NUMA-Aware Task Scheduling Strategy Aiming to Reduce Cache Conflicts

Thiago de Campos Ribeiro Nolasco[1], **Pedro Henrique Penna** [2], **Henrique Cota de Freitas** [1]

[1]Department of Computer Science
Pontificial University Catholic of Minas Gerais (PUC Minas)
30.535-901, Belo Horizonte, MG, Brazil

[2]Microsoft Research
Redmond, U.S.A.

tcrnolasco@sga.pucminas.br, ppenna@microsoft.com, cota@pucminas.br

Abstract. *This paper presents a NUMA-aware scheduling strategy that reduces cache conflicts by analyzing recent cache index access histories through cumulative distribution functions (CDF). The approach aims to minimize last-level cache (LLC) interference while maintaining load balance across CPUs. We developed a Rust-based simulator to evaluate the strategy under Zipf-distributed workloads, comparing it against the Distributed Intensity Online (DIO) strategy. Results show that the proposed method improves cache hit rates by up to 8.2%, reduces load imbalance up to 18%, and decreases tail latency by 14% relative to DIO. These improvements highlight the potential of fine-grained cache-oblivious scheduling strategies for real-world operating systems.*

1. Introduction

Large-scale decentralized servers commonly process massive volumes of workloads in short time frames to meet heterogeneous user demands (Villalba, 2023). However, during peak loads, task turnaround times often increase sharply; delays that can be amplified by many reasons such as task ordering, resource contention, and scheduling inefficiencies (Turchetta and Gardner, 2023). Such slowdowns risk degrading Quality of Service (QoS), and in extreme cases, violating Service Level Agreements (SLAs).

Most common mitigation strategies focus on infrastructure upgrades, adding hardware capacity, or replicating servers. While effective, these solutions involve significant cost-benefit trade-offs, as performance gains are not always proportional to investment. An alternative approach, often more cost-efficient, lies in optimizing the resource manager and scheduler, two core components of the operating system (OS) that directly influence system latency and throughput. Poor resource allocation can leave hardware underutilized or overloaded, while ineffective scheduling can significantly reduce throughput by introducing unnecessary execution delays.

The authors would like to thank the National Council for Scientific and Technological Development of Brazil (CNPq - Codes 311697/2022-4 and 402837/2024-0), the Coordination for the Improvement of Higher Education Personnel - Brazil (CAPES - Grant PROAP 88887.842889/2023-00 - PUC/MG, Grant PDPG 88887.708960/2022-00 - PUC/MG - Informatics, and Finance Code 001), and PUC Minas FIP 2024/30947.

Given that scheduling heuristics are highly context dependent, the topic has been extensively studied over the past decade with many different areas to study. For example, Penna et al. (2019) addresses the workload-aware loop scheduling. On the other side, Zhou et al. (2021) provide an extensive review of task scheduling strategies across diverse environments, targeting metrics such as execution time, energy efficiency, and resource utilization. Nevertheless, rapid technological evolution, shifting workload profiles, and increasingly complex application requirements highlight a persistent gap for new strategies and adaptive approaches (Gupta et al., 2021).

Non-Uniform Memory Access (NUMA) architectures have become the standard in modern multi-socket and manycore systems due to their scalability and energy efficiency. In a NUMA system, processors are organized into nodes, each comprising a group of cores and a local memory region. While cores can access memory across nodes, such remote access incurs higher latency and reduced bandwidth compared to local memory accesses. Additionally, NUMA designs often include shared cache hierarchies within nodes, introducing potential contention between tasks executed on cores that share the same cache (Majo and Gross, 2011; Jiang et al., 2008). This interplay between memory locality, cache contention, and task placement makes NUMA-aware scheduling a critical factor in achieving high performance. For example, poorly planned scheduling can lead to excessive cache evictions, increasing memory latency, and degraded temporal locality, all of which can reduce overall throughput.

In this context, prior work has shown that simply balancing load across cores is not sufficient. Zhuravlev et al. (2010) demonstrate that such strategies ignore contention effects arising from shared resources such as memory controllers and last-level caches (LLC). To address this, they propose the Distributed Intensity (DI) and its online variant DIO, which classify tasks by their LLC miss intensity and then distribute them to reduce harmful interference. While effective at mitigating contention, these strategies highlight that fairness in scheduling requires more than equalizing CPU load, since cores sharing caches may still slow each other down despite balanced distribution.

Building on these insights, we propose a NUMA-aware scheduling strategy that reduces cache index conflicts by leveraging tasks' recent LLC access patterns modeled as cumulative distribution functions (CDFs). Unlike DI/DIO, which relly on overall miss rates, our approach selects NUMA groups with minimal CDF's percentiles overlap, thereby extending cache residency, improving temporal locality, and naturally balancing load across vCPUs (more in Section 3.3). Implemented in a simulator of vCPUs, cache hierarchies, and Zipf-distributed accesses, our method achieves up to 8.2% higher shared-cache hit rates, 18% better load distribution across NUMA groups, and 14% lower long-tail waiting times compared to DIO. Therefore, the main contributions of this work are:

- **A novel NUMA-aware scheduling strategy**: We propose a new task-to-core mapping approach that reduces cache misses by minimizing conflicts in LLC sets within NUMA nodes. Unlike prior strategies such as DI/DIO, our method relies on recent cache index access history rather than overall miss rates, requires no offline profiling or repeated execution, and naturally facilitates load balancing across CPUs.
- **A custom simulation environment**: We develop a Rust-based simulator that

models vCPUs, NUMA cache hierarchies, and Zipf-distributed memory accesses. This environment enables controlled experimentation and evaluation of scheduling strategies under diverse workload and architectural configurations.

The rest of the paper is organized as follows: In Section 2, we discuss related works about scheduling strategies in NUMA contexts and also how cache behavior influences on scheduling. In Section 3 we present the architecture, modeling, and implementation details of the simulation environment, the details of the task model used, the online NUMA-aware approach, and the scenarios and metrics used for evaluations. In Section 4, we present the results and discuss the findings. In Section 5, we summarize the key findings and contributions, limitations, and directions for future research.

2. Related Work

Efficient task scheduling is a long-standing challenge in High Performance Computing (HPC) environments, especially where parallel applications require careful coordination to optimize resource use and minimize latency. A variety of approaches have been proposed to address these issues, often focusing on thread or task placement strategies that aim to reduce contention and improve locality, either in communication, memory access, or CPU utilization.

NUMA-aware scheduling has gained significant attention due to the challenges posed by non-uniform memory access latencies and shared cache hierarchies in modern multicore and multisocket systems. Several studies have specifically tackled contention management in NUMA systems. Drebes et al. (2016) present dynamic, application-independent runtime algorithms for NUMA-aware task and data placement in task-parallel applications, leveraging inter-task data dependencies to optimize locality and scalability. Daci and Tartari (2013) provide a comprehensive review of contention-aware scheduling algorithms, highlighting challenges unique to both UMA and NUMA architectures, and discussing the trade-offs involved in managing shared resource contention. Blagodurov et al. (2010) demonstrate that effective contention management in NUMA environments requires joint consideration of thread placement and memory location, as conflicts arise not only from remote access latency but also from shared cache and memory controller contention.

Cache behavior critically influences scheduling decisions because cache misses and evictions have a direct impact on performance. Guan et al. (2009) explore cache-aware scheduling strategies for real-time multicore systems, proposing cache space isolation techniques that allocate fixed cache partitions to tasks to avoid interference in shared L2 caches. Their work demonstrates how such isolation enables predictable timing behavior and reduces contention, highlighting the importance of cache-conscious scheduling policies. In contrast to strict partitioning, the Distributed Intensity Online (DIO) scheduler (Zhuravlev et al., 2010) dynamically measures cache miss rates in LLC using hardware counters and distributes tasks across cores to balance memory intensity. This approach adapts online to workload phases, reducing harmful interference while avoiding the rigidity of fixed partitioning. Similarly, PAM (Performance/Power Aware Meta-scheduler) (Banikazemi et al., 2008) adopts a dynamic strategy but extends the focus beyond performance to include power and energy. PAM uses hardware performance counters to monitor metrics such as cycles-per-instruction (CPI) and L2 miss ratios, and employs a cache occupancy model to predict the impact of process-to-core assignments. Rather than relying

on fixed partitions, PAM continuously remaps tasks across cores and leverages cpusets to avoid placing high-footprint processes on the same L2 cache, thereby reducing cache contention while adapting to system power constraints. In our paper, we include DIO as a comparison approach to analyze how our online load and contention-aware scheduling strategy performs in a multicore NUMA environment.

Despite these advances, existing scheduling approaches frequently operate at a coarse granularity, considering memory node affinity or cache sharing at the page level, without explicitly analyzing fine-grained cache set conflicts between concurrent tasks. Furthermore, many techniques rely on offline profiling or application-specific information, which can limit their adaptability to workload changes. In contrast, our approach introduces lightweight, online heuristic that dynamically assign tasks to NUMA groups and CPUs based on recent cache set access histories. This method aims to reduce cache evictions and strengthen temporal locality in shared-cache environments without requiring prior knowledge.

3. Task Scheduling Strategy

As presented previously, the main objective of this paper is to present a new NUMA-aware scheduling strategy that aims to reduce inter-task conflicts in the last-level NUMA cache. In this section, the simulation environment's modeling is presented, followed by the tasks' modeling, and finally, a detailed explanation of the scheduling strategy and the scenarios of evaluations.

3.1. Simulator Design

A simulated environment was used[1] to compare the results of the implementations due to its ability to provide greater control over the variables and minimize the influence of external factors. As aforementioned, the simulator has three modules in its architecture (Figure 1), namely "Scheduler", "vCPUs", and "Post-Processing".

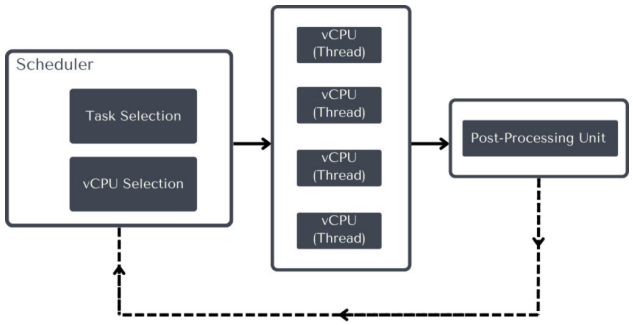

Figure 1. Example architecture of the simulation object.

For **Scheduler**, it is responsible for selecting Tasks and sending them to the vCPUs, this module can be seen as a load balancer. Currently, for task selection, the First-In, First-Out (FIFO) strategy is the only one implemented; for vCPU selection, the one proposed in this paper and DIO are implemented. This module repeats the selection steps until all tasks are completed. After selecting the task and the vCPU, the scheduler opens

[1] https://github.com/cart-pucminas/sched_sim

a communication channel with the vCPU to send the task. This communication channel is important because it is asynchronous, meaning that there is no blocking of thread execution while data is sent/received.

For the **vCPUs** module, they stay idle until a task arrives on their communication channel (i.e., a task is scheduled to them). When a task is received, the vCPU translates the task's virtual addresses to physical addresses (through 4-level pagination) and runs a cache lookup, as seen in Figure 2. In this step, all cache accesses, hits, and misses are stored. To simulate the access latency, the vCPU's thread is put to sleep for specific periods of time. When the vCPU finishes processing its time slice, it opens communication with the third module and sends the task. Finally, for **Post-Processing**, the role of this module is to check if the task has finished its processing; if it has, it "finishes" the task; otherwise, it resend it to the Scheduler.

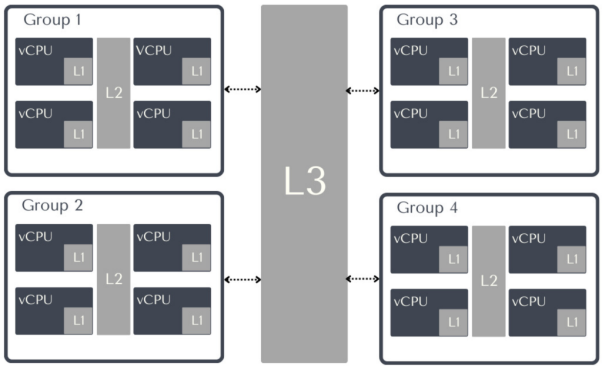

Figure 2. Representation of a NUMA architecture in the simulation environment. Each vCPU has its own L1 cache; each NUMA-group has its own L2 cache; all NUMA-groups share a single L3 cache.

Therefore, the simulation requires at least three threads to function (one for each module), and increasing the number of vCPUs increases the number of threads used in the vCPU module. The user can choose the number of vCPUs they want in the simulation, as well as the number of tasks, which scheduling strategy to use, and architecture-level details such as cache size (word size, number of words per block, associativity level) and NUMA size (number of vCPUs in each NUMA-group).

3.2. Task and Workload Modeling

In this paper, we call a *Task* the abstraction of a process or thread; in our context, anything that performs computation and issues memory accesses is modeled as a Task. In our simulation, a Task is defined by a set of synthetic virtual memory addresses of size equal to N.

To emulate realistic memory behavior, we generate Task addresses according to a Zipfian distribution. The Zipf law has been widely used to model skewed access patterns, including word frequencies in natural language, web access traces, and memory reference distributions (Yang and Zhu, 2016). It is particularly suitable for our purpose because it naturally captures the principle of *temporal locality*, i.e., a small subset of addresses is accessed disproportionately often, in contrast, the majority of addresses are accessed rarely. This property mirrors real workloads where a "hot set" of memory lines dominates cache activity.

Formally, the probability of accessing the k-th most popular item under a Zipf distribution is given by:

$$P(X = k) = \frac{1/k^s}{\sum_{n=1}^{N} 1/n^s}, \tag{1}$$

where N is the number of distinct items (in our case, the address space size), and s is the *skew parameter*. A larger s increases the bias toward frequently accessed addresses (strengthening locality), while smaller s values approach a uniform distribution over the N items. In our simulation, it was used $s = 1$, aiming for a distribution that has spots of high temporal locality; in this case, low addresses are more frequent.

Thus, in this work, each task is modeled as having a workload of 3000 addresses generated according to the Zipfian distribution. The workload size is 10 times the preemption *quantum*, which ensures that tasks lives long enough so both strategies have enough iterations to impact in the results. During execution, task's run information are obtained, and the resulting metrics are used in this work.

3.3. Scheduling Strategy

The simulated environment executes tasks on multiple vCPUs organized into NUMA-groups. Each group represents a set of vCPUs sharing a common last-level cache (L2 in our case), which makes the selection of the execution group critical to performance. The scheduler must balance load distribution while minimizing cache contention between co-located tasks. The implemented strategy to guide this section, the CDF percentile-based approach, operates dynamically, considering the recent cache set usage patterns of running tasks to make informed scheduling decisions. It was chosen to use cache sets instead of cache ways because this strategy is idealized to be agnostic to cache-way replacement policies.

The approach compares the distributions of cache index accesses between the task to be scheduled and the tasks currently running in each NUMA-group by examining their cumulative distribution functions. The scheduler calculates differences at key percentiles (25th, 50th, 75th) between the task's CDF and the aggregate CDF of tasks in each group, as seen:

$$cdf\_dist_g = \sum_{p \in P} \mathbf{1}(|CDF_{T_{new}}(p) - CDF_{group_g}(p)| > \theta_p) \tag{2}$$

if a percentile difference exceeds the predefined threshold θ_p, the percentile p is considered "valid". If the majority of percentiles (2 or more) are "valid", the group is considered a "good group" due to its low similarity in cache usage patterns. In our simulation, θ_p represents a percentage of $group_g$'s percentile, which represents a distance from the percentile where adding new accesses would be troublesome; the wider the percentage, the greater must be the distance to the percentile to be "valid".

As seen in Algorithm 1, if any group has no tasks, it is immediately selected. This ensures workload balance between groups. If multiple groups qualify as "good group", the one with the fewest running tasks is chosen. This also ensures workload balance between groups. Within the selected group, the scheduler picks the vCPU with the lowest current workload. Since the definition of a good group is discretized, when ties between

Algorithm 1: Best vCPU Selection Heuristic (CDF Percentile Distance + Least Workload)

Input: T_{new}: task to be scheduled, G: set of NUMA groups, P: set of percentiles, θ: percentile threshold
Output: $vCPU_{best}$: selected vCPU
foreach $g \in G$ **do**
  $count_g \leftarrow 0$;
  foreach $p \in P$ **do**
    if $|CDF_{T_{new}}(p) - CDF_g(p)| > \theta_p$ **then**
      $count_g \leftarrow count_g + 1$;
  $good_g \leftarrow (count_g \geq \lceil |P|/2 \rceil)$;
if $\exists g \in G$ *with no running tasks* **then**
  $g_{best} \leftarrow$ any such empty group;
else
  $G_{good} \leftarrow \{g \in G \mid good_g = true\}$;
  if $G_{good} \neq \emptyset$ **then**
    $g_{best} \leftarrow \arg\min_{g \in G_{good}} |tasks\_running(g)|$;
  else
    $g_{best} \leftarrow \arg\min_{g \in G} |tasks\_running(g)|$;
$vCPU_{best} \leftarrow \arg\min_{v \in g_{best}} workload(v)$;
return $vCPU_{best}$;

groups occur, the load balancing is easier to achieve, strategies such DI/DIO tend to be harder to discretize, since their key metric to schedule is the miss-ratio, a continuous value.

In the implementation, each NUMA-group maintains its own hash map (lookup cost $\mathcal{O}(1)$) that records the frequency of cache index accesses for all currently scheduled tasks within the group. This map is updated after every scheduling or preemption event, with an update cost of $\mathcal{O}(n)$, where n is the number of distinct cache indexes recently accessed by the task. To compute the NUMA-group's CDF, the map is traversed and the frequencies of all cache indexes are summed, requiring $\mathcal{O}(m)$ time, where m is the number of unique cache indexes accessed within the group. From that, the desired percentiles are obtained.

3.4. Experimental Setup

To obtain the following results, we created a base architecture for the simulation with the specifications in Table 1, using Intel's Alder Lake specs (Fog, 2025) as an approximated baseline. It is important to point out that changes in the architecture might impact in the simulation's hit rate, which might alter the effectiveness of the strategies (just like the task's memory addresses access pattern). Moreover, changing the access latencies will impact the results in the waiting times.

As for metrics of evaluation, they are: (i) proportion of cache hits per cache accesses (hit rate); (ii) 99th percentile (p99) of tasks' waiting time in simulation, since those

are the tasks that were most impacted by slowdown; and (iii) coefficient of variation of each group's load, to measure the load balance between NUMA-groups. A simple implementation of Round Robin, a strategy known for its lightweight nature and load balancing capabilities, and DIO are used as baseline algorithms for comparison. For each strategy, five executions were run, and the metrics obtained are the mean of these executions, with results normalized to those of Round Robin.

Table 1. Base architecture specifications used in the simulation.

Component	Specification
vCPUs per NUMA-group	4
Number of vCPUs	12
L1 Cache (per vCPU)	512 KiB, 4 ways, 4 words per block, LRU replacement, access latency = 2 ns
L2 Cache (per NUMA-group)	2 MiB, 4 ways, 4 words per block, LRU replacement, access latency = 6 ns
L3 Cache (shared)	8 MiB, 4 ways, 4 words per block, LRU replacement, access latency = 20 ns
RAM (shared)	4 GiB, frame size = 4 KiB, access latency = 100 ns

4. Results and Discussion

In this section, we present a comprehensive evaluation of the proposed NUMA-aware scheduling strategy, comparing it with the DIO (Zhuravlev et al., 2010) strategy, within our custom simulation environment. Our primary goal is to assess how effectively the proposed strategy minimizes cache set conflicts and increases cache hits across vCPUs grouped by shared L2 caches in a NUMA architecture, also keeping a good load balance between groups.

We analyze key performance metrics, including cache hit rate, task completion times, and load balance under varying workload characteristics generated using Zipf-distribution access patterns. This evaluation enables us to identify strengths and trade-offs of the CDF-approach, providing insights into its practical applicability in NUMA-aware task scheduling. It is important to point out that we considered DIO's LLC as the LLC of a NUMA-group (i.e., L2).

First of all, as presented in Section 3.3, the CDF-approach has a parameter θ_p (called here tol.) that defines how wide the distance (in percentage) must be to ensure that the task (to be scheduled) will not cause more conflicts. Figure 3 shows the normalized mean hit rates in each cache level per strategy. As intended, both the CDF-approach and DIO produces more hit rates in the NUMA-LLC when compared with the baseline, with DIO producing up to 17% more hit rate, and $CDF_{0.7}$ producing up to 26%. When comparing DIO with the CDF-approach, it is possible to see that the CDF-approach always (varying tol.) has a higher hit rate in the NUMA-LLC, with higher hit rates on $CDF_{0.7}$, up to 8.2% more hit rate; $CDF_{0.3}$ produced up to 3% more hit rate. The low values on hit rate in L3 are related both to the initial cold start (which brings down the hit rate) and the lack of accesses in L3 because of the high hit rate in L2. Since all strategies don't focus on L1, it is possible to see that those values are, pretty much, always the same.

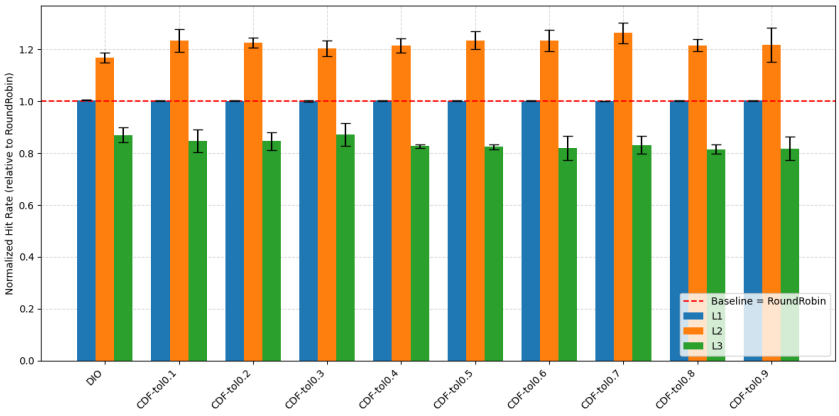

Figure 3. Comparison of mean executions' hit rate in different cache levels between CDF-approach and DIO, the higher the better. Results normalized.

As previously presented in Section 3.3, the CDF-approach tends to have a better load balance between NUMA-groups since it is easier to identify draws on the "best group" selection, which is not as easy in DIO. To assess the balance of the strategies' decisions, we compute the mean coefficient of variation (CV) of the total load distributed on each NUMA-group across all runs. As seen in Figure 4, the DIO strategy achieves the worst load balance when compared to the baseline. Because it relies on continuous values (LLC miss ratios) to guide task placement, small numerical differences between groups can bias scheduling decisions even when the groups are, in practice, nearly equivalent. In such cases, tasks may be steered toward one group unnecessarily rather than considering them as ties and applying a load-based tie-breaker.

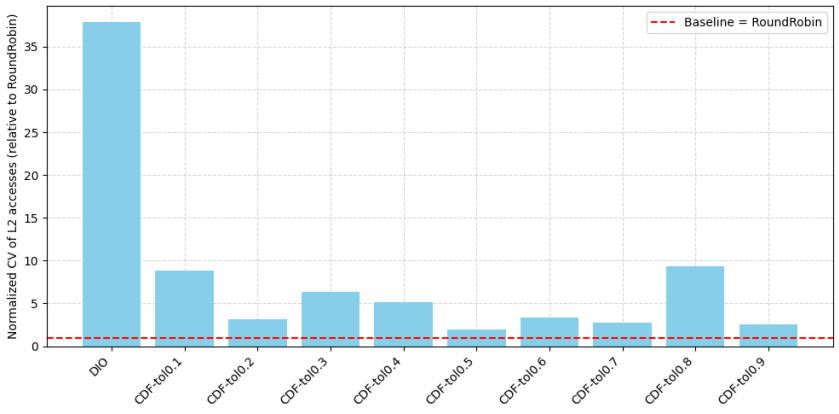

Figure 4. Comparison of mean executions' coefficient of variation of each NUMA-group's total load between CDF-approach and DIO, the lower the better. Results normalized.

Moreover, DIO evaluates groups solely on their aggregate miss intensity, which may overlook situations where a group exhibits a slightly lower miss ratio but already hosts a larger number of tasks. This can lead to unbalanced workloads, as the scheduler continues to favor a group that appears "lighter" in terms of intensity while actually being more heavily loaded in terms of task count. The CDF-approach produces better load balance when compared to DIO mainly because the key decision making is discretized

(valid or not valid), making it easier to find draws, and the tie breaking is to select the least loaded NUMA-group, but it is still worse than the baseline, which was expected since Round Robin focuses only on resource fairness.

Lastly, Figure 5 shows the final comparison of mean executions' p99. waiting times. Just like Figure 4, the values obtained with the CDF-approaches are always better (varying tol.) as expected. It is possible to see that the values in general do not vary as when tol. is in range of 0.5 to 0.8, but this value change a lot in $CDF_{0.1}$; 0.1 it implies that the distance of percentiles must be only 10%, so the majority of tasks will get the percentiles as *valid*, in this case, all groups tend to be considered as *good groups*, so they will be just scheduled to balance load (as seen in Algorithm 1). With this workload, $CDF_{0.3}$ was the one that produced the best results, reducing almost 14% on the p99. waiting times when compared to the DIO approach. When compared to the baseline, the CDF-approach yields worse values (in the best scenario, up to 17%) despite having competitive load balance and a higher hit rate. This is because Round Robin is a strategy that is faster to schedule and generates little contention, as it requires minimal information about the system. In contrast, the CDF-approach requires updating the NUMA-group CDF after each schedule decision and, also, identifying the least loaded group among the found *good groups*.

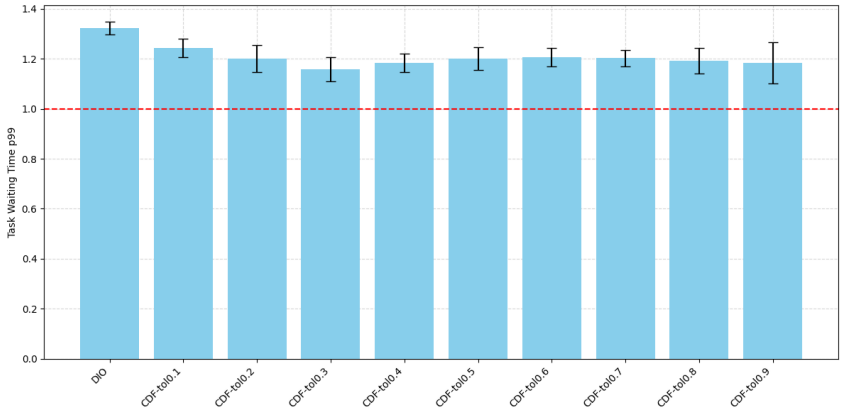

Figure 5. Comparison of mean executions' p99 waiting times between CDF-approach and DIO, the lower the better. Results normalized.

Overall, the experiments demonstrate that the CDF-based scheduler consistently reduces cache interference while maintaining balanced task distribution. The improvement in cache hit rates (up to 8.2%) and reduction in load imbalance (18%) are particularly significant, since they directly translate to reduced task waiting times and improved overall system throughput. Furthermore, the observed reduction of 14% in tail latency indicates that the strategy not only improves average performance but also provides more predictable execution, a critical aspect for latency sensitive applications. Taken together, these findings suggest that CDF-based scheduling offers a good direction for combining cache-awareness with load balancing in NUMA systems.

5. Conclusion and Future Work

In this work, we propose a NUMA-aware task scheduling strategy that reduces cache conflicts by leveraging the cumulative distribution functions (CDFs) of tasks' recent cache

index accesses. Our approach aims to minimize harmful interference in shared last-level caches while naturally facilitating load balancing among NUMA groups. To evaluate it, we developed a custom Rust-based simulation environment capable of modeling cache hierarchies, virtual CPUs, and Zipf-distributed workloads.

The evaluation demonstrated that the CDF-based strategy outperforms DIO in both cache hit rates and load balancing with the used workload. Specifically, it improved cache hit rates by up to 8.2%, reduced load imbalance by 18%, and decreased task tail latency by 14%. These quantitative gains reinforce the potential of cache-oblivious scheduling to enhance both performance and fairness in NUMA systems, particularly for applications that are latency-sensitive or cache-thrashing-sensitive. Moreover, the simulation environment itself proved to be a useful artifact; beyond serving as a testbed for our experiments, it can also be employed as a teaching tool to help the understanding of NUMA effects, cache contention, and scheduling trade-offs.

As future work, we plan to extend our study in four directions. First, we aim to incorporate real applications' memory access patterns into the simulation, making the evaluation closer to production workloads, comparing performance with applications that also have access patterns with low temporal locality. Second, we intend to integrate the CDF-based strategy into a real operating system scheduler, assessing its effectiveness and overhead in practice. Third, we compare the CDF-based approach with broader and more recent strategies presented in the literature. Finally, we plan to explore the impact of dynamically adjusting the scheduling threshold θ_p, investigating how task behavior during execution can guide adaptive policies that further improve performance and fairness.

References

M. Banikazemi, D. Poff, and B. Abali. Pam: A novel performance/power aware meta-scheduler for multi-core systems. In *SC '08: Proceedings of the 2008 ACM/IEEE Conference on Supercomputing*, pages 1–12, 2008. doi: 10.1109/SC.2008.5222643.

S. Blagodurov, S. Zhuravlev, A. Fedorova, and A. Kamali. A case for numa-aware contention management on multicore systems. In *Proceedings of the 19th International Conference on Parallel Architectures and Compilation Techniques*, PACT '10, page 557–558, New York, NY, USA, 2010. Association for Computing Machinery. ISBN 9781450301787. doi: 10.1145/1854273.1854350. URL https://doi.org/10.1145/1854273.1854350.

G. Daci and M. Tartari. A comparative review of contention-aware scheduling algorithms to avoid contention in multicore systems. In V. V. Das, editor, *Proceedings of the Third International Conference on Trends in Information, Telecommunication and Computing*, pages 99–106, New York, NY, 2013. Springer New York. ISBN 978-1-4614-3363-7.

A. Drebes, A. Pop, K. Heydemann, N. Drach, and A. Cohen. Numa-aware scheduling and memory allocation for data-flow task-parallel applications. In *Proceedings of the 21st ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, PPoPP '16, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 9781450340922. doi: 10.1145/2851141.2851193. URL https://doi.org/10.1145/2851141.2851193.

A. Fog. *The Microarchitecture of Intel, AMD and VIA CPUs*, 2025. URL https://www.agner.org/optimize/microarchitecture.pdf. [Online].

N. Guan, M. Stigge, W. Yi, and G. Yu. Cache-aware scheduling and analysis for multicores. In *Proceedings of the Seventh ACM International Conference on Embedded Software*, EMSOFT '09, page 245–254, New York, NY, USA, 2009. Association for Computing Machinery. ISBN 9781605586274. doi: 10.1145/1629335.1629369. URL https://doi.org/10.1145/1629335.1629369.

M. Gupta, L. Bhargava, and S. Indu. Mapping techniques in multicore processors: Current and future trends. *The Journal of Supercomputing*, 77:9308–9363, 2021. doi: 10.1007/s11227-021-03650-6.

Y. Jiang, X. Shen, J. Chen, and R. Tripathi. Analysis and approximation of optimal co-scheduling on chip multiprocessors. In *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques*, PACT '08, page 220–229, New York, NY, USA, 2008. Association for Computing Machinery. ISBN 9781605582825. doi: 10.1145/1454115.1454146. URL https://doi.org/10.1145/1454115.1454146.

Z. Majo and T. R. Gross. Memory management in numa multicore systems: trapped between cache contention and interconnect overhead. In *Proceedings of the International Symposium on Memory Management*, ISMM '11, page 11–20, New York, NY, USA, 2011. Association for Computing Machinery. ISBN 9781450302630. doi: 10.1145/1993478.1993481. URL https://doi.org/10.1145/1993478.1993481.

P. H. Penna, A. T. A. Gomes, M. Castro, P. D.M. Plentz, H. C. Freitas, F. Broquedis, and J.-F. Méhaut. A comprehensive performance evaluation of the binlpt workload-aware loop scheduler. *Concurrency and Computation: Practice and Experience*, 31 (18):e5170, 2019. doi: https://doi.org/10.1002/cpe.5170. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.5170. e5170 cpe.5170.

W. Turchetta and K. Gardner. Understanding slowdown in large-scale heterogeneous systems. In E. Hyytiä and V. Kavitha, editors, *Performance Evaluation Methodologies and Tools*, pages 197–206. Springer Nature, Cham, Switzerland, 2023. ISBN 978-3-031-31234-2.

M. Villalba. Aws lambda functions now scale 12 times faster when handling high-volume requests. AWS News Blog, Nov. 2023. URL https://aws.amazon.com/blogs/aws/aws-lambda-functions-now-scale-12-times-faster-when-handling-high-volume-requests/. [Online].

Y. Yang and J. Zhu. Write skew and zipf distribution: Evidence and implications. *ACM Trans. Storage*, 12(4), June 2016. ISSN 1553-3077. doi: 10.1145/2908557. URL https://doi.org/10.1145/2908557.

G. Zhou, W. Tian, and R. Buyya. Deep reinforcement learning-based methods for resource scheduling in cloud computing: A review and future directions. arXiv preprint arXiv:2105.04086, 2021. URL https://arxiv.org/abs/2105.04086. [Online].

S. Zhuravlev, S. Blagodurov, and A. Fedorova. Addressing shared resource contention in multicore processors via scheduling. In *Proceedings of the Fifteenth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XV, page 129–142, New York, NY, USA, 2010. Association for Computing Machinery. ISBN 9781605588391. doi: 10.1145/1736020.1736036. URL https://doi.org/10.1145/1736020.1736036.