Fog Computing: A Platform for Internet of Things and Analytics

Flavio Bonomi, Rodolfo Milito, Preethi Natarajan and Jiang Zhu

Abstract Internet of Things (IoT) brings more than an explosive proliferation of endpoints. It is disruptive in several ways. In this chapter we examine those disruptions, and propose a hierarchical distributed architecture that extends from the edge of the network to the core nicknamed Fog Computing. In particular, we pay attention to a new dimension that IoT adds to Big Data and Analytics: a massively distributed number of sources at the edge.

1 Introduction

The "pay-as-you-go" Cloud Computing model is an efficient alternative to owning and managing private data centers (DCs) for customers facing Web applications and batch processing. Several factors contribute to the economy of scale of mega DCs: higher predictability of massive aggregation, which allows higher utilization without degrading performance; convenient location that takes advantage of inexpensive power; and lower OPEX achieved through the deployment of homogeneous compute, storage, and networking components.

Cloud computing frees the enterprise and the end user from the specification of many details. This bliss becomes a problem for latency-sensitive applications, which require nodes in the vicinity to meet their delay requirements. An emerging wave of Internet deployments, most notably the Internet of Things (IoTs), requires mobility support and geo-distribution in addition to location awareness and low latency. We argue that a new platform is needed to meet these requirements; a platform we call Fog Computing [1]. We also claim that rather than cannibalizing Cloud Computing,

F. Bonomi · R. Milito · P. Natarajan (✉) · J. Zhu (✉)
Enterprise Networking Labs, Cisco Systems Inc., San Jose, USA
e-mail: prenatar@cisco.com

J. Zhu
e-mail: jiangzhu@cisco.com

N. Bessis and C. Dobre (eds.), *Big Data and Internet of Things: A Roadmap for Smart Environments*, Studies in Computational Intelligence 546,
DOI: 10.1007/978-3-319-05029-4_7,

Fog Computing enables a new breed of applications and services, and that there is a fruitful interplay between the *Cloud* and the *Fog*, particularly when it comes to data management and analytics.

Fog Computing extends the Cloud Computing paradigm to the edge of the network. While Fog and Cloud use the same resources (networking, compute, and storage), and share many of the same mechanisms and attributes (virtualization, multi-tenancy) the extension is a non-trivial one in that there exist some fundamental differences that stem from the Fog raison d'etre. The Fog vision was conceived to address applications and services that do not fit well the paradigm of the Cloud. They include:

- Applications that require very low and predictable latency—the Cloud frees the user from many implementation details, including the precise knowledge of where the computation or storage takes place. This freedom from choice, welcome in many circumstances becomes a liability when latency is at premium (gaming, video conferencing).
- Geo-distributed applications (pipeline monitoring, sensor networks to monitor the environment).
- Fast mobile applications (smart connected vehicle, connected rail).
- Large-scale distributed control systems (smart grid, connected rail, smart traffic light systems).

The emergence of the Internet of Things (IoT) brings a number of use cases of interest that fall in the above categories. It also brings Big Data with a twist: rather than high volume, in IoT big is the number of data sources distributed geographically. For these reasons we place IoT at the center stage of our discussion. We will examine some IoT use cases of interest, and will take a close look at analytics at the edge.

At this point we want to stress that the Fog complements the Cloud, does not substitute it. For the large class of Cloud intended applications the economies of scale (OPEX in particular) cannot be beaten. These efficiencies result from locating mega data centers where energy is inexpensive, and running them with a small team of qualified individuals. For the users (an individual, a small enterprise or even a big business), the flexibility of the pay-as-you-go model is very attractive. Combining these considerations, we state firmly that the Cloud paradigm is here to stay. On the other hand we argue that the model is not universally applicable, and that emergent IoT applications demand a platform with novel characteristics.

The rest of the chapter is organized as follows. Section 2 addresses the question: "what are the disruptive aspects of IoT?" to motivate the need for a hierarchical platform that extends from the edge to the core of the network. Section 3 brings up two use cases, chosen primarily to illustrate some key architectural and system-wide points. IoT brings a new dimension to Big Data and Analytics: massively distributed data sources at the edge. This is the theme of Sect. 4. Section 5 crystallizes a number of observations made in the previous section in some high-level architectural requirements. Section 6 presents an overview of the Fog software architecture, highlighting the essential technology components, followed by conclusions in Sect. 7.

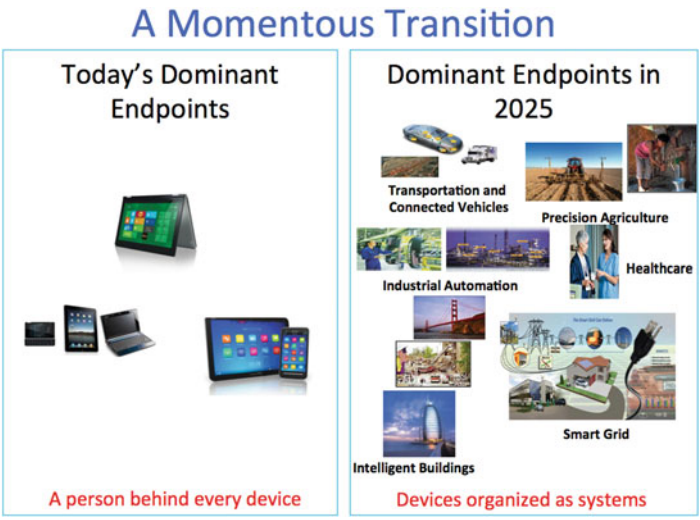

Fig. 1 A momentous transition in IoT

2 What is Disruptive with IoT?

Current estimates place the number of endpoints (mostly smart phones, tablets, and laptops) around three or four billions. This number is expected to grow to a trillion (two to three orders of magnitude!) in a few years. This phenomenal explosion points to a major scalability problem assuming that nothing changes in the nature of the endpoints, the architecture of the platforms at the edge, and the communication model. Some scalability will have to be addressed, but the first critical issue to examine is the disruptive changes that IoT brings to the fore, changes that force to rethink the existing paradigm.

The first change refers the very nature of the endpoints. Today there is a person behind the vast majority of endpoints. The dominant landscape in IoT is different. As depicted in Fig. 1 most endpoints are organized into systems, with the system being a "macro endpoint". For instance, the many sensors and actuators in a Smart Connected Vehicle (SCV) communicate among them, but it is the vehicle as a unit that communicates with other vehicles, with the roadside units (RSUs) and with the Internet at large. The same is valid in the Smart Cities, Oil and Gas, Industrial Automation, etc. The first conclusion from this observation is that IoT forces us to adopt "system view", rather than an "individual view" of the endpoints. A number of consequences derive from this conclusion:

- A vibrant community of individual developers, working on the Android or the Apple platform, contributes today with endpoint applications. This community will not disappear, but as the IoT takes hold, domain expert companies will play an increasingly important role in the development of systems and applications

- A number of important IoT use cases, including Smart Cities, pipeline monitoring, Smart Grid, Connected Rail (CR) are naturally geographically distributed
- Support for fast mobility is essential in some very relevant use cases (SCV, CR)
- Low and predictable latency are essential attributes in industrial automation and other IoT use cases.

This list is well aligned with the attributes of the Fog we outlined in the Introduction.

3 Illustrative Use Cases to Drive Fog Computing Requirements

In this section we describe and analyze a couple of use cases. While the use cases are relevant in themselves, the choices have been guided neither by ease of deployment nor market considerations. Rather, they serve to illustrate some key characteristics of the architecture. We should emphasize that point solutions for each use case, or even for each major vertical do not cut it. Our intent is to abstract the major requirements to propose an architecture that addresses all, or at least the vast majority of the IoT requirements.

3.1 Use Case 1: A Smart Traffic Light System (STLS)

This use case considers a system of a system of traffic lights deployed in an urban scenario. The STLS is a small piece of the full-fledged system envisioned by Smart Connected Vehicle (SCV) and Advanced Transportation Systems, but rich enough to drive home some key requirements for the platform at the edge.

3.1.1 System Outline

- STLS calls for the deployment of a STL at each intersection. The STL is equipped with sensors that (a) measure the distance and speed of approaching vehicles from every direction; (b) detect the presence of pedestrians and cyclists crossing the street. The STL also issues “slow down” warnings to vehicles at risk to crossing in red, and even modifies its own cycle to prevent collisions.
- The STLS has three major goals: (a) accidents prevention; (b) maintenance of a steady flow of traffic (green waves along main roads); (c) collection of relevant data to evaluate and improve the system. Note that the global nature of (b) and (c), in contrast with the localized objective (a). Also note the wide difference in time scales: (a) requires real time (RT) reaction, (b) near-real time, and (c) relates to the collection and analysis of global data over long periods.

- To be specific, consider some numbers. Let us say that the green wave is set at 64 km/h (40 miles/h). A vehicle moving at 64 km/h travels 1.7 m in 100 ms. The policy requires sending an urgent alarm to approaching vehicles when collision with crossing pedestrians is anticipated. To be effective the local control loop subsystem must react within a few milliseconds—thus illustrating the role of the Fog in supporting low latency applications. Accident prevention trumps any other consideration. Hence, to prevent a collision the local subsystem may also modify its own cycle.[1] In doing so it introduces a perturbation in the green wave that affects the whole system. To dampen the effect of the perturbation a re-synchronization signal must be sent along all the traffic lights. This process takes place in a time scale of hundreds of milliseconds to a few seconds.

3.1.2 Key Requirements Driven by the STLS

This section discusses the requirements of a smart traffic light system; the requirements are also highlighted in (Table 1)

1. *Local subsystem latency*: In STLS a subsystem includes the traffic light, sensors and actuators in a local region such that the reaction time is on the order of <10 ms.
2. *Middleware orchestration platform*: The middleware handles a number of critical software components across the whole system, which is deployed across a wide geographical area. The components include:
   - The decision maker (DM), which creates the control policies and pushes them to the individual traffic lights. The DM can be implemented in a centralized, distributed or in a hierarchical way. In the latter, the most likely implementation, nodes with DM functionality of regional scope must coordinate their policies across the whole system. Whatever the implementation, the system should behave as if orchestrated by a single, all knowledgeable DM.
   - The federated message bus, which passes data from the traffic lights to the DM nodes, pushes policies from the DM nodes to the traffic lights, and exchanges information between the traffic lights.
3. *Networking infrastructure*: The Fog nodes belong to a family of modular compute and storage devices. However different the form factors, and the encasings (a ruggedized version is required for the traffic light), they offer common interfaces, and programming environment.
4. *Interplay with the Cloud*: In addition to the actionable real-time (RT) information generated by the sensors, and the near-RT data passed to the DM and exchanged among the set of traffic lights, there are volumes of valuable data collected by the system. This data must be ingested in a data center (DC)/Cloud for deep analytics that extends over time (days, months, even years) and over the covered territory. In a cursory review of the potential value of this data we list: (a) evaluation of

[1] The design of a good control policy for all scenarios is certainly a non-trivial task, beyond the scope of this discussion.

Table 1 Attributes of a smart traffic light system

Mobility	
Geo-distribution	Wide (across region) and dense (intersections and ramp accesses)
Low/predictable latency	Tight within the scope of the intersection
Fog-cloud interplay	Data at different time scales (sensors/vehicles at intersection, traffic info at diverse collection points)
Multi-agencies orchestration	Agencies that run the system must coordinate control law policies in real time
Consistency	Getting the traffic landscape demands a degree of consistency between collection points

the impact on traffic (and its consequences for the economy and the environment) of different policies; (b) monitoring of city pollutants; (c) trends and patterns in traffic. We emphasize here the interplay between the Fog and the Cloud, which must operate in mutual support.

5. *Consistency of a highly distributed system*: Visualize the STLS as a highly distributed collector of traffic data over an extended geographically data. Ensuring an acceptable degree of consistency between the different aggregator points is crucial for the implementation of efficient traffic policies.
6. *Multi-tenancy*: The Fog vision anticipates an integrated hardware infrastructure and software platform with the purpose of streamlining and making more efficient the deployment of new services and applications. To run efficiently, the Fog must support multi-tenancy. It must also provide strict service guarantees for mission critical systems such as the STLS, in contrast with softer guarantees for say, infotainment, even when run for the same provider.
7. *Multiplicity of providers*: The system of traffic lights may extend beyond the borders of a single controlling authority. The orchestration of consistent policies involving multiple agencies is a challenge unique to Fog Computing.

3.2 Use Case 2: Wind Farm

A wind farm offers a rich use case of Fog Computing, one that brings up characteristics and requirements shared by a number of Internet of Everything (IoE) deployments:

1. Interplay between real time analytics and batch analytics.
2. Tight interaction between sensors and actuators, in closed control loops.
3. Wide geographical deployment of a large system consistent of a number of autonomous yet coordinated modules—which gives rise to the need of an orchestrator.

3.2.1 System Outline

Modern utility-scale wind turbines are very large flexible structures equipped with several closed control loops aimed at improving wind power capture (measured as the ratio of actual to full capacity output for a given period) and power quality (affected by harmonic distortion) as well as reducing structural loading (hence extending lifetime and decreasing maintenance costs). A large wind farm may consist of hundreds of individual wind turbines, and cover an area of hundreds of square miles.

Power curves [2] depict power [MW] (both wind power and turbine power) versus wind speed [m/s]. There are four typical operating regions:

1. In region 1 the wind speed is so low (say, below 6 m/sec) that the available wind power is in the order of losses in the system, so it is not economically sound to run the turbine.
2. Region 2 (e.g. winds between 6 and 12 m/s) is the normal operating condition, in which blades are positioned to effect the maximum conversion of wind power into electrical power
3. The turbine power curve plateaus when the wind exceeds a certain threshold (say 12 m/s). In this, Region 3, power is limited to avoid exceeding safe electrical and mechanical load limits
4. Very high wind speeds (say, above 25 m/s) characterize Region 4. Here, the turbine is powered down and stopped to avoid excessive operating loads

In short, several controllers are used to tune the turbine (yaw and pitch) to the prevailing wind conditions in order to increase efficiency, and to stop it to minimize wear and prevent damage. These controllers operate in a semi-autonomous way at each turbine. Note, however, that a global coordination at the farm level is required for maximum efficiency. In fact, optimization at the individual level of each turbine may "wind starve" the turbines at the rear.

Central to the process of assessing a potential wind farm deployment is the study of atmospheric stability and wind patterns on a yearly and monthly basis, along with terrain characteristics. An operational wind farm requires fairly accurate wind forecasting at different time scales. We refer the reader to Botterud and Wang [3] for interesting details on the market operation. It suffices for our purposes to consider the following:

- Daily forecast used to submit bids to the independent system operator (ISO)
- Hourly forecast to adjust the commitment to changes in the operating conditions (forced outages, deviations from the forecast loads, etc.)
- Finer granularity (5 min interval) to dynamically optimize the wind farm operation.

3.2.2 Key Requirements Driven by the Wind Farm Use Case

However coarse, the above description touches on a few interesting characteristics of a wind farm operation. These characteristics are shared by a number of cases

Table 2 Attributes from the wind farm use case

Mobility	
Geo-distribution	Wide and dense but confined to the farm
Low/predictable latency	Critical at the turbine level
Fog-cloud interplay	Diverse time scales and sources (weather forecasting and local wind, market conditions and trends)
Multi-agencies orchestration	
Consistency	Applies to the actionable wind data within the farm

emerging under the IoE umbrella. We have a large-scale, geographically distributed system that includes thousands of sensors and actuators. The system consists of a large number of semi-autonomous modules or sub-systems (turbines). Each subsystem is a fairly complex system on its own, with a number of control loops. Established organizing principles of large-scale systems (safety, among others) recommend that each subsystem should be able to operate semi-autonomously, yet in a coordinated manner. This system requires a platform that includes (Table 2):

- *Networking Infrastructure*: An efficient communication network between the subsystems, and between the system and the Internet at large (cloud).
- *Global controller*: A controller with global scope, possibly implemented in a distributed way. The controller builds an overall picture from the information fed from the subsystems, determines a policy, and pushes the policy for each subsystem. The policy is global, but individualized for each subsystem depending on its individual state (location, wind incidence, conditions of the turbine). The continuous supervisory role of the global controller (gathering data, building the global state, determining the policy) requires low latency achievable locally in the edge type deployment we call the Fog.
- *Middleware orchestration platform*: A middleware that mediates between the subsystems and the cloud.
- *Data Analytics*: This system generates huge amounts of data. Much of this information is actionable in real time. It feeds the control loops of the subsystems, and it also used to renegotiate the bidding terms with the ISO when necessary. Beyond this real (network) time applications, the data can be used to run analytics over longer periods (months, years), over wider scenarios (including other wind farms, and other energy data). The cloud is the natural place to run this rich batch analytics.

3.3 Key Attributes of the Fog Platform

The use cases discussed above bring up a number of attributes that differentiate the Fog Computing Platform from the Cloud. Those attributes do not apply uniformly to every use case. Mobility, for instance, a critical attribute in Smart Connected Vehicle

and Connected Rail, plays no role in the STLS and Wind Farm cases. Multi-agent orchestration, critical in STLS, does not apply in the Wind Farm case. This is another facet of the heterogeneity of the Fog. The idea is to deploy a common platform that supports a wide range of verticals, rather than point solutions for each vertical. However, different verticals may exercise different attributes of the platform.

Along these lines is worth analyzing the concept of multi-tenancy. Both Cloud and Fog support multi-tenancy, i.e., the support of a multiplicity of client-organizations without mutual interference. There are subtle differences, though, in the nature of those client-organizations in the Cloud and Fog environments.

Typical users of the Cloud include individuals, and enterprises. Some of the latter off-load completely their IT to the Cloud; many overflow during periods of peak demand. The pay-as-you-go, task-oriented model dominates. The Fog platform also supports task-oriented compute and storage requests, but the dominant mode is different. In the two use cases described the operations run 24×7 every day of the year, with possible periods of fluctuations in the demand. Similar models apply to many other verticals and use cases. In fact the IoT landscape brings a new set of actors and novel interactions among them: the owner of the infrastructure, the agency or agencies that run the service, the users of the service. Business models that determine the different levels of responsibility and the actual exchange of service and money among the participants will structure these interactions. Development of these business models goes beyond the scope of this chapter. It is worth, though, to revisit the STLS use case to get insight on the issues involved. One or more agencies run the system based on a platform owned by a service provider. Other services (a pollution monitoring system, for example), run by different agencies coexist on the same platform. Note that in this example the concept of the end user (ex: pedestrian crossing the street, approaching vehicles) is diluted, as is the way to pay for the service.

4 Geo-distribution: A New Dimension of Big Data

Big Data today is currently characterized along three dimensions: Volume, Velocity, and Variety. As argued in Sect. 2, Many IoT use cases, including STLS, Smart Cities, Smart Grids, Connected Rail (CR), and pipeline monitoring are naturally distributed. This observation suggests adding a fourth dimension to the characterization of Big Data, namely, *geo-distribution*. Consider, for instance, the monitoring of a pipeline, or the measurement of pollution level (air quality, pollen, and noise) throughout a city. The big N in these scenarios is neither the number of terabytes nor rate of data generated by any individual sensor, but rather the number of sensors (and actuators) that are *naturally distributed*, and that has to be managed as a coherent whole. The call for "moving the processing to the data" is getting louder. There is a need for *distributed intelligent platform at the Edge (Fog Computing)* that manages distributed compute, networking, and storage resources.

In short, IoT/IoE transforms the landscape more dramatically than the image of zillions of sensors suggests. Together with a number of challenges, IoT/IoE brings

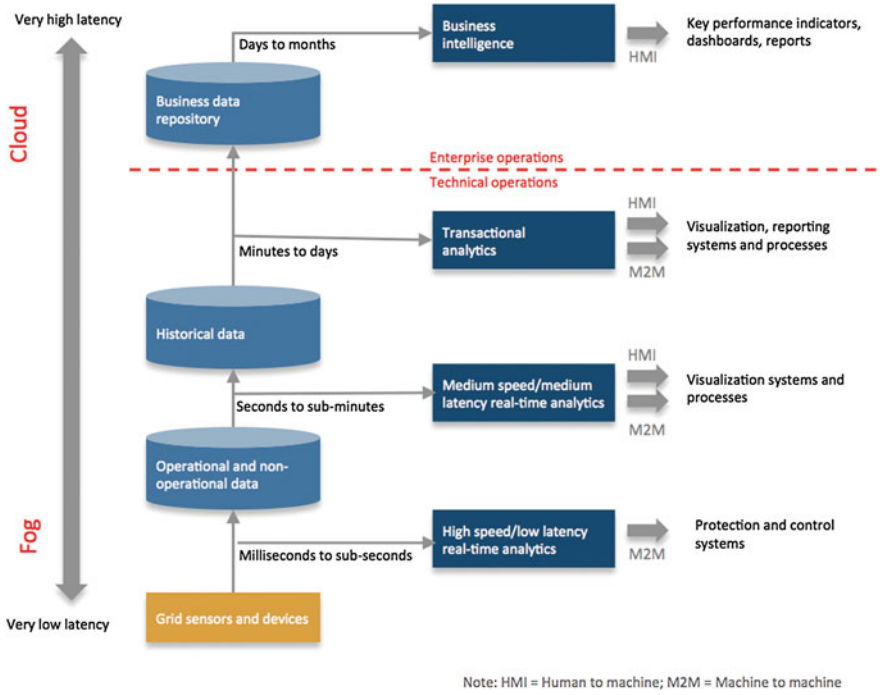

Fig. 2 Many uses of the same data [5]

new opportunities: this is a game that requires (a) a solid presence from the edge to the core that includes networking, compute and storage resources; and (b) the creation of an ecology of domain expert partners to attack these nascent market successfully.

In many use cases there is a rich interplay between the edge and the core of the network, because the data generated has different requirements and uses at different time scales. This topic is further discussed next.

4.1 Interplay Between the Edge (Fog) and the Core (Cloud)

Many use cases of interest actually demand an active cooperation between the Edge and the Core. The scope of the data processed, narrow in space and time at the edge, and wide at the core, is typical in many verticals of interest (Smart Connected Vehicle, Oil and Gas, Connected Rail, Smart Communities, etc.). It suggests a hierarchical organization of the networking, compute, and storage resources.

To illustrate the point consider an example taken from Smart Grid, as depicted in Fig. 2. The figure shows a whole range of time-scales, from milliseconds to months. The machine-to-machine (M2M) interactions at the millisecond-sub second

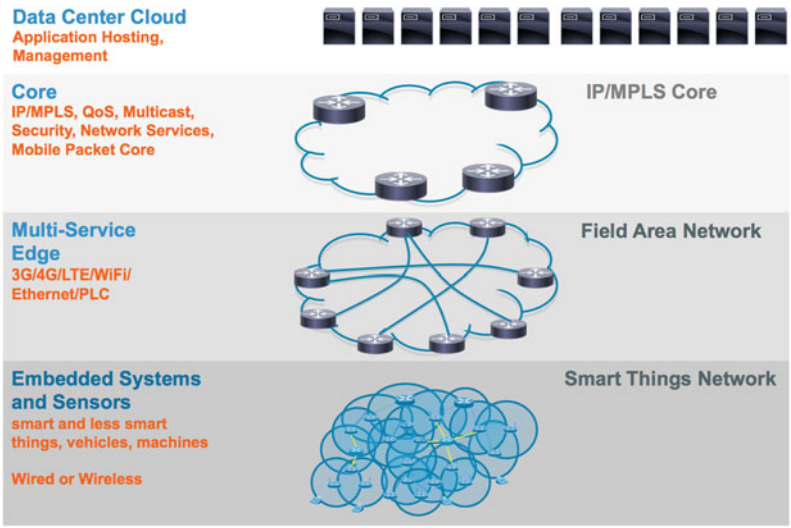

Fig. 3 Fog distributed infrastructure for IoT/IoE

level cover the real-time actionable analytics (breakers, etc.). The second level (second-to-sub minute) includes both M2M processes and human-to-machine (H2M) interactions (visualization, starting/stopping home appliances). Not reflected in the figure is the fact as we climb up the time scale the scope gets wider in coverage. For instance, the business intelligence level that operates on data collected over months may cover a wide geographical region, possibly a whole country and beyond. In contrast, the sub-second level refers to more localized loops aimed essentially to keep the grid stable.

5 High-Level Architectural and System-Wide View

Cloud Computing consists of mostly homogenous physical resources that are deployed and managed in a centralized fashion. Fog complements and extends the Cloud to the edge and endpoints; Fog's distributed infrastructure comprising of heterogeneous resources needs to be managed in a distributed fashion. Figure 3 shows the various players in the distributed Fog infrastructure, ranging from data centers, core of the network, edge of the network, and end points. The Fog architecture enables distributed deployment of applications requiring computing, storage and networking resources spanning across these different players.

Similar to Cloud, Fog architecture supports co-existence of applications belonging to different tenants. Each tenant perceives its resources as dedicated, and defines its own topology. Figure 4 shows two active tenants, A and B, with their respective applications. The distributed applications for A has one Cloud component, and two Fog components. The application for B has one cloud component, one component

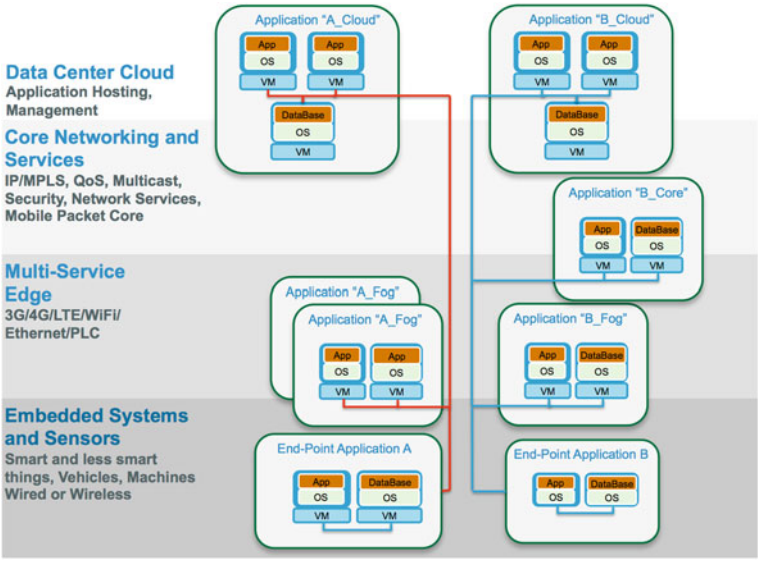

Fig. 4 Distributed IoT/IoE applications on the fog infrastructure

in the Core, and a Fog component. A virtual network topology is allocated for each tenant.

In order to fully manifest this distributed architecture, Fog relies on technology components for scalable virtualization of the key resource classes:

- Computing, requiring the selection of hypervisors in order to virtualize both the computing and I/O resources.
- Storage, requiring a Virtual File System and a Virtual Block and/or Object Store.
- Networking, requiring the appropriate Network Virtualization Infrastructure (e.g., Software Defined Networking technology).

Similar to Cloud, Fog leverages a policy-based orchestration and provisioning mechanism on top of the resource virtualization layer for scalable and automatic resource management. Finally, Fog architecture exposes APIs for application development and deployment.

6 Software Architecture

The use cases and the requirements discussed in previous sections help nail down the following key objectives of the Fog software architecture:

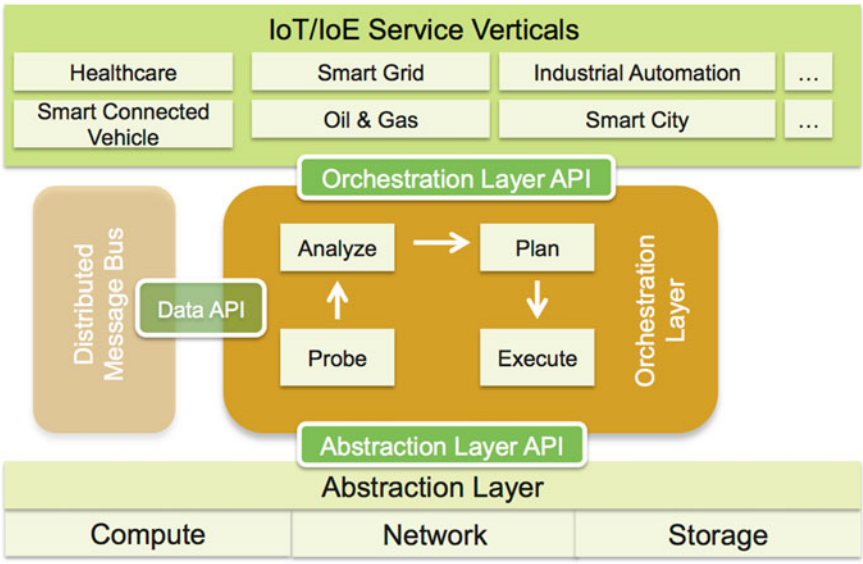

Fig. 5 Components in fog architecture

- Fog nodes are heterogeneous in nature and deployed in variety of environments including core, edge, access networks and endpoints. The Fog architecture should facilitate seamless resource management across the diverse set of platforms.
- The Fog platform hosts diverse set of applications belonging to various verticals—smart connected vehicles to smart cities, oil and gas, smart grid etc. Fog architecture should expose generic APIs that can be used by the diverse set of applications to leverage Fog platform.
- The Fog platform should provide necessary means for distributed policy-based orchestration, resulting in scalable management of individual subsystems and the overall service.

The rest of this section discusses the components of Fog architecture (Fig. 5), explaining in detail how the above objectives are met.

6.1 Heterogeneous Physical Resources

As discussed earlier, Fog nodes are heterogeneous in nature. They range from high-end servers, edge routers, access points, set-top boxes, and even end devices such as vehicles, sensors, mobile phones etc.[2] The different hardware platforms have varying

[2] Consistently with the "system view" of IoT we regard a Connected Vehicle, a manufacturing cell, etc. as end devices.

levels of RAM, secondary storage, and real estate to support new functionalities. The platforms run various kinds of OSes, software applications resulting in a wide variety of hardware and software capabilities.

The Fog network infrastructure is also heterogeneous in nature, ranging from high-speed links connecting enterprise data centers and the core to multiple wireless access technologies (ex: 3G/4G, LTE, WiFi etc.) towards the edge.

6.2 Fog Abstraction Layer

The Fog abstraction layer (Fig. 5) hides the platform heterogeneity and exposes a uniform and programmable interface for seamless resource management and control. The layer provides generic APIs for monitoring, provisioning and controlling physical resources such as CPU, memory, network and energy. The layer also exposes generic APIs to monitor and manage various hypervisors, OSes, service containers, and service instances on a physical machine (discussed more later).

The layer includes necessary techniques that support virtualization, specifically the ability to run multiple OSes or service containers on a physical machine to improve resource utilization. Virtualization enables the abstraction layer to support multi-tenancy. The layer exposes generic APIs to specify security, privacy and isolation policies for OSes or containers belonging to different tenants on the same physical machine. Specifically, the following multi-tenancy features are supported:

- Data and resource isolation guarantees for the different tenants on the same physical infrastructure
- The capabilities to inflict no collateral damage to the different parties at the minimum
- Expose a single, consistent model across physical machine to provide these isolation services
- The abstraction layer exposes both the physical and the logical (per-tenant) network to administrators, and the resource usage per-tenant.

6.3 Fog Service Orchestration Layer

The service orchestration layer provides dynamic, policy-based life-cycle management of Fog services. The orchestration functionality is as distributed as the underlying Fog infrastructure and services. Managing services on a large volume of Fog nodes with a wide range of capabilities is achieved with the following technology and components:

- A software agent, *Foglet*, with reasonably small footprint yet capable of bearing the orchestration functionality and performance requirements that could be embedded in various edge devices.

- A distributed, persistent storage to store policies and resource meta-data (capability, performance, etc) that support high transaction rate update and retrieval.
- A scalable messaging bus to carry control messages for service orchestration and resource management.
- A distributed policy engine with a single global view and local enforcement.

6.3.1 Foglet Software Agent

The distributed Fog orchestration framework consists of several Foglet software agents, one running on every node in the Fog platform. The Foglet agent uses abstraction layer APIs to monitor the health and state associated with the physical machine and services deployed on the machine. This information is both locally analyzed and also pushed to the distributed storage for global processing.

Foglet is also responsible for performing life-cycle management activities such as standing up/down guest OSes, service containers, and provisioning and tearing down service instances etc. Thus, Foglet's interactions on a Fog node span over a range of entities starting from the physical machine, hypervisor, guest OSes, service containers, and service instances. Each of these entities implements the necessary functions for programmatic management and control; Foglet invokes these functions via the abstraction layer APIs.

6.3.2 Distributed Database

A distributed database, while complex to implement is ideal for increasing Fog's scalability and fault-tolerance. The distributed database provides faster (than centralized) storage and retrieval of data. The database is used to store both application data and necessary meta-data to aid in Fog service orchestration. Sample meta-data include (discussed more in the next subsection):

- Fog node's hardware and software capabilities to enable service instantiation on a platform with matching capabilities.
- Health and other state information of Fog nodes and running service instances for load balancing, and generating performance reports.
- Business policies that should be enforced throughout a service's life cycle such as those related to security, configuration etc.

6.3.3 Policy-Based Service Orchestration

The orchestration framework provides policy-based service routing, i.e., routes an incoming service request to the appropriate service instance that confirms to the relevant business policies. The framework achieves this with the help of the policy

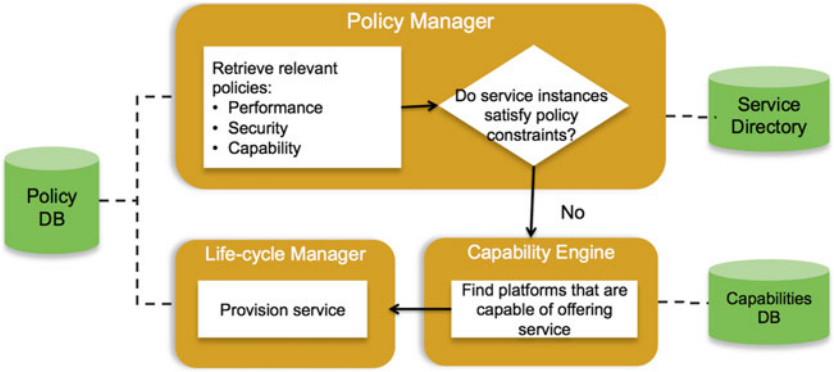

Fig. 6 Policy-based orchestration framework

manager (Fig. 6). This section discusses the workflow associated with policy-based orchestration.

Administrators interact with the orchestration framework via an intuitive dashboard-style user interface (UI). Admins enter business policies, manage, and monitor the Fog platform through this UI. The UI offers policy templates that admins can refine based on needs. The policy framework is extensible and supports a wide variety of policies. Few example policies include:

- Policies to specify thresholds for load balancing such as maximum number of users, connections, CPU load etc.
- Policies to specify QoS requirements (network, storage, compute) with a service such as minimum delay, maximum rate etc.
- Policies to configure device, service instance in a specific setting.
- Policies to associate power management capabilities with a tenant/Fog platform.
- Policies to specify security, isolation and privacy during multi-tenancy.
- Policies that specify how and what services must be chained before delivery, ex: firewall before video service.

Business policies specified via the UI are pushed to a distributed policy database (Fig. 6). The policy manager is triggered by an incoming service request. The policy manager gathers relevant policies i.e., those pertaining to the service, subscriber, tenant etc. from the policy repository. The policy manager also retrieves meta-data about currently active service instances from the services directory. With these two sets of data, the policy manager tries to find an active service instance that satisfies the policy constraints, and forwards the service request to that instance. If no such instance is available, then a new instance must be created. For that purpose, the policy manager invokes the capability engine whose job is to identify a ranked list of Fog nodes whose capabilities match the policy constraints for instantiating the new service. The capability engine hands over this ranked list to the life cycle manager that provisions the service on a Fog device. The life cycle manager may reach out to

the policy repository to identify device, service, and network configuration policies while provisioning the new instance.

The orchestration functionality is distributed across the Fog deployment such that the logic in Fig. 6 is embedded in every Foglet. The distributed control provides better resiliency, scalability, and faster orchestration for geographically distributed deployments.

6.4 North-Bound APIs for Applications

The Fog software framework exposes northbound APIs that applications use to effectively leverage the Fog platform. These APIs are broadly classified into data and control APIs. Data APIs allow an application to leverage the Fog distributed data store. Control APIs allow an application to specify how the application should be deployed on the Fog platform.

Few example APIs:

- Put_data(): To store/update application-specific data and meta-data on the Fog distributed data store.
- Get_data(): To retrieve application-specific data meta-data from the Fog distributed data store.
- Request_service(): To request for a service instance that matches some criteria.
- Setup_service(): To setup a new service instance that matches some criteria.
- Install_policy(): To install specific set of policies for a provider, subscriber in the orchestration framework.
- Update_policy (): To configure/re-configure a policy with a specific set of parameters (ex: thresholds for a load balancing policy).
- Get_stats(): To generate reports of Fog node health and other status.

7 Conclusions

This chapter looked at Fog Computing, a hierarchical and distributed platform for service delivery consisting of compute, storage, and network resources. We examined key aspects of Fog computing, and how Fog complements and extends Cloud computing. We looked at use cases that motivated the need for Fog, emphasizing Fog's relevance to several verticals within IoT and Big Data space. We also provided a high-level description of Fog's software architecture, highlighting the different technology components necessary to achieve the Fog vision.

Acknowledgments This work would not have been possible without the support of our colleagues, mentioned here in alphabetical order: Hao Hu, Mythili S. Prabhu, and Rui Vaz.

References

1. Bonomi, F., Milito, R., Zhu, J., Addepalli, S.: Fog computing and its role in the internet of things. In: Proceedings of the 1st edn. of the MCC workshop on Mobile cloud computing (2012)
2. Pao, L., Johnson, K.: A tutorial on the dynamics and control of wind turbines and wind farms. In: American Control Conference (2009)
3. Botterud, A., Wang, J.: Wind power forecasting and electricity market operations. In: International Conference of 32nd International Association for Energy Economics (IAEE), San Francisco, CA (2009)
4. Cristea, V., Dobre, C., Pop, F.: Context-aware environ internet of things. Internet of Things and Inter-cooperative Computational Technologies for Collective Intelligence Studies in Computational Intelligence, vol. 460, pp. 25–49 (2013)
5. Haak, D.: Achieving high performance in smart grid data management. White paper from Accenture (2010)