Zietsch, Jakob; Kulaga, Rafal; Held, Harald; Herrmann, Christoph; Thiede, Sebastian

Multi-layer edge resource placement optimization for factories

Journal of Intelligent Manufacturing

Provided in Cooperation with:
Springer Nature

Suggested Citation: Zietsch, Jakob; Kulaga, Rafal; Held, Harald; Herrmann, Christoph; Thiede, Sebastian (2023) : Multi-layer edge resource placement optimization for factories, Journal of Intelligent Manufacturing, ISSN 1572-8145, Springer US, New York, NY, Vol. 35, Iss. 2, pp. 825-840, https://doi.org/10.1007/s10845-022-02071-3

Journal of Intelligent Manufacturing (2024) 35:825–840
https://doi.org/10.1007/s10845-022-02071-3

Multi-layer edge resource placement optimization for factories

Jakob Zietsch[1] · Rafal Kulaga[2] · Harald Held[2] · Christoph Herrmann[1] · Sebastian Thiede[3]

Received: 29 June 2022 / Accepted: 24 December 2022 / Published online: 28 January 2023

Abstract

Introducing distributed computing paradigms to the manufacturing domain increases the difficulty of designing and planning an appropriate IT infrastructure. This paper proposes a model and solution approach addressing the conjoint application and IT resource placement problem in a factory context. Instead of aiming to create an exact model, resource requirements and capabilities are simplified, focusing on usability in the planning and design phase for industrial use cases. Three objective functions are implemented: minimizing overall cost, environmental impact, and the number of devices. The implications of edge and fog computing are considered in a multi-layer model by introducing five resource placement levels ranging from on-device, within the production system, within the production section, within the factory (on-premise), to the cloud (off-premise). The model is implemented using the open-source modeling language Pyomo. The solver SCIP is used to solve the NP-hard integer programming problem. For the evaluation of the optimization implementation a benchmark is created using a sample set of scenarios varying the number of possible placement locations, applications, and the distribution of assigned edge recommendations. The resulting execution times demonstrate the viability of the proposed approach for small (100 applications; 100 locations) and large (1000 applications, 1000 scenarios) instances. A case study for a section of a factory producing electronic components demonstrates the practical application of the proposed approach.

Keywords Edge computing · Resource placement · IT infrastructure optimization · Application allocation

Introduction

The transformation process towards edge-driven smart manufacturing is still in its infancy, facing many challenges (Basir et al., 2019). The introduction of distributed computing paradigms conflicts with the prevailing legacy systems historically dominated by Operational Technology (OT). Devices like Programmable Logic Controllers (PLCs) are embedded in a strict hierarchy, often referred to as the automation pyramid, with data and information mostly traveling northbound and aggregated on the higher levels (Vogel-Heuser et al., 2015). The design maxims were availability and reliability, and the computation capacity was planned to allow the execution of the predefined automation tasks at any given moment following strict cycle times. While both design maxims are still crucial, the transition to smart manufacturing systems requires adaptability and flexibility (Brettel et al., 2016). A suitable IT infrastructure providing sufficient computational capacity in the right place enables a successful transition to modern cyber-physical production systems (CPPS).

✉ Jakob Zietsch
j.zietsch@tu-braunschweig.de

Rafal Kulaga
rafal.kulaga@tum.de

Harald Held
harald.held@siemens.com

Christoph Herrmann
c.herrmann@tu-braunschweig.de

Sebastian Thiede
s.thiede@utwente.nl

[1] Chair of Sustainable Manufacturing and Life Cycle Engineering, Institute of Machine Tools and Production Technology, Technische Universität Braunschweig, Brunswick, Germany

[2] Corporate Technology, Siemens AG, Munich, Germany

[3] Chair of Manufacturing Systems, Department of Design, Production and Management, University of Twente, Enschede, The Netherlands

Upgrading an IT infrastructure is a resource-intensive process and should be critically assessed (Hertel & Wiesent, 2013). Purchasing or leasing an IT resource can be a significant investment considering the scale of an entire factory. Additionally, each IT resource affects the overall environmental impact of the factory due to the energy consumption of each IT resource during production, use, and recycling (Hischier et al., 2015). Hence, when striving towards smart and sustainable manufacturing, the cost and environmental impact of each IT resource should be considered (Thiede, 2021). These two aspects depend primarily on the hardware specifications, like processing power, memory, or storage, often referred to as the capacity of an IT resource. Considering the capacity in an early planning and design stage ensures an adequate resource selection before installation, avoiding costly change and transformation processes.

However, ensuring sufficient capacity at the right places within a highly interconnected factory network where all devices can communicate with one another poses a significant challenge to an IT infrastructure designer and planner. Three questions are in the center of the planning problem illustrated in Fig. 1:

- What kind of IT resources should be selected?
- Where should these IT resources be installed (vertical/horizontal dimension)?
- Which applications should be deployed on which level?

The vertical dimension refers to how these IT resources are integrated into the network topology of the factory. The placement level is essential due to the influence of the network, e.g., bandwidth limitation or network delay. Cloud computing providers offer to provide the required capacity dynamically; however, when considering the vast amount of production data on the scale of a factory aggregating raw data streams in a central location is time and cost-intensive and, therefore, often not feasible, compare (Mourtzis et al., 2016). Furthermore, the network load can negatively influence the communication delay added with each network hop, which is a disadvantage of centralized computing paradigms (Basir et al., 2019). Furthermore, the potentially varying delays are a challenge for traditional automation applications, which often have real-time execution requirements that need to be guaranteed for the safe and reliable operation of the production systems (Vogel-Heuser et al., 2015). Additionally, in the context of smart factories, running the analytical processing in (near) real-time becomes increasingly important (Trinks & Felden, 2018), paving the way for, e.g., real-time task processing for spinning cyber-physical production systems (Yin et al., 2020).

Distributed computing paradigms enable the handling of large quantities of data and reduce the network delay through the decentralization of data processing instead of central aggregation (Qi & Tao, 2019; Trinks & Felden, 2018; Ismail et al., 2015). The data is processed at the network's edge, hence the paradigm's name: edge computing. The term fog computing is another form of distributed computing paradigm originating from the perspective of a network provider and is often used interchangeably with edge computing (Aazam et al., 2018; Hong & Varghese, 2019). We consider fog computing a middle layer between edge and cloud, representing more powerful computational entities like a server cluster or an on-premise cloud. Edge computing platforms will allow applications to be deployed on any resource with computing and communication capabil-

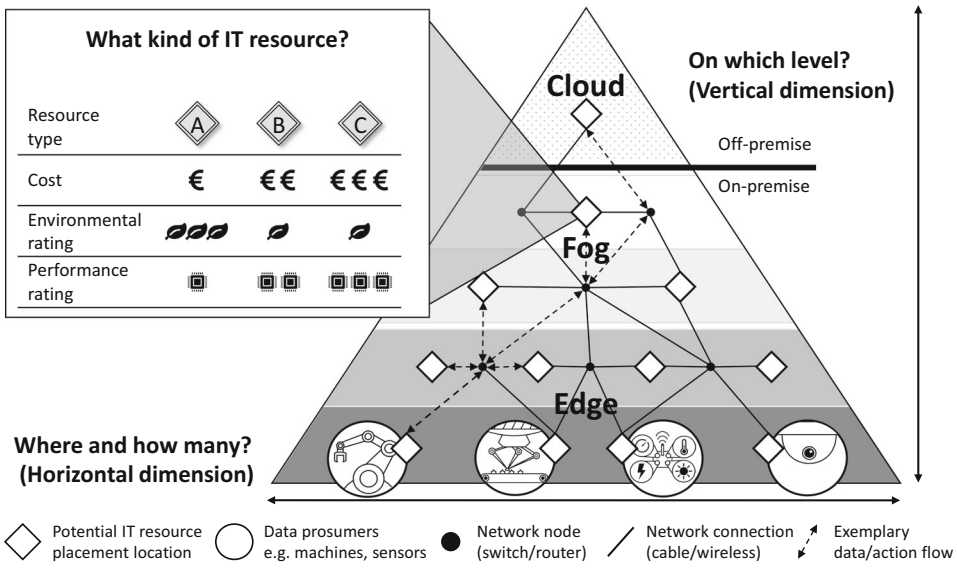

Fig. 1 Resource placement problem in edge-cloud continuum from Zietsch et al. (2020)

ities (Noghabi et al., 2020), increasing the flexibility and agility of the entire factory network (Chen et al., 2018).

This paper proposes an approach, a model, and an implementation to address the multi-layer IT resource placement problem for factories. The structure of the paper is as follows. First, the relation between the proposed work and previous publications is clarified. Second, the proposed model and the framework in which it is embedded are described. Third, the implementation is evaluated by simulating 16 test scenarios ranging from small to large scale. Fourth, the proposed work is applied in a case study. Finally, the experiences and limitations of the proposed multi-layer model are discussed, concluding this work.

Related work

Yousefpour et al. (2019) identified the planning and design of networks incorporating the distributed edge and fog computing paradigms as an essential topic with limited available publications compared to, e.g., resource management, provisioning, and operation. More recent literature reviews like Hong and Varghese (2019), Ghobaei-Arani et al. (2020), Kumar et al. (2022) reveal a further increase in interest in dynamic resource provisioning or application allocation. However, the lack of publications addressing the physical placement of IT resources, especially in a manufacturing context, persists.

There are various approaches outside of smart factories addressing the placement of IT resource in other contexts like mobile edge computing (Zhang et al., 2019; Wang et al., 2019), smart cities (Xu et al., 2016; Yin et al., 2017), vehicular networks (Guo et al., 2016; Mao et al., 2022). These contributions have in common that the network stretches over large areas, necessitating the geographical distance of nodes and users. The distance is either represented by a constraint or integrated into the target functions, which is necessary due to physical limitations, e.g., the range of a cell phone tower, the induced latency, or the added installation cost for additional cabling [compare Guo et al. (2016)]. Although the specific placement of IT resources in the factory (horizontal dimension) is of significant relevance, the geographical distance is of reduced importance since the areas that need to be covered and the movement of clients are more manageable compared to, e.g., vehicular networks.

Yin et al. (2017) propose a decision support framework for the provisioning of edge servers for online service providers optimizing the construction cost and bandwidth prices. The decision to place an additional edge server depends strongly on the number of users.

Lin and Yang (2018) establish an integer programming model for the cost-efficient deployment of a fog computing system at a logistics center populated with Automatic Guided Vehicles (AGVs), the IoT devices with resource demand. A meta-heuristic algorithm incorporating the discrete monkey algorithm is applied to determine the placement of gateway, fog devices, and edge devices, minimizing the overall installation cost. The focus on AGVs allowed the simplification of the resource demand such that an edge device can only support a set amount of AGVs with the fog devices and gateways following the same logic. However, the focus is a significant limitation when considering the heterogeneity of applications.

Jiang et al. (2021) introduce a k-means clustering algorithm for the deployment of edge computing nodes in the context of smart manufacturing. The proposed algorithm balances the network delay and computing resources deployment cost while considering the impact of spatial device distribution, device function, and computing capacity. Instead of offering a prior investigation and planning process, the algorithm focuses more on the possible node deployment during operation.

In summary, the previous research contributions do not address the planning phase of an IT infrastructure in the context of manufacturing. Although there are various examples of optimization algorithm implementations, they often require the availability of highly detailed information, which is challenging to provide when considering the required collaboration of multiple planning departments and the uncertainty involved in the planning phase. Primarily when considering that the deployment or placement of applications (sometimes called tasks) is often determined through minimizing or constraining latency as one of the essential advantages of edge/fog computing. Latency as a constraint in today's factory is highly relevant, e.g., time-sensitive automation tasks, but challenging to determine for other IoT solutions which might have other requirements like connectivity, data sensitivity, or data encapsulation.

For this work, the application deployment restrictions within a factory are consolidated in a so-called edge level that combines distance, latency, bandwidth, and data ownership in one value. Summarizing the latency requirements of the applications in the new edge level constraint allows omitting a latency-related term from the target function commonly part of similar models like the ones presented Lin and Yang (2018) or Wang et al. (2019). The consolidation simplifies the modeling for the practitioner of the multi-layer edge resource placement problem and the execution of multiple optimizations in a reasonable time frame. The edge level was first introduced in Zietsch et al. (2019) simplifying the vertical application placement within the edge-cloud continuum by allowing a streamlined modeling approach. The novel aspects and differences of this work can be summarized as follows:

– Targeting the factory environment shifting the focus from large amounts of moving clients (users) to fewer but more critical data sources modeling physical as well as logical placement constraints
– Focusing on the planning and re-planning phase for an IT infrastructure for the green field as well as brownfield applications
– Solving the NP-hard optimization challenge using MILP (Mixed integer linear programming) in a reasonable amount of time by utilizing the edge level as a constraint for a vertical application and resource placement

Multilayer model for resource placement

Methodology

In contrast to related publications regarding the optimized placement of IT resources in a manufacturing environment, the initialization of the developed model is embedded in a framework that enables an efficient determination of the required inputs. The framework's structure with its four process steps is illustrated in Fig. 2. Following these steps requires the availability of information about the production environment, applications, and IT resources, often mandating a high level of abstraction and deliberate simplification, limiting its application scope and the subsequent optimization to areas for which said information is obtainable (Zietsch et al., 2020). Even though it is possible to utilize the model disjoint from the framework, the concept of a predetermined edge level that constrains the application placement is an integral part of the optimization.

In the first step (1), experts from varying domains drive the initial modeling procedure. Software experts model the applications that can be deployed in the manufacturing environment. These applications, or *apps* in short, process data of specific types and sources, e.g., calculating and visualizing the Overall Equipment Efficiency (OEE), perform complex analytics to estimate the remaining useful lifetime of machinery, or provide advanced control functions. From an abstract point of view, an application is, therefore, something that takes data as input and outputs the results of specific computational tasks. Each application has resource requirements like CPU, RAM, and bandwidth and might be subject to specific demands like latency, data volume, or autonomy. The resource requirements determine which kind of IT resources shall be placed. These IT resources are modeled by a domain expert and added to a resource catalog accessible by the optimization algorithm.

In the second step (2), an expert selects apps from the application catalog intended to be deployed. Then, the expert attaches the App to a data producer since it is assumed to require a data stream from one or many data producers; in other words, the expert specifies the application's input values. Finally, a facility expert independently models these data producers as part of the factory structure. The factory is divided into factory buildings and systems containing multiple data producers. These elements can be considered potential locations for an IT resource, like an edge device or any appropriate computational device.

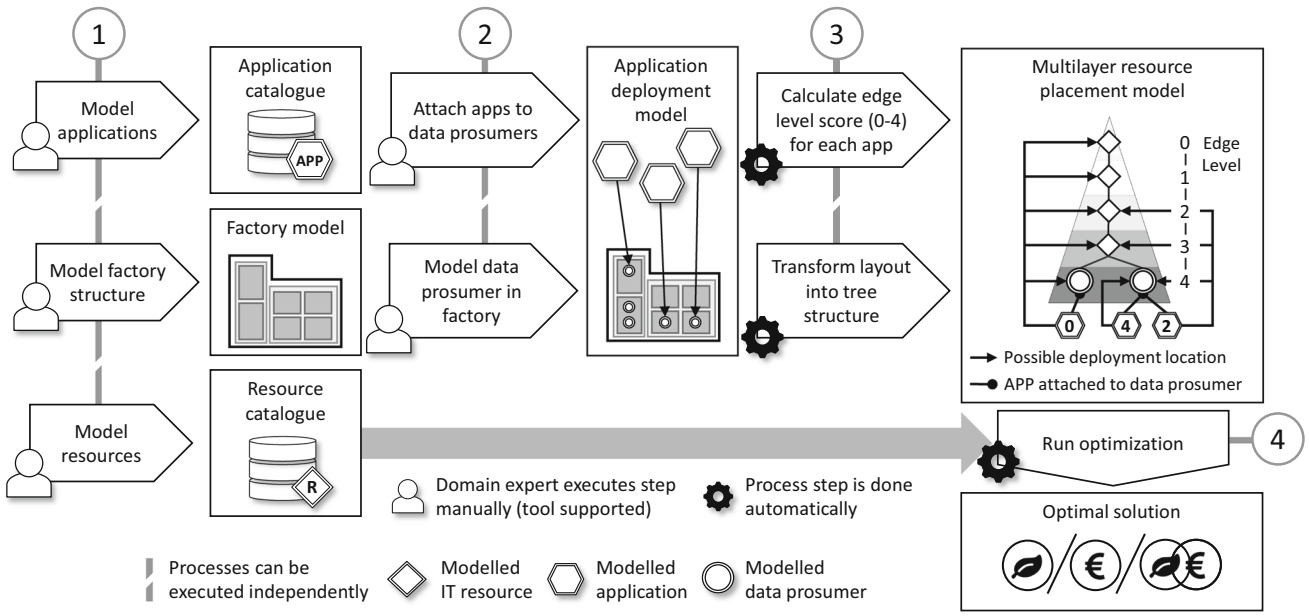

Fig. 2 Framework proposed in Zietsch et al. (2020) leading to the optimization execution

In the third step (3), the information about the specific demands enables an automatic calculation of an edge-level score ranging from zero to four. The score reflects a deployment recommendation for an application for the five levels in the edge-cloud continuum ranging from on-device-edge, edge, fog, cloudlet, or cloud level proposed in Zietsch et al. (2020). The calculation of a deployment restriction before the optimization execution is an essential difference compared to other optimization approaches found in the literature (compare Sect. 2). For brevity's sake, the specificities of the edge level calculation are not part of this work; please refer to Zietsch et al. (2020). The 2D factory structure is transformed into a tree structure with the cloud (level 0) at the top and the data producers at the bottom (level 4). All applications attached to data producers have an assigned edge level ranging from zero to four, restricting their placement in the tree.

In the fourth step (4), the optimization is executed with three distinct objectives: cost minimization, environmental impact minimization, or a combination of the two. The model containing the constraints and objectives is described in the following subsection.

Model

Sets and parameters

The hierarchical 2D factory layout, translated into a tree structure, captures the relational links between the locations. As illustrated in Fig. 3, each location (l) can be viewed as a slot where a resource (r) can be placed in order to complete computation tasks (Apps). A set of m locations $L = \{l_1, l_2, \ldots, l_m\}$ comprises all tree nodes ranging from the cloud (level 0) to the on-device edge (level 4). Parameter $\epsilon_l \in [0, 1, 2, 3, 4]$ stores information about the tree level of each location l.

The lowest layer extends the tree by a set of n applications $A = \{a_1, a_2, \ldots, a_n\}$ connected with appropriate location nodes. A performance requirement $\tau_a^{Performance}$ describes each application a, memory requirement τ_a^{Memory}, and bandwidth requirement $\tau_a^{Bandwidth}$. In addition, each application has an assigned edge level restricting its placement, $\epsilon_l \in [0, 1, 2, 3, 4]$ stores information about the tree level of each location a.

Resources required to accommodate the Apps are gathered in a set $R = \{r_1, r_2, \ldots, r_K\}$, where K denotes the total number of resource types (including different cloud services). Similar to the applications, each resource is denoted by a series of parameters. A resource's performance, memory, and bandwidth capacity are described by $\rho_r^{Performance}$, ρ_r^{Memory}, and $\rho_r^{Bandwidth}$. Furthermore, installation and usage of a resource result in capital expenditure ρ_r^{CapEx}, yearly operational expenditure ρ_r^{OpEx}, and environmental impact $\rho_r^{EcoImpact}$.

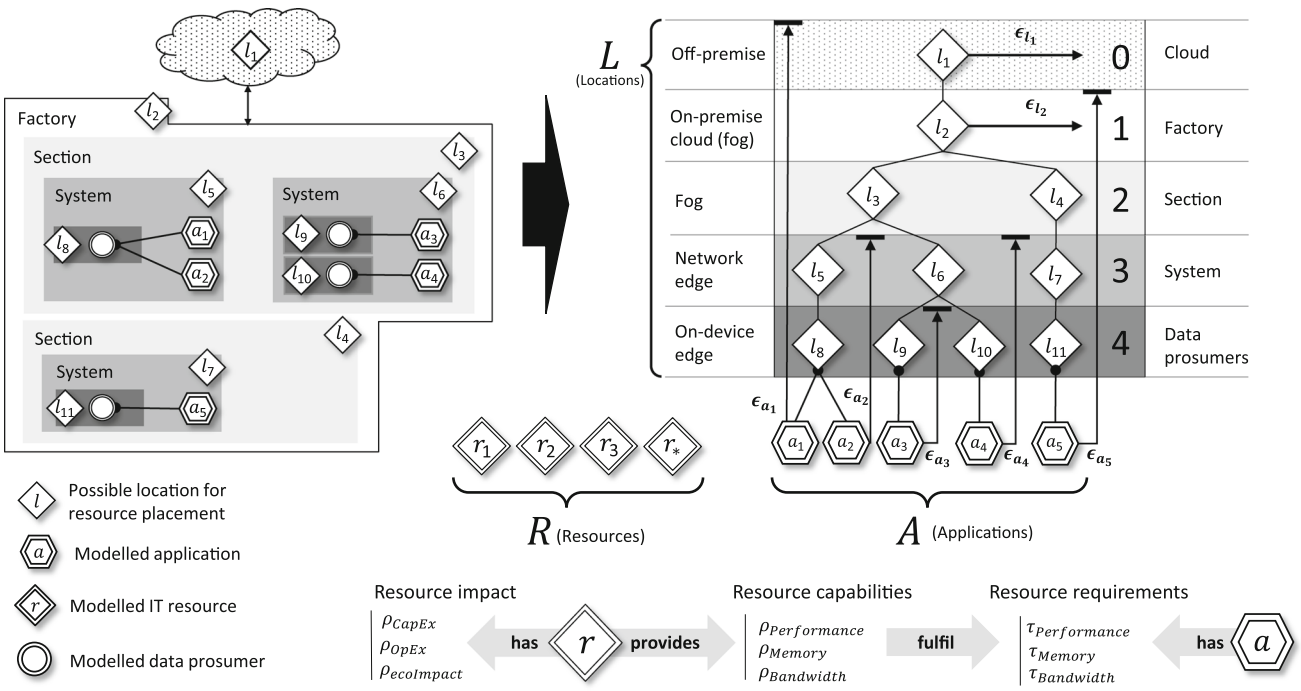

Fig. 3 Transforming the 2D factory layout into a tree structure

Optimization variables

The solver controls the decision variables $y_{a,r,l}$, which affects two auxiliary functions ($c_{r,l}$ and $\kappa_{r,l}$) that facilitate the model's description.

$$y_{a,r,l} = \begin{cases} 1, & \text{if a resource } r \text{ is allocated at the location} \\ l & \text{and runs application } a \\ 0, & \text{otherwise} \end{cases} \tag{3.1}$$

$c_{r,l}$ tracks the number of allocated applications for each pair of resources and location:

$$c_{r,l} = \sum_{a \in A} y_{a,r,l}, \ \forall r \in R, \ \forall l \in L. \tag{3.2}$$

The auxiliary variable $\kappa_{r,l}$ maps $c_{r,l}$ to a binary, which indicates if at least one application is placed at a specific location requiring the placement of a specific resource:

$$\kappa_{r,l} = \begin{cases} 1, & \text{if } c_{r,l} > 0 \\ 0, & \text{if } c_{r,l} = 0 \end{cases}, \ \forall r \in R, \ \forall l \in L. \tag{3.3}$$

Constraints

The following constraints are implemented to enforce compliance of the optimal solution with the requirements. Additional constraints can be added reflecting additional requirements like a budget limit, specific operating system, hardware inputs, and outputs for IT resources.

Resource level constraint restricts the placement of resour ces at certain positions. For example, cloud service cannot be placed on a machine or line.

$$\kappa_{r,l} \leq \rho_{l,r}^{ResLevel} \quad (r \in R, \ l \in L) \tag{3.4}$$

Allowed positions constraint restricts the placements of Apps to locations based on the assigned edge level. Each App, and each location, has a level score assigned to it. An app can only run on a location with the same or higher score. For example, if an app has a score of 4, it can run on level 4 (at the tree depth equal to 4) and nowhere else, as there are no different levels with higher scores. On the contrary, if an app has a score of 0, it can run on level 0 (cloud level) and lower levels (section, system, data source). Positions allowed for each App are encoded in the $\tau_{a,l}^{Allowed}$ variable, and the optimization constraint is enforced by:

$$\sum_{r \in R} y_{a,r,l} \leq \tau_{a,l}^{Allowed} \quad (a \in A, \ l \in L) \tag{3.5}$$

No app splitting constraint specifies that an application cannot be split and distributed to multiple resources. Instead, each App must be fully deployed and executed on one resource. This constraint ensures that none of the apps are skipped and enforces that each application is placed.

$$\sum_{r \in R} \sum_{l \in L} y_{a,r,l} = 1 \quad (a \in A) \tag{3.6}$$

Fulfill performance demand constraint assures that the resource selected for each application has a performance higher than or equal to the app requirement.

$$\sum_{r \in R} \sum_{l \in L} y_{a,r,l} \rho_r^{Performance} \geq \tau_a^{Performance} \quad (a \in A) \tag{3.7}$$

Fulfill memory demand constraint assures that the resource selected for each App has a memory capacity higher than or equal to the app requirement.

$$\sum_{r \in R} \sum_{l \in L} y_{a,r,l} \rho_r^{Memory} \geq \tau_a^{Memory} \quad (a \in A) \tag{3.8}$$

Fulfill bandwidth demand constraint assures that resource selected for each App has bandwidth capacity higher than or equal to the app requirement.

$$\sum_{r \in R} \sum_{l \in L} y_{a,r,l} \rho_r^{Bandwidth} \geq \tau_a^{Bandwidth} \quad (a \in A) \tag{3.9}$$

Resource performance limit constraint assures that total performance demand caused by the Apps (placed at a specific position) is lower than the performance capacity of a selected resource.

$$\sum_{a \in A} y_{a,r,l} \tau_a^{Performance} \leq \rho_r^{Performance} \quad (r \in R, \ l \in L) \tag{3.10}$$

Resource memory limit constraint is analogous to the resource performance limit constraint and ensures that the memory capacity of a selected resource is not exceeded.

$$\sum_{a \in A} y_{a,r,l} \tau_a^{Memory} \leq \rho_r^{Memory} \quad (r \in R, \ l \in L) \tag{3.11}$$

Resource bandwidth limit constraint is analogous to the resource performance limit constraint and ensures that the bandwidth capacity of a selected resource is not exceeded.

$$\sum_{a \in A} y_{a,r,l} \tau_a^{Bandwidth} \leq \rho_r^{Bandwidth} \quad (r \in R, \ l \in L) \tag{3.12}$$

One resource per location constraint enforces that, at most, one resource is placed at each location.

$$\sum_{r \in R} \kappa_{r,l} \leq 1 \quad (l \in L) \tag{3.13}$$

Objective functions

In the current version of the model, three optimization objectives are defined to minimize the overall cost (€), the environmental impact (🍃), and the number of placed resources.

However, it is essential to note that the model was specifically designed to facilitate new objectives. The previously defined constraints ensure that all applications can be deployed in the factory, an essential requirement for the whole model. Each solution the solver finds is viable, and the objective function allows these solutions to be compared.

In general, any desired goal that can be represented as a value allowing the solver to minimize or maximize said value can be added to the model to find the "best" solution. Examples of additional objectives that use only previously introduced variables are the minimization of operational expense (ρ_r^{OpEx}), upfront capital expense (ρ_r^{CapEx}), bandwidth consumption ($\rho_r^{Bandwidth}$), or maximization of performance ($\rho_r^{Performance}$).

More complex objectives, like maximizing the overall availability or flexibility of the resulting IT infrastructure, can also be added, resulting in the challenge of describing the desired objective as a function out of the scope of our work.

Since the resource catalog contains all resource types, it is necessary to distinguish between a set of resources that can be placed on-premise, denoted as R' (a subset of R), and a cloud service denoted as r_c to compute the overall cost and economic impact.

Overall cost minimization objective aims to minimize the overall cost of running on-premise resources and cloud service over a given period (T). A resource placement is associated with the upfront capital investment (ρ_r^{CapEx}) and operational expense (ρ_r^{OpEx}). A significant portion of the operational expense for the on-premise IT resources is due to the energy consumption of the resource. However, other reoccurring costs, e.g., maintenance or licensing, can also be included. Although there could be upfront capital expenditure for setting up cloud services (r_c), they are considered negligible compared to the operational expenses of the cloud service ($\rho_{r_c}^{OcpEx}$), which increases with each added App.

$$\min \left[\sum_{r \in R'} \sum_{l \in L} \kappa_{r,l} \rho_r^{CapEx} + T \sum_{r \in R'} \sum_{l \in L} \kappa_{r,l} \rho_r^{OpEx} + T \sum_{a \in A} \sum_{l \in L} y_{a,r_c,l} \rho_{r_c}^{OcpEx} \right] \tag{3.14}$$

Environmental impact minimization objective aims to minimize the overall environmental impact of running on-premise resources and cloud service over a given period (T). Each IT resource has an environmental impact. Accurately quantifying this impact is a significant challenge and continues to be subject to an ongoing debate about the sustainability of ICT in manufacturing (Thiede, 2021). The overall CO_2 emission during operation per year is used for the model. The overall CO_2 emissions per year ($\rho_r^{EcoImpact}$) can be calculated using each resource's energy consumption ($r \in R'$). The environmental impact of deploying Apps to the cloud depends on the amount of Apps and a separate impact factor ($\rho_{r_c}^{EcoImpact}$).

$$\min \left[T \left(\sum_{r \in R'} \sum_{l \in L} \kappa_{r,l} \rho_r^{EcoImpact} + \sum_{a \in A} \sum_{l \in L} y_{a,r_c,l} \rho_{r_c}^{EcoImpact} \right) \right] \tag{3.15}$$

The minimization of the number of resources aims to minimize the number of nodes (locations) on which resources are placed, including the cloud service.

$$\min \left[\sum_{r \in R} \sum_{l \in L} \kappa_{r,l} \right] \tag{3.16}$$

Combining multiple objectives

Executing the optimization with one of the above implemented objective functions yields one single optimal solution mathematically. However, minimizing cost, environmental impact, or the number of devices might not fulfill the desire of a user aiming for an optimal solution with contradicting goals or mandating the satisfaction of additional boundary conditions. In the model development process, three possible ways were identified that allow the consideration of potentially opposing objectives like € and 🍃 can be achieved:

1. *Adapt the resource catalog* in such a way that the solver can only select resources that are in a specific cost or environmental impact range
2. *Add an additional constraint* e.g. the overall capital investment cost must not exceed 10.000 €. The disadvantage of an additional constraint is that there might not exist any feasible solution that the solver can return
3. *Addressing multiple objectives* simultaneously transforming the problem into a multi-objective problem.

Whereas the first two options are relatively straightforward, a multi-objective problem can be addressed in various ways. However, to use the existing solver (SCIP), the two objectives, cost, and environmental impact, are combined by adding a priority indicator x that signifies the preference of the user alongside a conversion factor α. The priority indicator x ranges from zero to one. Three distinct values and their impact are presented as an example in Eq. (3.17).

$$x = \begin{cases} 1 & \text{Fully prioritize € disregard } \\ 0.5 & \text{€ and } \text{ are of equal importance} \\ 0 & \text{Fully prioritize } \text{ disregard €} \end{cases} \tag{3.17}$$

The conversion factor is required since the cost (€) and environmental impact ( ) can have different magnitudes. α is an additional parameter that the user can set. The resulting objective function combining € and is found in Eq. (3.18).

$$\min \left[x \left(T \sum_{r \in R} \sum_{l \in L} \kappa_{r,l} \rho_r^{OpEx} + \sum_{r \in R} \sum_{l \in L} \kappa_{r,l} \rho_r^{CapEx} \right) + (1-x) \alpha T \sum_{r \in R} \sum_{l \in L} \kappa_{r,l} \rho_r^{EcoImpact} \right] \tag{3.18}$$

Since the combination of multiple objective functions is not in the scope of this work, the value for α was not investigated further. For an initial test, it was calculated by dividing the mean cost and mean the environmental impact of all available resources in the repository over five years which had an insignificant effect on the run time of the optimization. Any objectives can be combined by introducing a priority indicator alongside a suitable conversion factor. From a factory planner's point of view, x can be represented as a slider whose two ends represent the two different objectives, cost, and environmental impact, respectively. The practitioners can experiment with different settings interactively and with ease.

Implementation in a factory context

Additional parameters and functions are defined to accommodate the specificities for optimizing an IT infrastructure in a factory.

The safety factor β can be set based on the factory's requirements. Then, β is applied to the individual resource requirements to create a safety margin for selecting the resources. Having a safety margin is considered good practice when planning capacity due to the various factors impacting an application's performance requirements, like a software upgrade, configuration adjustments, and unforeseen software or hardware interactions. The safety factor can be set to any value; however, the range from 1.1 - 1.5 is common (Wescott, 2013). The proposed safety margin is illustrated in Eq. (3.19).

$$\tau_{a,effective} = \tau_a \beta, \ \beta = \begin{cases} 1 & \text{No safety margin} \\ 1.2 & \text{Small safety margin} \\ 1.5 & \text{Medium safety margin} \\ 2 & \text{Large safety margin} \end{cases} \tag{3.19}$$

In addition, the $\tau_{a,l}^{Allowed}$ is used in the factory context to offer the user the ability to select and modify different deployment strategies. The content of the $\tau_{a,l}^{Allowed}$ matrix encodes which application can run at which location. The description, as mentioned earlier, of the "Allowed Positions Constraint" does not fully explain the rule that controls the construction of $\tau_{a,l}^{Allowed}$. The proposed implementation offers three strategies controlling which application is allowed to run at which location, which is illustrated in Fig. 4:

1. *Cloud-Fog-Edge* placement strategy means that the App can only be placed on the node in direct connection with the App (on-device edge - level 4), the factory level (fog - level 1), or the cloud (level 0) with the intermediate system and section resource locations (level 2 and 3) being unoccupied. The placement strategy is intended to simplify an edge or cloud implementation choice. In addition, the execution time of the optimization is reduced since the number of possible locations is decreased.
2. *Upward path only* strategy means that an App can only run at locations that can be accessed by going through the tree upwards only, starting from the App assigned to the factory object. In other words, if an app is not placed at the initially assigned factory object, a strictly upward path must exist from the initially assigned factory object to the newly selected location as long as the level allows it. This strategy is set as default for all optimizations forming a compromise of speed and optimization potential
3. *Entire Sub-tree* strategy allows the placement of an App on the entire subtree(s) of the node with the lowest level. This placement means that in the example Fig. 4 A_1 can be placed on any node in the entire tree. The increase in possible placement locations reduces the optimization speed, which might be impractical for scenarios with many applications.

The $\tau_{a,l}^{Allowed}$ (Eq. 3.20) is calculated according to the selected strategy for all applications before executing the solver.

$$\tau_{a,l}^{Allowed} = \begin{cases} 1, & \text{if } a \text{ is allowed at location } l \\ 0, & \text{if } a \text{ is not allowed at location } l \end{cases} \tag{3.20}$$

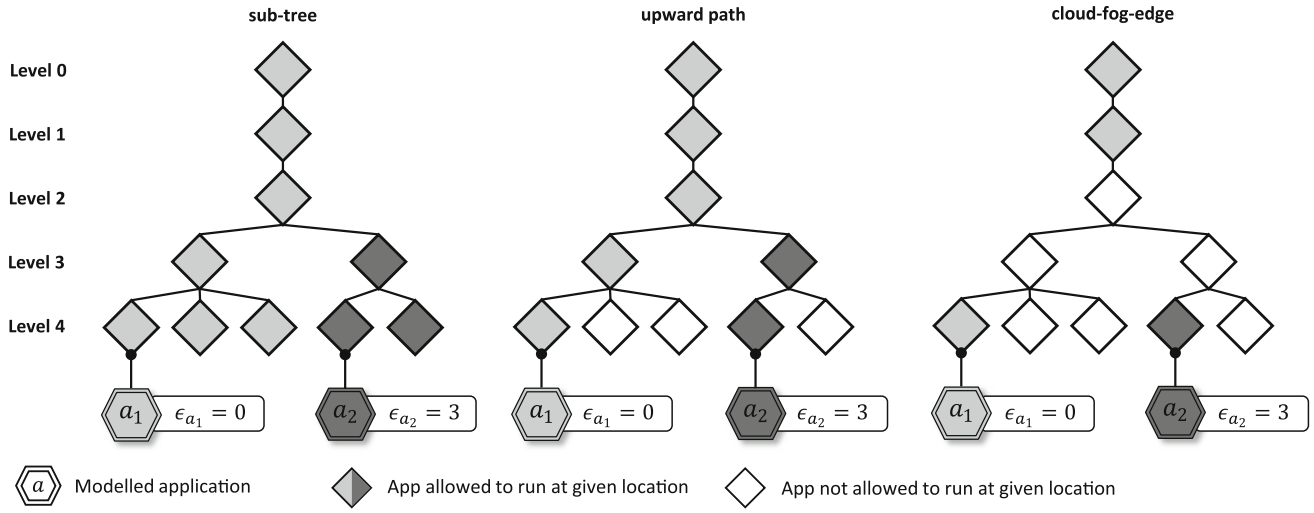

Fig. 4 Application placement strategy (cloud-fog-edge, path, and sub-tree) and allowed edge level determine the allowed positions (gray)

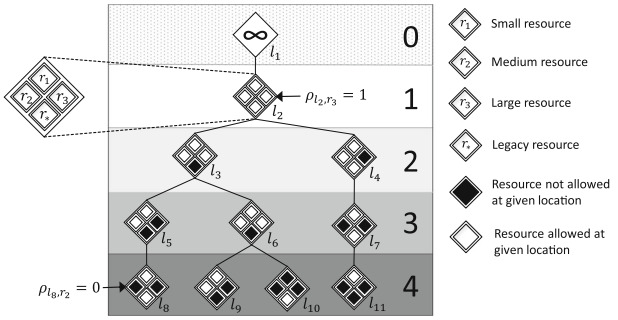

Fig. 5 Available placement locations for different resources

Similarly, $\rho_{l,r}^{ResLevel}$ has been created to control which resource type can run at which level. For example, in most cases, the placement of powerful computational resources should be kept from the data producer level. Such relations can be formulated as general rules and used to build matrix $\rho_{l,r}^{ResLevel}$ automatically. What is more, users can manipulate the matrix manually to add custom changes. For example, suppose a factory planner knows that one specific system or location can accommodate a more powerful resource or cannot accommodate any resource (for example, due to the lack of the necessary network or electrical infrastructure). In that case, the location can be deactivated (set to 0) for all resources. Such an approach of manual manipulation can also be used for allowed position $\tau_{a,l}^{Allowed}$.

The resource level parameter $\rho_{l,r}^{ResLevel}$ restricts the placement of resources at specific locations by linking the resource with the location; see Eq. (3.21). The $\rho_{l,r}^{ResLevel}$ is calculated automatically for all resources before executing the solver using the $level_r$ value defined for each resource and potential user settings.

$$\rho_{l,r}^{ResLevel} = \begin{cases} 1, & \text{if } r \text{ is allowed at location } l \\ 0, & \text{if } r \text{ is not allowed at location } l \end{cases} \tag{3.21}$$

The $\rho_{l,r}^{ResLevel}$ parameter has two purposes. First, it allows for avoiding unrealistic scenarios in which, for example, a cloud resource is placed on-premise or a high-end server unit is attached directly to a data source. Second, it allows the integration of existing IT resources already available in the factory. For example, the practitioner specifies for l_{10} that the value of $\rho_{l,r}^{ResLevel}$ is 1 for only the existing legacy resource (r_*), forcing the solver to use only the legacy resource at this position or none. Another way would be to set the $\rho_{l,r}^{ResLevel}$ for all resources for this location to one and leave it up to the algorithm to decide if the resource should be utilized or replaced (compare r_* at l_2).

Implementation and benchmark

The benchmark instances represent factories of different but realistic sizes. Since the proposed method relies on mixed-integer linear programming, being NP-hard problems, solution times for any problem instance size may vary significantly from one to another. However, the numerical experiments presented here give an idea of what computation times a planner could face for the different problem sizes. However, since the problem of factory planning we are addressing here is not time-critical, even longer computation times do not pose an issue.

Implementation

The proposed multi-layer resource placement model is implemented using the open-source modeling language Pyomo (Hart et al., 2011). The modeling language is independent of a specific solver. Based on prior experience, the solver SCIP (Gleixner et al., 2018) was used. The required preprocessing and model preparation are implemented in Python. The overall execution time of the optimization determines the usability of the proposed model and implementation for the planning and design of an IT infrastructure.

A benchmark procedure is designed to record the total optimization times to assess the optimization implementation's applicability to large-scale problems. The benchmark structure is illustrated in Fig. 6.

The experiment designer sets the sets for locations, applications, and the number of repetitions. Then, the sets of locations and applications are combined to create scenarios. The number of scenarios is equal to the application times location sets. Then, each scenario is repeated according to the number of specified loops. Currently, no large-scale edge implementations could be used as reference scenarios. Hence, a topology generator is created, enabling the automatic generation of topologies with randomized locations and applications. Firstly, the factory layout with sections and systems is created based on the location specification, compare Table 1.

Secondly, the specified amount of factory objects is randomly distributed and attached to the systems. Thirdly, applications are randomly selected from an application catalog containing 21 applications. The resource requirements for each application are either low or high. Each application consumes generic performance [2/7], memory [0.1/0.5], and bandwidth [1/50] units. Each of the formed connections

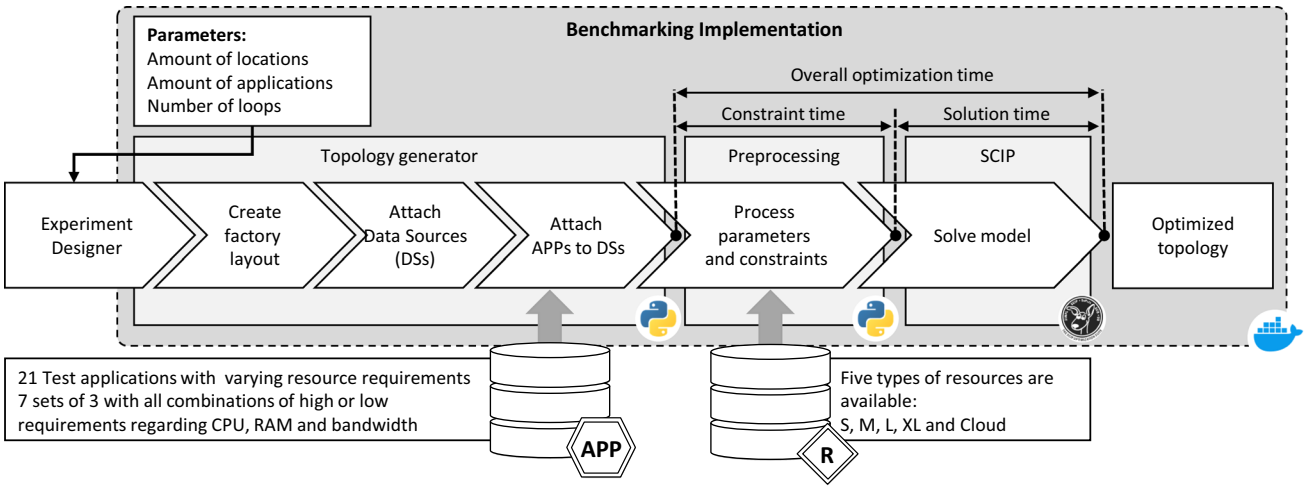

Fig. 6 Structure of the optimization benchmark

Table 1 Benchmark parameters

Experiment parameters	Repetitions	100				
	Location sets	100, 200, 500, 1000				
	App sets	100, 200, 500, 1000				
	Edge Level	0	1	2	3	4
	Distribution (weight)	0.4	0.3	0.2	0.1	0.1
Optimization parameters	Time period	5 years				
	Objective	Minimize cost				
	Placement strategy	Upward path				
Optimization configuration	Modeling language	Pyomo				
	Solver	SCIP version 7.0.1				
	Termination: Time limit	10 h				
	Termination: Gap limit	0.01				
Benchmark PC	CPU	Intel i7-6700 CPU @ 3.40 GHz				
	RAM	36 GB				
	OS	Ubuntu 20.04				

Table 2 Available resources for the optimization

Resources	S	M	L	XL	Cloud
Performance unit	5	50	500	1000	∞
Memory unit	1	5	20	70	∞
Bandwidth limitation	100	100	1000	1000	10,000
CapEx (€)	50	500	2000	7000	0
OpEx (€/a)	500	1000	2000	4000	0.1
Environmental impact (tCO_2/a)	5	15	25	30	0.1

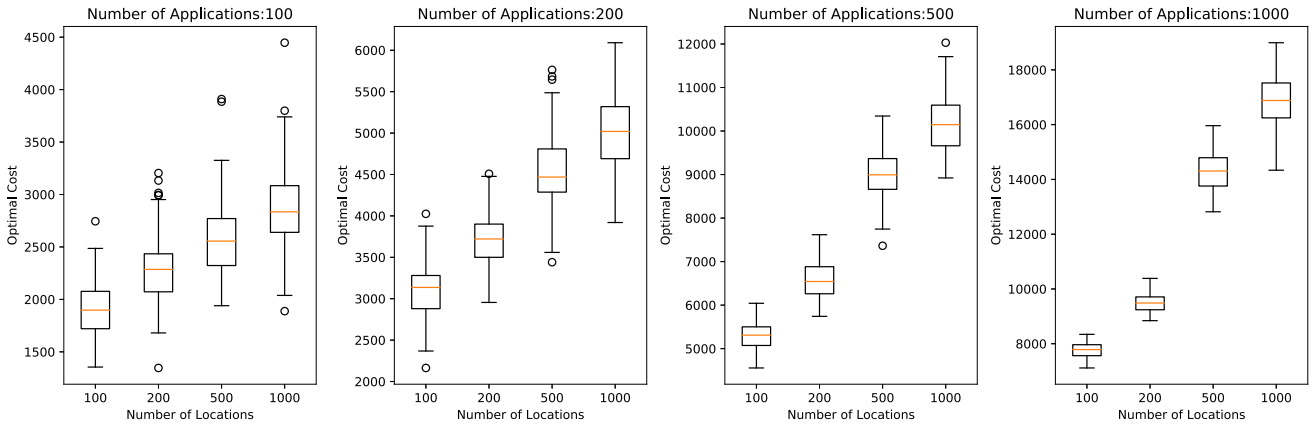

Fig. 7 Distribution of optimal cost for the 16 scenarios

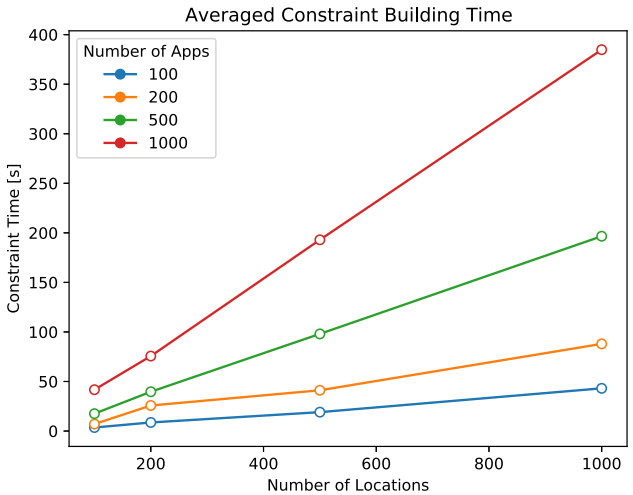

Fig. 8 Mean constraint building time for the 16 scenarios

of factory object and App is assigned an edge level. The assigned level is based on a predefined distribution function indicated in Table 1 that slightly favors level three or lower, assuming that Apps are less likely to be bound to the factory object location. The resource catalog contains five resource types; see Table 2. Similar to Lin and Yang (2018), generic units for performance (1–1000), memory (1–70), and bandwidth (100–1000) are specified. Cost is split in capital expenditure (€) and operational expenditure (€ per annum), and the environmental impact is defined through tons of CO_2 per annum. These values, including the values for a cloud implementation, are a grave simplification deemed necessary to estimate resource capabilities in a planning phase. For the benchmark, the resource and application catalog are stored locally in JSON files in order to be independent of any specific database implementation potentially influencing the benchmark.

After the topology creation, the resulting scenario is transferred to the optimization procedure consisting of the preprocessing script initializing the model and the solver. Both the constraint building and the solving process of the model of every execution are timed.

Benchmark results

The benchmark results were obtained using the parameters presented in Table 1. The docker instance ran for nine days on a dedicated server executing 100 repetitions of each of the 16 scenarios (4 location sets × 4 application sets). The optimizations and timings results are presented as standard box plots. The circles found in the graphs represent the outliers.

The optimized cost distribution of the 16 scenarios depicted in Fig. 7 illustrates the overall cost's dependence on the initial topology. The initial topology's significant effect underlines the importance of embedding the optimization in a framework that allows a transparent assessment of the overall factory topology before an optimization run.

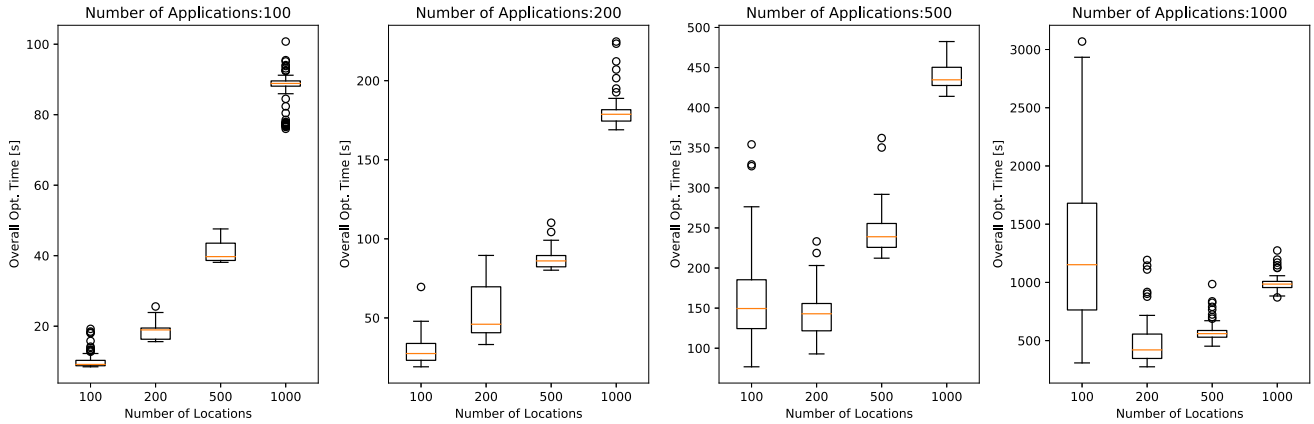

Fig. 9 Total optimization time (Includes constraint preprocessing and solving) for the 16 scenarios

Even though the constraint building times seen in Fig. 8 are increasing significantly with the number of possible combinations, their increase seems relatively linear. The constraint building times do not cause a bottleneck in the implementation, even for large-scale implementations.

Analyzing the benchmark timings, especially the total optimization time in Fig. 9, reveals that optimizing with SCIP can also address large-scale placement problems. Even when placing resources for 1000 Apps on potentially 1000 locations, the longest total optimization was under 22 min, with the mean being under 15 min. Small-scale problems with only 100 Apps and under 500 locations are solved in less than a minute. The results exceeded the initial expectations. However, the benchmark results also show that the total optimization time depends on the initial topology, which can be seen for 1000 Apps in 100 locations. The total optimization time distribution ranges from 50 minutes to under a minute. This particular scenario highlights an unfortunate disadvantage of our implementation. Even though the model and the generated topology are fully transparent, investigating the sudden rise in solve time bore no conclusive results.

Case study

The benchmarks showed that the total optimization times achieved by the presented implementation are low enough to make its application feasible for planning and design use cases. The case study aims to make the theoretical model more tangible and assess its application in practice. The case study's target is a factory producing electronic components for factory automation. It is considered a leading example of digitalization and innovation and is an early adopter of the Siemens edge computing ecosystem (Beitinger, 2021). The available IT infrastructure ensures the smooth operation of the production machines. However, novel applications like the ones presented in Filz et al. (2020), Schulte et al. (2020), which aim to improve overall quality control, are under development and shall be deployed in a designated area of the factory. Instead of relying on the existing IT resources, factory management aims to invest in a new, more distributed IT infrastructure profiting from the potential benefits of edge computing.

The factory is modeled following the approach presented in Sect. 3.1 specifying four sections, eight systems, and 18 data sources each with a unique ID illustrated in Fig. 10. The unsorted numbering indicates the modeling priorities of the practitioner, e.g., starting with modeling the data sources of system 2 in section 3 instead of completing the factory section by section. When adding the factory and cloud as resource placement options, the total number of locations is 32. The 2-D representation in Fig. 10 showcases the most cost-efficient IT topology over a period of five years. Each of the potential placement location has an ID (number), the colors represent the placed resource type (None to very large), the shape represent the placement level (on-device to cloud), and the dotted line shows the data flow indicating a connection between a data prosumer and an application. E.g., there is at least one application that consumes data from data source 14, within system three which is moved to the large resource placed in section two. In total twelve resources (4S, 4M, 4L) were placed at the locations.

The representation of the use case as graph in Fig. 11 shifts the focus more in the distribution of applications in the initial topology prior and after the optimization showcasing the concrete placement recommendation for both applications and resources in the factory. The result of the optimization execution for all three implemented objective functions can be directly compared with the initial situation in Table 3. The initial situation represents the not-yet optimized graph presented in Fig. 11 where each application is attached to its respective data source.

A default resource (R_L) is placed for each occupied location, guaranteeing the App's execution. Coincidentally, both

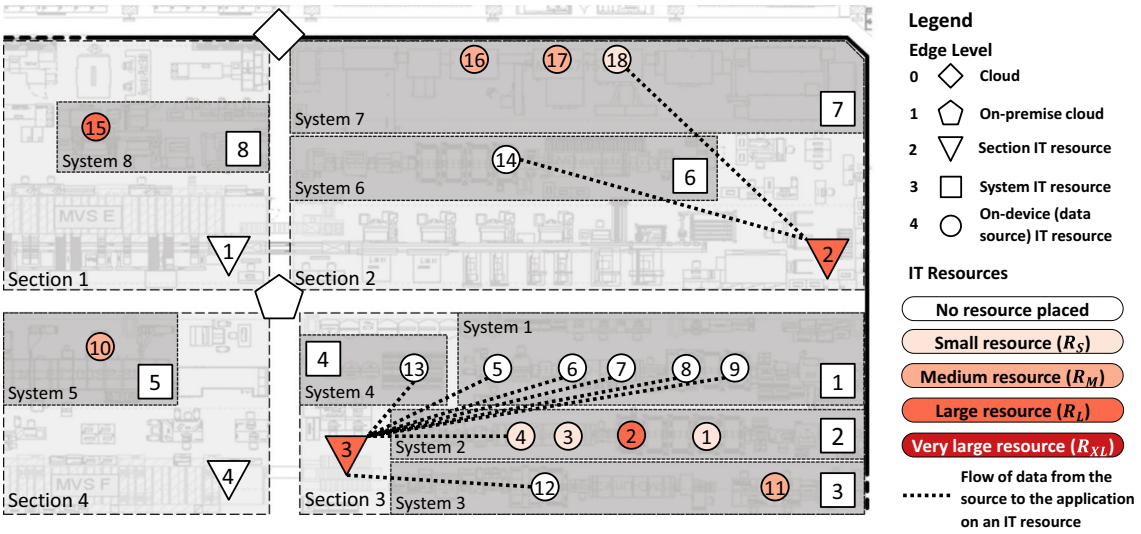

Fig. 10 Case study: cost optimized topology

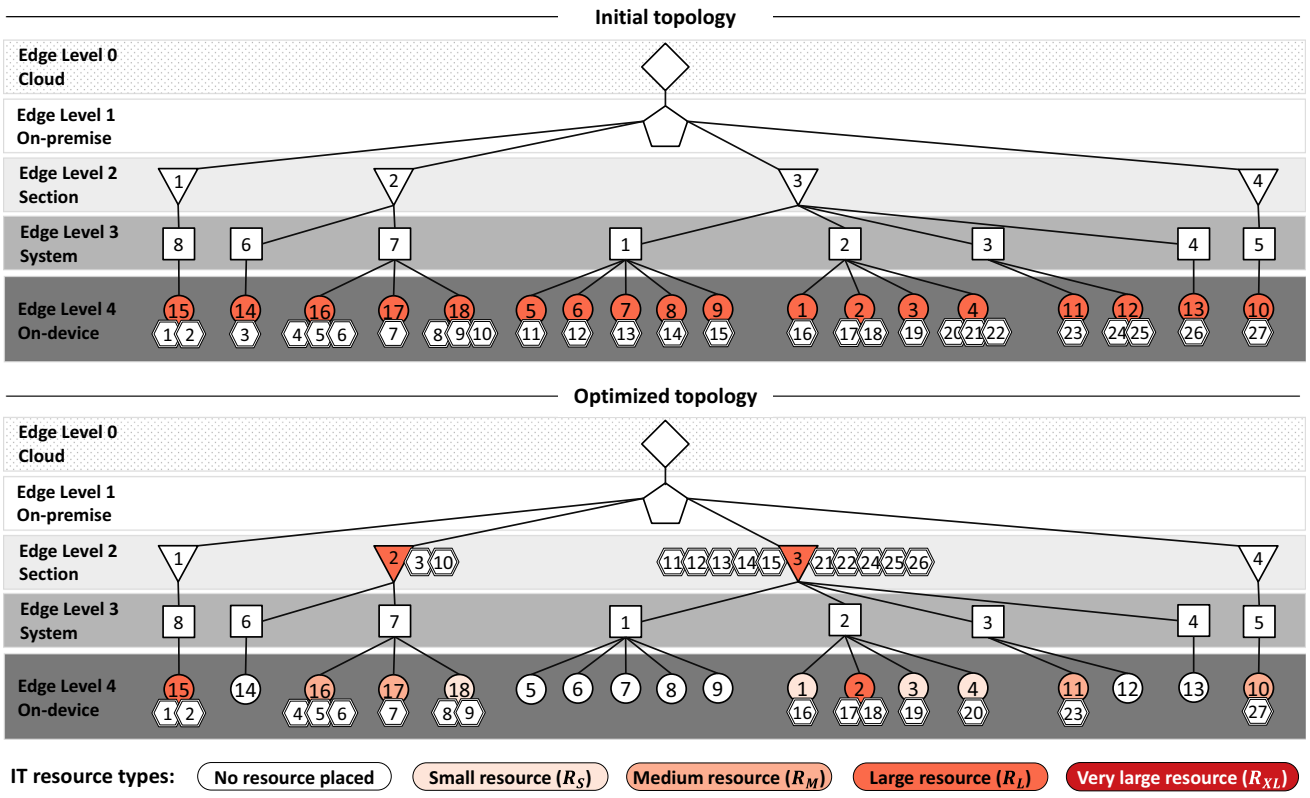

Fig. 11 Initial factory graph (top) compared to the optimized graph (bottom)

and € objectives result in an identical placement recommendation which is significantly cheaper and has less environmental impact than the third objective. The total cost of 80.000 € was significantly lower than the default cost of 216.000 € for the initial situation where only R_L were placed to ensure the operation and allow for extension. Even the longest total execution time was below three seconds, which is negligible compared with the initial modeling effort demonstrating the presented model's practical usability and implementation.

Table 3 Detailed result of the optimization execution

		Initial situation*	Objectives		
			Cost	Environmental impact	Number of nodes
Overall optimization time (s)		–	2.7	1.6	1.6
Constraint calculation time (s)		–	0.7	0.7	0.5
Solve time (s)		–	1.8	0.9	1.1
Number of placed resources		18	12	12	11
Total Cost (€)		216k	80k	80k	147k
Environmental impact (tCO_2)		457.2	135	135	429.4
Resource distribution	R_S	0	4	4	0
	R_M	0	4	4	0
	R_L	18	4	4	10
	R_{XL}	0	0	0	1

*For the initial situation, a R_L is placed on every location with an application. The large resource is chosen to ensure operation

Conclusion

This work presented, implemented, and assessed a novel modeling approach for the IT resource and application placement problem considering the edge computing paradigm in the context of manufacturing. Instead of intending to use other existing resource management modeling and optimization approaches with more modeling parameters, higher accuracy, and less execution time like the ones surveyed in Hong and Varghese (2019), our work complements them by addressing the design and planning phase. The NP-hard problem of determining the optimal physical placement, choice of IT resource type (capacity), and deployment of the given applications were addressed using mixed-integer linear programming (MILP). Using the open-source software package PYOMO to model the problem and SCIP as the solver minimizing cost, environmental impact, and device count proved to be an effective combination. The total solve times ranging from a few seconds for a small scale (100 Locations, 100 Applications) to around 15 min for large scale problems (1000 Locations, 1000 Applications) are unproblematic in an offline design an planning phase.

As pointed out, our approach does not aim at the optimization of the type and amount of specific manufacturing IT applications. However, the systematic procedure combined with a processable model provides early-stage decision support for the placement problem of involved IT resources. Thus, it is a promising approach for manufacturing and technology, providing companies with more time, cost savings, and energy-efficient IT architectures. Additionally, from a more scientific perspective, it gives relevant insights into boundary conditions and influencing factors on different target variables. The model is designed to be extendable enabling the integration of more sophisticated cost, environmental impact, and resource consumption functions.

Data availability can be an issue, especially in the early planning and design phases and the necessity of an edge-level value for each application could be considered a limitation of the proposed modeling approach. However, the approach is intentionally based on reduced (and thus easier available) input compared to other, more detailed optimization frameworks, like iFogSim (Gupta et al., 2017), favoring applicability in practice over accuracy facilitating reasonable computing times. Given that the model is designed to be extended, a wide range of scenarios can be considered, which may include the consideration of uncertainty for the input parameters toward the most robust results. In the future, it would be interesting to investigate and compare different ways of addressing multi-objective optimization and apply the approach to an increasing range of factory use cases. At the same time, a more sophisticated edge level determination and an optimization of the application selection procedure is a challenge that can be addressed decoupled from the model and its current implementation before being incorporated into one optimization framework.

Funding Open Access funding enabled and organized by Projekt DEAL.

Declarations

Conflict of interest The authors declare that they have no conflict of interest.

References

Aazam, M., Zeadally, S., & Harras, K. A. (2018). Deploying Fog computing in industrial internet of things and industry 4.0. *IEEE Transactions on Industrial Informatics, 14*(10), 4674–4682. https://doi.org/10.1109/TII.2018.2855198

Basir, R., Qaisar, S., Ali, M., Aldwairi, M., Ashraf, M. I., Mahmood, A., & Gidlund, M. (2019). Fog computing enabling industrial Internet of Things: State-of-the-art and research challenges. *Sensors, 19*(21), 4807. https://doi.org/10.3390/s19214807

Beitinger, G. (2021). Digitalization and automation are the game-changers. https://ingenuity.siemens.com/2021/03/digitalization-and-automation-are-the-game-changers/

Brettel, M., Klein, M., & Friederichsen, N. (2016). The relevance of manufacturing flexibility in the context of Industrie 4.0. *Procedia CIRP, 41*, 105–110. https://doi.org/10.1016/j.procir.2015.12.047

Chen, B., Wan, J., Celesti, A., Li, D., Abbas, H., & Zhang, Q. (2018). Edge computing in IoT-based manufacturing. *IEEE Communications Magazine, 56*(9), 103–109. https://doi.org/10.1109/MCOM.2018.1701231

Filz, M. A., Herrmann, C., & Thiede, S. (2020). Simulation-based assessment of quality inspection strategies on manufacturing systems. *Procedia CIRP, 93*, 777–782. https://doi.org/10.1016/j.procir.2020.04.069

Ghobaei-Arani, M., Souri, A., & Rahmanian, A. A. (2020). Resource management approaches in fog computing: A comprehensive review. *Journal of Grid Computing, 18*(1), 1–42. https://doi.org/10.1007/s10723-019-09491-1

Gleixner, A., Bastubbe, M., Eifler, L., Gally, T., Gamrath, G., Gottwald, R.L., Hendel, G., Hojny, C., Koch, T., Lübbecke, M.E., Maher, S.J., Miltenberger, M., Müller, B., Pfetsch, M.E., Puchert, C., Rehfeldt, D., Schlösser, F., Schubert, C., Serrano, F., Shinano, Y., Viernickel, J.M., Walter, M., Wegscheider, F., Witt, J.T., & Witzig, J. (2018). The SCIP Optimization Suite 6.0. Technical report, Optimization Online

Guo, P., Lin, B., Li, X., He, R., & Li, S. (2016). Optimal deployment and dimensioning of fog computing supported vehicular network. In: 2016 IEEE Trustcom/BigDataSE/ISPA, IEEE, Tianjin, China, (pp. 2058–2062). https://doi.org/10.1109/TrustCom.2016.0315

Gupta, H., Vahid Dastjerdi, A., Ghosh, S. K., & Buyya, R. (2017). iFogSim: A toolkit for modeling and simulation of resource management techniques in the Internet of Things, Edge and Fog computing environments: iFogSim: A toolkit for modeling and simulation of internet of things. *Software: Practice and Experience, 47*(9), 1275–1296. https://doi.org/10.1002/spe.2509

Hart, W. E., Watson, J. P., & Woodruff, D. L. (2011). Pyomo: Modeling and solving mathematical programs in Python. *Mathematical Programming Computation, 3*(3), 219.

Hertel, M., & Wiesent, J. (2013). Investments in information systems: A contribution towards sustainability. *Information Systems Frontiers, 15*(5), 815–829. https://doi.org/10.1007/s10796-013-9417-x

Hischier, R., Coroama, V. C., Schien, D., & Ahmadi Achachlouei, M. (2015). Grey energy and environmental impacts of ICT hardware. In L. M. Hilty & B. Aebischer (Eds.), *ICT innovations for sustainability* (pp. 171–189). Springer. https://doi.org/10.1007/978-3-319-09228-7

Hong, C. H., & Varghese, B. (2019). Resource management in fog/edge computing: A survey on architectures, infrastructure, and algorithms. *ACM Computing Surveys, 52*(5), 1–37. https://doi.org/10.1145/3326066

Ismail, B.I., Mostajeran Goortani, E., Ab Karim, M.B., Ming Tat, W., Setapa, S., Luke, J.Y., & Hong Hoe, O. (2015). Evaluation of Docker as Edge computing platform. In: 2015 IEEE Conference on Open Systems (ICOS), IEEE, Bandar Melaka, (pp. 130–135). https://doi.org/10.1109/ICOS.2015.7377291

Jiang, C., Wan, J., & Abbas, H. (2021). An edge computing node deployment method based on improved k-means clustering algorithm for smart manufacturing. *IEEE Systems Journal, 15*(2), 2230–2240. https://doi.org/10.1109/JSYST.2020.2986649

Kumar, D., Baranwal, G., & Vidyarthi, D. P. (2022). A survey on auction based approaches for resource allocation and pricing in emerging edge technologies. *Journal of Grid Computing, 20*(1), 3. https://doi.org/10.1007/s10723-021-09593-9

Lin, C. C., & Yang, J. W. (2018). Cost-efficient deployment of fog computing systems at logistics centers in industry 4.0. *IEEE Transactions on Industrial Informatics, 14*(10), 4603–4611. https://doi.org/10.1109/TII.2018.2827920

Mao, W., Akgul, O. U., Mehrabi, A., Cho, B., Xiao, Y., & Ylä-Jääski, A. (2022). Data-driven capacity planning for vehicular Fog computing. *IEEE Internet of Things Journal, 9*(15), 13179–13194. https://doi.org/10.1109/JIOT.2022.3143872

Mourtzis, D., Vlachou, E., & Milas, N. (2016). Industrial big data as a result of IoT adoption in manufacturing. *Procedia CIRP, 55*, 290–295. https://doi.org/10.1016/j.procir.2016.07.038

Noghabi, S. A., Cox, L., Agarwal, S., & Ananthanarayanan, G. (2020). The emerging landscape of edge computing. *GetMobile: Mobile Computing and Communications, 23*(4), 11–20. https://doi.org/10.1145/3400713.3400717

Qi, Q., & Tao, F. (2019). A smart manufacturing service system based on edge computing, fog computing, and cloud computing. *IEEE Access, 7*, 86769–86777. https://doi.org/10.1109/ACCESS.2019.2923610

Schulte, L., Schmitt, J., Meierhofer, F., & Deuse, J. (2020). Optimizing inspection process severity by machine learning under label uncertainty. Advances in intelligent systems and computingIn I. L. Nunes (Ed.), *Advances in human factors and systems interaction* (pp. 3–9). Springer. https://doi.org/10.1007/978-3-030-51369-6

Thiede, S. (2021). Digital technologies, methods and tools towards sustainable manufacturing: Does Industry 4.0 support to reach environmental targets? *Procedia CIRP, 98*, 1–6. https://doi.org/10.1016/j.procir.2021.02.001

Trinks, S., & Felden, C. (2018). Edge Computing architecture to support Real Time Analytic applications : A State-of-the-art within the application area of Smart Factory and Industry 4.0. In: 2018 IEEE International Conference on Big Data (Big Data), IEEE, Seattle, WA, USA, (pp. 2930–2939). https://doi.org/10.1109/BigData.2018.8622649

Vogel-Heuser, B., Fay, A., Schaefer, I., & Tichy, M. (2015). Evolution of software in automated production systems: Challenges and research directions. *Journal of Systems and Software, 110*, 54–84. https://doi.org/10.1016/j.jss.2015.08.026

Wang, S., Zhao, Y., Xu, J., Yuan, J., & Hsu, C. H. (2019). Edge server placement in mobile edge computing. *Journal of Parallel and Distributed Computing, 127*, 160–168. https://doi.org/10.1016/j.jpdc.2018.06.008

Wescott, B. (2013). Every Computer Performance Book: How to Avoid and Solve Performance Problems on the Computers You Work With. CreateSpace Independent Publishing Platform

Xu, Z., Liang, W., Xu, W., Jia, M., & Guo, S. (2016). Efficient algorithms for capacitated cloudlet placements. *IEEE Transactions on Parallel and Distributed Systems, 27*(10), 2866–2880. https://doi.org/10.1109/TPDS.2015.2510638

Yin, H., Zhang, X., Liu, H., Luo, Y., Tian, C., Zhao, S., & Li, F. (2017). Edge provisioning with flexible server placement. *IEEE Transactions on Parallel and Distributed Systems, 28*(4), 1031–1045. https://doi.org/10.1109/TPDS.2016.2604803

Yin, S., Bao, J., Zhang, J., Li, J., Wang, J., & Huang, X. (2020). Real-time task processing for spinning cyber-physical production systems based on edge computing. *Journal of Intelligent Manufacturing, 31*(8), 2069–2087. https://doi.org/10.1007/s10845-020-01553-6

Yousefpour, A., Fung, C., Nguyen, T., Kadiyala, K., Jalali, F., Niakanlahiji, A., Kong, J., & Jue, J. P. (2019). All one needs to know about fog computing and related edge computing paradigms: A complete survey. *Journal of Systems Architecture, 98*, 289–330. https://doi.org/10.1016/j.sysarc.2019.02.009

Zhang, D., Haider, F., St-Hilaire, M., & Makaya, C. (2019). Model and algorithms for the planning of fog computing networks. *IEEE Internet of Things Journal*. https://doi.org/10.1109/JIOT.2019.2892940

Zietsch, J., Vogt, M., Lee, B. D., Herrmann, C., & Thiede, S. (2020). Enabling smart manufacturing through a systematic planning framework for edge computing. *CIRP Journal of Manufacturing Science and Technology*. https://doi.org/10.1016/j.cirpj.2020.06.010

Zietsch, J., Weinert, N., Herrmann, C., & Thiede, S. (2019). Edge Computing for the Production Industry A Systematic Approach to Enable Decision Support and Planning of Edge. In: 2019 IEEE 17th International Conference on Industrial Informatics (INDIN), IEEE, Helsinki, Finland, (pp. 733–739). https://doi.org/10.1109/INDIN41052.2019.8972193