JSL, 9(6), June 2024: 414 - 424

Online ISSN: 2456-1495
CODEN (USA): JSLOBJ

LEARNING-BASED LAST-LEVEL CACHE OPTIMIZATION FOR MULTICORE SYSTEMS: ADAPTIVE ALLOCATION, INTERFERENCE CONTROL, AND THROUGHPUT ENHANCEMENT

Sajjade Zeba Shazesh

Research Scholar, Jayoti Vidyapeeth Women's University, Jaipur, Rajasthan

Dr. Sushma Agarwal

Research Supervisor, Jayoti Vidyapeeth Women's University, Jaipur, Rajasthan

ABSTRACT

Inefficient LLC allocation can increase cache misses, memory traffic, execution time, and performance interference among co-running workloads. Conventional static and heuristic partitioning methods often struggle to adapt to workload phase changes and heterogeneous cache sensitivities. This paper examines learning-based techniques for LLC optimization, with particular emphasis on adaptive cache allocation, interference control, workload classification, and system-throughput enhancement. Machine learning can exploit hardware performance-monitoring information to identify cache-sensitive applications, predict resource requirements, and select cache allocations dynamically. The study discusses learning-based performance prediction, cache partitioning, clustering, fairness-aware management, and coordinated cache and memory-bandwidth optimization. It further considers implementation challenges including model-training requirements, inference overhead, phase detection, generalization, hardware support, and quality-of-service constraints.

Keywords: Last-Level Cache, Multicore Systems, Machine Learning, Cache Partitioning, Adaptive Allocation, Cache Interference, Throughput Optimization, Workload Classification, Quality of Service, Performance Optimization.

Online ISSN: 2456-1495
CODEN (USA): JSLOBJ

I. INTRODUCTION

The increasing number of cores in contemporary processors has substantially improved computational parallelism, but it has also intensified contention for shared hardware resources. Among these resources, the last-level cache plays a particularly important role because it acts as a high-capacity interface between multiple processing cores and main memory. A well-managed LLC can reduce memory-access latency and bandwidth demand, whereas poorly managed sharing can produce cache pollution, increased miss rates, memory stalls, and substantial slowdowns. Cache partitioning has therefore become an important technique for controlling shared-cache interference and providing performance isolation in multicore systems. A major survey of cache-partitioning techniques identifies intelligent partitioning as increasingly important because naïve allocation can cause performance loss, unfairness, and inadequate quality-of-service guarantees.

The challenge becomes more complex when multicore systems execute heterogeneous applications simultaneously. Some workloads have small working sets and gain little from additional cache capacity, while others are highly cache-sensitive and benefit significantly from larger partitions. Streaming applications may pollute shared cache without deriving substantial benefit from retaining large portions of their data. Furthermore, application behaviour is rarely constant throughout execution. Programs can move between compute-intensive, memory-intensive, synchronization-heavy, and I/O-related phases, causing their cache requirements to change continuously. Recent research on dynamic cache apportioning emphasizes that such phase behaviour can occur at fine temporal granularities, making slow or static allocation policies ineffective.

Traditional cache-partitioning approaches use predefined heuristics, utility curves, profiling information, or fixed resource shares. Although these techniques can perform effectively under known conditions, they may require extensive configuration searches or prior knowledge of applications. Online approaches are particularly challenging because the resource manager must discover suitable allocations while the system is running. Recent research on coordinated LLC and memory-bandwidth management notes that repeated configuration searches can cause substantial management latency and reduce the responsiveness of dynamic resource allocation.

Machine learning offers an alternative by enabling resource managers to infer relationships

E-mail: ***editor.jslsci888@gmail.com***

Online ISSN: 2456-1495
CODEN (USA): JSLOBJ

between workload characteristics, cache allocation, and performance. Runtime information such as LLC misses, cache references, instructions per cycle, memory bandwidth, memory-level parallelism, and other hardware performance-counter measurements can be used to classify applications and predict their response to different cache allocations. Instead of testing every possible partition, a learning model may estimate which allocations are likely to provide the greatest benefit.

A representative example is Com-CAS, which combines compiler-level information with machine-learning mechanisms to predict dynamic cache requirements and proactively apportion LLC capacity among co-running applications using Intel Cache Allocation Technology. Reported experiments found improved average weighted throughput compared with an unpartitioned system and an earlier partitioning approach while maintaining application-level service constraints. Similarly, LFOC+ demonstrates that runtime performance-monitoring information can classify applications according to cache sensitivity and contentiousness and then support dynamic cache clustering. Its evaluation on commodity hardware reported improved fairness compared with earlier cache-clustering approaches.

II. LEARNING-BASED ADAPTIVE LLC ALLOCATION AND WORKLOAD CHARACTERIZATION

Learning-based LLC management begins with accurate characterization of the applications sharing the processor. Conventional allocation policies often assume that applications can be classified according to manually determined categories, but heterogeneous modern workloads make such assumptions increasingly unreliable. Machine learning can infer workload characteristics directly from runtime measurements and can adapt decisions as behaviour changes.

Hardware performance-monitoring counters provide a practical source of information. Relevant features include LLC miss rate, cache references, instructions retired, cycles, instructions per cycle, branch activity, memory-bandwidth consumption, memory stalls, and indicators of memory-level parallelism. By collecting these features periodically, the resource manager can construct a runtime representation of the application's current phase. A trained model can then classify the application or predict how its performance is expected to change when its cache allocation changes.

A basic learning-based architecture can be divided into four stages: monitoring, feature

E-mail: ***editor.jslsci888@gmail.com***

Online ISSN: 2456-1495
CODEN (USA): JSLOBJ

extraction, prediction, and allocation. During monitoring, the processor gathers performance-counter values. Feature extraction converts raw measurements into normalized characteristics. The prediction stage estimates cache sensitivity or expected performance under candidate allocations. Finally, the allocation controller selects a partition based on a specified system objective.

The classification of workloads by cache sensitivity and contentiousness is particularly relevant to cache clustering. LFOC+ uses runtime performance-monitoring information to identify such characteristics and separates sensitive workloads from aggressive programs to improve fairness. Its evaluation on an Intel Skylake system demonstrated measurable fairness improvements while maintaining acceptable throughput. This demonstrates that learning-like workload classification can be implemented without placing a large computational burden on the processor.

Another important concept is phase detection. Applications often exhibit recurring phases in which memory behaviour remains relatively stable for a period and then changes. A resource manager that detects phase transitions can trigger cache reallocation only when necessary. This reduces management overhead compared with continuously searching for optimal allocations. Com-CAS takes a predictive approach by leveraging compiler-generated information together with machine-learning mechanisms to anticipate rapidly changing cache requirements.

The quality of a learning-based manager depends heavily on feature selection. Too few features may fail to distinguish workloads with different cache behaviour, while excessive features increase model complexity and inference overhead. Normalization is also important because performance counters can have widely different numerical ranges. Lightweight feature sets that can be collected at low cost are generally more appropriate for online deployment.

Model selection should likewise reflect hardware constraints. Large deep-learning models may provide high predictive capacity but introduce unnecessary inference latency and energy consumption. Decision trees, linear regression, small neural networks, random forests, and compact ensemble models may provide better practical trade-offs. The objective is not necessarily maximum prediction accuracy but sufficiently accurate resource decisions at very low overhead.

Online ISSN: 2456-1495
CODEN (USA): JSLOBJ

Training generalization is another major issue. A model trained using one benchmark collection may perform poorly when encountering applications with different memory access patterns. System architecture also matters: models trained on one cache size, associativity, processor generation, or memory subsystem may not transfer directly to another platform. Transfer learning, online adaptation, and hardware-independent workload representations could improve portability.

Learning-based allocation can also incorporate application priorities. In cloud and multi-tenant systems, applications may possess different service-level requirements. The manager can first allocate the minimum cache capacity necessary to satisfy each quality-of-service constraint and then distribute remaining capacity according to throughput or fairness. This approach prevents an aggressively optimized throughput policy from causing unacceptable slowdown to latency-sensitive workloads.

Energy considerations further expand the role of adaptive allocation. Increasing LLC allocation can reduce memory traffic, which may lower off-chip memory energy, but maintaining larger active cache regions may also increase cache-related energy. The optimal allocation is therefore workload dependent. Coordinated resource-management research demonstrates that cache allocation can interact with processor configuration and energy consumption, suggesting that cache should be considered part of a larger system optimization problem rather than an isolated resource.

III. INTERFERENCE CONTROL, THROUGHPUT ENHANCEMENT, AND SYSTEM-LEVEL OPTIMIZATION

Shared-cache interference occurs when multiple applications compete for the same cache capacity and evict one another's useful data. Such interference can significantly affect performance, particularly when a cache-sensitive application shares a partition with a streaming or cache-aggressive workload. The objective of interference control is not necessarily to eliminate all sharing but to organize workloads so that cache capacity is used efficiently while harmful interactions are minimized.

One effective strategy is cache partitioning. Each application receives an allocation that limits its use of the shared cache. Hardware technologies such as Intel Cache Allocation Technology provide mechanisms through which system software can assign portions of the LLC to different classes of service. Learning-based resource managers can use these

E-mail: ***editor.jslsci888@gmail.com***

hardware facilities to implement dynamically generated allocations rather than relying on fixed configurations. Research on hardware-supported cache partitioning has also shown its usefulness for performance isolation and predictable execution in real-time systems.

Cache clustering represents a less rigid alternative. Rather than creating a separate partition for each workload, compatible applications can be grouped and allowed to share a cache region. This reduces fragmentation and can improve resource utilization. LFOC and LFOC+ demonstrate this principle by identifying cache-sensitive and aggressive applications at runtime and assigning them to different clusters. LFOC+ was evaluated in a real Linux environment and reported improvements in fairness over previous fairness-oriented policies.

Machine learning can improve clustering by identifying hidden relationships among applications. Two applications with similar access patterns may share a cache efficiently, while applications with highly conflicting patterns should be separated. The model can therefore estimate compatibility in addition to individual cache sensitivity. This converts cache allocation into a workload-placement problem.

Throughput enhancement is typically measured through metrics such as aggregate instructions per cycle, weighted speedup, or system throughput. A well-designed allocation policy should increase useful processor work while reducing memory stalls. Recent PaLLOC research addresses a closely related problem by coordinating LLC and memory-bandwidth allocation and reports substantial improvements in system IPC throughput compared with a state-of-the-art online partitioning method. Its evaluation across hundreds of workload mixes demonstrates the significance of reducing search overhead while adapting to different resource requirements.

Fairness is important because maximum throughput and equitable performance are not always aligned. An application that receives very little cache may experience severe slowdown even if total system throughput increases. Metrics such as maximum slowdown and weighted speedup can therefore be incorporated into the learning objective. The manager may select a slightly lower-throughput configuration when it substantially improves the worst-performing application.

This becomes especially important in cloud computing and multi-tenant environments, where users expect predictable service. A noisy neighbour that consumes excessive shared-cache resources can create significant performance variability for other tenants. Dynamic

Online ISSN: 2456-1495
CODEN (USA): JSLOBJ

partitioning can isolate such workloads. Learning-based classification can identify aggressor applications and automatically reduce their ability to disrupt sensitive workloads.

Latency is another critical metric. The cache manager itself should respond quickly enough that adaptation remains useful. PaLLOC research illustrates the importance of low management latency when dynamically allocating shared resources and reports approximately 300 ms management latency while maintaining throughput improvements over prior online methods. Although the exact target latency depends on workload dynamics, the broader lesson is that resource optimization cannot be separated from the cost of making decisions.

A practical intelligent controller should therefore use different decision frequencies for different tasks. Performance counters can be monitored frequently, phase changes can be detected at moderate intervals, and complete allocation optimization can occur only when significant changes are detected. This hierarchical structure reduces overhead while retaining responsiveness.

Model uncertainty should also be addressed. A prediction may be inaccurate because an application enters an unfamiliar phase or because multiple workloads interact in an unexpected manner. A robust manager can estimate prediction confidence and fall back to a conservative heuristic when uncertainty is high. Such hybrid strategies can provide more predictable behaviour than relying entirely on machine learning.

Security considerations are increasingly relevant because cache-sharing patterns can expose information about co-running applications. Static partitioning can improve isolation, but dynamic policies must ensure that resource adaptation does not violate security boundaries. In security-sensitive environments, partitions may need to be reserved or transitions carefully controlled. Learning mechanisms must also avoid using features that themselves expose sensitive application information beyond what is permitted.

The hardware implementation of an intelligent LLC manager involves several components: performance-monitoring infrastructure, feature collection, a prediction engine, an optimization controller, and an interface to cache allocation hardware. The learning model does not need to operate in the processor's critical execution path. It can run periodically in a dedicated controller, operating-system service, or management core. This separation limits the impact on normal instruction execution.

E-mail: ***editor.jslsci888@gmail.com***

Online ISSN: 2456-1495
CODEN (USA): JSLOBJ

Another important consideration is training overhead. Offline training can use benchmark suites and simulation environments to generate performance data, while online refinement can adapt the model to deployment conditions. A two-stage approach can provide a good balance: offline training provides initial knowledge, while lightweight online learning corrects model biases. The resulting system can adapt without repeatedly performing expensive searches over every cache configuration.

IV. CONCLUSION

Learning-based last-level cache optimization represents a promising approach to addressing the increasing complexity of shared-resource management in multicore processors. As core counts increase and workloads become more heterogeneous, the assumption that a single static cache partition can serve all applications efficiently becomes increasingly unrealistic. Applications differ substantially in cache sensitivity, memory intensity, working-set size, and interference behaviour, and many applications change these characteristics during execution. An intelligent LLC manager must therefore observe workload behaviour continuously and adapt resource allocation according to changing requirements.

Machine learning provides a useful mechanism for achieving this adaptability. Runtime performance-monitoring information can be transformed into workload features that describe cache sensitivity, memory behaviour, contention, and execution phase. Prediction and classification models can then estimate performance under alternative cache allocations and identify suitable partitioning or clustering strategies. Research such as Com-CAS demonstrates the potential of combining compiler information and machine learning with hardware-supported cache allocation for proactive response to changing application requirements. LFOC+ similarly illustrates that runtime performance monitoring and application classification can support lightweight cache clustering with measurable fairness benefits on commodity processors.

The paper also shows that adaptive cache management should not focus exclusively on cache hit rate. The broader system objectives include throughput, fairness, latency, energy efficiency, memory bandwidth, quality of service, and management overhead. Increasing cache capacity for one workload may reduce memory traffic and improve system performance, but it may simultaneously deprive another workload of useful capacity. Conversely, aggressive partitioning can provide isolation but cause fragmentation and lower

E-mail: ***editor.jslsci888@gmail.com***

Online ISSN: 2456-1495
CODEN (USA): JSLOBJ

overall utilization. The most effective allocation is consequently dependent on the characteristics and interactions of the current workload set.

A particularly important insight is that LLC management should be coordinated with other shared resources. Memory bandwidth, processor configuration, and cache capacity influence each other, and independent optimization can produce suboptimal outcomes. Recent online resource-management research demonstrates that jointly managing LLC and memory bandwidth can improve throughput while reducing the overhead associated with configuration search. This suggests that future learning-based resource managers should move beyond single-resource prediction and develop models capable of reasoning about multiple interacting resources.

The practical implementation of learning-based cache management requires careful attention to overhead. Large and computationally expensive learning models are not necessarily appropriate for processor resource management because inference must be rapid and energy efficient. Lightweight decision trees, regression models, clustering algorithms, compact neural networks, and hybrid learning-heuristic controllers may provide more practical solutions. Model confidence should also be considered, allowing the system to use conservative fallback policies when predictions are uncertain.

Another major issue is generalization. Workloads encountered during deployment may differ from those used for model training. Cache organization, processor generation, memory subsystem, operating system, and application behaviour can all change the relationship between observed features and optimal allocation. Online adaptation, transfer learning, periodically updated models, and hardware-independent feature representations can improve robustness.

Fairness and quality of service must remain fundamental design objectives. Throughput maximization alone can disadvantage applications that are highly sensitive to cache interference. An effective manager should enforce minimum performance guarantees where necessary and optimize remaining resources according to throughput, fairness, or energy objectives. In cloud and multi-tenant systems, such mechanisms can reduce noisy-neighbour effects and improve performance predictability.

V. REFERENCES

Online ISSN: 2456-1495
CODEN (USA): JSLOBJ

1. Mittal, S. (2017). A survey of techniques for cache partitioning in multicore processors. *ACM Computing Surveys, 50*(2), Article 27, 1–39.

2. Chatterjee, B., Khan, S., & Pande, S. (2021). *Effective cache apportioning for performance isolation under compiler guidance*. arXiv.

3. García-García, A., Sáez, J. C., Castro, F., & Prieto-Matías, M. (2024). *LFOC: A lightweight fairness-oriented cache clustering policy for commodity multicores*. arXiv.

4. Sáez, J. C., Castro, F., Fanizzi, G., & Prieto-Matías, M. (2024). *LFOC+: A fair OS-level cache-clustering policy for commodity multicore systems*. arXiv.

5. Bai, Y., Huang, Y., Chen, S., & Li, R. (2025). PaLLOC: Pairwise-based low-latency online coordinated resource manager of last-level cache and memory bandwidth on multicore systems. *Journal of Systems Architecture, 164*, 103427.

6. Nejat, M., Manivannan, M., Pericas, M., &Stenström, P. (2019). Coordinated management of processor configuration and cache partitioning to optimize energy under QoS constraints. arXiv.

7. Bitirgen, R., Ipek, E., & Martinez, J. F. (2008). Coordinated management of multiple interacting resources in chip multiprocessors: A machine learning approach. In *Proceedings of the 41st Annual IEEE/ACM International Symposium on Microarchitecture* (pp. 318–329).

8. Beckmann, N., & Sanchez, D. (2016). Modeling cache performance beyond LRU. In *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*.

9. Qureshi, M. K., & Patt, Y. N. (2006). Utility-based cache partitioning: A low-overhead, high-performance, runtime mechanism to partition shared caches. In *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture* (pp. 423–432).

Online ISSN: 2456-1495
CODEN (USA): JSLOBJ

10. Jaleel, A., Theobald, K. B., Steely, S. C., Jr., & Emer, J. (2010). High performance cache replacement using re-reference interval prediction. In *Proceedings of the 37th Annual International Symposium on Computer Architecture* (pp. 60–71).

E-mail: ***editor.jslsci888@gmail.com***