IBIMA Publishing
Communications of International Proceedings
https://ibimapublishing.com/p-articles/45AI/2025/4534325/
Vol. 2025 (20), Article ID 4534325

Evaluating the Scalability of Dependency-Aware Scheduling in Multicore Systems through Large-Scale Simulation Trials*

Patryk SERAFIN

Faculty of Cybernetics, Military University of Technology,
00-908 Warsaw, Kaliskiego 2 Street, Poland,
ORCID: 0009-0008-0573-1976

Correspondence should be addressed to: Patryk SERAFIN, patryk.serafin@wat.edu.pl

* Presented at the 45th IBIMA International Conference, 25-26 June 2025, Cordoba, Spain

Abstract

Task scheduling in multicore systems becomes increasingly complex in the presence of inter-task dependencies, which traditional operating system schedulers are not designed to manage. As a result, tasks are often activated before their prerequisites are satisfied, leading to passive waiting and inefficient CPU utilization. Although several dependency-aware approaches have been proposed, especially in heterogeneous or distributed computing, they often rely on centralized orchestration, static task graph analysis, or dedicated hardware support, limiting their applicability in general-purpose or dynamic environments. Addressing this gap, the Dependency-Aware Model (DAM) has been introduced as a lightweight, user-space scheduling mechanism that delays task activation until all declared dependencies are resolved. To evaluate the scalability and robustness of this model, an extensive empirical campaign was conducted using a dedicated simulation environment capable of executing compute-bound workloads under randomized configurations. The simulator supports variations in thread count, execution time distributions, CPU affinity, and dependency density, while ensuring that task relationships form valid Directed Acyclic Graphs (DAGs). Over 39,000 simulation trials were performed to compare standard and dependency-aware scheduling under diverse conditions. The results consistently demonstrate that DAM improves scheduling efficiency, reducing execution time by over 28% in single-test runs and exceeding 30% in ten-test batches. These findings confirm the model's practical effectiveness and its suitability for real-world multicore systems with nontrivial dependency structures.

Keywords: Dependency-aware Scheduling, Multicore Task Execution, Simulation-based Evaluation, Performance Analysis

Introduction

Modern multicore computing systems face increasing challenges in managing the efficient execution of parallel tasks, particularly in contexts involving complex inter-task dependencies. Scheduling such workloads requires not only optimal resource allocation across processor cores but also mechanisms that avoid activating tasks before their required conditions are satisfied. Traditional system schedulers, commonly used in general-purpose operating systems such as Windows or Linux, are not designed to take task dependency structures into account (Microsoft, 2021; Linux Foundation, 2010). As a result, processor resources may be unnecessarily occupied by threads that cannot proceed, leading to reduced performance and increased total execution time. This issue has been addressed in several studies through the introduction of high-level scheduling frameworks, middleware components, or dependency-aware algorithms designed for specific environments such as grid computing or real-time embedded systems (Topcuoglu and Hariri, 2002; Lyberis et al., 2016; Cheng et al., 2024). However, many of these approaches rely on centralized control, static analysis, or specialized hardware assumptions, making them difficult to adopt in dynamic, general-purpose settings.

To address these issues, a new task execution model was introduced in previous research (Serafin, 2025a). This model, known as the Dependency-Aware Model (DAM), delays task activation until all declared

Cite this Article as: Patryk SERAFIN, Vol. 2025 (20) "Evaluating the Scalability of Dependency-Aware Scheduling in Multicore Systems through Large-Scale Simulation Trials" Communications of International Proceedings, Vol. 2025 (20), Article ID 4534325, https://doi.org/10.5171/2025.4534325

dependencies have been fulfilled. The model has been integrated into a simulation environment that supports configurable workloads and provides reliable emulation of real CPU usage (Serafin, 2025b). The simulation framework enables precise comparisons between dependency-aware scheduling and standard scheduling methods, under diverse runtime configurations. The current study builds on this foundation by performing a large-scale empirical evaluation of the DAM model. An extensive number of simulation trials were conducted, covering a broad range of parameter combinations related to task dependencies, execution durations, arrival timings, and CPU affinity. The purpose of this experimental campaign was to quantify the performance gains achievable through dependency-aware scheduling and to determine the robustness of the approach across varied task scenarios.

Two separate sets of experiments were executed. The first involved simulations composed of a single test per run, offering highly granular insights into individual execution cases. The second group contained simulations with ten tests each, allowing aggregate performance to be visualized more effectively. Both sets contributed to a comprehensive statistical analysis of the model's effectiveness.

Results from this evaluation confirm that the DAM approach significantly reduces idle processor occupation and improves scheduling efficiency, especially in scenarios with a high proportion of dependent tasks. These outcomes also validate the underlying simulation platform as a reliable and extensible tool for testing scheduling methodologies in multicore environments.

Traditional system schedulers, commonly used in general-purpose operating systems such as Windows or Linux, are not designed to take task dependency structures into account (Microsoft, 2021; Linux Foundation, 2010). As a result, processor resources may be unnecessarily occupied by threads that cannot proceed, leading to reduced performance and increased total execution time. In this context, two core concepts become essential: Directed Acyclic Graphs (DAGs) and CPU affinity. A DAG represents a finite set of tasks and dependencies arranged in a directed graph without cycles, ensuring that no task is revisited and the execution order remains well-defined (IBM, 2024). CPU affinity, on the other hand, refers to the ability to bind specific threads or processes to particular processor cores to reduce context switching and improve performance, especially in multicore and NUMA systems (NVIDIA, 2023). Both constructs are central to the design and analysis of scalable task scheduling algorithms in parallel environments.

Experimental Setup

To rigorously evaluate the efficiency and reliability of the Dependency-Aware Model (DAM) across a broad range of scenarios, an extensive series of simulations was conducted. The experiments were designed to assess how the DAM performs under diverse system conditions and varying task dependency patterns. This section outlines the simulation framework used for testing, the randomized parameter generation strategy, and the overall structure of the simulation campaign.

Simulation Framework Overview

All simulations were executed using the simulation environment previously developed by Serafin (2025b). The environment supports real CPU-bound execution and enables direct comparisons between two task scheduling approaches: the standard scheduler (baseline model) and the Dependency-Aware Model (DAM) (Serafin, 2025a).

In the baseline model, tasks are launched immediately upon creation, regardless of whether their dependencies are satisfied. If a task includes dependencies, it must wait internally within its execution thread, leading to unnecessary CPU occupation. In contrast, DAM intercepts incoming tasks and holds their execution until all prerequisite signals have been received, thus avoiding idle or wasteful resource use.

The simulation system is modular, with dedicated components for configuration management, test execution, task generation, and dependency handling. This structure allows for reproducible tests and the consistent application of varying configurations across thousands of trials.

Parameter Sampling Strategy

Each simulation run is governed by a configuration that determines system and task-level behavior. These configurations were either defined manually or randomized using the Configuration Loader. Randomization covered the following key parameters:

- Initial number of tasks, available at simulation start,
- Number of additional task sets, which are introduced during runtime,
- Number of tasks per additional set,
- Minimum and maximum delay (in milliseconds) between arrivals of tasks within a set,
- Minimum and maximum task execution time (in milliseconds),
- Probability of assigning CPU affinity to a task,
- Probability of assigning dependencies to a task,
- Maximum number of dependencies per task.

Execution times were sampled using a beta distribution centered around focus values, allowing control over the typical task length while preserving variability. This distribution was selected for its flexibility in modeling skewed execution time profiles that commonly occur in real-world heterogeneous workloads (Pinedo, 2016). CPU affinity and dependency probabilities were drawn from a uniform distribution in the range [0,1], ensuring unbiased sampling across all configurations. Tasks with dependencies were connected through signal-based vectors, ensuring that the resulting task structure formed a valid Directed Acyclic Graph (DAG), with no self-referencing or cyclic dependencies. This configuration strategy enabled extensive coverage of the input space while maintaining the integrity and correctness of dependency constraints.

Simulation Campaign Structure

The simulation campaign was divided into two complementary groups to balance granularity and visualization clarity.

1. **Single-Test Simulations:**
   In this setup, each simulation executed a single test using a unique randomized configuration. A total of 37,938 such simulations were performed. This approach yielded highly detailed results, with each CSV record representing a single, distinct experiment. It allowed precise tracking of how configuration parameters affected execution outcomes on a per-test basis.
2. **Ten-Test Simulations:**
   In the second setup, 1,324 simulations were run, each consisting of ten individual tests using the same configuration. While this approach introduced result averaging within each simulation record, it was particularly suitable for visual interpretation. In this setup, bar plots were generated to illustrate the percentage improvement achieved across the ten tests, accompanied by the total number of task dependencies per test.

Each test execution compared the total runtime between the baseline model and the DAM-enhanced model. The primary metric collected was the percentage improvement in execution time achieved by DAM. Supporting statistics included average task duration, total number of dependencies, and configuration parameters. These values were stored in CSV format and served as the basis for the large-scale analysis discussed in the following sections.

To illustrate the diversity and structure of sampled configurations, Table 1 presents five randomly selected sets of input parameters used in the simulation campaign. Each column corresponds to an independent simulation run, showcasing the variation in thread generation parameters and dependency modeling. All values conform to the hierarchical structure defined in the YAML configuration format, as discussed in Section 0. Parameters are grouped according to their respective configuration domains: general and threads. These examples provide a concrete view of the randomized sampling strategy used to ensure a broad coverage of the parameter space.

Table 1 Example of randomized simulation parameters used in test generation

	No.	1	2	3	4	5
general	initial_threads_number	10	5	13	9	19
	additional_models_number	4	5	4	5	4
	additional_threads_number	6	9	4	2	2
	min_delta_time	2	14	9	15	16
	max_delta_time	30	26	50	17	47
threads	min_execution_time	645	2988	1027	17	3371
	max_execution_time	2756	3180	1626	197	4789
	cpu_affinity_enabled	True	True	True	True	True

	cpu_affinity_probability	0.55666	0.1642	0.79375	0.85104	0.73931
	dependencies_probability	0.99846	0.13857	0.90019	0.42879	0.18715
	max_dependencies	28	4	6	11	25

Statistical Overview of Results

This section presents a summary of the results obtained from the large-scale simulation campaign, which aimed to assess the effectiveness of the Dependency-Aware Model (DAM) in optimizing task scheduling. The analysis is based on the percentage time improvement (avg_improvement) observed when comparing DAM to the default system scheduler across two types of simulations. The first set includes single-test simulations with isolated executions, and the second set includes simulations that aggregate results from ten independent test cases.

Summary Statistics – Single-Test Simulations

The single-test simulation dataset consists of 37,938 unique records. Each record corresponds to a single configuration with randomly generated parameters and captures the direct performance comparison between DAM and the baseline approach. This method allows for precise and unaggregated insight into individual performance outcomes. Fig. 1 presents a histogram of the avg_improvement values. The distribution is clearly skewed toward positive values, indicating that in the vast majority of simulations, the use of DAM resulted in execution time reductions.

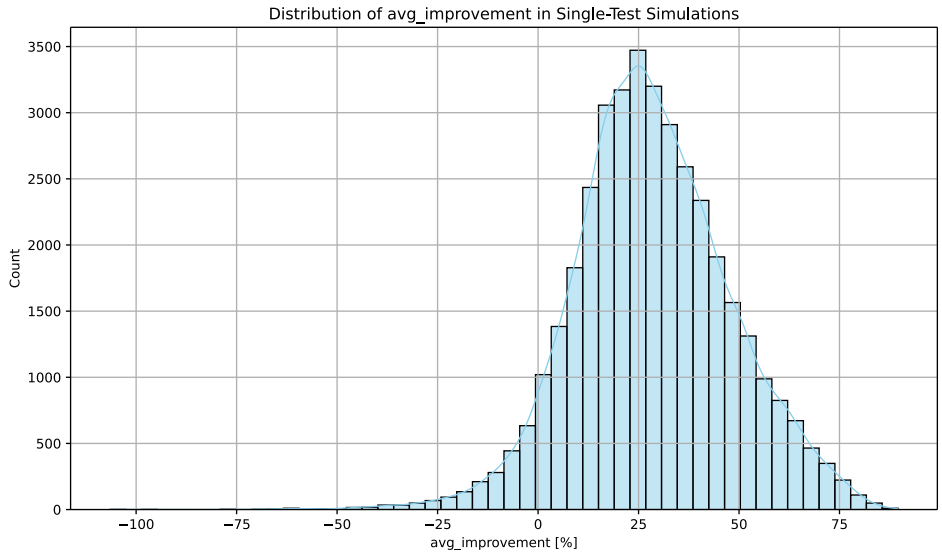

Fig. 1 Histogram of execution time improvements in single-test simulations using the DAM model.

A statistical summary of these values is presented in Table 2. The mean improvement was 28.43%, with 94.21% of all tests yielding positive values.

Summary Statistics – Ten-Test Simulations

The ten-test simulation dataset includes 1,324 records, each aggregating results from ten individual test cases. This approach enables more stable and visually interpretable trends, particularly when generating bar plots. Fig. 2 displays the histogram of avg_improvement values averaged over ten-test simulations. Compared to the single-test histogram, this distribution appears more concentrated around the mean, reflecting reduced variance due to averaging.

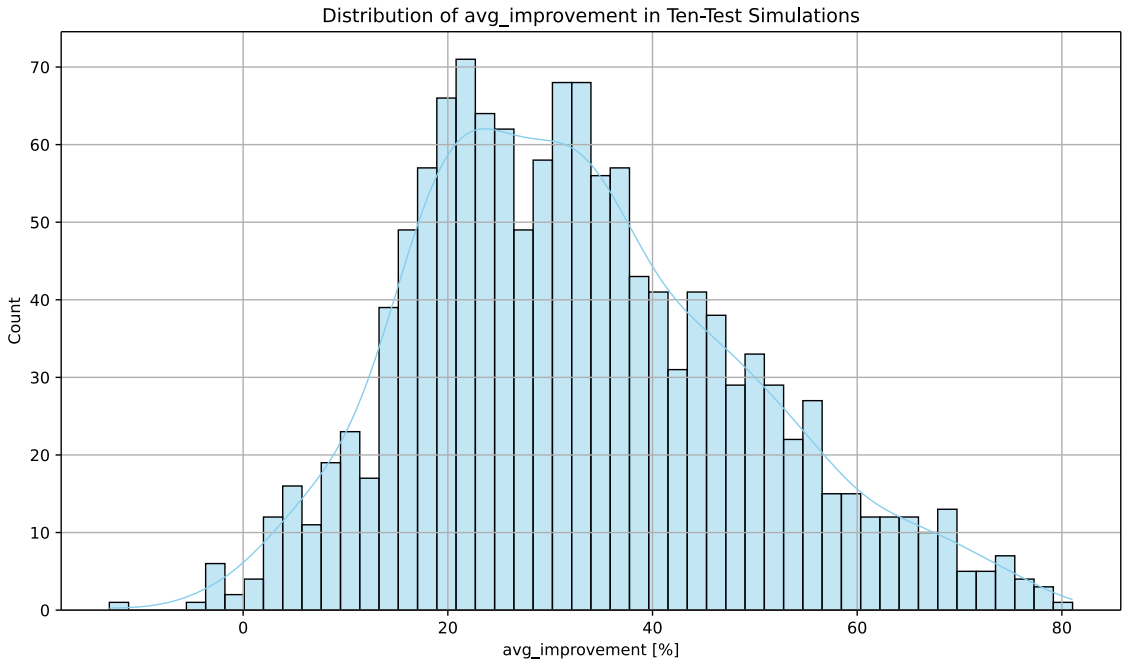

Fig. 2 Histogram of average execution time improvements across ten-test simulations using DAM.

Table 3 provides the corresponding descriptive statistics. The average improvement in this group reaches 32.80%, and 99.24% of all tests resulted in positive gains, indicating increased consistency in DAM's performance when tested repeatedly under fixed configurations.

Table 2 Descriptive statistics for avg_improvement (Single-Test Simulations)

Metric	Value
Mean	28.43%
Standard deviation	19.33
Max	89.53%
Min	–106.50%
Median	27.44%
1st quartile	16.07%
3rd quartile	40.69%
Number of unique values	37,938
Values > 0	35,741 (94.21%)
Values < 0	2,197 (5.79%)

Table 3 Descriptive statistics for avg_improvement (Ten-Test Simulations)

Metric	Value
Mean	32.80%
Standard deviation	16.11
Max	81.05%
Min	–13.08%
Median	31.09%
1st quartile	21.00%
3rd quartile	43.52%
Number of unique values	1,324
Values > 0	1,314 (99.24%)
Values < 0	10 (0.76%)

Interpretation of Distributions

The comparison between the two datasets highlights the strengths of DAM in diverse simulation scenarios. The high percentage of positive improvements in both cases confirms the model's general effectiveness. The single-test simulations provide detailed insight into edge cases and outliers, including the rare instances where DAM introduces scheduling overhead. These results emphasize the importance of evaluating scheduling models across a broad configuration space. The aggregated ten-test simulations offer a smoother performance profile and reflect a more reliable gain under repeated testing. Despite the smoothing effect, the improvements remain significant, and the higher average further validates the advantages of the model in controlled workloads. Together, the results provide robust empirical support for the effectiveness of the Dependency-Aware Model.

Visual Presentation of Selected Simulations

This section presents selected examples from the simulation campaign to illustrate various performance scenarios observed when using the Dependency-Aware Model (DAM). Each case is supported by a bar chart visualizing the percentage improvement per test. The examples were chosen to highlight the diversity of conditions under which the model was evaluated, including its effectiveness in simple, complex, and borderline configurations. All simulations were conducted on a real operating system with active CPU load. Due to the nature of system-level task execution, certain variability in measurements is inevitable, particularly in borderline cases where operating system background activity may interfere with precise timing.

Representative Positive Outcomes

Several simulations illustrate how the Dependency-Aware Model (DAM) improves task scheduling efficiency under various test conditions. One example includes a configuration with only two tasks and no inter-task dependencies. As expected, the model offered no measurable gain. In such cases, the execution overhead introduced by the model is unnecessary and both strategies perform identically.

In another simulation, 22 tasks with 13 dependencies yielded a 13.4% improvement in execution time. This moderate gain confirms that the DAM model begins to show tangible benefits as the number of dependencies increases, enabling it to prioritize ready tasks and reduce thread blocking.

A more complex case with 35 tasks and 97 dependencies showed a significant improvement of 71.77%. The model successfully avoided launching unresolved tasks and minimized unnecessary processor contention, leading to substantial execution time reduction.

Another scenario involved 170 tasks and 119 dependencies. Although the execution times of individual tasks were short, ranging from 55 to 624 milliseconds, the DAM model still provided a 9.31% gain. This result demonstrates that even in systems with short tasks and high concurrency, the model contributes positively by optimizing task readiness and reducing contention.

Ambiguous or Mixed Results

This subsection presents aggregated simulation outputs based on ten-test configurations. These examples provide a more stable overview of model behavior under diverse conditions. In Fig. 3, the total number of tasks was low and the average number of dependencies per test was close to zero. In this case, some tests resulted in positive gains up to 40%, while others showed small negative results down to –3.57%. These fluctuations reflect cases where the presence or absence of even a few dependencies has a strong impact on model efficiency.

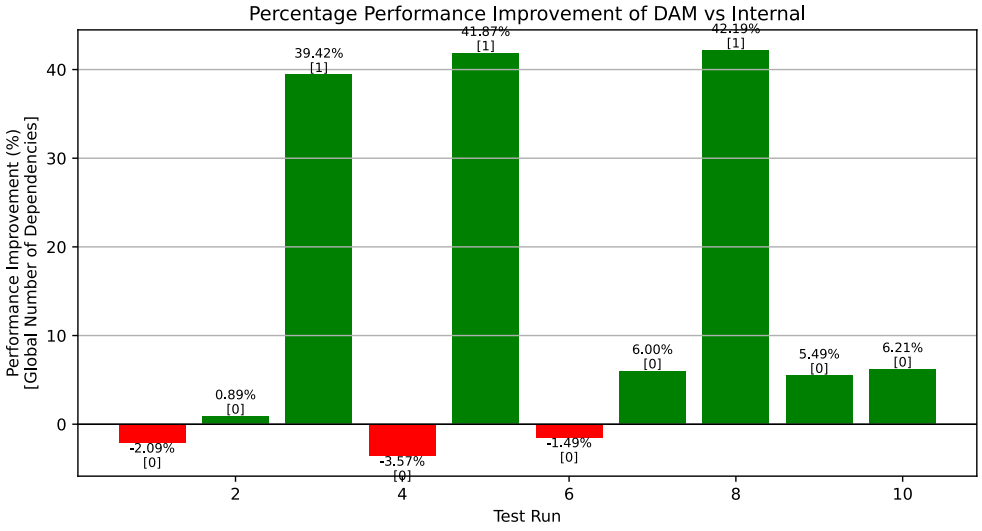

Fig. 3 Results from a ten-test simulation with low task count and minimal dependency density, showing mixed performance outcomes.

Fig. 4 presents a similar case with 23 tasks and short task durations. Nine of the ten test cases exhibited performance improvements, but one case showed a slight negative result due to the presence of only two dependencies. This confirms that in low-dependency scenarios with short execution windows, the DAM model may sometimes introduce minimal overhead.

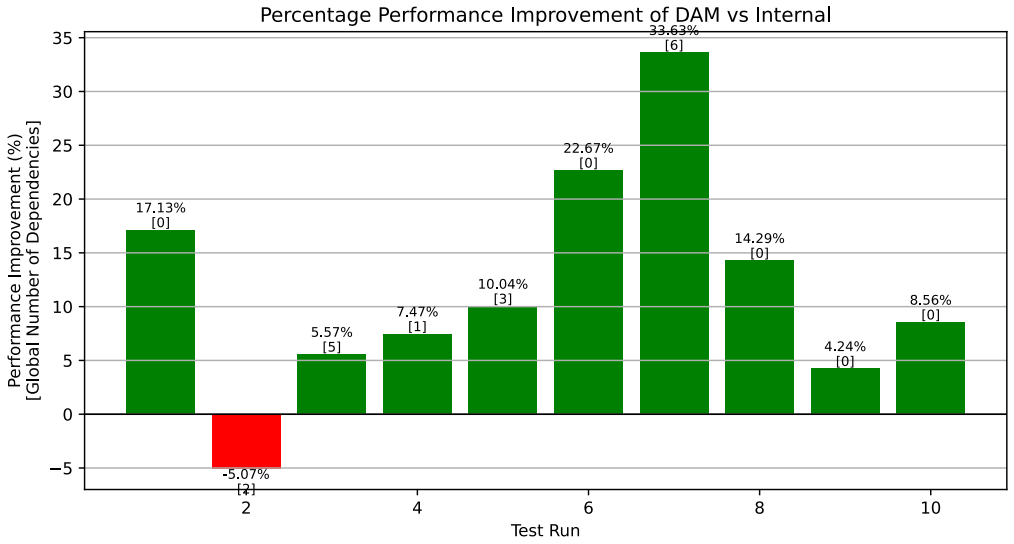

Fig. 4 Ten-test simulation with short task durations and sparse dependencies, mostly yielding positive improvements.

In Fig. 5, a mid-sized configuration with 33 tasks and over 9 average dependencies per test yielded consistently positive results, with all ten bars showing improvements between 11% and 40%. This test confirms the expected advantage of DAM when dependency complexity is moderate and task duration is substantial.

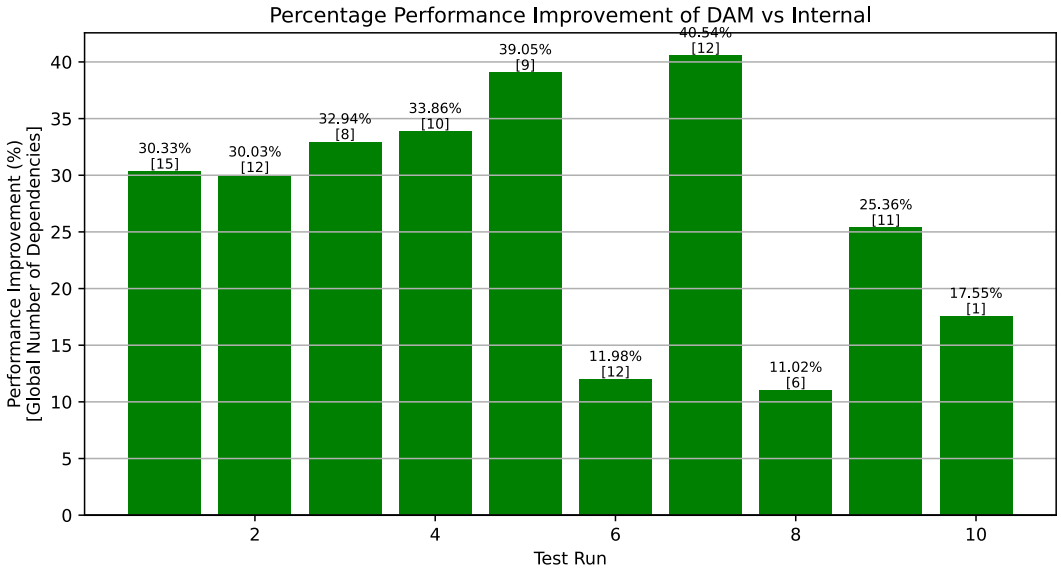

Fig. 5 Results from a ten-test simulation with moderate dependency density, consistently showing performance gains using DAM.

Finally, Fig. 6 demonstrates the performance of the model in a heavy-load configuration involving 159 tasks per test and more than 470 dependencies. In such conditions, the DAM model achieved stable and high gains ranging from 33% to 52.5%, confirming its value in dense, long-duration workloads.

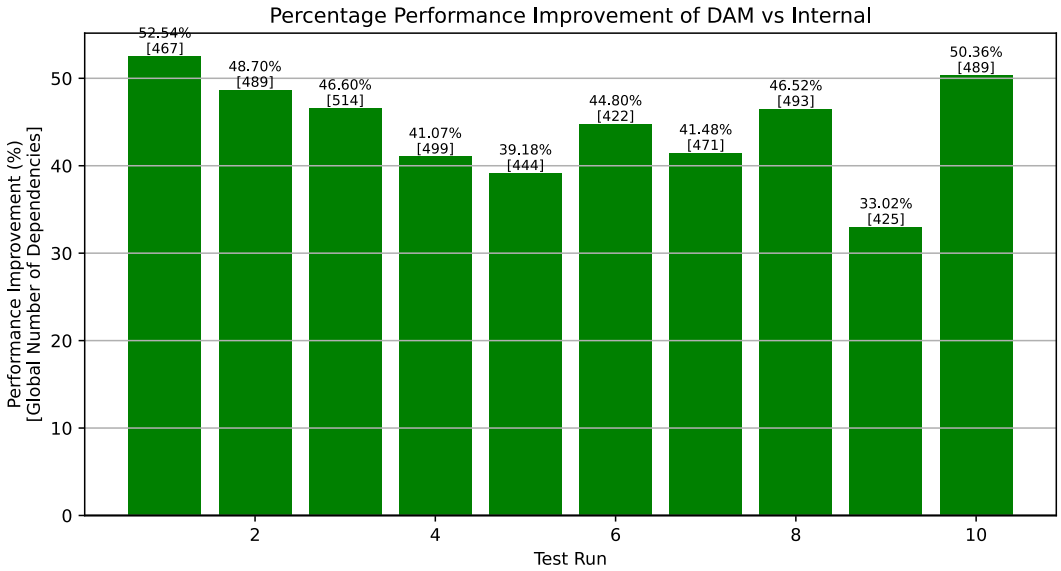

Fig. 6 Ten-test simulation with a high number of tasks and dense dependency graphs, exhibiting stable and significant execution time reductions.

Rare Negative Outlier Case

Among all simulations, one exceptional case produced a strongly negative result. The execution time under DAM was nearly double that of the baseline approach, resulting in a -106.5% difference. Analysis of the simulation parameters revealed that the test involved seven tasks with no dependencies and execution times below 230 milliseconds. The lack of dependencies left the DAM model without optimization opportunities. In this situation, even minimal scheduling overhead introduced by the model contributed to a performance decrease. Such deviations are extremely rare and accounted for only 0.01% of all single-test simulations. They likely reflect transient system-level activity during execution and do not challenge the general reliability of the results.

Such results reinforce the conclusion that DAM is not universally beneficial. Its application must be context-aware, especially in scenarios with few tasks and no dependency structures. However, given the extremely low frequency of such outliers across the full test population, their statistical impact is negligible in the broader performance evaluation.

Discussion

The comprehensive simulation results support the conclusion that the Dependency-Aware Model (DAM) offers measurable advantages in task scheduling efficiency under varied conditions. By deferring execution of tasks until all prerequisite dependencies are satisfied, the DAM significantly reduces unnecessary CPU occupation and improves overall task throughput.

This benefit becomes especially apparent in scenarios involving long task durations and complex dependency graphs, where early task activation in traditional models results in passive waiting and suboptimal core usage. The visualization examples included in this study highlight representative configurations, demonstrating that higher dependency density and moderate to long execution times consistently correlate with greater performance gains.

These results align with prior findings in heterogeneous task scheduling, where dependency-aware strategies such as HEFT (Topcuoglu and Hariri, 2002) and Myrmics (Lyberis et al., 2016) also demonstrated performance gains by avoiding premature task dispatch. Unlike those approaches, however, DAM operates entirely in user space without requiring centralized scheduling, static task graph analysis, or hardware specialization, making it both lightweight and portable across system architectures.

At the same time, the results confirm that the model remains neutral in configurations where its logic is unnecessary. For instance, in cases with no dependencies or minimal task counts, the DAM does not

significantly alter execution characteristics, and any differences are within the margin of system-induced variation. Occasional negative or ambiguous outcomes were observed as well. These are attributed to system-level noise in the live simulation environment, where the execution of concurrent system processes may impact task timing unpredictably. These rare effects underscore the importance of large-scale testing to derive statistically robust conclusions.

Given the increasing use of adaptive and learning-based approaches in scheduling research, future extensions of DAM may incorporate AI or ML techniques to dynamically adjust dependency thresholds or predict optimal dispatch timings. Preliminary frameworks such as diffusion-based reinforcement learning (Cheng et al., 2024) suggest that dependency models can be enhanced through learned policies, which could further improve DAM's efficiency in rapidly changing environments.

Overall, the DAM proved resilient to variability and consistently outperformed the baseline in the overwhelming majority of test cases, confirming its viability as an efficient, scalable addition to conventional scheduling strategies.

Conclusion

A large-scale experimental campaign was conducted to evaluate the Dependency-Aware Model (DAM) for task scheduling in multicore environments. The results, obtained from over 39,000 simulation runs, demonstrate that DAM provides a statistically significant reduction in execution time across a wide range of workload configurations.

The model's strength lies in its design simplicity and runtime efficiency. Without requiring prior global scheduling or centralized analysis, it enables task-level dependency management that dynamically adapts to system conditions. This reactive approach allows DAM to outperform traditional strategies in high-dependency workloads by ensuring that only fully eligible tasks consume CPU resources.

The conclusions drawn from this study are based on diverse test conditions, including varying levels of task concurrency, execution time heterogeneity, and dependency structure. The consistent improvements observed, with averages above 28% in single-test simulations and over 30% in ten-test simulations, provide strong empirical validation for the model's practical benefits.

The presented evidence establishes DAM as an effective, lightweight enhancement to conventional scheduling methods, particularly in systems where dependency patterns influence execution efficiency.

Acknowledgment

The work was financed by the Military University of Technology in Warsaw, Poland as part of the project No. UGB 531-000023-W500-22.

References

- Cheng, X., Mao, Z., Wang, Y. and Wu, W. (2024), 'Dependency-Aware CAV Task Scheduling via Diffusion-Based Reinforcement Learning', arXiv preprint, arXiv:2411.18230.
- IBM. (2024), 'Directed Acyclic Graph (DAG)', IBM Think Blog. [Online], Available at: https://www.ibm.com/think/topics/directed-acyclic-graph (Accessed: June 2025).
- Lyberis, S., Pratikakis, P., Mavroidis, I. and Nikolopoulos, D. S. (2016), 'Myrmics: Scalable, Dependency-aware Task Scheduling on Heterogeneous Manycores', arXiv preprint, arXiv:1606.04282.
- Microsoft. (2021), 'Scheduling'. [Online], Available at: https://learn.microsoft.com/en-us/windows/win32/procthread/scheduling (Accessed: May 2025).
- NVIDIA. (2023), 'What Is CPU Affinity?', NVIDIA Enterprise Support. [Online], Available at: https://enterprise-support.nvidia.com/s/article/what-is-cpu-affinity-x (Accessed: June 2025).
- Pinedo, M. L. (2016), Scheduling: Theory, Algorithms, and Systems, 5th ed., Springer.
- Serafin, P. (2025a), 'A Dependency-Aware Model for Task Execution Optimization in Multicore Systems', in Proceedings of the 45th IBIMA Conference, Córdoba, Spain, June 2025.
- Serafin, P. (2025b), 'Simulation Environment for Evaluating Dependency-Aware Scheduling on Multicore Systems', in Proceedings of the 45th IBIMA Conference, Córdoba, Spain, June 2025.

- The Linux Foundation. (2010), ‘Workqueue – The Linux Kernel Documentation’. [Online], Available at: https://docs.kernel.org/core-api/workqueue.html (Accessed: May 2025).
- Topcuoglu, H. and Hariri, S. (2002), ‘Performance-Effective Task Scheduling Algorithms for Heterogeneous Systems’, IEEE Transactions on Parallel and Distributed Systems, 13(3), pp. 260–274.