The Journal of Systems and Software 235 (2026) 112755

Contents lists available at ScienceDirect

The Journal of Systems & Software

journal homepage: www.elsevier.com/locate/jss

A self-sustainable service assembly for decentralized computing environments

Mauro Caporuscio [a,*], Mirko D'Angelo [b], Vincenzo Grassi [c], Raffaela Mirandola [d], Francesca Ricci [e]

[a] Linnaeus University, Växjö, Sweden
[b] Ericsson Research, Göteborg, Sweden
[c] Università di Roma Tor Vergata, Rome, Italy
[d] Karlsruhe Institute of Technology, Karlsruhe, Germany
[e] Spike Reply, Milan, Italy

ARTICLE INFO

Editor: Dr Alexander Chatzigeorgiou

Keywords:
Service assembly
Service composition
Software architecture
Sustainable computing
Decentralized systems

ABSTRACT

The landscape of modern computing systems is shifting towards architectures built by combining available services under the "everything as a service" paradigm. These architectures are deployed on distributed cloud-edge infrastructures, aiming to provide innovative services to a wide range of users. However, it is crucial for these systems to address environmental sustainability concerns. This poses challenges in operating such systems in open, dynamic, and uncertain environments while minimizing their energy consumption. To tackle these challenges, we propose a decentralized service assembly approach that ensures the assembly is energetically self-sustainable by relying on locally harvested and stored energy. In our contribution, we introduce a general service selection template that enables the derivation of different selection policies. These policies guide the construction and maintenance of the service assembly. To evaluate their effectiveness in meeting the sustainability requirements, we conduct a comprehensive set of simulation experiments, providing valuable insights.

1. Introduction

The Internet of Things (IoT) envisions digital transformation scenarios where data-driven and AI-augmented functionalities support human beings in their tasks, leveraging the sensing and actuating capabilities provided by a multitude of low-end devices embedded in everyday things. This vision relies on the existence of a diffused "fabric" made of high-speed wired and wireless communication technologies connecting end-user devices to a variety of computing nodes, spanning distant cloud servers, proximity edge/fog nodes, and even other end-user devices (the so-called *computing continuum* (Casamayor-Pujol et al., 2023)).

The *anything as a service* paradigm is commonly acknowledged as an appropriate engineering abstraction in this context. It offers a uniform perspective, as mentioned in the IoT roadmap (Bouguettaya et al., 2021). Essentially, all functionalities, even those provided by "things", are abstracted and made available as services that can be connected to and by other services (Bouguettaya et al., 2017).

Such a scenario presents challenges for software engineering due to its dynamic and open-ended nature. Factors such as the distribution and heterogeneity of the service ownership, the mobility of devices hosting the services, and the intermittent nature of communication channels contribute to the transient availability of services in a given area or time interval. This transient availability is more of a norm than an exception. Consequently, authors highlight the *serendipitous* nature of IoT-based services, as new value-added services often emerge from the *opportunistic* composition of available ones (Bouguettaya et al., 2021). There is a rich literature on possible approaches to address the problem of service composition (Razian et al., 2022). Regardless of their peculiarities, the proposed approaches to service composition share a common underlying assumption that the selected services to be used in the composition of a new value-added service are fully functional. However, this assumption can be challenged due to the dynamic and open-ended nature of the scenario being considered. Each deployed service could indeed rely on external resources such as other existing services, code libraries, data sources, or reusable assets, which we refer to as its *dependencies*. These dependencies must be resolved to ensure that the service is fully functional. However, the continuous availability of the needed resources cannot be guaranteed, as we have already pointed out in

* Corresponding author.
E-mail addresses: mauro.caporuscio@lnu.se (M. Caporuscio), mirko.dangelo@ericsson.se (M. D'Angelo), vincenzo.grassi@uniroma2.it (V. Grassi), raffaela.mirandola@kit.edu (R. Mirandola), fr.ricci@reply.it (F. Ricci).

https://doi.org/10.1016/j.jss.2025.112755
Received 10 March 2025; Received in revised form 23 October 2025; Accepted 17 December 2025
Available online 23 December 2025

relation to services. Following the paradigm of *anything as a service*, these resources can be considered as services themselves, which may also have their own dependencies that need to be resolved and maintained.

Hence, to enable the emergence and smooth operation of new services in a dynamic and open environment, such as the IoT and computing continuum, it is necessary to consistently activate a dependency resolution process that keeps all the dependencies of the already existing services resolved. Adopting the terminology used, for example, in Sykes et al. (2011), we call the problem of designing and implementing this process the *service assembly* problem, which corresponds, as discussed in Section 9, to finding a solution to the following research question[1]:

RQ0: *Given a dynamic set of services, how can we solve and maintain over time their dependencies using other services in the same set?*

Different answers to this question can already be found in the literature. Independent of the adopted approach, giving an effective answer to RQ0 means, in the end, favoring the pervasive diffusion of functionalities based on the use of information and communication technologies (ICT), as it facilitates the emergence and maintenance of new services from existing ones. In this respect, there is an increasing concern that the overwhelming diffusion of ICT could negatively impact the achievement of environmental sustainability objectives (Freitag et al., 2021), because of the increase in energy consumption and the potential rise in greenhouse gas emissions if the energy demand is met using *brown* energy sources.

To this extent, our goal is to investigate RQ0 from the perspective of the *energetic autonomy* concept. This means that the energy required for the services in a specific environment should come solely from local renewable energy sources (such as solar or wind energy), resulting in zero greenhouse gas emissions. The main research question we intend to address in this paper can be thus stated as follows:

RQ1: *How and to what extent can we answer* RQ0 *under the requirement that the assembled services fulfill the energetic autonomy constraint (as stated above)?*

Existing answers to RQ0 are natural candidates for investigating RQ1. Some of these answers employ optimization techniques that typically assume global and centralized knowledge, where complete information about the system is available. Since our focus is on decentralized systems, where only local knowledge is accessible, we address RQ1 by building on the solution proposed in Angelo et al. (2020), as its decentralization, self-adaptation, and scalability features make it particularly suitable for the IoT scenario we are considering.

Besides RQ1, we also address another related question. The primary obstacle to tackling RQ1 is to guarantee service availability, even with the intermittent nature of green energy sources. In this respect, any solution aimed at RQ1 must rely on data about the energy demand of the services and the availability of adequate energy (the system's *energy state*). Collecting and updating the energy state is essential in the dynamic scenario we are considering. However, this activity also contributes to the overall energy demand. The choice of different indicators to represent the energy state could have different impacts on energy consumption, depending on the complexity of their collection and updating process. Opting for simpler indicators that offer less detailed information could reduce the related energy consumption, but they could impede the goal of achieving energetic autonomy. This gives rise to an additional research question, which can be stated as follows:

RQ2: *Can we get insights about which energy state indicator is more suitable to reach a good trade-off between achieving the energetic autonomy goal and the complexity of keeping that indicator updated?*

Paper contribution. Answering the research questions RQ1 and RQ2 requires tackling issues, including: (*i*) the complexity caused by the high number of entities distributed in the system (e.g., IoT devices, edge/fog computing nodes), (*ii*) the lack of global knowledge, which is typical in large-scale dynamic distributed systems, (*iii*) the unpredictable variability of the environment (e.g., entities joining/leaving the system or changing their quality attributes).

To cope with these issues, the proposed approach to answer RQ1 and RQ2 has the following characteristics:

1. The system is completely decentralized, eliminating the requirement for each participating entity to possess complete knowledge of the system's state.
2. It handles on its own events such as nodes or services joining/leaving the system or changes in locally available energy.

Our contribution deeply refines and extends some preliminary results already presented in Caporuscio et al. (2020). With respect to that paper, the main extensions and refinements concern: (*i*) the consideration of a more complex reference scenario that includes intermittently available green energy sources and battery-powered devices and also the possibility of nodes hosting multiple services; (*ii*) the definition of a parameterized service selection policy, from which different policies can be easily derived to implement the service selection process; (*iii*) a more articulated presentation of the adopted approach and a deeper comparison with related works, and (*iv*) an extensive set of experiments referring to the more complex reference scenario we are considering.

Structure of the paper. Sections 2 and 3 lay the groundwork for this study by formalizing the system model and providing an overview of the *service assembly* approach we build on, respectively. Then, Section 4 introduces the adopted energy model, and Section 5 defines the metrics we adopt to measure the self-sustainability. Section 6 defines energy-aware service ranking metrics, introduces a parameterized service selection policy, and gives details on the service selection policies that can be derived from it. Section 7 shows the results of our experiments, Section 8 discusses how these results answer RQ1 and RQ2 and possible threats to their validity. Section 9 reviews related works, while Section 10 outlines conclusions and future work.

Table 1
Notation.

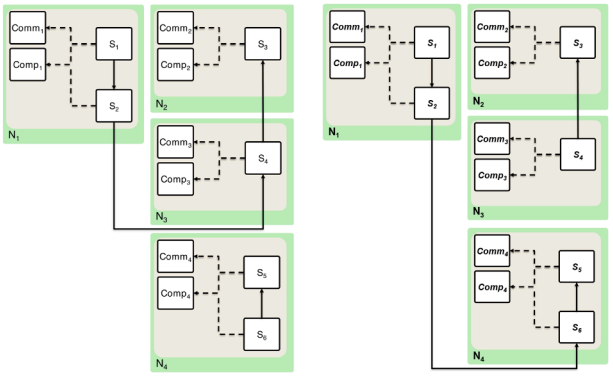
$\mathbf{N}$	Set of nodes
N_S	Node hosting the service S
$\mathbf{S}$	Set of services
$\mathbf{T}$	Set of service types
$Type_S$	Type of the provided interface by S
Dep_S	Set of dependencies of S
$Prov_S(t)$	Set of services selected to resolve the dependencies of S
$Req_S(t)$	Set of services bound to S to resolve one of their dependencies
$\mu_{S,d}$	Average number of times a service S requires dependency d
$B_N(t)$	Battery level of node N at time slot t
$G_N(t)$	Energy generated by node N within time slot t
C_N	Energy consumed by node N within time slot t
$L_S^{comp}(t)$	Node level (local scope) computation energy consumption
$O_S^{comp}(t)$	System level (global scope) computation energy consumption
$K_{S,S'}^{comm}$	Communication energy consumption for a single request from S to S'
$L_S^{comm}(t)$	Node level (local scope) communication energy consumption
$O_S^{comm}(t)$	System level (global scope) communication energy consumption
gef_S	Global (system-wide) energy footprint of service S
lef_S	Local (node level) energy footprint of service S
$ge_{S,S'}$	Global energy (Eq. (10))
$le_{S,S'}$	Local energy (Eq. (11))
$ne_{S'}$	Energy level of the node hosting service S' (Eq. (12))
$rl_{S'}$	Residual lifetime of the node hosting service S' (Eq. (13))

2. System model

Table 1 summarizes the notations used throughout the paper.

We consider a distributed system consisting of a set $\mathbf{N}$ of nodes (e.g., IoT and computing continuum nodes), and a set $\mathbf{S}$ of (software) services

[1] We discuss in Section 9 the relationship of this problem with the *service composition* problem.



(a) S_2 is linked to S_4 (b) S_2 is linked to S_6

Fig. 1. Service assembly examples: same dependencies but different bindings for full resolution.

deployed on these nodes, each of them implementing a specific functionality. A Node $N \in \mathbf{N}$ hosts one or more services $S \in \mathbf{S}$, providing them with the basic computing and communication resources needed for their operations and supplying the required energy (see Section 4).

Given a service $S \in \mathbf{S}$, we denote by N_S its hosting node, and by $Type_S \in \mathbf{T}$ the *type* of the functionality it offers (provided interface). We assume that, in general, $\mathbf{S}$ includes multiple services having the same type (functionally equivalent services), but different non-functional characteristics.

A service $S \in \mathbf{S}$ generally requires functionalities offered by other resources to carry out its own task. As already pointed out in the Introduction, we adopt a unifying service-oriented perspective and model each of these resources as a service $S' \in \mathbf{S}$. We denote by $Dep_S \subseteq \mathbf{T}$ the overall set of dependencies needed by S. In case $Dep_S = \emptyset$, this means that S needs only the basic computing and communication resources of its hosting node $N_S \in \mathbf{N}$ to provide its functionality.

Given Dep_S, $Prov_S(t) \subseteq \mathbf{S}$ denotes the set of services to which S is bound at time t to resolve such dependencies (i.e., the *Providers* of S). For the sake of symmetry, as a given service S can be used by other services to fulfill their dependencies, $Req_S(t) \subseteq \mathbf{S}$ denotes the set of services using S at time t (*Requesters* of S).

We assume that Dep_S is fixed for each service and known in advance. We remark that we have included the time index t to $Prov_S(t)$ and $Req_S(t)$ to evidence that they represent dynamic state information, which can change over time. To this end, we adopt a discrete-time model indexed by $t = 0, 1, \ldots$, where we conventionally assume that the length of a time slot is $\delta = 1$ measured in some suitable time unit.

A service is said to be *fully resolved* (and hence fully functional) when all its dependencies are resolved, that is, when for all $d \in Dep_S$ there exists a fully resolved service $S' \in Prov_S(t)$ whose type matches d (Razian et al., 2022; Paolucci et al., 2002). In the scenario we are considering, the dependencies in Dep_S are dynamically resolved at runtime.

The overall assembly of services resulting from the bindings between each service S and other services used to resolve its dependencies can be represented as a directed graph $\mathbf{G}(t) = (\mathbf{S}, \mathbf{E}(t))$, where $\mathbf{S}$ represents the graph nodes and $\mathbf{E}(t) \subseteq \mathbf{S} \times \mathbf{S}$ is the set of graph edges. Specifically, a directed edge $(S_i, S_j) \in \mathbf{E}(t)$ denotes that, at time t, S_i is using S_j to resolve one of its dependencies (i.e., $S_j \in Prov_{S_i}(t)$). An assembly of services represented by a graph $\mathbf{G}(t)$ is *fully resolved* if for all $S \in \mathbf{S}$, S is fully resolved at time t.

Fig. 1 shows an assembly of services (including services S_1 - S_6) that illustrates the actual deployment of services on nodes N_1, N_2, N_3 and N_4. The figure reports two scenarios, Fig. 1a and b, where service dependencies are (fully) resolved in different ways. The set of types and dependencies for this example is reported in Table 2, where for each scenario we show the sets $Prov_S(t)$ and $Req_S(t)$. Fig. 1 also shows the computing and communication resources ($comp_i$ and $comm_i$) of each node N_i, used by each service hosted by that node for its operations.

Table 2
Service assembly: Types and dependencies.

S	*Type*	*Dep*	*Prov*	*Req*	*Prov*	*Req*
			Scenario 1		Scenario 2	
S_1	T_1	T_2	S_2	–	S_2	–
S_2	T_2	T_3	S_4	S_1	S_6	S_1
S_3	T_4	–	–	S_4	–	S_4
S_4	T_3	T_4	S_3	S_2	S_3	–
S_5	T_4	–	–	S_6	–	S_6
S_6	T_3	T_4	S_5	–	S_5	S_2

From the model presented in this Section, it follows that building and maintaining over time a graph $\mathbf{G}(t) = (\mathbf{S}, \mathbf{E}(t))$, where each $S \in \mathbf{S}$ is fully resolved, means providing an effective answer to the RQ0 stated in the Introduction. In the next section, to make the paper self-contained, we outline the main elements of a decentralized approach presented in Angelo et al. (2020), which achieves this goal. This procedure provides the basis for the approach we propose to get an answer to RQ1 and RQ2, which are the focus of this paper. This approach is discussed in the following Sections 4–8.

3. Service assembly

The fully decentralized service assembly approach summarized in this section drives the system modeled in Section 2 toward the construction and maintenance of a fully resolved service assembly (RQ0). The approach is based on previous work (Angelo et al., 2020). However, for the sake of readability, we summarize it herein.

The proposed approach adheres to the *Service Statelessness* design principle (Erl, 2005), and services do not maintain the interaction state between service invocations Hence, we assume that the state of a given interaction $(S_i, S_j) \in \mathbf{E}(t)$ between S_i and S_j is kept by S_i, and requests include all information necessary for their processing. Service statelessness improves (*i*) decoupling of interacting services, (*ii*) flexibility of the model, since it allows for easily rearranging the assembly at runtime, and (*iii*) scalability, by exploiting service caching and replication.

Referring to Fig. 2, the core idea underpinning this approach is to deploy at each node $N_i \in \mathbf{N}$ a self-adaptation mechanism architected according to the well-known MAPE-K model (Weyns et al., 2013), with *Monitor* (M), *Analyze* (A), *Plan* (P) and *Execute* (E) components, plus a *Knowledge* (K) component that maintains relevant information about the *Managed System* (e.g., service type, energy consumption, quality of service).

To collectively act as a *Managing System* for the *Managed System* composed of the nodes and the services deployed on them (see Fig. 2), the distinct MAPE-K loops should coordinate their activities. To this end, we adhere to the *Information Sharing Pattern* (Weyns et al., 2013), where each MAPE-K loop requires information from other nodes in the system to drive its self-adaptation activities.

Continuing with Fig. 2, each service (e.g., S_1) exposes its dependencies through an `invoke` interface and provides a `probe` interface to support the monitoring activities. The MAPE-K activities are carried out at each node by three different components, namely M (Monitor), AP (Analyze and Plan), and E (Execute).

The M (Monitor) component is responsible for the information dissemination activity. This activity allows for the update of the knowledge base (K) at each node N, thanks to the interaction with sibling M components hosted by the other nodes in the system. In turn, the AP component locally implements at each node the analysis and planning activities aimed at (*i*) analyzing the information kept by the knowledge K and (*ii*) selecting the services of interest that resolve the dependencies of the local services.

The dissemination activity performed by the M components is based on a classical advertisement schema (Meshkova et al., 2008), implemented through a Publish/Subscribe mechanism, which has been

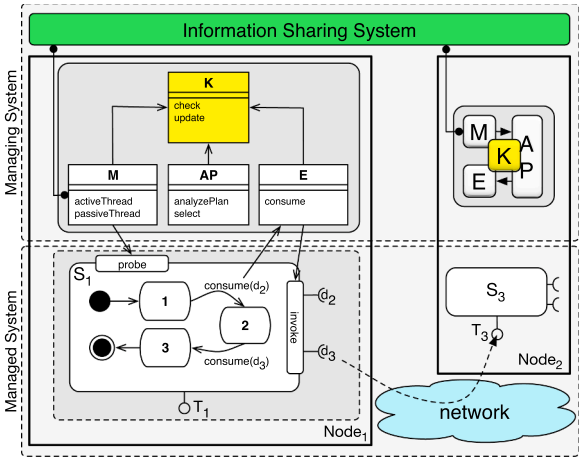


Fig. 2. Node architecture.

proven to be effective in pervasive computing environments (Nast et al., 2023). We refer to Angelo et al. (2020) for details on the adopted advertisement schema.

Based on the information disseminated in this way, the AP component performs its operations, implemented by the CHECK(), UPDATE() and SELECT() functions shown in Fig. 2. Some details of these functions are outlined below. As shown in Algorithms 1 and 2, the key element of both UPDATE() and SELECT() is the $P(rank, \alpha, h, sgn)$ function, fully described in Section 6, whose goal is to drive the operations of the functions UPDATE() and SELECT() to properly address RQ1 and RQ2.

UPDATE(). The UPDATE() function checks each message **m** received through the dissemination activity, to see whether an advertised service S' could be used to resolve some dependency $d \in Dep_S$ for any $S \in ServicePool_N$ (the set of services hosted by a node N). In the positive case, the function UPDATE() considers S' as a possible member of the set $Cand_{S,d}$ that collects the "best" known H or less candidates (according to $P(rank, \alpha, h, sgn)$) to solve the dependency d. The upper bound H on the cardinality of $Cand_{S,d}$ is a system parameter, set to a suitable value to limit memory occupancy. Algorithm 1 outlines the UPDATE() operations.

Algorithm 1 Function UPDATE().

// Input parameters
1: $ServicePool_N$; **m**; $P(rank, \alpha, h, sgn)$
// Algorithm
2: **for all** $S \in ServicePool_N$ **do**
3: **for all** $d \in Dep_S$ **do**
4: **for all** $S' \in$ **m do**
5: **if** $Type_{S'} = d$ **then**
6: $Cand_{S,d} \leftarrow Cand_{S,d} \cup \{S'\}$
7: **if** $|Cand_{S,d}| \leq H$ **then**
8: **continue**
9: **else** *// Drop the worst service to keep* $|Cand_{S,d}| \leq H$
10: $S^* \leftarrow$ "worst" $S \in Cand_{S,d}$ according to $P(rank, \alpha, h, sgn)$
11: $Cand_{S,d} \leftarrow Cand_{S,d} \setminus \{S^*\}$

CHECK() and SELECT(). The CHECK() function analyzes the knowledge K (i.e., the set of service candidates $Cand_{S,d}$ for each $S \in ServicePool_N$ and $d \in Dep_S$). This may happen proactively (e.g., every Δt time unit) or reactively (e.g., when values in the monitored data are detected outside predefined tolerance windows). Whenever the analysis performed by the CHECK() function notices a significant variation in K, then a new plan is required by calling the SELECT() function.

The SELECT() function selects from the set $Cand_{S,d}$ the "best" known service (according to $P(rank, \alpha, h, sgn)$) to resolve a dependency d, for any $d \in Dep_S$ and for any $S \in ServicePool_N$. Algorithm 2 outlines the SELECT() operations.

Algorithm 2 Function SELECT().

// Input parameters
1: $P(rank, \alpha, h, sgn)$
// Algorithm
2: **for all** $S \in ServicePool_N$ **do**
3: **for all** $d \in Dep_S$ **do**
4: $S^* \leftarrow$ "best" $S \in Cand_{S,d}$ according to $P(rank, \alpha, h, sgn)$
5: use S^* to resolve dependency d (add S^* to $Prov_S(t)$)

4. Energy model

In this section, we introduce the model and the associated notation that we will use to characterize the relevant energy-related information for the system we are considering. They integrate the model and notation introduced in Section 2. We will use them in Section 5 to define the metrics measuring the system's energetic autonomy, and in Section 6 to define the procedure that will drive the assembly process described in Section 3 towards achieving the goal of energetic autonomy.

We assume that the computing and communication resources associated with each node are the only direct sources of energy consumption, while the energy consumption of services in **S** is related to the use they directly or indirectly make of these resources.

4.1. Node energy model

The computing and communication resources present in each node draw the energy required for their operations from a local "green" source (e.g. solar or wind generator), whose production rate can vary over time or be temporarily null. In case of insufficient or null energy production, these resources can draw energy from a locally available battery, which is recharged by the locally produced excess energy. As we are interested in investigating the ability to be energetically autonomous, we do not consider the possible connection of the system nodes to a global power grid that could compensate for the lack of local energy. We use the following notation to characterize the energetic state of a node $N \in$ **N**:

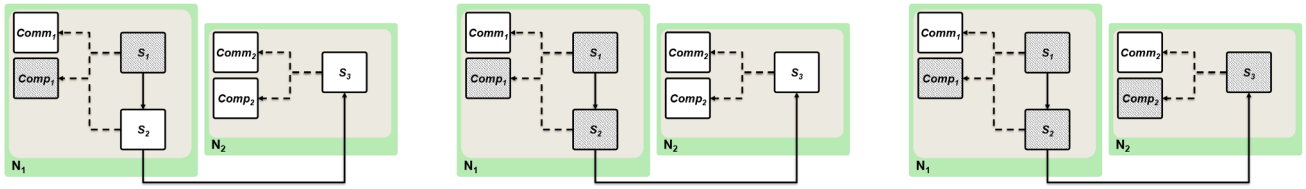
- $B_N(t)$: battery level at time slot t
- $G_N(t)$: energy generated within time slot t
- $C_N(t)$: energy consumed within time slot t

$G_N(t)$ is an uncontrollable parameter whose value can change over time, as is typical for renewable energy sources (see Appendix A). The value of $C_N(t)$ and $B_N(t)$ instead depends on the activities of the services hosted by N, and hence depends on how those services are assembled with other services. We assume that the values of $G_N(t)$, $C_N(t)$ and $B_N(t)$ are estimated by the monitor component at each node $N \in$ **N**.

4.2. Service energy model

When a service request arrives at a service $S \in$ **S**, S starts executing its internal operations, which consume energy because of the use of the computing and communication resources of its hosting node N_S. Besides this, the request arrived at S can generally trigger a flow of cascading requests addressed to services in $Prov_S(t)$, which in turn will cause additional energy consumption by the use of the computing and communication resources of their hosting nodes (which could include N_S, in case of co-located services).

As a first step in the definition of a model of the energy consumption for service S, we separately characterize in the following paragraphs the energy consumption caused, respectively, by the computation and the communication activities triggered by a *single request* addressed to



(a) Individual computation energy consumption of S_1 : $I^{comp}_{S_1}$

(b) Local computation energy consumption of S_1 : $L^{comp}_{S_1}(t)$

(c) Overall computation energy consumption of S_1 : $O^{comp}_{S_1}(t)$

Fig. 3. Visualization of the computation energy consumption indexes for the service S_1 in an example scenario (see shaded boxes).

S. In both cases, we will give two different characterizations that differ concerning the extent to which they take into account the cascading process of requests addressed to services in $Prov_S(t)$.

Computation energy. Let us denote by I^{comp}_S the average *individual* computation energy consumption of S, i.e., the energy directly consumed by S using the local computing resource of the hosting node N_S for its internal operations. Hence, I^{comp}_S does not consider at all the indirectly consumed computation energy caused by the use of services in $Prov_S(t)$. Moreover, for each $d \in Dep_S$, let us denote by $\mu_{S,d}$ the average number of times S requires the dependency d to fulfill a request it has received. For both I^{comp}_S and $\mu_{S,d}$, we assume that their value is known (e.g., through a monitoring activity locally performed at the node hosting S).

We now introduce the following two definitions:

- $O^{comp}_S(t)$: *overall* (system-wide) average computation energy consumption of a service S, which is equal to I^{comp}_S plus the computation energy indirectly consumed by S because of its use of services $S' \in Prov_S(t)$, wherever they are located in the system:

$$O^{comp}_S(t) = I^{comp}_S + \sum_{S' \in Prov_S(t)} \mu_{S,Type_{S'}} \cdot O^{comp}_{S'}(t) \tag{1}$$

- $L^{comp}_S(t)$: *local* (node level) average computation energy consumption of a service S, which is equal to I^{comp}_S plus the average computation energy indirectly consumed by S because of its use of services $S' \in Prov_S(t)$, but limited to services co-located with it on the same node:

$$L^{comp}_S(t) = I^{comp}_S + \sum_{\substack{S' \in Prov_S(t) \\ \wedge N_{S'} = N_S}} \mu_{S,Type_{S'}} \cdot L^{comp}_{S'}(t) \tag{2}$$

Fig. 3 shows, with a simple deployment scenario, the set of nodes, software services and computing services involved in the definition of each computation energy consumption index introduced above.

Communication energy. Let us denote by $K^{comm}_{S,S'}$ the average energy directly consumed by a service S using the local communication resources of N_S for a single interaction between S and service $S' \in Prov_S(t)$ (this cost is considered negligible and conventionally assumed to be zero when S and S' are hosted by the same node). Also in this case, we assume that the $K^{comm}_{S,S'}$ value is known (e.g., through a monitoring activity locally performed at the node hosting S).

We now introduce the following two definitions:

- $O^{comm}_S(t)$: *overall* (system-wide) average communication energy consumption caused by a single request addressed to S, which is equal to the overall communication energy directly consumed at N_S plus the communication energy consumed by services $S' \in Prov_S(t)$, wherever they are located:

$$O^{comm}_S(t) = \sum_{\substack{S' \in Prov_S(t) \\ \wedge N_{S'} \neq N_S}} \mu_{S,Type_{S'}} \cdot K^{comm}_{S,S'} + \sum_{S' \in Prov_S(t)} \mu_{S,Type_{S'}} \cdot O^{comm}_{S'}(t) \tag{3}$$

- $L^{comm}_S(t)$: *local* (node level) average communication energy consumption, which is equal to the overall communication energy directly consumed at N_S plus the communication energy consumed by services $S' \in Prov_S(t)$ co-located with S:

$$L^{comm}_S(t) = \sum_{\substack{S' \in Prov_S(t) \\ \wedge N_{S'} \neq N_S}} \mu_{S,Type_{S'}} \cdot K^{comm}_{S,S'} + \sum_{\substack{S' \in Prov_S(t) \\ \wedge N_{S'} = N_S}} \mu_{S,Type_{S'}} \cdot L^{comm}_{S'}(t) \tag{4}$$

Based on the definitions above, we now give two characterizations, which differ in their scope, of the *energy footprint* of a service $S \in \mathbf{S}$, defined as the overall average energy consumption caused by a *single request* addressed to S:

- *global (system-wide) energy footprint*, which includes the energy directly consumed by S and the energy indirectly consumed by S because of its use of services $S' \in Prov_S(t)$, wherever they are located in the system:

$$gef_S(t) = O^{comp}_S(t) + O^{comm}_S(t) \tag{5}$$

- *local (node level) energy footprint*, which includes the energy directly consumed by S and the energy indirectly consumed because of its use of services $S' \in Prov_S(t)$, but limited to services co-located with it on the same node:

$$lef_S(t) = L^{comp}_S(t) + L^{comm}_S(t) \tag{6}$$

We remark that, based on their definitions, $lef_S(t)$ requires only a *local* monitoring activity for its estimation, limited to the node hosting S, while $gef_S(t)$ requires the acquisition of information from other nodes.

5. Energetic autonomy metrics

We present the metrics that we use to measure the extent to which a given assembly of services achieves the energetic autonomy goal defined in the Introduction. We will use these metrics to assess the effectiveness of answering RQ1 of the management policies introduced in the next Section 6. In the definition of these metrics, our perspective is that the assembly of services achieves the goal of energetic autonomy as much as it can keep "alive" the nodes on which it is deployed, that is, with sufficient energy to support the operations of the services. We point out that this required energy includes not only the energy spent for the operations that implement the service functionality, but also the energy spent in complementary operations like service setup and shutdown.

As a first step, we define the following index that expresses the instantaneous energy-based node availability. We point out that it is defined from an energy perspective only, in the sense that a node is considered available if it can satisfy the energy demand of the services it hosts. Other possible causes of (un)availability are ignored in this definition:

- $A_N(t)$, $N \in \mathbf{N}$, $t = 0, 1, \ldots$: *node instantaneous availability*, defined as:

$$A_N(t) = \begin{cases} 1 & \text{if } B_N(t-1) + G_N(t) - C_N(t) > 0 \\ 0 & \text{otherwise} \end{cases} \tag{7}$$

Based on this definition, we now define the following energy-related system-wide index:

- $A_{SYS}(t)$, $t = 0, 1, \ldots$: *instantaneous system availability*, expressing the fraction of nodes in $\mathbf{N}$ that are available at time t, defined as:

$$A_{SYS}(t) = \frac{\sum_{N \in \mathbf{N}} A_N(t)}{|\mathbf{N}|} \tag{8}$$

$A_{SYS}(t)$ is a time-dependent index that can be used to trace how the availability of the system evolves over time as a consequence of variations in its energy state.

We also define the following time-independent index, which can be used as a figure of merit summarizing in a single number the system's energetic autonomy:

- $\overline{A}_{SYS}$: *steady-state system availability*:

$$\overline{A}_{SYS} = \lim_{t\to\infty} \frac{\sum_{\tau=0}^{t} A_{SYS}(\tau)}{t} \tag{9}$$

From (8) and (9) it follows that $0 \leq A_{SYS}(t), \overline{A}_{SYS} \leq 1$. From the energetic autonomy perspective, the optimal scenario corresponds to the case where these indices are equal to one, as it indicates that the system is able to keep its nodes fully operational, relying only on locally available/produced (green) energy, without resorting to an external power grid. Keeping these indices as close as possible to this optimal value thus represents the *energetic autonomy* goal we intend to pursue with the approach presented in the following sections.

6. Service ranking and selection

In this section, we first define (Section 6.1) the metrics that we will use to rank functionally equivalent services with respect to their suitability in meeting the energetic autonomy goal. Then, we present (Section 6.2) a *parameterized selection policy* that, based on the defined metrics, selects one service within a given set. Actually, this policy defines a template for generating a family of possible service selection policies. Each policy in this family can be instantiated by the assignment of specific values to the template parameters. This guarantees a high flexibility in the exploration of different policies. We will exploit this flexibility in the experiments discussed in Section 7, where we assess the effectiveness of different policies in answering RQ1 and RQ2.

6.1. Energy-aware service ranking metrics

Given a service S and a dependency $d \in Dep_S$, we recall that the assembly procedure outlined in Section 3 keeps updated at the node hosting S a set $Cand_{S,d}$, which collects functionally equivalent candidate services the node has become aware of to resolve d. The goal of the metrics we define here is to rank the services belonging to $Cand_{S,d}$, in order to drive the selection of one of them as a member of the set $Prov_S(t)$ (this selection is performed by the SELECT() function introduced in Section 3). The ultimate aim is to favor, thanks to this selection, the achievement of the energetic autonomy objective (RQ1).

We recall from the Introduction that we also intend to address the research question RQ2, which concerns getting insights about possible trade-offs between the effectiveness of the assembly process and the "complexity" of the indicators (metrics) used to drive its operations (with respect to the energetic autonomy goal). To this end, we define four different energy-aware ranking metrics with different complexity characteristics, as discussed at the end of this subsection. We classify these metrics into two groups. The first group includes two *service-based* metrics that rank services based on their energy footprint, as defined in Section 4.2. The rationale for these two metrics is that a service S' should be selected to resolve a dependency of a service S on the basis of the impact that its operations have on the overall energy consumption (as seen from the perspective of S, i.e., also taking into account the communication cost between S and S'). The second group includes two *node-based* metrics that instead rank services by considering the "energy wealth" of the node hosting them. The rationale for this second set of metrics is that a service should be selected, taking into account the amount of available energy it can rely on for its operations. As an example, in comparison with the metrics in the former set, these latter metrics could lead to prefer a service that consumes more energy (greater energy footprint), if it is hosted by a node with a fully charged battery and/or a higher energy generation rate.

Service-based metrics.

Table 3
Overhead for energy-aware service ranking metrics.

ranking	monitoring	dissemination
metric	overhead	overhead
$ge_{S,S'}(t)$	*high*	*high*
$le_{S,S'}(t)$	*high*	*low*
$ne_{S'}(t)$	*low*	*high*
$rl_{S'}(t)$	*low*	*high*

- *global energy* metric $ge_{S,S'}(t)$: this metric ranks $S' \in Cand_{S,d}$ according to its system-wide energy footprint $gef_{S'}(t)$, defined by Eq. (5), plus the communication cost between S and S':

$$ge_{S,S'}(t) = gef_{S'}(t) + \mu_{S,Type_{S'}} \cdot K^{comm}_{S,S'} \tag{10}$$

- *local energy* metric $le_{S,S'}(t)$: this metric has a more limited scope than $ge_{S,S'}(t)$, as it ranks $S' \in Cand_{S,d}$ according only to its local energy footprint $lef_{S'}(t)$ (Eq. (6) if S and S' are co-located, otherwise according to the communication cost between S and S':

$$\begin{aligned} le_{S,S'}(t) = {} & lef_{S'}(t) \cdot U_{\{N_{S'}=N_S\}} \\ & + \mu_{S,Type_{S'}} \cdot K^{comm}_{S,S'} \cdot U_{\{N_{S'}\neq N_S\}} \end{aligned} \tag{11}$$

where $U_{\{cond\}}$ is the indicator function that holds 1 when condition $cond$ is true, and 0 otherwise.

Node-based metrics.

- *node energy* metric $ne_{S'}(t)$: this metric ranks $S' \in Cand_{S,d}$ according to the current battery level of the node that hosts it ($N_{S'}$), defined as:

$$ne_{S'}(t) = B_{N_{S'}}(t) \tag{12}$$

- *residual lifetime* metric $rl_{S'}(t)$: this metric ranks $S' \in Cand_{S,d}$ according to the currently estimated residual lifetime of $N_{S'}$; the estimate is based on the current values of the node battery level and energy production/consumption rate:

$$rl_{S'}(t) = \begin{cases} \frac{B_{N_{S'}}(t)}{C_{N_{S'}}(t) - G_{N_{S'}}(t)} & \text{if } C_{N_{S'}}(t) - G_{N_{S'}}(t) > 0 \\ \infty & \text{otherwise} \end{cases} \tag{13}$$

Discussion. We evidence some characteristics of the defined metrics:

(i) $ge_{S,S'}(t)$ and $le_{S,S'}(t)$ are "lower-is-better" metrics, while $ne_{S'}(t)$ and $rl_{S'}(t)$ are "higher-is-better" metrics;

(ii) $ne_{S'}(t)$ and $rl_{S'}(t)$ can be considered as "simple" metrics with respect to the information needed for their measurement, as they require the collection of relatively easy-to-measure coarse-grained physical quantities (battery level, charge/discharge rate). Comparatively, the $ge_{S,S'}(t)$ and $le_{S,S'}(t)$ metrics are more complex, as they require finer-grained measurements (computation and communication energy consumed by the activation of each service).

(iii) when S and S' are not colocated, the estimation at the node hosting S of the metrics $ge_{S,S'}(t)$, $ne_{S'}(t)$ and $rl_{S'}(t)$ requires the acquisition of information disseminated by other nodes; $le_{S,S'}(t)$ is instead the only metric whose estimation at the node hosting S requires only locally monitored information (i.e., $lef_{S'}(t)$, $\mu_{S,Type_{S'}}$ and $K^{comm}_{S,S'}$, see Section 4.2), independently of whether S and S' are colocated or not. Hence, in terms of the communication overhead, $le_{S,S'}(t)$ is a "simpler" metric than $ge_{S,S'}(t)$, $ne_{S'}(t)$ and $rl_{S'}(t)$.

Table 3 summarizes the characteristics discussed above of the four considered metrics and gives a qualitative assessment of the monitoring and dissemination overhead caused by the MAPE-K loop (see Fig. 2). It is worth noticing that we consider the computation overhead of the analysis and planning activities negligible, because it only consists of updating the efficiency estimator of the policies discussed in the next section.

6.2. Parameterized service selection policy

The functions UPDATE() and SELECT() introduced in Section 3 rely on the parameterized selection policy described in this section. It is based on a generalization and adaptation of a method originally proposed in Schaerf et al. (1995) for a scenario of decentralized load balancing in a distributed system with load-dependent QoS. The key elements of our policy are: (*i*) the *forecasting method* used to update the value of the adopted service ranking metric and (*ii*) the *selection criterion* used to select a service within a given set of candidates, based on the ranking value associated with each of them. We denote this policy by $P(rank, \alpha, h, sgn)$, where $rank, \alpha, h, sgn$ are the policy parameters. In the following two paragraphs, we explain in detail their role in the definition of the two key policy elements.

Forecasting method. We denote by $rank$ the ranking metric used to drive the selection process; in our setting, we have $rank \in \{ge_{S,S'}, le_{S,S'}, ne_{S'}, rl_{S'}\}$. The values of these metrics are kept up to date at each node, thanks to data locally monitored or obtained through the dissemination procedure described in Section 3. The update procedure is based on the simple exponential smoothing method (Hyndman and Athanasopoulos, 2021):

$$\widehat{rank}_{new} = \alpha \cdot rank_{last} + (1 - \alpha) \cdot \widehat{rank}_{old} \tag{14}$$

where $\widehat{rank}_{new}$ is the new forecast for the value of $rank$, $\widehat{rank}_{old}$ is the previous forecast and $rank_{last}$ is the last collected value (by direct observation or calculated on the basis of disseminated data), while α is the smoothing parameter, with $0 < \alpha \leq 1$.

When $\alpha = 1$, Eq. (14) defines a "memoryless" forecasting method based only on the last observed value; otherwise, α values increasingly close to zero give more and more weight to the accumulated past experience (for example, if α is set equal to $1/k$, where k is the number of collected observations for $rank$ up to the present time, Eq. (14) corresponds to the sample mean of $rank$ that, by definition, gives increasingly more weight to past observed values).

Selection criterion. The goal of this criterion is to pick a single service within the set $Cand_{S,d}$ of candidates. We adopt a probabilistic method to this end that depends as follows on the parameters h and sgn, where $h \in \mathbb{N}$ and $sgn \in \{-1, +1\}$. Given a service $S' \in Cand_{S,d}$ and its ranking value $rank$, we define the following function:

$$\overline{p(S')} = rank^{(sgn)\cdot h} \tag{15}$$

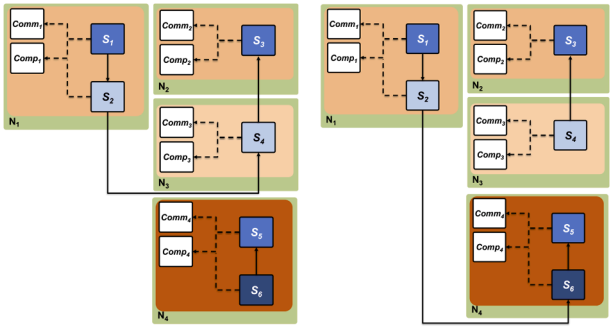
where $sgn = +1$ if $rank$ is a "higher-is-better" metric, while $sgn = -1$ if it is a "lower-is-better" metric. Then, we normalize $\overline{p(S')}$ in the interval $[0, 1]$:

$$p(S') = \frac{\overline{p(S')}}{\psi} \tag{16}$$

where $\psi = \sum_{S' \in Cand_{S,d}} \overline{p(S')}$ is a normalization factor. Finally, we interpret $p(S')$ as a *selection probability* associated with S' and use it to probabilistically select a service within $Cand_{S,d}$.

According to Eqs. (15) and (16), different values of parameter h lead to different probabilistic selection policies:

- $h = 0$: *random* policy that selects with uniform probability any service in $Cand_{S,d}$. We will use this policy as a baseline policy against which we will compare the other ones that we will consider;
- $h = 1$: *weighted fair* policy that selects a service in $Cand_{S,d}$ with probability proportional to the rank associated with that service through metric $rank$;
- $h \geq 2$: *biased-towards-best* policy that, for increasing values of h, selects a service in the set $Cand_{S,d}$ with probability increasingly biased towards the "best" service in that set, according to metric $rank$.
- For $h \to \infty$ we obtain a *greedy* policy that always selects the "best" service, i.e., the service with the maximum (minimum) $rank$ value within the set $Cand_{S,d}$ (as noted in Schaerf et al. (1995), a value $h \geq 20$ is sufficient to get a greedy behavior).



(a) Service-based metric (b) Node-based metric

Fig. 4. Assembly decisions driven by different service ranking metrics.

It should be noted that, for increasing values of h, the selection policy increases its degree of exploitation with respect to exploration. The dualism between exploration and exploitation in learning techniques is an important aspect that has long been investigated (Ishii et al., 2002). A high degree of exploration allows algorithms to rapidly traverse the solution space in search of the best achievable result to the detriment of the short-term solution's effectiveness. The extreme case of an explorative algorithm is the random policy. On the other hand, a high level of exploitation favors the best-known solution at the disadvantage of missing better candidates from an unknown set. In most cases, it is not trivial to establish the right balance between exploration and exploitation in view of the complexity of the task these techniques are required to solve.

An example of the different selection decisions driven by the ranking metrics we have defined is shown in Fig. 4, which depicts the same service assembly scenario already shown in Fig. 1 (the service types and dependencies are those reported in Table 2). We use a "color code" to graphically illustrate an example of the possible relative values of service-based and node-based metrics, where in both cases a darker tone is intended to represent a higher value of the metric: in particular, (i) different orange tones represent different values of a node-based metric (e.g., the *node energy* metric $ne_{S'}(t)$ defined by Eq. (12); (ii) different blues tones represent different values of a service-based metric (e.g., the *global energy* metric $ge_{S,S'}(t)$ defined by Eq. (10). Consider the two functionally equivalent services S_4 and S_6: they both offer a service of type T_3 and hence any of them could be used to solve the dependency of S_2. If we rank these services according to a service-based metric, then S_4 is better than S_6, from the perspective of a service requesting a type T_3 service: the lighter blue tone of S_4 indicates that each invocation of S_4 causes a lower energy consumption; this, hopefully, should reduce the possibility that requests made by S_2 for a T_3 service leads some node to run out of energy, thus becoming unavailable. Fig. 4a shows the assembly decision for S_2 that could result from this ranking. On the other hand, if we rank the services according to a node-based metric, then S_6 is better than S_4: the darker orange tone indicates that node N_4 hosting S_6 has a higher battery level than node N_3 hosting S_4, which means that S_6 can rely on a higher energy budget for its operations. Fig. 4b shows the assembly decision for S_2 that could result from this alternative ranking. It is not easy to decide a priori which of the two selection decisions leads to better results. In the next section, we investigate both this concern and the balance between exploitation and exploration discussed above.

7. Experimental results

In this section, we experiment with the different policies to address the research questions RQ1 and RQ2. To this end, we compare the performance of ranking metrics (Section 6.1) and selection policies (Section 6.2), under various conditions, focusing on how effectively they

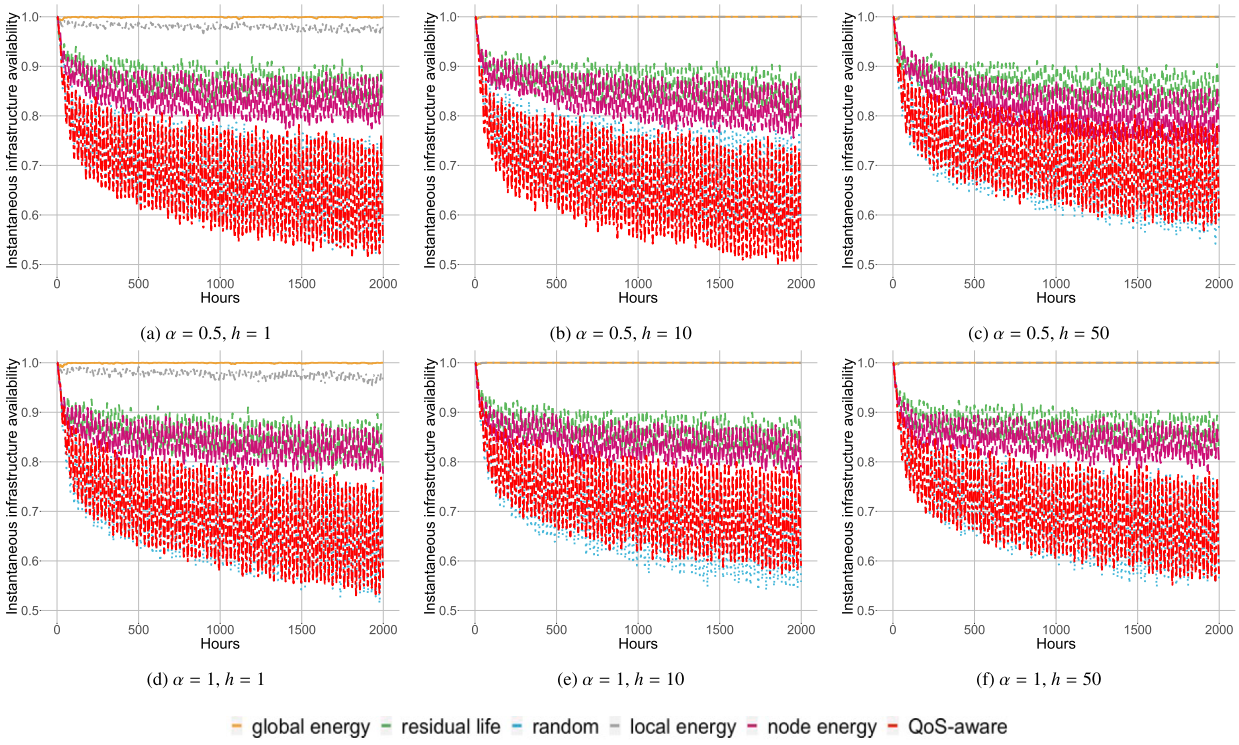


Fig. 5. $A_{SYS}(t)$ in scenarios with green energy source (solar panel) and battery (small). (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

support the system in maintaining energetic autonomy. By analyzing their behavior, we aim to identify which policy offers the most balanced and practical approach to energy-aware self-adaptation.

As remarked at the end of Section 3, the decentralized service assembly procedure described there eventually leads to the creation of a fully resolved assembly. This result is achieved independently of the rule used by the UPDATE() and SELECT() functions to update the content of the set of functionally equivalent services $Cand_{S,d}$ (all services in the set implement d), and to select within it the service to be included in $Prov_S(t)$. In this section, we assume that the UPDATE() and SELECT() functions introduced in Section 3 operate according to the selection policy described in Section 6.2. In particular, UPDATE() updates $Cand_{S,d}$ possibly dropping from it the "worst" ranked service according to $P(rank, \alpha, h, sgn)$, and SELECT() selects from $Cand_{S,d}$ the service to be included in $Prov_S(t)$ as the "best" ranked service according to $P(rank, \alpha, h, sgn)$.

Different instances of $P(rank, \alpha, h, sgn)$ (corresponding to different selections among the possible values for its parameters $rank, \alpha, h, sgn$) lead to the construction of assemblies that, albeit being functionally equivalent, have different energetic autonomy characteristics. We thus present in this section a set of simulation experiments to assess the effectiveness of different service selection policies (instances of $P(rank, \alpha, h, sgn)$) with respect to the issues raised by RQ1 and RQ2. To this end, we experiment with instances of $P(rank, \alpha, h, sgn)$ obtained by making its parameters range over the set of values shown in Table 4, for a total of twenty-four different policies. We have selected these values to cover a large set of different regions of the overall space of all possible policies that can be instantiated from $P(rank, \alpha, h, sgn)$. In particular, the possible values for the *rank* parameter are those described in Section 6.1. For the other two parameters α and h, we recall from Section 6.2 that $\alpha = 1$ defines a memoryless policy that considers only the last observed value of the ranking metric, while $\alpha = 0.5$ defines a policy that takes into account also past values. On the other hand, the three values selected for the h parameter define selection policies ranging from a probabilistic service selection strictly proportional to the services' rank ($h = 1$), to a probabilistic service selection biased towards the best-ranked service ($h = 10$), to a greedy policy that (almost surely) selects the best-ranked service ($h = 50$).

Table 4
Parameters of the service selection policy.

parameter	*possible values*
rank	$ge_{S,S'}, le_{S,S'}, ne_{S'}, rl_{S'}$
α	0.5, 1
h	1, 10, 50

In addition, for comparison, we consider two other policies:

- a baseline *Random* policy (corresponding to $h = 0$) that randomly selects a service within the set of known functionally equivalent candidates. This policy serves as a benchmark to compare the "smartness" of the other considered policies with respect to a blind selection policy;
- a *QoS-aware* policy taken from Angelo et al. (2020), which selects a service using a QoS-based ranking metric. To this end, we assume that each service has an associated QoS index, measuring some (non-energy) QoS-related aspects of the service (e.g., performance, reliability, cost, or some suitable combination of them). According to Angelo et al. (2020), we assume that this index is load-dependent: the higher the load addressed to a service, the worse the value of its QoS index. This policy serves as a benchmark to compare the impact on the system's energetic autonomy of a QoS-focused criterion with respect to the energy-focused selection criteria adopted in this paper.

We conduct experiments with all these selection policies for two kinds of scenarios named *energetic autonomy* and *energetic survivability*, respectively.

In the *energetic autonomy* scenario, we consider nodes equipped with a local green energy generator and a battery that is recharged by excess produced energy and compensates as long as possible for temporary interruptions in energy generation. In this scenario, we investigate to

what extent the different policies can maintain the functionality of the system over time, as measured by the availability indices presented in Section 5.

In the *survivability* scenario, we consider nodes equipped only with a (initially fully charged) battery. In this case, we investigate how long the different policies are able to maintain some functionality and how quickly they degrade before the definitive collapse.

Both the *energetic autonomy* and the *survivability* scenarios do not consider the existence of an external source of energy (e.g., power grid) that could potentially compensate for the energy in the batteries and/or energy gathered by local green energy generators (e.g., solar panels). Indeed, our focus is to investigate how much the system can do without it.

To carry out the experiments, we implemented a large-scale simulation model for the PeerSim simulator (Montresor and Jelasity, 2009). The replication package is publicly available to researchers interested in replicating and independently verifying the results presented in this paper[2].

7.1. Experiments settings

Our experimentation mimics a decentralized computing scenario where $|\mathbf{S}|$ services with NUM_INT different types $\mathbf{T} = \{T_1, T_2 \ldots, T_{\text{NUM\_INT}}\}$ are deployed on a network of $|\mathbf{N}|$ nodes.

We randomly position the network nodes in an area with a diameter of D meters and assume each node hosts $node\_serv$ services, randomly selected from the $\mathbf{S}$ set. Each node is equipped with a solar panel and a battery (in the experiments, we consider two different battery capacities, respectively named $small = 3000mAh$ and $big = 6000mAh$; see also Appendix B). We adopt a suitable model to keep track of the solar radiation variations at different day times, according to a 24-hour cycle. Details about this model are reported in the Appendix A.

Each simulation run simulates an observation period of 2000 hours for a network with $|\mathbf{N}| = 50$ nodes, $|\mathbf{S}| = 250$ services, and NUM_INT $= 10$ service types, deployed in an area of diameter $D = 200$ meters[3]. Estimates for each quality index of interest are calculated from 50 simulation runs. During the simulation, a node without sufficient energy is not available for the assembly and is made available again once its charging state reaches at least 20 % of the node's battery capacity.

The values of the simulation parameters (e.g., battery capacities, solar panel energy generation rate) are taken from wireless sensor network literature (see Appendices A and B).

7.2. Energetic autonomy in green-powered scenarios

Tables 5 and 6 show the value of the steady-state availability $\overline{A}_{SYS}$ (Eq. (9)) for the different combinations of policy parameter values, and the two different battery capacities. For the sake of simplicity, we display only the midpoint of the calculated confidence intervals. Both tables show that all the considered energy-aware policies perform consistently better than the baseline random policy. A noteworthy result is that, from the energetic autonomy perspective, QoS-aware selection has a very negative impact, as its performance is almost indistinguishable from the random policy.

Besides this, the tables show that the two energy-aware policies that base their selection on information about the service energy footprint (*global* and *local energy* policies) clearly outperform the two policies that base their selection on information about the energy wealth of the node hosting the service (*residual life* and *node energy* policies). Indeed, the $\overline{A}_{SYS} \approx 1$ value for the former two policies indicates that they can keep almost all the nodes active, thus guaranteeing (on average) full-service

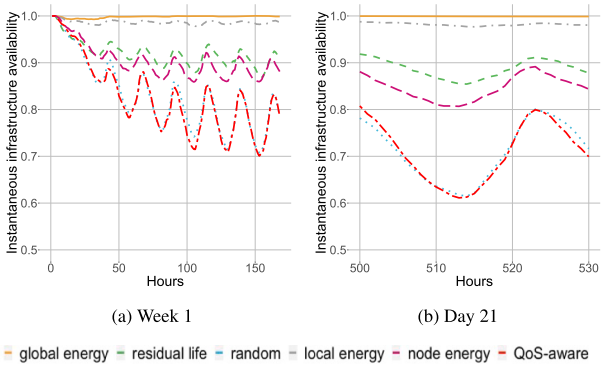


Fig. 6. Zoom-in at week 1 (a) and day 21 (b) of $A_{SYS}(t)$, in a scenario with solar panel and a small battery, $\alpha = 0.5$, $h = 1$.

Table 5
$\overline{A}_{SYS}$: Steady-state availability values (**small** battery).

α	0.5	0.5	0.5	1	1	1
h	1	10	50	1	10	50
global energy ($ge_{S,S'}$)	0.99	0.99	0.99	0.99	0.99	0.99
local energy ($le_{S,S'}$)	0.98	0.99	0.99	0.98	0.99	0.99
residual life ($rl_{S'}$)	0.87	0.87	0.87	0.85	0.86	0.87
node energy ($ne_{S'}$)	0.84	0.84	0.82	0.84	0.84	0.85
random	0.69	0.69	0.70	0.69	0.70	0.70
QoS-aware	0.68	0.67	0.72	0.72	0.71	0.70

continuity. On the other hand, the $\overline{A}_{SYS} < 0.9$ value for the latter two policies indicates that they instead cause some nodes to run out of energy.

The plots of $A_{SYS}(t)$ (Eq. (8)) in Figs. 5 and 7 provide finer-grain insights about the effectiveness of the considered policies by showing how they perform with respect to the optimal value $A_{SYS}(t) = 1$. The figures clearly show that the *global energy* policy provides the best result, constantly guaranteeing over time the full availability of all nodes ($A_{SYS}(t) \approx 1$ for all t values), despite the intermittent availability of solar energy. The *local energy* policy provides results almost indistinguishable from the *global energy* one, or slightly worse.

On the other hand, the plots for the *residual life* and *node energy* policies confirm their less satisfactory performance. Indeed, apart from the initial phase, where they take advantage of the full battery charge (as defined in our experiment settings), they quickly stabilize around $A_{SYS}(t)$ values that are quite far from 1. Besides this, a (negatively) noteworthy behavior of these two policies is given by the oscillations of the $A_{SYS}(t)$ value, which are particularly pronounced in the small battery case (Fig. 5). Because of them, the actual number of available nodes (and consequently of active services) may temporarily drop to values much smaller than the average. These oscillations follow a 24-hour period and are thus clearly correlated with the solar energy availability (as evidenced in Fig. 6, which shows a zoom-in of Fig. 5a around the first week and the 21st simulated day, respectively). This indicates that, with respect to the *local* and *global energy* policies, the *residual life* and *node energy* policies are less effective in utilizing the backup energy provided by the batteries to mask the temporary unavailability of green energy sources.

Finally, Figs. 5 and 7 confirm the bad and comparable performance of the random and QoS-aware policies.

7.3. Energetic survivability in battery-only scenarios

In these experiments, we assume that, for some reason, the green energy source is not available at all for all nodes. Consequently, each node will remain active as long as its battery provides sufficient energy. Eventually, all the nodes will exhaust their batteries, and the system availability will drop to zero. Hence, it does not make sense to con-

[2] https://github.com/mi-da/Self-Sustainable-Service-Assembly

[3] These numbers are similar to several WSN environmental monitoring scenarios, for example, in air quality monitoring (Han et al., 2019) or smart farming (Bandara et al., 2020).

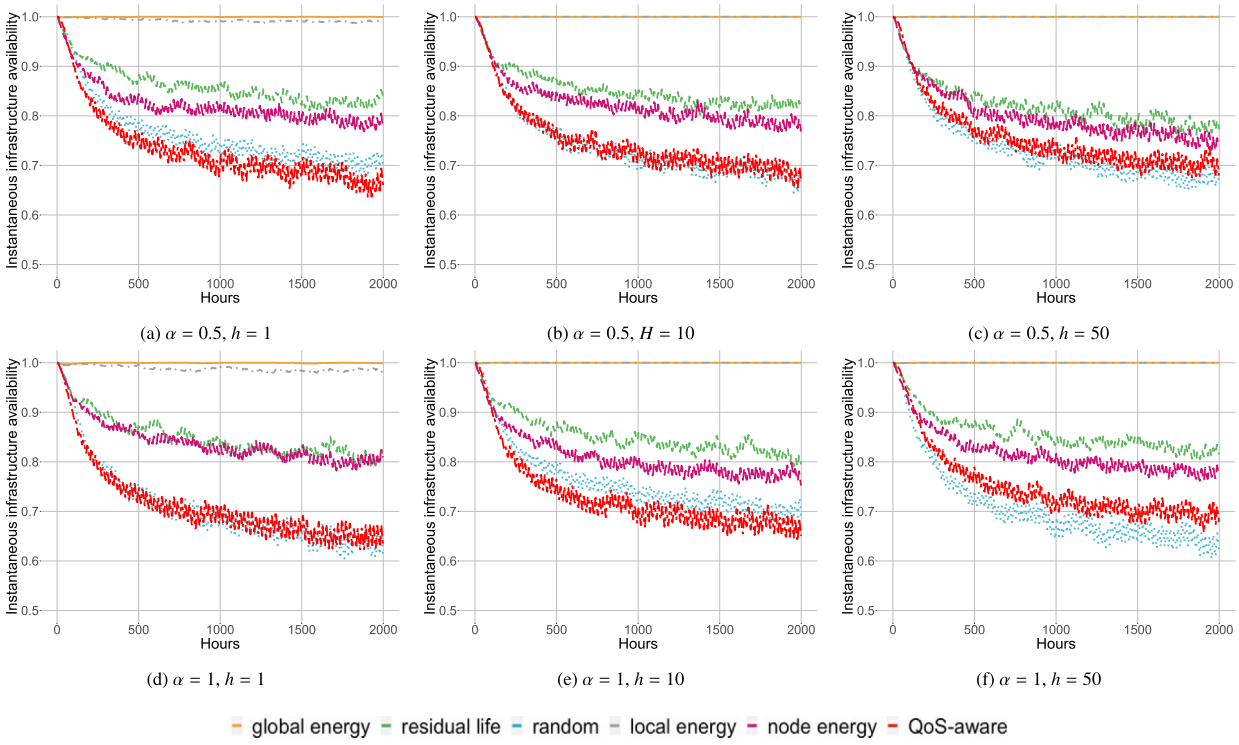


Fig. 7. $A_{SYS}(t)$ in scenarios with green energy source (solar panel) and battery (big). (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

Table 6
$\overline{A}_{SYS}$: Steady-state availability values (**big** battery).

α	0.5	0.5	0.5	1	1	1
h	1	10	50	1	10	50
global energy ($ge_{S,S'}$)	0.99	0.99	0.99	0.99	0.99	0.99
local energy ($le_{S,S'}$)	0.99	0.99	0.99	0.99	0.99	0.99
residual life ($rl_{S'}$)	0.86	0.85	0.82	0.85	0.85	0.86
node energy ($ne_{S'}$)	0.82	0.82	0.80	0.84	0.81	0.82
random	0.76	0.74	0.74	0.70	0.76	0.70
QoS-aware	0.73	0.74	0.75	0.71	0.73	0.75

sider the steady-state availability A_{SYS} (Eq. (9)). Figs. 8 and 9 show instead the plots of the instantaneous availability $A_{SYS}(t)$ (Eq. (8)), in the small and big battery cases, respectively (in all cases, we assume that all nodes start with a fully charged battery). As expected, the figures show that $A_{SYS}(t)$ drops towards zero for increasing values of t. However, the figures clearly show that the *global* and *local energy* policies vastly outperform the *residual life* and *node energy* policies in guaranteeing the system's survivability in the considered critical situation. Another interesting result that can be observed from these plots is that the value of parameter h impacts the system survival ability, differently from the previous experiments where a green energy source was available. Indeed, both Figs. 8 and 9 show that the system survival ability benefits from some amount of greediness in service selection (i.e., $h > 1$).

Finally, Figs. 8 and 9 show the very negative impact of the QoS-aware policy on the system's survivability.

8. Learned lessons: Answers to RQ1 and RQ2

We have considered two types of metrics to rank functionally equivalent services, thus driving their selection during the dependencies resolution process in an assembly of services: (i) metrics based on the energy footprint of the service itself (*global energy* $ge_{S,S'}$ and *local energy* $le_{S,S'}$), and (ii) metrics based on the energy wealth of the node hosting the service (*residual lifetime* $rl_{S'}$ and *node energy* $ne_{S'}$).

We briefly recall the considerations made in Section 6.1 about the monitoring and dissemination overhead caused by the MAPE-K loop. With respect to the monitoring effort at each node, metrics (i) cause a higher overhead with respect to metrics (ii), as they require finer-grained measurements. With respect to the information dissemination overhead, it is low for the *local energy* metric $le_{S,S'}$, as it only requires the use of locally collected information for its estimation; the dissemination overhead is instead higher for the other three metrics, as the exchanged messages should carry the information needed for their estimation, in addition to the functional information required by the dependency resolution procedure.

From the experimental results discussed in Section 7, we see that the selection policies based on the metrics *global energy* $ge_{S,S'}$ and *local energy* $le_{S,S'}$ (type (i)) vastly outperform those based on the other two policies (type (ii)). In particular, for the "energetic autonomy" scenario (green energy source plus battery), they allow achieving values of the considered metrics very close to their maximum (= 1), while this is not the case for the other policies. This provides a strong indication that, with respect to the monitoring overhead, type (i) policies are worth the greater effort required for their implementation, in the perspective of guaranteeing the energetic autonomy of green-powered service assembly.

With respect to the information dissemination overhead, a positive outcome of the experiments is that system-wide dissemination of non-functional information about the energetic profile of each service or node is unnecessary. Indeed, the policy based on the *local energy* metric $le_{S,S'}$, whose dissemination overhead is low, reaches results very close to those achieved by the policy based on the *global energy* metric $ge_{S,S'}$, which appears to be the optimal one. This implies that the adoption of $le_{S,S'}$ considerably reduces the amount of information to be disseminated (and the consequent energy overhead), limiting it to only functional information (i.e., service types needed to build and maintain a fully resolved assembly), with negligible impact on energy performance.

In summary, these results indicate that we can identify a service selection policy (based on the $le_{S,S'}(t)$ ranking metric) that, used in com-

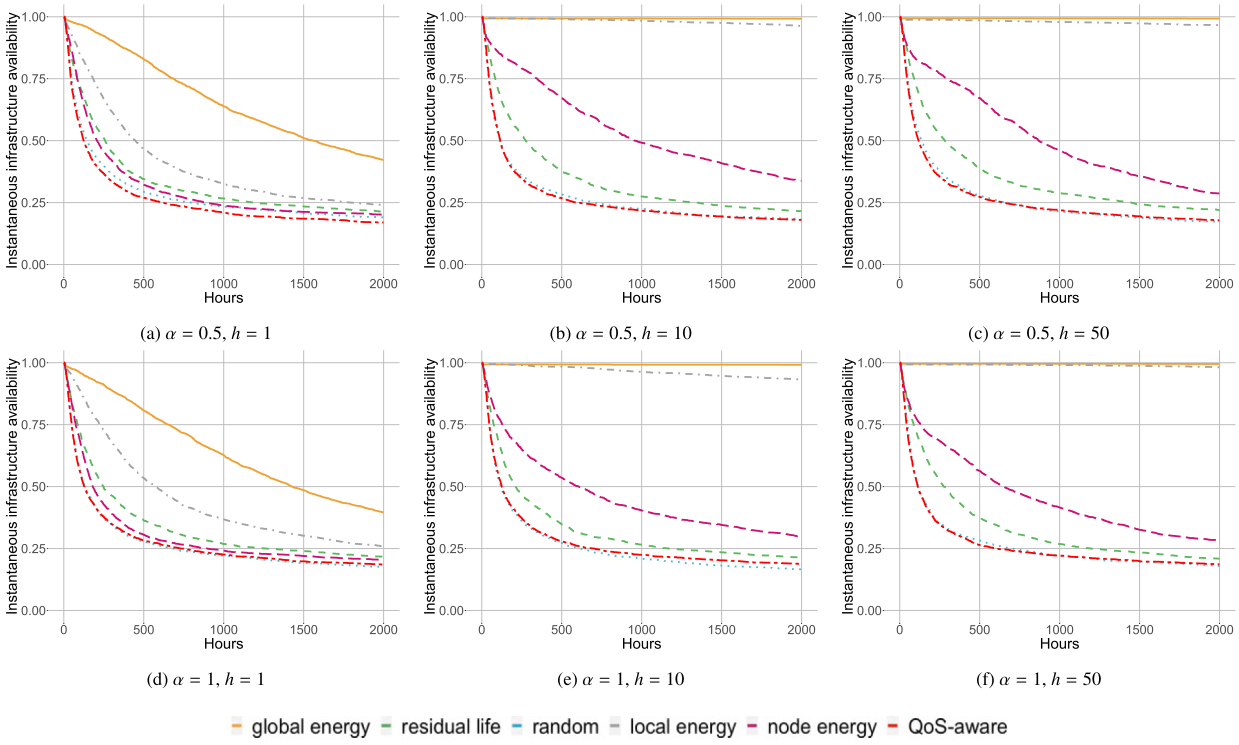


Fig. 8. $A_{SYS}(t)$ in scenarios with battery only (small).

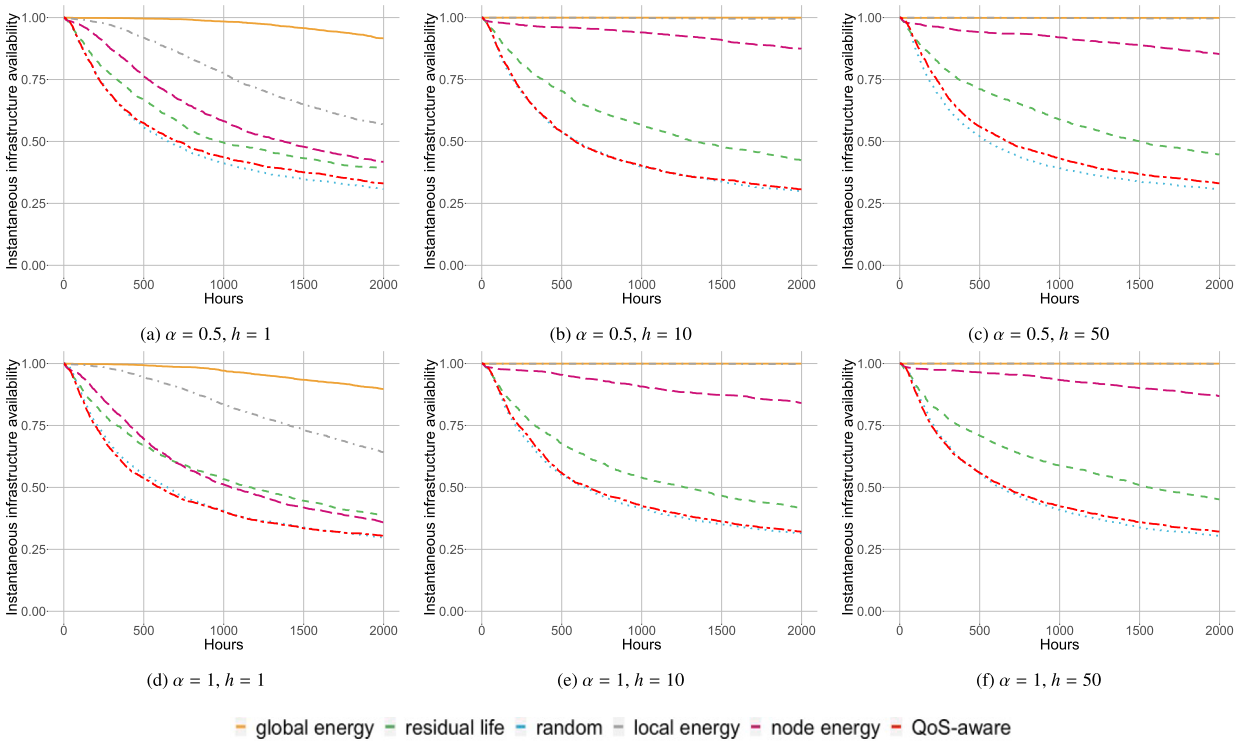


Fig. 9. $A_{SYS}(t)$ in scenarios with battery only (big).

bination with a decentralized assembly procedure, allows us to achieve two goals: (*a*) building and maintaining over time an energetically autonomous fully resolved assembly of services (RQ1), and (*b*) achieving this result paying a limited energy overhead (RQ2).

As a final remark, the experiments in Section 7 also show that driving the service selection by QoS-focused ranking metrics without considering the energetic profile of each service might have a dramatically negative impact on the system's energetic autonomy. This indicates that when QoS issues other than energy are considered relevant, they should be carefully balanced with energetic issues. We leave it to future work to investigate the trade-off between these two issues.

8.1. Threats to validity

There are some potential threats to the validity (Wohlin et al., 2012) of the proposed approach.

A threat to *internal validity* is represented by the selection of the self-sustainability indexes. To smooth this threat, we adopted different metrics that looked at self-sustainability from different perspectives (e.g., service- and node-based, local and global scope).

Construct validity: We mitigated construct validity threats using a large-scale simulation model for our experimentation in the PeerSim simulator (Montresor and Jelasity, 2009). PeerSim has been developed with extreme scalability and support for dynamism in mind, and it is released to the public under the GPL open-source license. We use the simulator's cycle-based engine to allow for scalability. The cycle-based engine uses some simplifying assumptions, such as ignoring the details of the transport layer in the communication protocol stack. Since our results focus on the application layer energy behavior, simplifying assumptions do not impact the validity of our experiments. However, as our evaluation is based on simulation rather than execution on physical hardware, we could not directly monitor or measure the actual energy consumption of the system. Instead, we estimated energy behavior based on modeled parameters and abstractions defined at the application level. While this approach enables large-scale and repeatable experimentation, it inevitably limits the precision of energy measurements compared to empirical observations from real deployments. We plan to address this limitation in future work by conducting real-world experiments and measuring energy consumption through appropriate hardware (e.g., Otii Arc[4]) and software monitoring tools (e.g., Prometheus[5]). Finally, the solar radiation model implemented has been constructed and validated against real data captured at the Bioparco of Rome (Stazione meteo di bioparco di roma, 2023) (as illustrated in the Appendix A). The energy consumption model and its parameters have been retrieved from previous literature and have been proven to model electronic and amplifier power consumption (Heinzelman et al., 2002; Ikpehai et al., 2019; Kang et al., 2006) in a fairly accurate way.

Conclusion validity: We mitigated conclusion validity threats by using a large number of simulation repetitions to reduce the confidence interval of the simulation results. Indeed, this threat is inherent to all simulated systems, as probabilistic functions govern their behavior (Banks et al., 2013). For this reason, each experiment has been executed 50 times.

A threat to *external validity* concerns the approach's evaluation. Indeed, we adopted an evaluation based on *extensive simulations*, instead of considering single case studies, to perform a general analysis of the approach's effectiveness. Specifically, we focused on experimenting with an extensive range of possible selection policies to identify the most promising ones from the self-sustainability perspective. In future work, we plan to extend the evaluation by investigating how the identified policies behave with respect to different settings of the system parameters, according to real-world cases.

9. Related work

While a large body of related work exists on the addressed topic, we summarize hereafter only those papers that are closely related to the proposed approach and concern *IoT and edge computing scenarios*, *service composition and service assembly*, and *energy awareness*.

Reference scenario. Our work concerns the dynamic assembly and composition of services in an IoT and edge computing scenario. Some papers present examples of this scenario and focus on related issues. As an example, Nasir et al. (2022) present NexusEdge, a system based on a decentralized IoT-edge architecture that enables applications to run on edge gateways without the support of the cloud. Examples of dynamic IoT environments can also be found in Razian et al. (2022), Sun et al. (2019), Khanouche et al. (2016) and Halba et al. (2023).

Service composition and service assembly. Service composition refers to the process of creating and sustaining a *new value-added* service using existing services in a dynamic environment. Research in this field can be categorized into two main categories (Rao and Su, 2004). Category 1 ($C1$) involves dynamically decomposing a general task into a set of subtasks that can be fulfilled by existing services, whose operations must then be appropriately coordinated. Hence, the composition process can lead to different decompositions of the overall task, depending on the available services (e.g., Razian et al. (2022), Chen et al. (2018), Gabrel et al. (2018) and Rodriguez-Mier et al. (2016)). The second category ($C2$) instead includes works that assume that the general task to be carried out is already divided into predetermined activities, specified through a workflow using specific composition patterns (such as sequence, parallel, iteration, and conditional selection), before starting the service composition process. The objective here is to identify the already-existing services that carry out the predetermined activities (e.g., Razian et al. (2022), Rao and Su (2004), Cardellini et al. (2012) and Lemos et al. (2015)).

The *service assembly* problem we consider in this paper focuses instead on the dynamic maintenance of *existing* services through the solution of their dependencies. Different approaches to solving this problem can already be found in the literature. Some of them explicitly address the service assembly problem (e.g., Sykes et al. (2011) and Angelo et al. (2020)). Other solutions can be obtained by suitably adapting solutions primarily addressing *Service Composition* $C2$ cited above. Indeed, service assembly can be considered a special case of service composition $C2$, with a "null" workflow, under the hypothesis that the dependencies of a service are known at deployment time[6]. Hence, most of the approaches concerning the $C2$ service composition can also be adopted to tackle the service assembly problem, and they are thus directly related to the work presented in this paper. Focusing in particular on energy-aware solutions to this problem, they have been considered in several papers (Sun et al., 2019; Hamzei et al., 2023; Li and Zhu, 2023; Baker et al., 2017; Xiang et al., 2014; Tong et al., 2020; Zeng et al., 2020; Wang et al., 2018). Indeed, Table 7 reports papers that address $C2$ service composition by explicitly considering energy awareness (RQ1).

In these papers, energy concerns are typically addressed in terms of building a service composition with minimal energy consumption, given an assumed energy consumption model. In our approach, we have addressed a similar issue, but in the context of intermittently available green energy sources (RQ1). In addition, we have also addressed another issue (RQ2), which appears to have been overlooked by existing approaches. It concerns getting insights about the trade-off between building a service assembly that meets a given energy-aware goal, and the (energetic) cost that must be paid to keep updated the information used to drive the system towards the achievement of that goal.

Concerning the approaches adopted in these papers, many of them are based on the definition of single (Tong et al., 2020; Zeng et al., 2020) or multi-objective optimization problems (Baker et al., 2017; Xiang et al., 2014; Wang et al., 2018; Sun et al., 2018), which are then solved through some heuristic, given their (usually) NP-hard complexity. Other approaches have been proposed based on Bayesian networks (e.g., Kazem et al. (2015)), temporal dependencies (e.g., Guidara et al. (2016)), and machine learning methodologies (e.g., Wang et al. (2020), Ren et al. (2020) and Ekie et al. (2021)).

A common trait of most of the reviewed literature addressing questions related to RQ1 is that the proposed solutions are based on central-

[4] https://www.qoitech.com/

[5] https://prometheus.io/

[6] By null workflow, we mean a workflow consisting only of the indication of a set of required services, without the indication of any structure (e.g., precedence relationship) among them.

Table 7
Related work classification with respect to the addressed RQs.

	Centralized										Decentralized									
	Cardellini et al. (2012)	Hamzei et al. (2023)	Li and Zhu (2023)	Baker et al. (2017)	Xiang et al. (2014)	Tong et al. (2020)	Zeng et al. (2020)	Sun et al. (2019)	Wang et al. (2018)	Sun et al. (2018)	Wang et al. (2020)	Ren et al. (2020)	Sykes et al. (2011)	Angelo et al. (2020)	Caporuscio et al. (2020)	
RQ0	✓															
RQ1		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	
RQ2			✓	✓	✓	✓	✓	✓	✓	✓	✓	✓			✓	
																previous

ized approaches that, besides relying on the implicit assumption of the feasibility of maintaining consistent global knowledge, may suffer from the typical scalability and single point of failure problems. A centralized solution might be a viable approach to service composition when the underlying assumption is to have a set of known services that operate in a closed environment. This assumption does not match the open and highly dynamic characteristics of the IoT-edge computing scenario we are focusing on. Hence, we relax this assumption and, in addition to specifically focusing on the service assembly problem, we propose a fully decentralized approach that appears to be best suited to an open and dynamic scenario.

In this regard, decentralized approaches that specifically address the problem of service assembly have already been proposed (Sykes et al., 2011; Angelo et al., 2020; Caporuscio et al., 2020). Similarly to our work, the authors in Sykes et al. (2011) propose a decentralized approach to the autonomous assembly of services based on a gossip protocol. They prove that the use of gossiping techniques for realizing an autonomous assembly of services is a viable solution for scale and information dissemination convergence. However, unlike us, they focus on the fulfillment of functional requirements (dependency resolution) and the enforcement of architectural constraints (e.g., specific connection topologies). As already remarked in the introduction, our work is based on the decentralized solution to service assembly proposed in Angelo et al. (2020), which was focused on satisfying general QoS-aware requirements. We build on it to address the sustainability concerns expressed by RQ1 and RQ2. As detailed in the Introduction, we already partially addressed RQ1 in Caporuscio et al. (2020), where we present an early approach for energy-aware service assembly.

Table 7 summarizes reviewed papers addressing the service assembly problem explicitly or implicitly as part of the $C2$ service composition problem (RQ0). The table highlights which of these papers address RQ1 and RQ2. In the table, "*previous*" refers to the works we build on and extend.

Energy-saving solutions for software applications. The general increasing concern about sustainability issues motivates the growing attention to reducing the carbon footprint of software applications (Fonseca et al., 2019; Nardi et al., 2018).

The proposed approaches address this issue at different levels. At the hardware level, edge-fog-specific energy management techniques have been proposed based on the idea of suitably modulating the energy consumption of each node through DVFS (Dynamic Voltage and Frequency Scaling) and DMS (Dynamic Modulation Scaling) mechanisms (Karimi-afshar et al., 2020; Kwak et al., 2015; Toor et al., 2019). Our proposal complements these approaches, as we instead adopt a higher-level software architecture-oriented perspective.

At the software level, the software engineering community at large has been paying increasing attention to energy efficiency solutions, a summary can be found in Horcas et al. (2019) and Kanso and Exposito (2023). In addition, from the point of view of software architecture, the need to consider the energy attribute at the architectural level is gaining consensus (Vos et al., 2022).

At the infrastructure level, approaches to energy management for applications running in cloud infrastructures have already been proposed, based on techniques such as virtual machine consolidation and on/off switching of servers (Beloglazov et al., n.d.; Lee and Zomaya, 2012; Ni and Bai, 2017). However, they rely on the characteristics of large cloud data centers, where servers are often homogeneous, close to each other, and connected through high-speed networks. These techniques are hardly applicable to the scenario we are focusing on, where nodes are usually heterogeneous, highly distributed, and with limited power and computing/communication capabilities. On the other hand, the distribution of edge nodes lends itself to the local harvesting of renewable (green) energy from on-site sources, for example, solar and wind (Zeng et al., 2020; Jalali et al., 2017), as we assume to do in our approach.

10. Conclusion and future work

This paper focuses on the challenges of supporting the construction of new service-based systems in decentralized computing scenarios by considering environmental sustainability. These systems are inherently open, highly dynamic, and operate in uncertain environments. In this context, we have evidenced the role of the *service assembly* issue. To address the related challenges, we have presented a solution that builds upon a previously proposed fully decentralized service assembly procedure. In particular, we have defined a suitable energy model for the scenario we are considering and have specialized the service selection policy underpinning that procedure to achieve energetic autonomy, relying solely on locally harvested and stored energy. To this end, we have proposed a general template for defining a service selection policy, from which different policies can be easily derived to drive the construction and maintenance of the service assembly. Through extensive simulation experiments, we have gained valuable insights into the assembly management policies that promise to be most effective in meeting the sustainability requirements.

From these results, we plan to extend our work in several directions. They include (1) improving the model for incorporating advanced energy-aware heuristics (e.g., turning off monitoring when not needed), (2) extending the evaluation of the approach to include trade-off analysis between QoS and energy consumption, and (3) applying the proposed approach to real-world case studies sourced from the existing literature and industrial partners.

CRediT authorship contribution statement

Mauro Caporuscio: Writing – review & editing, Writing – original draft, Methodology, Investigation, Conceptualization; **Mirko D'Angelo:** Writing – review & editing, Writing – original draft, Software, Methodology, Investigation, Data curation, Conceptualization; **Vincenzo Grassi:** Writing – review & editing, Writing – original draft, Methodology, Investigation, Conceptualization; **Raffaela Mirandola:** Writing – review & editing, Writing – original draft, Methodology, Investigation, Conceptualization; **Francesca Ricci:** Writing – review & editing, Writing – original draft, Software, Methodology, Investigation, Data curation, Conceptualization.

Acknowledgements

This work was partially supported by the Helmholtz Association with the KiKIT project and the Grant 46.23 (Engineering Secure Systems) and by the German Research Foundation (DFG) - SFB 1608 - 501798263.

Data availability

Link above

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Appendix A. Solar source model

The model of the energy production follows the values of the solar radiation on the ground recorded by the "Bioparco" weather station in Rome (Italy) as illustrated in Fig. A.1 (Stazione meteo di bioparco di roma, 2023).

We assume that each node is equipped with a solar panel of 5 × 5cm. Using panels with polycrystalline solar cells, the maximum energy that can be supplied is about 0.016 Wh/cm2 [41]. Consequently, the maximum green energy production $G_N(t)$ of each node is about 0.4 Wh.

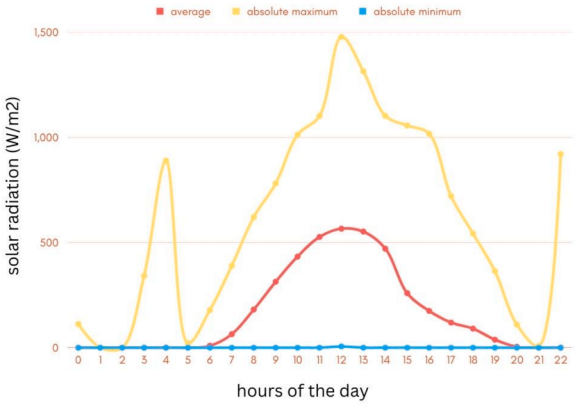


Fig. A.1. Solar radiation measured at Bioparco di Roma.

We use a six state model to simulate the solar radiation variations (see Fig. A.2 and Table A.1), where each state represents a four-hour time interval.

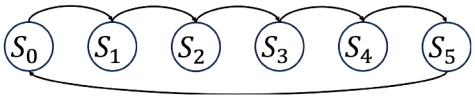


Fig. A.2. Solar states state-model representation.

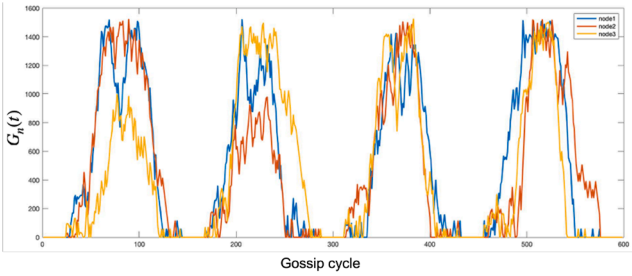


Fig. A.3. Energy production of three nodes over a period of four days.

Transitions from one state to another occur deterministically: the simulator spends exactly 24 simulation cycles in one state and then moves on to the next (as explained in the paper, one cycle models ten minutes of wall-clock time).

Table A.1
Solar states and time of the day.

Solar state	*Time interval (hh:mm)*
S_0	0:00 - 4:00
S_1	4:00 - 8:00
S_2	8:00 - 12:00
S_3	12:00 - 16:00
S_4	16:00 - 20:00
S_5	20:00 - 24:00

In order to create a realistic trend for energy production, we use a random walk to update the green energy production rate $G_N(t)$:

$$G_N(t) = \begin{cases} G_N(t-1) + m_s + e & \text{with probability } p_s \\ G_N(t-1) - m_s + e & \text{with probability } p_s - 1 \end{cases}$$

where s indicates the current solar model state, p_S is a state-dependent probability value, m_s a state-dependent constant, and e_s is a disturbance factor that is randomly calculated. Table A.2 shows the parameters used for the solar state model.

Table A.2
State-dependent parameters of the solar model.

	s_1	s_2	s_3	s_4	s_5
p_s	0.4	0.8	0.6	0.2	0.2
m_s	0.1	0.15	0.15	0.15	0.1

We introduced the disturbance factor to simulate a node-to-node variation in energy production. Indeed, even if the nodes are located in a limited area, the energy production can be slightly different (e.g., presence of clouds or shadows).

To model the complete absence of light in the night, the value of $G_N(t)$ is equal to zero when the system is in the state s_0.

As an example of the energy production trend, Fig. A.3 shows a graph where the green energy production rates $G_N(t)$ of three nodes of the network are compared for a time interval of approximately four days (i.e., around 600 simulation cycles).

Appendix B. Energy Consumption model

For each node, the energy consumption is given by the sum of three components: (*i*) energy consumption related to CPU operations, (*ii*) energy consumption for receiving messages and, (*iii*) the energy consumption for sending messages. Each of these modeled contributions is computed at simulation time.

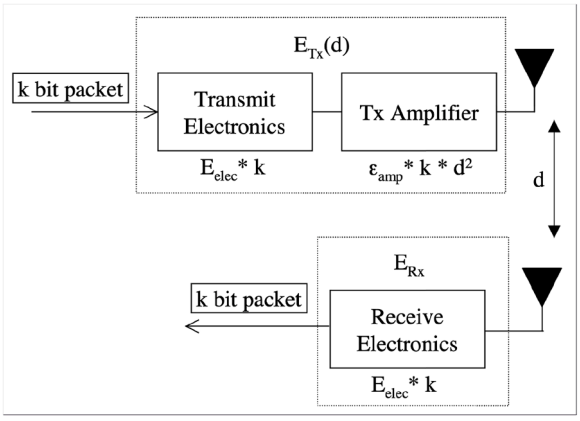


Fig. B.1. First order radio model (Heinzelman et al., 2000).

For the energy spent in transmission and reception, we use the first-order radio model, see Fig. B.1. The equations modeling transmission and energy consumption are the following:

$$E_{Tx}(k,d) = \lambda \cdot k \cdot (E_{elect} + d^2 \cdot \epsilon_{amp}) \quad \text{(B.1)}$$

$$E_{Rx}(k) = \lambda \cdot k \cdot E_{elect} \quad \text{(B.2)}$$

where:

- λ represents the flow of incoming requests to the service;
- k is the number of bits in a message;
- $E_{Tx}(k,d)$ represents the energy spent for the transmission of a k-bit message between two services that are hosted on nodes distant d;
- $E_{Rx}(k)$ represents the energy spent for receiving a k-bit message;
- ϵ_{amp} is a constant representing the energy spent by the amplification circuit for sending messages.

The parameters configuring the energy consumption model have been retrieved from existing literature (Heinzelman et al., 2002; Ikpehai et al., 2019; Kang et al., 2006):

- $E_{elect} = 50nJ/bit$
- $\epsilon_{amp} = 10pJ/bit/m^2$
- $k = 1264bit = 158bytes$

With respect to the computational energy model, the single-operation CPU energy consumption has been set to $50nJ$ (Grochowski and Annavaram, 2006).

On average, the big battery model in the simulations corresponds to a battery of $6000mAh$, while the small battery model to a battery of $3000mAh$.

References

Angelo, M.D., Caporuscio, M., Grassi, V., Mirandola, R., 2020. Decentralized learning for self-adaptive qos-aware service assembly. Future Gene. Comput. Syst. 108, 210–227.

Baker, T., Asim, M., Tawfik, H., Aldawsari, B., Buyya, R., 2017. An energy-aware service composition algorithm for multiple cloud-based iot applications. J. Netw. Comput. Appl. 89, 96-108. Emerging Services for Internet of Things.

Bandara, T.M., Mudiyanselage, W., Raza, M.S., 2020. Smart farm and monitoring system for measuring the environmental condition using wireless sensor network - iot technology in farming. In: 5th International Conference on Innovative Technologies in Intelligent Systems and Industrial Applications. CITISIA, pp. 1–7.

Banks, J., Carson, J., Nelson, B.L., Nicol, D., 2013. Discrete-Event System Simulation. Prentice Hall. 5th edition. 5th edition edition.

Beloglazov, A., Abawajy, J., Buyya, R., . Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing. Future Gen. Comput. Syst. 28 (5), 755–768.

Bouguettaya, A., et al., 2017. A service computing manifesto: the next 10 years. Commun. ACM 60 (4), 64–72.

Bouguettaya, A., et al., 2021. An internet of things service roadmap. Commun. ACM 64 (9), 86–95.

Caporuscio, M., Angelo, M., Grassi, V., Mirandola, R., 2020. Decentralized architecture for energy-aware service assembly. In: Jansen, A., Malavolta, I., Muccini, H., Ozkaya, I., Zimmermann, O. (Eds.), Software Architecture. Springer International.

Cardellini, V., Casalicchio, E., Grassi, V., Iannucci, S., Presti, F.L., Mirandola, R., 2012. MOSES: a framework for qos driven runtime adaptation of service-oriented systems. IEEE Trans. Software Eng. 38 (5), 1138–1159.

Casamayor-Pujol, V., Donta, P.K., Morichetta, A., Murturi, I., Dustdar, S., 2023. Edge intelligence - research opportunities for distributed computing continuum systems. IEEE Internet Comput. 27 (4), 49–62.

Chen, N., Cardozo, N., Clarke, S., 2018. Goal-driven service composition in mobile and pervasive computing. IEEE Trans. Serv. Comput. 11 (1), 49–62.

Ekie, Y.J., Gueye, B., Niang, I., Ekie, A.M.T., 2021. Web based composition using machine learning approaches: a literature review. In: Proceedings of the 4th International Conference on Networking, Information Systems & Security, NISS '21.

Erl, T., 2005. Service-Oriented Architecture: Concepts, Technology, and Design. Prentice Hall PTR, Upper Saddle River, NJ, USA.

Fonseca, A., Kazman, R., Lago, P., 2019. A manifesto for energy-aware software. IEEE Software 36 (6), 79–82.

Freitag, C., Berners-Lee, M., Widdicks, K., Knowles, B., Blair, G.S., Friday, A., 2021. The real climate and transformative impact of ict: a critique of estimates, trends, and regulations. Patterns 2 (9).

Gabrel, V., Manouvrier, M., Moreau, K., Murat, C., 2018. Qos-aware automatic syntactic service composition problem: complexity and resolution. Future Gen. Comput. Syst. 80, 311–321.

Grochowski, E., Annavaram, M., 2006. Energy per instruction trends in intel microprocessors. Technol. Intel Mag. 4 (3).

Guidara, I., Al Jaouhari, I., Guermouche, N., 2016. Dynamic selection for service composition based on temporal and qos constraints. In: 2016 IEEE International Conference on Services Computing (SCC), pp. 267–274.

Halba, K., Griffor, E., Lbath, A., Dahbura, A., 2023. Iot capabilities composition and decomposition: a systematic review. IEEE Access 11, 29959–30007.

Hamzei, M., Khandagh, S., Navimipour, N.J., 2023. A quality-of-service-aware service composition method in the internet of things using a multi-objective fuzzy-based hybrid algorithm. Sensors 23 (16), 7233.

Han, Q., Liu, P., Zhang, H., Cai, Z., 2019. A wireless sensor network for monitoring environmental quality in the manufacturing industry. IEEE Access 7, 78108–78119.

Heinzelman, W., Chandrakasan, A., Balakrishnan, H., 2002. An application-specific protocol architecture for wireless microsensor networks. IEEE Trans. Wireless Commun. 1 (4), 660–670.

Heinzelman, W.R., Chandrakasan, A., Balakrishnan, H., 2000. Energy-efficient communication protocol for wireless microsensor networks. In: Proceedings of the 33rd Annual Hawaii International Conference on System Sciences.

Horcas, J.M., Pinto, M., Fuentes, L., 2019. Context-aware energy-efficient applications for cyber-physical systems. Ad Hoc Netw. 82, 15–30.

Hyndman, R., Athanasopoulos, G., 2021. Forecasting: Principles and Practice. Vol. 3. rd edition edition.

Ikpehai, A., Adebisi, B., Rabie, K.M., Anoh, K., Ande, RE., Hammoudeh, M., Gacanin, H., Mbanaso, U.M., 2019. Low-power wide area network technologies for internet-of-things: a comparative review. IEEE Internet Things J. 6 (2), 2225–2240.

Ishii, S., Yoshida, W., Yoshimoto, J., 2002. Control of exploitation-exploration meta-parameter in reinforcement learning. Neural Netw. 15 (4), 665–687.

Jalali, F., Khodadustan, S., Gray, C., Hinton, K., Suits, F., 2017. Greening iot with fog: a survey. In: International Conference on Edge Computing.

Kang, K.D., Liu, K., Abu-Ghazaleh, N., 2006. Securing Geographic Routing in Wireless Sensor Networks, In: 9th Annual NYS Cyber Security Conference: Symposium on Information Assurance.

Kanso, A.N.H., Exposito, E., 2023. A review of energy aware cyber-physical systems.

Karimiafshar, A., Hashemi, M.R., Heidarpour, M.R., Toosi, A.N., 2020. Effective utilization of renewable energy sources in fog computing environment via frequency and modulation level scaling. IEEE Internet Things J. 7 (11), 10912– 10921.
Kazem, A.A.P., Pedram, H., Abolhassani, H., 2015. Bnqm: a bayesian network based qos model for grid service composition. Expert Syst. Appl. 42 (20), 6828–6843.
Khanouche, M.E., Amirat, Y., Chibani, A., Kerkar, M., Yachir, A., 2016. Energy-centered and qos-aware services selection for internet of things. IEEE Trans. Autom. Sci. Eng. 13 (3), 1256–1269.
Kwak, J., Kim, Y., Lee, J., Chong, S., 2015. Dream: dynamic resource and task allocation for energy minimization in mobile cloud systems. IEEE J. Sel. Areas Commun. 33 (12), 2510–2523.
Lee, Y.C., Zomaya, A.Y., 2012. Energy efficient utilization of resources in cloud computing systems. J. Supercomput. 60 (2), 268–280.
Lemos, A.L., Daniel, F., Benatallah, B., 2015. Web service composition: a survey of techniques and tools. ACM Comput. Surv. 48 (3), , 1–41.
Li, J., Zhu, S., 2023. Service composition considering energy consumption of users and transferring files in a multicloud environment. J. Cloud Comput. 12 (1), 43.
Meshkova, E., Riihijarvi, J., Petrova, M., Mahonen, P., 2008. A survey on resource discovery mechanisms, peer-to-peer and service discovery frameworks. Comput. Netw. 52 (11), 2097–2128.
Montresor, A., Jelasity, M., 2009. Peersim: a scalable p2p simulator. In: IEEE Ninth International Conference on Peer-to-Peer Computing, pp. 99–100.
Nardi, B., Tomlinson, B., Patterson, D.J., Chen, J., Pargman, D., Raghavan, B., Penzenstadler, B., 2018. Computing within limits, communications of the. ACM 61 (10), 86–93.
Nasir, N., Sobral, V.A.L., Huang, L.P., Campbell, B., 2022. Nexusedge: leveraging iot gateways for a decentralized edge computing platform. In: 7th Symposium on Edge Computing.
Nast, M., Raddatz, H., Rother, B., Golatowski, F., Timmermann, D., 2023. A survey and comparison of publish/subscribe protocols for the industrial internet of things. In: Proceedings of the 12th International Conference on the Internet of Things.
Ni, J., Bai, X., 2017. A review of air conditioning energy performance in data centers. Renewable Sustainable Energy Rev. 67, 625–640.
Paolucci, M., Kawamura, T., Payne, T.R., Sycara, K., 2002. Semantic matching of web services capabilities. Berlin Heidelberg, Berlin, Heidelberg, Springer. The Semantic Web - ISWC 2002.
Rao, J., Su, X., 2004. A survey of automated web service composition methods. In: International Workshop on Semantic Web Services and Web Process Composition. Springer, pp. 43–54.
Razian, M., Fathian, M., Bahsoon, R., Toosi, A.N., Buyya, R., 2022. Service composition in dynamic environments: a systematic review and future directions. J. Syst. Softw. 188, 111290.
Ren, L., Wang, W., Xu, H., 2020. A reinforcement learning method for constraint-satisfied services composition. IEEE Trans. Serv. Comput. 13 (5), 786–800.
Rodriguez-Mier, P., Pedrinaci, C., Lama, M., Mucientes, M., 2016. An integrated semantic web service discovery and composition framework. IEEE Trans. Serv. Comput. 9 (4), 537–550.
Stazione meteo di bioparco di roma , 2023. https://stazioni2.soluzionimeteo.it/bioparcoroma/mobile/pages/station/day.php.
Schaerf, A., Shoham, Y., Tennenholtz, M., 1995. Adaptive load balancing: a study in multi-agent learning. J. Artif. Int. Res. 2 (1), 475–500.
Sun, M., Zhou, Z., Duan, Y., 2018. Energy-aware service composition of configurable iot smart things. In: 14th International Conference on Mobile Ad-Hoc and Sensor Networks. IEEE.
Sun, M., Zhou, Z., Wang, J., Du, C., Gaaloul, W., 2019. Energy-efficient iot service composition for concurrent timed applications. Future Gen. Comput. Syst. 100, 1017–1030.
Sykes, D., Magee, J., Kramer, J., 2011. Flashmob: distributed adaptive self-assembly. In: Giese, H., Cheng, B.H.C. (Eds.), ICSE Symposium on Software Engineering for Adaptive and Self-Managing Systems, ACM.
Tong, E., Chen, L., Li, H., 2020. Energy-aware service selection and adaptation in wireless sensor networks with qos guarantee. IEEE Trans. Serv. Comput. 13 (5), 829–842.
Toor, A., Islam, S., Sohail, N., Akhunzada, A., Boudjadar, J., Khattak, H.A., Din, I.U., Rodrigues, J.J., 2019. Energy and performance aware fog computing: a case of dvfs and green renewable energy. Future Gen. Comput. Syst. 101, 1112–1121.
Vos, S., Lago, P., Verdecchia, R., Heitlager, I., 2022. Architectural tactics to optimize software for energy efficiency in the public cloud. In: 2022 International Conference on ICT for Sustainability.
Wang, H., Li, J., Yu, Q., Hong, T., Yan, J., Zhao, W., 2020. Integrating recurrent neural networks and reinforcement learning for dynamic service composition. Future Gen. Comput. Syst. 107, 551–563.
Wang, S., Zhou, A., Bao, R., Chou, W., Yau, S.S., 2018. Towards green service composition approach in the cloud.
Weyns, D., et al., 2013. On patterns for decentralized control in self-adaptive systems. LNCS 7475, 76–107.
Wohlin, C., Runeson, P., Hst, M., Ohlsson, M.C., Regnell, B., Wessln, A., 2012. Experimentation in Software Engineering. Springer Publishing Company.
Xiang, F., Hu, Y., Yu, Y., Wu, H., 2014. Qos and energy consumption aware service composition and optimal-selection based on pareto group leader algorithm in cloud manufacturing system. Central Eur. J. Oper. Res. 22 (4), , 663–685.
Zeng, D., Gu, L., Yao, H., 2020. Towards energy efficient service composition in green energy powered cyber-physical fog systems. Future Gen. Comput. Syst. 105, 757–765.