Brief Announcement: BALM: QoS-Aware Memory Bandwidth Partitioning for Multi-Socket Cloud Nodes

David Gureya*
INESC-ID, Universidade de Lisboa
Portugal
daharewa@kth.se

Vladimir Vlassov
KTH Royal Institute of Technology
Sweden
vladv@kth.se

João Barreto
INESC-ID, Universidade de Lisboa
Portugal
joao.barreto@tecnico.ulisboa.pt

Abstract

The recent emergence of novel hardware-based resource partitioning mechanisms has unveiled the opportunity for a new generation of QoS-aware resource allocation approaches for workload consolidation. Still, to the best of our knowledge, existing proposals are, by design, not tailored to the growing prevalence of multi-socket systems in contemporary warehouse-scale data centers. We propose BALM, a QoS-aware memory bandwidth allocation technique for multi-socket architectures that combines commodity bandwidth allocation mechanisms with a novel adaptive cross-socket page migration scheme. Our experimental evaluation with real applications on a dual-socket machine shows that BALM can overcome the efficiency limitations of state-of-the-art. BALM can ensure marginal SLO violation windows while delivering up to 87% throughput gains to bandwidth-intensive best-effort applications when compared to state-of-the-art alternatives.

CCS Concepts

• **Computer systems organization → Multicore architectures**.

Keywords

QoS-aware resource allocation, Cloud computing, Multi-socket systems

ACM Reference Format:
David Gureya, Vladimir Vlassov, and João Barreto. 2021. Brief Announcement: BALM: QoS-Aware Memory Bandwidth Partitioning for Multi-Socket Cloud Nodes. In *Proceedings of the 33rd ACM Symposium on Parallelism in Algorithms and Architectures (SPAA '21), July 6–8, 2021, Virtual Event, USA.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3409964.3461823

1 Introduction

The business models for cloud and data center computing emphasize reducing infrastructural costs. One primary way to achieve this is through *workload consolidation*, i.e., by co-locating applications on the same physical host. Some co-located applications have quality of service (QoS) requirements, as determined by one or more service-level objectives (SLOs). These are commonly called *latency-critical* applications (LCAs). In contrast, the so-called *best-effort* applications (BEAs) have no SLO and are meant to run in some best-effort fashion that aims to maximize their throughput.

The co-located applications contend for shared resources, such as network and storage bandwidth, CPU cores, last-level caches (LLC), and memory. This poses a challenging *QoS-aware resource allocation problem*: the shared resources should be allocated in such a way that safeguards the SLO of the LCAs while maximizing the throughput of the BEAs. Although this problem is not new, the recent emergence of novel hardware-based resource partitioning mechanisms has unveiled the opportunity for a new generation of QoS-aware resource allocation approaches. One notable example of such mechanisms is Intel Resource Director Technology's (RDT) support for hardware-based memory bandwidth allocation (MBA). Recent proposals such as PARTIES [3] and CLITE [8] take advantage of such new mechanisms to enforce QoS-aware resource allocation with unprecedented effectiveness.

Another technological trend with a significant impact on workload consolidation is the growing prevalence of multi-socket systems in contemporary data centers [9]. Unfortunately, PARTIES and CLITE, as well as the vast majority of their predecessors, are, by design, tailored to single-socket architectures only.

This paper advocates that, in order to properly utilize over-provisioned memory resources in multi-socket hosts, state-of-the-art QoS-aware resource allocation systems need to be generalized to allow cross-socket sharing of memory. To achieve that, the low-level memory bandwidth partitioning mechanisms on which existing solutions rely need to be redesigned to address the new constraints of multi-socket architectures.

We propose BALM (memory Bandwidth ALlocation for Multi-socket), a novel QoS-aware memory bandwidth allocation technique for cross-socket sharing of memory in multi-socket architectures. The key insight of BALM is to combine commodity bandwidth allocation mechanisms originally designed for single-socket (such as MBA) with a novel adaptive cross-socket page migration scheme. By doing so, BALM can overcome the limitations of the original mechanisms when deployed in multi-socket scenarios. We evaluate BALM by co-locating, in a dual-socket system, a real LCA (namely, the Memcached key-value store [5]) and different realistic memory-intensive workloads. Our evaluation shows that BALM can safeguard the LCA with marginal SLO violation windows, while delivering up to 87% throughput gains to bandwidth-intensive BEAs when compared to state-of-the-art alternatives.

*Also with KTH Royal Institute of Technology, Stockholm, Sweden.

This work was partially supported by Fundação Ciência e Tecnologia (FCT) under grant UIDB /50021/2020, and the Erasmus Mundus Joint Doctorate in Distributed Computing (EMJD-DC) funded by the Education, Audiovisual and Culture Executive Agency (EACEA) of the European Commission under FPA 2012-0030.

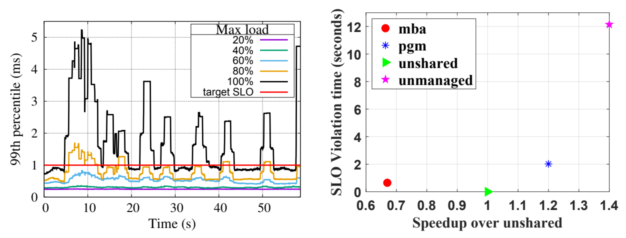


Figure 1: Impact of co-locating Memcached (LCA) with Ocean_cp (BEA) on a dual-socket machine (each on its own socket). Left: tail latency of Memcached for different loads. Right: Performance of Ocean_cp with different allocation approaches, where Memcached operates at 80% **of max load.**

2 Background

Thread packing and clock modulation [4] have been used for partitioning memory bandwidth in single-socket systems. Recently, Intel released MBA, which provides per-core control over memory bandwidth by injecting a delay value to each outgoing LLC request. The MBA level can be changed from 100% (no throttling) to 10% in steps of 10%. Although widely-used in single-socket scenarios, the above-mentioned mechanisms are not tailored to multi-socket scenarios.

To illustrate why, consider a dual-socket machine, where an LCA, Memcached [5], running in socket 0 is co-located with a BEA, Ocean_cp [10], running in socket 1. The LCA places all its pages locally. Assume its SLO is at most 1ms of the 99th percentile of client-side latency. The BEA is bandwidth-intensive, and, to optimize its throughput, interleaves its pages across both memory nodes.

Figure 1 shows how the sensitivity of the LCA to the available local memory bandwidth changes as a function of the LCA load. The observable peaks are caused by the fact that the BEA has phases with distinct memory intensities. As long as the LCA runs under low load, allowing the BEA to also use the LCA's memory node does not threaten the LCA's SLO. Still, as one increases the LCA's load, the memory access peaks of the BEA start to interfere with increasing intensity on the LCA's performance and, consequently, yield longer periods where the LCA no longer meets its SLO.

To fix the SLO violation in our example, we might employ MBA to throttle down the memory demand of the BEA. However, this comes with a performance penalty for the BEA, since MBA slows down both its remote accesses and its local accesses. Figure 1 (right) compares the performance of the BEA with the lowest value of MBA set to cure SLO violations, against an *unshared* alternative, where the BEA maps its pages locally; and an *unmanaged* alternative, where no partitioning mechanisms are used. As expected, *unmanaged* achieves higher throughput than MBA but fails to safeguard the SLO. On the other hand, *unshared* safeguards the performance of the LCA, but it uses each socket's resources sub-optimally.

Migrating pages of the noisy neighbour BEA(s) away from the memory node where the victim LCA runs can be an alternative to MBA (or other single-socket mechanisms). A recent proposal, BWAP [7], has shown that a *weighted* interleaving approach, where each memory node holds a specific fraction of the application's pages, is typically better than an *uniform* interleaving – provided the weights are adequately tuned, considering different factors related

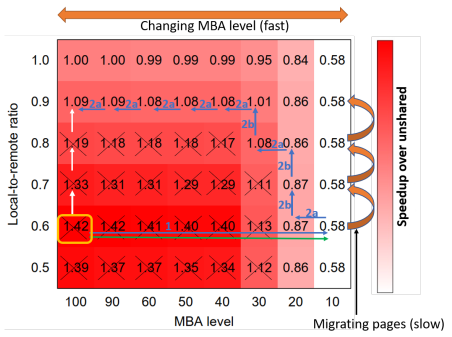


Figure 2: Performance of Ocean_cp colocated with Memcached operating at max load on a dual-socket machine. Each cell shows the speedup of Ocean_cp over the *unshared* approach for different configurations. The arrows denote the transitions of different mechanisms when fixing SLO violations: MBA (green), page migration (white), BALM (blue).

to the computer architecture and the memory access behavior of the workload [7]. Although BWAP is not designed for QoS-aware bandwidth allocation in multi-socket systems, it can be converted into a memory bandwidth allocation mechanism.

In contrast to MBA, page migration is able to adjust memory access demand on a *per-memory node granularity*. Hence, upon an SLO violation, in theory there is a weighted page interleaving of the BEA that reduces the access demand only on the saturated memory node, as needed to fix the SLO violation, while providing throughput gains to the BEAs. This is evident when, in Figure 1, we compare the throughput that page migration can attain when compared to MBA. However, page migration's latency is higher than that of MBA by many orders of magnitude. Such latency implies prohibitively long SLO violation windows. For this reason, page migration is unsuitable to QoS-aware memory bandwidth allocation, if used as a stand-alone mechanism.

3 BALM

We propose BALM, a novel approach to QoS-aware memory bandwidth allocation in multi-socket hosts. While the general approach of BALM is easily generalized to multi-socket systems of large sizes, this paper focuses on dual-socket systems. The key insight behind BALM is that, by using MBA and page migration as a 2-dimensional allocation mechanism, we unveil new opportunities to eliminate SLO violations. To illustrate this claim, Figure 2 depicts the example from Section 2 in a 2-dimensional perspective. Each dimension represents the single parameter that tunes each allocation mechanism: with page migration, the ratio between the local and remote interleaving weights (yy axis); with MBA, the MBA level assigned to the BEA (xx axis). The cell values are the BEA's speedup over the *unshared* approach (the top-left configuration).

Recalling the example, the BEA and LCA execute on opposite sockets. Initially, the LCA is running under a negligible load. Thus, BALM chooses the configuration that maximizes BEA's throughput – it places pages in a local-to-remote ratio of 0.6:0.4, with no MBA throttling. Later on, the LCA enters a high load phase. Consequently, some configurations become invalid. These are marked with 'X' in Figure 2. Since the initial configuration is invalid, an SLO violation occurs. Ideally, it should be fixed by quickly transitioning the BEA from the initial invalid configuration to some configuration that, among the valid alternatives, maximizes the BEA's throughput.

The matrix in Figure 2 sheds light on the virtues and limitations of MBA and page migration when used alone to handle SLO violations. Using MBA alone restricts the space of available valid configurations to those located on the *same row* as the initial configuration; hence, it is fast but will not reach the optimal valid configuration in this example. In turn, using page migration alone can only exploit the configurations in the first column; thus, it can reach the optimal valid configuration, albeit by slow steps.

In contrast, BALM exploits both, MBA and page migration, to fix the SLO violation by unveiling the entire 2-dimensional configuration space and combining the virtues of each of the mechanisms. To illustrate this, Figure 2 depicts the path that BALM follows to solve the SLO violation in our example. This path heals the SLO violation as quickly as using MBA alone, while eventually reaching the valid configuration that maximizes the BEA's throughput (as page migration does). In a nutshell, BALM fixes the SLO violation in two steps: first, it sets MBA to the most restrictive level, trying to fix the violation as fast as possible; next, it incrementally migrates pages to make the BEA converge to the best local-to-remote ratio. As the bandwidth demand on the saturated memory node is alleviated after each page migration, BALM gradually releases the MBA throttling when it observes that doing that still leaves the system in a valid configuration. We expect BALM to be as quick as MBA in fixing SLO violations, while converging to the same optimal valid configuration that the page migration will reach.

Uncertain and dynamic workloads. The 2-dimensional configuration matrix in Figure 2 is not known a priori, and, furthermore, its values change dynamically as workload changes. Hence, to find a valid configuration, BALM resorts to an online hill-climbing search, which gradually finds its way to the optimal valid configuration as illustrated in Figure 2. To accomplish this, BALM relies on a monitoring component, which continuously samples each LCA's SLO metrics and provides frequent system-wide diagnostics of the QoS health of the LCAs; namely, whether any SLO violations are already happening or prone to happen, and at which sockets they occur. Such diagnostics feed into the BALM controller, which uses it to decide when to trigger a new reconfiguration cycle and infer the outcome of each adaptation action (enforced on a BEA).

Multiple co-located applications. BALM also supports QoS-aware memory bandwidth allocation in general scenarios where multiple LCAs and BEAs may be co-located at each socket of the machine. In such scenarios, BALM monitors the SLO of the ensemble of LCAs at its detection and adaptation stages. if at least one LCA's SLO is violated, BALM uses MBA and page migration to reach a valid configuration in which *every* LCA in the system has its SLO met and throughput of BEAs is optimized.

Cross-socket and intra-socket interference. In a general scenario, multiple LCAs may also be running on the same socket as the BEA. In such scenario, besides SLO violations due to cross-socket interference, intra-socket interference may also trigger SLO violations. BALM deals with both situations by choosing the right page migration direction when healing. Cross-socket interference is alleviated by remote-to-local migration (i.e., stepping up in Figure 2), whereas intra-socket interference triggers local-to-remote migration (i.e., stepping down). Furthermore, since the LCAs may be running at different sockets, using local-to-remote page migration to fix an intra-socket SLO violation can cause cross-socket SLO violations and visa versa. To prevent this side effect, BALM collectively monitors the SLO of every LCA after each page migration step and rolls back to the previous valid configuration if a new SLO violation is observed. A consequence of this measure is that, in such complex scenarios, an optimal valid configuration that totally disables MBA throttling of the BEA (i.e., MBA 100) may no longer exist.

Multiple noisy neighbours. Finally, one needs to consider that more than one BEA may simultaneously enter a bandwidth-intensive phase, and cause SLO violations. In such scenario, to fix the violation, BALM throttles memory bandwidth consumption of multiple BEAs by considering one BEA at a time, starting by those that, in the near past, have consumed the most memory bandwidth of the memory node where SLO violations have been detected. To acquire memory bandwidth usage on a per-socket and per-application basis, BALM employs the Memory Bandwidth Monitoring feature of the Intel RDT technology. For each BEA, the 2-dimensional online search described above is carried out. When the search completes, BALM has reached either a valid or an invalid configuration. In the former case, the SLO violations have been healed, and no more BEAs need to be adapted. In the latter case, the memory bandwidth diet imposed on the current BEA was not enough to fix the SLO violations, thus BALM moves to the next BEA in the queue.

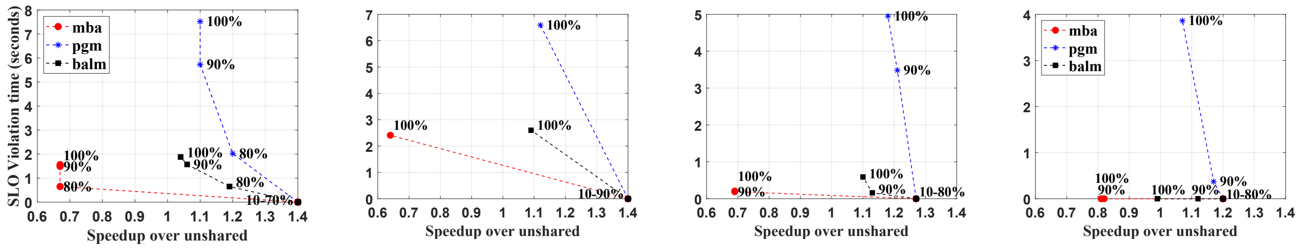
4 Evaluation

Our evaluation addresses two key questions: *1. What performance advantage does BALM bring to memory-intensive BEAs on dual-socket NUMA systems? 2. How effective is BALM in fixing SLO violations?*

Methodology. We evaluate BALM on a dual-socket machine with two Intel Xeon Silver 4114 CPUs, with 10 cores per CPU, 128GB DRAM (64GB at each NUMA node), running Linux 4.15. It supports MBA, with 8 available levels. We compare BALM to *MBA* (*mba*) and *page migration (pgm)*, each used stand-alone; as well as the *unshared* and *unmanaged* approaches (see Section 2). For space limitations, we present the evaluation of BALM for two consolidated applications, one LCA and one BEA, each on its own socket in a dual-socket machine. Our complete evaluation has shown, the main conclusions drawn for the scenario with two applications also hold in the scenarios with multiple LCAs and BEAs.

As a representative LCA, we use Memcached [5] in our experiments. Our default Memcached deployment is 10 million items, each with a 30B key and a 200B value; the SLO target is set to 1ms for 99^{th} percentile latency, which is in line with the experimental deployment methodology in previous works (e.g., [3, 8]. We assume that the SLO of Memcached is defined by tail latency (99^{th} percentile) of request-to-response latency observed by its clients.

To monitor the SLO of the LCAs, BALM's monitoring component keeps a sliding window of all the recent requests that have occurred in the last n seconds and polls the SLO metric, such as tail latency at m milliseconds interval (which is fine-grained). We configure BALM with n and m to 3 seconds and 20 ms, respectively. This choice of parameters allowed the SLO metric to be calculated over large-enough samples, which reduce measurement noise; while allowing BALM to react quickly after a sample yields an SLO violation.



(a) Ocean_cp **(b) MG.C** **(c) SP.C** **(d) UA.C**

Figure 3: Performance of BEAs and SLO violation time of high-load LCA. The plots show the speedup of BEAs (x-axis) and SLO violation time of LCA (y-axis) that can be achieved by different mechanisms when LCA is running at the fraction of its *max load* indicated by the % values.

For the BEAs, we used memory-intensive multi-threaded benchmarks from several benchmark suites, i.e., NAS [1], PARSEC [2] and SPLASH [10]. These benchmarks represent a wide diversity of application domains which are typically throughput-oriented, which are also used as such in related works, e.g., [6, 8]. We pin the threads of each benchmark on the cores allocated to it. All BEAs are characterized by multiple phases with different memory intensities. The spikes in Figure 1 illustrate this for Ocean_cp.

Results. Figure 3 presents the results for each metric (BEA performance and LCA SLO violation time) for increasing LCA loads. As expected, when the LCA runs at a modest load levels, no SLO violation occurs and the BEA achieves its maximum performance since it runs with no bandwidth allocation restrictions – regardless of which mechanism is used. This corresponds to the bottom-right point at each plot in Figure 3. However, as we increase the LCA load beyond a critical level (which, depending on the bandwidth intensity of each BEA, ranges between 70% and 90% of *max load*), QoS violations arise at increasing frequency and intensity. These trigger the different mechanisms to allocate less memory bandwidth to the BEA, thus reducing its throughput.

Figure 3 also makes it evident that, in such high load situations, each mechanism handles the SLO violations with very distinct effectiveness. As one increases the LCA load beyond a critical level, the *mba* curve quickly expands towards the left-hand extreme of the plot (i.e., sacrifices the throughput of the offending BEA), while *pgm* quickly grows upwards (i.e., taking an increasingly longer time to heal SLO violations). These trends confirm the preliminary observations from Section 2. In contrast, BALM's curves in the same plots manage to stay closer to the initial optimal point (the low-load point). Hence, BALM handles increasing LCA loads at relatively lower costs on *both* axis. Most importantly, if we chose a given LCA load and observe how each mechanism performs at both criteria, then it becomes clear that BALM's performance on each axis is typically close to the alternative mechanism that is best-performing in that axis. For instance, with Ocean_cp, the SLO violation time of BALM at 80%/90%/100% (resp.) of the LCA's *max load* are just 0%/17%/5% (resp.) above *mba*'s marginal values. If, instead, we consider the throughput of Ocean_cp, we conclude that BALM is just 1%/4%/6% (resp.) below the performance that *pgm* achieves on that dimension. This translates to BALM outperforming *mba* and *unshared* by up to 1.78× and 1.4×, resp.. To understand why BALM does not always achieve the same BEA throughput as *pgm*, recall that BALM activates MBA until the page migration process completes, which temporarily hinders the BEA. The above results confirm that BALM attains the virtues of each extreme (*mba* and *pgm*), making BALM a well-balanced compromise between both conflicting criteria.

5 Conclusion

We propose BALM, a QoS-aware memory bandwidth allocation technique for multi-socket systems. Our evaluation shows that, compared to state-of-the-art alternatives, BALM ensures marginal SLO violation windows for latency-sensitive applications while delivering up to 87% throughput gains to best-effort applications.

References

[1] D. H. Bailey, E. Barszcz, J. T. Barton, D. S. Browning, R. L. Carter, L. Dagum, R. A. Fatoohi, P. O. Frederickson, T. A. Lasinski, R. S. Schreiber, H. D. Simon, V. Venkatakrishnan, and S. K. Weeratunga. 1991. The NAS Parallel Benchmarks – Summary and Preliminary Results. In *Proceedings of the 1991 ACM/IEEE Conference on Supercomputing* (Albuquerque, New Mexico, USA) *(Supercomputing '91)*. ACM, New York, NY, USA, 158–165. https://doi.org/10.1145/125826.125925

[2] Christian Bienia. 2011. *Benchmarking Modern Multiprocessors.* Ph.D. Dissertation. Princeton University.

[3] Shuang Chen, Christina Delimitrou, and José F. Martínez. 2019. 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* (Providence, RI, USA) *(ASPLOS '19)*. ACM, New York, NY, USA, 107–120. https://doi.org/10.1145/3297858.3304005

[4] R. Cochran, C. Hankendi, A. K. Coskun, and S. Reda. 2011. Pack & Cap: Adaptive DVFS and thread packing under power caps. In *2011 44th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 175–185.

[5] Brad Fitzpatrick. 2004. Distributed Caching with Memcached. *Linux J.* 2004, 124 (Aug. 2004), 5.

[6] Joshua Fried, Zhenyuan Ruan, Amy Ousterhout, and Adam Belay. 2020. Caladan: Mitigating Interference at Microsecond Timescales. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. 281–297.

[7] D. Gureya, J. Neto, R. Karimi, J. Barreto, P. Bhatotia, V. Quema, R. Rodrigues, P. Romano, and V. Vlassov. 2020. Bandwidth-Aware Page Placement in NUMA. In *2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. 546–556. https://doi.org/10.1109/IPDPS47924.2020.00063

[8] T. Patel and D. Tiwari. 2020. CLITE: Efficient and QoS-Aware Co-Location of Multiple Latency-Critical Jobs for Warehouse Scale Computers. In *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 193–206.

[9] L. Tang, J. Mars, X. Zhang, R. Hagmann, R. Hundt, and E. Tune. 2013. Optimizing Google's warehouse scale computers: The NUMA experience. In *2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA)*. 188–197.

[10] Steven Cameron Woo, Moriyoshi Ohara, Evan Torrie, Jaswinder Pal Singh, and Anoop Gupta. 1995. The SPLASH-2 Programs: Characterization and Methodological Considerations. In *Proceedings of the 22Nd Annual International Symposium on Computer Architecture* (S. Margherita Ligure, Italy) *(ISCA '95)*. ACM, New York, NY, USA, 24–36. https://doi.org/10.1145/223982.223990