SeMaFoR - Self-Management of Fog Resources with Collaborative Decentralized Controllers

Abdelghani Alidra, Hugo Bruneliere, Hélène Coullon, Thomas Ledoux, Charles Prud'Homme, Jonathan Lejeune, Pierre Sens, Julien Sopena, Jonathan Rivalan

SeMaFoR - Self-Management of Fog Resources with Collaborative Decentralized Controllers

Abdelghani Alidra, Hugo Bruneliere, Hélène Coullon, Thomas Ledoux, Charles Prud'homme
IMT Atlantique, LS2N (UMR CNRS 6004)
Nantes, France
firstname.lastname@imt-atlantique.fr

Jonathan Lejeune, Pierre Sens, Julien Sopena
Sorbonne Université, Inria
Paris, France
firstname.lastname@lip6.fr

Jonathan Rivalan
Smile
Asnières sur Seine, France
jonathan.rivalan@smile.fr

***Abstract*—Fog Computing is a paradigm aiming to decentralize the Cloud by geographically distributing away computation, storage and network resources as well as related services. This notably reduces bottlenecks and data movement. However, managing Fog resources is a major challenge because the targeted systems are large, geographically distributed, unreliable and very dynamic. Cloud systems are generally managed via centralized autonomic controllers automatically optimizing both application QoS and resource usage. To leverage the self-management of Fog resources, we propose to orchestrate a fleet of autonomic controllers in a decentralized manner, each with a local view of its own resources. In this paper, we present our SeMaFoR (Self-Management of Fog Resources) vision that aims at collaboratively operating Fog resources. SeMaFoR is a generic approach made of three cornerstones: an Architecture Description Language for the Fog, a collaborative and consensual decision-making process, and an automatic coordination mechanism for reconfiguration.**

***Index Terms*—Fog Computing, Architecture Description Language, Autonomic Controllers, Decision-making**

I. Introduction

Fog Computing [1]–[3] is a paradigm aiming to decentralize the Cloud by geographically distributing away computation, storage and network resources, and related services. Instead of considering a huge centralized Cloud system, both data centers of various sizes in the core network and smaller data centers or devices at the edge of the network can be collaboratively used to form a single *large-scale geo-distributed system*. Thanks to resource locality, this paradigm allows better performance in terms of service latency, power consumption, network traffic or content distribution. The main gains are twofold: 1) Avoid network bottlenecks and single points of failure; 2) Keep the data as close as possible to their sources (*e.g.*, sensors) and to their final usage (*i.e.*, end-users). However, managing resources in such heterogeneous and dynamic large systems is highly challenging and requires fully decentralized solutions for scalability and reliability reasons. Notably, in Fog Computing, network interruptions and faults are very frequent [4][1].

Centralized Cloud systems are often designed with a set of centralized *autonomic controllers*[2] [5], [6] in order to automatically manage (*i.e.*, without human intervention) different levels of Cloud administration such as applications Quality of Service (QoS) or virtual resources optimization, etc. However, in Fog Computing, the system is much larger (*i.e.*, number of resources and services), more heterogeneous (*i.e.*, variety of hardware), unreliable and highly dynamic (*e.g.*, faults, mobility of devices). This prevents from building a consistent centralized view to take control decisions. Thus, to automatically operate Fog systems, distributed solutions are needed to coordinate (*i.e.*, orchestrate) a possibly large number of small autonomic controllers, each one having a local view of their controllable resources, *i.e.*, of their respective *Fog area*. In order to address a potentially wide range of large-scale Fog systems, the concept of Fog area must be generic enough to represent a *logical* partition of the Fog configuration such as a delay-aware network community, a trusted data zone, a hardware-specific subset, etc.

In this paper, we present our SeMaFoR (Self-Management of Fog Resources) vision towards a generic, end-to-end, decentralized and collaborative self-management solution to operate different kinds of Fog systems. To this end, we aim at addressing the following challenges:

1) Provide a generic and customizable *Architecture Description Language (ADL)* for modeling any kind of Fog system (*i.e.*, infrastructure topologies, resources configurations) and their specific features such as the locality concept, QoS constraints applied on resources (*e.g.*, energy, data privacy, latency) and their dependencies, the dynamicity of considered workloads, the heterogeneity of both applications and devices, etc.
2) Support *collaborative decision-making* between a fleet of small autonomic controllers distributed over the Fog. We think that tackling the convergence of local decisions to obtain a shared consistent decision among these autonomic controllers requires new distributed agreement protocols based on distributed consensus algorithms.
3) Support the *automatic generation and coordination of reconfiguration plans* between the autonomic controllers. The controller gets a new local target configuration to apply from the consensus, but the execution plan of the overall reconfiguration also needs to be generated and coordinated to minimize the disruption time and avoid reaching errors or inconsistent states.

For validation purposes, and with the support of the Smile company, we plan to design and implement a concrete implementation of the SeMaFoR approach that can be used for managing large-scale distributed systems composed of different Fog areas. Notably, we plan to evaluate the relevance and

[1] https://www.microsoft.com/en-us/research/publication/the-emerging-landscape-of-edge-computing/

[2] In the remainder of the paper, autonomic controllers are also called autonomic managers or MAPE-K loops.

efficiency of the SeMaFoR solution on both simulated Fog infrastructures (simulating practical examples of Fog systems) and real infrastructures (*e.g.*, the French experimental platform Grid'5000[3], or commercial ones such as Amazon EC2).

The rest of this document is organized as follows. Section II presents the background of and motivation for our SeMaFoR vision. Section III introduces the overall SeMaFoR approach and describes each step of the approach by using a running example. Finally, Section IV discusses open challenges to be tackled in SeMaFoR or in other research efforts.

II. Motivation and background

A. Motivation and approach

The main objective of SeMaFoR is to go beyond existing autonomic Cloud approaches [6], [7] and towards emerging large-scale and geographically-distributed Fog environments.

In the past, we have addressed the challenge of coordinating multiple autonomic managers in the Cloud between the SaaS and the IaaS layers [8] in a ad hoc manner. Then, we proposed a generic model-based architecture for autonomic management of Cloud systems [6] where any XaaS (Anything as a Service) layer can be defined (e.g., energy as a service). However, our approach has been specific to client-server relationship, with a set of centralized autonomic controllers and does not address large-scale geo-distributed system with its different issues (e.g., heterogeneity, unreliability, highly dynamicity).

Going further, we now propose an end-to-end approach to specify, (re)configure and orchestrate Fog resources in a generic and automated way. We advocate for a generic approach to address a wide range of topologies, different use cases and various reconfiguration scenarios such as, for instance:

- Dynamically and continuously migrate, scale, and load balance some services from one Fog area to another to ensure a constant Quality of Service despite user mobility;
- Reconfigure some applications, services, or even the controllers themselves after faults;
- Support the geographically distributed deployment of new multi-region applications in the Fog system;
- Reconfigure resources usage to reduce the overall energy consumption, or dynamically and continuously migrate services to follow solar energy within the system, etc.

A Fog system is made of a set of various resources – from hardware ones (*e.g.*, IoT devices, servers or any data center distributed over the network) to software ones (*e.g.*, virtual machines, micro-services) – potentially interconnected (*e.g.*, network connections, dependencies between resources). As introduced in Section I, our assumption is not to have a centralized view of the overall Fog system, nor to take global control decisions, but rather to consider *Fog areas* with controllers handling a local subset of interconnected resources.

To preserve the elasticity of the Cloud paradigm in the Fog, collaboration is required between Fog areas when one (or more) autonomic controller is not able to find locally a satisfying configuration (*e.g.*, due to some services that cannot be physically hosted, or QoS expectations) and thus needs the resources from other Fog areas.

[3]https://www.grid5000.fr

B. Fog and Autonomic Computing

Decentralized self-management in Fog Computing is a recent research area and, to the best of our knowledge, the related literature is currently limited. To realize the Fog Computing paradigm, several approaches have been proposed [1], [3], [9]. Some works propose a hierarchical architecture of Fog Computing, while others advocate a more decentralized approach relying on peer-to-peer (P2P) networking at the edge. Some approaches emphasize on the difference between application-agnostic and specific Fog architectures, while others highlight multi-tenancy or data flow as essential features. However, only a few works address the self-management of Fog resources [10], [11] and none of them proposes a generic approach for distributed and collaborative decision making. For example, a recent work studies the impact of different decentralization and coordination schemes [12] but, contrary to us, does not intend to propose an end-to-end approach.

In Autonomic Computing, the decentralization of self-adaptive systems has been largely studied during the last decade [13]–[16]. It has been shown that the realization of the coordination between several MAPE-K loops follows a set of design patterns [13]. Based on these, we decided to adopt the *Coordinated Control Pattern* in the SeMaFoR approach. Indeed, each control loop must coordinate with its peers (*i.e.*, other Fog areas) to reach some joint decision about how to adapt to a particular situation. More recently, a mapping study [15] listed open challenges for future work on decentralized self-adaptive systems. It appears that only two studies worked on the need for a disciplined engineering approach to realize more easily decentralized self-management. Moving forward in this direction, SeMaFoR plans to provide a generic, integrated and end-to-end approach allowing Fog architects to build decentralized and collaborative autonomic Fog systems.

Finally, specific decentralized, hierarchical, or federated versions of Kubernetes (*e.g.*, KubeEdge, mck8s [17], Admiralty [18], KubeFed [19], me-kube [20]) and other decentralized orchestrators [21], [22] have been proposed to address the management of container-based Edge/Fog resources. However, most of these solutions either decentralize a subpart of the control, or adopt a federated or hierarchical approach. We rather intend to reach a fully decentralized solution.

C. Architecture Description Languages and Fog Computing modeling

Architecture Description Languages (ADLs) and modeling languages have been widely used since a long time, including in the context of distributed systems and Cloud Computing [23]. Among others, a standard called TOSCA [24] for modeling portable Cloud applications and supporting their life-cycle management is getting more attention. With the emergence of Fog Computing, the research community then started to work on the definition of Fog common architectures and underlying concepts [25]. Nevertheless, according to our detailed study of the state-of-the-art in terms of already existing Fog Modeling Languages [26], none of the available solutions appears to come with a generic reusable language that would allow engineers to easily specify, share, maintain and evolve Fog system's models supporting different kinds of Fog architectures. We intend to fill this gap in SeMaFoR.

To this end, we already worked in the past on a model-based approach for heterogeneous Cloud systems [6]: we notably proposed a corresponding Cloud modeling language that can interoperate with TOSCA [27]. More recently, we also integrated various languages and related models of large and heterogeneous systems [28]. Based on these previous efforts, we now go further by adapting the core of our Cloud modeling language and refining it in order to also integrate more specific concepts and properties related to Fog systems.

D. Collaborative decision-making

In the context of SeMaFoR, the decision problem must be considered through two dimensions: 1) The *intra-controller* dimension, specific to a controller that has to manage resources and services within its logical Fog area; 2) The *inter-controller* dimension, wherein two or more controllers must reach a stable point, i.e., a fixed point, between them. The controllers involved define a *neighbourhood* and a controller can be in several different neighbourhoods. Finding a fixed point within a neighbourhood is intrinsically decentralized. Indeed, a controller has to take decisions with partial knowledge on an unreliable and dynamic environment, in a competitive way. Thus, there exist links between the Fog area controllers and these links imply that some of them share common resources, or *variables*. Consequently a logical common variable is replicated physically on all concerned controllers. Then, a collaborative decision making about its value has to be done consensually. This matches the well-known fundamental *consensus problem* [4] [29], [30] of distributed systems. It specifies that all non-faulty nodes will decide definitively (integrity property) in a finite time a common value among a set of proposed values. However implementing such autonomic controller federation leads to multiple decisions taken asynchronously by several subsets of nodes. Therefore, potential decision conflict between two agreements may happen on nodes involved in several agreements due to the system's dynamics. In this case, a previous decision can be canceled to resolve conflicts that violate the integrity property. This questions the original definition of the consensus problem and the existing distributed algorithms.

There are multiple ways to solve the decision problem a controller has to deal with, known as mathematical programming techniques. In [6], we proposed a generic approach based on Constraint Programming (CP) techniques to handle centralized generic controller decision-making. Indeed, CP provides a generic and declarative way to describe combinatorial problems, known as Constraint Satisfaction Problem (CSP). Based on high-level objects (constraints), the paradigm keeps the code human-readable and eases its implementation and maintenance. In addition, CP also comes with automatic solving techniques which exploit the sub-problems captured in the constraints to reduce the search space by eliminating the impossible areas. Constraint-based techniques exist to solve problems in decentralised ways, namely distributed CSP [31]. Most of the time, these algorithms are situated in contexts where agents are hierarchically organized to cooperate in a static environment. In addition, it is also assumed that an agent is responsible for a single variable. Unfortunately, these assumptions do not hold in SeMaFoR since some variables need to be shared among many controllers and the controllers may not be ordered and are competitors in a dynamic environment.

[4]Consensus algorithms aim to solve common problems in distributed systems such as transaction commit in replicated databases, clock synchronization, control of autonomous vehicles or blockchains.

III. The SeMaFoR approach

A. Overview of the approach

By considering that each controller follows a local MAPE-K model [5], Figure 1 depicts the overview of the SeMaFoR approach for three controllers, one for each Fog area. We use the Coordinated Control Pattern defined by [13] where all the 'M', 'A', 'P' and 'E' components of different loops can interact to share particular information and/or coordinate their actions. The different loops in each Fog area run asynchronously, *i.e.*, a loop can be in the 'A' state while another can be in a 'P' state. Synchronization occurs when it is necessary to collectively make a decision that impacts the state of several Fog areas in the same neighborhood. In the SeMaFoR approach, two controllers are considered as neighbors (or peers) if their corresponding Fog area are logically linked implying thus a potential collaboration in their control. A link can represent for instance the offering of same type of physical or logical resources, a geographical area sharing, services offered by one and consumed by the other one, etc. In Figure 1, the controller $C1$ is linked with controller $C2$ (in red) and $C3$ (in green). However, $C2$ and $C3$ are not neighbors and do not interact. The different colors represent the different neighborhoods. Note that a neighborhood can have more than two controllers.

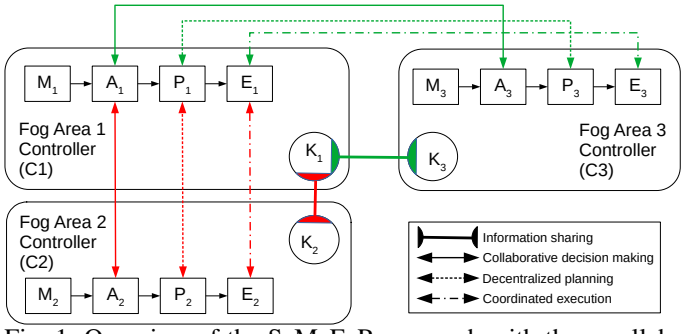

Fig. 1: Overview of the SeMaFoR approach with three collaborating MAPE-K controllers of Fog areas.

Inside a loop, a local monitoring is performed to continuously keep an updated state of the set of resources (nodes and services) within a Fog area. This knowledge 'K' also contains the set of constraints and goals to continuously satisfy on the nodes and services (*e.g.*, QoS). From this local knowledge, some high-level information can be shared between neighbor areas. However, building a complete global knowledge of the Fog system is not considered: our approach is fundamentally based on information decentralization.

When some constraints are identified as not satisfied through monitoring, and if the problem cannot be solved locally, a collaborative decision making (through a consensus protocol) between controllers in the relevant neighborhood is initiated. The neighborhood is established depending on the type of constraints violated. More than one neighborhood can be required to satisfy again a set of constraints. Once a neighborhood is established, all associated controllers participate in the decision making and potentially to the rest of the adaptation process.

After this collaborative step and if all concerned neighborhoods find a fixed point, a new state to reach is computed for each Fog area involved in the neighborhood. Then, a reconfiguration plan (*i.e.*,, a program) is built from the knowledge of both the current state and the new state to reach for each Fog area. In this step, even if a local state is used, the plan may require some synchronization with the plans of other areas. For this reason, the plan needs to be decentralized. Finally, the plan of each Fog area is executed which requires some point-to-point decentralized coordination between Fog areas.

The rest of this section gives details on three challenges we address: the description language required to model the Fog resources in 'K'; the collaborative decision-making in 'A'; and the decentralized plans in 'P'. The monitoring 'M' and the execution 'E' are discussed in Section IV.

Running example - Throughout this section we will use a running example to illustrate our approach. This example focuses on a placement problem (*i.e.*, the first configuration) but the approach is valid for any (re)configuration. Let us consider a new request from a user of the platform to host a new service-oriented application $\mathcal{A}$. $\mathcal{A}$ is made of four types of services, namely s_A, s_B, s_C, and s_D. As illustrated in Figure 2 the service provided by s_A is used by the four instances of the service s_B and by one instance of the service s_D. In addition, the service provided by s_D is used by one instance of the service s_C. Each service is associated to a set of constraints to be continuously satisfied:

- s_A should have access to CO_2 sensors;
- s_B, as running AI algorithms, should have access to GPU hardware;
- s_D must be geographically close to s_A for latency reasons;
- s_C need a machine with at least 100GB of RAM.

This is a typical application that may run in the Fog, as for example a smart-city application where dynamic CO_2 data are used to build a machine learning model (by using GPUs), and aggregated to other information to perform a postmortem big data computation that requires more than 100GB of RAM.

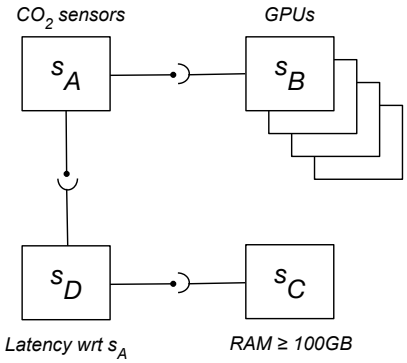

Fig. 2: Example of a new application $\mathcal{A}$ to host on the Fog resources. $\mathcal{A}$ is made of one instance of the service s_A, s_C, and s_D, and four instances of the service s_B. Each service has some specific constraints.

B. A Fog architecture description language

The main objective of the SeMaFoR Fog Modeling Language is to allow specifying both the resources composing the Fog resources and the required knowledge associated to the services and applications hosted on this infrastructure. In the context of our ongoing work on this modeling language, we have already identified key characteristics to be supported: at the Fog system level, it is required to properly model the different Fog areas and their possible interconnections; at the Fog area level, it is needed to adequately represent the various involved Fog resources, their type, corresponding constraints, and tags (to label these resources with required metadata according to specific properties of interest).

Running example - In the model defined with the SeMaFoR Fog Modeling Language, we can notably specify the following information in the context of our running example with more than ten Fog areas (for instance):

1) The user is located in the Fog area F_1;
2) The service s_A can be hosted either on F_1 or F_4 that provide CO_2 sensors;
3) s_B can be hosted either on F_1 or F_{10} that provide GPUs;
4) s_D must be hosted in the same area as s_A;
5) s_C can be hosted either on F_2, F_3, or F_4 that provide machines with sufficient RAM.

This can be formalized as follows with the notation $\twoheadrightarrow$ for "hosted on":

$$user \twoheadrightarrow F_1 \tag{1}$$
$$s_A \twoheadrightarrow F_1 \vee F_4 \tag{2}$$
$$s_B \twoheadrightarrow F_1 \vee F_{10} \tag{3}$$
$$s_D \twoheadrightarrow Area(s_A) \tag{4}$$
$$s_C \twoheadrightarrow F_2 \vee F_3 \vee F_4 \tag{5}$$

From an end-user perspective, this will be modeled using a YAML-like textual syntax proposed with our ADL (currently still under development), as previewed in Listing 1.

Listing 1: Running example specified in the SeMaFoR Fog Modeling Language (excerpt)

```
resourceType: Application
metadata:
alias: co2DataCollection
Spec:
  services:
  - sA:
    metadata:
      alias: sensingService
      tags:
          - sensor : co2
    Spec:
      networkPorts:
        - 'servicePort':
          protocol: TCP
          portNumber: 3333
          hostPort: 8181
      hostedOn:
        softwarePacakge:
        - sensorCode:
          command: 'python featureExtraction.py'
          ...
      constraints:
        replicas: 1
        hostingNode:
          tags:
          - co2Sensor: true
  - sB:
    metadata:
      alias: ML
      ...
  - sC:
    metadata:
      alias: BigData
      ...
  - sD:
    metadata:
```

```
    alias: dataAgregation
    ...

connectors:
- sATosB:
  ...
- sATosD:
  sourceServices:
  - sensingService:servicePort
  targetServices:
  - dataAgregation:receptionPort
  constraints:
  - latency:
      lesserThan: 100 miliseconds
- sDTosC:
  ...
```

C. Collaborative analysis by combining DCSP and Consensus algorithms

The model describes a set of rules to be respected in order to decide where each service must be located. Such decision making may involve several Fog areas. Therefore, the decision is made collaboratively between the controllers distributed over the Fog logical areas (*a.k.a.* neighbors). First, each autonomic controller will be defined as a constraint satisfaction problem, to take into account its local constraints. Then, the inter-controller decision requires a distributed consensus algorithm where multiple decisions are taken asynchronously by several subsets of controllers, managing potential conflicts.

Running example - From the running example, we can note that satisfying the constraints on a single local area is impossible. Hence, a collaboration between areas is required. The set of areas to consider for the consensus is the union of areas of Equations (1) to (5): $\{F_1, F_2, F_3, F_4, F_{10}\}$. The goal is to collaboratively decide on which area each service will be hosted. For instance, the result of the consensus could be the following, by denoting s_{B_i} the i^{th} instance of the service s_B:

$$s_A \twoheadrightarrow F_1 \tag{6}$$
$$s_{B_1}, s_{B_2} \twoheadrightarrow F_1 \tag{7}$$
$$s_{B_3}, s_{B_4} \twoheadrightarrow F_{10} \tag{8}$$
$$s_D \twoheadrightarrow F_1 \tag{9}$$
$$s_C \twoheadrightarrow F_3 \tag{10}$$

In particular, no single Fog area is able to host all instances of s_B. A collaborative decision must be made between F_1 and F_{10} to spread instances of s_B among these Fog areas.

Each request (or reconfiguration stimuli) questions either the internal state of a controller or a shared state. Each involved controller integrates the request within its CSP, solves the new problem and indicates in return whether or not a consensus is needed. If exactly one controller is able to satisfy the request, a solution is found and no consensus is required (intra-controller dimension). When several controllers can individually satisfy the request or when none of them can, there is a need for a consensus (inter-controller dimension). In the first case, a collaborative decision must be taken to assign the request to a single Fog area. The second case concerns two different situations. When several controllers can jointly satisfy the request, a consensus will define how to allocate the request over these Fog areas. When there is no solution without changing the data of the problems, a consensus is required to decide on the action to be executed : *a*) revocation of previous decisions, *b*) relaxation of some constraints or *c*) rejection of the request. Such a decision could be facilitated by querying the different CSPs to determine the conflicting variables and constraints, namely the minimal conflict set [32]. On the running example, the explanation of why a single Fog Area cannot host all S_B is a high-level information that allows the consensus algorithm to limit the scope of the decision to be arbitrated.

D. Decentralized plan with CSPs

As explained in Section III-A, the planning step of the SeMaFoR approach takes as input the current state of the local Fog area, and the goal state for this Fog area that has been decided by the collaborative analysis step (see previous section). Each Fog area takes its own local inputs and will have to build their reconfiguration plan. Hence, the output is a set of plans, one for each Fog area concerned by the collaboration. Intuitively, as Fog areas have collaborated in the analysis step to find a solution to the global current problem (*e.g.*, new request, reconfiguration, etc.), the plan of each Fog area needs at some point to synchronize with the plans of other neighbor Fog areas. In the literature the inference of reconfiguration plans has been previously studied, notably in our previous work [33], however, as far as we know, a decentralized and synchronized plan among different controllers has never been explored.

Running example - From the decision taken by the consensus with Equations (6) to (10), the goal here is to synthesize or infer a decentralized reconfiguration plan between areas. The decentralized reconfiguration plan could be simplified as the three following reconfiguration plans on F_1, F_{10}, and F_3 (respectively). The `wait` instruction in this example illustrates the required point-to-point synchronizations (between areas) when deploying the application $\mathcal{A}$ of Figure 2. For instance, as s_B is using s_A, the instances s_{B_3} and s_{B_4} deployed on F_{10} cannot be deployed before s_A is deployed on F_1.

Algorithm 1 Reconfiguration plan on F_1

deploy s_A
deploy s_{B_1}, s_{B_2}
wait(s_{B_3}, s_{B_4} on F_{10})
deploy s_D

Algorithm 2 Reconfiguration plan on F_{10}

wait(s_A on F_1)
deploy s_{B_3}, s_{B_4}

Algorithm 3 Reconfiguration plan on F_3

wait(s_D on F_1)
deploy s_C

Note that this example is voluntarily simplified for clarity. Indeed, at least one local CSP is usually needed after the consensus to precisely decide where to deploy a service in the Fog area [34], as well as how to deploy it [35], [36].

Inferring a plan requires to solve a specific kind of scheduling problem [33]. Indeed, generating a program means finding the right order of instructions to move from the current state to the goal state. CP is extensively used in the literature to solve scheduling problems so we plan to follow this direction again. The modeling of the problem consists in modeling the set of

available reconfiguration instructions (*e.g.*, adding, removing, connecting some elements, waiting) and their impact on the state of the Fog area, and the set of constraints on these reconfiguration instructions (*e.g.*, impossible combinations). However, as the global knowledge of the target state of each Fog area is unknown, several interdependent planning problems have to be solved. For example, a simple synchronization can rely on the knowledge of which Fog area to wait and when. To go further, more complex collaborations between neighboring Fog areas could be considered as well to, for instance, optimize their reconfiguration duration.

IV. Discussions

We previously described the core of our vision of Fog autonomic computing. However, there are still some interesting medium-term challenges, notably when it comes to integrate our solution into open source ecosystem(s).

A. Models and Monitoring

One of the main benefits of a Fog Modeling Language is to be able to create and reuse models describing snapshots of the modeled Fog system and resources at different points in time. Such a feature is fundamental for supporting Fog system monitoring within the approach we propose. To this end, existing model-based techniques could be studied. Notably, *models@runtime* proposes to rely on models describing actual system's states to reason on it and then react accordingly [37]. This paradigm has already been applied in the context of dynamic adaptive systems, but some challenges remain. For example, it is often complex to maintain the required synchronization between the system and its model. This is even more complicated when dealing with highly decentralized systems, such as Fog ones, possibly represented by several interrelated (sub)models. Indeed, this requires the modeling system to 1) efficiently collect the needed system runtime data from various heterogeneous Fog resources, 2) correctly federate this data into consistent model(s), and 3) propagate eventual model changes back to the appropriate parts of the system. In the context of SeMaFoR, we plan to realize this work in close collaboration with our industrial partner Smile who has a strong expertise on monitoring solutions (*e.g.*, Prometheus[5], Jaeger[6]).

B. Constraints relaxation

In dynamic environments such as Fog systems, it is quite possible that no consensus can be reached without revisiting the constraints that should be satisfied by a controller, *i.e.*, relaxing its CSP. In this situation, deciding what and how to relax the controller's constraints is also a collaborative process. In SeMaFoR, we plan to exploit the minimum conflict sets [32] established by the controllers to identify more specifically the incompatibilities. This phase could be improved by leveraging resolution proofs [38]. The latter should define precisely the reasons for the failure, *i.e.*, the constraints involved, in order to relax the problem in a relevant way. Actually, it is quite possible to automatically relax the CSPs formulation by softening the constraints [39]. This possibly leads to the reformulation of the original problem into a Valued CSP [40]. Such a VCSP is then able to take into account costs, uncertainties, priorities, etc. However, this raises the question of the distance which affects the quality of solutions of the relaxed problem. Indeed, the functions that define the distance to the satisfaction of each constraint can be very varied. Such choices introduce biases on the solutions to be produced. In SeMaFoR, we plan to study first distance functions on the constraints that govern the Fog areas, that is, those of resource constrained problems.

C. Execution of the reconfiguration and DevOps ecosystem integration

Many DevOps frameworks/tools offer specific adaptation capabilities. For instance, AWS Cloud Formation[7] and OpenStack Heat[8] can switch from one resource template (virtual machines and services) to another. Some configuration tools (*e.g.*, Puppet, Salt, Anisble etc.) and service orchestrators (*e.g.*, Kubernetes, Nomad) are now used to handle infrastructures as a code, thus automating the *on the fly* changing of infrastructure configurations. However, these techniques are often based on specific virtualization techniques and thus technology-dependent. For instance, Kubernetes[9] and Docker Swarm[10] support scaling and reboot-on-crash for services in containers [41]. Service meshes even go further by dynamically handling the connections between services and associated network aspects. For example, solutions like Istio[11] (on top of Kubernetes) can handle circuit breaking to prevent service faults.

SeMaFoR focuses on a generic and abstracted reconfiguration without considering specific virtualization techniques or targeted platforms unlike the above approaches. Such genericity enhances reuse, maintainability and extensibility of reconfiguration and make our solution theoretically compatible with all the above tools if an integration effort is made. Thanks of its extensibility properties, Component-Based Software Engineering (CBSE) has been widely used as an enabler technology to enhance dynamic reconfiguration of distributed software systems. In particular, Aeolus [42] and Concerto [35], [43] offer a very powerful expressiveness to customize any reconfiguration operation and the associated coordination points between components (*resp.* inspired by state machines and Petri nets). These reconfiguration models also offer the finest granularity which enhances efficiency capabilities. Recently, the semantics of Concerto has been adapted to a decentralized execution, making this tool a good candidate for the reconfiguration language of SeMaFoR (the (E)xecution of the MAPE-K model). However, even though the goal of SeMaFoR is to be as generic as possible, we plan to integrate the SeMaFoR solution with different concrete backends such as Kubernetes or OpenStack.

Acknowledgment

The authors acknowledge the support of the French Agence Nationale de la Recherche (ANR), under grant ANR-20-CE25-0017 (SeMaFoR project).

[5]https://prometheus.io/
[6]https://www.jaegertracing.io/
[7]https://aws.amazon.com/cloudformation/
[8]https://wiki.openstack.org/wiki/Heat
[9]https://kubernetes.io/
[10]https://docs.docker.com/engine/swarm/
[11]https://istio.io/

References

[1] C.-H. Hong and B. Varghese, "Resource management in fog/edge computing: A survey on architectures, infrastructure, and algorithms," *ACM Comput. Surv.*, vol. 52, no. 5, pp. 97:1–97:37, Sep. 2019.

[2] A. Yousefpour, C. Fung, T. Nguyen, K. Kadiyala, F. Jalali, A. Niakanlahiji, J. Kong, and J. P. Jue, "All one needs to know about fog computing and related edge computing paradigms: A complete survey," *Journal of Systems Architecture*, vol. 98, pp. 289 – 330, 2019.

[3] I. B. Lahmar and K. Boukadi, "Resource allocation in fog computing: A systematic mapping study," in *2020 Fifth International Conference on Fog and Mobile Edge Computing (FMEC)*, 2020, pp. 86–93.

[4] R.-A. Cherrueau, A. Lebre, D. Pertin, F. Wuhib, and J. M. Soares, "Edge computing resource management system: a critical building block! initiating the debate via openstack," in *USENIX Workshop on Hot Topics in Edge Computing (HotEdge 18)*, Boston, MA, Jul. 2018.

[5] J. Kephart and D. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 41–50, 2003.

[6] Z. Al-Shara, F. Alvares, H. Bruneliere, J. Lejeune, C. Prud'Homme, and T. Ledoux, "Come4acloud: An end-to-end framework for autonomic cloud systems," *Future Generation Computer Systems*, vol. 86, pp. 339 – 354, 2018.

[7] M. Ghobaei-Arani, S. Jabbehdari, and M. A. Pourmina, "An autonomic resource provisioning approach for service-based cloud applications: A hybrid approach," *Future Generation Computer Systems*, vol. 78, pp. 191 – 210, 2018.

[8] F. A. d. Oliveira, T. Ledoux, and R. Sharrock, "A framework for the coordination of multiple autonomic managers in cloud environments," in *2013 IEEE 7th International Conference on Self-Adaptive and Self-Organizing Systems*, Sep. 2013, pp. 179–188.

[9] C. Mouradian, D. Naboulsi, S. Yangui, R. H. Glitho, M. J. Morrow, and P. A. Polakos, "A comprehensive survey on fog computing: State-of-the-art and research challenges," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 1, pp. 416–464, 2018.

[10] S. Dlamini, J. Mwangama, N. Ventura, and T. Magedanz, "Design of an autonomous management and orchestration for fog computing," in *2018 Int. Conf. on Intelligent and Innovative Computing Applications (ICONIC)*, 2018, pp. 1–6.

[11] L. Baresi, D. F. Mendonça, and G. Quattrocchi, "Paps: A framework for decentralized self-management at the edge," in *Service-Oriented Computing*, S. Yangui, I. Bouassida Rodriguez, K. Drira, and Z. Tari, Eds. Springer, 2019, pp. 508–522.

[12] Z. A. Mann, "Decentralized application placement in fog computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, pp. 3262–3273, 12 2022.

[13] D. Weyns, B. Schmerl, V. Grassi, S. Malek, R. Mirandola, C. Prehofer, J. Wuttke, J. Andersson, H. Giese, and K. M. Göschka, *On Patterns for Decentralized Control in Self-Adaptive Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 76–107.

[14] A. Belhaj Seboui, N. Ben Hadj-Alouane, G. Delaval, E. Rutten, and M. Yeddes, "An approach for the synthesis of decentralised supervisors for distributed adaptive systems," *International Journal on Critical Computer-based Systems*, vol. 2, no. 3/4, pp. 246–265, 2011.

[15] F. Quin, D. Weyns, and O. Gheibi, "Decentralized self-adaptive systems: A mapping study," in *2021 International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*. IEEE, 2021, pp. 18–29.

[16] S. Forti, M. Gaglianese, and A. Brogi, "Lightweight self-organising distributed monitoring of fog infrastructures," *Future Generation Computer Systems*, vol. 114, pp. 605–618, 1 2021.

[17] M. A. Tamiru, G. Pierre, J. Tordsson, and E. Elmroth, "mck8s: An orchestration platform for geo-distributed multi-cluster environments," in *2021 International Conference on Computer Communications and Networks (ICCCN)*, 2021.

[18] "Admiralty." [Online]. Available: https://github.com/admiraltyio/admiralty

[19] "Kubefed." [Online]. Available: https://github.com/kubernetes-sigs/kubefed

[20] F. Rossi, V. Cardellini, and F. L. Presti, "Hierarchical scaling of microservices in kubernetes," in *2020 IEEE International Conference on Autonomic Computing and Self-Organizing Systems (ACSOS)*, 2020.

[21] A. Pires, J. Simão, and L. Veiga, "Distributed and decentralized orchestration of containers on edge clouds," *Journal of Grid Computing*, vol. 19, no. 3, pp. 1–20, 2021.

[22] L. L. Jiménez and O. Schelén, "Docma: A decentralized orchestrator for containerized microservice applications," in *2019 IEEE Cloud Summit*, 2019.

[23] A. Bergmayr, U. Breitenbücher, N. Ferry, A. Rossini, A. Solberg, M. Wimmer, G. Kappel, and F. Leymann, "A Systematic Review of Cloud Modeling Languages," *ACM Computing Surveys (CSUR)*, vol. 51, no. 1, p. 22, 2018.

[24] OASIS, "Topology and Orchestration Specification for Cloud Applications (TOSCA)," Nov. 2013. [Online]. Available: http://docs.oasis-open.org/tosca/TOSCA/v1.0/TOSCA-v1.0.html

[25] P. Habibi, M. Farhoudi, S. Kazemian, S. Khorsandi, and A. Leon-Garcia, "Fog Computing: A Comprehensive Architectural Survey," *IEEE Access*, vol. 8, pp. 69 105–69 133, 2020.

[26] A. Alidra, H. Bruneliere, and T. Ledoux, "A feature-based survey of fog modeling languages," *Future Generation Computer Systems*, vol. 138, pp. 104–119, 2023.

[27] H. Bruneliere, Z. Al-Shara, F. Alvares, J. Lejeune, and T. Ledoux, "A model-based architecture for autonomic and heterogeneous cloud systems," in *8th Int. Conference on Cloud Computing and Services Science*. SciTePress, 2018, pp. 201–212.

[28] H. Bruneliere, F. M. de Kerchove, G. Daniel, S. Madani, D. Kolovos, and J. Cabot, "Scalable model views over heterogeneous modeling technologies and resources," *Software and Systems Modeling*, 2020.

[29] L. Lamport, R. Shostak, and M. Pease, "The byzantine generals problem," *ACM Trans. Program. Lang. Syst.*, vol. 4, no. 3, pp. 382–401, Jul. 1982.

[30] M. J. Fischer, N. A. Lynch, and M. S. Paterson, "Impossibility of distributed consensus with one faulty process," *J. ACM*, vol. 32, no. 2, pp. 374–382, Apr. 1985.

[31] F. Fioretto, E. Pontelli, and W. Yeoh, "Distributed constraint optimization problems and applications: A survey," *Journal of Artificial Intelligence Research*, vol. 61, 02 2016.

[32] U. Junker and F-Valbonne, "Quickxplain: Conflict detection for arbitrary constraint propagation algorithms," 2001.

[33] S. Robillard and H. Coullon, "Smt-based planning synthesis for distributed system reconfigurations," in *Fundamental Approaches to Software Engineering*, 2022.

[34] H. Coullon, G. Le Louët, and J.-M. Menaud, "Virtual Machine Placement for Hybrid Cloud using Constraint Programming," in *ICPADS 2017*, Shenzhen, China, Dec. 2017.

[35] M. Chardet, H. Coullon, and C. Pérez, "Predictable efficiency for reconfiguration of service-oriented systems with concerto," in *CCGrid 2020 - 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing*. Melbourne, Australia: IEEE, Nov. 2020.

[36] M. Chardet, H. Coullon, and S. Robillard, "Toward Safe and Efficient Reconfiguration with Concerto," *Science of Computer Programming*, vol. 203, pp. 1–31, Mar. 2021.

[37] N. Bencomo, S. Götz, and H. Song, "Models@ run. time: a guided tour of the state of the art and research challenges," *Software & Systems Modeling*, vol. 18, no. 5, pp. 3049–3082, 2019.

[38] M. Veksler and O. Strichman, "A proof-producing CSP solver," in *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2010, Atlanta, Georgia, USA, July 11-15, 2010*, M. Fox and D. Poole, Eds. AAAI Press, 2010.

[39] P. Meseguer, F. Rossi, and T. Schiex, "Soft constraints," in *Handbook of Constraint Programming*, ser. Foundations of Artificial Intelligence, F. Rossi, P. van Beek, and T. Walsh, Eds. Elsevier, 2006, vol. 2, pp. 281–328.

[40] M. Cooper, S. De Givry, and T. Schiex, "Valued Constraint Satisfaction Problems," in *A Guided Tour of Artificial Intelligence Research*, ser. AI Algorithms. Springer Int. Publishing, 2020, vol. 2, pp. 185–207.

[41] H. Coullon, D. Pertin, and C. Pérez, "Production deployment tools for iaases: an overall model and survey," in *The IEEE 5th International Conference on Future Internet of Things and Cloud (FiCloud)*, Prague, Czech Republic, Aug. 2017, pp. 183–190.

[42] R. Di Cosmo, J. Mauro, S. Zacchiroli, and G. Zavattaro, "Aeolus: a component model for the Cloud," *Information and Computation*, pp. 100–121, Jan. 2014.

[43] M. Chardet, H. Coullon, D. Pertin, and C. Pérez, "Madeus: A formal deployment model," in *4PAD 2018 - 5th International Symposium on Formal Approaches to Parallel and Distributed Systems (hosted at HPCS 2018)*, Orléans, France, Jul. 2018, pp. 1–8.