POSTER: Improving Datacenter Efficiency through Partitioning-Aware Scheduling

Harshad Kasture*‡ Xu Ji†§ Nosayba El-Sayed*† Nathan Beckmann¶ Xiaosong Ma† Daniel Sanchez*
*MIT CSAIL †QCRI, HBKU ‡Oracle Labs §Tsinghua University ¶CMU SCS
{harshad, nosayba, sanchez}@csail.mit.edu ji-x13@mails.tsinghua.edu.cn beckmann@cs.cmu.edu xma@hbku.edu.qa

I. Introduction

Datacenters and shared clusters often colocate multiple applications or virtual machines per server to improve utilization. However, colocated applications interfere in shared resources, such as the last-level cache (LLC) and DRAM bandwidth, leading to performance degradation [3, 4, 7, 8]. Prior work has proposed two disjoint approaches to deal with the problem of interference: resource partitioning within a node, and job scheduling across nodes.

On the one hand, cache partitioning techniques divide cache capacity among applications to maximize throughput [2, 11] and fairness [6, 9], or to guarantee quality of service [7]. However, these techniques are limited to partitioning cache capacity among a fixed set of colocated applications, and have no influence over what applications get scheduled on each node. This limits their efficacy when colocated applications have competing resource requirements.

On the other hand, cluster managers and schedulers attempt to improve performance by colocating applications that do not interfere in shared resources. These schedulers use either offline profiling [3, 8] to identify non-interfering applications, or online monitoring [10, 16] to throttle or migrate offending applications. While these schedulers can identify safe colocations and do improve utilization, they do not take advantage of memory-system partitioning techniques. Therefore, they must be conservative in colocating applications, since they cannot control how applications share resources.

We show that cache partitioning and cluster scheduling are complementary techniques, and performing them in a coordinated manner significantly boosts performance. We present Shepherd, a joint scheduler and resource partitioner that seeks to maximize cluster-wide throughput. On each machine, Shepherd uses detailed application profiling data to partition the shared LLC and to estimate the impact of DRAM bandwidth contention among colocated applications. Across machines, Shepherd uses this information to colocate applications with complementary resource requirements, improving resource utilization and cluster throughput. As a result, Shepherd improves cluster throughput over an unpartitioned system by 38% on average.

II. Shepherd Design

Shepherd seeks to maximize cluster-wide performance by scheduling applications across cluster nodes in a partitioning-aware fashion. In this work, we use throughput, specifically weighted speedup, as the performance metric, although additional metrics (e.g., fairness) can easily be incorporated. Shepherd takes as inputs a set of applications and a set of machines, and produces a schedule that maps each application to a machine and partitions the LLC on each machine.

Profiling: Shepherd relies on detailed application profiling data, gathered offline, that capture each application's sensitivity to cache capacity and memory bandwidth, as well as its memory bandwidth demands. First, Shepherd profiles an application's performance, measured in instructions per cycle (IPC), for several amounts of available cache capacity and memory bandwidth. This produces an *IPC surface* over the 2D space of cache capacity and bandwidth. Second, Shepherd profiles an application's *bandwidth curve*, which captures its bandwidth demands at various cache capacities.

Node performance optimization: Shepherd partitions the cache among applications in an iterative fashion. Each iteration first estimates the total bandwidth consumed by all applications, then uses this bandwidth to project per-application IPC surfaces into *IPC curves* that report performance as a function of cache partition size. Shepherd then uses standard dynamic programming to find the partition sizes that maximize weighted speedup [9, 15]. Finally, Shepherd uses partition sizes and per-application bandwidth curves to update the total bandwidth estimate. The process is repeated until partition sizes converge. Beyond finding high-performance configurations, this method accurately estimates application performance for a given mix.

Cluster scheduling: Shepherd uses per-node performance estimates to drive application placement, co-optimizing cache partitioning and cluster scheduling. Finding the optimal schedule is an NP-hard problem, so exact solutions are impractical. Instead, Shepherd uses simulated annealing [1], a well-known randomized search algorithm, to quickly converge to a high-performance schedule. Starting from a random schedule, Shepherd iteratively searches over the space of possible schedules. At each iteration, a new schedule is generated by randomly swapping two applications. Shepherd estimates the weighted speedup for this new schedule, and compares it against the previous speedup. If the new speedup is higher, Shepherd "accepts" the new schedule and repeats the process. If the new speedup is lower, however, the new schedule may still be accepted with a probability $p < 1$. By occasionally accepting a *worse* schedule, Shepherd's randomized search avoids getting stuck in local minima. The probability p is lowered over time, and the algorithm eventually converges to a high-performance schedule.

In our experiments, evaluating a swap takes a few 10s of μs, so scheduling decisions for a few hundred applications can be made in under a second.

III. Evaluation

Methodology: We evaluate Shepherd in simulation, using zsim [13] to simulate each node. Each node has eight cores and a three-level memory hierarchy, with parameters that closely match a Broadwell Xeon D-1540 processor. Our simulated system has a 12 MB, 32-way set-associative last-level cache, and uses Vantage [12] to partition the LLC.

Workloads: We use 20 memory-intensive server, scientific, and analytics workloads. Nine applications come from the SPEC CPU2006 suite, and seven are graph-analytics applications from PBBS [14]. We select the benchmarks that have at least 10 L2 misses per kiloinstruction (MPKI) and show at least a 10% change in L3 MPKI across the range of cache partition sizes. The remaining four, from the TailBench [5] suite, are request-driven workloads typical in datacenter servers.

Schemes: We compare four different schemes:

- The baseline scheme does not use cache partitioning and places applications randomly across nodes.
- Shepherd performs cache partitioning and partitioning-aware placement using simulated annealing.
- *PartOnly* performs cache partitioning like Shepherd does, but places applications randomly like the baseline.
- *PlaceOnly* does not perform cache partitioning, but does partitioning-aware placement like Shepherd.

PartOnly and PlaceOnly are intermediate design points that allow us to analyze where Shepherd's benefits come from.

Fig. 1 shows the performance of different schemes on an 8-node cluster. We evaluate eight mixes of 64 randomly-chosen applications each, and report the weighted speedup over the baseline for all the schemes we study. Each group of bars reports results for a different mix; the rightmost bar shows the weighted speedup across all eight mixes.

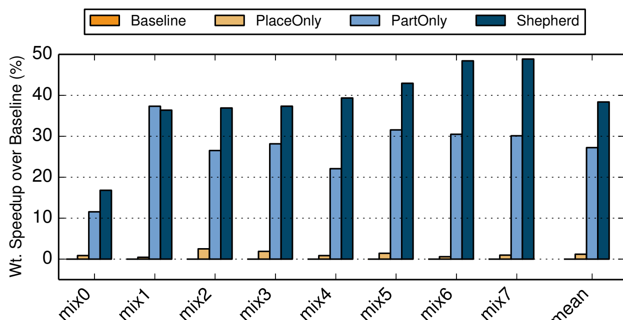

Figure 1. Weighted speedup over the baseline for Shepherd, PartOnly, and PlaceOnly, on eight random 64-app mixes. Each mix is scheduled on an 8-node cluster.

We observe that Shepherd improves performance considerably, by 38% on average and by up to 49%. This happens because Shepherd ensures that applications that derive the highest benefit from the LLC do not compete with each other, maximizing cache utility and cluster throughput.

PartOnly improves performance significantly, by 27% on average and by up to 37%. But these gains are substantially lower than Shepherd's. This happens because PartOnly is constrained by the set of applications colocated on a given machine. Thus, while some mixes experience large gains, others have modest improvements, as little as 11%. Beyond outperforming PartOnly, Shepherd also yields less-variable gains: the lowest speedup is 17%, and all other mixes experience gains of at least 36%.

Finally, PlaceOnly has a negligible effect on performance: all mixes are within 2% of the baseline, and the overall improvement is less than 1%. This is not surprising: with an unpartitioned cache, placement alone offers limited control over LLC contention among applications, and although individual application performance varies with different placements, the overall effect is a wash. This result shows that Shepherd's benefits are not simply the combination of two independent effects (partitioning and placement). Instead, they come from placement boosting the effectiveness of partitioning.

References

[1] D. Bertsimas and O. Nohadani, "Robust optimization with simulated annealing," *J. of Global Optimization*, 2010.

[2] H. Cook, M. Moreto, S. Bird, K. Dao, D. A. Patterson, and K. Asanovic, "A hardware evaluation of cache partitioning to improve utilization and energy-efficiency while preserving responsiveness," in *ISCA-40*, 2013.

[3] C. Delimitrou and C. Kozyrakis, "Paragon: QoS-aware scheduling for heterogeneous datacenters," in *ASPLOS-XVIII*, 2013.

[4] H. Kasture and D. Sanchez, "Ubik: Efficient cache sharing with strict QoS for latency-critical workloads," in *ASPLOS-XIX*, 2014.

[5] H. Kasture and D. Sanchez, "TailBench: A benchmark suite and evaluation methodology for latency-critical applications," in *IISWC*, 2016.

[6] J. Lin, Q. Lu, X. Ding, Z. Zhang, X. Zhang, and P. Sadayappan, "Gaining insights into multicore cache partitioning: Bridging the gap between simulation and real systems," in *HPCA-14*, 2008.

[7] D. Lo, L. Cheng, R. Govindaraju, P. Ranganathan, and C. Kozyrakis, "Heracles: Improving resource efficiency at scale," in *ISCA-42*, 2015.

[8] J. Mars, L. Tang, R. Hundt, K. Skadron, and M. L. Soffa, "Bubble-Up: Increasing utilization in modern warehouse scale computers via sensible co-locations," in *MICRO-44*, 2011.

[9] M. Moreto, F. J. Cazorla, A. Ramirez, R. Sakellariou, and M. Valero, "FlexDCP: A QoS framework for CMP architectures," *ACM SIGOPS Operating Systems Review*, vol. 43, no. 2, 2009.

[10] D. Novaković, N. Vasić, S. Novaković, D. Kostić, and R. Bianchini, "DeepDive: Transparently identifying and managing performance interference in virtualized environments," in *ATC*, 2013.

[11] M. Qureshi and Y. Patt, "Utility-based cache partitioning," in *MICRO-39*, 2006.

[12] D. Sanchez and C. Kozyrakis, "Vantage: Scalable and Efficient Fine-Grain Cache Partitioning," in *ISCA-38*, 2011.

[13] D. Sanchez and C. Kozyrakis, "ZSim: Fast and accurate microarchitectural simulation of thousand-core systems," in *ISCA-40*, 2013.

[14] J. Shun, G. E. Blelloch, J. T. Fineman, P. B. Gibbons, A. Kyrola, H. V. Simhadri, and K. Tangwongsan, "Brief announcement: The problem based benchmark suite," in *SPAA*, 2012.

[15] H. S. Stone, J. Turek, and J. L. Wolf, "Optimal partitioning of cache memory," *IEEE Trans. Comput.*, vol. 41, no. 9, 1992.

[16] X. Zhang, E. Tune, R. Hagmann, R. Jnagal, V. Gokhale, and J. Wilkes, "CPI2: CPU performance isolation for shared compute clusters," in *EuroSys*, 2013.

This work was supported in part by NSF grant CCF-1318384 and by a grant from the Qatar Computing Research Institute.