Decentralized Architecture for Energy-Aware Service Assembly

Mauro Caporuscio[1(✉)], Mirko D'Angelo[1], Vincenzo Grassi[2], and Raffaela Mirandola[3]

[1] Linnaeus University, Växjö, Sweden
{mauro.caporuscio,mirko.dangelo}@lnu.se
[2] Università di Roma Tor Vergata, Rome, Italy
vincenzo.grassi@uniroma2.it
[3] Politecnico di Milano, Milan, Italy
raffaela.mirandola@polimi.it

Abstract. Contemporary application domains make more and more appealing the vision of applications built as a dynamic and opportunistic assembly of autonomous and independent resources. However, the adoption of such paradigm is challenged by: (*i*) the openness and scalability needs of the operating environment, which rule out approaches based on centralized architectures and, (*ii*) the increasing concern for sustainability issues, which makes particularly relevant, in addition to QoS constraints, the goal of reducing the application energy footprint. In this context, we contribute by proposing a decentralized architecture to build a fully functional assembly of distributed services, able to optimize its energy consumption, paying also attention to issues concerning the delivered quality of service. We suggest suitable indexes to measure from different perspectives the energy efficiency of the resulting assembly, and present the results of extensive simulation experiments to assess the effectiveness of our approach.

1 Introduction and Motivation

Contemporary systems (for domains including, for example, smart cities, intelligent transportation systems, augmented reality) more and more envision the definition of applications that dynamically emerge as an opportunistic aggregation of autonomous and independent resources available within the execution environment. Service-oriented architecture (SOA), in particular its microservice evolution, appears well suited as reference architectural model for this kind of applications, as it supports the vision of new services built as an assembly of independent services, where each service offers specific functionalities, and could require functionalities offered by others to carry out its own task.

However, to be successfully adopted in these emerging computing environments, the service assembly procedure should be able to tackle the following main issues: (*i*) *decentralization*: services are offered by autonomous and independent

A. Jansen et al. (Eds.): ECSA 2020, LNCS 12292, pp. 57–72, 2020.
https://doi.org/10.1007/978-3-030-58923-3_4

resources distributed in the environment, which makes hardly usable assembly procedures based on the presence of some centralized assembly manager; (*ii*) *dynamics*: the offered services are not statically defined but they appear and disappear or change their behavior; (*iii*) *quality-awareness*: the assembly should be able to guarantee quality of service (QoS) requirements (e.g., timeliness, availability, cost). Besides them, another major issue to be considered is: (*iv*) *energy-awareness*: the assembly should be able to take into account the energy consumption caused by its computation and communication activities. This latter issue is particularly important for several reasons: besides sustainability concerns that are more and more important in the contemporary world, systems often rely on battery-powered resources, where a parsimonious and effective use of available energy is mandatory to extend the system lifetime.

Considerable effort has been already devoted in the past to QoS-aware services assembly procedures [14]. On the other hand, energy efficiency of composite services has received less attention until recently, when the growing interest on sustainability themes has put this issue in the foreground [17,19]. However, to the best of our knowledge, no available solution exists yet that is able to deal with all the issues (*i*)-(*iv*) mentioned above.

Paper Contribution. In this respect, we propose an initial solution of the following problem: *how to devise a decentralized architecture that supports the dynamic building of a fully resolved assembly of distributed services, collectively fulfilling functional requirements while minimizing the energy consumption, in an open and variable execution environment.* On answering to this question we also take into account the impact on the system's QoS.

Specifically, we propose a (fully) decentralized and dynamic service assembly framework whose main characteristics are: (*i*) a system architecture for service assembly management; (*ii*) explicit modelling of service energy consumption for both processing and communication activities; (*iii*) an energy-aware service selection and composition procedure; (*iv*) a set of "social welfare" indexes aimed at measuring the system effectiveness with respect to QoS and energy objectives.

Related Work. Our work lies within the general area of service selection and composition problem in a distributed environment. There is quite a large amount of literature on the topic (e.g., [2,4,14] and references therein); hereafter, we briefly review works closer to ours having a focus on energy-aware solutions. Recently, the software engineering community at large is paying increasing attention to energy efficiency solutions, a summary can be found in [6]. Also from a software architecture point of view, the need to consider the energy attribute at architectural level is gaining consensus [18]. However, to the best of our knowledge, only a limited amount of effort has been devoted up to now to the definition of architectural approaches for energy-aware decentralized service assemblies. Examples of existing solutions can be found in [15–17,19], where service assemblies are considered for cloud-based applications [17], in wireless-sensor-networks context/domain [16] and for cyber-physical systems [15,19]. However, the proposed solutions are based on the definition of single [16,19] or multi-objective optimization problems [15,17], which leverage on centralized approaches.

Structure of the Paper. Section 2 presents the system model we refer to Sect. 3 introduces the service assembly energy consumption model, and Sect. 4 the indexes derived from this model to measure the assembly energy and QoS effectiveness. Section 5 illustrates the decentralized architecture for the assembly construction and maintenance. Section 6 shows the results of our experiments, Sect. 7 discusses threats to validity, while Sect. 8 provides conclusions and hints for future work.

2 System Model

We consider a distributed system consisting of a set $\mathbf{N}$ of nodes (e.g., nodes of an edge cloud architecture), and a set $\mathbf{S}$ of (sw-implemented) services that must be deployed on these nodes. Each node provides basic computing and communication services, used by the other services hosted by the same node for their operations. For the sake of simplicity, we assume that each node offers only a single type of computing and communication service, and do not consider other basic service categories (e.g., storage). We leave to future work the extension of our model to these other categories, and to different types of services within each category (e.g., both specialized GPU and general purpose CPU as computing services). Each service $S \in \mathbf{S}$ must be bound to a computing and communication service. Besides this, S could require functionalities offered by other services in the set $\mathbf{S}$ to carry out its own task. We denote by $\mathbf{B}$ the set of all computing and communication services offered by nodes in $\mathbf{N}$, and by $node(S) \in \mathbf{N}$ the node hosting service $S \in \mathbf{S} \cup \mathbf{B}$. We assume that services in the set $\mathbf{B}$ are the only direct sources of energy consumption, while the energy consumption of services in $\mathbf{S}$ is related with the use they directly or indirectly make of services in $\mathbf{B}$.

For the purpose of the service assembly procedure we intend to devise, we now introduce a more detailed service model. A service $S \in \mathbf{S}$ is represented as a tuple $\langle Type, Deps, Prov, Req \rangle$, where:

- $S.Type \in \mathbf{T}$ denotes the type of the provided interface (we say that $S.Type$ is the type of S). We assume the existence of a function $match : \mathbf{T} \times \mathbf{T} \rightarrow [0, 1]$ such that $match(T_1, T_2) = 0$ if type T_1 does not match type T_2 and $match(T_1, T_2) > 0$ if a matching exists according to some suitable matching criterion [1,9].
- $S.Deps \subseteq \mathbf{T} \bigcup \{comp, comm\}$ is the set of required dependencies for S. We assume that $S.Deps$ is fixed for each service and known in advance. Note that the dependency set $S.Deps$ does not contain duplicates, meaning that a service may depend at most once on any specific interface type. We assume that $S.Deps$ always includes two dependencies $d_1 = comp$ and $d_2 = comm$: in this way we model the fact that S needs at least to be bound to a computing and a communication service, for its internal operations and for its interactions with other services. For each $d \in S.Deps$ we assume that it is known (e.g., through a locally performed monitoring activity) a value $\mu_{S,d}$, which represents the average number of times service S requires dependency d to fulfill each request it has received.

- $S.Prov \subseteq \mathbf{S}$ is the set of *Providers* of S, i.e., the set of services to which S is bound to resolve its dependencies. We denote by $comp(S) \in \mathbf{B}$ and $comm(S) \in \mathbf{B}$ the computing and communication services used to resolve dependencies $comp \in S.Deps$ and $comm \in S.Deps$, respectively. It must obviously hold $node(comp(S)) = node(comm(S))$.
- $S.Req \subseteq \mathbf{S}$ is the set of *Requesters* of S, i.e., is the set of other services that are bound to S to resolve one of their dependencies.

A service is either *fully resolved* or *partially resolved*. Basic services in the set $\mathbf{B}$ are fully resolved by definition. A service $S \in \mathbf{S}$ is *fully resolved* if for all $d \in S.Deps$ there exists a fully resolved service $S' \in S.Prov$ such that $match(d, S'.Type) > 0$. On the other hand, a *partially resolved* service $S \in \mathbf{S}$ has at least one dependency that is either not matched, or is matched by a partially resolved service.

Finally, a *service assembly* $\mathbf{A}$ is a directed graph $\mathbf{A} = (\mathbf{S}, \mathbf{E})$, where $\mathbf{E} \subseteq \mathbf{S} \times \mathbf{S}$ is the set of resolved dependencies. Specifically, a directed edge $(S_i, S_j) \in \mathbf{E}$ denotes that S_i is using S_j to resolve one of its dependencies. In general, a given S_i has multiple simultaneous outgoing bindings (towards $S_i.Prov$), one for each dependency, and can have multiple simultaneous incoming bindings from other services (belonging to $S_i.Req$), using S_i to resolve one of their dependencies.

Figure 1 shows an example of a simple service assembly (including services S_1, S_2, S_3 and S_4) that illustrates the actual deployment of services on nodes N_1, N_2 and N_3. The figure highlights also the service dependencies and their binding to computation and communication services.

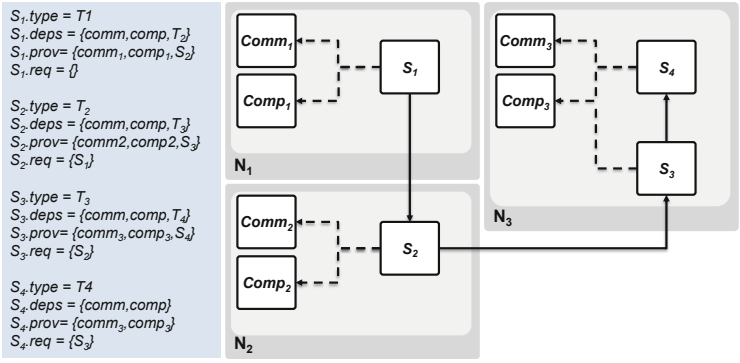


Fig. 1. Service assembly example

3 Energy Model

In this section we introduce the model we adopt to estimate the energy consumption of each service $S \in \mathbf{S}$, as a function of the bindings it establishes

with other services to resolve its dependencies. As we are considering computing and communication services as the only "physical" resources causing energy consumption, the model consists of two parts: a *computing energy* model and a *communication energy* model.

3.1 Computation Energy

Let us consider a service $S \in \mathbf{S}$. When the flow of requests addressed to services in $S.Prov$ eventually reach a service of type *comp*, it will cause some computation energy consumption. This will happen in one step for the dependency of type *comp* of S ("internal operations" of S). Otherwise, the flow of requests will go through a number of virtual services before reaching a service of type *comp*. To model this process, we introduce the following three indexes $S.I^{comp}$, $S.L^{comp}$ and $S.E^{comp}$ that model, respectively, the *individual*, *node level* and *system level* computation energy consumption caused by a single request addressed to S. They are defined as follows:

$$S.I^{comp} = h_{node(S)}(\mu_{S,comp}) \tag{1}$$

$$S.L^{comp} = S.I^{comp} + \sum_{\substack{S' \in S.Prov, \\ s.t.S'.Type \neq comp \\ \wedge node(S') = node(S)}} \mu_{S,S'.Type} \cdot S'.L^{comp} \tag{2}$$

$$S.E^{comp} = S.I^{comp} + \sum_{\substack{S' \in S.Prov, \\ s.t.S'.Type \neq comp}} \mu_{S,S'.Type} \cdot S'.E^{comp} \tag{3}$$

where $h_n(\mu)$ represents the energy consumption of the *comp* service hosted by a node n for the execution of μ operations[1]. As an example, $h_n(\mu)$ could be instantiated as $h_n(\mu) = a_n + e_n \cdot \mu$, with a *fixed* part a_n (energy consumed when the *comp* service is switched on, independently of its operations), and a *dynamic* part $e_n \cdot \mu$ linearly depending on the load addressed to the *comp* service (e_n represents the energy consumption for a single operation).

From the definitions given above, we see that $S.I^{comp}$ models only the energy consumption directly consumed by S for its internal operations. Besides the directly consumed energy, $S.L^{comp}$ includes also the computation energy indirectly consumed by S because of its use of services $S' \in S.Prov$, but limited to services co-located with S on the same node (their energy consumption is multiplied by the average number of times S uses S', given by $\mu_{S,S'.Type}$). Finally, $S.E^{comp}$ adopts a system-wide perspective and models the overall computation energy consumption caused by S on any node in the system.

We point out that $S.I^{comp}$, $S.L^{comp}$ and $S.E^{comp}$ refer to computing energy consumption caused by a single request addressed to S. To get measures of the

[1] By "operation" we mean a conventional average unit of computation. We make an analogous assumption for the communication model.

energy consumption per unit time (energy consumption rate), we introduce the concept of *load vector* $\mathbf{\Lambda}_S = [\lambda_S(n)]_{n \in \mathbf{N}}$ associated with any service $S \in \mathbf{S}$, where each vector entry $\lambda_S(n)$ denotes a flow of requests (expressed as requests per unit time) addressed to service S by services hosted by node $n \in \mathbf{N}$. $\mathbf{\Lambda}_S$ can be easily estimated by some local monitoring activity. Given a load vector $\mathbf{\Lambda_S}$, we can derive from $S.I^{comp}$, $S.L^{comp}$ and $S.E^{comp}$ corresponding measures of the computing energy consumption rate:

$$S.\rho_X^{comp} = \Big(\sum_{n \in \mathbf{N}} \lambda_S(n) \Big) \cdot S.X^{comp} \tag{4}$$

where X stands for any of: I, L, E.

3.2 Communication Energy

Let us consider a service $S \in \mathbf{S}$. As already stated in the previous subsection, S represents a "virtual" (software implemented) resource: the communication energy consumption caused by S depends on the interactions that S has both with services that use it to resolve their dependencies (services in the set $S.Req$) and services used by S itself to resolve its dependencies (services in the set $S.Prov$). In this respect, we assume that the energy spent by a communication service of a node for these interactions depends both on the data volume, and the latency and bandwidth of the links connecting it with other nodes [5].

To model this process, we introduce the three indexes $S.I_n^{comm}$, $S.L_n^{comm}$ and $S.E_n^{comm}$ that model, respectively, the *individual*, *node level* and *system level* communication energy consumption caused by a single request addressed to S by some other service hosted by a node $n \in \mathbf{N}$. They are defined as follows:

$$\begin{aligned} S.I_n^{comm} &= \phi_{node(S)}^{req}(\delta_S^{rcv}, bw(n, node(S)), lt(n, node(S))) \\ &+ \sum_{\substack{S' \in S.Prov, \\ s.t. node(S) \neq node(S')}} \mu_{S,S'.Type} \cdot \phi_{node(S)}^{prov}(\delta_{S,S'.Type}^{snd}, bw(node(S), node(S')), lt(node(S), node(S'))) \end{aligned} \tag{5}$$

$$S.L_n^{comm} = S.I_n^{comm} + \sum_{\substack{S' \in S.Prov, \\ s.t. node(S') = node(S)}} \mu_{S,S'.Type} \cdot S'.L_{node(S)}^{comm} \tag{6}$$

$$S.E_n^{comm} = S.I_n^{comm} + \sum_{S' \in S.Prov} \mu_{S,S'.Type} \cdot S'.E_{node(S)}^{comm} \tag{7}$$

where:

- $bw(n_1, n_2)$ and $lt(n_1, n_2)$, with $n_1, n_2 \in \mathbf{N}$, denote, respectively, the bandwidth and latency of the link connecting nodes n_1 and n_2;
- δ_S^{rcv} and $\delta_{S,d}^{snd}$ denote, respectively, the average amount of data S receives for each service request addressed to it, and the average amount of data S sends for each invocation of its dependency d, to fulfill that request;

- $\phi_n^{req}(\delta, b, l)$ denotes the energy consumed by the *comm* service of node $n \in \mathbf{N}$ (in the following we denote it as $commserv(n)$), when it receives an amount δ[2] of data addressed to a service hosted by n over a link with bandwidth b and latency l;
- $\phi_n^{prov}(\delta, b, l)$ denote the energy consumed by $commserv(n)$, $n \in \mathbf{N}$, when it sends an amount δ of data to a service hosted by another node over a link with bandwidth b and latency l.

The first term in the r.h.s. of Eq. (5) represents the energy consumed by $commserv(node(S))$ for the reception of a service request addressed to S, coming from an external node n. The second term in the r.h.s. of Eq. (5) represents the energy consumed by $commserv(node(S))$ to send the requests S addresses to services solving its dependencies (i.e., services in $S.Prov$) and hosted by different nodes. Hence, $S.I_n^{comm}$ models the communication energy consumption of $commserv(node(S))$ caused only by the direct interactions S has with other services hosted by different nodes.

On the other hand, $S.L_n^{comm}$ in Eq. (6) adds to the energy consumption measured by $S.I_n^{comm}$ also the communication energy consumption indirectly caused by S, corresponding to interactions that services in $S.Prov$ have with other services to carry out their own task. As it can be seen from the r.h.s. of Eq. (6), $S.L_n^{comm}$ limits its scope to the energy consumption of $commserv(node(S))$ only.

Finally, $S.E_n^{comm}$ in Eq. (7) adopts a system-wide perspective, measuring the communication energy consumption directly or indirectly caused by S on any node in the system, when S receives a single request from a service hosted by a node n.

Analogously to the computing energy case, we can derive from $S.E_n^{comm}$, $S.I_n^{comm}$ and $S.L_n^{comm}$ measures of the communication energy consumption rate, given a load vector $\mathbf{\Lambda}_S = [\lambda_S(n)]$:

$$S.\rho_X^{comm} = \sum_{n \in \mathbf{N}} \lambda_S(n) \cdot S.X_n^{comm} \tag{8}$$

where X stands for any of: I, L, E.

4 Welfare Indexes

In this section we formally define the indexes, based on the model defined in the previous section, that we will use to measure the effectiveness of our approach with respect to its ability in achieving a good local and social welfare. By this we mean that our goal is to analyze our approach effectiveness from a two-fold perspective. On the one side, we measure the achievement of some average global system "quality", thanks to the contribution of all services. On the other side, we measure whether there is an unbalanced distribution among services of this global quality.

[2] δ is measured in terms of a conventional average communication unit.

Besides energy consumption, which is our main focus in this paper, we include in our notion of quality also the delivered QoS. For space reasons, we do not introduce an explicit QoS model of a service assembly. Several models of this kind have been already introduced (e.g., [4,19]). For notational purposes, we just assume that a QoS index $S.Q$ is associated with each service $S \in \mathbf{S}$, related with suitable QoS measures like response time or reliability, where the value of $S.Q$ is estimated at each node by some monitoring activity. We only point out that, for the sake of realism, we assume that the QoS delivered by a service is *load-dependent*, in the sense that it degrades with the increase of the load of requests addressed to it [10,12]. The welfare indexes we adopt are defined as follows.

Given a fully resolved assembly $\mathbf{A} = (\mathbf{S}, \mathbf{E})$, we first define the average *Global Energy Consumption* rate delivered by all services in $\mathbf{A}$:

$$GEC(\mathbf{A}) = \frac{1}{|\mathbf{N}|} \sum_{n \in \mathbf{N}} \Big(\sum_{\substack{S \in \mathbf{S} \\ s.t. node(S)=n}} (S.\rho_I^{comp} + S.\rho_I^{comm}) \Big) \tag{9}$$

We use $S.\rho_I^{comp}$ and $S.\rho_I^{comm}$, defined as in Eqs. (4) and (8), respectively, in the definition of $GEC(\mathbf{A})$ to avoid counting more than once the energy consumption caused by a service. Note that $GEC(\mathbf{A})$ is a "lower is better" index.

From the QoS perspective, we define as follows the average *Global QoS* delivered by all services in $\mathbf{A}$:

$$GQoS(\mathbf{A}) = \frac{1}{|\mathbf{S}|} \sum_{S \in \mathbf{S}} S.Q \tag{10}$$

However, both $GEC(\mathbf{A})$ and $GQoS(\mathbf{A})$ do not allow to capture to what extent all involved services and the nodes hosting them fairly contribute to the measured average quality.

To this end, we introduce the following *fairness* indexes based on the *Jain's fairness index* [7] as additional measures of the achieved social welfare, to measure how uniform is the quality achieved by all the participating services, from the energy consumption and QoS perspectives, respectively:

$$FEC(\mathbf{A}) = \frac{\Big(\sum_{n \in \mathbf{N}} \Big(\sum_{\substack{S \in \mathbf{S} \\ s.t. node(S)=n}} (S.\rho_I^{comp} + S.\rho_I^{comm}) \Big) \Big)^2}{|\mathbf{N}| \sum_{n \in \mathbf{N}} \Big(\sum_{\substack{S \in \mathbf{S} \\ s.t. node(S)=n}} (S.\rho_I^{comp} + S.\rho_I^{comm}) \Big)^2} \tag{11}$$

$$FQoS(\mathbf{A}) = \frac{(\sum_{S \in \mathbf{S}} S.Q)^2}{|\mathbf{S}| \sum_{S \in \mathbf{S}} S.Q^2} \tag{12}$$

The value of these fairness indexes ranges from $\frac{1}{|\mathbf{N}|}$ or $\frac{1}{|\mathbf{S}|}$, respectively (worst case), to 1 (best case), and it is maximum when all nodes experiment the same

energy consumption rate or all services deliver the same QoS, respectively. In general, indexes of this type penalize situations where the quality achieved by different entities is highly unbalanced. Hence, by using $FEC(\mathbf{A})$ or $FQoS(\mathbf{A})$, we intend to reward assemblies that result in a fair share of the overall quality measured by $GEC(\mathbf{A})$ and $GQoS(\mathbf{A})$, respectively.

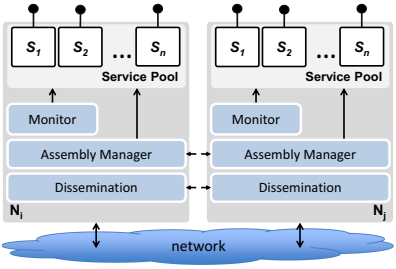


Fig. 2. Node architecture

```
1:  procedure ActiveThread
2:      loop
3:          Wait Δt
4:          for all S_j ∈ GetPeers() do
5:              Send ⟨S.Prov ∪ {S}⟩ to S_j

6:  procedure PassiveThread
7:      loop
8:          Wait for message ⟨M⟩ from S_j
9:          for all S_k ∈ M do
10:             if ∃d ∈ S.Deps — matches(d, S_k.Type) > 0 then
11:                 Best_{S,d} ← Update(Best_{S,d}, S_k)
12:                 S.Prov ← Select(Best_{S,d}, S.Prov)
```

Fig. 3. Gossip based dissemination

Regarding energy-related indexes, we note that the relative importance of the indexes $GEC(\mathbf{A})$ and $FEC(\mathbf{A})$ could depend on the scenario where they are applied. In scenarios where all nodes have access to continuous power sources, the most relevant effectiveness index could be $GEC(\mathbf{A})$, for system sustainability reasons. On the other hand, in scenarios where system nodes are battery-powered, the most relevant effectiveness index could be $FEC(\mathbf{A})$, as a highly unbalanced energy consumption among nodes could lead to the premature "death" of some node, with possible negative consequences on the whole system lifetime.

5 System Architecture

In this section we present a fully decentralized architecture that drives a service-oriented system towards the construction of an energy-efficient fully resolved service assembly. The core idea underpinning this architecture is the use of a decentralized information dissemination procedure, based on a gossiping protocol [13], through which each service S advertises its functional ($S.Type$) and extra-functional (*energy* and *QoS*) characteristics. Thanks to this procedure, services at each node become eventually aware of other services in the system that can resolve their dependencies, thus providing the basis for the fulfillment of the goal of driving the system toward the construction of a fully resolved assembly. To fulfill the goal of energy-driven assembly, the advertised information is used to select, within a set of functionally equivalent candidates, the best suited service.

Figure 2 shows the main architecture components deployed at each node N_i: *Monitor*, *Assembly Manager* and *Dissemination*. Besides them, the figure shows also *Service Pool*, the set of services $S \in \mathbf{S}$ running on node N_i.

Monitor is in charge of monitoring the energy consumption (i.e., $S.X^{comp}$ and $S.X_n^{comm}$) for each service S in the Service Pool, and notifying detected changes to the *Assembly Manager*.

Assembly Manager receives information about the type, QoS and energy consumption of local and remote services from *Monitor* and *Dissemination*, respectively, which are used to build the assembly. In particular, it receives from *Dissemination* the set $S.Prov$ that specifies which services should currently be used to solve the dependencies of a local service S, and manages the corresponding bindings. Moreover, it receives notifications of incoming binding requests for each local service S, and keeps updated the corresponding set $S.Req$.

Finally, *Dissemination* implements decentralized information dissemination by exploiting a gossip communication model [13]. This model relies on the continuous execution of two concurrent threads, *ActiveThread* and *PassiveThread* (see Algorithm in Fig. 3).

For each service S hosted by the node, *ActiveThread* periodically sends a Gossip message to its *peer set*[3]. The message payload is a set of services, with the associated type, QoS and energy information, containing the list of currently bound dependencies $S.Prov$ plus S itself.

PassiveThread listens for messages coming from other peers. Upon receiving a message containing the set $\mathbf{M}$, it checks all services $S_k \in \mathbf{M}$ to see whether some of them can be used to resolve some dependency of S. If $S_k.Type$ is required as a dependency, then S_k is considered as a candidate to be added to $Best_{S,d}$ (line 10), where $Best_{S,d}$ collects the currently known "best" services according to the specific service selection criterion used to solve the dependency d (see Sect. 5.1). The decision whether to include S_k in $Best_{S,d}$ is taken by function UPDATE() (line 11), possibly dropping from $Best_{S,d}$ some other service whose utility is worse than S_k. The update of the sets $Best_{S,d}$ can lead to a substitution of the service currently used to solve dependency d (as specified in the set $S.Prov$) with a new "better" service taken from $Best_{S,d}$. The decision about this possible substitution is taken by function SELECT() (line 12), implemented following one of the selection criteria described in Sect. 5.1.

As it is typical with gossip-based protocols, a new instance of the algorithm in Fig. 3 is created at each node for each service S in the Service Pool.

5.1 Energy-Aware Service Selection

In this section we present possible energy-aware service selection criteria that could be used in the implementation of the SELECT() function in Fig. 3.

Given a service S and a set of candidates $Best_{S,d}$, we recall that SELECT() must select, within that set, a service that resolves a dependency $d \in S.Deps$.

Energy-Aware Overall. The *Energy-aware Overall* criterion aims at selecting the service that causes the minimal energy consumption on a system-wide basis

[3] The peer set is provided by the underlying gossip communication protocol [13].

(i.e., $GEC(\mathbf{A})$). In order to use this criterion, each service S is required to disseminate the values $S.E^{comp}$ and $S.E^{comm}_n$, $n \in \mathbf{N}$, defined by Eqs. (3) and (7), respectively. This criterion can be stated as:
Select $\overline{S} \in Best_{S,d}$ such that:

$$\overline{S} = \underset{S' \in Best_{S,d}}{\arg\min} \{S'.E^{comp} + \mu_{S,S'.Type} \cdot S'.E^{comm}_{node(S)}\} \tag{13}$$

Energy-Aware Local. The *Energy-aware Local* criterion is similar to the previous one, but it acts on the basis of a more limited scope, as it focuses on the minimization of the energy consumption involving $node(S)$ only (the definition of Eq. (14) is derived from the definition of the $S.L^{comp}$ and $S.L^{comm}_n$ indexes in Eqs. (2) and (6), respectively). For this reason, differently from the *Overall* criterion, it does not require the dissemination of any energy consumption value, as all the information needed for its application can be collected at each node by a local monitoring activity. This criterion can be stated as:
Select $\overline{S} \in Best_{S,d}$ such that:

$$\begin{aligned}\overline{S} = \underset{S' \in Best_{S,d}}{\arg\min} \Big\{ & \mu_{S,S'.Type} \cdot S'.L^{comp} \cdot I_{\{node(S')=node(S)\}} \\ & + \mu_{S,S'.Type} \cdot \phi^{prov}_{node(S)}(\delta^{snd}_{S,S'.Type}, bw(node(S), node(S')), lt(node(S), node(S')) \cdot I_{\{node(S') \neq node(S)\}} \\ & + \mu_{S,S'.Type} \cdot S'.L^{comm}_{node(S)} \cdot I_{\{node(S')=node(S)\}} \Big\}\end{aligned} \tag{14}$$

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

As pointed out in Sect. 4, focusing on the minimization of $GEC(\mathbf{A})$ could not be a good choice in contexts where one should instead aim at fairly balancing energy consumption among all nodes. We thus propose a third criterion, aimed at the maximization of the fairness index $FEC(\mathbf{A})$.

Energy-Aware Learning. The *Energy-aware Learning* criterion selects the service in $Best_{S,d}$ hosted by the node that currently results to have the lowest energy consumption rate. This criterion can be stated as:
Select $\overline{S} \in Best_{S,d}$ such that:

$$node(\overline{S}) = \underset{n \in node(Best_{S,d})}{\arg\min} \{ \sum_{\substack{S \in \mathbf{S} \\ s.t. node(S)=n}} (S.\rho^{comp}_I + S.\rho^{comm}_I)\} \tag{15}$$

where, with a little abuse of notation, $node(Best_{S,d}) \subseteq \mathbf{N}$ denotes the set of all nodes hosting services that belongs to the set $Best_{S,d}$.

The actual application of this criterion deserves however more attention with respect to the former two criteria. Indeed, we can note that both Eqs. (13) and (14) used in the definition of *Energy-aware Overall* and *Energy-aware Local*, respectively, are based on load-independent indexes. They are thus well suited

for the greedy approach underlying these two criteria. On the other hand, the indexes used in Eq. (15) are load-dependent, with consequent worsening of their value when the load of node n increases. The greedy approach underlying the definition of *Energy-aware Learning* given above can thus lead to well known problems of system instability. Indeed, what currently results to be the node with the smallest energy consumption rate could rapidly become overloaded, thus triggering the need of new selections, and so on. A more judicious definition of this criterion is thus necessary, more suited for the goal of achieving a fair energy consumption balance.

To this end, we implement this criterion according to the learning method proposed in [11], originally proposed for a scenario of decentralized load balancing in a distributed system with load-dependent QoS. In this method, resource selection by the participating services is based on a suitable balance between exploitation (what services have learnt from past observations about the current "best" resource, giving proper weight to the received information based on its age) and exploration (random selection of apparently "not best" resources). In our adaptation of this method, we assume that each node advertises the current value of $S.\rho_I^{comp}$ and $S.\rho_I^{comm}$, using the gossiping procedure we have described above. This information is then managed according to the method proposed in [11], to which we refer for details, omitted here for space reasons.

6 Experimental Evaluation

In this section we present a set of simulation experiments to assess the effectiveness of different service selection strategies on the social welfare of the system. To this end, we experiment with the three energy-aware service selection strategies introduced in Sect. 5.1. In addition, we consider a baseline *Random* strategy that randomly selects a functionally matching service; and a state-of-the-art *QoS-aware Learning*-based criterion [4], which serves as a benchmark to compare the impact on QoS of our energy-focused selection criteria.

We implemented a large-scale simulation model for the PeerSim simulator [8]. The replication package is publicly available to researchers interested in replicating and independently verifying the results presented in this paper[4].

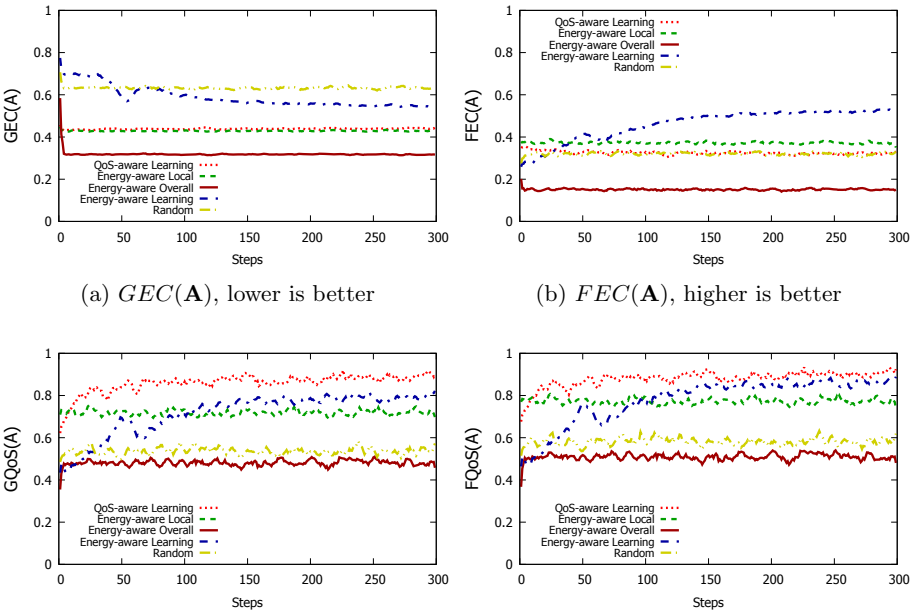
6.1 Experimental Settings

Our experimentation mimics a wireless sensor network (WSN) deployment scenario of an edge computing application. We consider a system with N services and NUM_INT different interface types $\mathbf{T} = \{T_1, \ldots, T_{\text{NUM\_INT}}\}$. Without loss of generality we assume that each sensor node hosts a single service. We create $\lfloor N/\text{NUM\_INT} \rfloor$ services of each type and, for each service, we define a probabilistic attachment to interface types to generate $S.Deps$ with probability p.

We define the energy cost of a k-bits CPU operation equal to the average energy cost of sending k-bits. Moreover, the energy cost of sending k-bits is on

[4] https://github.com/mi-da/Energy-Aware-Service-Assembly.

average two times more costly than the energy cost of receiving k-bits [5]. We deploy the services in a network area with a diameter of 200 m. The nodes are randomly positioned in the area and are endowed with symmetric latency links. Each node adopts a decentralized network coordinate system to estimate latency values [3]. Without loss of generality, we assume that the packet loss in the network is null. We adopt the first order radio model, a commonly used communication energy consumption model for WSN [5], which leads to the instantiation of $\phi_n^{req}(\delta, b, l)$ and $\phi_n^{prov}(\delta, b, l)$ (see Sect. 3.2). Finally, we assume that the load-dependent QoS function of each service S is a randomly monotonic decreasing function that returns values in the range $(0, 1]$ [4].



(a) $GEC(\mathbf{A})$, lower is better

(b) $FEC(\mathbf{A})$, higher is better

(c) $GQoS(\mathbf{A})$, higher is better

(d) $FQoS(\mathbf{A})$, higher is better

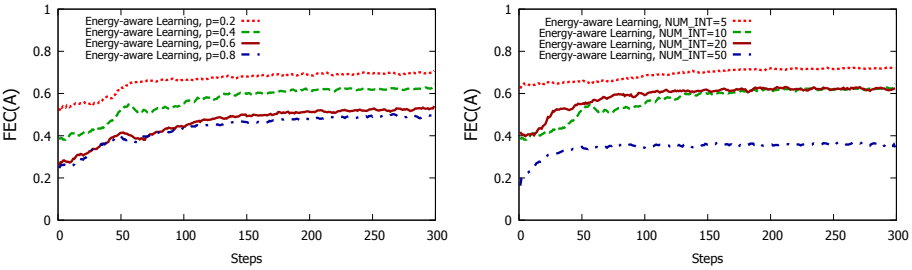
Fig. 4. Selection criteria effectiveness: $N = 500$, NUM_INT $= 10$, $p = 0.6$

6.2 Experimental Results

Each experiment shows the progress of our gossip-based decentralized architecture towards the construction of fully resolved assemblies[5].

Let us consider first the impact on the global indexes $GEC(\mathbf{A})$ and $GQoS(\mathbf{A})$. Figure 4a shows that all strategies are better than the baseline *Random* with respect to the overall energy consumption. The greatest energy saving

[5] The gossip procedure eventually leads to the creation of fully resolved assemblies.



(a) NUM_INT = 10, variable p (b) $p = 0.4$, variable NUM_INT

Fig. 5. $FEC(\mathbf{A})$ of different classes of applications, higher is better, $N = 500$

is achieved by the *Energy-aware Overall* strategy, thus confirming the effectiveness of its system-wide perspective, which comes however at the cost of disseminating energy-related information. This cost is not incurred by the *Energy-aware Local* strategy, whose energy saving performance is slightly worse and comparable to the one achieved by the energy-unaware *QoS-aware Learning* strategy. The *Energy-aware Learning* shows instead that its goal of leveling the energy consumption rate of all nodes is not very compatible with the goal of minimizing the overall energy consumption.

On the other hand, Fig. 4c obviously shows that the best overall QoS is achieved by the *QoS-aware Learning* strategy, while *Energy-aware Overall* has the worst performance, even worse than *Random*. *Energy-aware Local* and *Energy-aware Learning*, notwithstanding their focus on energy, have instead a good impact on the overall QoS, quite close to the result achieved by *QoS-aware Learning*. Given our assumption of load-dependent QoS, this is an indication that both these strategies distribute better the load among the different nodes with respect to *Energy-aware Overall*: in case of *Energy-aware Learning*, this is likely due to its intrinsic balancing attitude; in case of *Energy-aware Local* this is likely due to its myopic perspective, which has in this case the positive side-effect of leading to the selection of services hosted by nearby nodes, as they make each node incur in less energy consumption (which is greatly influenced by the energy required to amplify the signal in WSN [5]). This behaviour causes a sort of geographical load-balancing effect, where clusters of nodes form service assemblies that run approximately in the same geographical area.

Let us consider now the fairness indexes $FEC(\mathbf{A})$ and $FQoS(\mathbf{A})$. In this case, Fig. 4b shows that *Energy-aware Learning* achieves the best result in balancing the energy consumption rate among all nodes, according to its primary goal, while *Energy-aware Overall* is the worst. This very bad performance is due to its energy information global sharing that leads every node to get the same energy-related knowledge. As a result, the most energy efficient services will be globally targeted by binding requests, with very good results on the overall energy consumption, but at the cost of an unfair energy allocation. All the other three strategies achieve instead quite similar and satisfactory results with

respect to $FEC(\mathbf{A})$, with *Energy-aware Local* performing slightly better than the other two: for different reasons (the intrinsic randomness of *Random*, the energy-unawareness of *QoS-aware Learning*, the myopic perspective of *Energy-aware Local*), all these strategies have as a side effect a quite fair distribution of the energy load on the different nodes.

Figure 4d shows that also from a QoS perspective *Energy-aware Overall* performs very bad also in terms of fairness, thus confirming the negative impact that this strategy has on QoS because of the load unbalance it tends to favor. The balancing attitude of *Energy-aware Learning* leads instead to good results also in terms of QoS fairness (after a number of learning steps), quite close to the best result achieved by QoS-aware strategy.

Finally, we analyze the effectiveness of the *Energy-aware Learning* strategy on the $FEC(\mathbf{A})$ index for different classes of applications: they are simulated by varying independently the probabilistic attachment parameter (i.e., p) affecting the dependency set of a service, and the number of interface types (i.e., NUM_INT), affecting the maximum depth of a fully resolved assembly.

Our experiments (Figs. 5b-a) show that $FEC(\mathbf{A})$ decreases with increasing p and NUM_INT. These results highlight that architectures with many dependencies or many interface types impair the energy fairness of the system.

7 Threats to Validity

A threat to external validity concerns the approach evaluation. Indeed, we adopted an evaluation based on extensive simulations, instead of considering single case studies. However, to evaluate the practical implication of the adoption of our service assembly framework, we plan to select one of the existing service discovery platforms to support the actual implementation of our approach, so to validate it in a real-word settings.

A threat to internal validity is represented by the selection of the social welfare indexes. To smooth this threat we adopted two different indexes to complement measures of the overall energy consumption and overall QoS with fairness indexes. We are also planning to investigate the definition of other social welfare indexes to extend the validity of our approach.

8 Conclusion and Future Work

In this paper, we have proposed a decentralized architecture to build a fully functional assembly of distributed services, able to optimize its energy consumption in an open and heterogeneous execution environment, paying also attention to issues concerning the delivered quality of service. We also suggested suitable indexes to measure from different perspectives the energy efficiency of the resulting assembly, and presented the results of extensive simulation experiments to assess the effectiveness of our approach.

As future work we plan to analyse combined energy and QoS selection criteria, and to take into account also other sources of energy consumption. We

also plan to investigate issues concerning the presence of finite sources of energy (e.g., batteries) and differentiate between green and brown energy sources.

References

1. Caporuscio, M., Ghezzi, C.: Engineering future Internet applications: the prime approach. J. Syst. Softw. **106**, 9–27 (2015)
2. Cardellini, V., Casalicchio, E., Grassi, V., Iannucci, S., Presti, F.L., Mirandola, R.: MOSES: a framework for QoS driven runtime adaptation of service-oriented systems. IEEE Trans. Softw. Eng. **38**(5), 1138–1159 (2012)
3. Dabek, F., Cox, R., Kaashoek, F., Morris, R.: Vivaldi: a decentralized network coordinate system. In: Proceedings of SIGCOMM 2004, pp. 15–26. ACM, New York (2004)
4. D'Angelo, M., Caporuscio, M., Grassi, V., Mirandola, R.: Decentralized learning for self-adaptive QoS-aware service assembly. Future Gener. Comput. Syst. **108**, 210–227 (2020)
5. Heinzelman, W.R., Chandrakasan, A., Balakrishnan, H.: Energy-efficient communication protocol for wireless microsensor networks. In: Proceedings of HICSS 2000, vol. 2, p. 10 (2000)
6. Horcas, J.M., Pinto, M., Fuentes, L.: Context-aware energy-efficient applications for cyber-physical systems. Ad Hoc Netw. **82**, 15–30 (2019)
7. Jain, R.K., Chiu, D.M.W., Hawe, W.R.: A quantitative measure of fairness and discrimination for resource allocation in shared computer systems. Technical report. DEC-TR-301, Digital Equipment Corporation (1984)
8. Montresor, A., Jelasity, M.: PeerSim: a scalable P2P simulator. In: 2009 IEEE Ninth International Conference on Peer-to-Peer Computing, pp. 99–100 (2009)
9. Paolucci, M., Kawamura, T., Payne, T., Sycara, K.: Semantic matching of web services capabilities. In: First International Semantic Web Conference (2002)
10. Paschalidis, I.C., Tsitsiklis, J.N.: Congestion-dependent pricing of network services. IEEE/ACM Trans. Netw. **8**(2), 171–184 (2000)
11. Schaerf, A., Shoham, Y., Tennenholtz, M.: Adaptive load balancing: a study in multi-agent learning. J. Artif. Int. Res. **2**(1), 475–500 (1995)
12. Schroeder, B., Gibson, G.A.: A large-scale study of failures in high-performance computing systems. In: Proceedings of DSN 2006, pp. 249–258 (2006)
13. Shah, D.: Gossip Algorithms. Foundations and Trends in Networking, Now Publishers (2009)
14. She, Q., Wei, X., Nie, G., Chen, D.: QoS-aware cloud service composition: a systematic mapping study from the perspective of computational intelligence. Expert Syst. Appl. **138**, 112804 (2019)
15. Sun, M., Zhou, Z., Duan, Y.: Energy-aware service composition of configurable IoT smart things. In: 2018 14th International Conference on Mobile Ad-Hoc and Sensor Networks (MSN), pp. 37–42. IEEE (2018)
16. Tong, E., Chen, L., Li, H.: Energy-aware service selection and adaptation in wireless sensor networks with QoS guarantee. IEEE Trans. Serv. Comput. (2017)
17. Wang, S., Zhou, A., Bao, R., Chou, W., Yau, S.S.: Towards green service composition approach in the cloud. IEEE Trans. Serv. Comput. (2018)
18. Woods, E., Fairbanks, G.: The pragmatic architect evolves. IEEE Softw. **35**(6), 12–15 (2018)
19. Zeng, D., Gu, L., Yao, H.: Towards energy efficient service composition in green energy powered cyber-physical fog systems. Future Gener. Comput. Syst. **105**, 757–765 (2020)