Fog Computing and Aggregate Programming for Distributed Coordination

Lorenzo Rosa
Collegio Superiore
Laurea magistrale in Ingegneria Informatica

Introduction

The exponential increase of the diffusion of smart, interconnected devices (estimated to reach 50 billion units by 2020[1]) is powered by the proliferation of mobile devices (e.g. mobile phones and tablets) and smart sensors. Since we need new concepts and technologies to manage this growing number of Internet of Things (IoT) devices, industry and academia have proposed several solutions based on cloud computing for service delivery in the Internet of Things (Toll, 2014). Nevertheless, cloud-based solutions have some limitations in the considered IoT scenarios, such as scalability, high demand for bandwidth and high latency. In order to address such issues, the fog computing model has been developed as an extension of cloud computing, specifically designed as a layered model for enabling ubiquitous access to a shared continuum of scalable computing resources (Bonomi, Milito, Zhu, & Addepalli, 2012; Iorga et al., 2018).

However, even fog computing solution are typically tied to a device-centric programming model, which tend not to scale well in scenarios where a huge number of heterogeneous devices is free to move unpredictably in the environment. Aggregation programming face this challenge from a different programming perspective: the computational devices spread in the pervasive computing environment are considered as a single "machine" to program: you do not program the single device, but a whole region, namely a collection of devices which cooperate using resilient coordination mechanisms and proximity-based interactions. Following this approach, based on *field calculus* for coordination (Viroli, Damiani, & Beal, 2013), aggregate computing succeeds in defining an abstraction layer on top of large scale ubiquitous systems (Viroli, Casadei, & Pianini, 2016; Beal, Pianini, & Viroli, 2015).

Cloud and Fog computing

According to NIST, cloud computing is a computing model enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction (Mell & Grance, 2011). The enormous and ever-increasing interest of enterprise IT operators in cloud computing is motivated by the huge advantages they gain from it. Indeed, the properties of instant availability and elasticity of computational resources allow organizations to save a significant amount of economic resources.

Fog computing[2] "extends the Cloud Computing paradigm to the edge of the network, thus enabling a new breed of applications and services". It focuses specifically on IoT scenarios: the name "fog" itself refers to "a cloud close to the ground" (Bonomi et al., 2012). Indeed, the model is conceived as a platform for several IoT services and applications: it facilitates the deployment of distributed, latency-aware applications and services, and consists of fog nodes[3], positioned between smart end-devices and centralised (cloud) services (Iorga et al., 2018). Without the aspiration of giving a complete overview of this model, in the following we try to sum up the reasons behind its development and some of its key features.

First of all, fog computing addresses location-aware applications with constraints of low latency, which cannot tolerate the delays of the communication with a centralised cloud: indeed, they are often characterised by a strong presence of streaming, with the consequent real time and quality requirements (e.g. gaming, video streaming, augmented reality, etc.). These constraints were at the origins of the early proposals of this paradigm, aiming at supporting endpoints with rich services at the edge of the network.

[1] https://www.cisco.com/c/dam/en_us/about/ac79/docs/innov/IoT_IBSG_0411FINAL.pdf

[2] Initially proposed by Cisco (Bonomi et al., 2012), the fog computing model (Fog) has recently undergone a standardisation effort inside the Open Fog Consortium (*OpenFog Consortium*, 2017). In March, 2018 the National Institute of Standards and Technology (NIST) released a special publication where it gives a definition of fog computing (Iorga et al., 2018).

[3] Fog computing particularly emphasises virtualisation techniques, so fog nodes are typically virtual, such as virtualized switches, virtual machines, cloudlets, etc. However, they can be also physical devices.

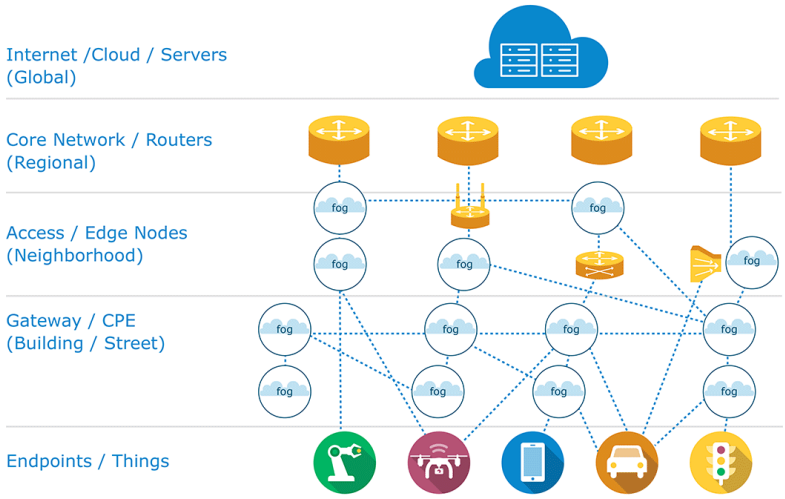

Figure 1. Layered architecture of fog computing. Image from the website of the OpenFog consortium.

Secondly, the geographical distribution of the targeted applications and services is wide-spread, in sharp contrast to the more centralised cloud. Strictly related to that, it is essential to have a support for mobility: applications have to communicate directly with mobile devices. For instance, a clear example of a possible application of the Fog is the delivering of high quality streaming to moving vehicles, through proxies and access points positioned along highways and tracks.

In addition, all these requirements have to be matched for a very large number of heterogeneous nodes in a wide variety of environments, which access the Internet mainly through wireless endpoints.

Finally, one of the main features that the model should present is the scalability and the agility of federated, fog-node clusters. Fog computing is adaptive in nature, at cluster or cluster-of-clusters level, supporting elastic compute, resource pooling, data-load changes, and network condition variations.

Therefore, Fog computing is not an alternative to cloud computing, but rather an extension of it. Indeed, while Fog nodes provide localisation, therefore enabling low latency and context awareness, the Cloud provides global centralisation. Many applications require both Fog localisation and cloud globalisation, particularly for analytics and Big Data. (Bonomi et al., 2012).

From an architectural point of view, fog computing could be described as "a system-level horizontal architecture that distributes resources and services of computing, storage, control and networking anywhere along the continuum from Cloud to Things" (*OpenFog Consortium*, 2017). More specifically, it is said that it has a horizontal architecture because it supports multiple industry verticals and application domains. Services and applications can be distributed not only close to Things, but "anywhere along the continuum between Cloud and Things". Furthermore, it is a system-level model because it "extends from the Things, over the network edges, through the Cloud, and across multiple protocol layers – not just radio systems, or a specific protocol layer, or one part of an end-to-end system, but a system spanning between the Things and the Cloud.

Aggregate Programming

Even with the introduction of specific computational models, the currently most diffused programming models for mobile/IoT applications consider the individual device as the basic unit of computation. Therefore, for instance, such applications are structured client-side as single-device programs, which leverage device-centric interactions/protocols (using APIs for MoM/SOA/ad-hoc-communications). However, when such applications grow in complexity, they tend not to scale well in environments where a huge number of heterogeneous devices can move unpredictably. For instance, they suffer from design problems, since a cloud-oriented designed server (needed for scalability) is not really orthogonal to the whole programming model, but often affects system design in a significant way. One of the main reasons of those issues is that the considered environment is not considered as a "computational system" to program.

This new approach is the main idea behind aggregate programming, which could be defined as a paradigm that models large-scale adaptive systems, shifting the focus from the perspective of a single component to the whole aggregate. With this technique, the basic unit of computing is no longer a single device but instead a cooperating collection of devices: details of their behaviour, position, and number are largely abstracted away, replaced with a space-filling computational environment (Beal et al., 2015). Therefore, aggregate computing proposes an "aggregate viewpoint, in which one sees

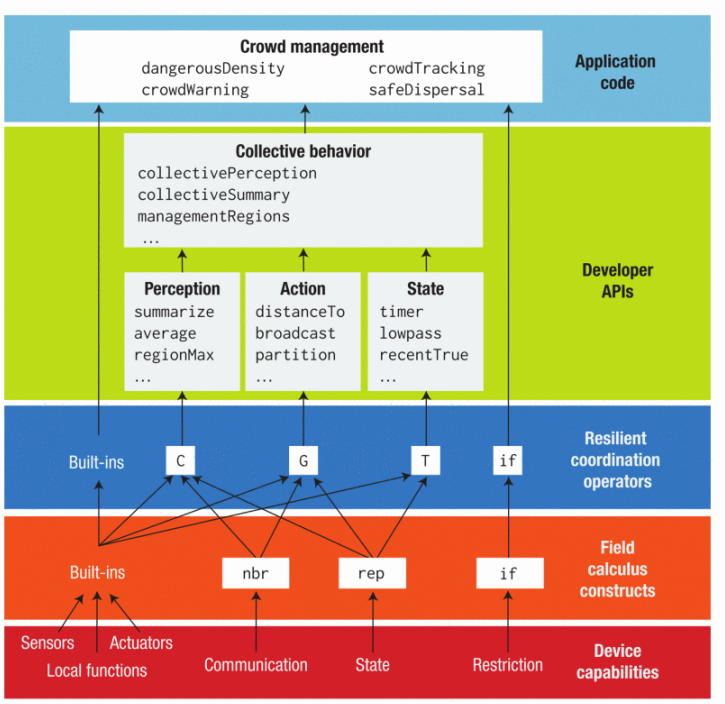

Figure 2. Layered approach to aggregate computing. Image from (Beal et al., 2015).

the overall set of computational devices spread in the pervasive computing environment as a single "machine", a sort of diffused computational fabric (Viroli et al., 2016). Hence, there is not a device, but a region to be programmed.

Its layered approach (Figure 2) which addresses IoT (Internet of Things) systems with five abstraction layers, which hide the complexity of distributed coordination in IoT networks. Following a bottom-up view, we try to summarise its main features:

(i) Device capabilities. Each device has its own capabilities. Since the communication and the interaction among heterogeneous devices are extremely different, they might require ad-hoc considerations that should not impact on the design of the aggregate.

(ii) Field calculus. *Field calculus* is a functional language which provides basic constructs to work with computational fields. Within this language, a distributed system can be considered as a functional composition of computational fields (Viroli et al., 2013). A field maps each networked device to some local value: for instance, a collection of temperature sensors produces a field of ambient temperatures. The *field calculus* is the very core of the aggregate programming approach: it is this layer where aggregate programming interfaces with the open world of device infrastructures and non-aggregate software services.

(iii) Resilient coordination. Field calculus is too low-level for programming resilient systems (Beal & Viroli, 2015). Therefore, at this layer, field calculus constructs are combined into resilient coordination operators, which we can see as a sort of higher-level "building block" algorithms. Such operators contribute to the definition of a simple and generalised "algebra" of programs with desirable resilience properties. Specifically, four self-stabilising algorithms are introduced: G (spreading), C (aggregation), T (temporary state), and S (symmetry breaking through mutual inhibition). They are self-stabilising, because they are able to adjust to changes in network structure or input values, they are scalable to large networks and preserve these resilience properties when composed with one another. Consequently, any application built only using these operators for coordination and state is guaranteed to have good scaling properties and to be self-stabilising (Beal et al., 2015; Viroli, Beal, Damiani, & Pianini, 2015).

(iv) User-friendly APIs. On the top of those building blocks, this layer defines APIs (application programming interfaces) to compose them into functions which safely encapsulate general complex mechanisms. With the definition of those reusable building-block APIs, we can consider completed the core of the aggregate computing ecosystem: they can be organised in libraries, which in turn can be leveraged to define higher-level API upon which the application code is written.

(v) Application code. From an application perspective, it is sufficient to import APIs and built-in functions to develop IoT services using distributed coordination.

Discussion

Based on the notion of *computational field* and *field calculus*, aggregate computing proposes a new programming perspective which has been described as follows: "aggregate programming is about defining global behaviours which can be "pulverised" into a computational "dust", formed by atomic actions", called computation rounds, "to be repetitively executed through space and time by the many devices available" (Viroli et al., 2016). Therefore, the importance of individual computing devices is reduced in a significant way.

At this point, it raises the question of if and how this approach can fit cloud-/fog-oriented platforms. This question is the subject of (Viroli et al., 2016). Here, the authors point out that aggregate computing model can smoothly adapt to such platforms, even though they originated in the context of peer-to-peer, fully distributed and self-organising systems. Indeed, if we consider real highly distributed and heterogeneous IoT scenarios, it turns out that different issues can occur. Just to mention some of them, certain devices may not be able to communicate in a peer-to-peer manner, or certain areas may become overcrowded, thus exceeding the capabilities of the local base station and resulting in the loss of mobile web access.

Hence, pragmatically, aggregate programming platforms must take into account those issues and provide support to cloud-/fog-based integration. This approach is not a contradiction, but rather an opportunity to extend aggregate platforms using cloud-oriented approaches. One of the proposed solution which follows this idea could be "to store the whole computational field as a big data" in the cloud. Aggregate computation would then be structured as a "myriad of stateless computing services concurrently working on a big shared database".

More interestingly, an even more dynamic integration is proposed, really near to the spirit which inspired the definition of fog computing model, because it implicitly refers to the "continuum from Cloud to Thing" which characterises it. Indeed, according to the authors, "the global aggregate computation might even be executed partially "on ground" and may "flow" up and down depending upon context and contingencies — energetic issues, presence of congestions, unexpected storage requirements, changes in wireless availability, and so on".

In conclusion, on the one hand the integration between the aggregate computing approach and the cloud-oriented platforms would allow to benefit from the abstraction layer provided by the former, based on resilient coordination operators and which allows an enormous simplification of processes such as the design, creation, and maintenance of complex and large-scale IoT software systems. On the other hand, the cloud/fog support would guarantee the scalability and the elasticity of the built solution, addressing those issues that aggregate computing, on its own, could not solve.

References

Beal, J., Pianini, D., & Viroli, M. (2015, Sept). Aggregate programming for the internet of things. *Computer*, *48*(9), 22-30.

Beal, J., & Viroli, M. (2015). Space–time programming. *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, *373*(2046).

Bonomi, F., Milito, R., Zhu, J., & Addepalli, S. (2012). Fog computing and its role in the internet of things. In *Proceedings of the first edition of the mcc workshop on mobile cloud computing* (pp. 13–16).

Iorga, M., Goren, N., Feldman, L., Barton, R., Martin, M., & Mahmoudi, C. (2018). *Sp 500-325. Fog Computing Conceptual Model.* Gaithersburg, MD, United States: National Institute of Standards & Technology.

Mell, P. M., & Grance, T. (2011). Sp 800-145. the nist definition of cloud computing.

Openfog consortium. (2017). Retrieved from `http://www.openfogconsortium.org`

Toll, W. (2014). *Top 49 tools for the internet of things.* Retrieved from `https://blog.profitbricks.com/top-49-tools-internet-of-things/` ([Online] accessed 25-July-2018)

Viroli, M., Beal, J., Damiani, F., & Pianini, D. (2015, Sept). Efficient engineering of complex self-organising systems by self-stabilising fields. In *2015 ieee 9th international conference on self-adaptive and self-organizing systems* (p. 81-90).

Viroli, M., Casadei, R., & Pianini, D. (2016). On execution platforms for large-scale aggregate computing. In *Proceedings of the 2016 acm international joint conference on pervasive and ubiquitous computing: Adjunct* (pp. 1321–1326).

Viroli, M., Damiani, F., & Beal, J. (2013). A calculus of computational fields. In C. Canal & M. Villari (Eds.), *Advances in service-oriented and cloud computing* (pp. 114–128). Berlin, Heidelberg: Springer Berlin Heidelberg.