

Performance Prediction for Microservice-Based Cyber-Physical Systems: A Cross-Domain Literature Review

Jan Przystał
j.p.przystal@student.vu.nl
University of Amsterdam
The Netherlands
Vrije Universiteit Amsterdam
The Netherlands

ABSTRACT

Cyber-Physical Systems (CPS) are increasingly complex, integrating numerous software components and heterogeneous hardware platforms. These systems often impose strict timing requirements, which means that the system's performance is of high importance. However, predicting the performance of a CPS during the design phase is a major challenge, particularly due to resource contention—where shared system resources such as memory and caches become bottlenecks when multiple tasks compete for them. Traditional performance evaluation methods, such as simulations or hardware prototyping, are often too time-consuming to be viable for early-stage Design Space Exploration (DSE), where rapid and scalable evaluation is essential.

This study investigates current methods for predicting performance degradation in CPS, with a focus on microservice-based architectures. It examines techniques such as sensitivity and contentions profiling, vector-based interference prediction, and dynamic scheduling strategies. While these approaches offer fast performance estimation, they often face limitations in scalability, generality, and accuracy—especially when applied to multi-component CPS with diverse workloads. The analysis reveals that no existing method fully satisfies the need for a quick, accurate, and scalable performance prediction framework in large CPS design spaces. The study concludes that further research is needed to develop an integrated, efficient, and analytically sound solution that supports early DSE of complex CPS, particularly those that leverage microservice architectures.

1 INTRODUCTION

Cyber-Physical Systems (CPS) are an important part of the technology sector, especially in fields such as health industries, industrial automation, robotics [11]. They are composed of a computer system and physical processes. Many of these systems are very complex, with hundreds of software and hardware components, heterogeneous processing elements, and physical parts. The software often needs to react to or manage the physical processes. Because of this, usually the "cyber" part of a CPS needs to be predictable and reliable. This includes both the computer hardware and software. Especially important are the often present timing requirements. Managing the complexity during the design process while adhering to these constraints is a big challenge.

One major problem in designing a complex CPS with many software components is that resource sharing becomes nearly unavoidable. This can lead to resource contention, which is a situation where multiple programs compete for system resources, which can

lead to performance degradation. This of course has negative effects and can disrupt the whole system. Because of this, it is crucial to evaluate the performance of a system during the design phase.

A way to predict resource contention and its severity would be very useful in performance estimation. When designing a system with many different software components and multiple hardware platforms, there is a multitude of ways in which they can be deployed together. A quick method of evaluation is necessary to test the viability of all configurations in a reasonable time. Traditional performance evaluation methods often rely on high-level simulations [11]. Although this approach provides high fidelity, it is computationally expensive and time consuming, making them impractical for the early stages of Design Space Exploration (DSE).

As Cyber-Physical Systems grow in complexity, it becomes difficult to develop new solutions and modernize old ones. Some CPS developers have opted to migrate to a microservice architecture [9]. This solution is popular and tested in the field of cloud computing, but it is not thoroughly explored in the context of CPS. Projects are using microservices in CPS for the ease of deployment and maintenance while maintaining the required level of security and performance [10].

This literature study explores the topic of resource contention and the current methods of performance prediction. It also goes over Cyber-Physical Systems and their design process, discussing the specific challenge related to performance prediction.

The remainder of this literature study is structured as follows: Section 2 provides the background knowledge required to understand the presented topics. Section 3 provides an overview of the work related to performance prediction methods and CPS design, Section 4 discusses the related work and how relevant and useful it is in solving the presented problem, and Section 5 summarizes the findings of this study.

2 BACKGROUND

This section introduces and explains important terms and concepts that will be discussed throughout the rest of the study.

2.1 Cyber-Physical Systems

Cyber-Physical Systems are integrations of computation with physical processes. Embedded computers and networks monitor and control the physical processes, usually with feedback loops where physical processes affect computations and vice versa [13]. Cyber-Physical Systems are very diverse. They include: autonomous automotive systems, medical monitoring equipment, flight control systems, robots, etc.

There are many domain-specific challenges in designing a CPS, like integrating the software with the various mechanical parts or sensors. For this research, the most important design consideration is the timing requirements. Many CPS are hard real-time systems, which require predictable and reliable performance to guarantee the timing requirements.

A distributed Cyber-Physical System (dCPS) combines multiple sub-systems with physical processes to achieve greater efficiency, reliability, and functionality [20].

2.2 System Model

Cyber-Physical Systems have two distinct parts. The defining characteristic of a CPS is the integration of physical processes. The "cyber" part is the software and the compute hardware on which it runs, which can be a number of (heterogeneous) compute nodes, or hosts, connected by a network or other IO. The software is comprised of interdependent tasks which can be implemented using different technologies, e.g. microservices. This study does not explore the physical processes but rather the performance of these software tasks and the computer systems that are parts of CPS. The focus is specifically on microservices, explained in Section 2.5, but most of the research also applies to any other application, process, or task. Therefore, these terms are used interchangeably.

2.3 Resource Contention

Any computing system has limited hardware resources, such as CPU, memory, storage, and I/O bandwidth. When the system is under heavy load, multiple tasks compete for these shared resources. This competition is called resource contention. As contention increases, tasks may experience delays and reduced throughput, leading to performance degradation.

Contention can occur whenever system resources are shared among multiple applications. For example, two tasks do not execute simultaneously on the same processor core (assuming no simultaneous multithreading), so they do not compete for execution units. Private caches are not shared so isolated workloads will not interfere with the private caches of other cores. However, if multiple microservices are scheduled sequentially on the same core, they can still interfere indirectly by evicting the cache lines belonging to the previously running service, leading to additional cache misses after a context switch.

Other resources like the main memory, Network Interface Cards (NIC), or storage are shared, and therefore contention between different applications may occur. Shared caches, like the Last Level Cache (LLC), also become a problem. Because almost all main memory requests need to travel through the LLC, contention is likely. The LLC has limited capacity and under heavy load a program can easily evict another's cache lines, which, unless the line will no longer be used, causes performance degradation.

2.4 Design Space Exploration

Design Space Exploration (DSE) is the search and evaluation of possible design solutions in order to find the ones that best satisfy defined design objectives. Although DSE can be applied in many domains, this research discusses it in the context of Cyber-Physical Systems. There can be many design objectives for DSE depending on

the system. Common ones include: energy consumption, thermals, reliability, cost, performance [11]. For the purpose of the study, the design space is considered a set of all possible mappings of the software components to the hardware components.

DSE can be considered an optimization problem. Given a design space of configurations, what is the configuration that maximizes or minimizes the objective function, subject to constraints. The constraints cannot be changed, they must be respected as they define what are valid solutions. The problem lies in the fact that as the design space grows, so does the time cost of exploring it because more design points need to be evaluated. The number of configurations can get out of control quickly because it grows rapidly with every added software or hardware component. In a complex dCPS comprised of many components, the design space can be huge - potentially hundreds of thousands or even more possible configurations [11, 20, 25]. This creates an obvious problem - exploring a massive configuration space requires a quick evaluation method to be feasible.

2.5 Microservices

Microservice (MS) architecture is a type of Service-Oriented Architecture [12]. The idea is to divide an application into smaller, independent services that communicate with each other through message passing. Each Microservice has a single well-defined functionality it fulfills.

Microservices have many benefits. They are loosely coupled, which means that the services themselves are independent. This improves maintainability by minimizing the costs of modifying services, fixing errors, or adding new functionality [8]. It also means, they can be tested and deployed separately. Because microservices are small they are less fault prone, and even if they experience failure, they do not necessarily bring down the entire system. On the downside, performance can be negatively impacted by the increased communication over a network, because network latency is much greater than that of memory.

While Microservices are widely adopted in cloud-native applications, they are also being used in Cyber-Physical Systems. Removing a single point of failure that can cause a system to stop working is very useful in systems that require high availability [17]. Microservices can also be helpful for distributed Cyber-Physical Systems because they are meant to be easy to deploy and maintain. In short, there are many advantages of using Microservices in (distributed) Cyber-Physical Systems, but the most important are: high availability, scalability, and maintainability.

3 RELATED WORK

This section presents the related work. It has been split into sections focusing on specific topics that cover: design of CPS, resource contention, and performance prediction in various fields.

3.1 Design Challenges of Cyber-Physical Systems

CPS and especially dCPS keep expanding in complexity and designing them is becoming especially difficult because they can have many objectives, while having strict constraints (power, reliability,

etc.). Although research in other domains, such as cloud computing, may provide relevant and beneficial insights, its applicability cannot be assumed given the unique properties of CPS.

The Design Space Exploration process can be implemented in many ways depending on the specific use-case. Although approaches to DSE vary, they generally follow these 4 steps [11]:

- (1) Description and creation of **models** based on existing systems.
- (2) Creation of Design Space.
- (3) Exploration - Analysis and evaluation of configurations in the Design Space.
- (4) Results - Presentation or processing of the results.

There is a distinct problem in the Exploration step. As the number of possible configurations grows, efficient and scalable ways of exploration are required. This includes the search, pruning, and evaluation of configurations. The evaluation of a single design point takes longer for dCPS compared to classical DSE due to their complexity [11].

CompDSE is a methodology for designing complex dCPS [20]. It collects specific system data from the OS during runtime in the form of event traces. The traces are collected over time at specifically chosen locations in the code, so that important events, such as the sending and receiving of messages between processes, are captured. From this it automatically derives models of the software, hardware platform, and the mapping between them. Afterwards, changes are made to the model to explore a certain configuration. Then a high-level simulation model is created that explores the execution and performance of such a system. CompDSE also has a way to include the impact of delays in the physical environment. This comprehensive methodology can produce very accurate results.

3.2 Resource Contention

3.2.1 Problem Description. Multi-core processors are widespread even in the domain of embedded and Cyber-Physical Systems [3]. Although they offer many benefits, they are also a major source of unpredictability [4, 6]. System resources are shared between applications running on different cores, which can lead to resource contention and performance degradation. Reasoning about time and safety guarantees becomes extremely difficult in this scenario, and the complexity of the analysis increases drastically [1]. Naturally, this creates a huge problem during the design of a CPS where the evaluation of requirements, such as performance, has to be performed for a huge number of possible configurations.

3.2.2 Where Contention Occurs. While in theory contention can occur in any shared resource, there are specific hardware components that are common bottlenecks. This is of course highly dependent on the system and the workload, but in the evaluated papers, LLC has consistently been reported to have one of, if not the highest impact on performance when many programs share hardware [2, 7, 15, 21, 22, 24]. Especially in cloud workloads, for example Virtualized Network Functions, LLC and network I/O, including Intel Data Direct I/O (DDIO), are the main points of contention [15].

One method of mitigating this problem is cache partitioning, for example the Intel® Cache Allocation Technology (CAT), which provides hardware mechanisms for partitioning the LLC across

cores [22]. This eliminates LLC interference, but does not completely remove the possibility of performance degradation. Firstly, if the cache is not large enough to accommodate the needs of all the applications after being split, performance degradation will still occur due to cache misses. Dynamic cache partitioning can help with this, but it is outside the scope of the research. Secondly, there may be other factors in this scenario that limit the performance, such as poor buffer management with Intel Data Direct I/O, a processor feature that enables direct NIC-to-LLC transfers [22].

3.2.3 Virtualization. Microservices aim to slice a bigger software system into smaller parts. These smaller services can be containerized and executed in isolation for a clean, maintainable, and scalable solution. Many of these containers are deployed on shared hardware. Virtualization allows for the isolation of different services, ensuring they run independently of other workloads. However, this mainly provides guarantees on function and security [24]. Because the underlying hardware is still shared, there are no performance guarantees, i.e. performance degradation can still occur due to interference. This again leads to a problem, as applications sharing the hardware resource often suffer performance interference from other virtual machines [24].

3.3 Interference Prediction

Bubble-Up is a methodology for predicting performance degradation that results from contention for shared resources [16]. The method relies on profiling the target applications in 2 steps:

- **Measuring sensitivity**

For each application a test is performed, in which it gets deployed with a memory stress test on shared hardware. This stress test is the **Source of Interference (SoI)**. By iteratively increasing the pressure applied by the SoI to the memory subsystem and measuring the performance of the tested application, a sensitivity curve is created. It shows how the application performs under each level of memory stress, which is called **sensitivity**.

- **Characterizing contentiousness**

In this step, the **contentiousness** of each application is measured by deploying it on shared hardware with a **Reporter**. Contentiousness can be described as the amount of pressure that an application is putting on a shared resource. The Reporter is a special application that is sensitive to interference, and by monitoring its own performance it determines how contentious the tested application is.

From this profiling, a sensitivity curve and a contentiousness score are obtained for every application. The SoI and the Reporter serve as common reference points for all applications. Profiling the applications with just these two tests is done to avoid running all possible combinations of the applications. After the software has been profiled, predicting performance relies on taking the contentiousness of one application and applying it to the sensitivity curve of another application to get the latter's performance.

A very similar method can also be applied to CPS that use microservices. Each microservice needs to be profiled to determine its sensitivity and contentiousness. Once these scores are obtained, it is possible to accurately predict the performance degradation caused by memory contention in pairwise deployments [5].

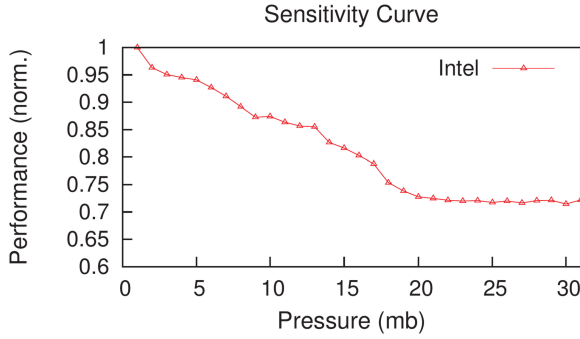


Figure 1: A sensitivity curve showing the degradation of application performance as the memory pressure from the SoI increases by Mars et al. [16]

User-Level Prediction. uPredict is a user-level predictor trained using micro-benchmarks in a live cloud environment [18]. The fact that it is user-level means that it can be used by ordinary users in multi-tenant cloud platforms, who have no control of the execution environments, no knowledge of the co-running applications, and no access to hardware performance counters. It relies on profiling the virtual machine’s performance. This is done by deploying several micro-benchmarks to assess the interference by observing the slowdowns. It is an interesting way to get performance estimations on multi-tenant platforms in which the users have limited privileges which prevent them from fully accessing the hardware, but that is not the scenario during CPS design.

3.4 Metrics-based Performance Prediction

A common way to measure performance is to use Hardware Performance Counters (HPC). HPC are low-level metrics which count hardware-related events such as cache misses, branch mispredictions, etc., and provide insight into how hardware resources are being utilized by programs [24]. More importantly, some metrics such as cache misses or cycles per instruction (CPI) are closely correlated with performance and can be used as an indicator of interference [15, 19, 21]. These metrics can be collected, for example, using the Intel® Performance Counter Monitor (Intel® PCM).

Prediction. HPC are useful for detecting resource contention, but can also be used to predict performance. It is possible to estimate the timing behavior of tasks by measurements. A resource-stressing benchmark can be co-deployed with the target application to obtain an estimate of the slowdown [1].

SLOMO is a performance prediction framework for Network Functions (NF) [15]. An NF is a virtualized software-based network service that performs a specific task. SLOMO works similarly to the Bubble-Up methodology. It measures sensitivity and contentiousness by deploying an application with a configurable synthetic workload. The difference is that it uses vectors to represent the contentiousness. The vectors contain the HPC metrics with the highest Pearson’s correlation coefficient between them and the performance. Furthermore, SLOMO allows for the calculation of the combined contentiousness of multiple Network Functions by

composing two vectors into one. This data is then used to build prediction models, with the best one being Gradient Boosting Regression. For each NF, a set of vectors \mathbf{V}_i^x is collected. Each vector represents the contentiousness of NF_i associated with every synthetic run x .

An extension of this approach is present in iPlace [2]. It is an interference-aware microservice placement approach based on clustering. From the perspective of this research, the most important parts of iPlace are the offline profiling and online prediction. iPlace works specifically with microservices. It also uses contentiousness vectors. Each vector is denoted by $\hat{\mathbf{V}}_r^{(k)}(\mathbf{x})$, where r is the profiled MS, k is the number of competing MSs, and x represents the competing workload and its specific configuration. The results are then averaged to get a representative contentiousness vector $\mathbf{V}_r^{(k)}$ that is then used in the sensitivity model during the online prediction phase. The sensitivity model simply consists of a mapping from the contentiousness vector to the observed throughput. During the online prediction phase, the performance of a given MS will be calculated from its sensitivity by applying the combined contentiousness vectors of the competing MSs.

Limitations. Measurement based approaches are not suitable in the context of hard real-time embedded systems. They cannot derive safe, analytical guarantees because it cannot be established whether the actual worst case execution time (WCET) has been encountered during measurements [1]. Additionally, research confirms that hardware metrics, such as L3 Miss Rate, can detect resource contention but are not always sufficient to detect performance degradation [19].

3.5 Cloud Datacenters

Research in cloud environments focuses on the scheduling of new microservice instances on a running system in a manner that will not cause performance degradation.

Modern cloud datacenters run a huge variety of workloads. To efficiently utilize the available hardware, multiple VMs are co-hosted on the same physical machine. As already established, this leads to performance degradation, so it is crucial for the datacenters to address this problem during scheduling. Naturally, some form of contention prediction is necessary.

ResQ is a resource manager for multi-tenant Network Function Virtualization clusters [22]. Its authors test the idea that cache contention is the main source of performance variability in Network Functions, which was established by prior research [7]. ResQ relies on CAT to mitigate LLC contention. The authors established that CAT is not always sufficient to prevent interference. The reason for this is what the authors named *The Leaky DMA Problem* which is related to the DDIO. ResQ first profiles any given NF to establish its resource requirements. More precisely, the throughput and latency are measured for different LLC allocations. This data is then used for scheduling in a way that prevents performance degradation.

C-Koordinator is an open-source solution, which incorporates co-location and interference mitigation strategies [21]. It uses machine learning models for interference prediction. The predictor is used by the interference detector and can trigger the interference mitigator. The explanation of these components of the C-Koordinator is omitted because they are not relevant to this research. The chosen prediction metric is CPI. The rationale for it is that C-Koordinator

focuses on large-scale, co-located microservice-based clusters with dynamic workloads and heterogeneous hardware configurations. In this context, high-level application performance metrics are insufficient. The measured Response Time (RT) can be influenced by many external factors and pinpointing the source of interference in hardware is difficult. Additionally, acquiring fine-grained RT data across thousands of production nodes is constrained by privacy, overhead, and access limitations. CPI is a low-level performance counter available on modern processors. It captures the combined contention across various hardware components well, because it is not only affected by CPU performance, but also memory delays, cache interference, and more. In the end nine metrics were used to train a model to predict CPI spikes.

For cloud datacenter workload, a similar approach can be taken using the CPU-Ready metric, which is also a good indicator of resource contention [23]. It starts with finding operational metrics (which can be HPC) that have a high correlation with the CPU-Ready metric. Traces with these metrics are then used to create a regression model which can then be used to predict the contention.

Although research seems to suggest that CPI and cache miss rates are sufficient for detecting interference, it may not always be the case. CPI may not be effective in identifying response time problems in highly multi-threaded applications [19]. L3 miss rates are not always sufficient to detect performance degradation. Relying on performance counters is not ideal in all cases. A different approach is to deploy a probe along with the target workload [19]. The role of the probe is to detect the contentiousness. It consists of a tunable micro-benchmark, and by observing its performance it is possible to detect interference.

3.6 Performance Prediction in CPS

One approach to evaluating a given configuration during the exploration of the design space is to perform a simulation. Assuming the simulation is adequate, this produces accurate results, which is important for systems with hard requirements and constraints. The problem with this approach is that due to the computational cost, an exhaustive simulation of the configuration space is often infeasible. To mitigate this issue, it is possible to run a shortened simulation on a subset of configurations and use the results to train a predictive model that will estimate the quality of other configurations. Research shows that simulating the operation of a CPS over a shortened period of time does not significantly deteriorate the accuracy of the model compared to a full simulation [25]. The sample size was also tested, but the results varied. For a sample size of only 1% of all evaluated configurations, the predictions were unreliable. For a sample size of 10%, the predictions became more stable and usable. When using all configurations for the training, the results yielded strong generalization for all mission durations.

An interesting way of looking at performance interference is from an attacker's perspective. It can cause timing issues, which may affect the system's behavior. An attacker can leverage this to cause unintended behavior in a CPS [14]. Once an exploitable condition is found, the attacker can trigger it by causing a specific level of interference. To induce the expected timing, it is necessary to interfere with only a subset of tasks while keeping the remaining ones untouched. Additionally, the delay should be manipulated

within a specific range, because it needs to be large enough to trigger the unintended behavior but not severe enough that it gets detected by possible check conditions. To establish the needed amount of interference, a profiling step is performed. Each task is divided into a sequence of execution phases where similar resources are intensively used in each phase. Then, the sensitivity of each phase to individual resource contention channels is characterized. From this a sensitivity curve is obtained. It contains data on what amount of delay is caused by a specific attack intensity. This is essentially performance estimation, except the goal is to establish what amount of contention needs to be caused to trigger specific timing delays in a CPS.

4 DISCUSSION

This section focuses on evaluating the gathered research papers and the solutions they contain in the context of the presented problem. The goal is to establish their relevancy and what techniques could actually be of benefit in the context of performance prediction in cyber-physical systems.

4.1 Cloud Environment

There is a significant amount of research focusing on performance prediction and minimizing resource contention in cloud environments. It provides useful techniques and insights on the topic. Unfortunately, cloud servers, their workloads, and objectives are noticeably different from those found in CPS. Firstly, they are not real-time systems that require safety and reliability guarantees. More importantly, the workload in a general cloud environment is dynamic. A system must be able to run any microservice, which can be unknown, and switch between them on demand. Although predictive models can be used in this case for performance estimation, the goal is different. The problem can be described as efficient load balancing during runtime. Since the focus of this research is the design-time, where the circumstances and goals are different. Because of this, it cannot be assumed that the performance prediction methods used in cloud environments are appropriate and satisfactory for the purposes of DSE.

In a cloud datacenter for usual workloads like Network Functions the bottleneck is usually the network bandwidth or shared cache. Solutions tend to focus on addressing bottlenecks present in those two hardware components. In a CPS, the workloads and bottlenecks could be more diverse. A better exploration of CPU contention is necessary. Alternatively, it would need to be confirmed that the LLC is also the main contributing factor for interference which is hard to establish for all Cyber-Physical Systems.

4.2 Simulation-based Approach

Using simulations for evaluation of the design points produces very accurate results, but with one obvious limitation - it is nearly impossible to make them fast enough to be viable for evaluating a huge design space in reasonable time.

Data from simulations can be used to build a prediction model, but this is also not ideal. The model will obviously not be as accurate as running a full simulation. This in itself is not a huge problem because the goal is to get an estimate, but the fact that it still requires running simulations makes the time savings not significant

enough to effectively address the discussed issue present during the evaluation phase of DSE. Additionally, simulations are required for all components of the CPS, which may also be a problem in the early design phase. Lastly, although the authors of [25] obtained some impressive results for the tested CubeSat system, it is not clear if it works as well with other systems, especially those that use microservices.

Although CompDSE [20] is an extensive and efficient methodology for DSE in the domain of dCPS that includes modeling and evaluation, it does not directly solve the huge time cost of the latter. Work has been done to make the process efficient, but it still requires manually adding trace points to the code and performing simulations on the execution traces. This makes it unclear whether it is feasible to perform the exploration step in a reasonable time for any given CPS.

4.3 Sensitivity and Contentiousness

The Bubble-Up methodology [16] seems to suit the objective of this research well. It only requires initial profiling of each application, but afterward the performance of two applications running together can be predicted very quickly by just getting the values from the sensitivity curves. This approach is simple, fast, and can be used in CPS [5], but it has one serious limitation. It can only be applied to pairs of applications and only takes into account the memory contention. During the design of CPS the number of applications running on a single hardware system cannot be limited. A compositional approach to estimating interference for multiple hardware resources is necessary.

The papers that use contentiousness vectors [2][15] initially seem to solve the problem. They offer metric-based performance estimation for an unlimited amount of applications, because the vectors can be combined to get the total contentiousness of multiple applications together. However, multiple vectors need to be calculated per application. There are two variables: the number of competing MSs, and the competing workload with its configuration. For each combination of those variables a contentiousness vector needs to be built via the method of measurements. It is unclear how well this approach scales. The focus of this study is to examine potential performance estimation methods that will significantly speed up the current DSE evaluation process while maintaining acceptable, but not necessarily perfect accuracy. Compared to the Bubble-Up solution, which only required one set of measurements per application, these approaches require $k \times x$ measurements **per application**. The total number of measurements needed for this method can skyrocket quickly with an increasing number of possible applications. Therefore, it cannot be established that it will produce significant time savings during the design process.

It is important to remember that these prediction methods cannot be used to reason about strict performance requirements and timing guarantees that are sometimes needed in CPS. Despite this, they are still extremely useful during the exploration of the design space because a general method for estimating application performance in CPS is what is required. Using quick, accurate-enough performance estimates can help narrow down the best and most viable configurations in a short amount of time.

4.4 Research in the CPS Domain

A large portion of the research specifically with CPS focuses on simulations [20][25]. Although these approaches work well, they are not really suitable for the early stages of the design process when the number of configurations is overwhelmingly large.

The other approach is to profile the applications once and construct sensitivity curves for them. This would work well for early design phases because it only requires the applications to be profiled on all the available hardware. While this could still necessitate a lot of runs, it is far better than running every configuration. After the profiling, the time cost of performance prediction is extremely small. This approach has the same limitations and benefits as the methods described in Section 4.3, but it is good to have confirmation that the method also works for CPS [5][14].

The work of *Li et al.* [14] which focuses on timing violation attacks is especially interesting because it not only uses sensitivity curves to precisely calculate the interference necessary to obtain a certain delay, but it also confirms that resource interference and performance degradation are a serious threat to the safety and reliability of a CPS.

5 CONCLUSION

Resource contention is a serious problem in computer systems and often leads to performance degradation. During the design of a Cyber-Physical System (CPS), there are many possible ways in which the software components can be mapped to the hardware. It is important to have the ability to estimate the performance of any such configuration in order to evaluate how good it is, based on the design objectives and constraints. This requires a method of predicting performance on platforms with shared resources.

Deploying multiple applications on the same hardware with shared resources often leads to resource contention and performance degradation. Because of this, getting an accurate estimate of the application performance indirectly relies on interference prediction. Hardware Performance Counters (HPC) are very useful in observing resource contention, because they report various hardware-level metrics such as cache miss rates.

The performance of a system can be estimated by running a simulation. This strategy is prevalent in CPS. It usually offers very good accuracy and potentially information about other important system behavior unrelated to performance. It offers strong benefits but has one potential shortcoming. Given a very large design space, it is infeasible to perform simulations on all the necessary configurations in an acceptable time-frame.

There is a lot of research on quick performance estimation in cloud environments. There, efficient scheduling is highly important, therefore methods of detecting and predicting interference are beneficial. These methods often use HPC to gather data about hardware usage. This data is then used to build a prediction model that can estimate the performance of co-deployed applications. However, methods developed for cloud environments cannot be assumed to work for CPS, especially when focusing on design-time instead of run-time. Further testing would be required to confirm whether these methods can be successfully used for CPS design.

The current quickest viable method for estimating performance in CPS is through measuring sensitivity and contentiousness. After

obtaining these scores from the initial profiling step, calculating performance estimates is simply a matter of combining the two values. Unfortunately, this method has its limitations. This simplified approach cannot be used for any number of applications. Its accuracy has only been tested on pairwise deployments. Another problem is that it does not take into account the resources in which contention may occur. There is only one sensitivity curve and one contentiousness score for all resources. Because the tests were mostly done on the memory subsystem, the method does not differentiate between e.g. CPU-heavy and memory-heavy workloads. A way to address both issues is to use contentiousness vectors instead of single scores. This method seems good, but it is unclear how well it scales with the number of software and hardware components. It would require significantly more runs during profiling.

There is no tested method for estimating performance of Cyber-Physical Systems during Design Space Exploration that would be quick and accurate enough to be viable for large design spaces. Therefore, further research is needed to establish such a solution.

REFERENCES

- [1] Andreas Abel, Florian Benz, Johannes Doerfert, Barbara Dörr, Sebastian Hahn, Florian Hauptenthal, Michael Jacobs, Amir H. Moin, Jan Reineke, Bernhard Schommer, and Reinhard Wilhelm. 2013. Impact of Resource Sharing on Performance and Performance Prediction: A Survey. In *CONCUR 2013 – Concurrency Theory*, Pedro R. D’Argenio and Hernán Melgratti (Eds.). Springer, 25–43. https://doi.org/10.1007/978-3-642-40184-8_3
- [2] Madhura Adeppady, Paolo Giaccone, Holger Karl, and Carla Fabiana Chiasserini. 2023. Reducing Microservices Interference and Deployment Time in Resource-Constrained Cloud Systems. 20, 3 (09 2023), 3135–3147. <https://doi.org/10.1109/TNSM.2023.3235710>
- [3] Benny Akesson, Mitra Nasri, Geoffrey Nelissen, Sebastian Altmeyer, and Robert I. Davis. 2022. A comprehensive survey of industry practice in real-time systems. 58, 3 (09 2022), 358–398. <https://doi.org/10.1007/s11241-021-09376-1>
- [4] Ayoosh Bansal, Jayati Singh, Yifan Hao, Jen-Yang Wen, Renato Mancuso, and Marco Caccamo. 2020. Cache Where you Want! Reconciling Predictability and Coherent Caching. In *2020 9th Mediterranean Conference on Embedded Computing (MECO)*, 1–6. <https://doi.org/10.1109/MECO49872.2020.9134262> arXiv:1909.05349 [cs]
- [5] Dzikowski Bruno. 2025. Practical Recommendations for Accurately Predicting Performance Degradation Caused by Memory Contention. https://scripties.uba.uva.nl/search?id=record_57157
- [6] Dakshina Dasari, Benny Akesson, Vincent Nélis, Muhammad Ali Awan, and Stefan M. Petters. 2013. Identifying the sources of unpredictability in COTS-based multicore systems. In *2013 8th IEEE International Symposium on Industrial Embedded Systems (SIES)*, 39–48. <https://doi.org/10.1109/SIES.2013.6601469> ISSN: 2150-3117.
- [7] Mihai Dobrescu, Katerina Argyraki, and Sylvia Ratnasamy. 2012. Toward Predictable Performance in Software Packet-Processing Platforms. 141–154. <https://www.usenix.org/conference/nsdi12/technical-sessions/presentation/dobrescu>
- [8] Nicola Dragoni, Saverio Giallorenzo, Alberto Lluch Lafuente, Manuel Mazzara, Fabrizio Montesi, Ruslan Mustafin, and Larisa Safina. 2017. Microservices: yesterday, today, and tomorrow. (04 2017). <https://doi.org/10.48550/arXiv.1606.04036> arXiv:1606.04036 [cs]
- [9] Jonas Fritzsche, Justus Bogner, Markus Haug, Ana Cristina Franco da Silva, Carolin Rubner, Matthias Saft, Horst Sauer, and Stefan Wagner. 2023. Adopting microservices and DevOps in the cyber-physical systems domain: A rapid review and case study. 53, 3 (2023), 790–810. <https://doi.org/10.1002/spe.3169> eprint: <https://onlinelibrary.wiley.com/doi/pdf/10.1002/spe.3169>
- [10] Teun Hendriks, Benny Akesson, Jeroen Voeten, Martijn Hendriks, Javier Coronel Parada, Miguel García-Gordillo, Sergio Sáez, and Joan J. Valls. 2023. Thirteen concepts to play it safe with the cloud. In *2023 IEEE International Systems Conference (SysCon)*, 1–7. <https://doi.org/10.1109/SysCon53073.2023.10131180> ISSN: 2472-9647.
- [11] Marius Herget, Faezeh Sadat Saadatmand, Martin Bor, Ignacio González Alonso, Todor Stefanov, Benny Akesson, and Andy D. Pimentel. 2022. Design Space Exploration for Distributed Cyber-Physical Systems: State-of-the-art, Challenges, and Directions. In *2022 25th Euromicro Conference on Digital System Design (DSD)*, 632–640. <https://doi.org/10.1109/DSD57027.2022.00090> ISSN: 2771-2508.
- [12] Pooyan Jamshidi, Claus Pahl, Nabor C. Mendonça, James Lewis, and Stefan Tilkov. 2018. Microservices: The Journey So Far and Challenges Ahead. 35, 3 (05 2018), 24–35. <https://doi.org/10.1109/MS.2018.2141039>
- [13] Edward A. Lee. 2008. Cyber Physical Systems: Design Challenges. In *2008 11th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC)*, 363–369. <https://doi.org/10.1109/ISORC.2008.25> ISSN: 2375-5261.
- [14] Ao Li, Jinwen Wang, Sanjoy Baruah, Bruno Sinopoli, and Ning Zhang. 2024. An Empirical Study of Performance Interference: Timing Violation Patterns and Impacts. In *2024 IEEE 30th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 320–333. <https://doi.org/10.1109/RTAS61025.2024.00033> ISSN: 2642-7346.
- [15] Antonis Manousis, Rahul Anand Sharma, Vyas Sekar, and Justine Sherry. 2020. Contention-Aware Performance Prediction For Virtualized Network Functions. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication (SIGCOMM ’20)*. Association for Computing Machinery, 270–282. <https://doi.org/10.1145/3387514.3405868>
- [16] Jason Mars, Lingjia Tang, Robert Hundt, Kevin Skadron, and Mary Lou Soffa. 2011. Bubble-Up: increasing utilization in modern warehouse scale computers via sensible co-locations. In *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-44)*. Association for Computing Machinery, 248–259. <https://doi.org/10.1145/2155620.2155650>
- [17] Manel Mena, Javier Criado, Luis Iribarne, Antonio Corral, Richard Chbeir, and Yannis Manolopoulos. 2023. Towards high-availability cyber-physical systems using a microservice architecture. 105, 8 (08 2023), 1745–1768. <https://doi.org/10.1007/s00607-023-01165-x>
- [18] Hamidreza Moradi, Wei Wang, Amanda Fernandez, and Dakai Zhu. 2020. uPredict: A User-Level Profiler-Based Predictive Framework in Multi-Tenant Clouds. In *2020 IEEE International Conference on Cloud Engineering (IC2E)*, 73–82. <https://doi.org/10.1109/IC2E48712.2020.00015>
- [19] Joydeep Mukherjee, Divakar Krishnamurthy, and Jerry Rolia. 2015. Resource Contention Detection in Virtualized Environments. 12, 2 (06 2015), 217–231. <https://doi.org/10.1109/TNSM.2015.2407273>
- [20] Faezeh Sadat Saadatmand, Todor Stefanov, Ignacio González Alonso, Andy D. Pimentel, and Benny Akesson. 2025. CompDSE: A Methodology for Design Space Exploration of Computing Subsystems Within Complex Cyber-Physical Systems. 10, 1 (2025), e70019. <https://doi.org/10.1049/cps2.70019> eprint: <https://ietresearch.onlinelibrary.wiley.com/doi/pdf/10.1049/cps2.70019>
- [21] Shengye Song, Minxian Xu, Zuowei Zhang, Chengxi Gao, Fansong Zeng, Yu Ding, Kejiang Ye, and Chengzhong Xu. 2025. C-Koordinator: Interference-aware Management for Large-scale and Co-located Microservice Clusters. (07 2025). <https://doi.org/10.48550/arXiv.2507.18005> arXiv:2507.18005 [cs]
- [22] Amin Tootoonchian, Aurojit Panda, Chang Lan, Melvin Walls, Katerina Argyraki, Sylvia Ratnasamy, and Scott Shenker. 2018. ResQ: Enabling SLOs in Network Function Virtualization. 283–297. <https://www.usenix.org/conference/nsdi18/presentation/tootoonchian>
- [23] Vincent van Beek, Giorgos Oikonomou, and Alexandru Iosup. 2019. A CPU Contention Predictor for Business-Critical Workloads in Cloud Datacenters. In *2019 IEEE 4th International Workshops on Foundations and Applications of Self* Systems (FAS*W)*, 56–61. <https://doi.org/10.1109/FAS-W.2019.00027>
- [24] Sa Wang, Wenbo Zhang, Tao Wang, Chunyang Ye, and Tao Huang. 2015. VMon: Monitoring and Quantifying Virtual Machine Interference via Hardware Performance Counter. In *2015 IEEE 39th Annual Computer Software and Applications Conference*, Vol. 2, 399–408. <https://doi.org/10.1109/COMPSAC.2015.14> ISSN: 0730-3157.
- [25] Marco Wijaya, Sami Lazreg, Tagir Fabarisov, Andreas Hein, and Maxime Cordy. 2025. Performance Prediction of Cyber-Physical Systems Product Lines in Dynamic Environments. In *Proceedings of the 29th ACM International Systems and Software Product Line Conference - Volume A (SPLC-A ’25)*. Association for Computing Machinery, 184–189. <https://doi.org/10.1145/3744915.3748476>