A Decentralized Approach to Network-Aware Service Composition

Valeria Cardellini, Mirko D'angelo, Vincenzo Grassi, Moreno Marzolla, Raffaela Mirandola

A Decentralized Approach to Network-Aware Service Composition

Valeria Cardellini[1], Mirko D'Angelo[1], Vincenzo Grassi[1], Moreno Marzolla[2], and Raffaela Mirandola[3]

[1] Università di Roma Tor Vergata (Italy), Dip. di Ingegneria Civile e Ingegneria Informatica
cardellini@ing.uniroma2.it, mirko.dng@gmail.com, vgrassi@info.uniroma2.it
[2] Università di Bologna (Italy), Dip. di Informatica–Scienza e Ingegneria
moreno.marzolla@unibo.it
[3] Politecnico di Milano (Italy), Dip. di Elettronica, Informazione e Bioingegneria
raffaela.mirandola@polimi.it

Abstract. Dynamic service composition represents a key feature for service-based applications operating in dynamic and large scale network environments, as it allows leveraging the variety of offered services, and to cope with their volatility. However, the high number of services and the lack of central control pose a significant challenge for the scalability and effectiveness of the composition process. We address this problem by proposing a fully decentralized approach to service composition, based on the use of a gossip protocol to support information dissemination and decision making. The proposed system builds and maintains a composition of services that fulfills both functional and non functional requirements. For the latter, we focus in particular on requirements concerning the composite service completion time, taking into account both the response time and the impact of network latency. Simulation experiments show that our solution converges quickly to a feasible composition and can self-adapt to dynamic changes concerning both service availability and network latency.

Keywords: Peer to peer systems, service composition

1 Introduction

In this paper we address the problem of discovering and selecting the services needed to dynamically compose a given application developed according to the service-oriented paradigm and deployed in large scale networked systems. We assume that the application is architected as a set of required services logically connected through a workflow that specifies control and data dependencies among them. Such a workflow could be the result of a manual design process, or automatically created by planning approaches. Besides functional requirements concerning the desired service types, we also assume that the resulting composition must fulfill non-functional quality of service (QoS) requirements; in

particular, we focus on the overall completion time for the service delivered by the composition. Given the large scale environment we are considering, network latency may have a relevant impact on this performance goal.

The composition process must face uncertainty and complexity issues to fulfill both its functional and QoS requirements. Uncertainty is caused by the lack of stable and globally available information about available services, because of reachability problems and service autonomy. Complexity is caused by the potentially high number of required services in an application, spanning from tens up to hundreds or thousands [20], and corresponding candidate functionally equivalent services in the network.

Centralized approaches to service composition can hardly tackle these issues. Rather, they motivate the need of decentralized and self-adaptive approaches to achieve an adequate degree of robustness, resiliency and scalability. Approaches of this kind have already been proposed [9,19], mostly based on the assumption of a decentralized orchestration of the application workflow, using structured or unstructured P2P network architectures.

However, as we argue in Sect. 2, composite services workflows could be orchestrated in a decentralized or centralized way, and network latency affects their overall QoS differently, depending on which orchestration model is used. Hence, a comprehensive QoS-aware approach to service composition should take into account workflows orchestrated according to both models.

In this respect, the main contribution of this paper is a QoS-aware fully decentralized and self-adaptive approach to service composition, whose main features are: (*i*) the ability to deal with composite services workflows orchestrated according to both centralized and decentralized model; (*ii*) the adoption of an unstructured P2P approach to resource discovery and selection, based on the use of a gossip protocol that guarantees resilience to dynamic changes concerning service availability and network latency, and scalability with respect to the system size, thanks to the bounded amount of information maintained and exchanged among nodes. Simulation experiments show that our approach is able to quickly complete the composition process, and to quickly self-adapt to dynamic changes concerning both service availability and network latency.

The paper is organized as follows. We present in Sect. 2 the system model and in Sect. 3 the architecture of the decentralized P2P system and the algorithms for the self-adaptive dynamic service composition. In Sect. 4 we discuss simulation results that assess the system performance. Related works are briefly discussed in Sect. 5. Finally, we conclude and present directions for future work in Sect. 6.

2 System Model

We consider a large scale distributed system consisting of a dynamic set $\mathbf{N}$ of nodes, collectively offering a set $\mathbf{S}$ of *concrete services*. We denote by $node(s)$ the node hosting service $s \in \mathbf{S}$. Both the services in $\mathbf{S}$ and the latency among hosting nodes may dynamically change, because of events such as service providers disabling/enabling services, reachability problems of hosting nodes, or variations

in the network topology. We assume that a descriptor is associated with each concrete service, providing information about its functional and non functional characteristics. For the latter, we assume in particular that the descriptor of a service s includes the specification of $resptime(s)$, the estimated completion time of a service request addressed to s. We also assume the availability of function $dist : \mathbf{N} \times \mathbf{N} \rightarrow \Re$ that returns the round-trip latency between a pair of nodes. We discuss in Sect. 3 how it can be implemented in a scalable way, integrated in the overall architecture of the solution we propose.

This system is intended to support the execution of service-based distributed applications dynamically composed according to a workflow that defines control and data dependencies among different services. A workflow W is modeled as a directed acyclic graph (DAG) $W = (\Sigma_W, E_W)$, where:

- Σ_W is a set of nodes. Each node $\sigma \in \Sigma_W$ represents a *required service* for W, labeled with a specification of its functional requirements.
- E_W is a set of edges modeling data and control flow between services. Multiple edges exiting from or entering a node may model XOR, OR, or AND control logic; however, for the purpose of the problem addressed in this paper, we do not need to explicitly specify it.

Given a workflow W, each abstract service $\sigma \in \Sigma_W$ must be bound to a suitable concrete service $s \in \mathbf{S}$ for the workflow to be executed. To this end, we assume that there exists a function $matches : \Sigma_W \times \mathbf{S} \rightarrow [0, 1]$ such that $matches(\sigma, s) = 0$ if the concrete service s does not match the functional requirements of the abstract service σ, and $matches(\sigma, s) > 0$ if some matching exists according to some matching criterion [18,22]. Function $res : \Sigma_W \rightarrow (\mathbf{S} \bigcup \{null\})$ specifies the concrete service bound to an abstract service σ, where $res(\sigma) = s$ if σ is bound to $s \in \mathbf{S}$, $res(s) = null$ otherwise.

When the composition process for a workflow W starts, W has no matching concrete service bound to an abstract one, and can be considered as the template that drives the dynamic composition of the application. Workflow W becomes *fully resolved* when each $\sigma \in \Sigma_W$ is bound to a suitable concrete service, i.e., $res(\sigma) \neq null$ and $matches(\sigma, res(\sigma)) > 0$. Otherwise, W is *partially resolved*.

Centralized and Decentralized Orchestration. Once a workflow has been fully resolved, it may be orchestrated either according to a Centralized Orchestration (CO) or Decentralized Orchestration (DO) model [4]. In the DO model each service receives control and data directly from its immediate predecessors and, once it terminates its task, directly transfers them to its immediate successors. On the other hand, in the CO model interactions among services are mediated by a centralized coordinator, which receives control and data from each service and then transfers them to the suitable successor(s). The CO model simplifies the workflow management, but introduces additional delays caused by the indirect interaction between consecutive services, and may suffer from the typical problems of a centralized solution (bottleneck node, single point of failure). The DO model overcomes these problems, but requires the instantiation of workflow control logic at each node hosting a workflow resource [2, 4]. This

can be problematic, as such nodes could not be willing to host this logic, or could have limited capabilities that make them not capable of coordinating the workflow operations (as could be the case of nodes involved in Internet-of-Things scenarios). As a consequence, a comprehensive solution for QoS-aware decentralized service composition should consider both CO and DO models, taking into account their different impact on the global QoS. In this respect, an important QoS attribute for a fully resolved workflow is the time required to complete its operations. In the following we precisely define the *worst case completion time* of a fully resolved workflow W, denoted by $wcct(W)$, as a QoS metric based on this attribute, and provide expressions for calculating it in case of workflows orchestrated according to the CO or DO model.

Worst Case Completion Time. We define $wcct(W)$ as the maximum elapsed time from the arrival of a service request to W and the delivery of the final result. Roughly speaking, it corresponds to the length of the longest path in a fully resolved instance of W. For a more precise definition, we introduce the following notation.

Given an abstract service $\sigma \in \Sigma_W$, functions $Pred(\sigma)$ and $Succ(\sigma)$ return the set of predecessors and successors of σ in W respectively, i.e., $Pred(\sigma) = \{\zeta \in \Sigma_W \mid (\zeta, \sigma) \in E_W\}$, and $Succ(\sigma) = \{\tau \in \Sigma_W \mid (\sigma, \tau) \in E_W\}$. A *path* π in W is a sequence of abstract services $\pi = (\sigma_0, \sigma_1, \ldots, \sigma_k)$ such that there exists a dependency between each service and its successor, i.e., $(\sigma_i, \sigma_{i+1}) \in E_W$ for each $i \in \{0, 1, \ldots, k-1\}$. π is a *fully resolved path* if each service in it is bound to a concrete service matching its functional requirements, i.e., $res(\sigma_i) \neq null$ and $matches(\sigma_i, res(\sigma_i)) > 0$ for each $i \in \{0, 1, \ldots, k\}$.

The length $len(\pi)$ of a fully resolved path $\pi = (\sigma_0, \sigma_1, \ldots, \sigma_k)$ in a workflow W is defined as:

$$len(\pi) = \sum_{i=0}^{k} resptime(res(\sigma_i)) + \sum_{i=0}^{k-1} del\left(node(res(\sigma_i)), node(res(\sigma_{i+1}))\right) \tag{1}$$

where $del\left(node(res(\sigma_i)), node(res(\sigma_{i+1}))\right)$ denotes the network delay for transferring control and data from the node hosting $res(\sigma_i)$ to the node hosting $res(\sigma_{i+1})$. This delay depends on the adopted orchestration model and can be expressed as follows for the CO and DO models, respectively:

$$del_{\mathrm{CO}}\left(node(R_i), node(R_{i+1})\right) = 2 \cdot dist(node(R_i), node(coord_W)) \tag{2}$$

$$del_{\mathrm{DO}}\left(node(R_i), node(R_{i+1})\right) = dist\left(node(R_i), node(R_{i+1})\right) \tag{3}$$

where $R_k := res(\sigma_k)$ and $coord_W$ denotes a resource acting as coordinator of W in the CO case.

From (1) and (2) we see that the path length (and hence the value of $wcct(W)$) in the CO case is the sum of uncorrelated terms: changing the concrete service bound to an abstract service σ has only a local impact, as it does not affect the contribution to the overall path length of concrete services bound to other abstract services in the path. On the other hand, from (1) and (3) we see that the path length in the DO case is the sum of *correlated* terms: changing the

concrete service bound to an abstract service σ_i does affect the contribution to the overall path length of concrete services bound to abstract services σ_{i-1} and σ_{i+1}, since the delay for data and control transfer to/from σ_i could change.

Problem Statement. Given the system model described above, the problem addressed in this paper is how to devise a fully decentralized protocol that, given a workflow template W, is able to fully resolve W through a suitable composition of services offered by nodes in $\mathbf{N}$, and fulfill a QoS requirement on $wcct(W)$, that can be *threshold-based* (e.g., $wcct(W) < T$ for some suitable threshold T), or *min-based*, requiring the minimization of $wcct(W)$. Given the intrinsic dynamism of the system, the protocol must be able to adapt to modifications of the set of available resources and nodes topology.

3 System Architecture

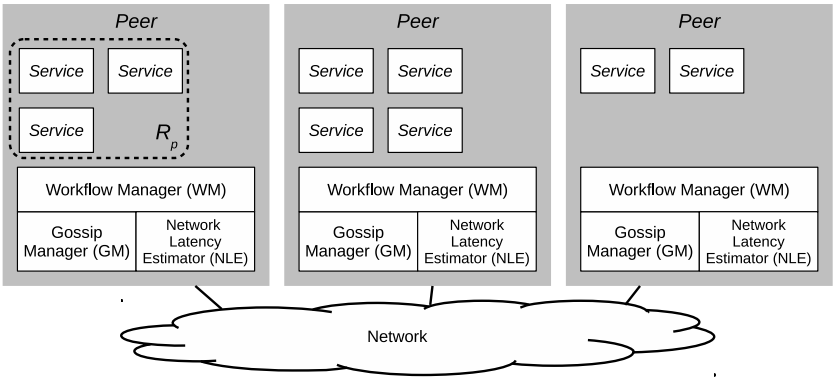

Fig. 1: System architecture.

Figure 1 shows the fundamental components of the architecture that allows achieving the goals stated in the previous section. Each node $n \in \mathbf{N}$ hosts three macro components: Network Latency Estimator (NLE), Workflow Manager (WM), and Gossip Manager (GM). The cooperation among instances of these three components hosted at each node produces the overall self-adaptive service composition process. In particular, NLE estimates the network delay among pairs of nodes in order to instantiate workflows satisfying the required QoS requirement, as discussed at the end of this section. WM is responsible for starting the composition of a new workflow according to the template received from the application layer, or the repair of an already composed workflow in case of modifications in the available resources. GM implements the decentralized information dissemination and decision-making, leading to the dynamic composition

and self-adaptation of workflows according to their functional and QoS requirements. GM instances hosted by different nodes cooperate to this end using a gossip communication model, which leverages epidemic protocols to achieve reliable information exchange among large sets of interconnected peers, also in presence of network instability (e.g., peers join/leave the system suddenly). Although it is in general not possible to analytically estimate the convergence speed of gossip algorithms, it is known that in most cases the number of iterations required to reach a "sufficiently good" solution is proportional to the logarithm of the network size [12]. Indeed, in Sect. 4 we will show that our gossip-based procedure achieves its goal very quickly. The WM and GM components hosted by a node n share a common state made of: (i) the set S_n of concrete services offered by n, (ii) the set $work_n$ of workflows node n is aware of and whose composition is underway, and, (iii) for each $W \in work_n$, the set $C_W = \{C_W(\sigma) \mid \sigma \in \Sigma_W\}$, where $C_W(\sigma)$ is a set of currently known concrete services that could be bound to the required service σ.

The Workflow Manager Component. The WM instance hosted by a node n starts its operations by receiving a fully unresolved or partially resolved workflow W, the former in case of a newly entering workflow, the latter for a workflow under repair. In both cases, WM adds W to the set $work_n$ of workflows under construction. This action triggers a composition phase for W (see Alg. 1). WM also associates W with a globally unique identifier denoted by $id(W)$, which remains associated with all the different instances of W resulting from the decentralized composition process. Hence, given two workflow instances W' and W'', $id(W') = id(W'')$ indicates that they both originate from the same template.

After that, WM enters a stage where it is ready to receive fully resolved instances of W as effect of the operations of the GM components (described below). This stage ends when either a maximum allowed number of candidates has been collected, or a timer expires. Then, WM selects the "best" realization of W from the set of candidates; the selection is based on some criterion that takes into account the requirement on $wcct(W)$ and, possibly, other quality factors (e.g., cost or reliability). Finally, WM sends a commit message to each concrete service bound to an abstract service in the selected W instance, and starts its execution and monitoring.

When the WM monitoring activity detects relevant changes for a workflow W in execution (e.g., the failure of a node hosting one of its services, a change in the actual latency for the interactions with a node), an adaptation action is triggered. It consists in reactivating the workflow composition procedure by inserting a partially resolved instance of W into the set $work_n$. This instance is built differently, depending on the orchestration model adopted for W:

- in the CO case, the instance is built by setting as *unbound* the abstract services originally bound to concrete services whose parameters (response time, latency) changed, while abstract services bound to unaffected concrete services can retain their bindings, because of the uncorrelation among different services.

– in the DO case, the partially resolved instance of W is built by setting as *unbound* the abstract service originally bound to a concrete service whose parameters (response time, latency) changed and, recursively, all services having at least one of their immediate predecessors set as *unbound* are also set as *unbound*. The rationale is that, as we will see in Alg. 3, binding a concrete service to an abstract service in the DO case depends on the selection performed for its immediate predecessors. Hence, invalidating a service invalidates all its successors.

The Gossip Manager Component. Algorithm 1 describes the core of the gossip algorithm for workflow composition, cooperatively executed by the GM components hosted by peer nodes. The goal of this algorithm is to determine a binding between the unbound abstract services in W, and matching concrete services in $\mathbf{S}$. To achieve this, all nodes iteratively exchange and merge their local state information concerning the workflow(s) whose composition is underway (i.e., the content of set $work_n$ and, for each $W \in work_n$, the corresponding set C_W). The algorithm is parametric with respect to function MERGE() used to this end, since its actual definition depends on the type of orchestration model (centralized or decentralized) that has been adopted to coordinate the workflow operations, as detailed below. Algorithm 1 includes an initialization phase and two concurrent threads: an active thread that starts an interaction by sending a message to a randomly selected node, and a passive thread that responds to messages received from other nodes. In the active thread, node n sends a message to the randomly selected peer p every Δt time units, where the message payload is the content of $work_n$. p is provided by the SELECTRANDOMNODE() function implemented in a underlying layer (e.g., based on the NEWSCAST service [11]). The passive thread listens for messages coming from other nodes. Upon receiving a message containing the set $work_q$ from some node q, the passive threads merges $work_n$ and $work_q$ (line 10). After that, it checks whether some workflow becomes fully resolved by effect of the merging procedure. If so, the fully resolved workflow is sent to the node that initiated its composition. In the next two subsections we complete the definition of the gossip-based service composition procedure by specifying the MERGE() function passed as input to Alg. 1, for the CO and DO scenario, respectively.

State Merging under the CO Model. Algorithm 2 implements the MERGE() function for the CO scenario where $wcct(W)$ is computed based on (2). MERGE() treats the resolution of each $\sigma \in \Sigma_W$ separately, aiming at the minimization of its completion time independently of the other services in Σ_W. Thanks to the uncorrelation among different services, as remarked in Sect. 2, this local minimization eventually leads also to the minimization of the overall value of $wcct(W)$. As a consequence, the gossip-based procedure will make the system eventually able to deliver one or more fully resolved instances of W that fulfill the min-based requirement on $wcct(W)$. For the same reason, the threshold-based requirement can be eventually fulfilled, provided that resources suitable to this

Algorithm 1 GM algorithm executed by node n

```
1: work_n ← ∅
2: procedure ActiveThread
3:     loop
4:         Wait Δt
5:         p ← SelectRandomNode()
6:         Send ⟨work_n⟩ to p
7: procedure PassiveThread
8:     loop
9:         Receive ⟨work_q⟩ from q
10:        work_n ← Merge(work_n, work_q)
11:        for all W ∈ work_n s.t. W is fully resolved do
12:            Send ⟨W⟩ to the initiator
```

Algorithm 2 Merge state information of n and q under a CO scenario

```
1: procedure Merge(work_p, work_q)
2:     work_n ← work_n ∪ work_q
3:     for all W ∈ work_n do
4:         W' ← {w ∈ work_n — id(w) = id(W)}
5:         for all σ ∈ Σ_W do
6:             C_W(σ) ← C_W(σ) ∪ C_W'(σ) ∪ {s ∈ S_n s.t. matches(σ, s) > 0}
7:             if C_W(s) ≠ ∅ then
8:                 s_best ← arg min_{s∈C_W(σ)} {resptime(s) + dist(node(s), node(coord_W))}
9:                 C_W(σ) ← {s_best}
10:                bind s_best to σ
11:        work_n ← work_n \ {W'}
```

purpose exist in the system (i.e. if the achievable minimum value for $wcct(W)$ is less than the required threshold).

State Merging under the DO Model Algorithm 3 implements the Merge() function for the DO scenario where $wcct(W)$ is computed according to (3). Differently from the centralized case, minimizing $wcct(W)$ cannot be decomposed into the local problems of minimizing the completion time of each abstract service $\sigma \in \Sigma_W$. This makes finding the optimal solution computationally infeasible [14]. Therefore, Alg. 3 adopts a heuristic procedure that tries to determine a "good enough" composition, without a strict guarantee that an optimal solution will be found. The procedure is based on a greedy approach similar to the ones proposed in [9, 19], which resolves services in Σ_W on a step-by-step basis, starting from the initial node of W: indeed, a required service σ is bound to a matching concrete service only if all preceding services in $Pred(\sigma)$ have already been resolved (line 7). The rationale for this mechanism is that only when $Pred(\sigma)$ is fully resolved, it is possible to know the worst case increment to the path length caused by resources in the set of known candidates for σ, and to select

Algorithm 3 Merge state information of n and q under a DO scenario

1: **procedure** MERGE($work_n, work_q$)
2: $work_n \leftarrow work_n \cup work_q$
3: **for all** $W \in work_n$ **do**
4: $W' \leftarrow \{w \in work_n$ — $id(w) = id(W)\}$
5: **for all** $\sigma \in \Sigma_W$ s.t. σ is not resolved **do**
6: $C_W(\sigma) \leftarrow C_W(\sigma) \cup C_{W'}(\sigma) \cup \{s \in S_n$ s.t. $matches(\sigma, s) > 0\}$
7: **if** $resolved(Pred(\sigma))$ **then**
8: $s_{\text{best}} \leftarrow \arg\min_{s \in C_W(\sigma)}\{resptime(s) + \max_{\sigma' \in Pred(\sigma)}\{dist(node(s), node(res(\sigma')))\}\}$
9: bind s_{best} to σ
10: **else**
11: keep in $C_W(\sigma)$ at most $K_{\max}$ concrete services s with smallest $resptime(s)$
12: **for all** $W \in work_n$ **do**
13: $W' \leftarrow w \in work_n$ such that $id(w) = id(W)$
14: $W_{\text{worst}} \leftarrow \arg\max_{w \in \{W, W'\}}\{max_{\pi \in \Pi_w}(len_{\text{DO}}(\pi)\}$
// Π_W is the set of all resolved paths in W
15: $work_p \leftarrow work_p \setminus \{W_{\text{worst}}\}$

the one causing the minimal increment (lines 7–9). Once σ has been resolved, the algorithm no longer tries to modify its binding (line 5), even if some better resource could be discovered in next rounds of the algorithm. This avoids cascading effects on successors of σ, which could lead to combinatorial explosion of the number of possible alternatives. However, it could prevent the discovery of a better solution. Hence, this greedy approach ensures that the system progresses towards the fulfillment of its functional requirement (i.e., the full resolution of W), but without a strict guarantee of eventually achieving the minimum value for $wcct(W)$, differently from the CO case.

As a final note, we point out that both merging algorithms guarantee that at each round of the gossip algorithm the total amount of exchanged information for the composition of a workflow W is upper bounded by $N \cdot |W|$, where N is the number of peer nodes and $|W|$ is the size of the W representation. This makes the composition procedure scalable, as its complexity at each round grows at most linearly with the number of nodes and workflow size.

Network Delay Estimation Estimating the network delay between pairs of peer nodes plays a crucial role for the selected QoS metric ($wcct$) driving the QoS-aware composition process. Indeed, in this context, the communication delay for the interactions with services located at different nodes could have a non-negligible impact, as some services could be offered by distant cloud servers. However, estimating the latency among services located at different nodes would in principle require probing all pairwise link distances, which would not scale with high numbers of concrete services and hosting nodes. For this reason the NLE components (see Fig. 1) collectively implement a *network coordinates* (NC)

system that provides an accurate estimate of the round-trip latency between any two network locations, without the need of an exhaustive probing. The NLE components maintain the NC system through a decentralized algorithm [6] with linear complexity with respect to the number of network locations, thus ensuring scalability. Moreover, as this NC algorithm adopts a gossip-based information dissemination scheme, the NLE components operations are easily integrated with the overall approach to service composition described above.

4 System Assessment

In this section we present a set of simulation experiments to assess the performance of the algorithms implementing the decentralized service composition. We evaluate the proposed approaches for the CO and DO scenarios over a large P2P network by exploiting the event-driven engine of PeerSim [16], a widely used discrete event simulator for P2P systems. We implemented the overlay network and the system architecture described in Sect. 3 on top of PeerSim. We initialized the P2P network topology with a specific number of nodes (by default 1000), modeled according to the scale-free Barabasi-Albert graph with power-law degree $\gamma = 3$. We set the replication degree of each resource to 5. For each concrete service s, value $resptime(s)$ is linearly distributed in the range $[100, 140]$ ms. We randomly select the nodes that host concrete services, without placing any concrete service on the workflow initiator; moreover, each node hosts a single concrete service. Such a service placement allows us to evaluate the performance of the algorithms in a worst case scenario.

For the workflow, we considered a layered structure, where each layer has one or more activities. We experimented with various alternatives, ranging from a "long" workflow with n layers and characterized by a sequence of n activities and 1 activity per layer, to a "short" workflow, characterized by having a single service in the first and last layers and $n-2$ parallel services in the middle layer, to a line workflow, where all services are in parallel on a single layer. For space reason, we report the results only for the long workflow, which represents a worst case scenario for the decentralized orchestration. If not otherwise specified, the sequential workflow has 10 distinct activities.

We modeled the network latency as a uniform random variable in the range $[10, 130]$ ms, which is consistent with Internet latency values. The NC system described in Sect. 3 is maintained by means of the Vivaldi algorithm [6]. With the described setting, each gossip round requires about 1 s.

For each experimental scenario, we run a set of 1000 experiments, each corresponding to a different network topology and a different random allocation of the services to the network nodes. Most of the experiments assume a single workflow to be resolved within $30s$, which, in our setting, represents a reasonable timeout for the resolution time (in the experiments that terminate with a fully resolved workflow, the resolution time is almost always less than $10s$).

In the discussed experiments, we mainly focus on the average *resolution time* as performance index, which is the time required for the initiator to receive a fully

resolved workflow averaged out the 1000 experiments. Specifically, we consider both the *threshold resolution time* and the *first resolution time*. The first is the time to receive a fully resolved instance of the workflow W that satisfies its $wcct(W)$ threshold requirement, while the latter is the time to receive the first fully resolved workflow. As additional performance index we consider the *recall*, which measures the system's ability to timely provide a fully resolved workflow. It is computed as the ratio of the number of experiments where the workflow is fully resolved within the timeout to the total number of experiments.

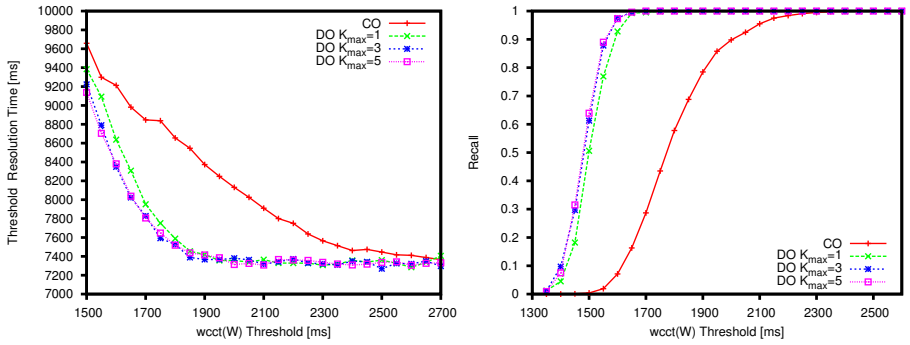

Fig. 2: Base scenario. (a) Resolution time vs. $wcct(W)$ threshold. (b) Recall vs. $wcct(W)$ threshold.

In the first set of experiments, we consider the base scenario, with 1000 network nodes and a long workflow with 10 activities. Figs. 2a and 2b show the threshold resolution time and the recall for the CO and DO models, when the $wcct(W)$ threshold varies. For the DO case, the peer can retain at most $K_{\max}$ candidate resources with the smallest execution time (see Alg. 3). We observe that the system gets more quickly and with a higher success probability a fully resolved workflow matching the threshold-based requirement in the DO case than in the CO case, because it is advantaged by the intrinsic greater efficiency of DO workflows, which makes more likely to match that requirement. For example, when the $wcct(W)$ threshold is set to 1.7 s, DO with $K_{\max} = 1$ finds the solution in less than 8 gossip rounds (8 s) and with a 99.5% probability, while CO requires almost 9 rounds (8.85 s) but with a low success probability equal to 29%. The DO curves show that the $K_{\max}$ parameter does not impact significantly on the performance; therefore, in the remaining experiments, we set $K_{\max} = 1$, thus saving storage space on the peer and network bandwidth.

We now analyze with Fig. 3 the scalability of the CO and DO models with respect to the number of nodes in the network, the length of the sequential workflow, and the resource replication degree. In these experiments, we consider the first resolution time as performance metric. We observe that under such a metric the performance achieved by the two orchestration models is quite similar: keep-

ing in $C_W(s)$ the matching resources as soon as they are discovered alleviates the performance penalty the DO model could suffer by the step-by-step composition. As expected, when the number of network nodes increases (see Fig. 3a), the resolution time augments as well, since the resources are more spread in the network and a large number of peers needs to be contacted. However, increasing by one order of magnitude the size of the network just requires two additional gossip rounds to fully resolve the workflow. In the remaining experiments, we set again the number of network nodes equal to the default value (1000). When the workflow length increases (see Fig. 3b), the resolution time grows as well, because a larger number of services needs to be resolved, but such increase is quite limited: changing the number of activities from 10 to 60 increases the resolution time only by 11.2% and 11.5% for CO and DO, respectively. Figure 3c shows that keeping the workflow length set to 10 sequential activities and increasing the replication degree of resources available for each service in the same response time range $[100, 140]$ ms, the first resolution time quickly decreases, because finding a good resource requires less message exchange.

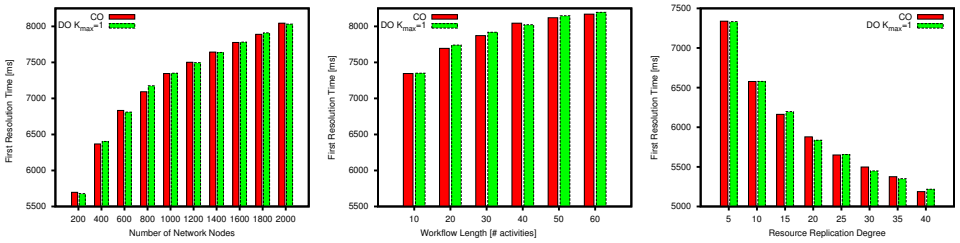

Fig. 3: Scalability analysis: first resolution time vs. (a) number of network nodes; (b) workflow length; (c) replication degree of resources.

Finally, we examine in Fig. 4 the ability of the proposed approach to quickly self-adapt to dynamic changes concerning both service availability and network latency. Differently from the previous sets of experiments, we now consider 30 concurrent long workflows to resolve, and inject the failure of either 10% of network nodes (Fig. 4a) or 10% of network links (Fig. 4b) at 10 and 20 s simulation times. Each long workflow is randomly composed by choosing 10 distinct activities between 20 available ones; all the other parameters take the default value. As performance metric, we consider the percentage of fully resolved workflows as the time flows and set the *wcct* threshold requirement to 1.9 s. As explained in Sect. 3, the WM component that detects the failure reactivates the workflow composition procedure. We see from Fig. 4 that the repair of workflows orchestrated according to the DO model is faster and more extensive than that achieved by the CO model, whose repair is slower and limited to less than 70% of the workflows. The reason is that even if CO needs to repair only the failed nodes or links, it is penalized by the higher communication latency between consecutive services which makes harder to stay below the required threshold.

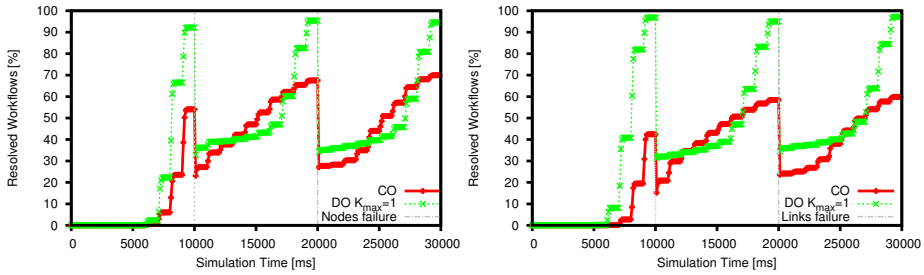

Fig. 4: Self-adaptation. (a) Failure of network nodes. (b) Failure of network links.

5 Related Work

The dynamic service composition problem in pervasive computing systems has different challenging aspects, mainly related to possible changes in the environment and types of available resources. Many approaches dealing with these issues have been proposed in the literature (e.g., [5, 7–9, 17]). Existing solutions can be broadly divided into centralized or decentralized service composition. Considering *centralized* composition environments, surveys on the topic can be found in [3] and [10], the latter for reliability aspects. Centralized approaches are based on a single broker that determines the service selection and coordinates the service orchestration. This broker may represent a processing and communication bottleneck, and a single point of failure. Therefore, traditional centralized techniques are not sufficient to address the application needs in dynamic and heterogeneous decentralized environments, and various recent works have focused on *decentralized* approaches to service composition on top of P2P overlay networks. Broadly speaking, there are two main families of P2P overlay networks: *structured* and *unstructured* overlays.

Structured overlay networks are tightly controlled and are generally based on a Distributed Hash Table (DHT) for resource and data management. Groba and Clarke [9] propose a service composition protocol that invokes service providers in ad hoc networks with the goal of minimizing the impact of topology changes and reducing failures. The proposed approach supports parallel service flows, but presents some limits with respect to composite complexity, network density, and service demand. Repantis et al. [19] focus on stream processing applications and propose a service composition approach where the component discovery phase exploits a structured P2P network and precedes the probe-based composition phase. We also mention the work in [15], that proposes DANS, a decentralized multimedia workflow processing system. DANS exploits a DHT and deals with scalability considering redundancy in the system, in terms of availability of multiple nodes able to perform the same task.

Unstructured overlays do not impose any constraint on the structure of the network, and allow peers to join and leave freely. A gossip-based decentralized technique for service composition in unstructured P2P networks has

been exploited by Furno and Zimeo [7]. However, they focus on cooperative semantic discovery and composition using each peer's local service repository by means of semantic matchmaking capabilities, rather than on QoS-aware composition; therefore, their proposal can complement our own. The works in [8, 21] propose QoS-driven gossip-based approaches to resolve hierarchies of component dependencies. However, they do not consider issues concerning workflow orchestration and impact of network delays on the overall system QoS. Other decentralized service composition methods include nature-inspired approaches. Mostafa et al. [17] propose a decentralized composition mechanism based on the notion of stigmergy, taking inspiration from the interactions exhibited by social insects to coordinate their activities. However, they focus on trust measures as a criterion for service selection, without considering the latency issue, and leave open the question of how the decentralized mechanism could be architected. A physics-inspired approach based on the friction concept is presented in [1], with the goal of minimizing the waiting time of service requests. However, only sequential workflows are supported and latencies between services are not taken into account. Multi-agent techniques have also been investigated to deal with service composition in mobile ad hoc and pervasive environments, e.g., [5].

6 Conclusions

In this paper we have presented a decentralized approach to network-aware service composition, based on the use of an epidemic protocol for information dissemination. Differently from most previous works that only focus on service composites with decentralized orchestration, we explicitly consider the case of centrally orchestrated service composites, and the different impact these two orchestration models have on a decentralized network-aware procedure for service composition. Simulation experiments show that our approach can quickly build a service composite fulfilling given functional and QoS requirements, and can self-adapt to changes concerning resource availability and network latency. Moreover, the proposed gossip-based procedure requires a bounded amount of information to be exchanged and maintained at each peer for each composite service, independently of the overall number of peers in the system, thus guaranteeing scalability.

We plan to extend our approach along several directions: (i) consider multiple QoS attributes; (ii) explicitly take into account resources offered at different levels (e.g., IaaS, PaaS, SaaS) and investigate whether this would require a refinement of our approach; (iii) assess whether hierarchical gossiping protocols [13] can improve the scalability of our approach; (iv) extend the present work to workflow graphs with cycles (observe that, in case an upper bound can be given to the number of times a cycle is executed, the extension is straightforward).

Acknowledgments V. Cardellini acknowledges the support of ICT COST Action IC1304 ACROSS.

References

1. Ahmed, T., Srivastava, A.: Minimizing waiting time for service composition: A frictional approach. In: Proc. of IEEE ICWS 2013. pp. 268–275 (2013)
2. Atluri, V., Chun, S.A., Mukkamala, R., Mazzoleni, P.: A decentralized execution model for inter-organizational workflows. Distrib. Parallel Databases 22(1) (2007)
3. 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. Software Eng. 38(5), 1138–1159 (2012)
4. Chafle, G.B., Chandra, S., Mann, V., Nanda, M.G.: Decentralized orchestration of composite web services. In: Proc. of WWW Alt. 2004. pp. 134–143. ACM (2004)
5. Cruz Torres, M.H., Holvoet, T.: Self-adaptive resilient service composition. In: Proc. of 2nd Int'l Conf. on Cloud and Autonomic Computing (2014)
6. Dabek, F., Cox, R., Kaashoek, F., Morris, R.: Vivaldi: A decentralized network coordinate system. In: Proc. of ACM SIGCOMM 2004. pp. 15–26 (2004)
7. Furno, A., Zimeo, E.: Self-scaling cooperative discovery of service compositions in unstructured P2P networks. J. Parallel Distrib. Comput. 74(10), 2994–3025 (2014)
8. Grassi, V., Marzolla, M., Mirandola, R.: Qos-aware fully decentralized service assembly. In: Proc. of SEAMS 2013. pp. 53–62. IEEE (2013)
9. Groba, C., Clarke, S.: Opportunistic service composition in dynamic ad hoc environments. IEEE Trans. Serv. Comput. 7, 642–653 (2014)
10. Immonen, A., Pakkala, D.: A survey of methods and approaches for reliable dynamic service compositions. Serv. Oriented Comput. Appl. 8(2), 129–158 (2014)
11. Jelasity, M., Kowalczyk, W., van Steen, M.: Newscast computing. Tech. Rep. IR-CS-006.03, Dept. of Computer Science, Vrije Universiteit (2003)
12. Kempe, D., Dobra, A., Gehrke, J.: Gossip-based computation of aggregate information. In: Proc. of SFCS 2003. pp. 482–491 (2003)
13. Kermarrec, A.M., Massoulié, L., Ganesh, A.J.: Probabilistic reliable dissemination in large-scale systems. IEEE Trans. Parallel Distrib. Syst. 14(3), 248–258 (2003)
14. Klein, A., Ishikawa, F., Honiden, S.: SanGA: A self-adaptive network-aware approach to service composition. IEEE Trans. Serv. Comput. 7(3), 452–464 (2014)
15. Kwon, G., Candan, K.S.: DANS: Decentralized, autonomous, and network-wide service delivery and multimedia workflow processing. In: Proc. of ACM MULTIMEDIA 2006. pp. 549–558 (2006)
16. Montresor, A., Jelasity, M.: PeerSim: A scalable P2P simulator. In: Proc. of 9th Int'l Conf. on Peer-to-Peer Computing. pp. 99–100 (2009)
17. Mostafa, A., Zhang, M., Bai, Q.: Trustworthy stigmergic service composition and adaptation in decentralized environments. IEEE Trans. Serv. Comput. (2015)
18. Paolucci, M., Kawamura, T., Payne, T., Sycara, K.: Semantic matching of web services capabilities. In: The Semantic Web - ISWC 2002, LNCS, vol. 2342 (2002)
19. Repantis, T., Gu, X., Kalogeraki, V.: Qos-aware shared component composition for distributed stream processing systems. IEEE Trans. Parallel Distrib. Syst. 20(7), 968–982 (2009)
20. Schuhmann, S., Herrmann, K., Rothermel, K., Boshmaf, Y.: Adaptive composition of distributed pervasive applications in heterogeneous environments. ACM Trans. Auton. Adapt. Syst. 8(2), 10:1–10:21 (2013)
21. Sykes, D., Magee, J., Kramer, J.: FlashMob: distributed adaptive self-assembly. In: Proc. of SEAMS 2011. pp. 100–109. ACM (2011)
22. Val, E.D., Rebollo, M., Vasirani, M., Fernández, A.: Utility-based mechanism for structural self-organization in service-oriented MAS. ACM Trans. Auton. Adapt. Syst. 9(3), 12:1–12:24 (2014)