Hardware Dimensioning for Microservice Applications in Cyber-Physical Systems: Current Directions and Challenges

Marijn Vollaard
Universiteit van Amsterdam (UvA), The Netherlands
Vrije Universiteit Amsterdam (VU), The Netherlands
m.j.vollaard@student.vu.nl

ABSTRACT

With the rise of large-scale web applications, microservices were introduced as an alternative to monolithic systems. Microservices provide scalability, agile development, and customization in the deployment of microservices. For these reasons, microservices have not only been adopted in a web environment but also in cyber-physical systems (CPS). With this development, a new problem arose. There is little predictability of the performance of microservice-based applications, because of their complex structure. In a cloud context, performance prediction for a given hardware configuration is not essential, since typical cloud applications do not have rigid real-time requirements and applications can be elastically scaled. However, applications for CPSs generally cannot be scaled dynamically, as a complete product has to be delivered to a customer, and an estimate has to be made on the necessary hardware to satisfy rigid performance requirements within the application. Thus, a structured method of hardware dimensioning, assigning hardware to an application, becomes necessary.

This literature study investigates state-of-the-art hardware dimensioning by discussing various application profiling and system profiling methods, performance prediction models, and design space exploration methods in the context of microservice-based applications. It then discusses the applicability of the state-of-the-art in the context of CPSs and explains the method of leveraging the concepts for offline hardware dimensioning. It concludes that future work has to be conducted in profiling, performance prediction, and design space exploration to investigate whether the methods used in cloud environments are also applicable in the cyber-physical space.

KEYWORDS

Microservice Applications, Cyber-Physical Systems, Hardware Dimensioning, Application Profiling, System Profiling, Performance Prediction, Performance Modeling, Design Space Exploration, Literature Review

1 INTRODUCTION

In the past 20 years, large-scale web services have had an increasing need for scalability in their applications [31]. Big monolithic applications that run on multiple servers could not cope with the heavy load that was requested with the increasing user base of the applications. To solve this problem, microservice architectures were introduced [18, 28]. Microservices provide scalability, agile development, and more customization in the deployment of services of an application [31].

Microservices are adopted, not only in the field of large-scale web services, but also in applications in the cyber-physical space [25]. A cyber-physical system (CPS) is a system with both computational and physical components [32, 34]. In such a system, it is not primarily the scalability that is advantageous, but the agile characteristics of a microservice architecture. A CPS is often an instance of a product, and it can go through various iterations of development. Based on the demands of the users and stakeholders of the product, it may need various features and capabilities, which can influence performance and latency requirements. Thus, a product can have variability per product instance. A microservice architecture is advantageous in this context since services can be used and deployed according to the needs of the customer and a system is therefore more customizable [45]. Additionally, some literature provides evidence that microservice architectures do not necessarily degrade performance and may improve performance in various contexts [7, 12, 47].

The variability per instance of a product causes problems in terms of performance guarantees. It is often the case that building a full CPS is very costly, so there is a need for a way to predict the performance of a system before it is built for different configurations. This is made more critical by the fact that the performance of a system can be influenced by the amount of hardware as well as the quality of hardware. To guarantee the satisfaction of performance requirements, sufficient hardware needs to be provided. This process of assigning hardware to an application is called hardware dimensioning. In the cloud, this problem is generally solved by deploying randomly and adapting to the needs of the application while it is running [20, 42]. For a cyber-physical system, this is not an option, since a product is often presented in full, including the hardware it runs on, and it may not be possible to add hardware after the fact due to limited energy supply or available space. Typically, not all hardware of a system is available, except for some testing infrastructure. Guaranteeing performance beforehand therefore requires some prediction of the performance of an application given a hardware configuration.

This literature study gives an overview of the current methods of hardware dimensioning, discusses their uses and shortcomings in the context of cyber-physical systems and microservice systems, and looks at future directions in the field. The remainder of this literature study is structured as follows: Section 2 provides the background knowledge required to understand the concept in the rest of the study. Section 3 provides an overview of the related work in the area of hardware dimensioning, Section 4 discusses the related work and how it applies to the hardware dimensioning

problem in the context of cyber-physical systems and Section 5 reviews and concludes the findings of this study.

2 BACKGROUND

This background section aims to give an understanding of the fundamental concepts of the material discussed in the sections that follow it. Section 2.1 describes cyber-physical systems and explains their characteristics, Section 2.2 explains what microservices are and why they are increasingly used, Section 2.3 explains the concept of Design Space Exploration (DSE), and Section 2.4 will explain the concept of hardware dimensioning, and what it aims to do.

2.1 Cyber-Physical Systems

Cyber-physical systems are systems that consist of computational and physical components. Usually in such systems, feedback loops are employed, where physical processes affect computation, and computation affects physical processes [9]. CPSs are generally held to a higher standard of reliability and predictability than general-purpose applications [32] and most industrial CPSs will have mission-critical functionalities, or functions, that need to work correctly, while satisfying performance requirements, to do their job correctly. This means that a CPS will often be held to rigid resource utilization and latency requirements. At the same time, the real world is not always predictable, and physical components can behave unexpectedly. Unexpected behavior in physical components can influence computational components negatively, which makes claims regarding performance and latency requirements harder to guarantee. The combination of higher reliability standards and unexpected behavior makes predicting the performance of a CPS different from general application performance prediction.

Often multiple instances of a Cyber-Physical System are developed. Each instance is tailored to the needs of the stakeholders involved. Due to this variability in a product, microservices are increasingly used in such systems, because of their flexibility in development, deployment, and placement on hardware [25].

2.2 Microservices

When monolithic applications started growing, scaling of the application became more difficult. Microservices architectures came up as a solution for this scalability issue. A microservice is a small, independent, loosely coupled service that has a specific functionality within a system [28, 40]. It communicates and exchanges data with other services so that the services together make up an application or microservice architecture. Because of their independent characteristics, microservices can be developed, deployed, and scaled up without needing to fully redeploy the whole system of services.

Additionally, microservices in an application can be deployed in heterogeneous environments, meaning a microservice can be deployed on different hardware, as well as be developed with different software. The only environment variable that has to be agreed upon is the communication method. A few of the most used methods are RESTful services [52], gRPC [2], or publish/subscribe channels [6]. This communication can be synchronous or asynchronous, depending on the application.

The distribution of services and functionalities of a system also has disadvantages. With an increase in microservices in systems

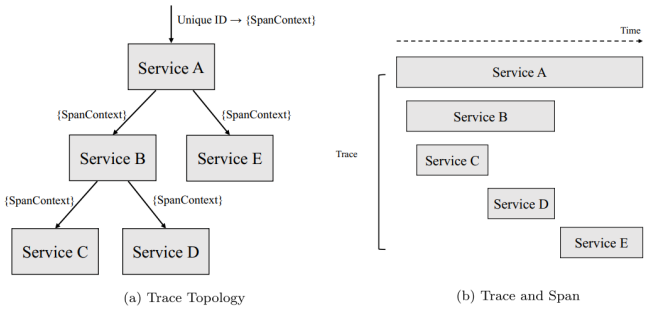


Figure 1: Example of trace spans for a simple microservice architecture [33]

running in heterogeneous environments, ensuring the maintainability, availability, security, performance, and testability of a system becomes more challenging [18, 27]. Both the independent nature and the broad deployment potential of microservices make it an attractive software architecture to use in the context of cyber-physical systems. However, this high potential for customization may bring with it a lack of predictability of the performance of a microservice system, which forms a problem in this context.

To get insight into the performance of a microservice system, there is a need for observability [33]. Observability is a measure of how well the state of a service can be known from external outputs [21]. These external outputs consist of the three pillars of observability. They are (i) traces, (ii) logging, and (iii) metrics. A Trace is a record of activities or interactions that occur within a service or application that processes requests or performs tasks. Figure 1a shows the trace topology for a simple microservice architecture. Logs are records of events, actions, or messages within a service or application. Metrics are measurements that quantify the behavior and performance of a service or application, such as CPU and memory usage, over time. Of these three pillars, gathering useful traces for microservice architectures proves more complex than for monolithic architectures. Mostly due to the complexity of dependencies between microservices [33]. A trace is often made up of various subtraces that originate in other microservices as is shown in Figure 1.

A directed acyclic graph (DAG) can be used as an overview of the dependencies within a microservice-based application and represents all its possible traces. A DAG consists of edges and vertices, where the edges point towards one of its vertices. The path through a DAG is never cyclic. Figure 1a is an example of a DAG.

2.3 Design Space Exploration

"Design Space Exploration (DSE) is the process of discovering one or many design solutions that best satisfy defined design objectives given a space of tentative solutions called design points" [26]. It is used to solve an optimization problem. In the context of hardware dimensioning, model-driven DSE can be used to find an optimal, or near-optimal, deployment of microservices over available hardware [49]. By finding a good deployment configuration of an application, the amount of hardware that is needed to satisfy performance

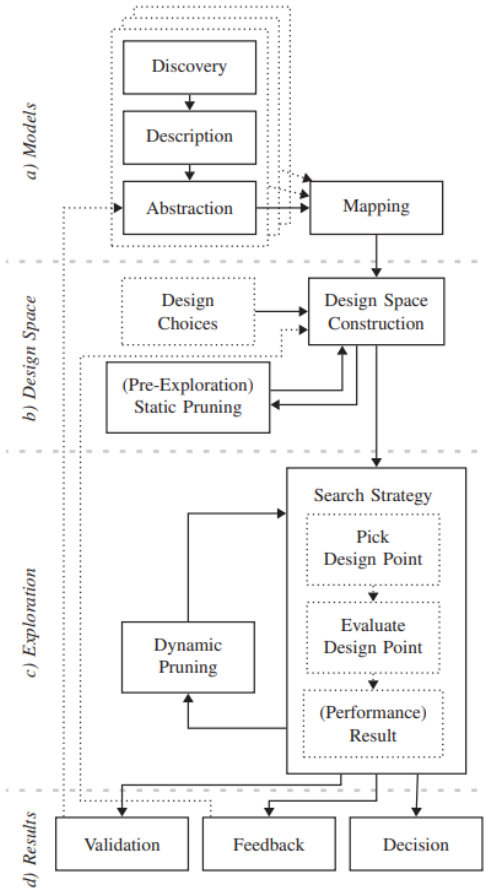


Figure 2: General DSE workflow by Herget et al. [26]

requirements may be reduced. A general DSE workflow is shown in Figure 2. The workflow is split into four main steps. In Step 1 models, and Step 2, design space, shown in Figure 2a and Figure 2b respectively, the construction of design points is defined and the design space is identified. The design points and design space are determined by the context and goal of the design space exploration.

To find an optimal design point in the design space, it needs to be evaluated. The evaluation is represented by Step 3, exploration, shown in Figure 2c. Then, a search strategy is needed to traverse the design space to find the next design point. A search strategy determines the next design point to be evaluated. The method of picking the next design point is what defines the search strategy. DSE moves towards an optimal, nearly optimal, or acceptable design point, which is then the result of the optimization problem. This is the last step, results, shown in Figure 2d. The evaluation of a design point can be done by the performance prediction methods discussed in Section 3.2. A prediction, thus, maps a design point to performance.

2.4 Hardware Dimensioning in Cyber-Physical Systems

As mentioned in Section 2.1, cyber-physical systems are often developed for multiple instances of one product. Each instance may have different stakeholders, functionalities, and, therefore, performance requirements. To ensure the performance of a product, sufficient hardware needs to be provided to the application. The process of assigning hardware to an application is called hardware dimensioning. It is an important part of the development of cyber-physical systems and is often done without much scientific backing. A product can have multiple different instances that each have vastly different functionalities. This means that the hardware assigned to each instance may differ as well. When a product has had many iterations, experts in the system can assign hardware based on previous experience. However, with new products, or instances that have very different functionalities, this experience does not exist yet. It is often the case that a full hardware system is not yet available in the development stage of a product, except for some testing infrastructure. Then, a prediction of the performance of the system needs to be made based on this.

In a system with a monolithic application, this is also a problem [10]. Measurements can be done on the testing infrastructure, but a prediction needs to be made on the effect of upscaling more servers with the same piece of software on it. With the introduction of microservices, performance satisfaction may be achieved differently: by varying the placement of microservices over nodes. The flexibility of a microservice system brings with it the possibility for optimization by deployment. On the other hand, it raises questions regarding what services should be run on what hardware, and how they should be configured. Combining certain microservices, or splitting them over various nodes may improve the performance of an application. Additionally, hardware can be heterogeneous, so services may run differently on different nodes. To fully leverage the flexibility and apply it for hardware dimensioning, there needs to be a thorough insight into the performance of the application and the impact of deployment and placement of microservices over the hardware.

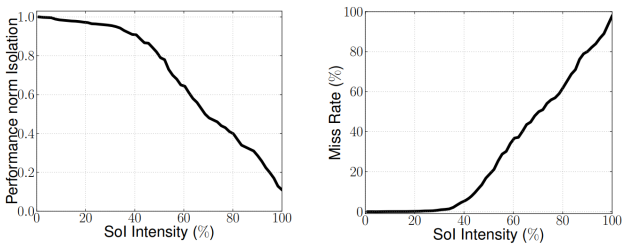
The reason for hardware dimensioning is mainly economic. The goal is often to satisfy performance requirements with as few resources as possible, to reduce costs. The use of too much hardware, or over-dimensioning, causes unnecessary costs. Of course, providing too little hardware may result in unmet performance requirements.

Thus, there is a need to predict the performance of an application given certain hardware and a configuration of microservices over nodes in the context of cyber-physical systems that employ a microservice architecture. Section 3 will further discuss current findings and related work in this area.

3 RELATED WORK

In this section, related work will be laid out. The process of hardware dimensioning has been split into three topics: Profiling, performance prediction, and design space exploration. These together can be used to achieve hardware dimensioning, with a focus on optimizing microservice placement. The scope of the related work is not limited to cyber-physical systems, since knowledge from other

systems and fields can be used to apply hardware dimensioning for CPSs. Section 3.1 is about application profiling as well as system profiling. Section 3.2 describes work on the performance prediction of applications, in various environments. Section 3.3 showcases ways to leverage a performance prediction to perform design space exploration with various purposes. The goal of this section is not to judge the various methods on their applicability to hardware dimensioning in CPSs, but rather to give an outline of the current state-of-the-art in these areas.



(a) (b)

Figure 3: The mfc application performance from SPEC-CPU2006 [1] benchmark suite, influenced by increasing intensity of an LLC-SoI (Last Level Cache Source of Interference). 3a shows the normalized performance of mfc with increasing intensity of the LLC-SoI. 3b shows the LLC miss rate of mfc with increasing intensity of the LLC-SoI

3.1 Profiling

Profiling is the process of monitoring and characterizing behavior, performance, and resource utilization. It can have various purposes and is generally used to gain insights into how an application or system is functioning. It can also be used to identify areas where improvements can be made. There is a distinction between application profiling and profiling of the system the application runs on. Application profiling can be based on the design knowledge of the applications or their components. For example, Do et al. [17] consider application-specific metrics, such as HTTP requests processed per second, for Apache, or the number of frames processed per second, for X264, a video encoding program. Metrics may also be more general. Han et al. [24] use metrics like CPU and memory usage to profile various applications. These metrics would not only be useful for a specific application, but can be used to profile any general application.

System profiling may include available resources in hardware, or resource utilization. For example, Bao et al. [11] define system specifications such as the number of cores per CPU, available RAM, and storage as indicators of a system's profile. It, thus, becomes clear that profiling can be done with many different metrics in mind, depending on the purpose of the profiling, the stage of development, and the target hardware and infrastructure [50].

A profiling and prediction categorization of the related work is shown in Table 1, Table 2, and Table 3 to give an overview of the distinctions between the literature described in Section 3.

Some of the literature performs methods of offline profiling [6, 16, 30]. Offline profiling is done in the development stage of an application, where the application can run on early available hardware, without the need for the full context in which the application will finally be used and run. There are also methods of online profiling [19, 23, 35]. Online profiling is done while an application or component is running and being used. Generally, it leverages the fact that much data can be gathered while an application is being used and deployment optimization can be done on the fly. While the quantity of data is often higher, the type of data is the same. Most work includes memory, CPU, and network utilization in both application profiling, as well as system profiling. An example of a system profile is various metrics per available nodes in a cluster of nodes [30]. The purpose of online profiling is generally similar to that of offline profiling: to optimize resource usage or predict QoS satisfaction. However, online profiling can adapt to changing situations in an environment, while offline cannot.

On the other hand, an advantage of offline profiling over online profiling is that it can be done in various stages of the development of an application. For example, Aksakalli et al. [6] describe early profiling by analyzing design-level aspects of an application, while considering hardware and the system it will run on. They leverage application characteristics in the early design phase of an application to develop a systematic approach to service placement over nodes in the deployment phase. The idea is that it requires a lot of expertise in an application to place services over nodes in an efficient way. This task becomes increasingly difficult with the number of services and possible combinations of services over nodes. Finding a systematic approach can alleviate this problem. Additionally, generating deployment configurations early on may aid in the development of services and avoid re-work of detailed design, development, or testing [6]. The profile is made up of a service data exchange metamodel, which specifies types and object sizes of communication between services. Moreover, it uses a communication model that displays communication type and frequency among services. These models are then used to estimate the communication load for each service in a system. Lastly, the profile includes the expected resource utilization of services on a node, of which examples are CPU utilization and memory usage. The profiling is then used to predict whether there will be a suitable deployment configuration that satisfies performance expectations, like Quality of Service (QoS) requirements or Server Level Objectives (SLOs).

As with bottleneck analyses, profiling can be used to improve an application in the design and implementation phase [6], but it is also often used in predicting the behavior of an application or system in a certain context.

For example, Delimitrou et al. [16] create an application profile based on interference workloads. It deploys a contention benchmark suite, that includes 15 Sources of Interference (SoI), that can highlight interference of co-scheduled applications in data centers, or applications that run concurrently on the same server. Data centers are facilities that provide hosting and managing services of servers for cloud computing, storage, and a wide range of applications. An example of the use of an SoI is shown in Figure 3. The LLC-SoI (Last Level Cache Source of Interference) is a small application that can be used from 0% to 100% intensity. For this SoI, 0% intensity means that the small application occupies 0% of the LLC, and 100% means it occupies 100% of the LLC. As shown

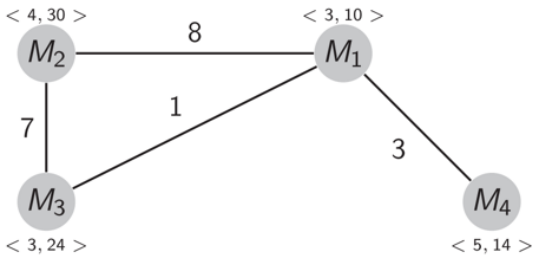


Figure 4: Undirected doubly weighted interaction graph, for a toy application with four microservices [30]

in Figure 3a, the performance of the mfc application from SPEC-CPU2006 [1] degrades with higher occupancy of the LLC by the SoI. Furthermore, the LLC miss rate of mfc goes up with the intensity of the SoI. This performance degradation indicates a sensitivity to LLC interference for the mfc application. An application is profiled based on all SoIs that influence the performance of an application. In this example, the profile has an emphasis on the behavior of an application under certain specific conditions or under specific interference. This profile can then be used to schedule an application on a server, where applications with similar profiles will not be scheduled together to minimize performance degradation due to interference.

Adeppady et al. [4] also use the idea of identifying interference patterns, where microservices of an application are run separately and then together in pairs of two on a node. When both microservices compete for resources, the performance of both goes down relative to the performance of the same microservices that are run in isolation. This difference will then be part of a profile of a microservice in the form of a contentiousness vector. Each value in the contentiousness vector stands for a pair of microservices and all vectors together form the profile of the application.

Profiling is often done per component in an application. Joseph et al. [30] aims to improve upon the random placement of microservices over resources by Kubernetes [3]. It does so by building IntMA, which is a profile represented by an interaction graph. The graph includes the amount of communication between nodes, as shown on the weight of the edges of the graph in Figure 4. The weight indicates the amount of communication between the two vertices, or microservices. Each microservice has its own weighted vertex, where the processing and memory requirements of a microservice are stored. The profile of an application is then used to find an optimal deployment configuration of microservices over nodes, while trying to minimize the interaction between services.

3.2 Performance Prediction

Based on profiling of an application and system, a performance prediction for an application can be made. The prediction method depends on its purpose and how the performance is defined. As with profiling, performance prediction can be done for vastly different purposes and can be done both online and offline. This is shown in Table 1. Generally, a performance model is used for the prediction. Various studies create an analytical model to predict the performance of an application.

Both Chen et al. [14] and Bao et al. [11] use a similar analytical model to predict the response time per component of an application. Although the purpose and environment of the works are different, the performance definition is very similar for both works. It is defined as the response time or end-to-end latency of an application. The prediction is made by analyzing the response time of the components in an application and some form of the sum of the response times is then used as the application response time.

Chen et al. [14] perform offline prediction of component-based applications in large-scale systems with an analytical model. It tries to predict the response time per component of an application, based on a profile of each component and the number of threads used by the application. The goal is to assign the optimal amount of threads to a component, minimizing the response time. The profile includes the overhead ratio per concurrent request to access the container, the average processing time for a single thread, and the processing time on the back-end resources, like databases. Prediction is done with a simple analytical model, based on the number of concurrent requests, x, and the number of threads used by the application, y. The model is shown in Equation (1), where T is the expected response time, a is the overhead ratio per concurrent request to access the container, b is the average processing time for a single thread and c is the processing time on the back-end resources. With this model, the response time of a component can be calculated for different amounts of threads. By varying the thread count and comparing the results, an optimal number of threads can be found for a component.

$$T = ax + \frac{bx}{y} + cy \tag{1}$$

Bao et al. [11] also aims to predict response time, but in the context of microservices in a cloud environment. Its purpose is to minimize costs for an application that has to be scheduled in a public cloud. Costs are assigned to VMs that are deployed with one or more microservices. It tries to both predict the execution time of microservices, as well as the monetary costs of running the microservice on a VM in the public cloud infrastructure. Microservices are deployed and scheduled on demand, as to minimize costs. The execution time of microservice i at time t is split up as shown in Equation (2), where $ET_i(t)$ is the execution time of microservice i at time t, $IT_i(t)$ denotes the initialization time of microservice i on the cloud infrastructure, $PT_i(t)$ denotes the processing time in microservice i, depending on the concurrent microservices running in its VM and node, and $RT_i(t)$ denotes the time it takes microservice i to transfer its output data to all succeeding microservices.

$$ET_i(t) = IT_i(t) + PT_i(t) + RT_i(t) \tag{2}$$

For the prediction of the overall application end-to-end latency, the latencies of each microservice on the critical path [39] are added together. It will also predict the monetary costs of one or more requests to the application in the cloud based on the processing time of microservices and the costs per second to run a VM in the cloud environment. Thus, with this model, a prediction can be made on the expected costs of an application, and an optimal configuration of microservices over VMs and nodes can be found to minimize the costs of a full application. The goal is to find an optimal point where both the number of VMs is low, as well as

the end-to-end latency of requests for the application, so that the performance is sufficient, while the costs are kept low.

While the details for the prediction of response time per component differ for both analytical models, the predictions show similarities. For example, both models sum the response time of the application components to come to a full application response time prediction. They use the sum of individual characteristics as a prediction for the characteristics of the application as a whole. Moreover, the models are linear, in the sense that they are additions of execution times of different processes within the components of an application.

Adeppady et al. [4] show a prediction model that does not predict the response time of an application, but rather the throughput per microservice. As described in Section 3.1, Adeppady et al. [4] builds a profile of an application based on the contentiousness of each microservice. Its prediction model consists of two components: An offline component and an online component. The offline component leverages the contentiousness of microservices. This contentiousness is computed by running microservices isolated and then in pairs, where the contentiousness is calculated from the difference in performance between those two setups, with increasing load for the microservices. The contentiousness of each microservice is represented by a vector, where each value in the vector represents the interference of each pair of microservices. An offline prediction of the interference for microservice a, running concurrently with b and c, between three microservices can then be made based on the profile of the contentiousness of microservices. The aggregate contentiousness vector of a and b is shown in Equation (3).

$$V_{bc} = V_b + V_c \tag{3}$$

Here, V_b and V_c are the contentiousness vectors of b and c, and V_{bc} represents the expected interference of the microservices combined. The second component of the prediction model is based on the expected interference, or contention, for microservice a. The throughput of a can be predicted based on the contentiousness vector of microservices b and c with an online regression model [51]. The full prediction model is then used for dynamic microservice placement over nodes in the cloud, where minimizing the throughput of each microservice is the goal.

Various works use machine learning in performance prediction [22, 23, 35, 36] to dynamically improve the performance of an application in a cloud environment. Both Rahman et al. [40] and Zhang et al. [53] fall under this category. They aim to make an online prediction of the end-to-end tail latency of an application in a cloud environment, without needing much knowledge of the mechanics and structure of an application.

Rahman et al. make a prediction based on CPU utilization of both VMs and pods in which the microservices run, a directed acyclic graph (DAG) of the application, and usage of last-level cache. These form the input for a machine learning model. The output of the model is the expected end-to-end tail latency of requests for the application. Various machine learning methods are applied, including linear regression [51], support vector regression [43], a decision tree model [38], and a deep neural network [41]. In this experiment, the linear regression has the lowest prediction accuracy, and the deep neural network the highest. Based on the prediction and the SLOs set for the application, a recommendation can be made for upscaling of number of instances per microservice. The goal is, thus, to satisfy SLOs, and dynamically predict when upscaling of resources is needed in the short term, while minimizing resource utilization.

Zhang et al.[53] also dynamically predict the end-to-end tail latency in the short term with a deep neural network. The input of the model is the number of requests per second, CPU, memory, and network usage. However, they additionally employ a decision tree model to try to predict whether the QoS target will be met in the long term. The results of these models can then be combined and used as input by an online scheduler, that maps microservice instances to nodes.

3.3 Design Space Exploration

As discussed in Section 2.3, the workflow of DSE can be split up into four main steps. The works in this section are structured along them. In Step 1 and Step 2 of Figure 2, the design space and design space points that make up the design space are identified. Both are heavily influenced by the purpose and environment in which the DSE takes place.

For example, Delimitrou et al. [16] try to optimize the scheduling of applications over available servers. The design space points are abstracted to the mapping, or placement, of applications on servers. The design space is every distribution of applications over the available servers, which grows exponentially with the number of applications. Similarly, Joseph et al. [30] define the initial distribution of microservices over available servers as design points. The distribution of services over nodes may also be considered in an iterative sense, where an initial distribution is made, after which a redistribution of services can be made based on monitoring of performance data [19, 46].

Rahman et al. [40] and Grohmann et al. [23] define an optimization problem of minimizing instances per microservice, and therefore, resource usage, while satisfying performance requirements. A design point is made up of a list of the number of instances per microservice. The design space is in theory infinite, since the number of instances per microservice can scale to infinity, although in practice, there will be a limit. Since optimization is done toward resource efficiency, design points with increasingly large instances per microservice will not be considered.

Somashekar et al. [44] aim to optimize performance in terms of end-to-end latency of applications by optimizing the configuration parameters of individual microservices that make up the application. Each design point represents a list of values for the considered parameters per microservice. Since the number of microservices in an application can be very large, and the number of configuration parameters grows exponentially with the number of microservices [44], the design space may become very large. A dimensionality reduction, or design space reduction, is therefore explored in three different ways: (i) Only microservices on the critical path are considered, where the critical path is retrieved based on current practice on critical path identification [39], (ii) Only microservices with high variability in performance are considered, (iii) Only the microservices identified by prior work as bottlenecks

are considered. Reducing the design space in such a way may aid in finding an optimal design point more quickly.

To find an optimal design point in the design space, multiple design points need to be evaluated. This is represented by Step 3. A design point can, for example, be defined as a placement of services over nodes [4], characteristics of applications [6], or parameter configuration per microservice [44]. The performance is defined as one or more metrics, like predicted end-to-end latency [11] or expected throughput of a service [4]. Then, a search strategy is needed to traverse the design space to find the next design point.

For example, Han et al. [24] use a greedy heuristic to pick a suitable cluster of nodes for a microservice-based application, where it chooses between multiple different clusters in a cloud environment. Since the evaluation, with the performance prediction, is done offline, the DSE can also be done offline. The microservices of the application are ranked in decreasing order by interaction rate, explained in Section 3.1 and Joseph et al. [30]. Then they are placed on nodes that are also ranked in decreasing order by the node's CPU utilization capacity. If an application does not fit on a cluster, the next cluster will be tried out, until a cluster is found on which the application fits. In this sense, the DSE is not necessarily looking for an optimal deployment, but rather the first acceptable option.

Somashekar et al. [44] aim to optimize configuration parameters for individual microservices in an application and consider 6 different black-box search strategies to come to a near-optimal configuration. Dynamically Dimensioned Search (DDS) [48] arrives at the highest performance in terms of performance, measured in end-to-end tail latency of requests in an application, while taking to least time to arrive at the result. The algorithm starts with an initial parameter configuration and then perturbs the values of the parameters based on a perturbation factor [48]. The most important characteristic of the search strategy is that it moves quickly from a global search to a local search, where it will arrive at a local optimum.

Several studies [13, 29, 35] use reinforcement learning [37] to redistribute microservices over a deployment environment. The idea is that both the environment in which the microservices are running and the arrival pattern and number of requests can change, which may limit the efficiency of offline methods. Reinforcement learning uses objective functions as models, which function as the evaluation for a given configuration of microservices. For reinforcement learning, this is called a reward. The reinforcement learning tries to get the highest rewards, which it retrieves based on the prediction model for a configuration of microservices and arrives at optimal configuration by iteratively changing it. The evaluation and DSE are inherently done online for these works, since it depends on the changing environment and incoming requests.

Ma et al. [36] use and improve upon an evolutionary algorithm [8], NSGA-III [15], to come to an optimal initial distribution of microservices over nodes. An evolutionary algorithm works with a population of individuals, where each individual is a design point. Mutation and crossover are used to create new individuals for the population. The fittest individuals will be selected and go on to the next generation, or iteration of the evolution, where they will, again, mutate and crossover to create a new generation. This process is continued until a stop condition is met. The stop condition can be defined in different ways. For example, it can stop after a specified number of generations, after a suitable solution is found, or when the improvement per generation is very minimal, indicating that either a local or global optimum is found. The best individual of the last generation then represents an acceptable placement configuration of microservices over available nodes.

Fu et al. [19] consider both the initial distribution of microservices in the cloud-edge continuum and the redistribution of microservices over available nodes and edge devices. An initial distribution is made according to expected communication overhead among services, based on graph interaction between them. The redistribution of microservices is done based on an evaluation of IO-sensitive microservices and load-dynamic sensitive microservices. The search strategy is not random, like various works discussed here [36, 44], but it is steered by run-time data collection and online performance prediction. In this case, the performance prediction is not only used as an evaluation in the DSE, but also as part of the search strategy.

4 DISCUSSION

In this section, the related work from Section 3 is put in the context of hardware dimensioning in microservice-based cyber-physical systems. Section 4.1 discusses the applicability of the related work in a microservice architecture, Section 4.2 explores the relevance of the related work in the context of CPSs, and Section 4.3 examines how the related work can be leveraged in hardware dimensioning in this context.

4.1 Microservice Architecture

Microservices-based applications consist of smaller, more granular, loosely coupled services. An application can vary from a few to hundreds of microservices. The related work discusses both methods of profiling with application-specific metrics [17], as well as more general metrics [24]. Application-specific and microservice-specific metrics can be useful for a more detailed profile, but the method of profiling may not scale well with the number of microservices in an application. Additionally, comparisons between different microservices are harder to make when the metrics for each microservice differ. For this reason, most profiling methods discuss general metrics, like CPU usage and memory usage, to profile larger-scale applications. These metrics apply to any microservice, regardless of their structure or function.

The related work of Section 3.1 indicates a need to both include communication between microservices as a factor of expected performance [30], as well as the contention, or interference, microservices may inflict on each other when they run on the same node [4, 16]. This highlights an optimization trade-off in the optimal deployment of microservices over nodes: To minimize communication, microservices should be deployed on the same node as much as possible. To minimize interference, microservices should each be spread out. This indicates an optimization problem, where the optimum will be somewhere in between all microservices on one node, and each microservice being deployed on a separate node.

As shown in Table 1, most methods are performed in the context of microservices-based applications. Microservice-specific methods are shown to be useful, such as IntMA [30]. However, aspects of

application profiling can be taken from other contexts as well. Profiles for applications in data centers are made as a basis to predict the interference of applications on each other. An SoI suite is built by Delimitrou et al. [16] and may be adapted to the context of microservices, where the interference of each individual microservice is profiled. Adepaddy et al. adapted the idea of interference in a similar way, creating a contentiousness profile for each microservice in an application.

The same is true for prediction models. Chen et al. [14] predict the response time of a component-based application by predicting the response time of each individual component, using a linear analytical model. Bao et al. extend this idea into the context of VMs and microservices. An analytical model is made, where the details of the model are different, but the linear computation of response time is very similar. An important characteristic of these models is that they use the sum of individual components to come to the overall performance of an application. In the context of microservice-based applications, this may be used to individually profile each microservice and then use the sum of the profiles to come to a full application profile.

4.2 The Cyber-Physical Systems Context

A CPS application is often held to rigid performance requirements. The performance requirements are generally made up of resource usage requirements, as well as latency requirements. This means that both profiling and performance prediction will likely need to include both resource utilization of microservices, as well as some way to identify traces of requests through the application. These traces represent paths through the application that represent latency requirements. Both Chen et al. [14] and Bao et al. [11] discuss profiles for models that try to predict the end-to-end tail latency of requests through an application, by analyzing the processing time of an application. The focus of these methods lies on the end-to-end latencies of requests. However, it is useful to analyze subtraces within a CPS if a latency requirement is defined within a system. Various works also discuss resource utilization. CPU and memory utilization metrics for both applications, as well as systems, are processed by most works that perform profiling or performance prediction, as shown in Table 2.

The literature displayed in this literature review has a focus on profiling, performance prediction, and design space exploration in cloud environments. This is the case because most work in this area is done within this context. Research into performance prediction for CPSs is lacking in this area as well. The expectation is that the work shown in the concepts for the cloud environment can be at least partially used in the context of CPSs. This is supported by Section 4.1, where profiling and prediction between applications in data centers is discussed and where is shown that it applies to microservices in the cloud as well. Nonetheless, the nature and strictness of performance requirements of CPSs may differ from those of applications in the cloud, and CPSs may be subject to unexpected behavior caused by real-world components. Therefore, further research is needed in this area.

Another distinction may be in the run-time environment of CPSs. Section 3.3 describes that several studies use reinforcement learning to redistribute microservices based on cloud environment changes and request arrival pattern changes for an application. The expectation in cyber-physical systems is that, while request arrival patterns may change, the environment is not as fluid. Therefore, the redistribution of microservices may not be as essential as in cloud environments, where change is much more common.

It should be noted that a CPS can be a complex distributed system featuring a variety of compute nodes [5]. They range from big Intel machines to small embedded boards designed for real-time control applications. A challenge within the context of CPSs can be that the compute nodes, such as the embedded boards, are resource-constrained and must operate within short deadlines of micro- or milliseconds. These parts of the system typically require static analysis of timing and schedulability rather than measurement-based performance prediction as discussed in the literature shown here. These compute nodes may be the cause of the more strict latency requirements of a CPS. This highlights that the related work discussed in this literature study does not necessarily apply to all aspects of a CPS, but only to the software-dominated components. These special hardware components fall out of the scope of this work, but should be taken into consideration when hardware dimensioning in the context of CPSs is performed.

4.3 Hardware Dimensioning

From the literature, a method of hardware dimensioning can be derived that can be split up into four steps:

(1) Make a profile of an application and system.
(2) Develop a model that maps the deployment of microservices over a cluster of nodes to the performance of a microservice-based application. Input of the prediction is the application profile and the system profile.
(3) Use the performance prediction to explore the design space for optimal, near-optimal, or acceptable deployment of microservices over nodes.
(4) Choose the number of servers for which the discovered solution is sufficient.

The first two steps of hardware dimensioning, profiling and performance prediction, have to be done offline. The goal is to predict the amount of hardware necessary to run an application while satisfying performance requirements, and, therefore, the system is not yet available. However, aspects of online profiling and performance prediction can still be used in offline profiling. As shown in Table 1 and Table 2, the purpose and metrics of both online and offline methods can be similar, and ideas of online profiling are also used in offline profiling. For example, both Lv et al. [35] and Han et al. [24] build a profile based on communication overhead within an application with the purpose of optimizing its resource usage. The former builds an online profile, and the latter an offline profile. The method through which the profile is then leveraged differs, as the resource assignment is done dynamically [35], or before the application is deployed [24].

The current literature does not consider profiling in the context of CPSs, as shown in Table 1. Although it could be the case that the profiling methods used in the context of applications and microservices in the cloud can be applied directly in the CPS context, no experiments have yet been performed to verify this is the case.

Similarly, performance prediction is explored thoroughly in the literature, but only in a cloud or data center environment.

Machine learning models are generally not suitable for offline performance prediction. The literature that performs online prediction often uses machine learning models, because there is much data available in a specific system context. A characteristic of performance prediction in hardware dimensioning is that there is not much historical data available yet, and therefore it will likely not be a useful model type in this context. Additionally, black box machine learning models do not give insight into the workings of an application, but predict its performance without a method of how. Analytical models are likely more useful in this context, since they require less data, and can give better insight into the characteristics of an application. While that is not the main goal of hardware dimensioning, it can still aid in the development process of an application.

A performance prediction in hardware dimensioning does not need to be precise. The goal is to give a scientifically backed supporting structure for hardware dimensioning. Therefore, the profile and model can stay simple, to keep the process simple as well. A 20% accuracy may be reasonable, although it will depend on the predictability of the behavior of an application, as well as the system that it runs on. With increased unexpected behavior, the prediction accuracy may go down significantly. As the performance model may stay simple, the linear analytical models that are common [11, 14] could be a good approach to hardware dimensioning in this context.

The third step of hardware dimensioning includes finding an optimal deployment of microservices over a set of nodes. The literature provides various works that look into this optimization problem [16, 19, 30, 46] and find optimal methods for specific optimization problems. Research in this area is relatively mature and various search strategies have been tried, like greedy heuristic algorithms, reinforcement learning methods, and evolutionary algorithms. The optimal method is dependent on the type of model and predicted metrics.

There is no structured approach yet for hardware dimensioning in the context of microservice-based applications in the cyber-physical space. Further research is necessary in this area to verify that the four steps of hardware dimensioning identified in this literature study can be applied in that specific context. Furthermore, none of the related work discusses hardware dimensioning as a purpose of profiling and performance prediction. Therefore, further research can be conducted into profiling and performance prediction as a basis for hardware dimensioning.

5 CONCLUSION

This literature study looks at the current state-of-the-art and research directions in hardware dimensioning for microservice applications in cyber-physical systems. This study splits hardware dimensioning into (i) application and system profiling, (ii) performance prediction for microservice-based applications, and (iii) design space exploration in the initial distribution of microservices over a set of nodes. It then discusses the findings for each area and explains the process in which they can be used to perform hardware dimensioning in the context of microservice-based applications in cyber-physical systems.

To perform hardware dimensioning, both an application profile and a system profile should be made to form the input for a prediction model. Various analytical performance prediction models use the sum of the performance of different components of an application as a viable performance prediction for the full application. This indicates that the sum of the performance of microservices can be suitable as a performance prediction of the full application. For example, a prediction of the CPU usage of a node may be made based on the sum of the CPU usage of each microservice that runs on it.

Additionally, an application profile often consists of communication overhead between services and contention between services. These two characteristics highlight an optimization trade-off in the optimal deployment of microservices over nodes: To minimize communication, microservices should be deployed on the same node as much as possible. To minimize interference, microservices should each be spread out. This indicates an optimization problem that can be solved with design space exploration. The optimum will be somewhere in between all microservices on one node, and each microservice being deployed on a separate node.

Although this literature study identifies four steps in the hardware dimensioning of microservice-based applications in the cyber-physical space, no work showcases a structured approach to hardware dimensioning a microservice-based application. Additionally, current literature does not discuss profiling and performance prediction in the context of cyber-physical systems. Future work has to be conducted to definitively conclude that the various methods of profiling, performance prediction, and design space exploration that are shown in this literature study are also applicable in the context of microservice-based applications in Cyber-Physical Systems.

The area of Design Space Exploration in the context of microservice deployment over a set of nodes is done extensively and is fairly mature in a cloud environment. However, all research is done in the cloud, and future research may be done on the applicability of DSE methods in cyber-physical systems.

ACKNOWLEDGMENTS

This literature study was produced as part of the XM_0131 course at the Vrije Universiteit Amsterdam in cooperation with the University of Amsterdam. It was supervised by Prof. Dr. Benny Akesson, professor by Special Appointment with the University of Amsterdam & Senior Researcher at TNO-ESI, and Ben Pronk, System Architect at TNO-ESI.

REFERENCES

[1] 2006. SPEC CPU. Accessed: Oct. 2, 2023 [Online]. Available: http://www.spec.org/cpu2006/index.html.
[2] 2019. gRPC. Accessed: Aug. 14, 2023 [Online]. Available: https://grpc.io/.
[3] 2023. Production-Grade Container Orchestration. Accessed: Oct. 4, 2023 [Online]. Available: https://kubernetes.io/.
[4] Madhura Adeppady, Paolo Giaccone, Holger Karl, and Carla Fabiana Chiasserini. 2023. Reducing microservices interference and deployment time in resource-constrained cloud systems. *IEEE Transactions on Network and Service Management* (2023).
[5] Benny Akesson, Mitra Nasri, Geoffrey Nelissen, Sebastian Altmeyer, and Robert I Davis. 2022. A comprehensive survey of industry practice in real-time systems. *Real-Time Systems* 58, 3 (2022), 358–398.
[6] Isil Karabey Aksakalli, Turgay Celik, Ahmet Burak Can, and Bedir Tekinerdogan. 2021. Systematic approach for generation of feasible deployment alternatives for microservices. *IEEE Access* 9 (2021), 29505–29529.

[7] Omar Al-Debagy and Peter Martinek. 2018. A comparative review of microservices and monolithic architectures. In *2018 IEEE 18th International Symposium on Computational Intelligence and Informatics (CINTI)*. IEEE, 000149–000154.

[8] Thomas Bäck and Hans-Paul Schwefel. 1993. An overview of evolutionary algorithms for parameter optimization. *Evolutionary computation* 1, 1 (1993), 1–23.

[9] Radhakisan Baheti and Helen Gill. 2011. Cyber-physical systems. *The impact of control technology* 12, 1 (2011), 161–166.

[10] Simonetta Balsamo, Antinisca Di Marco, Paola Inverardi, and Marta Simeoni. 2004. Model-based performance prediction in software development: A survey. *IEEE Transactions on Software Engineering* 30, 5 (2004), 295–310.

[11] Liang Bao, Chase Wu, Xiaoxuan Bu, Nana Ren, and Mengqing Shen. 2019. Performance modeling and workflow scheduling of microservice-based applications in clouds. *IEEE Transactions on Parallel and Distributed Systems* 30, 9 (2019), 2114–2129.

[12] Grzegorz Blinowski, Anna Ojdowska, and Adam Przybyłek. 2022. Monolithic vs. microservice architecture: A performance and scalability evaluation. *IEEE Access* 10 (2022), 20357–20374.

[13] L Chen, Y Xu, Z Lu, J Wu, K Gai, PCK Hung, and M Qiu. 2020. IoT microservice deployment in edge-cloud hybrid environment using reinforcement learning. IEEE Internet Things J. 8 (16), 12610–12622 (2021).

[14] Shiping Chen, Yan Liu, Ian Gorton, and Anna Liu. 2005. Performance prediction of component-based applications. *Journal of Systems and Software* 74, 1 (2005), 35–43.

[15] Kalyanmoy Deb and Himanshu Jain. 2013. An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part I: solving problems with box constraints. *IEEE transactions on evolutionary computation* 18, 4 (2013), 577–601.

[16] Christina Delimitrou and Christos Kozyrakis. 2013. ibench: Quantifying interference for datacenter applications. In *2013 IEEE international symposium on workload characterization (IISWC)*. IEEE, 23–33.

[17] Anh Vu Do, Junliang Chen, Chen Wang, Young Choon Lee, Albert Y Zomaya, and Bing Bing Zhou. 2011. Profiling applications for virtual machine placement in clouds. In *2011 IEEE 4th international conference on cloud computing*. IEEE, 660–667.

[18] Nicola Dragoni, Saverio Giallorenzo, Alberto Lluch Lafuente, Manuel Mazzara, Fabrizio Montesi, Ruslan Mustafin, and Larisa Safina. 2017. Microservices: yesterday, today, and tomorrow. *Present and ulterior software engineering* (2017), 195–216.

[19] Kaihua Fu, Wei Zhang, Quan Chen, Deze Zeng, and Minyi Guo. 2021. Adaptive resource efficient microservice deployment in cloud-edge continuum. *IEEE Transactions on Parallel and Distributed Systems* 33, 8 (2021), 1825–1840.

[20] Kaihua Fu, Wei Zhang, Quan Chen, Deze Zeng, Xin Peng, Wenli Zheng, and Minyi Guo. 2021. Qos-aware and resource efficient microservice deployment in cloud-edge continuum. In *2021 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 932–941.

[21] Shivakumar R Goniwada and Shivakumar R Goniwada. 2022. Observability. *Cloud Native Architecture and Design: A Handbook for Modern Day Architecture and Design with Enterprise-Grade Examples* (2022), 661–676.

[22] Johannes Grohmann, Patrick K Nicholson, Jesus Omana Iglesias, Samuel Kounev, and Diego Lugones. 2019. Monitorless: Predicting performance degradation in cloud applications with machine learning. In *Proceedings of the 20th international middleware conference*. 149–162.

[23] Johannes Grohmann, Martin Straesser, Avi Chalbani, Simon Eismann, Yair Arian, Nikolas Herbst, Noam Peretz, and Samuel Kounev. 2021. Suanming: Explainable prediction of performance degradations in microservice applications. In *Proceedings of the ACM/SPEC International Conference on Performance Engineering*. 165–176.

[24] Jungsu Han, Yujin Hong, and Jongwon Kim. 2020. Refining microservices placement employing workload profiling over multiple kubernetes clusters. *IEEE access* 8 (2020), 192543–192556.

[25] Robert Harrison, Daniel Vera, and Bilal Ahmad. 2016. Engineering methods and tools for cyber–physical automation systems. *Proc. IEEE* 104, 5 (2016), 973–985.

[26] 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)*. IEEE, 632–640.

[27] Marcus Hilbrich and Fabian Lehmann. 2022. Discussing Microservices: Definitions, Pitfalls, and their Relations. In *2022 IEEE International Conference on Services Computing (SCC)*. IEEE, 39–44.

[28] Pooyan Jamshidi, Claus Pahl, Nabor C Mendonça, James Lewis, and Stefan Tilkov. 2018. Microservices: The journey so far and challenges ahead. *IEEE Software* 35, 3 (2018), 24–35.

[29] Zhaolong Jian, Xueshuo Xie, Yaozheng Fang, Yibing Jiang, Tao Li, and Ye Lu. 2023. DRS: A Deep Reinforcement Learning enhanced Kubernetes Scheduler for Microservice-based System. (2023).

[30] Christina Terese Joseph and K Chandrasekaran. 2020. IntMA: Dynamic interaction-aware resource allocation for containerized microservices in cloud environments. *Journal of Systems Architecture* 111 (2020), 101785.

[31] Gopal Kakivaya, Lu Xun, Richard Hasha, Shegufta Bakht Ahsan, Todd Pfleiger, Rishi Sinha, Anurag Gupta, Mihail Tarta, Mark Fussell, Vipul Modi, et al. 2018. Service fabric: a distributed platform for building microservices in the cloud. In *Proceedings of the thirteenth EuroSys conference*. 1–15.

[32] 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)*. IEEE, 363–369.

[33] Bowen Li, Xin Peng, Qilin Xiang, Hanzhang Wang, Tao Xie, Jun Sun, and Xuanzhe Liu. 2022. Enjoy your observability: an industrial survey of microservice tracing and analysis. *Empirical Software Engineering* 27 (2022), 1–28.

[34] Carolina Villarreal Lozano and Kavin Kathiresh Vijayan. 2020. Literature review on cyber physical systems design. *Procedia manufacturing* 45 (2020), 295–300.

[35] Wenkai Lv, Quan Wang, Pengfei Yang, Yunqing Ding, Bijie Yi, Zhenyi Wang, and Chengmin Lin. 2022. Microservice deployment in edge computing based on deep Q learning. *IEEE Transactions on Parallel and Distributed Systems* 33, 11 (2022), 2968–2978.

[36] Wubin Ma, Rui Wang, Yuanlin Gu, Qinggang Meng, Hongbin Huang, Su Deng, and Yahui Wu. 2021. Multi-objective microservice deployment optimization via a knowledge-driven evolutionary algorithm. *Complex & Intelligent Systems* 7 (2021), 1153–1171.

[37] P Read Montague. 1999. Reinforcement learning: an introduction, by Sutton, RS and Barto, AG. *Trends in cognitive sciences* 3, 9 (1999), 360.

[38] Anthony J Myles, Robert N Feudale, Yang Liu, Nathaniel A Woody, and Steven D Brown. 2004. An introduction to decision tree modeling. *Journal of Chemometrics: A Journal of the Chemometrics Society* 18, 6 (2004), 275–285.

[39] Haoran Qiu, Subho S Banerjee, Saurabh Jha, Zbigniew T Kalbarczyk, and Ravishankar K Iyer. 2020. {FIRM}: An intelligent fine-grained resource management framework for {SLO-Oriented} microservices. In *14th USENIX symposium on operating systems design and implementation (OSDI 20)*. 805–825.

[40] Joy Rahman and Palden Lama. 2019. Predicting the end-to-end tail latency of containerized microservices in the cloud. In *2019 IEEE International Conference on Cloud Engineering (IC2E)*. IEEE, 200–210.

[41] Wojciech Samek, Grégoire Montavon, Sebastian Lapuschkin, Christopher J Anders, and Klaus-Robert Müller. 2021. Explaining deep neural networks and beyond: A review of methods and applications. *Proc. IEEE* 109, 3 (2021), 247–278.

[42] Adalberto R Sampaio, Julia Rubin, Ivan Beschastnikh, and Nelson S Rosa. 2019. Improving microservice-based applications with runtime placement adaptation. *Journal of Internet Services and Applications* 10, 1 (2019), 1–30.

[43] Alex J Smola and Bernhard Schölkopf. 2004. A tutorial on support vector regression. *Statistics and computing* 14 (2004), 199–222.

[44] Gagan Somashekar and Anshul Gandhi. 2021. Towards optimal configuration of microservices. In *Proceedings of the 1st Workshop on Machine Learning and Systems*. 7–14.

[45] Davide Taibi, Valentina Lenarduzzi, Claus Pahl, and Andrea Janes. 2017. Microservices in agile software development: a workshop-based study into issues, advantages, and disadvantages. In *Proceedings of the XP2017 Scientific Workshops*. 1–5.

[46] Bing Tang, Feiyan Guo, Buqing Cao, Mingdong Tang, and Kuanching Li. 2022. Cost-aware Deployment of Microservices for IoT Applications in Mobile Edge Computing Environment. *IEEE Transactions on Network and Service Management* (2022).

[47] Freddy Tapia, Miguel Ángel Mora, Walter Fuertes, Hernán Aules, Edwin Flores, and Theofilos Toulkeridis. 2020. From monolithic systems to microservices: A comparative study of performance. *Applied sciences* 10, 17 (2020), 5797.

[48] Bryan A Tolson and Christine A Shoemaker. 2007. Dynamically dimensioned search algorithm for computationally efficient watershed model calibration. *Water Resources Research* 43, 1 (2007).

[49] Bram Van der Sanden, Yonghui Li, Joris van den Aker, Benny Akesson, Tjerk Bijlsma, Martijn Hendriks, Kostas Triantafyllidis, Jacques Verriet, Jeroen Voeten, and Twan Basten. 2021. Model-driven system-performance engineering for cyber-physical systems. In *Proceedings of the 2021 International Conference on Embedded Software*. 11–22.

[50] Rafael Weingärtner, Gabriel Beims Bräscher, and Carlos Becker Westphall. 2015. Cloud resource management: A survey on forecasting and profiling models. *Journal of Network and Computer Applications* 47 (2015), 99–106.

[51] Sanford Weisberg. 2005. *Applied linear regression*. Vol. 528. John Wiley & Sons.

[52] E. Wilde and C. Pautasso. 2011. REST API Tutorial. Accessed: Aug. 14, 2023 [Online]. Available: https://restfulapi.net/.

[53] Yanqi Zhang, Weizhe Hua, Zhuangzhuang Zhou, G Edward Suh, and Christina Delimitrou. 2021. Sinan: ML-based and QoS-aware resource management for cloud microservices. In *Proceedings of the 26th ACM international conference on architectural support for programming languages and operating systems*. 167–181.

Table 1: A characterization of the profile and prediction methods and context of the related work

Related works	Context		Profile
	Environment	Purpose	Application/System Profiling
Do et al. [17]	VMs in the cloud	Optimize resource usage	Both
Han et al. [24]	MSs in the cloud	Optimize resource usage	Application
Bao et al. [11]	MSs in the cloud	Optimize cloud costs	Both
Aksakalli et al. [6]	MSs in the cloud	Predict QoS satisfaction	Both
Delimitrou et al. [16]	Applications in datacenters	Minimize interference	Application
Joseph et al. [30]	MSs in the cloud	Minimize application response time	Both
Chen et al. [14]	Applications in datacenters	Predict QoS satisfaction	Both
Adeppady et al. [4]	MSs in the cloud	Optimize resource usage	Both
Ma et al. [36]	MSs in distributed resource centers	Minimize MS idle rate	Both
Lv et al. [35]	MSs in edge computing	Optimize resource usage	Application
Grohmann et al. [23]	MSs in the cloud	Prevent performance degradation	Application
Rahman et al. [40]	MSs in the cloud	Predict end-to-end latency	Both
Zhang et al. [53]	MSs in the cloud	Predict QoS satisfaction	Both
Fu et al. [19]	MSs in cloud-edge continuum	Optimize resource usage	Both

Table 2: A characterization of the profile and prediction methods and context of the related work

Related works	Profile	Profile metrics					
	Offline/Online	Communication overhead	Contention	Application specific metrics	CPU	Memory	Network
Do et al. [17]	Offline			X	X	X	
Han et al. [24]	Offline	X			X	X	X
Bao et al. [11]	Offline				X	X	
Aksakalli et al. [6]	Offline	X			X	X	X
Delimitrou et al. [16]	Offline		X		X	X	X
Joseph et al. [30]	Offline	X			X	X	X
Chen et al. [14]	Offline				X	X	
Adeppady et al. [4]	Both		X		X	X	
Ma et al. [36]	Online				X	X	
Lv et al. [35]	Online	X			X	X	X
Grohmann et al. [23]	Online	X			X	X	X
Rahman et al. [40]	Online		X			X	X
Zhang et al. [53]	Online	X	X		X		X
Fu et al. [19]	Both	X	X		X	X	X

Table 3: A characterization of the profile and prediction methods and context of the related work

Related works	Prediction model	
	Model type	Predicted metric
Do et al. [17]	Analytical	Performance difference
Han et al. [24]	None	None
Bao et al. [11]	Analytical	Processing time/cost
Aksakalli et al. [6]	None	None
Delimitrou et al. [16]	None	None
Joseph et al. [30]	None	None
Chen et al. [14]	Analytical	Processing time
Adeppady et al. [4]	Analytical + ML	MS throughput
Ma et al. [36]	None	MS idle rate
Lv et al. [35]	None	Application response time/load balance
Grohmann et al. [23]	Analytical + ML	Performance degradation
Rahman et al. [40]	ML	Tail latency
Zhang et al. [53]	ML	QoS satisfaction
Fu et al. [19]	ML	Latency/throughput