: Proceedings of the 42nd Annual International Symposium on Computer Architecture. pp. 450–462 (2015)
14. Mars, J., Tang, 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 (MICRO). pp. 248–259. ACM (2011). https://doi.org/10.1145/2155620.2155654, https://doi.org/10.1145/2155620.2155654
15. Muralidhara, S.P., Subramanian, L., Mutlu, O., Kandemir, M.T., Moscibroda, T.: Reducing memory interference in multicore systems via application-aware memory channel partitioning. In: Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO). pp. 374–385. IEEE (2011). https://doi.org/10.1109/MICRO.2011.6119152, https://doi.org/10.1109/MICRO.2011.6119152
16. Newman, S.: Building microservices. " O'Reilly Media, Inc." (2021)
17. NGINX: Nginx webserver (2021), https://nginx.org.
18. Nishtala, R., Petrucci, V., Carpenter, P., Sjalander, M.: Twig: Multi-agent task management for colocated latency-critical cloud services. In: 2020 IEEE International Symposium on High Performance Computer Architecture (HPCA). pp. 167–179. IEEE (2020)
19. Park, J., Park, S., Baek, W.: Copart: Coordinated partitioning of last-level cache and memory bandwidth for fairness-aware workload consolidation on commodity servers. In: Proceedings of the Fourteenth EuroSys Conference 2019. pp. 10:1–10:16. ACM (2019). https://doi.org/10.1145/3302424.3303963, https://doi.org/10.1145/3302424.3303963
20. Pérez, J.F., Borrás, J.S., Martí, S.V.P., Marín, J.F.D.: L1-bandwidth aware thread allocation in multicore smt processors. In: Proceedings of the 22nd International Conference on Parallel Architectures and Compilation Techniques (PACT 2013). pp. 123–132. IEEE (2013). https://doi.org/10.1109/PACT.2013.6618810, https://doi.org/10.1109/PACT.2013.6618810
21. Pérez, J.F., Martí, S.V.P., Sahuquillo, J., Duato, J.: Cache-hierarchy contention aware scheduling in cmps. In: Proceedings of the 2012 IEEE 26th International Parallel and Distributed Processing Symposium (IPDPS). pp. 1033–1044. IEEE (2012). https://doi.org/10.1109/IPDPS.2012.95, https://doi.org/10.1109/IPDPS.2012.95
22. Rixner, S., Dally, W.J., Kapasi, U.J., Mattson, P., Owens, J.D.: Memory access scheduling. SIGARCH Computer Architecture News **28**(2), 128–138 (2000). https://doi.org/10.1145/339647.339668, https://doi.org/10.1145/339647.339668

23. Schurman, E., Brutlag, J.: Performance related changes and their user impact. In: Proceedings of the O'Reilly Velocity Web Performance and Operations Conference. San Jose, CA, USA (Jun 2009)
24. Shahrad, M., Fonseca, R., Goiri, I., Chaudhry, G., Batum, P., Cooke, J., Laureano, E., Tresness, C., Russinovich, M., Bianchini, R.: Serverless in the wild: Characterizing and optimizing the serverless workload at a large cloud provider. In: 2020 USENIX annual technical conference (USENIX ATC 20). pp. 205–218 (2020)
25. Shehabi, A., Smith, S., Sartor, D., Brown, R., Herrlin, M., Koomey, J., Masanet, E., Horner, N., Azevedo, I., Lintner, W.: United states data center energy usage report (2016)
26. Subramanian, L., Seshadri, V., Ghosh, A., Khan, S., Mutlu, O.: The application slowdown model: Quantifying and controlling the impact of inter-application interference at shared caches and main memory. In: Proceedings of the 48th International Symposium on Microarchitecture, MICRO 2015, Waikiki, HI, USA, December 5-9, 2015. pp. 62–75. ACM (2015). https://doi.org/10.1145/2830772.2830803, https://doi.org/10.1145/2830772.2830803
27. Thönes, J.: Microservices. IEEE software **32**(1), 116–116 (2015)
28. Zhuravlev, S., Blagodurov, S., Fedorova, A.: Addressing shared resource contention in multicore processors via scheduling. In: Proceedings of the Fifteenth Edition of ASPLOS on Architectural Support for Programming Languages and Operating Systems (ASPLOS). pp. 129–142. ACM (2010). https://doi.org/10.1145/1736020.1736036, https://doi.org/10.1145/1736020.1736036