QoS-Based Elasticity for Service Chains in Distributed Edge Cloud Environments

Valeria Cardellini[1], Tihana Galinac Grbac[2](✉), Matteo Nardelli[1], Nikola Tanković[3], and Hong-Linh Truong[4]

[1] University of Rome Tor Vergata, Rome, Italy
{cardellini,nardelli}@ing.uniroma2.it
[2] University of Rijeka, Rijeka, Croatia
tihana.galinac@riteh.hr
[3] Juraj Dobrila University of Pula, Pula, Croatia
nikola.tankovic@unipu.hr
[4] TU Wien, Vienna, Austria
hong-linh.truong@tuwien.ac.at

Abstract. With the emerging IoT and Cloud-based networked systems that rely heavily on virtualization technologies, elasticity becomes a dominant system engineering attribute for providing QoS-aware services to their users. Although the concept of elasticity can introduce significant QoS and cost benefits, its implementation in real systems is full of challenges. Indeed, nowadays systems are mainly distributed, built upon several layers of abstraction, and with centralized control mechanisms. In such a complex environment, controlling elasticity in a centralized manner might strongly penalize scalability. To overcome this issue, we can conveniently split the system in autonomous subsystems that implement elasticity mechanisms and run control policies in a decentralized manner. To efficiently and effectively cooperate with each other, the subsystems need to communicate among themselves to determine elasticity decisions that collectively improve the overall system performance. This new architecture calls for the development of new mechanisms and efficient policies. In this chapter, we focus on elasticity in IoT and Cloud-based systems, which can be geo-distributed also at the edge of the networks, and discuss its engineering perspectives along with various coordination mechanisms. We focus on the design choices that may affect the elasticity properties and provide an overview of some decentralized design patterns related to the coordination of elasticity decisions.

1 Introduction

Elasticity is a quality attribute that is widely used in virtual environments together with the "as a service" paradigm to deal with on-demand changes. Although elasticity is multi-dimensional [27,72], in most cases, elasticity techniques just focus on offering elastic resources on demand and dynamically provision them to fluctuating workload needs based on the "pay-per-use" concept [23].

I. Ganchev et al. (Eds.): Autonomous Control for a Reliable Internet of Services, LNCS 10768, pp. 182–211, 2018.
https://doi.org/10.1007/978-3-319-90415-3_8

In this sense, elasticity mechanisms automatize the process of reconfiguring virtualized resources, mostly at infrastructural levels, at runtime with the goal of sustaining offered Quality of Service (QoS) levels and optimizing resource cost.

Due to its usefulness, there are many works that have addressed issues related to elasticity [23]. However, most of them discuss elasticity in specific environments, such as Cloud systems in centralized, large-scale data centers (e.g., [46]), edge/fog-based systems (e.g., [54]), network function virtualization (NFV) (e.g. [67]), except a few works that consider Internet of Things (IoT) Cloud systems, e.g., [70]. In this chapter, we investigate how distributed systems can be efficiently executed in the emerging context resulting from the convergence of IoT, NFV, edge systems, and Clouds. More precisely, our goal is to survey elasticity needs, mechanisms, and policies for geo-distributed systems running over multiple edge/fog[1] and Cloud infrastructures. Furthermore, we present several design patterns that help to efficiently decentralize and coordinate the elasticity control of such systems. The main contributions of this chapter are the following:

- We present how the emerging computing paradigms and technologies help to realize elastic systems, which can execute with guaranteed QoS even in face of changing running conditions.
- We survey the key elasticity properties and techniques that have been presented so far in the related literature. Specifically, we survey the approaches that enable elasticity at different stages of the system life time, distinguishing between design-time and runtime.
- Motivated by the scalability limitation of distributed complex systems, we propose different coordination patterns for decentralized elasticity control. The latter represent architectural design guidelines that help to oversee large scale systems with the aim to improve performance and reliability without compromising scalability.
- We describe the main challenges of nowadays systems so to identify research directions that are worth of investigation, in order to develop seamlessly elastic systems that can operate over geo-distributed and Cloud-supported edge environments.

The rest of the chapter is organized as follows. In Sect. 2 we provide an overview about elasticity. In Sect. 3 we briefly present the large-scale distributed systems we focus on in this chapter, that is systems of IoT, NFV and Clouds and discuss their elasticity coordination needs. In Sect. 4 we provide an overview of optimization approaches used to take elasticity choices. In Sect. 5 we present some design patterns that can be used in distributed edge Cloud environments to coordinate elasticity decisions in a decentralized fashion. In Sect. 6 we discuss some research challenges for elasticity control. We conclude the chapter in Sect. 7 with some final remarks.

[1] From our point of view, in this chapter we consider edge computing as interchangeable with fog computing, although we are aware that some differences exist [53].

2 Overview of Elasticity

Elasticity has become one of the key attributes of self-adaptive software systems. Although it has been and is widely investigated, there is no unique consensus related to elasticity definition. The most frequently used definition of elasticity has been formulated by Herbst et al. [36] as follows: "Elasticity is the degree to which a system is able to adapt to workload changes by provisioning and deprovisioning resources in an autonomic manner, such that at each point in time the available resources match the current demand as closely as possible". The elasticity quality attribute is tightly related to the scalability and efficiency attributes. Scalability addresses a typically static system attribute related to the ability of a system to adjust its resources to changing load. However, volatile software environments demand a continuous adaptation process [78], which yields considerable additional costs if applied manually. Another, closely related quality attribute is efficiency, that is related to the amount of the resources consumed to process traffic needs. Traditionally, these terms were related to a static system configuration and not considered in terms of dynamical system architecture models.

With the emergence of virtualization technologies, especially lightweight ones such as containers [12] and unikernels [13], there are new automation possibilities no longer related to the physical scaling of system resources, but rather to the dynamic adaptation of the system to deal with changing environment conditions. System/application components can scale out according to traffic needs to accommodate changes in the traffic volumes and avoid SLA violations, and can scale in to save energy and costs caused by over–dimensioning. Virtualization technologies have opened new possibilities to system automation and implementation of elastic attribute into dynamic systems. However, when implemented in real systems, the beneficial effects of elasticity can be limited mainly by the speed of the system adaptation process and by the precision in aligning the allocated virtual resources to the temporal resource demands. Therefore, dynamic adaptation models have also to consider limitations of real systems to adapt timely and precisely.

The main aim of dynamic adaptation models is to exploit optimization algorithms that guide elastic decisions at runtime, as traffic changes for the best QoS and cost gains, while considering a large combinatorial set of architectural design options that are no longer manageable by human designers [77]. Optimization solutions can be categorized according to several key aspects [4]:

– *Which software attributes are to be optimized?* Every software attribute for which a representing quantifiable model can be provided is a candidate to be used in the quality evaluation function. Quality attributes also include economical attributes, such as associated costs [10], among which operational infrastructure costs prevail in the Cloud era [27]. According to the selected quality attributes, optimization approaches can be single- or multi-objective.
– *What design choices are considered under optimization?* In order to provide an automatized optimization process, a machine-readable format of the

software architecture is required. These can vary from formal models, UML models, or models in different architectural languages such as ADL. On top of that model, there must also exist an unambiguous definition of what combinatorial, categorical or ordinal variables are to be considered in forming an optimization search space. These definitions may also yield additional design constraints, which exclude some of the combinations due to some architectural constraints (e.g., applying certain architectural style). In literature, these variables are referred as architectural degrees of freedom (DoF) [42].
– *In which phase does optimization take place?* According to this dimension, solutions vary from design-time optimization to runtime optimization methods. In design-time approaches, the system is first modeled in the desired language where optimization is performed on derived models according to specific quality attributes. These can include block diagrams, Markov chains, queuing networks, Petri nets with quality attributes predicted by using a computer simulation or analytical models when they are available in closed form. Runtime approaches are generally simpler due to stringent execution speed and overhead constraints, so they often consider optimizing only a single attribute or they naively combine several attributes using the simple additive weighting (SAW) method [39].

A thorough literature review of existing optimization methods used in software architectures was performed in [4]; therefore, we analyze only research works that have been conducted afterwards. We focus on emerging systems of IoT, virtual network functions, and distributed Clouds. We also give special attention to optimization in the domain of distributed system environments and classify existing works according to the phase of execution. Furthermore, we consider decentralized coordination design patterns that can be employed to realize a distributed elasticity control where elasticity decisions have to be taken at multiple layers.

3 Systems of IoT, NFV and Clouds and Their Elasticity Coordination Needs

Research works related to elastic architectures and applications spawn multiple areas, ranging from embedded systems and information systems design, to software performance engineering and quality attributes [4]. A general observation from all involved research communities is that system complexity generally increases and, as such, it is hard to manage and scale, is expensive to maintain and change. A general trend is to define new system architectural models that decompose complex system architectures into smaller and easily self-manageable objects, e.g., microservices [26]. These new system architectures are based on virtualization and automatic software provisioning and configuration technologies to enable dynamic system models that can autonomously adapt to face varying operating conditions.

Emerging systems and services are and will be characterized by the integration and convergence of different paradigms and technologies that span from

IoT, virtual network functions, distributed edge/fog computing, and Cloud computing [73]. We briefly review the main features of some of these paradigms and technologies prior to analyze their coordination needs.

NFV is a new network architecture framework where network functions, which traditionally used dedicated hardware (e.g., network appliances), are implemented in software that runs on top of general purpose hardware, exploiting virtualization technologies. Virtual network functions can be interconnected into simple service compositions (called chains) to create more complex communication services. Component network functions in the service chain can be scaled either vertically or horizontally (i.e., either acquiring more powerful computing resources or spawning more replicas of the same virtual network function and load balancing among them).

Edge and fog computing paradigms provide a distributed computing and storage infrastructure located at the edges of the network, resulting in low latency access and faster response times to application requests. These paradigms turn out to be particularly effective in moving computation and storage capabilities closer to data production sources (e.g., IoT devices) and data consumption destinations, which are heavily dispersed and typically located at the edges of the network. Therefore, they can better meet the requirements of IoT applications with respect to the use of a conventional Cloud [64].

Dealing with elasticity for such emerging systems is important and challenging. However, elasticity techniques that have been separately studied for virtualized systems mainly in large-scale and centralized Cloud data centers or less frequently in distributed edge/fog environments, may not be sufficient to efficiently manage more complex environments that arise from the convergence of IoT, NFV and Clouds. Figure 1 outlines the concept view of such virtualized systems, built atop various views on IoT Cloud [44,70]. With such systems, it is crucial to have an end-to-end elasticity [71], requiring a strong elasticity coordination between the IoT, NFV and Clouds. For example, let us consider how elasticity coordination would help to prepare at best the Cloud to serve data from the edge. Currently, most of the times, the Cloud does not really care about the edge - if more data come, the Cloud reacts and provisions more resources. However, if the elasticity demands from the edge were known and propagated to the Cloud in advance, the Cloud could be able to provision resources in a more effective way. This can be done when we consider that we control on both sides - edge and Cloud. On the one hand, the end-to-end elasticity requires us to work horizontally across IoT, NFV, and Cloud. On the other hand, each system might have different layers, as shown in Fig. 1 and discussed in [65]. Therefore, it is crucial to coordinate elasticity both horizontally and vertically across layers and across subsystems. This leads to our focus on models and techniques to control and manage elasticity.

The following key elasticity properties and techniques are crucial to us to understand:

- Which types of elasticity properties are suitable for which layers (resources, data, service, network)?

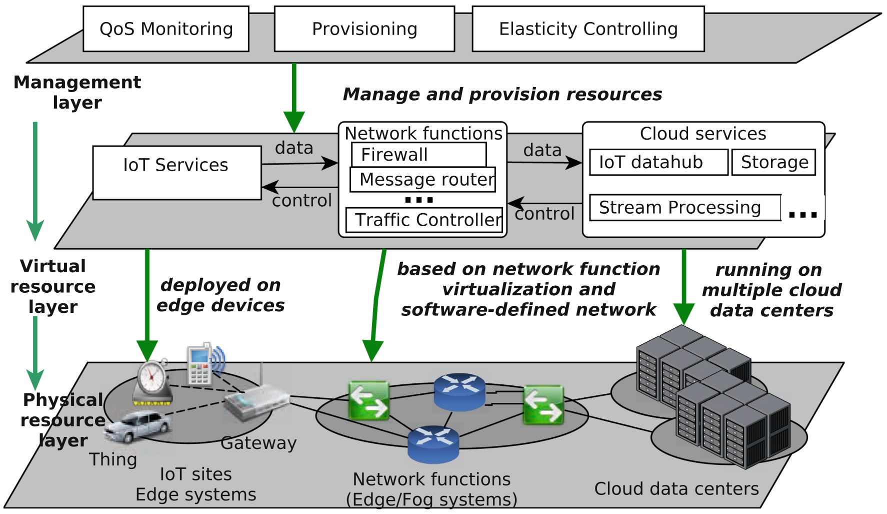

Fig. 1. System of IoT, virtual network functions and Cloud (adapted from [44]).

– Which elasticity control techniques are suitable for which parts (edge, network, or data center) and which models are useful for coordinating them?
– How to connect elasticity coordination between software engineering view and system engineering view?

4 Existing Solutions – Pros and Cons

4.1 Software Attributes and Design Choices

For a successful software optimization it is important to select appropriate software attributes that reflect the users perception of the quality. The most prominent software attribute is performance as it is the subject of most optimization techniques. Performance expresses timings involved around different computation paths. There are many metrics that express software performance with most important being: response time, throughput, and utilization [40].

Another common attribute that is optimized is reliability: the system ability to correctly provide a desired functionality during a period of time in the given context. Another term closely related to reliability is availability: the percentage of time a system is up and running to serve requests [10]. Both these terms are contained in dependability attribute: overall probability that system will produce desired output under specified performance, thus overall user confidence that system will not fail in normal operation.

System costs can also be considered as a business quality attribute [10]. They can be divided to design-time costs: development costs, licensing, hardware acquiring, and maintenance costs as well as runtime costs: operational infrastructure costs and energy costs.

Design choices that are considered in optimization process should not alter any functionality of the end-system, but affect only its quality attributes. Choices can be software related or deployment related [42] and are categorized in Table 1.

4.2 Design-Time Approaches

Historically, design-time optimization solutions were oriented to embedded systems because of their stringent extra-functional properties (EFPs) requirements. For that purpose, ArcheOpteryx tool [3] is an Eclipse plug-in that implements AADL specifications [30] and employs multi-objective evolutionary heuristics for approximating optimal solution of embedded component-based systems. Specifically, ArcheOpteryx optimizes communication cost between components in two ways: it optimizes data transmission reliability formed around total frequency of component interactions against network connection reliability; and communication overhead due to limited network bandwidth and delays. Another representative solution from the automotive domain is EAST-ADL language [74], inspired by MARTE modeling language [59]. EAST-ADL also employs genetic algorithms (GAs) with multi-objective selection procedure NSGA-II [25], quite common in all multi-objective approaches. Quality is evaluated using fault-tree models for safety analysis and the MAST analysis tool was used to derive mean system response times. Component life-cycle cost was also one of the objectives.

Recently, the focus of design-time optimization shifted towards information systems, as systems became more complex and at the same time more reliable with stricter EFP requirements regulated through service-level agreements (SLAs). The majority of research works employs search heuristics through various multi-objective evolutionary algorithms. Li et al. [45] applied a model-based methodology to size and plan enterprise application under SLAs, considering response time and cost as optimization quality attributes. They modeled a multi-tier enterprise application with a closed queuing network model and applied an evolutionary algorithm to evaluate different configurations. They parametrized queue network models by measuring the real system and applied exponential arrival and service times. Mean Value Analysis was used to obtain the response time in a stationary state. A similar approach was also employed in [60], where multi-objective evolutionary algorithms have been used to optimize performance and reliability of system expressed through AADL models. Menascé et al. [48] proposed to optimize performance and availability of service-based information systems by applying a hill-climbing optimization method. Overall system is represented as a service-activity model which models execution sequence of different services. The PerOpteryx tool [43] applied a Palladio Component Model (PCM) [11] for predicting the performance of various architecture configurations of component-based information applications. Optimized attributes also included system performance and cost. Industrial case study of PerOpteryx tool was conducted in [32]. The underlying PCM model is automatically transformed to Layered Queue Models (LQM) [66] with predicted values obtained using a simulation. PerOpteryx also applies multi-objective genetic algorithm with NSGA-II

Table 1. Possible software and deployment design choices

Software design choices	
Selection of components	Wherever functionality is encapsulated within interchangeable components like in component-based or service-based architectures a set of compatible components can be expresses with different quality properties. In component-based system such selection is often available only at design-time, while service-based systems enable services to be selected in run-time
Component configuration parameters	Often components provide further configuration parameters that affect their delivered quality. This is especially the case in component-based architectures. For example, in a component that processes and compresses input data, a compression ratio can be altered which can balance the output quality over processing performance. Parameters can also be non-numerical, like selection of compression algorithms or supported encryption algorithms in SSL communication. Such parameters also include the multiplicity of logical resources like limits for allowed number of threads or database connections or state the priorities for certain actions in concurrent processing scenarios. These all affect overall delivered component quality and thus can be subject to optimization
Deployment design choices	
Allocation	Allocation is defined by a mapping from software components to available hardware resources. Each component can be allowed on only a single resource or deployed across several resources. Components can possess certain allocation constraints that need to be satisfied such as minimal amount of RAM required. Distributed systems are very sensitive to allocation as it affects quality attributes like response time, throughput, reliability and availability of system. Performance is affected with the communication overheads between components allocated on different servers, where reliability suffers if components are deployed on same servers which requires a careful balance

(*continued*)

Table 1. (*continued*)

Deployment design choices	
Replication	Replication design choice states the number of deployed component instances required. Replication affects reliability and overall performance. When component replication is present, additional components are required like load-balancers for balancing workload between several components or switches that route traffic from primary components to fail-over components in passive replication scenarios. Replication design freedom is the key run-time parameter in elastic systems as it altered to continually adapt maximal component processing capacity to current workload requirements
Resource selection	When performing software component allocation to hardware resources a number of different configuration options is present: selecting appropriate disk storage, type of CPU/GPU, etc. In embedded systems these are predetermined at design-time but for elastic information systems they can be varied in runtime as well in reconfiguration process. Resource selection primarily affect costs and performance attributes but can also affect dependability attributes. Resource selection can be achieved at different granularity levels. Sometimes selection refers to individual hardware components, but more often it refers to selecting pre-configured available resource types, like selecting virtual machine type from Cloud provider. In the case of selecting whole servers, resource selection can also provided software packages like OS, pre-installed tools and platforms etc.
Resource parameters	Selected resources, both hardware and software, can have many tunable parameters that can be altered at selection/installation time, or sometimes even at runtime. At selection, resources can be chosen based on different parameters (e.g., CPU clock-rate, number of cores, amount of RAM) and during installation different platform parameters can be altered (e.g., virtual memory available, TCP stack parameters, JVM configuration). If supported, some parameters can also be altered during runtime
Resource provider	When selecting resources, different competing providers can be chosen. Differences lay in hardware offers, pricing amount, pricing model options, and offered SLAs. Greatest benefit from choosing diverse resource providers is increase in system reliability and prevention of vendor lock-in
Resource location	In the era of IoT, edge computing and latency critical applications, resource location is also an important factor to optimize. Data center location, whether Cloud data center or micro edge/fog data center, impacts largely on network latencies, especially in distributed mobile systems

selection method. By employing simulation, a more sophisticated set of measures, such as percentiles which are often agreed in SLAs, can be obtained. A faster evaluation method that can also predict performance measures beyond mean values is fluid analysis [69]. Pérez and Casale [57] suggested a method for deriving fluid models LQN networks obtained through PCM models. Fluid models are described by a set of ordinary differential equations that approximate the evolution of Markovian models, in their case closed class queue networks. Malek et al. [47] proposed a method for optimizing availability, communication security, latency, and energy consumption that are influenced from various deployments of a distributed information system. They applied both Mixed-Integer Nonlinear Programming (MINLP) algorithms and genetic algorithms to solve the derived optimization problems. They also provided guidelines on strengths and weaknesses of both approaches. There is also a semi-automatized approach which employs formalized expert knowledge used to suggest different solutions to recurrent problems, like performance bottlenecks as presented in [7]. In [8] anti-patterns are mitigated using a fuzzy approach so that each anti-pattern is labeled with a probability of occurrence. Similar efforts tailored for Cloud environments have been also proposed [62]. Perez-Palancin et al. [58] suggested a framework for analyzing trade-offs between system cost and adaptability. They modeled service adaptability through several metrics based on the number of used components for providing a given service and the total number of components offering such service.

There are also recent solutions that are specialized for dealing with dynamically used logical resources such as elastic Cloud infrastructure. These solutions must take into account the dynamics of used resources over time, which was not supported in before-mentioned approaches. The SPACE4CLOUD project [31] resulted in a design-time tool for predicting costs and performance of certain Cloud information system topology expressed in PCM. In order to enable fully automated search over design space, the SPACE4CLOUD tool was combined with PerOpteryx evolutionary heuristics in a separate study [20]. Evangelinou et al. [19,29] further developed such a tool to provide a methodology for migrating existing enterprise applications to Cloud by selecting an optimal deployment topology that takes topology cost and performance into account. To enable faster search, initial solutions for evolutionary algorithm are provided through Mixed-Integer Linear Programming (MILP) algorithm. Evolutionary algorithms are supplemented with local search heuristics. Like before, application topology in SPACE4CLOUD is optimized for a specific workload intensity, typically at peak. Andrikopoulos et al. [6] employed a graph-based model to represent a Cloud application topology with a complementary method for selecting the best topologies based only on operational infrastructure cost provided by simple analytical models.

4.3 Runtime Approaches

In contrast to design-time approaches, runtime approaches continually variate the chosen architecture DoFs in order to adapt to volatile environments while keeping the desired application attributes optimal. Runtime optimization is primarily focused on, but not limited to, availability, performance, and cost quality attributes and is considered the key characteristic of self-adaptive systems [24]. Since algorithms are running online at all times, they are forced to apply simpler but very fast analytical models like simple aggregation functions (summation, maximal and average values) or analytical models of M/M/1 queues. Research efforts have been mostly oriented towards service-based [52] and Cloud systems. Calinescu et al. [14] systematized a majority of runtime optimization research involved in service-based systems, and based their approach around Discrete Time Markov-Chain models. They provided a means to formally specify QoS requirements, model-based QoS evaluation, and a MAPE-K cycle [38] for adaptation. Passacantando et al. [56] formulated runtime management of IaaS infrastructure from a SaaS Cloud provider viewpoint as a Generalized Nash Equilibrium Problem (GNEP). SaaS providers strive to minimize the costs of used resources, and in parallel IaaS providers tend to maximize profits. From performance aspect, services are modeled as simple M/G/1 queues. A distributed algorithm based on best-reply dynamics is used to compute the equilibrium periodically. Gomez Saez et al. [61] provided a conceptual framework for achieving optimal distribution of application that involves both runtime and design-time processes. Nanda et al. [51] formulated the optimization problem for minimizing the SLA penalty and dynamic resource provisioning cost. Their model defined only single DoF expressed as number of virtual machines designated to each application tier. Grieco et al. [33] proposed an algorithm for the continuous redeployment of multi-tier Cloud applications due to system evolution. They proposed an adaptation graph aimed to find the best composition of adaptation processes satisfying a goal generated at runtime. Goals are defined as transitions from original to destination state. Recently, the SPACE4CLOUD tool was extended to provide optimal runtime scaling decisions limited to replication DoF [34], while Moldovan et al. [49] provided a cost model for resource replication that is more aligned with public Cloud offerings.

4.4 Other Relevant Research

The third group of works we consider is not directly targeting optimization itself, but exploit techniques and mechanisms that are relevant for further optimization. A mapping study that identifies relevant research around modeling QoS in Cloud is in [9]. Copil et al. [22] provided general guidelines to build elastic systems in Cloud, IoT, or hybrid human-computer context. A research agenda for implementing optimization tools for data-intensive applications has been presented in [18,21]; the main concepts to consider are volume, velocity, and location of data. Kistowski et al. [41] proposed to model incoming workload intensity using time-series decomposition to identify seasonal, trend and noise components which could yield in more robust optimization techniques. Andrikopoulos et al. [5] proposed a GENTL language for modeling multi-Cloud applications as the foundation for any optimization of its deployment. They argued that GENTL contains the right amount of abstraction that captures essential concepts of multi-Cloud applications. Similar claim and model are also the result of research by Wettinger et al. [75], where a concept of deployment aggregate is introduced to automate deployment of Cloud applications. Etxeberria et al. [28] argued there is a large amount of uncertainty present in performance results and proposed a technique to tame such uncertainty, while Nambiar et al. [50] highlighted all challenges involved in model-driven performance engineering and proposed a more modular approach to modeling performance. Pahl and Lee [55] demonstrated the application of more lightweight virtualization solutions in the context of edge computing. Such virtualization capabilities should also be integrated in architecture optimization techniques.

A systematic mapping study on software architectural decisions like documenting decisions or functional requirements is provided in [68]. It identifies a recent increase in interest involved around architectural decisions. Considering all research involved on architecture optimization with these conclusions, there is a need for further incentives in closing the gap between human and automated processes around architecture formation and optimization.

5 Coordination Patterns for Decentralized Elasticity Control

An elastic system has the ability to dynamically adjust the amount of allocated resources to meet variations in workload demands [2,23]. To realize an elastic system, we need to perform several operations aimed to observe the system evolution, determine the scaling operations to be performed, and finally reconfigure the system (if needed). A prominent and well-known reference model to organize the autonomous control of a software system is MAPE [24,63]. It includes four main components, namely Monitor, Analyze, Plan, and Execute, which are responsible for the key functions of self-adaptation, and specifically of elasticity.

The Monitor component collects data about the controlled system and the execution environment. Furthermore, the Monitor component specifies the interaction mode (e.g., push, pull) and the interaction frequency (e.g., time-based, event-based) that starts the control loop. Afterwards, the Analyze component processes the harvested data, so to identify whether adapting the system (e.g., scaling out the number of system resources) can be beneficial for its performance. During this phase, the costs related to the reconfiguration (e.g., due to the migration and/or replication of the resource and its state) should be also taken into account, because as a side effect the reconfiguration could impact negatively on the system performance. For example, too much frequent reconfigurations that require data movement and/or freezing the application can determine a QoS degradation (e.g., in terms of availability).

If some adaptation action is needed, the Plan component is triggered and is responsible for: determining which system component needs to be reconfigured; identifying whether the number of resources (e.g., computing, network, storage) needs to be increased or decreased; and computing the number of resources to be added/removed/migrated and, if required, their new location. As soon as the reconfiguration strategy is computed, the Execute component puts it in action. According to the controlled system, enacting a reconfiguration can be translated, e.g., in updating routing rules, in replicating processing elements, in migrating state information and component code.

When the controlled system is geographically distributed (e.g., Fog computing, distributed Cloud computing) or when it includes a large number of components (e.g., IoT devices, network switches), a single MAPE loop, where decisions are centralized on a single component, may not effectively manage the elasticity. As described by Weyns et al. in [76], different patterns to design multiple MAPE loops have been used in practice by decentralizing the functions of self-adaptation. In this section, we customize the patterns proposed in [76] aiming to provide some guidelines for the development of systems that control the elasticity of geographically distributed resources. The distributed system components running the MAPE loop can be arranged in a hierarchical architecture (Sect. 5.1) or in a flat architecture (Sect. 5.2). In the first case, MAPE loops are organized in a hierarchy, where some control loops supervise the execution of subordinate MAPE loops. In the latter case, MAPE loops are peers one another; as such, they can work autonomously or coordinate their execution by exchanging control messages.

5.1 Hierarchical Patterns

In this section, we present three patterns that organize the MAPE loops in a hierarchy, where a higher-level control loop manages subordinated control loops.

Master-Worker Pattern. When a system includes a large number of components, having a (single) centralized component that performs elasticity decisions might easily become the architecture bottleneck. To overcome this issue, the system can be organized so to decentralize some of the MAPE operations, exploiting the ability of distributed components to run control operations. Nevertheless, the system may need to perform the monitoring and planning operations locally at each distributed component, e.g., because of special equipment, size of exchanged data, specificity of operations. On the other hand, to preserve a consistent view of the system and meet global guarantees while keeping the system simple, the latter can include a centralized entity which coordinates the elasticity decisions. As such, it can easily prevent unneeded reconfigurations or conflicting scaling operations. Differently from a completely centralized approach, this design pattern relieves the burden of the central component, which now oversees only a subset of the MAPE phases, by including and integrating multiple, decentralized control cycles, in charge of performing locally some control activities. Specifically, this pattern is well suited when the distributed entities to be controlled have monitoring and actuating capacity and can change their behavior according to external decisions (e.g., machines in smart manufacturing, SDN devices, Virtual Network Functions).

Pattern: A *master-worker pattern* structures the system in a two-level hierarchical architecture. At the highest level, a single master component oversees the analysis and planning of scaling operations. At the lowest level, multiple independent components run the distributed Monitor and Execute operations. Figure 2 provides a graphical representation of this pattern. Each distributed Monitor component communicates with a centralized Analyze component by providing aggregated (or high-level) information on the nodes, which can be used to steer some elasticity action on the system. Should a scaling operation be performed, the centralized component plans an adaptation strategy, which consists in determining the resources to be scaled and the magnitude of the scaling operation. The planned decision is sent back to the distributed nodes, which will ultimately enact them. Observe that, by centralizing the Analyze and Plan components, this pattern facilitates the implementation of efficient scaling policies that aim at achieving global objectives and guarantees. On the other hand, sending the collected monitoring information to the master component and distributing the subsequent scaling actions may impose a significant communication overhead. Moreover, the centralized component that runs the Analyze and Plan phases may become a bottleneck in case of large-scale distributed systems.

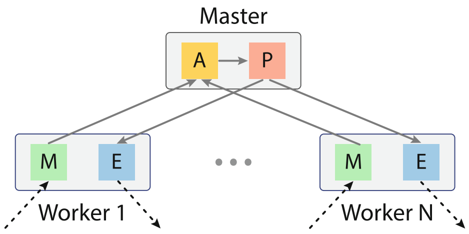

Fig. 2. Hierarchical MAPE: master-worker pattern.

Example: SDN-switches are in charge of forwarding data as requested by the SDN controller. To guarantee performance, a SDN controller can allocate network path to route traffic with specific QoS requirements. For example, a path can be dedicated to a specific data-intensive and latency-sensitive application, or multiple paths can be used in parallel to increase the bandwidth in specific network segments. The allocation of resources can be changed at run-time, by monitoring and analyzing the network, so to plan a strategy for reallocating resources (i.e., network paths). In this setting, an elastic system can include components that realize MAPE control cycles at two different levels. At the lowest level, SDN devices run the Monitor and Execute components of MAPE, whereas at the higher level, the SDN controller runs the Analyze and Plan components. A SDN controller retrieves network information (e.g., link utilization) from distributed SDN-enabled devices. By analyzing this information, the controller can plan to scale network resources, aiming to improve or reduce the bandwidth capacity of a network (logical) path between two communicating devices. To scale the capacity of a network path, the SDN controller can allocate multiple parallel paths to route data. Afterwards, the distributed SDN devices can enact the new forwarding rules, and reroute packets accordingly.

Regional Planning Pattern. A large scale system can be organized in multiple, distributed, and loosely coupled parts (or regions), which cooperate to realize a complex integrated system. Computing and performing scaling decisions on this system might be challenging, because we would like to control elasticity of subsystems within a single region as well as the elasticity of the overall system distributed across multiple regions. Typical scenarios involve federated infrastructures, where networks, Cloud infrastructures, or Cloud platforms should be controlled to realize an elastic system. In this context, scaling operations within regions may aim to optimize resource allocation, while adaptations between regions may optimize load distribution or improve communications under particular conditions. For example, in the Fog environment, an elastic system can improve and reserve fast communications links from resources at the edge of the network to the Cloud, in response to emergency events (e.g., earthquakes, floods, tsunami).

Pattern: In the *regional planning pattern*, represented in Fig. 3, the system is organized in regions. A region has a two-level hierarchical structure, where the top level includes a Plan component (a regional planner), and the lower level includes components performing the four MAPE phases. The regional planner collects the necessary information from the underlying subsystems, so to determine when and how to scale the system components. Moreover, regional planners interact with one another to coordinate adaption actions that span multiple regions. Within each region, the Monitor component observes the region subsystem, the local Analyze component elaborates the collected data and reports the outcomes to the regional planner. Leveraging on the collected information, the latter can plan a scaling operation that involves a single region or that spans across multiple regions. In the latter case, the regional planner might interact with other regional planners to coordinate the scaling operation. Once they agree on a scaling strategy, they can enact the adaptation by activating the Execute components of the respective regions. This pattern is well suited when regions are under different ownership, because the MAPE loop of a region exposes only limited information (i.e., the outcome of the analysis phase), without providing raw data (which result from the monitoring components). Similarly, once the scaling strategy is devised, the region is responsible of enacting the required adaptation actions; as such, the implementation details can be hidden to the regional planner.

Example: In a Fog computing environment, near-edge micro data centers support the execution of distributed applications by providing computing resources near to the users (or to data sources). In a wide area, these micro data centers can be managed by different authorities (e.g., university campus, IT company) and usually expose Cloud-like APIs, which allows to allocate and release micro-computing resources as needed [53]. The combination of Fog and Cloud allows

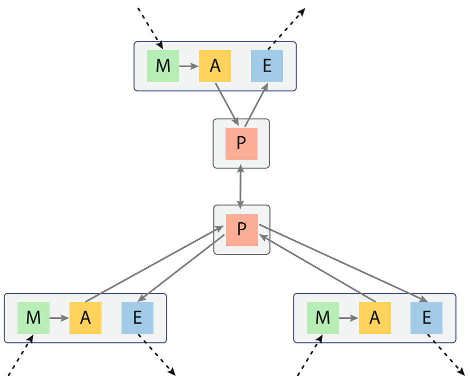

Fig. 3. Hierarchical MAPE: regional pattern.

the provisioning of resources with different computational and network capabilities, thus opening a wide spectrum of approaches to realize system elasticity. For example, a system can be scaled within a region, using multiple resources belonging to the same infrastructure (i.e., Fog, Cloud), or can be scaled across multiple regions, so to take advantage of their different features (e.g., the computational power of Cloud resources, the reduced network delays of near-edge devices). In general, separate Fog/Cloud data centers can be regarded as different regions, possibly under different ownership domains. Within each region, the system runs a MAPE control cycle which comprises only the Monitor, Analyze, and Execute components. Relying on these components, the system can monitor resource utilization as well as incoming workload variations, and trigger scaling operations. In such a case, the regional planner, which can run inside or outside the region (e.g., in the Cloud), is invoked. When the planner determines the scaling strategy, it can decides to offload some computation to other regions (i.e., by possibly acquiring resources in the Cloud) or to change resource allocation within the region under its control.

Hierarchical Control Pattern. When the complexity of a distributed system increases, also controlling its elasticity might involve complex machinery. In this case, a classic approach to rule the system complexity relies on the *divide et impera* principle, according to which the system is split in different subsystems, which can be more easily controlled. To steer the adaptation of the overall system behavior, another control loop coordinates the evolution of each subsystem. The resulting system includes multiple control loops, which work at different time scales and manage different resources or different kinds of resources. In this context, control loops need to interact and coordinate their actions to avoid conflicts and provide certain guarantees about elasticity.

Pattern: The *hierarchical control pattern* provides a layered separation of concerns to manage the elasticity of complex systems. According to this pattern, the adaptation logic is embedded in a hierarchy of MAPE loops. Layers of the hierarchy oversee different concerns at a different level of abstraction and, possibly, by working at a different time scale. Usually, each layer includes a MAPE loop which comprises all the four control steps. However, different sub-patterns can be obtained by customizing the hierarchical MAPE and the way the hierarchical layers interact with one another. As regards the latter, a wide range of opportunities can be elaborated: on the one side, a higher level component works without a direct interaction with lower levels; on the other side, a higher level component (e.g., Monitor) recursively interrogates the lower level components (e.g., Monitors) to perform its tasks. Figure 4 illustrates the hierarchical control pattern, where the Monitor and Execute components strictly cooperate with the lower levels components, whereas the Analyze and Plan components work autonomously for each level. This approach is well suited for a system where multiple but dependent levels of control can be easily identified, such as distributed applications (or services), which are made as a combination of small, elastic building blocks.

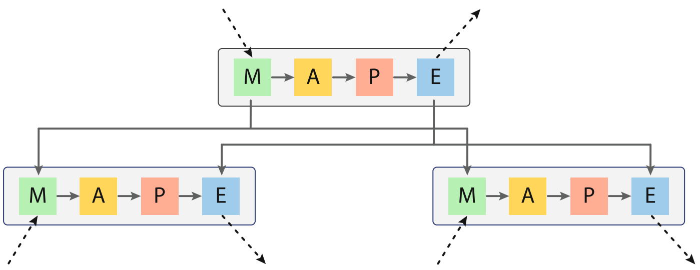

Fig. 4. Hierarchical MAPE: hierarchical control pattern.

Example: Data Stream Processing (DSP) applications are a prominent approach for processing Big Data; indeed, by processing data on-the-fly (i.e., without storing them), they can produce results in a near real-time fashion. A DSP application is represented as directed acyclic graph, where data sources, operators, and final consumers are interconnected by logical links, where data streams flow. Each operator can be regarded as a black-box processing element that receives incoming streams and generates new outgoing streams. To seamlessly process huge amount of data, DSP applications usually exploit data parallelism, which consists in increasing or decreasing the number of instances for the operators [37]. Multiple instances of the same operator can be executed over multiple computing nodes, thus increasing the amount of incoming data processed in parallel.

To control the elasticity of DSP applications in a scalable and distributed manner, a DSP system can include multiple MAPE control loops, organized according to the hierarchical control pattern [17]. We consider a two layered approach with separation of concerns and time scale between layers, where the higher level MAPE loop controls subordinate MAPE components. At the lower level and at a faster time scale, an operator manager is the distributed entity in charge of controlling the replication degree of a single DSP application operator through a local MAPE loop. It monitors the system logical and physical components used by the operator and then, by analyzing the monitored data, determines whether a local operator scaling action is needed. In positive case, the lower-level analyze component issues an operator adaptation request to the higher layer. At the higher level and at a slower time scale, an application manager is the centralized entity that coordinates the elasticity of the overall DSP application through a global MAPE loop. First, it monitors the global application behavior. Then, it analyzes the monitored data and the reconfiguration requests received by the multiple operator managers, so to decide which reconfigurations should be granted. Afterwards, the granted decisions are communicated to each operator manager, which can, finally, execute the operator adaptation actions. The higher level control loop has a more strategic view of the application evolution, therefore it coordinates the scaling operations. Since performing a scaling operation introduces a temporary application downtime, the global MAPE loop limits the number of reconfigurations when they are not needed (e.g., when the application performance requirements are satisfied). Conversely, when the application performance is approaching a critical value (e.g., maximum response time), the global MAPE loop is more willing to grants reconfigurations, so to quickly settle the performance issues.

Such hierarchical design of the elasticity control allows to overcome the system bottleneck represented by the centralized components of the MAPE loop in the master-slave pattern (e.g., see [16] for its application to elastic data stream processing), especially when the system is composed by a multitude of processing entities scattered in a large-scale geo-distributed environment.

5.2 Flat Patterns

We now discuss two patterns that organize the MAPE loops in a flat structure, where multiple control loops cooperate as peers to manage the elasticity of a distributed system. Due to the lack of central coordination, designing a stable scaling strategy is challenging, although the resulting control architecture makes the system highly scalable.

Coordinated Control Pattern. Sometimes controlling the elasticity of a system in a centralized component is unfeasible. Such a lack of feasibility may arise for several reasons, among which the scale of the system and the presence of multiple ownership domains. As regards the former issue, a large scale system makes difficult (or impractical) to quickly move all the monitored data to a single node, which is prone to become the system bottleneck. Nevertheless, in such a context, we still need to develop a system which can control the system elasticity so to meet certain QoS attributes. In this case, multiple MAPE loops can be employed so to control the distributed system. Each control loop supervises one part of the system; the resulting control loops must also coordinate with one another as peers so to reach, if needed, some joint adaptation decision about elasticity.

Pattern: The *coordinated control pattern* employs multiple MAPE loops, which are disseminated within the system. Each loop is in charge of controlling one part of the system. To compute scaling decisions, the phases of each loop can coordinate their operation with corresponding phases of other peer loops. The pattern does not provide regulations on the number of peer loops that should coordinate with one another: in some implementations peers are completely autonomous; in others, the cooperation is restricted to neighbor peers; and in some others all the peers communicate one another. Figure 5 provides a graphical representation of this pattern. For example, the distributed Analyze components exchange information so to determine whether some part of the system needs to perform a scaling decision. Then, after planning the reconfiguration, the distributed Execute components exchange messages to synchronize the adaptation actions, which should be performed without compromising the application integrity.

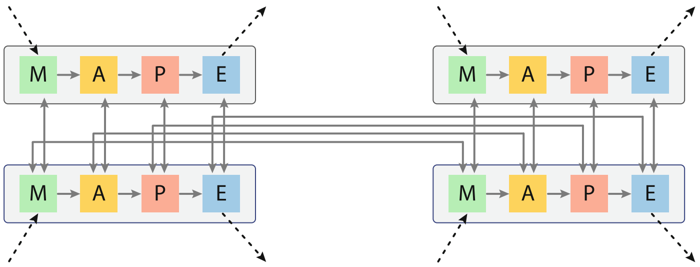

Fig. 5. Flat MAPEs: coordinated control pattern.

Example: This pattern can be useful to control elasticity when the system spans multiple ownership domains with no trustworthy authority to control adaptation. We consider the example of a monitoring application that manages smart power plugs (i.e., a special kind of IoT device) disseminated on multiple cities. We further assume that these IoT devices reside under different authority domains, e.g., one for each city (or neighborhood). To support the proper execution of the monitoring application, the nowadays network and computing infrastructure should adapt itself to support the varying load imposed by the application. Specifically, the IoT devices continuously emit a varying load of data that should be pushed towards the core of the Internet, so to reach Cloud data centers, where the applications extract meaningful information (e.g., predict energy consumption, identify anomalies). The communication between IoT devices and the Cloud is often mediated by IoT gateways, which allow to overcome the heterogeneity of the two parts, in terms of connectivity, energy power, and availability. To properly control this distributed infrastructure, a MAPE control loop can be installed within each authority domain, so to elastically scale the number of resources needed to realize the communication between the involved parties (i.e., smart power plugs, Cloud). In this case, the Monitor component of the MAPE loop collects data on the working conditions of IoT devices. These data are analyzed so to determine whether new IoT gateways should be allocated to meet the application requirements. Since allocating a gateway imposes a monetary cost, the multiple MAPE loops can coordinate their action so to limit the execution costs and do not exceed the allocated budget. Ultimately, when a scaling action is granted, the Execute component starts a new IoT gateway on the authority domain specified by the Plan component.

Information Sharing Pattern. Some large scale systems comprise distributed peers which cooperatively work to accomplish tasks. In particular, each peer is able of performing some tasks (e.g., it offers services), but could require an interaction with other peers to carry out these tasks (e.g., to solve service dependencies). Examples of this scenario come from the pervasive computing domain like ambient intelligence or smart transportation systems, where peers work together to reach some common goals. Each distributed peer can locally take scaling decisions. Nevertheless, since a local adaptation may influence the other system components, taking scaling decisions require some form of coordination that can be reached by sharing information among system components.

Pattern: The *information sharing pattern* is a special case of coordinated control pattern, where the interaction between the decentralized MAPE control loops involves only the Monitor phase (see Fig. 6). The pattern does not strictly regulate the way peers interact with one another: for example, when the system comprises a large number of peers, only a subset of them (i.e., neighbors) exchange monitoring information. The following MAPE phases operate on (approximately) the same view of the system, thus allowing the Analyze, Plan, and Execute phases to be performed locally. On the one hand, this pattern helps to realize scalable and elastic systems. On the other hand, since there is no explicit coordination among peers (i.e., they operate autonomously), conflicting or sub-optimal scaling actions can be enacted; in the worst case, the system enters in an unstable state, where adaptation actions are continuously applied.

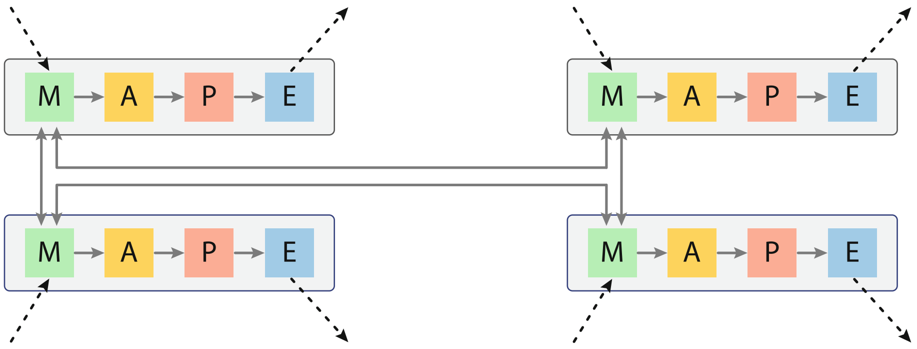

Fig. 6. Flat MAPEs: information sharing pattern.

Example: Relying on this pattern, the system can elastically acquire and release resources in a fully decentralized manner, leveraging only on monitoring information which is shared among the distributed controllers. We consider the problem of executing long-running workflow of services in a fully decentralized system. The system comprises peers that run and expose some services. A peer can receive requests of service workflow execution; a service workflow is a graph of abstract services (i.e., definition of required services) that needs to be resolved in a set of concrete services (i.e., implementation of abstract services). To realize the service choreography, each peer needs to discover the services offered by other peers, together with their utilization level, so to determine the best mapping that satisfy the workflow requirements (e.g., minimum response time, maximum throughput). Similarly to the approach presented in [15], the system can employ the information sharing pattern to share, among peers, knowledge about the services offered by peers and their utilization state. Relying on this information, at run-time, the service choreography can be adapted so to automatically scale the number of concrete services to be used to run the workflow. Aside the shared monitoring information, the scaling decisions are performed locally to each peer.

6 Challenges and Future Perspectives

Although many research efforts have investigated how to efficiently achieve elasticity, most of them relies on a centralized Cloud environment. With the diffusion of the edge/fog computing, we have witnessed a paradigm shift with the execution of complex systems over distributed Cloud and edge/fog computing resources, which brings system components closer to the data, rather than the other way around. This new environment, which offers geo-distributed resources, promises to open new possibilities for realizing elasticity, thanks to the cooperation of computing resources at different levels of the overall infrastructure (i.e., at the network edge and in the network core). Nevertheless, the full potentialities, together with the challenges, of this distributed edge cloud environments are still, to the best of our knowledge, largely unexplored.

We identify several main challenges and research directions that could benefit from further investigation, so to bring improvements to the current state of the art.

Strategies for Decentralization. Thanks to their widespread adoption, IoT devices act as geo-distributed data sources that continuously emit data. The most diffused approaches for processing these data rely on a centralized Cloud solution, where all the data are collected in a single data center, processed, and ultimately sent back to (possibly distributed) information consumers. Although the Cloud offers flexibility and elasticity, such a centralized solution is not well suited to efficiently handle the increased demand of real-time, low-latency services with distributed IoT sources and consumers. As envisioned by the convergence of edge/fog and Cloud computing resources, this diffused environment can support the execution of distributed applications by increasing scalability and reducing communication latencies. Nevertheless, in this context, computing resources are often heterogeneous and, most importantly, can be interconnected by network links with not negligible communication latencies. In this geo-distributed infrastructure, a relevant problem consists in determining, within a set of available distributed computing resources, the ones that should execute each component of the distributed application, aiming to optimize the application QoS attributes. Nevertheless, most of the existing deployment solutions have been designed to work in centralized environment, where network latencies are (almost) zero. As such, these policies cannot be easily adapted to run in the emerging environment. To the best of our knowledge, efficient approaches to deploy applications in distributed hybrid edge Cloud environments are still largely unexplored.

Infrastructure-awareness. The convergence of distributed edge/fog environments with Cloud environments results in a great variety of resources, whose peculiar features can be exploited to perform specific tasks. For example, resources located at the network edges, which are usually characterized by a medium-low computing capacity and possibly limited battery, can be used by a monitoring system to filter and aggregate raw data as soon as they are generated. Conversely, clusters of specialized machines (e.g., [1]) can be exploited to efficiently perform machine learning tasks. Most of the existing distributed systems, which manage data coming from decentralized sources, are infrastructure oblivious, i.e., their deployment neglects the peculiar characteristics of the available computing and networking infrastructure. In the IoT context, where huge amount of data have to be moved between geo-distributed resources, inefficient exploitation of resources can strongly penalize the resulting performance of distributed applications. To deliver efficient and flexible solutions, next generation systems should consider, as key factor, the physical connection and the relationship among infrastructural elements.

Elasticity in the Emerging Environment. The combination of edge/fog and Cloud computing results in a hierarchical architecture, where multiple layers are spread as a continuum from the edge to the core of the network. The presence of multiple layers, each with different computational capabilities, opens a wide spectrum of approaches for realizing elasticity. For example, we could scale horizontally the application components, so to use multiple resources that belong to the same infrastructural layer (i.e., edge, Cloud); alternatively, we could employ resources belonging to multiple layers, so to take advantage of their different features (e.g., the computational power of Cloud servers, the closeness of edge devices). Moreover, the presence of multiple degrees of freedom raises concerns regarding the coordination among the different scaling operations. When is it more convenient to use resources from the same layer? When should we employ resources from multiple layers? Can communication delays obfuscate the benefit of operating with resources belonging to multiple layers?

The Cost of Elasticity. Reconfiguring an application at runtime involves the execution of management operations that enact the deployment changes while preserving the application integrity. The latter is a critical task especially when the application includes components that cannot be simply restarted on a new location, but require, e.g., to export and import on the new location some internal state. Therefore, together with long term benefits, adapting the application deployment also introduces some adaptation costs, usually expressed in terms of downtime, that penalize the application performance in the short period. Because of these costs, reconfigurations cannot be applied too frequently. A key challenge is to wisely select the most profitable adaptation actions to enact, so to identify a suitable trade-off, in terms of performance, between application elasticity and adaptation costs.

Multi-dimensional Elasticity. Besides resource elasticity, we can identify different elasticity dimensions, as envisioned by Truong et al. [72]. Examples of other dimensions are cost, data, and fault tolerance. Indeed, during the execution of a complex distributed system, the cost of using computing resources or the benefits coming from the output of the system may change at runtime. Similarly, the quality of data can be elastically managed, in a such a way that when it is too expansive to produce results with high quality, we can tune the system to temporary degrade result quality, in a controlled manner, so to save resources. For example, this could be helpful during congestion periods, when we might accept to discard a wisely selected subset of the incoming data. As regards fault tolerance, for some kinds of applications, we might be willing to sacrifice fault tolerance during congestion periods so to perform computation with reduced costs. As expected, finding an optimal trade-off between the different elasticity dimensions strongly depends on the application at hand and, in general, is not an easy task.

Resource Management. The resulting infrastructure is complex: multiple heterogeneous resources are available at different geo-distributed locations; distributed applications expose different QoS attributes and requirements; and different elasticity dimensions can be controlled. Moreover, the elastic adaptation of applications might require infrastructure-awareness, that enables to conveniently operate at different levels of the computing infrastructure.

To rule this complexity, a new architectural layer should be designed so to support the execution of (multiple) applications over a continuum set of edge/fog and Cloud resources. This intermediate layer can be implemented as a distributed resource manager, which should be able to efficiently control the allocation of computing and network resources, by conveniently exposing different views of the infrastructure. On the one hand, the resource manager allows to fairly execute multiple applications by better exploiting the presence of resources. On the other hand, by taking care of managing the computing infrastructure, it enables distributed applications to more easily control their elasticity.

A side effect of the introduction of a resource manager is the need of designing standardized interfaces between the applications and the decentralize resources. To the best of our knowledge, today there are no standard mechanisms that allow resources to announce their availability to host software components as well as for distributed applications to smoothly control edge/fog and Cloud resources.

Accountability, Monitoring, and Security. Together with the specific challenges previously identified, we have several other more general challenges.

They regard the accountability of resource consumption, the monitoring of elastic applications/systems, and security aspects that arise from multi-tenancy and data distribution across several locations.

We need to investigate methodologies for the accountability, because in the envisioned edge/fog computing environment, users can flexibly share their spare resources to host applications. The hybrid resource continuum from edge/fog to Cloud calls for studying dynamic pricing mechanisms, similar to the spot instance pricing from Amazon EC2 service[2].

The ability of monitoring the elasticity of a system/application deployed in a large-scale, dispersed and multi-provider hybrid environment requires investigation. How to quantify and measure the elasticity of a complex distributed system? As regards elasticity, we can quantify its performance by considering the number of missing or superfluous adaptations over time, the durations in sub-optimal states, and the amount of over-/under-provisioned resources [35]. However, how to measure such quantities in a dispersed, large-scale environment with multiple providers turns out to be challenging.

Similarly to Cloud computing, we need to identify (or develop) efficient business models that support and encourage the diffusion of trusted computing resources and the elasticity requirements for such business models. One of the most important challenge arises from the lack of central controlling authorities in the edge/fog computing environment, which makes it difficult to assert whether a device is hosting an application component. Security aspects are of key importance, because nowadays the value of data is very high and an infrastructure that does not guarantee stringent security properties will be hardly adopted. Similarly for the accountability issue, the decentralization of the emerging environment requires to deal with the lack of a central security authority. Sophisticated yet lightweight security mechanisms and policies should be introduced, so to create a disseminated trustworthy environment.

7 Conclusions

In this chapter, we presented an analysis of QoS-based elasticity for service chains in distributed edge Cloud environments. Firstly, we introduced the elasticity concept that arises in emerging systems of systems, which are complex, distributed, and based on various virtualization technologies. Then, we focused on IoT and Cloud systems, in whose context we elaborated the need and meaning of elasticity.

A key ingredient of elasticity is the optimization technique aiming to optimize some QoS attributes. Firstly, we identified the key attributes that are frequently optimized with elasticity. Then, we introduced a software engineering viewpoint to model elasticity as one of the system attributes. In that respect, elasticity mechanisms can be implemented in the system design phase to model software systems that exploit at best elasticity during runtime. Furthermore, elasticity

[2] https://aws.amazon.com/ec2/.

involves a runtime choice for the best optimal solution and such a selection has also to be properly designed. Therefore, we reviewed the research works on modeling elasticity in the context of design and runtime choices aiming to provide the best elasticity model and optimal solution.

In distributed environments, elasticity mechanisms may arise not only at different layers of system abstraction, but also within each segment of the distributed system that, as a whole, has to deliver service to the end users. Therefore, key elements for running QoS-aware service compositions are the coordination mechanisms; the latter have to be efficiently implemented in order to deliver high-level user-experience. In this chapter, we also provided a review of several design patterns for decentralized coordination, aiming to realize elasticity in complex systems.

Finally, we discussed the challenges related to designing elasticity mechanisms in geo-distributed environments. Software engineering decisions and coordination mechanisms among segments of distributed systems need further investigation based on empirical evidence from the real technical environments.

References

1. Abadi, M., Barham, P., Chen, J., Chen, Z., et al.: TensorFlow: a system for large-scale machine learning. In: Proceedings of USENIX OSDI 2016, pp. 265–283 (2016)
2. Al-Dhuraibi, Y., Paraiso, F., Djarallah, N., Merle, P.: Elasticity in cloud computing: state of the art and research challenges. IEEE Trans. Serv. Comput. **PP**, 1 (2017). https://doi.org/10.1109/TSC.2017.2711009
3. Aleti, A., Björnander, S., Grunske, L., Meedeniya, I.: ArcheOpterix: an extendable tool for architecture optimization of AADL models. In: Proceedings of 2009 ICSE Workshop on Model-Based Methodologies for Pervasive and Embedded Software, pp. 61–71 (2009)
4. Aleti, A., Buhnova, B., Grunske, L., Koziolek, A., Meedeniya, I.: Software architecture optimization methods: a systematic literature review. IEEE Trans. Softw. Eng. **39**(5), 658–683 (2013)
5. Andrikopoulos, V., Reuter, A., Gómez Sáez, S., Leymann, F.: A GENTL approach for cloud application topologies. In: Villari, M., Zimmermann, W., Lau, K.-K. (eds.) ESOCC 2014. LNCS, vol. 8745, pp. 148–159. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-44879-3_11
6. Andrikopoulos, V., Gómez Sáez, S., Leymann, F., Wettinger, J.: Optimal distribution of applications in the cloud. In: Jarke, M., Mylopoulos, J., Quix, C., Rolland, C., Manolopoulos, Y., Mouratidis, H., Horkoff, J. (eds.) CAiSE 2014. LNCS, vol. 8484, pp. 75–90. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-07881-6_6
7. Arcelli, D., Cortellessa, V., Trubiani, C.: Antipattern-based model refactoring for software performance improvement. In: Proceedings of ACM SIGSOFT QoSA 2012, pp. 33–42 (2012)
8. Arcelli, D., Cortellessa, V., Trubiani, C.: Performance-based software model refactoring in fuzzy contexts. In: Egyed, A., Schaefer, I. (eds.) FASE 2015. LNCS, vol. 9033, pp. 149–164. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46675-9_10
9. Ardagna, D., Casale, G., Ciavotta, M., Pérez, J.F., Wang, W.: Quality-of-service in cloud computing: modeling techniques and their applications. J. Int. Serv. Appl. **5**(1), 11 (2014). https://doi.org/10.1186/s13174-014-0011-3

10. Bass, L., Clements, P., Kazman, R.: Software Architecture in Practice, 3rd edn. Addison-Wesley Professional, Reading (2012)
11. Becker, S., Koziolek, H., Reussner, R.: The palladio component model for model-driven performance prediction. J. Syst. Softw. **82**(1), 3–22 (2009)
12. Bernstein, D.: Containers and cloud: from LXC to Docker to Kubernetes. IEEE Cloud Comput. **1**(3), 81–84 (2014)
13. Bratterud, A., Walla, A.A., Haugerud, H., Engelstad, P.E., Begnum, K.: IncludeOS: a minimal, resource efficient unikernel for cloud services. In: Proceedings of IEEE CloudCom 2015, pp. 250–257 (2015)
14. Calinescu, R., Grunske, L., Kwiatkowska, M., Mirandola, R., Tamburrelli, G.: Dynamic QoS management and optimization in service-based systems. IEEE Trans. Soft. Eng. **37**(3), 387–409 (2011)
15. Caporuscio, M., D'Angelo, M., Grassi, V., Mirandola, R.: Reinforcement learning techniques for decentralized self-adaptive service assembly. In: Aiello, M., Johnsen, E.B., Dustdar, S., Georgievski, I. (eds.) ESOCC 2016. LNCS, vol. 9846, pp. 53–68. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-44482-6_4
16. Cardellini, V., Lo Presti, F., Nardelli, M., Russo Russo, G.: Optimal operator deployment and replication for elastic distributed data stream processing. Concurr. Comput. (2017). https://doi.org/10.1002/cpe.4334
17. Cardellini, V., Lo Presti, F., Nardelli, M., Russo Russo, G.: Towards hierarchical autonomous control for elastic data stream processing in the fog. In: Heras, D.B., Bougé, L. (eds.) Euro-Par 2017. LNCS, vol. 10659, pp. 106–117. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-75178-8_9
18. Casale, G., Ardagna, D., Artac, M., Barbier, F., et al.: DICE: quality-driven development of data-intensive cloud applications. In: Proceedings of 7th International Workshop on Modeling in Software Engineering, pp. 78–83. IEEE Press (2015)
19. Ciavotta, M., Ardagna, D., Gibilisco, G.P.: A mixed integer linear programming optimization approach for multi-cloud capacity allocation. J. Syst. Softw. **123**, 64–78 (2017)
20. Ciavotta, M., Ardagna, D., Koziolek, A.: Palladio optimization suite: QoS optimization for component-based cloud applications. In: Proceedings of 9th EAI International Conference on Performance Evaluation Methodologies and Tools, pp. 170–171 (2016)
21. Ciavotta, M., Gianniti, E., Ardagna, D.: D-SPACE4Cloud: a design tool for big data applications. In: Carretero, J., Garcia-Blas, J., Ko, R.K.L., Mueller, P., Nakano, K. (eds.) ICA3PP 2016. LNCS, vol. 10048, pp. 614–629. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-49583-5_48
22. Copil, G., Moldovan, D., Truong, H.L., Dustdar, S.: Continuous elasticity: design and operation of elastic systems. it-Inf. Technol. **58**(6), 329–348 (2016)
23. Coutinho, E.F., de Carvalho Sousa, F.R., Rego, P.A.L., Gomes, D.G., de Souza, J.N.: Elasticity in cloud computing: a survey. Ann. Telecomm. **70**(7), 289–309 (2015)
24. de Lemos, R., et al.: Software engineering for self-adaptive systems: a second research roadmap. In: de Lemos, R., Giese, H., Müller, H.A., Shaw, M. (eds.) Software Engineering for Self-Adaptive Systems II. LNCS, vol. 7475, pp. 1–32. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-35813-5_1
25. Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.: A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Trans. Evol. Comput. **6**(2), 182–197 (2002)
26. Dragoni, N., Lanese, I., Larsen, S.T., Mazzara, M., Mustafin, R., Safina, L.: Microservices: how to make your application scale. CoRR abs/1702.07149 (2017)

27. Dustdar, S., Guo, Y., Satzger, B., Truong, H.L.: Principles of elastic processes. IEEE Int. Comput. **15**(5), 66–71 (2011)
28. Etxeberria, L., Trubiani, C., Cortellessa, V., Sagardui, G.: Performance-based selection of software and hardware features under parameter uncertainty. In: Proceedings of ACM QoSA 2014, pp. 23–32. ACM (2014)
29. Evangelinou, A., Ciavotta, M., Ardagna, D., Kopaneli, A., Kousiouris, G., Varvarigou, T.: Enterprise applications cloud rightsizing through a joint benchmarking and optimization approach. Future Gener. Comput. Syst. **78**, 102–114 (2018)
30. Feiler, P., Gluch, D., Hudak, J.: The architecture analysis and design language (AADL): an introduction. Technical report. CMU/SEI-2006-TN-011, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA (2006)
31. Franceschelli, D., Ardagna, D., Ciavotta, M., Di Nitto, E.: Space4cloud: a tool for system performance and costevaluation of cloud systems. In: Proceedings of 2013 International Workshop on Multi-cloud Applications and Federated Clouds, pp. 27–34. ACM (2013)
32. de Gooijer, T., Jansen, A., Koziolek, H., Koziolek, A.: An industrial case study of performance and cost design space exploration. In: Proceedings of ACM/SPEC ICPE 2012, pp. 205–216 (2012)
33. Grieco, L.A., Colucci, S., Mongiello, M., Scandurra, P.: Towards a goal-oriented approach to adaptable re-deployment of cloud-based applications. In: Proceedings of CLOSER 2016, pp. 253–260. SciTePress (2016)
34. Guerriero, M., Ciavotta, M., Gibilisco, G.P., Ardagna, D.: A model-driven DevOps framework for QoS-aware cloud applications. In: Proceedings of SYNASC 2015, pp. 345–351. IEEE (2015)
35. Herbst, N., Becker, S., Kounev, S., Koziolek, H., Maggio, M., Milenkoski, A., Smirni, E.: Metrics and benchmarks for self-aware computing systems. Self-Aware Computing Systems, pp. 437–464. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-47474-8_14
36. Herbst, N.R., Kounev, S., Reussner, R.H.: Elasticity in cloud computing: what it is, and what it is not. In: Proceedings of 10th International Conference on Autonomic Computing, ICAC 2013, pp. 23–27 (2013)
37. Hirzel, M., Soulé, R., Schneider, S., Gedik, B., Grimm, R.: A catalog of stream processing optimizations. ACM Comput. Surv. **46**(4), 46:1–46:34 (2014)
38. Huebscher, M.C., McCann, J.A.: A survey of autonomic computing - degrees, models, and applications. ACM Comput. Surv. **40**(3), 7:1–7:28 (2008)
39. Hwang, C., Yoon, K.: Multiple criteria decision making. Lecture Notes in Economics and Mathematical Systems. Springer, New York (1981)
40. Jain, R.: The Art of Computer Systems Performance Analysis, vol. 491. Wiley, New York (1991)
41. Kistowski, J.V., Herbst, N.R., Kounev, S.: Modeling variations in load intensity over time. In: Proceedings of 3rd International Workshop on Large Scale Testing, LT 2014. ACM (2014)
42. Koziolek, A.: Automated Improvement of Software Architecture Models for Performance and Other Quality Attributes. Ph.D. thesis, Karlsruhe Institute of Technology (2011)
43. Koziolek, A., Koziolek, H., Reussner, R.: PerOpteryx: automated application of tactics in multi-objective software architecture optimization. In: Proceedings of ACM SIGSOFT QoSA-ISARCS 2011, pp. 33–42 (2011)
44. Le, D., Narendra, N.C., Truong, H.L.: HINC - harmonizing diverse resource information across IoT, network functions, and clouds. In: Proceedings of 4th International Conference on Future Internet of Things and Cloud, FiCloud 2016, pp. 317–324 (2016)

45. Li, H., Casale, G., Ellahi, T.N.: SLA-driven planning and optimization of enterprise applications. In: Proceedings of 1st Joint WOSP/SIPEW International Conference on Performance Engineering, pp. 117–128. ACM (2010)
46. Lorido-Botran, T., Miguel-Alonso, J., Lozano, J.: A review of auto-scaling techniques for elastic applications in cloud environments. J. Grid Comput. **12**(4), 559–592 (2014)
47. Malek, S., Medvidovic, N., Mikic-Rakic, M.: An extensible framework for improving a distributed software system's deployment architecture. IEEE Trans. Software Eng. **38**(1), 73–100 (2012)
48. Menascé, D.A., Ewing, J.M., Gomaa, H., Malex, S., Sousa, J.A.P.: A framework for utility-based service oriented design in SASSY. In: Proceedings of 1st Joint WOSP/SIPEW International Conference on Performance Engineering, pp. 27–36. ACM (2010)
49. Moldovan, D., Truong, H.L., Dustdar, S.: Cost-aware scalability of applications in public clouds. In: Proceedings of IEEE IC2E 2016, pp. 79–88 (2016)
50. Nambiar, M., Kattepur, A., Bhaskaran, G., Singhal, R., Duttagupta, S.: Model driven software performance engineering: current challenges and way ahead. ACM SIGMETRICS Perform. Eval. Rev. **43**(4), 53–62 (2016)
51. Nanda, S., Hacker, T.J., Lu, Y.H.: Predictive model for dynamically provisioning resources in multi-tier web applications. In: Proceedings of IEEE CloudCom 2016, pp. 326–335 (2016)
52. Neto, P.A.S., Vargas-Solar, G., da Costa, U.S., Musicante, M.A.: Designing service-based applications in the presence of non-functional properties: a mapping study. Inf. Softw. Technol. **69**, 84–105 (2016)
53. OpenFog Consortium: OpenFog reference architecture (2017). https://www.openfogconsortium.org/ra/
54. Orsini, G., Bade, D., Lamersdorf, W.: Computing at the mobile edge: designing elastic android applications for computation offloading. In: Proceedings of 8th IFIP Wireless and Mobile Networking Conference, WMNC 2015, pp. 112–119, October 2015
55. Pahl, C., Lee, B.: Containers and clusters for edge cloud architectures-a technology review. In: Proceedings of FiCloud 2015, pp. 379–386. IEEE (2015)
56. Passacantando, M., Ardagna, D., Savi, A.: Service provisioning problem in cloud and multi-cloud systems. INFORMS J. Comput. **28**(2), 265–277 (2016)
57. Pérez, J.F., Casale, G.: Assessing SLA compliance from Palladio component models. In: Proceedings of SYNASC 2013, pp. 409–416 (2013)
58. Perez-Palacin, D., Mirandola, R., Merseguer, J.: On the relationships between qos and software adaptability at the architectural level. J. Syst. Softw. **87**, 1–17 (2014)
59. Quadri, I.R., Gamatié, A., Boulet, P., Meftali, S., Dekeyser, J.L.: Expressing embedded systems configurations at high abstraction levels with UML MARTE profile: advantages, limitations and alternatives. J. Syst. Architect. **58**(5), 178–194 (2012)
60. Rahmoun, S., Borde, E., Pautet, L.: Automatic selection and composition of model transformations alternatives using evolutionary algorithms. In: Proceedings of 2015 European Conference on Software Architecture Workshops, ECSAW 2015, pp. 25:1–25:7. ACM (2015)
61. Sáez, S.G., Andrikopoulos, V., Leymann, F., Strauch, S.: Towards dynamic application distribution support for performance optimization in the cloud. In: Proceedings of IEEE CLOUD 2014, pp. 248–255 (2014)

62. Sáez, S.G., Andrikopoulos, V., Wessling, F., Marquezan, C.C.: Cloud adaptation and application (re-)distribution: bridging the two perspectives. In: Proceedings of IEEE 18th International Enterprise Distributed Object Computing Conference Workshops and Demonstrations, pp. 163–172 (2014)
63. Salehie, M., Tahvildari, L.: Self-adaptive software: landscape and research challenges. ACM Trans. Auton. Adapt. Syst. **4**(2), 1–42 (2009)
64. Sarkar, S., Chatterjee, S., Misra, S.: Assessment of the suitability of fog computing in the context of internet of things. IEEE Trans. Cloud Comput. **PP**, 1 (2015)
65. Schatzberg, D., Appavoo, J., Krieger, O., Van Hensbergen, E.: Scalable elastic systems architecture. In: Proceedings of ASPLOS RESoLVE Workshop, March 2011
66. Shoaib, Y., Das, O.: Web application performance modeling using layered queueing networks. Electr. Notes Theor. Comput. Sci. **275**, 123–142 (2011)
67. Szabo, R., Kind, M., Westphal, F.J., Woesner, H., Jocha, D., Csaszar, A.: Elastic network functions: opportunities and challenges. IEEE Netw. **29**(3), 15–21 (2015)
68. Tofan, D., Galster, M., Avgeriou, P., Schuitema, W.: Past and future of software architectural decisions-a systematic mapping study. Inf. Softw. Technol. **56**(8), 850–872 (2014)
69. Tribastone, M.: Efficient optimization of software performance models via parameter-space pruning. In: Proceedings of ACM/SPEC ICPE 2014, pp. 63–73 (2014)
70. Truong, H.L., Dustdar, S.: Principles for engineering IoT cloud systems. IEEE Cloud Comput. **2**(2), 68–76 (2015)
71. Truong, H.L., Dustdar, S.: Programming elasticity in the cloud. IEEE Comput. **48**(3), 87–90 (2015)
72. Truong, H.L., Dustdar, S., Leymann, F.: Towards the realization of multi-dimensional elasticity for distributed cloud systems. In: Proceedings of 2nd International Conference on Cloud Forward, pp. 14–23 (2016). https://doi.org/10.1016/j.procs.2016.08.276
73. Truong, H.L., Narendra, N.C.: SINC - an information-centric approach for end-to-end IoT cloud resource provisioning. In: Proceedings of International Conference on Cloud Computing Research and Innovations, ICCCRI 2016, pp. 17–24 (2016)
74. Walker, M., Reiser, M.O., Tucci-Piergiovanni, S., Papadopoulos, Y., et al.: Automatic optimisation of system architectures using EAST-ADL. J. Syst. Softw. **86**(10), 2467–2487 (2013)
75. Wettinger, J., Görlach, K., Leymann, F.: Deployment aggregates-a generic deployment automation approach for applications operated in the cloud. In: Proceedings of IEEE 18th International Conference on Enterprise Distributed Object Computing Workshops and Demonstrations, EDOCW 2014, pp. 173–180 (2014)
76. Weyns, D., et al.: On patterns for decentralized control in self-adaptive systems. In: de Lemos, R., Giese, H., Müller, H.A., Shaw, M. (eds.) Software Engineering for Self-Adaptive Systems II. LNCS, vol. 7475, pp. 76–107. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-35813-5_4
77. Wu, W., Kelly, T.: Towards evidence-based architectural design for safety-critical software applications. In: de Lemos, R., Gacek, C., Romanovsky, A. (eds.) WADS 2006. LNCS, vol. 4615, pp. 383–408. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-74035-3_17
78. Yoder, J.W., Johnson, R.: The adaptive object-model architectural style. In: Bosch, J., Gentleman, M., Hofmeister, C., Kuusela, J. (eds.) Software Architecture. ITI-FIP, vol. 97, pp. 3–27. Springer, Boston, MA (2002). https://doi.org/10.1007/978-0-387-35607-5_1