An Interference-Aware vCPU Scheduling Framework for Paravirtualized Real-Time Industrial Control Systems

Jessica Müller*, Steven Dietrich*, Michael Massoth†
*Bosch Rexroth AG; †Hochschule Darmstadt
*Lohr a. Main, Germany; †Darmstadt, Germany
email: {jessica.mueller4, steven.dietrich}@boschrexroth.de
email: michael.massoth@h-da.de

***Abstract*—Virtualization enables flexible, software-defined architectures in industrial automation, but introduces new challenges, such as resource contention and unpredictable latencies. This paper presents an interference-aware scheduling approach based on paravirtualized VM profiling. By dynamically classifying virtual CPUs (vCPUs) considering dominant I/O usage and preventing simultaneous execution of tasks with overlapping I/O demands, the method improves determinism and responsiveness. Simulated under realistic workloads, the scheduler significantly reduces utilization peaks, eliminates overload conditions, and stabilizes workload distribution. These results demonstrate the potential of the approach to enhance the predictability and efficiency of virtualized industrial control systems.**

Keywords-virtualization; industrial automation; real-time systems; VM scheduling; interference mitigation; vCPU classification; hypervisor scheduling; industry 4.0.

I. Introduction

The ongoing shift towards flexible production systems is a defining feature of Industry 4.0 (I4.0), where adaptability, modularity, and responsiveness are critical design goals [1]. To support this transformation, industrial control systems are deployed increasingly as Virtual Machines (VMs) hosted on centralized hypervisor platforms. This virtualization enables software- defined control, efficient resource utilization, and dynamic system reconfiguration without modifying physical hardware. However, the consolidation of time-sensitive applications onto shared virtualized infrastructures introduces new challenges. In particular, resource contention at the I/O or CPU level can lead to unintended temporal interference between virtual machines [2]. Such interference may impact the timing behavior of control applications and thus affect the predictability and reliability required in industrial automation environments. We contribute a new interference-aware scheduling approach that explicitly accounts for cross-VM interference at scheduling time rather than relying on conservative worst-case scheduling.

The remainder of this paper is organized as follows. Section II introduces Multi-Virtual-Machine (Multi-VM) environments, outlining industrial use cases, the state of VM scheduling in practice, and the shortcomings that motivate our work. Section III details the proposed interference-aware scheduling approach based on paravirtualized VM profiling, covering its design rationale, architectural components, and integration workflow. Section IV describes the experimental testbed and simulation scenarios used to evaluate the scheduler under realistic industrial conditions. Section V presents and interprets the results, with a focus on latency, interference mitigation, and their implications for industrial automation. Section VI concludes the paper and sketches avenues for future research.

II. Multi-Virtual-Machine Environments

To understand the challenges and design requirements of interference-aware scheduling, it is first necessary to analyze how industrial multi-VM environments are structured, how scheduling is currently implemented, and where existing limitations arise.

A. Industrial Use Cases and Requirements for Multi-VM Systems

In the context of I4.0, industrial control systems are deployed increasingly as VMs hosted on centralized computing platforms. Instead of being distributed across multiple embedded devices, control logic, HMIs, and edge analytics are consolidated into a single physical system running multiple VMs concurrently [3]. This architectural shift enables streamlined system integration, centralized updates, and flexible resource allocation in modular and reconfigurable production environments. To ensure strong isolation and low overhead, these virtualized control systems typically are managed by a Type 1 hypervisor [4].

A central requirement for such deployments is deterministic behavior for time-critical control loops. In particular, short and stable control cycle times – typically in the range of 1–10 ms – are essential for guaranteeing timely responses to sensor inputs and actuator commands [5][6]. Any temporal deviations caused by VM scheduling delays or resource contention at the hypervisor layer must therefore be minimized to maintain the overall system's functional integrity and reliability. An overview of this architecture is illustrated in Figure 1.

Within a Systems-of-Systems (SoS) setup, a hybrid control architecture is feasible: autonomous local real-time loops handle fast dynamics, while a lightweight supervisory layer coordinates setpoints and resource constraints across VMs.

B. Technical Overview: Current VM Scheduling in Industry

In modern industrial environments, Type-1 hypervisors play a critical role in consolidating control systems, HMIs, and edge computing workloads into virtualized infrastructures. These bare-metal hypervisors, such as VMware ESXi, Microsoft Hyper-V, or open-source solutions like Xen and KVM (with

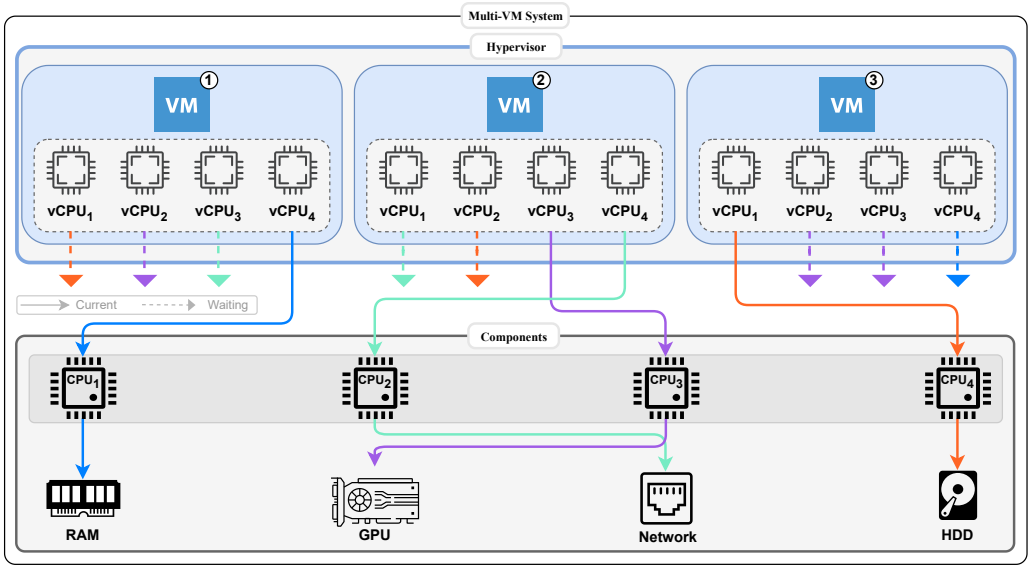

Figure 1. Multi-VM architecture with shared resources under a Type 1 hypervisor.

real-time extensions), operate directly on the host hardware and manage the allocation of physical CPU resources to VMs.

The core responsibility of the hypervisor's scheduler is to map vCPUs of the guest VMs to physical CPUs (pCPUs) on the host system [7]. Current scheduling strategies predominantly rely on variants of fair-share, priority-based, or real-time aware algorithms:

- **Fair-Share Schedulers**, such as the default Credit Scheduler in Xen or the Completely Fair Scheduler in KVM, aim to distribute CPU time evenly across VMs, based on configurable weights or credits. These are designed for general-purpose workloads and maximize overall utilization [7].
- **Priority-Based Scheduling** is commonly used to assign static or dynamic priorities to VMs or individual vCPUs. High-priority tasks receive preferential CPU access, which is suitable for scenarios with mixed workloads where certain VMs are more critical than others [8].
- **Real-Time Extensions** are offered in hypervisors, such as VMware ESXi with *Latency Sensitivity Mode* or KVM with the *PREEMPT_RT* patch. These mechanisms allow for stricter control over scheduling behavior, including CPU pinning (affinity), isolation from non-real-time workloads, and reservation of exclusive resources [9].

In the context of industrial automation, schedulers often leverage CPU affinity and isolation techniques to bind critical control VMs to dedicated cores, thereby reducing variability introduced by co-located workloads. Additionally, reservation mechanisms allow guaranteeing a minimum share of CPU time to latency-sensitive VMs [7].

Hypervisors may also employ I/O-aware scheduling policies, attempting to balance compute and I/O workloads across VMs. However, in standard configurations, CPU and I/O scheduling remain decoupled, which can introduce indirect effects on determinism – especially under high system load [10].

Overall, current hypervisor scheduling mechanisms are designed to ensure fair, efficient, and scalable CPU usage across virtual machines. While real-time features exist, their practical integration into industrial VM setups often requires careful tuning and architectural planning.

C. Identified Shortcomings and Interference Issues

Despite the availability of real-time extensions and resource isolation features, current hypervisor scheduling mechanisms remain susceptible to temporal interference – particularly in I/O-intensive scenarios [4]. In virtualized industrial environments, where deterministic control loops must operate within strict cycle times of 1–10 ms, even minor deviations in execution timing can compromise system integrity.

A key source of such deviation lies in the interaction between vCPU scheduling and I/O operations. Although CPU time may be reserved or pinned for a control VM, I/O subsystems (e.g., disk, network, or fieldbus interfaces) are typically shared among multiple VMs and rely on asynchronous handling through interrupt-driven mechanisms or hypervisor-level emulation [11][7]. These operations introduce latency that is neither fully visible nor fully controllable by the guest operating system, leading to non-deterministic delays in input acquisition or actuator response.

Moreover, hypervisors often decouple I/O scheduling from CPU scheduling, which makes it difficult to coordinate compute and communication timing holistically [12]. For instance, when multiple VMs compete for access to shared I/O resources – such as a virtual NIC or storage backend – context switches, interrupt storms, or emulation delays may disrupt the timing guarantees required by control applications [11][12]. These effects further

are amplified under system load, where best-effort workloads or background processes inadvertently interfere with time-critical VMs, despite configured priorities or affinity.

As a result, cycle-time violations and jitter become increasingly probable in consolidated setups, particularly when industrial controllers, HMIs, and monitoring tools coexist on the same host [12]. Without holistic temporal coordination across all relevant subsystems – including CPU, memory, and I/O paths – the promise of determinism in virtualized control architectures remains difficult to fulfill under real-world conditions.

III. Interference-Aware Scheduling via Paravirtualized VM Profiling

To address the timing deviations and interference issues identified in multi-VM environments, a novel scheduling approach is introduced that explicitly considers the I/O behavior of virtual machines and their interactions at runtime.

A. Design Motivation and Objectives

In industrial environments increasingly shaped by digitalization and I4.0, conventional scheduling mechanisms are reaching their limits. These mechanisms were not typically designed to meet the specific demands of virtualized control systems [13]. A major issue in this context is I/O interference, which leads to unpredictable latencies and violations of strict cycle times. This undermines the reliability of industrial control applications, where, for instance, a guaranteed 1 ms cycle time is critical – but in practice, often only a worst-case latency of around 100 ms can be assured [6].

The aim of the newly conceived scheduling approach is therefore to proactively mitigate such interference through deliberate planning. This enables more reliable system availability, as the state of the I/Os is known at all times. It not only facilitates dynamic load balancing but also allows for foresighted resource allocation for potential emergency scenarios, such as interrupt-driven, I/O-intensive operations. In addition to the classical objective of optimal process and vCPU distribution, this approach strengthens overall system stability under real-time conditions.

B. Architectural Overview of the Profiling Scheduler

The proposed scheduler architecture consists of two tightly integrated components: A classification unit and a scheduling unit. As soon as a vCPU becomes eligible for execution, it is passed to the classification unit, which determines the dominant I/O resource it is expected to interact with. This classification is based on a lightweight analysis of the task characteristics within the vCPU and assigns it to an I/O category, such as GPU-bound, RAM-bound, cache-bound, network-bound, or disk-bound. The process is performed immediately before each scheduling decision, ensuring that classification always reflects the current system context without relying on historical profiling data. An example for the classification is shown in Figure 2.

Once classified, the vCPU is passed to the scheduling unit, which maps it to an appropriate pCPU core. The central policy

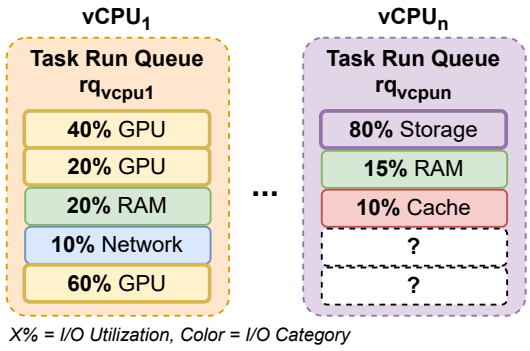

Figure 2. Classification of vCPUs based on their run queue.

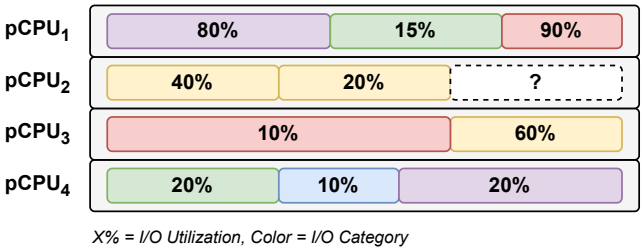

Figure 3. Scheduling Timelines.

enforced by the scheduler is to avoid concurrent execution of vCPUs from the same I/O category on different physical cores. This interference-aware constraint ensures that no two vCPUs with similar I/O access patterns simultaneously contend for the same shared hardware resource. By isolating I/O categories across cores within a given time window, the system prevents unpredictable latency spikes caused by overlapping access to memory buses, storage devices, or network interfaces. An example for the scheduling is shown in Figure 3.

The entire process is executed synchronously and on-demand: Every time a vCPU enters the ready queue, the classification and scheduling decisions are computed in a single step. This approach maintains high responsiveness while avoiding background profiling overhead.

C. Integration into Virtualized Environments

Practical deployment of the interference-aware scheduler requires integration at the hypervisor's kernel scheduling layer. On Linux-based hypervisors, such as KVM, this can be realized via the `sched_ext` framework, which permits external schedulers to be loaded without modifying the core kernel [14]. Hypervisors lacking comparable extensibility – such as Xen, VMware ESXi, or Microsoft Hyper-V – necessitate direct modification of the scheduler code, although the required changes remain confined to the scheduling path and do not affect device drivers or memory management [15].

The logic supports two operating modes. First, it can function as a standalone scheduler that assigns vCPUs solely on the basis of I/O classification. Second, it can act as a refinement stage atop an existing real-time scheduler (e.g., Earliest Deadline

First), reordering the run queue to prevent concurrent execution of vCPUs with matching I/O profiles and thus minimizing interference while preserving deadline guarantees.

Effective classification depends on visibility into each VM's internal run queue. To provide this information, every guest transmits a compact summary of its runnable tasks to the hypervisor via a dedicated hypercall or paravirtual channel. Implemented as a small guest-kernel module, this mechanism imposes no changes on user-space applications and can be shipped alongside standard paravirtualization drivers [16].

Because the classification and mapping occur only when a vCPU becomes ready, the additional computational burden is negligible, making the approach suitable for resource-constrained industrial hosts where deterministic timing and minimal overhead are paramount.

IV. Experimental Setup and Simulation

To evaluate the effectiveness and timing behavior of the proposed interference-aware scheduling concept, a controlled simulation environment was developed that allows systematic analysis under reproducible conditions.

A. Simulation Environment

The simulation was implemented in a Python-based Jupyter Notebook environment. The scheduler was developed as a custom computation that calculates the run queue assignments for all virtual CPUs based on predefined workload scenarios. These run queues represent the scheduling decisions over time and were subsequently used as input for a discrete-event simulation implemented with the SimPy framework. In this setup, SimPy emulates the execution of the virtual CPUs according to the generated schedule and enables measurement of timing-related metrics, such as waiting times and utilization. All experiments were conducted offline without deploying an actual hypervisor or virtual machines, allowing controlled and repeatable evaluation of the scheduling logic under synthetic conditions.

B. Scenarios and Assumptions

The simulation comprised a set of predefined scenarios with varying workload intensities, resource utilization patterns, and virtual machine configurations. For each scenario, synthetic datasets were generated to represent categorized vCPUs, including their expected resource demands and arrival times. It was assumed that all vCPUs were pre-classified and that the system operated under ideal conditions without allocation delays or interference between components. Resources were modeled deterministically, with fixed maximum capacities and no variability due to physical hardware behavior or contention effects. The main objective of this simulation was to validate the feasibility of the proposed scheduling approach and to provide initial performance insights under controlled conditions. Due to these simplifications, results should be interpreted as indicative rather than fully representative of complex real-world environments.

V. Results and Interpretation

The subsequent analysis summarizes the outcomes of the conducted simulation experiments, highlighting key behavioral differences between the baseline and the optimized scheduling strategies.

A. Scenario Overview and Scheduling Behavior

Figure 4 illustrates the execution timeline of the baseline scheduling strategy, where vCPUs are assigned to the shortest available run queue without considering their expected runtimes or I/O dependencies. In this scenario, all physical CPU cores initially process tasks in a balanced manner, resulting in nearly synchronous task completions across the cores. However, during execution, a pronounced idle period occurs on a single CPU core that must wait for a shared I/O operation to complete before further processing can continue. This blocking leads to an extended idle phase on that core and increases the total processing time for the workload.

Additionally, Figure 5 illustrates the utilization of the I/O components observed during the simulation of the same baseline execution. The diagram highlights a specific time interval between 13 and 16 time units, where the GPU utilization temporarily exceeds 100% due to concurrent access from multiple tasks. This overcommitment results in contention for the shared GPU resource, causing blocking delays that propagate back to the scheduling timeline and extend the overall processing time. The example demonstrates that purely queue-length-based scheduling not only produces unpredictable idle periods but also leads to excessive I/O load peaks that further degrade system performance and determinism.

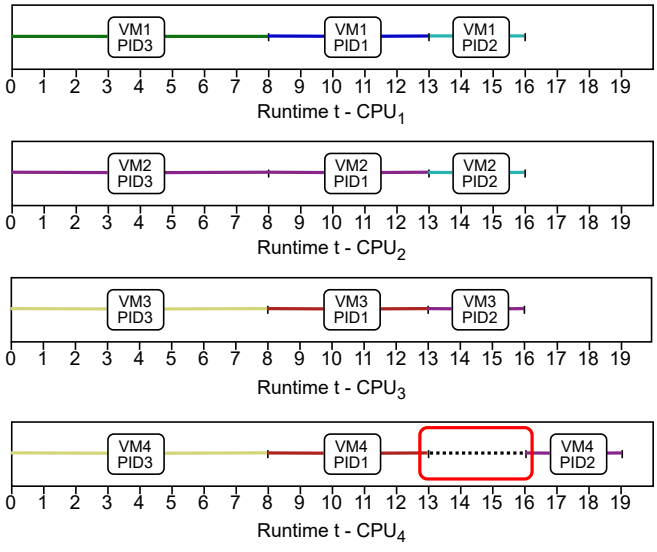

Figure 4. Scheduling Timelines without Optimization.

Figure 6 shows the execution timeline obtained with the proposed scheduling approach, where overlapping execution of equally categorized tasks is explicitly avoided. In this configuration, the scheduler assigns vCPUs so that tasks of

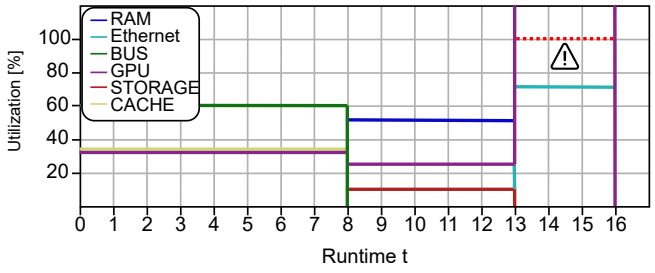

Figure 5. I/O Component Utilization after Simulation without Optimization.

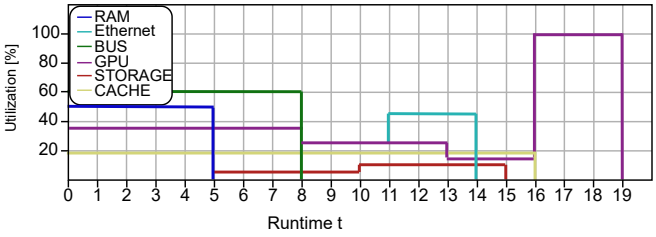

Figure 7. I/O Component Utilization after Simulation with 0% Overlapping Optimization.

the same category do not run concurrently on different cores, thereby preventing the I/O blocking effects observed in the baseline scenario. As a result, no idle periods occur during execution, and the overall processing time is reduced. However, this strict separation also leads to a less uniform workload distribution across cores, as visible in the timeline. While this setup demonstrates the feasibility of deterministic, non-overlapping scheduling, the approach can be relaxed to allow controlled overlap between task categories, providing additional flexibility to balance I/O and CPU utilization more evenly if required.

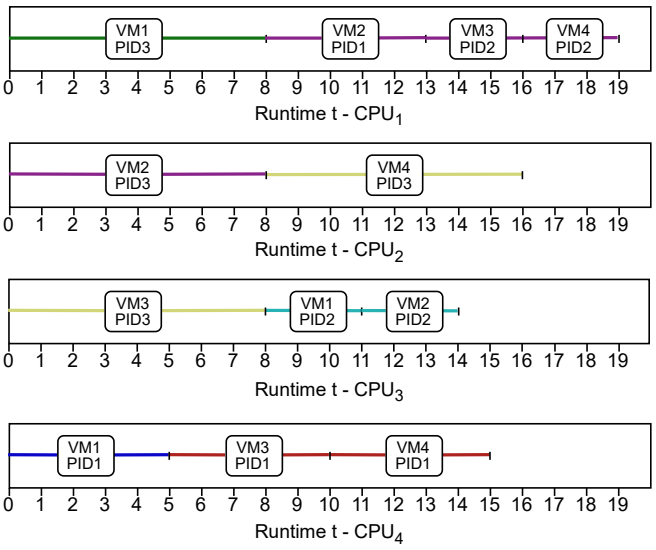

Figure 6. Scheduling Timelines with 0% Overlapping Optimization.

Additionally, Figure 7 illustrates the I/O utilization observed during the simulation of the optimized scheduling scenario. In contrast to the baseline case, no overcommitment beyond 100% occurs, confirming that the separation of I/O categories effectively reduces contention and stabilizes resource usage over time.

B. Quantitative Metrics and Performance Comparison

To objectively evaluate the effectiveness of the proposed vCPU scheduling optimization, a set of quantitative utilization and CPU load metrics was collected before and after the optimization. The components were categorized as RAM, ETHERNET, BUS, GPU, STORAGE, and CACHE. The results demonstrate several significant improvements.

First, the optimization led to a more constant utilization of critical I/O components over time. For example, the average utilization of ETHERNET decreased from 12.35% to 10.5%, while CACHE utilization was reduced from 16.0% to 13.6%. This more even distribution of load helps prevent unpredictable fluctuations and enables better planning of system resources.

Second, the optimization effectively reduced utilization peaks. The maximum utilization of ETHERNET dropped from 70% to 45%, representing a reduction of more than one third, while CACHE gets its maximum utilization cut by half, from 34% to 17%. For STORAGE, the maximum utilization was also reduced by approximately 33%. By lowering these peaks, the system achieves a smoother and more predictable workload profile, which is particularly important for time-sensitive applications.

Third, the optimization ensured that no component exceeded 100% utilization at any time. Before the optimization, GPU occasionally reached utilization peaks of up to 113%, indicating that tasks temporarily demanded more I/O capacity than was available, which resulted in waiting times and delays. After the optimization, all components remained consistently below 100% utilization, preventing overload conditions and eliminating unnecessary queuing of I/O operations.

In addition to improvements in I/O utilization, the distribution of CPU load across cores became more balanced. While the CPU loads were initially nearly identical across cores, but after optimization, the loads were more differentiated. Although this led to slightly differing completion times for the individual CPU cores in this synthetic example, this effect is not critical in real-world applications. In practical scenarios, there is a continuous inflow of new tasks, so the timing of core idle phases becomes irrelevant. The system benefits far more from the improved predictability and absence of overload situations than it is impacted by minor variations in per-core runtime.

Overall, the results clearly show that the optimization keeps component utilization more constant, reduces peak loads, prevents overload conditions, and distributes CPU workloads more evenly. This combination significantly increases the stability and responsiveness of the system without introducing adverse side effects for unaffected components, such as RAM or BUS.

C. Implications for Industrial Deployments

The presented optimization is highly relevant for industrial control environments, where virtualized systems must deliver consistent performance and comply with strict timing requirements.

By ensuring that critical I/O resources, such as Ethernet interfaces, storage subsystems, and GPU accelerators remain reliably below full utilization, the approach effectively prevents situations where tasks are forced to wait due to resource contention. This directly supports predictable cycle times, which are essential for machine control and safety-related processes.

The increased stability of resource usage also simplifies planning and verification against industrial standards, reducing the need for oversized hardware reserves and enabling more efficient system designs.

In real-world deployments, minor differences in CPU completion times, as observed in synthetic tests, have no practical impact, since industrial workloads are typically characterized by continuous streams of tasks. Under these conditions, the advantages of smoother utilization profiles and the elimination of overload situations clearly outweigh any variations in per-core timing, resulting in higher system availability and more robust operation under changing load conditions.

Moreover, the more balanced distribution of CPU load contributes to improved thermal behavior and can help extend the lifespan of hardware components, which is an important factor in embedded and industrial-grade platforms. Overall, the optimization provides a practical means of enhancing determinism, efficiency, and resilience in virtualized industrial environments.

VI. Conclusion and Future Work

This paper presented an interference-aware scheduling approach based on paravirtualized VM profiling, designed to improve the determinism and predictability of virtualized industrial control systems. By classifying vCPUs according to their dominant I/O resource usage and preventing the concurrent execution of equally categorized tasks, the proposed method effectively reduced utilization peaks and eliminated overload conditions that often lead to unpredictable latencies.

Experimental evaluation under synthetic conditions demonstrated that the optimization can maintain consistently lower maximum utilization across critical components, such as Ethernet, storage, and GPU, while achieving a smoother distribution of workload over time. Although slight variations in per-core completion times were observed, these effects are negligible in real-world industrial environments where continuous task streams are common.

Future work will focus on extending the approach beyond offline simulation and integrating the scheduler into production-grade hypervisors to validate its effectiveness under real workloads and mixed I/O patterns. Additionally, further research will explore adaptive scheduling strategies that dynamically adjust the degree of task separation based on system load and application criticality. Investigating the impact of the approach on power consumption, thermal behavior, and long-term hardware reliability in embedded industrial platforms also represents an important direction for future studies.

References

[1] R. Stark, *Virtual Product Creation in Industry*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2022, ISBN: 978-3-662-64299-3. DOI: 10.1007/978-3-662-64301-3.

[2] V. Meyer, U. L. Ludwig, M. G. Xavier, D. F. Kirchoff, and C. A. F. De Rose, "Towards interference-aware dynamic scheduling in virtualized environments", in *Job Scheduling Strategies for Parallel Processing*, D. Klusáček, W. Cirne, and N. Desai, Eds., Cham: Springer International Publishing, 2020, pp. 1–24.

[3] C. Serôdio, P. Mestre, J. Cabral, M. Gomes, and F. Branco, "Software and architecture orchestration for process control in industry 4.0 enabled by cyber-physical systems technologies", *Applied Sciences*, vol. 14, no. 5, 2024, ISSN: 2076-3417. DOI: 10.3390/app14052160.

[4] J. Müller, M. Giani, D. Deubert, and M. Massoth, "Virtualization in industrial production - a survey focusing on virtual and virtualized industrial controls", in *2024 IEEE 29th International Conference on Emerging Technologies and Factory Automation (ETFA)*, 2024, pp. 1–7. DOI: 10.1109/ETFA61755.2024.10710668.

[5] M. Gundall, C. Glas, and H. D. Schotten, "Feasibility study on virtual process controllers as basis for future industrial automation systems", in *2021 22nd IEEE International Conference on Industrial Technology (ICIT)*, vol. 1, 2021, pp. 1080–1087. DOI: 10.1109/ICIT46573.2021.9453651.

[6] S. Dietrich, G. May, O. Wetter, H. Heeren, and G. Fohler, "Performance indicators and use case analysis for wireless networks in factory automation", in *2017 22nd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, 2017, pp. 1–8. DOI: 10.1109/ETFA.2017.8247605.

[7] S. Xi, J. Wilson, C. Lu, and C. Gill, "Rt-xen: Towards real-time hypervisor scheduling in xen", in *Proceedings of the Ninth ACM International Conference on Embedded Software*, ser. EMSOFT '11, Taipei, Taiwan: Association for Computing Machinery, 2011, pp. 39–48, ISBN: 9781450307147. DOI: 10.1145/2038642.2038651.

[8] V. Roy, "A context-aware internet of things (iot) founded approach to scheming an operative priority-based scheduling algorithms", *Journal of Cybersecurity and Information Management*, vol. 13, pp. 28–35, Jan. 2024. DOI: 10.54216/JCIM.130103.

[9] M. A. Altahat, K. Mhaidat, and O. Al-Khaleel, "Quantitative analysis of hypervisor efficiency and energy consumption in heterogeneous multi-vm environments with varied server workloads", *Simulation Modelling Practice and Theory*, vol. 141, p. 103 102, 2025, ISSN: 1569-190X. DOI: https://doi.org/10.1016/j.simpat.2025.103102.

[10] E. Jeannot, G. Pallez, and N. Vidal, "Io-aware job-scheduling: Exploiting the impacts of workload characterizations to select the mapping strategy", *The International Journal of High Performance Computing Applications*, vol. 37, no. 3-4, pp. 213–228, 2023. DOI: 10.1177/10943420231175854. eprint: https://doi.org/10.1177/10943420231175854.

[11] J. Peixoto, J. Martins, D. Cerdeira, and S. Pinto, "Birtio: Virtio for real-time network interface sharing on the bao hypervisor", *IEEE Access*, vol. 12, pp. 185 434–185 447, 2024. DOI: 10.1109/ACCESS.2024.3512777.

[12] N. Borgioli *et al.*, "An i/o virtualization framework with i/o-related memory contention control for real-time systems", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 11, pp. 4469–4480, 2022. DOI: 10.1109/TCAD.2022.3202434.

[13] U. Bilgen, E. Y. Tat, and M. ErelÖzçevik, "A novel digital twin enabled weighted task optimization framework for software defined data centers in industry 4.0", in *2025 IEEE Wireless Communications and Networking Conference (WCNC)*, 2025, pp. 1–6. DOI: 10.1109/WCNC61545.2025.10978632.

[14] T. L. K. Community, *Sched_ext – bpf-based scheduler class*, https://docs.kernel.org/scheduler/sched-ext.html, Accessed: 2025-09-12, 2024.

[15] N. Kraljevic, B. Djordjevic, and V. Timcenko, "File system performance comparison in full hardware virtualization with esxi, kvm, hyper-v and xen hypervisors", *Advances in Electrical and Computer Engineering*, vol. 21, no. 1, pp. 7–14, 2021. DOI: 10.4316/AECE.2021.01002.

[16] K. T. Raghavendra, S. Vaddagiri, N. Dadhania, and J. Fitzhardinge, "Paravirtualization for scalable kernel-based virtual machine (kvm)", in *2012 IEEE International Conference on Cloud Computing in Emerging Markets (CCEM)*, 2012, pp. 1–5. DOI: 10.1109/CCEM.2012.6354619.