Context-Aware Service Delegation for Opportunistic Pervasive Computing

Context-Aware Service Delegation for Opportunistic Pervasive Computing

Juan Luis Herrera[1][0000−0002−2280−2878], Hsiao-Yuan Chen[2][0009−0002−5635−8354], Javier Berrocal[3][0000−0002−1007−2134], Juan M. Murillo[3][0000−0003−4961−4030], and Christine Julien[2][0000−0002−4131−4642]

[1] University of Bologna, Italy
juanluis.herrera@unibo.it
[2] University of Extremadura, Spain
{jberolm, juanmamu}@unex.es
[3] University of Texas at Austin, USA
{littlecircle0730, c.julien}@utexas.edu

Abstract. The growth of capabilities of mobile devices allows them to host increasingly sophisticated application services. Emerging paradigms within the Cloud Continuum are based on the concept of running services closer to users, even on their own devices. Nonetheless, running collaborative services on these devices requires attention to constraints that differentiate the devices from cloud servers, such as limited battery or constrained data plans. New techniques are required to empower users to, first, execute services locally and, second, allow services to migrate to other available devices. We define *Pervasive Delegation Of Services* (PODS), a framework to opportunistically select the best candidate to run a service from among those in a local network, as well as to delegate it to others whenever the contextual circumstances change.

Keywords: Service Offloading, Service Delegation, Pervasive Computing, Opportunistic Computing, Mobile Computing

1 Introduction

Computing capabilities of end devices have increased enormously, with new paradigms in the *cloud continuum* and *mist computing* [1] bringing computation closer to users. Increasingly, services are deployed on nodes closer to or even owned by end users, increasing privacy and decreasing application response time. The development of new, more collaborative applications with the potential to leverage these paradigms is also growing. Liveboard [2] allows multiple people to edit whiteboards locally and have them shared and synchronized with others when connectivity allows. Such applications still often make use of the cloud, or the furthest parts of the cloud continuum, to enable data integration, processing, and communication between users. This may result in a loss of privacy and a delay in response time, metrics that can be critical to the user experience.

The increasing capabilities of devices allow them to be used not only as a user interface to applications, but also as communication mediators that can manage

exchanges with peers (e.g., through device-to-device communication or opportunistic networks [3]). Shared whiteboards can be locally hosted and aggregated among co-located devices, without the need to relegate such services to the cloud or even an intermediate edge node. Applications like HTC Power to Give [4] allow mobile users to host services that can be consumed using BOINC [5]. Games like Minecraft [6] also leverage users' mobile devices to host multiplayer servers.

While such services can be supported using various levels of the cloud continuum, we examine what is necessary to support service execution on devices that are completely separated from the cloud. In some sense, this is extreme—we are rarely completely disconnected from the Internet. However, from a research perspective, such an approach allows us to examine what is feasible and to determine the degree to which the approach increases user privacy by providing local data isolation, decreases response time by reducing round-trips to the cloud, and lessens bandwidth demands on networks that connect users to the cloud.

We focus on *collaborative services* and in particular on services that facilitate digital collaboration among co-located users. Our end devices like smartphones are always with us and seem a natural conduit for facilitating these services. However, these devices have several limitations: they are resource constrained in terms of energy and computation; they are mobile, resulting in changing connectivity over time; and they have other workloads and are not dedicated solely to the service.

Therefore, while there is promise for delegating locally available services to end devices, new techniques are required to support delegation that is: locally facilitated among the participating devices themselves; context-dependent, using the state of the devices to decide delegations; and dynamic, periodically adjusting the configuration as the context changes. We characterize the use of a completely decentralized architecture for such services as the *collaborative mist* and place it in the context of other paradigms in Section 2, where we also examine existing approaches to service delegation, off-loading, and on-loading. We then present the *Pervasive Delegation Of Services* (PODS) framework in Section 3; PODS provides dynamic, context-dependent service delegation that is coordinated entirely among a set of co-located nodes using only device-to-device interactions. The use of PODS is likely to be highly applicable for services such as collaborative multimedia sharing and processing, collaborative office automation suites, or applications deployed in intermittent connectivity scenarios. The key novelty of PODS is the delegation of services to available nearby devices based on the context of the situation, automatically adapting the collaborative services' deployment characteristics to the dynamic scenario over time, in an entirely peer-to-peer manner.

2 Motivation

We next provide an introduction of aspects of the cloud continuum, and place the *collaborative mist* in this context. We then examine work related dynamic

placement of services and motivate the gap that we seek to fill with PODS. We close this section with a detailed motivating application example for PODS.

The Cloud Continuum (and Beyond). The *cloud continuum* characterizes a design space that moves from completely centralized services in a pure cloud approach, through *fog computing*, which pushes infrastructure resources nearer the edge devices and *mist computing*, which starts to leverage the computational resources of the end devices themselves [1]. As the services become increasingly decentralized, end devices take on more responsibility and ownership of user data, increasing privacy. To leverage the capabilities of these architectures and better utilize end devices, conceptual frameworks for organizing applications and services have also emerged [7,8]. However, these applications and paradigms still require a central coordinator, usually the cloud, to at least make decisions about where services should be placed and distributed.

These paradigms have made great strides to improve user privacy, leverage the rich capabilities of end devices, and reduce the load on the infrastructure. However, the feasibility and implementation of such approaches depend on the computing capacity and available resources of the mobile end-user devices. Deciding which device or devices should host each service should account for the capabilities and situations of nearby devices. Hence, closing the gap remaining in realizing the collaborative mist requires higher-level yet completely distributed coordination system.

A Motivating Application. Consider an application to generate and compose video collaboratively among people at a party. From a computational perspective, such a video composition application can be easily executed on a mobile device. Collaboratively created videos can also be shared among the attendees in an opportunistic way, increasing the privacy of the attendees and reducing the network load. A service running on the mobile phone of the video organizer would be responsible for composing videos received from party-goers. Architecturally, the application could be composed of a single service whose API would have, as input, images from attendees, and as output, the generated video.

There are several barriers to implementing such an application. First, smartphones have limited resources: during video generation, the device hosting the video composition service could run out of battery, or have its user launch applications that compete for its resources. Moreover, the user running the collaborative service may move or even leave the party. Finally, there may be more powerful nodes available in the surroundings, able to host the service. The collaborative mist can bring the computation to the end devices themselves. End devices should also be supported in determining whether the service to be executed can be hosted locally, or whether it needs to be migrated to another node. In this application, the collaborative video organizer device itself can check if it ought to delegate the video composition service to another device. To empower users to manage their own information and applications in collaborative communities, we present PODS, a framework to perform service delegation in opportunistic and pervasive environments.

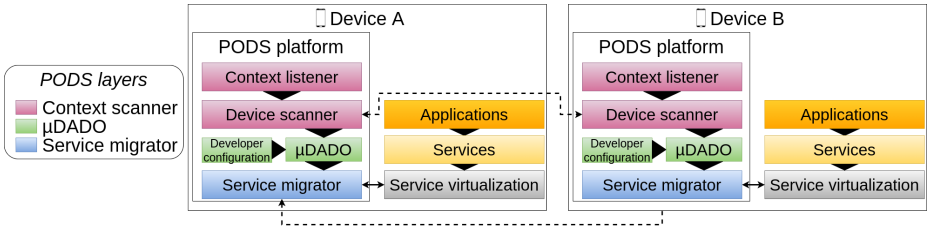

Fig. 1: Strawman architecture of the delegation system

3 Proposed architecture

We next introduce the PODS architecture (Fig. 1). PODS provides services in the collaborative mist. Each service may be hosted by more than one device and may receive requests from more than one device. In this context, we identify two roles: *delegatee*, i.e., a device that is currently running an instance of the service, and *consumer*, i.e., a device that requests to use the service's functionality. These roles are not mutually exclusive: a delegatee may also be a consumer of the same service. A service of a collaborative application executing on a delegatee may have been started by that device, or the instance of the service may have been assigned to it at some point during the execution of the application.

PODS consists of three high-level functionalities. First, PODS monitors the resources and contextual attributes of nearby computing nodes, including those devices participating in a collaborative session (context scanner in Fig. 1). Second, PODS analyzes which device(s) in the near surroundings should host each collaborative service based on their context, optimizing application-specific Key Performance Indicators (KPIs) to obtain a good user experience (μDADO in Fig. 1). Third, in response to detected changes and updated analyses, PODS migrates services among co-located nodes (service migrator in Fig. 1). This allows running collaborative applications in a more reliable way when multiple devices make their resources available to support service execution.

3.1 Context scanner

To enable PODS to detect the best suitable delegatees based on their context, as well as to adjust the number of replicas used, there are two key requirements that must be fulfilled. On the one hand, PODS requires information about a device's context to evaluate the device's suitability as a delegatee, as well as to determine if the number of executed replicas is appropriate. On the other hand, PODS needs to know about the context, not only of the device itself but also of other nearby devices. The context scanner fulfills both of these roles in PODS, through the use of two submodules, the *context listener* and the *device scanner*.

The context listener runs on every PODS device; its objective is to obtain a view of the current context of its device. PODS relies on a generic representation of context, using a simple $\langle key, value \rangle$ pair for each context attribute. This

allows the use of context to be flexible and extensible for different applications to use diverse attributes. In our implementation, we rely on measurements of the device's *battery level*, *available RAM*, and *network connection to other devices*. The context listener updates the *snapshot* within PODS when triggered, either by a timer or by a change in a sensed value that directly impacts the availability of a service on the device, e.g., low battery warning or a network disconnection or connection event, which is notified through a publish-subscribe mechanism.

For the service migrator to make delegation decisions, it needs the context of the local device and the context of other connected devices. The device scanner queries other nearby devices for their context attributes. Once the context listener updates the local context, it triggers the device scanner to solicit the other devices' contexts. This allows each PODS device to maintain an up-to-date view of the status of the local network: current delegatees for each service, context status of each device, and, implicitly, the availability of services (e.g., if the only delegatee of a service has disconnected from the network, the service is unavailable for all its consumers in the local network). Possible interruptions of a delegation process are also handled by the device scanner. If, during the delegation of a service (described in Section 3.3), either the current or target delegatees disconnect, the device scanner re-triggers the context listener to update the context information. Once new context information is available, the context scanner feeds it into the next subsystem: μDADO.

3.2 μDADO

The next step is for PODS to decide which devices are the most suitable delegatee(s) for each service, as well as how many replicas should be instantiated. In environments with centralized coordination, similar problems have been addressed, for instance using the DADO framework [9]. DADO can optimize the number of replicas of each service, the device each replica should be deployed on, and the network configuration. However, DADO is unsuitable for the collaborative mist scenarios. First, while DADO can *output* the desired replica configuration, it is unable to instantiate the replicas directly. Moreover, DADO operates over a centrally controlled software-defined network, rather than an entirely opportunistic one. Third, although DADO automatically optimizes KPIs such as response time, services in the collaborative mist may want to optimize other (service-specific) KPIs, for instance, to minimize their effects on user activity (e.g., energy usage, relative RAM consumption). Finally, in situations with large numbers of devices, DADO can take several hours to generate a deployment plan [9]. This is not acceptable in opportunistic situations, which are highly dynamic and require quickly reaction to environmental changes.

Thus, we propose μDADO, a decision subsystem for PODS, adapted to opportunistic environments to execute in a fast and lightweight way. Given the input from the context scanner, μDADO outputs the number of replicas and optimal placement for a service. For flexibility, μDADO allows developers to configure and define the KPIs that their services should optimize. μDADO uses

Mixed-Integer Linear Programming (MILP) to model the mathematical optimization problem, with a set of parameters (i.e., inputs of the system), decision variables (i.e., expected outputs the system will optimize), an objective function (i.e., metrics to be optimized), and a set of constraints (i.e., rules of validity).

We first model the information obtained from the context scanner as an input to the problem. Let D be the set of devices detected by the context scanner. Let K be the set of needed KPIs that μDADO obtains from all neighboring devices based on their context snapshots. We define the matrix V, of dimensions $|K| \times |D|$, which contains the values of each KPI in each device. Hence, $V_{kd}, k \in K, d \in D$, is the value of KPI k in the device d, as obtained by the device scanner. We assume that each group of users is strongly connected, i.e., that all the devices are neighbors of each other. If a user loses connection with devices in a group, it is considered to be its own group. If the device is able to connect to the devices in the group again, it will *rejoin* the group. Each device independently computes its own V, but, the computation of V_{kd} is deterministic, i.e., any two devices will compute the same value of V_{kd} because they receive the same context information from d. Nonetheless, the local decision taken by μDADO is only considered if either the device is a delegatee, or if there are no available delegatees.

Continuing with the developer-defined configuration, let O be a vector of integer values of size $|O| = |K|$. Each element $O_k, k \in K$ takes the value of 1 if the KPI should be minimized, -1 if the KPI should be maximized, and 0 otherwise. This lets the developer decide which metrics are to be optimized for each service. Similarly, the developer can define constraints over these KPIs that should be met by delegatees. Let EQ be a binary vector, of size $|EQ| = |K|$, and an additional vector of real numbers, $EQ^v, |EQ^v| = |K|$. A binary element $EQ_k, k \in K$ will take the value of 1 if a valid delegatee d must have a value on their KPI k equal to EQ^v_k. Analogously, the vectors GT and GT^v are defined to require a KPI of delegatees to be *greater than* a given value, and LT, LT^v to require them to be *less than* a value. These can be combined to constrain KPIs to be within a given range using GT and LT, or to create constraints such as "greater than or equal to" combining GT and EQ. For a given service, the vectors O, EQ, LT, and GT are identical for all of the devices.

Next, we define the problem's decision variables. To perform delegatee selection we define a binary vector, X, of size $|X| = |D|$. Each binary element in the vector, $X_d, d \in D$, will take the value of 1 if the device d is selected to host a replica of the service. Moreover, it is important to consider whether each device can or cannot access the service. Thus, we define a binary vector $A, |A| = |D|$, where $A_d, d \in D$ will take the value of 1 if the device d can access the service.

There are three essential elements that the objective function must optimize: the KPIs the developer configured, the service's availability (i.e., the number of devices that can communicate with at least one delegatee), and the number of instantiated replicas. The term of the objective function handling the developer-configured KPIs is calculated as the summation of the KPIs in a delegatee; the KPIs used are limited to those that are indicated by the O vector. The term handling the service's availability calculates and minimizes the number of

devices that cannot access the service. The third term, the number of replicas, simply counts the number of delegatees. Thus, the objective function (Eq. 1) is the minimization of the summation of these three terms:

$$\min \sum_{k \in K} \sum_{d \in D} (O_k V_{kd} X_d) + \sum_{d \in D} (1 - A_d) + \sum_{d \in D} X_d \tag{1}$$

To account for the developer-defined constraints on the KPIs, we include:

$$X_d V_{kd} EQ_k = EQ_k EQ_k^v \forall d \in D, k \in K \tag{2}$$

$$X_d V_{kd} GT_k \geq GT_k^v \forall d \in D, k \in K \tag{3}$$

$$X_d V_{kd} LT_k \leq LT_k^v \forall d \in D, k \in K \tag{4}$$

Formally, we can declare the problem μDADO solves as optimizing Eq. 1, subject to Eqs. 2-4. The developer provides μDADO with a configuration file , specifying their KPIs and constraints of interest. This configuration is then passed to a solver for this MILP formulation. In terms of temporal complexity, there are a total of $2^{2|D|}$ solutions for any given scenario, and hence, the problem has exponential complexity. On the other hand, in terms of memory, the total memory required for a problem instance of μDADO grows as a second-degree polynomial function of the number of devices and KPIs ($2|D| + 8|K| + |K||D| + |D|^2$).

3.3 Service migrator and virtualization platform

The service migrator takes on the task of executing the migration to achieve the configurations that μDADO identifies. To do so, the service migrator first checks if a migration must be performed, as the target delegatee(s) may be the current delegatee(s). In case a delegatee differs, the migrator must know the number of currently executing replicas and the target number of replicas, creating or removing as many replicas as necessary. Finally, because many services are stateful, to move a service to another delegatee, the service migrator must take care to transfer the state along with the service. In particular, the service migrator packages the state of the service, including an data, files, or databases it accesses or modifies, and shares these with the target delegatee, which must reinstate the service in the same state as the previous delegatee. To maintain a coherent system of services across devices using PODS, these services should be provided as self-contained, self-provisioning, and ideally cross-platform packages containing the service, as well as its dependencies, if they exist. To have self-contained and self-provisioning packages that run in a cross-platform manner and have their status saved on one machine and loaded in another, PODS relies on an underlying service virtualization platform. The role of such a platform is to minimize the friction in service delegation, allowing services to run *machine-agnostically*, i.e., not needing to control the operating system they are running on or the dependencies and programs installed within the device. To maintain service availability continuously, the migration process first stands up the service on any new delegatees. Once these new instances are running, the PODS instance on the delegatee device notifies the PODS instance on the device that previously hosted the service, notifying it that the service can be shut down.

4 Conclusion and Future Work

The advent of paradigms in the *Cloud Continuum* motivates research focused on moving services closer to users. However, most approaches currently focus on moving services from the cloud to the mist layers, rather than into peer-to-peer delegation. This paper presents PODS, a platform for the management and delegation of services for opportunistic pervasive computing environments. In the future, we expect to perform experiments over multiple changing conditions.

Acknowledgment

This work has been partially funded by the projects TED2021-130913B-I00 and PDC2022-133465-I00, funded by MCIN/AEI/10.13039/501100011033 and by the European Union "Next GenerationEU/PRTR", by the Department of Economy, Science, and Digital Agenda of the Government of Extremadura (GR21133), and by the European Regional Development Fund. This work was also partially supported by the European Union under the Italian National Recovery and Resilience Plan (NRRP) of NextGenerationEU, partnership on "Telecommunications of the Future" (PE00000001 - program "RESTART"). CUP: J33C22 002880001 and by the *Whole Communities–Whole Health* Research Grand Challenge at the University of Texas at Austin.

References

1. L. Bittencourt *et al.*, "The internet of things, fog and cloud continuum: Integration and challenges," *Internet of Things*, vol. 3, pp. 134–155, 2018.
2. LiveBoard Inc., "LiveBoard - Online Interactive Whiteboard App For Educators," 2022. [Online]. Available: https://www.liveboard.online/
3. X. Fu, H. Yao, O. Postolache, and Y. Yang, "Message forwarding for wsn-assisted opportunistic network in disaster scenarios," *Journal of Network and Computer Applications*, vol. 137, pp. 11–24, 2019.
4. HTC, "HTC Power To Give," p. 1, 2015. [Online]. Available: https://www.htc.com/us/go/power-to-give/
5. D. P. Anderson, "Boinc: a platform for volunteer computing," *Journal of Grid Computing*, vol. 18, no. 1, pp. 99–122, 2020.
6. Mojang, "Minecraft - Applications in Google Play," 2023. [Online]. Available: https://play.google.com/store/apps/details?id=com.mojang.minecraftpe
7. S. Laso *et al.*, "Human microservices: A framework for turning humans into service providers," *Softw. Pract. Exp.*, vol. 51, no. 9, pp. 1910–1935, 2021.
8. M. Villari *et al.*, "Osmosis: The osmotic computing platform for microelements in the cloud, edge, and internet of things," *Computer*, vol. 52, no. 8, pp. 14–26, 2019.
9. J. L. Herrera, J. Galán-Jiménez, J. Berrocal, and J. M. Murillo, "Optimizing the response time in sdn-fog environments for time-strict iot applications," *IEEE Internet of Things Journal*, vol. 8, no. 23, pp. 17 172–17 185, 2021.