Coordinating Computation at the Edge: a Decentralized, Self-Organizing, Spatial Approach

Roberto Casadei and Mirko Viroli
Department of Computer Science and Engineering
ALMA MATER STUDIORUM–*Università di Bologna*
Cesena, Italy
Email: {roby.casadei,mirko.viroli}@unibo.it

***Abstract*—Fog and Mobile Edge Computing are promising paradigms aimed at bringing cloud-like functionality at the edge of the network, close to end users and IoT devices, hence complementing the offer of traditional cloud computing, which is based on powerful but distant data centers. A notable, specific thread of research explores so-called *edge-clouds*, i.e., small clouds emerging from the combination of resources of proximate edge-devices, whose goal is to provide on-demand storage and computation power to nearby users. In order to support the engineering of such edge computing ecosystems, in this paper we describe an approach for coordinating resources and computations in edge-clouds that assumes connectivity only to nearby devices (abstracting from the concrete communication technology) and tolerates unreliability by self-adaptation to device failure, mobility and withdrawal. Most notably, we delineate a decentralized, self-organizing, spatial approach that works by dynamically partitioning the system into areas, each one governed by an elected manager, and setting up downstream and upstream coordination flows from managers to peripheral nodes (i.e., workers and users) and vice versa. We provide an implementation schema in the SCAFI framework for aggregate programming and evaluate a basic request scheduling scenario through simulation.**

***Index Terms*—edge computing, self-organization, decentralized computing, spatial computing, distributed coordination**

I. INTRODUCTION

Recently, the Cloud Computing paradigm [1] has become mainstream, reliably providing elastic, virtually unlimited, on-demand resources (i.e., services, processing power, storage) to users through Internet connectivity and large data-centers sparse around the globe. Together with the progress in data center management, early efforts were directed at exploiting one such model for disparate scenarios, leading to fields such as *Mobile Cloud Computing (MCC)* [2]. In MCC, the goal is to give a support to mobile devices for *offloading* data and computations to the cloud [3]. However, such an approach requires communicating with distant data centers, leading to high latency, energy, and bandwidth consumption, and additional load on mobile networks. In order to solve these issues, the idea is to bring cloud-like functionality closer to where resources are needed. Several concepts (with similar or overlapping meanings) emerged in this direction, including *cyber foraging* [4], *cloudlets* [5], as well as *ad-hoc* [6], *peer-to-peer* [7], and *mobile edge-clouds* [8]. These efforts have been somewhat subsumed by *fog* [9], [10] and *(mobile) edge computing* [11], [12] paradigms, whose goal is indeed to support computation at the edge of the network.

Fog and Mobile Edge Computing (FMEC) leverage dense geographical deployments of resource providers and infrastructural elements, as well as the corresponding proximity to users, to both enable new scenarios and improve the efficiency of the system itself and the Quality of Service (QoS) of applications running atop. Indeed, FMEC is not to be intended as a replacement for traditional Cloud Computing but rather as a *complementary paradigm*, providing options to system designers when, e.g.: the cloud is (temporarily or permanently) not available; the cloud is available but undesirable or incompatible with cost, latency or other non-functional requirements; or when the kind of services to be provided operate inherently at the edge (cf., mobile crowdsensing [13]). Accordingly, FMEC represents a facilitator for Internet of Things (IoT) and smart city applications [14], where humans, intelligence, and control are largely in the edge (*edge-centrism* [15]), locality plays a fundamental role, and global connectivity and centralization fall short. However, to realize the FMEC vision, multiple challenges need to be addressed, including programmability, mobility support, resource management, service management, adaptivity, reliability, as well as security and privacy [12], [16].

In this paper, we focus on the problem of *decentralized coordination of edge resources and computations in open scenarios*. Most specifically, we describe a flexible design approach for the development of **edge computing *ecosystems*** (see Figure 1). The idea is to leverage decentralized coordination, self-organization, and spatial patterns in order to provide system-level adaptivity and resilience. The system works by dynamically partitioning itself into areas (which can be thought of as edge-clouds) governed by corresponding managers, and setting up downstream and upstream coordination flows from managers to peripheral nodes (i.e., workers and users) and vice versa. One such design finds straightforward implementation in the *Aggregate Computing* paradigm [17], [18]—in which the entire set of participant devices is turned into, and programmed as, a *collective adaptive system (CAS)*, while abstracting from low-level networking issues. So, the contribution of this paper is threefold:

1) we address the problem of decentralized coordination of large-scale, dynamic, situated edge ecosystems;

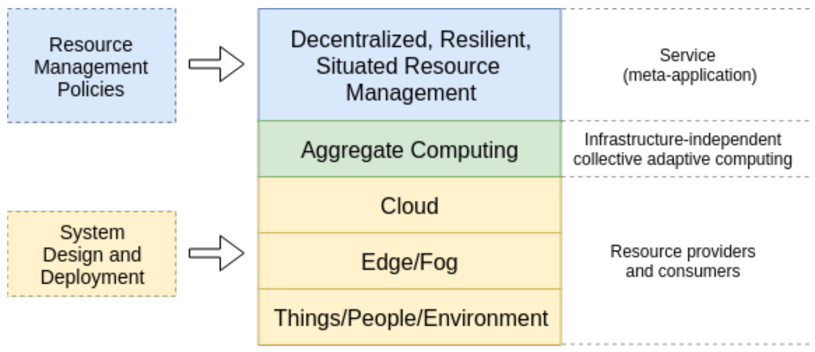

Resource Management Policies
Decentralized, Resilient, Situated Resource Management
Service (meta-application)
Aggregate Computing
Infrastructure-independent collective adaptive computing
System Design and Deployment
Cloud
Edge/Fog
Things/People/Environment
Resource providers and consumers

Fig. 1: Overview of the approach: the layers involved.

2) we describe a collective approach [19] for the engineering of a distributed edge computing platform; and
3) we provide an implementation schema of the coordination logic that leverage the abstraction, compositionality, and execution independence aspects of the Aggregate Computing framework.

Most specifically, our approach only requires connectivity to nearby devices (abstracting from the concrete communication technology), and is designed so as to self-adapt to device failure, mobility and withdrawal.

The remainder of the paper is organized as follows. Section II describes the problem and design context addressed, together with desired qualitative requirements for the solution. Section III presents a model of the solution and discusses the main design dimensions. Section IV presents a solution schema based on Aggregate Computing. Section V evaluates the proposed solution through simulation. Section VI provides a summary of related work. Finally, Section VII provides a wrap-up and outlines future work.

II. Problem Definition

The recurring theme in FMEC is the *smart exploitation* of resources, i.e., the *utilization of idle resources* from devices that were not usually *fully* considered for computational or storage purposes (e.g., networking and end devices), and the *coordination of resources and tasks* to both extend the possibilities of individual components and attain non-functional advantage in system as well as user processes. In this paper, we focus on large-scale scenarios characterized by dense groupings of mobile and heterogeneous devices with diverse computational and networking capabilities. The question addressed is: *how can we expose, manage, and coordinate the resources made available by such computational collective in order to build a scalable, adaptive edge computing platform?*

Our reference scenario is a city, more or less smart (in the sense that it might provide no, little, or much infrastructure), that hosts a large number of computational things and agents (possibly mobile, such as people with mobile phones or wearable devices); then, all these devices may *offer/advertise resources* (e.g., à la *volunteer computing* [20]) or *request resources* to the system—e.g., for task execution. The resource providers and consumers are *situated* entities—i.e., they reside in a environment (which can be perceived and manipulated) and their position is generally relevant since it affects interaction (e.g., who can be contacted, or the cost of communications). Accordingly, we address the problem of building a *system for large-scale, opportunistic, situated resource management and scheduling that spans the thing, edge, and fog layers*. Specifically our goal is to present a design approach that fosters some key properties:

- *Scalability.* The system should be able to scale with the number of devices and the size of geographical deployments (i.e., also with the density of devices). This calls for decentralization in interaction and decision-making.
- *Minimal connectivity requirements.* Devices do not need to be connected to the Internet; we only assume a device is able to send messages within its neighborhood.
- *Minimal infrastructural requirements.* The approach should work seamlessly with or without preexisting infrastructure in place, i.e., it could leverage *mobile ad-hoc networking (MANET)* [21].
- *Opportunism.* The system and its users leverage opportunistic interactions [22] to coordinate and carry out activities.
- *Adaptivity.* The system should be self-adaptive with respect to infrastructural changes as well as perturbations induced by the environment and the autonomous behavior of agents.
- *Openness.* Components can dynamically enter or exit the system in order to participate in it or not.
- *Resilience and graceful degradation.* Permanent or temporary failures of devices and infrastructural elements should not significantly affect the system functionality.
- *Hybrid resource coordination style.* The system should balance centralized and decentralized decision-making for the allocation of tasks to resources.
- *Global strategies and local tactics.* The system should balance between the exploitation of local opportunities and the pursue of global-level benefits.

In other words, we make very few assumptions on connectivity and reliability and rather address dinamicity through adaptivity and opportunistic coordination. Also, we trade off performance for adaptivity, as our focus is not on statically computing an optimal resource allocation, but rather supporting edge computations in highly dynamic scenarios.

III. Approach

In this section, we outline a model of a system for the management of situated resources and the scheduling of activities on top. The approach we propose is:

- **Decentralized.** It models the system such that its functionality and decision-making is not located into a single point, but rather distributed among several components.
- **Self-organizing.** It models the system such that it is itself able to create global structures out of local interactions to sustain its purposeful functions and increase inner order.
- **Spatial.** It models the system such that space and situatedness, rather than low-level networking, play a key role in the accomplishment of design goals.

Figure 2 provides an informal summary of the proposed solution, which is described in the following.

A. Key Entities and Collaborations

We consider an *environment* inhabited by (possibly *mobile*) *devices*. These devices are entities of any sort capable of computation and networking; they may include user devices (e.g., smartphones or other wearables), IoT devices (e.g., smart light poles, traffic lights), edge devices (e.g., gateways, servers, roadside units) and fog devices (e.g., routers, cellular base stations). We assume the network topology is dynamic and not known apriori: these devices can only interact in an opportunistic fashion by exchanging messages with other devices (also called ***neighbors***) located in the vicinity. A device, according to its characteristics and configuration, may play one or more of the following *roles*:

- **Resource providers (aka workers).** These devices offer (a share of their) *resources* to the system and are available for running tasks in a sandboxed environment on the behalf of client devices. They may provide both a share of resources for peer-to-peer negotiation and a share of resources for orchestrated coordination.
- **Resource consumers (aka users, clients).** These devices delegate the execution of *tasks* to the system by requesting appropriate resources. Clients may contact nearby workers directly (e.g., within a 3-hop range) or dispatch a request to their master (whose location may be unknown).
- **Manager nodes (aka leaders).** These devices are responsible for managing (i.e., monitoring and controlling) workers and satisfying requests from clients. They are typically resourceful computers (e.g., edge servers or fog nodes), preferably non-mobile, and located in correspondence of hotspots to ensure wide "coverage" of devices.
- **Relay nodes (aka links).** These devices collectively create a mesh network to ensure there exists a hop-by-hop path from workers and clients to orchestrators. The Link role is just an optimization (with respect to the case in which any device contributes to information spreading) to limit the amount of energy spent in sustaining the system through continuous coordination.

These roles represent a way to support various levels of commitment to the system (for flexibility with respect to local resources and the will to participate) as well as to reason about the different functions that need to be maintained. Notice that all these roles may coexist in a given device.

In order to deal with a huge number of situated devices, we apply the *divide-et-impera* principle: the environment is partitioned into a number of ***(management) areas*** (aka *partitions*, *localities*, *regions*, or *edge-clouds*), each managed by a different master. It comes natural to perform such division *spatially*—which is also coherent with the *locality principle*. All the resource providers and consumers refer to the master of the corresponding partition: they upstream data and requests, and receive control data emitted by the master downstream.

We stress that the system should be able to operate in dynamic environments where devices may move or fail; indeed, users, workers, and even managers and relay nodes might be autonomous with respect to many aspects, and only be required to respect the coordination protocol for their role in order to sustain the self-organizing behavior of the system.

The proposed design can be adapted to find suitable trade-offs between centralized and decentralized coordination and decision-making. For instance, simple tasks may be offloaded from clients to workers without requiring any intermediation from the manager; in such case, the manager could just monitor the activity in its area and interact with other managers to perform meta-coordination—e.g., negotiating resources and dispatching reconfiguration of workers among areas.

B. Design Issues and Dimensions

This paper describes a schema for developing a class of systems, that allows for multiple instantiations, customizations, and extensions. Indeed, the presented model is independent of a number of lower-level issues and mechanisms, and is hence highly configurable. For instance:

- *How are the management areas determined and structured* — Since areas are a notion used to decentralize management activities for scalability purposes, these should be created in order to evenly balance load in the system as well as exploit locality (co-located users may issue similar requests, and co-located workers can interact with low latency). Thus, the management regions should be dynamically created and gracefully adapted by considering the distribution of workers, consumers and managers. Often, managers are determined first, and then regions are negotiated in turn, based on the desired shape of the edge-clouds in terms of exposed resources.
- *How are the managers chosen among eligible devices* — The system should self-organize by finding consensus on leadership among the set of candidates. The point is not the actual algorithm but the properties that need to be enforced and maintained. Generally, the managers should uniformly cover the situated workers and users, but more advanced choices could also consider the trust [23],

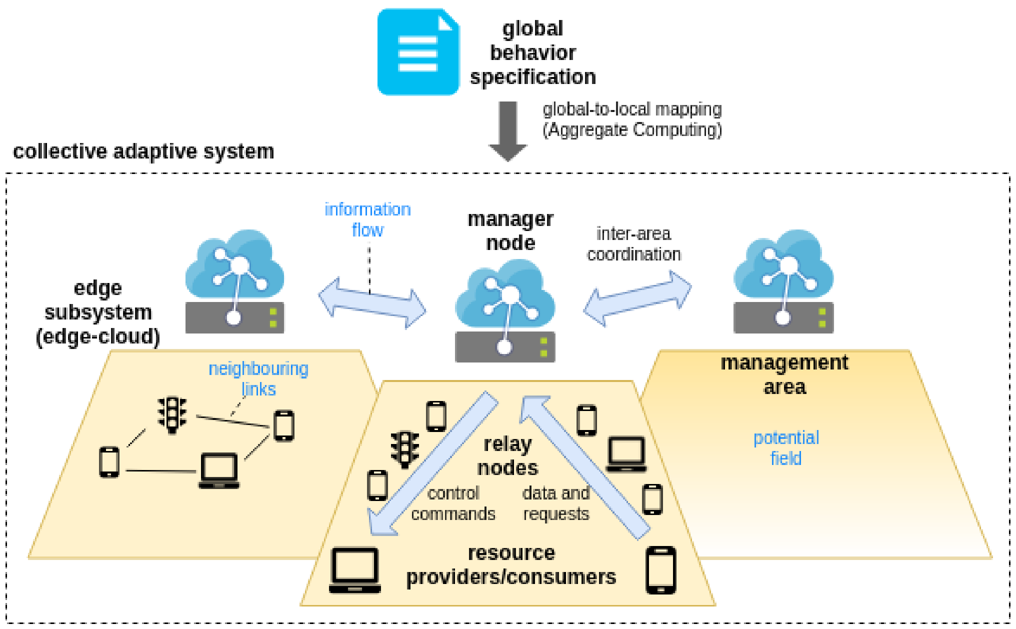

global behavior specification
global-to-local mapping (Aggregate Computing)
collective adaptive system
information flow
manager node
inter-area coordination
edge subsystem (edge-cloud)
neighbouring links
management area
potential field
relay nodes
control commands
data and requests
resource providers/consumers

Fig. 2: Visual overview of the proposed design for self-organizing edge-clouds.

resourcefulness and dependability of manager candidates, or the density and profiles of users and workers.

- *As leaders and areas are dynamically determined, how can workers and users know how to contact the respective manager* — The model abstracts from these details. However, we generally assume every node is at least able to communicate with neighbor devices and there exists a sequence of other devices (i.e., a path of relays) connecting users/workers to leaders. This is a sufficient condition for setting up communication paths (e.g., via patterns like potential/gradient fields [24] unfolding from leaders) to enable self-organizing *information flows* [25].
- *What should happen when a master node fails* — The failure of a master usually invalidates the invariants required from a configuration of masters. In any case, an area loses its leader and, unless that area could dissolve by feeding adjacent areas, a new leader must be elected. While this happens, requests from users and feedback from workers cannot be handled, but they eventually will be as soon as a newly elected master will receive them.
- *What should happen when a link node fails* — This should not have serious consequences, unless this results in permanent network partitioning. The system should self-organize to properly correct the paths followed by information flows; the aforementioned gradient fields could inherently handle this adaptation.
- *What should happen when a worker node fails* — It essentially depends on what kind of guarantees must be provided by the system. In any case, the leader should become aware of such failure (e.g., by requiring a *heart beat* and considering a temporal threshold that may also depend on a volatility metric characterizing the risk for communication delays in the area under supervision). Also, if the assigned task is continuable, intermediate results might be stored in nearby devices or in the master, so that they can be retrieved by a newly allocated worker.

The focus of the approach is on the *coordination* logic for an edge computing platform. However, there are some important concerns (application-specific issues or challenges on their own from which we abstract from) that should be defined and developed to actually design a working ecosystem:

- *Management functions.* These may include monitoring (e.g., resource usage or availability), control (e.g., assignment of tasks), orchestration (i.e., control of coalitions of workers), or choreography (e.g., managers may group workers into “teams” for cluster computing, and let these work autonomously). Such aspects may affect the logical structure of the system, e.g., whether masters directly communicate with users or always interface with workers.
- *Task scheduling strategies.* Assigning tasks to workers requires to solve issues like task placement and partitioning [26]. It is reasonable to centralize such issues in manager nodes, which have a global view of the corresponding areas. For optimizing choices, the designers should consider what information needs to be collected into those decision points (e.g., QoS preferences and requirements, locations, accurate task models, and so on).
- *Sociality and economics.* The model is configurable with respect to the aspects related to consumption and offering of resources. E.g., in the vision of social clouds [27], dis/incentives can be used to regulate sharing, trading, and interactions through socially corrective mechanisms.
- *System structure and environment.* Particular applications could impose more or less constraints on the physical and logical structure of the system. Specific decisions should be taken regarding how leaders are elected, how areas are

determined, the concrete shape of components, their number and requirements, the assumptions on infrastructure (e.g., WAPs) and so on.

IV. Implementation

In this section, we provide an implementation schema for the coordination model described in Section III; it leverages the aggregate computing framework ScaFi [28] in order to reach the intended properties of the outlined solution (Table I).

A. Background: Aggregate Computing

Aggregate Computing [17] is a paradigm for programming CASs by a global perspective. It builds on the *field calculus* [18], [29]—a core, universal, functional language for compositionally expressing space-time computations and emergent behaviors. A *computational field* is just a map from devices to values. Fields changing over time are created through expressions interpreted continuously by a system of devices. This theory of computational fields formally draws a bidirectional link from local and global behaviors. On top of the field calculus, full-fledged aggregate programming languages, such as ScaFi [28], can be developed, offering application developers high-level abstractions and macro building blocks for the engineering of self-organizing systems. The programming activity consists in reusing and defining functions that ultimately manipulate whole fields by composition, restriction, observation, and evolution operations.

In this framework, an *aggregate program* is a field computation locally interpreted by a network of devices connected through a neighboring relationship. An aggregate program is repeatedly executed in its entirety in a round-by-round fashion; i.e., any device continuously (i) builds a context by sensing its environment and communicating with neighbors, and (ii) interpret the aggregate logic against that context to determine the local actions and communication acts that need to be performed to cooperatively sustain the global behavior.

A full introduction of Aggregate Computing is beyond the scope of this paper. The reader interested in the details and the various directions of such paradigm can refer, e.g., to [17], [18], [28], [29]. However, the reader should keep in mind that:

- *Aggregate Computing is declarative.* Its abstraction provides a lot of flexibility concerning the actual deployment and execution steps of aggregate systems, envisioning platforms enacting optimized, dynamic execution plans in environments comprising edge/fog/cloud layers [30].
- *Aggregate Computing has formal foundations.* The field calculus formalizes the link between micro- and macro-levels of collective adaptation. Related research has identified interesting properties and guarantees; e.g., a key result relates to *self-stabilization* (i.e., the property of a system to eventually converge to a consistent state), which is proved to be preserved by any composition obtained by a proper subset of the field calculus [31].
- *Aggregate Computing is practical.* There exist languages and tools for system development. In this paper, we use ScaFi, a toolkit that includes (i) a Scala-internal DSL for programming with fields, (ii) simulation modules, and (iii) an actor-based middleware for distributed support.

B. Aggregate Coordination Logic: Implementation Schema

In order to implement the solution described in Section III, we consider the underlying system of devices (including fog nodes, edge devices, and user equipment) as a *heterogeneous aggregate* exhibiting global-level behaviors: i.e., we program it into a CAS that self-organizes so as to realize its functionality in face of change and failure. In pratice, every device must "continuously" interpret the aggregate program of Figure 3 and respect the aggregate coordination protocol (i.e., sharing coordination data with neighbors).

In the specification, the instructions are "global", i.e., they model the behavior not of a single device but of a *collective* of devices; however, it is possible to execute a computation on a restricted domain of devices through *branching* constructs. `S` is an aggregate building block for self-organizing leader election, where leaders are elected so that they are at a mean distance `grain`; such a process is only carried out by powerful fog nodes (through a branch on `FOG` boolean field holding true in fog nodes and false elsewhere). Function `distanceTo` basically represents a collective computation of a "gradient" field [32], i.e., a field where any device is mapped to its minimum distance from "source" nodes; applied to the leader field, this yields a potential attracting nodes to the corresponding leader: the effect is the creation of the "supervision areas" around leaders. At this point, the *feedback/control loop* can be created through *information flows* going from peripheral nodes to leaders (*upstream*) and vice versa (*downstream*). In aggregate programming, such information and processing flows can be implemented through blocks, called `G` and `C`, that generalize gradients and collection over spanning trees [18], respectively. The `rep(init){fun}` construct is used to keep memory of the `DownstreamData` field for the subsequent round, by mapping its previous value (`init` the first round) through the provided `fun`ction, so that it can be used as input for function `execute`. Notice that adaptivity and resilience follow directly from the computational field model. A field expression, by describing a field computation, may define a field that changes over time and depends on other fields. So, as the input fields change, the output field will adjust as a consequence according to the expressed input-output relationship. For instance, as mutations occur in the `leaders` field (e.g., due to a failure in a leader), the potential field sustaining the communication structure of the system will self-heal to correct the direction of information flows. In summary, self-organization stems from the ability of the given aggregate behaviour of restoring order after chaos possibly fed by failure, mobility, environment change, and conflicting autonomy.

V. Evaluation

The approach described in Section III is empirically evaluated through synthetic experiments built on the ScaFi-Alchemist simulation framework [28]. In particular, we complete the core implementation schema provided in Section IV

Property	Solution
Scalability	● Partition into areas covering a subset of resource consumers/providers ● Decentralized coordination with neighbors
Connectivity requirements/ Infrastructural requirements	● Only local, short-range connectivity is assumed ● Independence from the concrete communication technology
Opportunism	● Coordination works through opportunistic interactions with nearby devices
Adaptivity/Resilience	● Continuous sensing, interaction, and actuation ● Self-organization
Openness	● Participation only requires executing the program/protocol
Hybrid coordination	● Centralized coordination is partitioned & built on decentralized coordination ● Flexibility on what responsibilities are assigned to managers or workers

TABLE I: Adopted solutions for the desired properties discussed in Section II.

```
class EdgeCloudCoordinationWorkflow extends AggregateProgram with ... {
  // Some definitions (excluded for brevity)

  def main = { // Entry point of the specification, to be interpreted in full every round
    // 1) Elect managers among powerful "fog" nodes; output is a bool field, 'true' in current leaders
    val leaders = branch(FOG){ S(grain) }{ false }

    branchOn(EDGE || FOG) {
      // 2) Build the adaptive communication structure,
      //    based on a potential field pointing to leaders for information down-/up-streaming
      val potential = branch(leaders || RELAY) { distanceTo(leaders) } { +∞ }
      val cs = CommunicationStructure(leaders, potential)

      // 3) Sets up a "continuous" feedback control loop base
      rep(DownstreamData.empty){ case dFlow =>
        val data = branchOn(isWorker || isConsumer){ execute(dFlow) }
        val uFlow = dataUpstreaming(cs, data)
        val controlData = branchOn(leaders){ processData(uFlow) }
        dataDownstreaming(cs, controlData)
  } } }

  // Workers/clients receive instructions/events from leaders and produce data
  def execute(dd: DownstreamData): Data
  // Data/events produced by workers/clients has to be aggregated and streamed to leaders
  def dataUpstreaming(cs: CommunicationStructure, data: Data): UpstreamData
  // Leaders process upstream data/events and produce control instructions/events
  def processData(ud: UpstreamData): ControlData
  // Control instructions/events or area-wide information is sent around the area
  def dataDownstreaming(cs: CommunicationStructure, cd: ControlData): DownstreamData
}
```

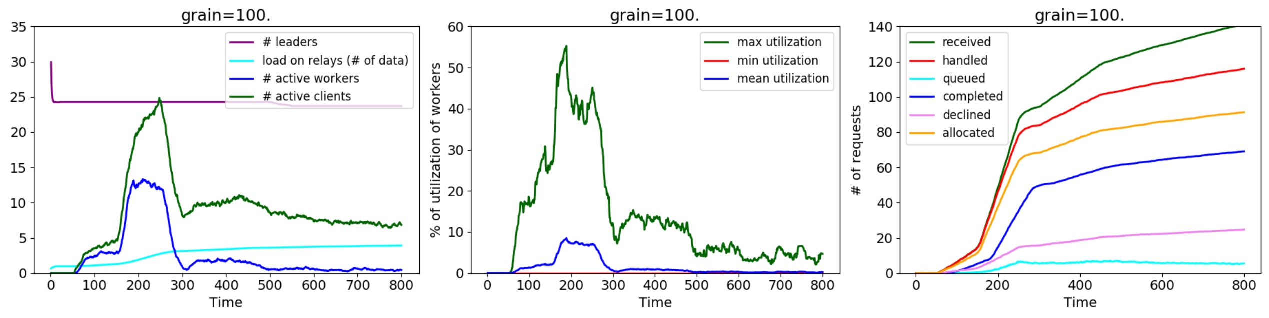
Fig. 3: Core aggregate implementation schema for an edge computing ecosystem: symbols in bold black and gray are methods and types, resp., to be implemented for application-specific functionality (e.g., using Aggregate building blocks as per [28]); red and purple symbols denote primitive and derived field constructs; blue symbols are Scala keywords.

with methods and types to support a service and request management functionality, where (i) workers advertise the services they provide; (ii) consumers can send requests for services to the area manager; and (iii) the area manager handles requests by allocating them to workers or declining them if no worker in the area supports the requested service. The source code of the simulations, launch scripts, and additional information are available at the accompanying repository[1].

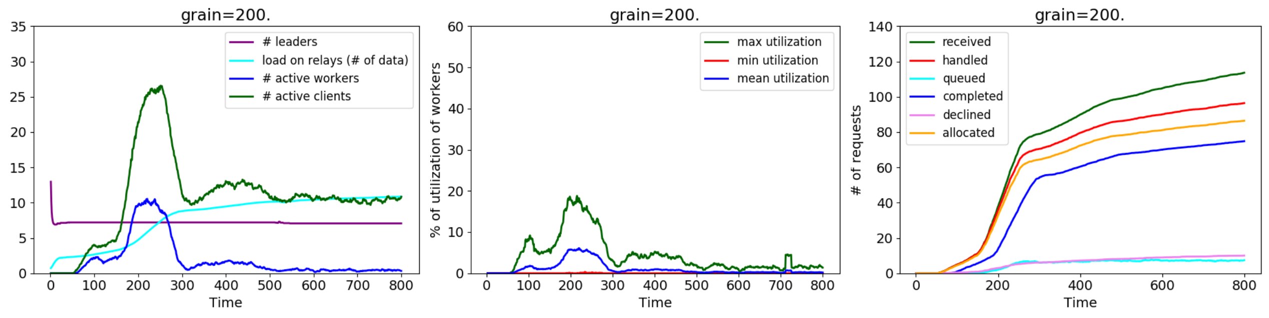
Setup: We assume to be in a (smart) city. Our system consists of 50 fog nodes irregularly covering the urban area, 200 worker nodes supporting zero or more services (one of them being more rare than the others) and also working as relays, and 500 clients unevenly distributed in the city. The sleeping period between execution rounds in each device is about 1 second. The fog nodes can interact with one another, whereas clients and workers are connected only with proximate devices (50 metres range). Over time, the clients request services, with a peak in the time frame $[150, 250]$. Multiple simulation configurations are considered, by varying the granularity of the areas. We launch 30 simulation runs with different random seeds for each configuration.

Results and discussion: Our goal is to show functional correctness and basic performance. Figure 4 shows some key metrics of the system, for different partitioning granularities. With respect to more coarse-grained partitioning, in the case of many, smaller areas (Figure 4a), we observe a minor load on relays (as fewer events and less worker/consumer data have

[1]https://github.com/metaphori/fmec19-edgecloud



(a) System performance with many, small areas.



(b) System performance with few, large areas.

Fig. 4: Evaluation graphs.

to be propagated), but a higher risk of being saturated (i.e., reaching 100% utilization of worker resources) as well as higher reject rate (since fewer workers and a minor variety of resources will be available). By contrast, larger areas can satisfy more requests (since they can generally count on more kinds of resources), and have greater capacity for dealing with localized spikes of activity. However, larger areas also means that higher load is put on managers and relay nodes.

VI. Related Work

Cluster management platforms: Systems like Mesos [33] and YARN [34] provide an abstraction layer over the physical resources of a cluster, exposing a "distributed virtual machine" for job execution. They usually adopt a master-worker architecture. In a Mesos cluster, a *Mesos master*, elected from a set of candidates using ZooKeeper, receives *resource advertisements* from its set of supervised *agents*, and sends *resource offers* to user *frameworks*, which are responsible for their own task scheduling on assigned resources. The presented approach shares similarities with Mesos, but it focuses on dynamic edge environments, considers multiple regions, and leverages self-organization and data streams for "continuous coordination". Also, unlike cluster computing, nodes do not need to know each other as communication happens opportunistically.

Mobile edge-clouds, peer-to-peer clouds, and volunteer computing: The idea of aggregating edge devices to build nearby "edge-clouds" is not new. The research panorama is various, with approaches differing by assumptions or the nature of proposed solutions. A category includes approaches where devices can offer their own resources to the system, forming so-called *volunteer clouds*. Nebula [35] is a cloud service that uses volunteer edge resources for geographically distributed data-intensive computing. It has a web front-end node for users to join the system as volunteers or executing applications. Each Nebula application has a master node for handling computations and a master node for dealing with data storage, each controlling a group of volunteer nodes. Related to volunteer clouds is the notion of *community clouds* [36], proposed as an extension of community networks [37].

Another category (related to volunteer computing) considers fully decentralized, *peer-to-peer clouds* [7]. In this context, focus is largely on algorithms enabling direct offloading of computations to nearby devices. For instance, Honeybee [8] is a work-sharing model for independent jobs that address dynamism through opportunistic computing. Mycocloud [38] is another algorithm – bio-inspired, self-organizing – for service placement in decentralized clouds. With respect to the aforementioned efforts, the present paper describes a pattern for implementing self-organizing mobile edge-clouds (possibly above fully peer-to-peer networks) in unknown, loosely connected, and changing environments—with a balanced trade-off between centralization and decentralization. In particular, it provides a coordination schema where custom task allocation strategies can be injected and performed on a regional basis.

Our work is also inspired by and leverages Aggregate Computing for creating a self-organizing edge-cloud that exploits locality and decentralization for large-scale coordination of edge computations. Indeed, a relevant contribution of this paper also lies in showing how edge ecosystems may be concretely *designed and programmed* adopting novel paradigms. Research in Aggregate Computing has mainly focused on application scenarios such as crowd management [17], [22], distributed sensing [39], and management of unmanned vehicle swarms [40]. Despite such inherent orientation towards edge computing scenarios, this is the first work that explicitly adopts aggregate techniques for FMEC.

VII. Conclusion and Future Work

In this paper, we have proposed a decentralized, self-organizing, spatial approach for coordinating services and computations at the edge. It is like a pattern that can be refined and extended to design various strategies and tactics for the management of peer-to-peer edge-clouds. We also provided an aggregate implementation schema and evaluated it through simulations. As future work, we would like to provide an extended quantitative analysis as well as exploring the technique in more realistic case studies.

References

[1] P. Mell, T. Grance *et al.*, "The NIST definition of cloud computing," 2011.

[2] N. Fernando, S. W. Loke, and W. Rahayu, "Mobile cloud computing: A survey," *Future Generation Computer Systems*, vol. 29, no. 1, pp. 84–106, 2013.

[3] P. Mach and Z. Becvar, "Mobile edge computing: A survey on architecture and computation offloading," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 3, pp. 1628–1656, 2017.

[4] M. Sharifi, S. Kafaie, and O. Kashefi, "A survey and taxonomy of cyber foraging of mobile devices," *IEEE Communications Surveys & Tutorials*, vol. 14, no. 4, pp. 1232–1243, 2012.

[5] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The case for vm-based cloudlets in mobile computing," *IEEE Pervasive Computing*, vol. 8, no. 4, 2009.

[6] I. Yaqoob, E. Ahmed, A. Gani, S. Mokhtar, M. Imran, and S. Guizani, "Mobile ad hoc cloud: A survey," *Wireless Communications and Mobile Computing*, vol. 16, no. 16, pp. 2572–2589, 2016.

[7] O. Babaoglu, M. Marzolla, and M. Tamburini, "Design and implementation of a P2P cloud system," in *Proceedings of the 27th Annual ACM Symposium on Applied Computing*. ACM, 2012, pp. 412–417.

[8] N. Fernando, S. W. Loke, and W. Rahayu, "Computing with nearby mobile devices: a work sharing algorithm for mobile edge-clouds," *IEEE Transactions on Cloud Computing*, no. 1, pp. 1–1, 2016.

[9] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*. ACM, 2012, pp. 13–16.

[10] L. M. Vaquero and L. Rodero-Merino, "Finding your way in the fog: Towards a comprehensive definition of fog computing," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 5, pp. 27–32, 2014.

[11] Y. C. Hu, M. Patel, D. Sabella, N. Sprecher, and V. Young, "Mobile edge computinga key technology towards 5g," *ETSI white paper*, vol. 11, no. 11, pp. 1–16, 2015.

[12] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, 2016.

[13] R. K. Ganti, F. Ye, and H. Lei, "Mobile crowdsensing: current state and future challenges," *IEEE Communications Magazine*, vol. 49, no. 11, 2011.

[14] A. V. Dastjerdi and R. Buyya, "Fog computing: Helping the internet of things realize its potential," *IEEE Computer*, vol. 49, no. 8, pp. 112–116, 2016.

[15] P. Garcia Lopez, A. Montresor, D. Epema, A. Datta, T. Higashino *et al.*, "Edge-centric computing: Vision and challenges," *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 5, pp. 37–42, 2015.

[16] R. Roman, J. Lopez, and M. Mambo, "Mobile edge computing, fog et al.: A survey and analysis of security threats and challenges," *Future Generation Computer Systems*, vol. 78, pp. 680–698, 2018.

[17] J. Beal, D. Pianini, and M. Viroli, "Aggregate programming for the internet of things," *IEEE Computer*, 2015.

[18] M. Viroli, J. Beal, F. Damiani, G. Audrito, R. Casadei, and D. Pianini, "From field-based coordination to aggregate computing," in *Int. Conf. on Coordination Languages and Models*. Springer, 2018, pp. 252–279.

[19] R. Casadei and M. Viroli, "Collective abstractions and platforms for large-scale self-adaptive IoT," in *IEEE 3rd FAS* Workshops*, Sep. 2018, pp. 106–111.

[20] D. P. Anderson, "Volunteer computing: the ultimate cloud." *ACM Crossroads*, vol. 16, no. 3, pp. 7–10, 2010.

[21] P. Bellavista, G. Cardone, A. Corradi, and L. Foschini, "Convergence of MANET and WSN in IoT urban scenarios," *IEEE Sensors Journal*, vol. 13, no. 10, pp. 3558–3567, 2013.

[22] R. Casadei, G. Fortino, D. Pianini, W. Russo, C. Savaglio, and M. Viroli, "Modelling and simulation of opportunistic iot services with aggregate computing," *Future Generation Computer Systems*, 2018.

[23] R. Casadei, A. Aldini, and M. Viroli, "Combining trust and aggregate computing," in *International Conference on Software Engineering and Formal Methods*. Springer, 2017, pp. 507–522.

[24] T. De Wolf and T. Holvoet, "Design patterns for decentralised coordination in self-organising emergent systems," in *Workshop on Engineering Self-Organising Applications*. Springer, 2006, pp. 28–49.

[25] ——, "Designing self-organising emergent systems based on information flows and feedback-loops," in *Self-Adaptive and Self-Organizing Systems, 2007. SASO'07. 1st Conf. on*. IEEE, 2007, pp. 295–298.

[26] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 4, pp. 2322–2358, 2017.

[27] K. Chard, K. Bubendorfer, S. Caton, and O. F. Rana, "Social cloud computing: A vision for socially motivated resource sharing," *IEEE Transactions on Services Computing*, vol. 5, no. 4, pp. 551–563, 2012.

[28] R. Casadei, D. Pianini, and M. Viroli, "Simulating large-scale aggregate MASs with Alchemist and Scala," in *FedCSIS, Proceedings of*. IEEE, 2016, pp. 1495–1504.

[29] G. Audrito, M. Viroli, F. Damiani, D. Pianini, and J. Beal, "A higher-order calculus of computational fields," *ACM Transactions on Computational Logic*, vol. 20, no. 1, pp. 5:1–5:55, Jan. 2019.

[30] M. Viroli, R. Casadei, and D. Pianini, "On execution platforms for large-scale aggregate computing," in *UbiComp, Proceedings of*. ACM, 2016, pp. 1321–1326.

[31] M. Viroli, G. Audrito, J. Beal, F. Damiani, and D. Pianini, "Engineering resilient collective adaptive systems by self-stabilisation," *ACM Transactions on Modeling and Computer Simulation*, vol. 28, no. 2, pp. 16:1–16:28, 2018.

[32] G. Audrito, R. Casadei, F. Damiani, and M. Viroli, "Compositional blocks for optimal self-healing gradients," in *Conf. on Self-Adaptive and Self-Organizing Systems (SASO)*. IEEE, 2017, pp. 91–100.

[33] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph *et al.*, "Mesos: A platform for fine-grained resource sharing in the data center," in *Proceedings of the 8th USENIX Conf. on Networked Systems Design and Implementation*, ser. NSDI'11, 2011, pp. 295–308.

[34] V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar *et al.*, "Apache hadoop yarn: Yet another resource negotiator," in *Proceedings of the 4th annual Symposium on Cloud Computing*. ACM, 2013, p. 5.

[35] A. Chandra, J. Weissman, and B. Heintz, "Decentralized edge clouds," *IEEE Internet Computing*, no. 5, pp. 70–73, 2013.

[36] A. M. Khan, L. Navarro, L. Sharifi, and L. Veiga, "Clouds of small things: Provisioning infrastructure-as-a-service from within community networks," in *Wireless and Mobile Computing, Networking and Communications (WiMob), 9th Int. Conf. on*. IEEE, 2013, pp. 16–21.

[37] J. Jiménez, R. Baig, P. Escrich, A. M. Khan, F. Freitag *et al.*, "Supporting cloud deployment in the guifi.net community network," in *Global Information Infrastructure Symposium, 2013*. IEEE, 2013, pp. 1–3.

[38] D. Dubois, G. Valetto, D. Lucia, and E. Di Nitto, "Mycocloud: Elasticity through self-organized service placement in decentralized clouds," in *Int. Conf. on Cloud Computing (CLOUD)*. IEEE, 2015, pp. 629–636.

[39] R. Casadei and M. Viroli, "Programming actor-based collective adaptive systems," in *Programming with Actors: State-of-the-Art and Research Perspectives*, 2018, vol. 10789, pp. 94–122.

[40] J. Beal, K. Usbeck, J. Loyall, M. Rowe, and J. Metzler, "Adaptive opportunistic airborne sensor sharing," *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, vol. 13, no. 1, p. 6, 2018.