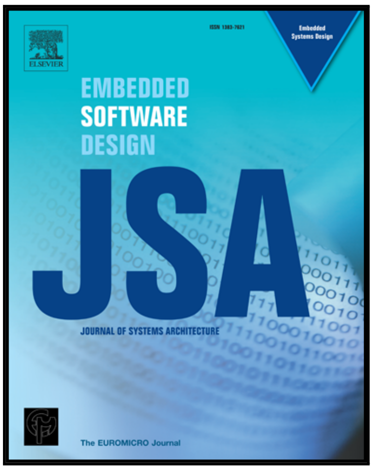


Introducing the new paradigm of Social Dispersed Computing: Applications, Technologies and Challenges

Marisol García-Valls, Abhishek Dubey, Vicent Botti

PII: S1383-7621(18)30103-6
DOI: 10.1016/j.sysarc.2018.05.007
Reference: SYSARC 1498

To appear in: *Journal of Systems Architecture*

Received date: 9 March 2018
Accepted date: 14 May 2018

Please cite this article as: Marisol García-Valls, Abhishek Dubey, Vicent Botti, Introducing the new paradigm of Social Dispersed Computing: Applications, Technologies and Challenges, *Journal of Systems Architecture* (2018), doi: 10.1016/j.sysarc.2018.05.007

Introducing the new paradigm of Social Dispersed Computing: Applications, Technologies and Challenges

Marisol García-Valls[*], Abhishek Dubey[†], Vicent Botti[‡]

June 1, 2018

Abstract

If last decade viewed computational services as a *utility* then surely this decade has transformed computation into a *commodity*. Computation is now progressively integrated into the physical networks in a seamless way that enables cyber-physical systems (CPS) and the Internet of Things (IoT) meet their latency requirements. Similar to the concept of “platform as a service” or “software as a service”, both *cloudlets* and *fog computing* have found their own use cases. Edge devices (that we call *end* or *user devices* for disambiguation) play the role of personal computers, dedicated to a user and to a set of correlated applications. In this new scenario, the boundaries between the network node, the sensor, and the actuator are blurring, driven primarily by the computation power of IoT nodes like single board computers and the smartphones. The *bigger data* generated in this type of networks needs clever, scalable, and possibly decentralized computing solutions that can scale independently as required. Any node can be seen as part of a graph, with the capacity to serve as a computing or network router node, or both. Complex applications can possibly be distributed over this graph or network of nodes to improve the overall performance like the amount of data processed over time. In this paper, we identify this new computing paradigm that we call *Social Dispersed Computing*, analyzing key themes in it that includes a new outlook on its relation to agent based applications. We architect this new paradigm by providing supportive application examples that

[*] Universidad Carlos III de Madrid, `mvalls@it.uc3m.es`

[†] Vanderbilt University, `abhishek.dubey@vanderbilt.edu`

[‡] Universitat Politècnica de València, `vbotti@dsic.upv.es`

include next generation electrical energy distribution networks, next generation mobility services for transportation, and applications for distributed analysis and identification of non-recurring traffic congestion in cities. The paper analyzes the existing computing paradigms (e.g., cloud, fog, edge, mobile edge, social, etc.), solving the ambiguity of their definitions; and analyzes and discusses the relevant foundational software technologies, the remaining challenges, and research opportunities.

1 Introduction

Social computing applications are smart applications, where the results received by the end users or the performance that they experience is affected by the other users using the same application. A classical example of this kind is traffic routing, implemented by many commercial mobility planning solutions like Waze and Google. The routes provided to the end users depend upon the interaction that other users in the systems have had with the application. An effective route planning solution will be proactive in the sense that it will analyze the demands being made by users and will use the dynamic demand model for effectively distributing vehicle and people across space, time, and modes of transportation, improving the efficiency of the mobility system and leading to a reduction of congestion. However, due to its nature, this computing application requires large scale real-time data ingestion, analysis, and optimization. We call such applications *social computing applications*.

With the burst of the cloud computing paradigm, systems requiring intensive computations over large data volumes have relied on the usage of shared data centers to which they transfer their data for processing. This is a powerful scheme for application scenarios that benefit from deep processing and data availability, but it brings in non negligible problems to meet the time requirements of *time sensitive* social computing applications. While not necessarily real-time in the strict sense, such applications have built in penalty (user aversion) if they are not responsive; they must be low-latency; however, the traditional cloud computing architecture is problematic in a number of application domains that are latency sensitive. Precisely, the delay incurred by data propagation across the backhaul is not suited to the needs of applications that require (near) real-time response or high quality of service guarantees. Backhaul data handling latency is severe in the unpredictable occasions where the network throughput is limited. Furthermore, a community deploying such smart applications often finds it difficult to scale the system to the cloud due to economic constraints.

To alleviate these situations, engineers have looked around towards "what is available", i.e., to leverage the computing power of the available near by resources, leading to a profound discussion on the opportunistic usage of the computing resources dispersed in the community. Out of this new scenario, we have identified this new computing approach that we call "Social Dispersed Computing". This is a powerful paradigm that can significantly improve the performance experienced by applications in what concerns latency and available throughput that will, in turn, have an indirect impact on other measures such as the energy consumption.

Unlike cloud computing, resource scalability comes from the participatory nature of the system, i.e., having a larger number of users. The key driver is the social benefit behind the achieved collaboration and the great value obtained from the aggregation of the individual information. Users have to perceive hardly no entry barriers to use these applications; barrier elimination is done by fulfilling the technical requirements of these applications such as providing low cost computation resources, reliability, and data privacy guarantees, over a low overhead management structure that achieves low latency in service provisioning.

Enabling Social Dispersed Computing. The next computing generation is one in which the computing platform and the social applications will be tightly integrated. For example, sharing computing resources can be used as incentive for participation. Moreover, providing the users with the capability of deciding where their computations will run for security and privacy concerns will likely be a major factor for enrolling in application usage.

To enable this, the corresponding transformations are already happening in the communications and persistent storage mechanisms. For example, Software Defined Network [86] addresses the required mechanisms to create a flexible overlay network over dispersed resources. The concept of decentralized distributed ledgers like Ethereum [5] and similar distributed ledgers enable immutable event chronology across computing resources. New concepts such as the inter-planetary files system (IPFS) [29] extend blockchains and the concept of distributed file systems to provide a shared, decentralized, and world-wide persistent information store.

In this paper, we claim that social dispersed computing systems require fog infrastructures to take a predominant role; fog infrastructures will support the mobility of the users, enabling them to offload heavy tasks such as those implying machine learning services to more powerful nodes in their vicinity. However, the great push of relatively very novel computation paradigms such as fog-, edge-, cloud, social, and dispersed computing (among other *computing* paradigms) has resulted in a non-negligible level of terminology

confusion in the community. In different research contributions, the reader can find these terms signifying differently. This paper aims at shedding some light by clarifying the meanings, and defining the boundaries (where possible) of these paradigms, guided by their goals and application-level motivation.
Paper Outline. This paper is structured as follows. Section 2 defines a number of computing paradigms that are simultaneously used nowadays; some of these paradigms are very recent and still the scientific community has not fully agreed on what they actually are; we clarify the paradigms and introduce the concept of *social dispersed computing*. Section 3 describes the concept of social dispersed computing and illustrates it through a set of application scenarios in domains such as energy, social routing and distributed traffic congestion analysis. Section 4 presents the enabling technologies that will allow the development of social dispersed applications; for this, a selected set of computational approaches are presented, followed by a selection of supporting software tools. Section 5 compiles the main challenges for the design and development of social dispersed applications. Finally, section 6 draws the conclusions presented as the opportunities for research.

2 Computing paradigms: Definitions and Evolution

Distributed computing systems date back decades ago enabled by the first communication schemes for remote machines. Figure 1 shows a general view since the 90's; a time where a number of important software and hardware developments came together, and hardware and software schemes start to become more sophisticated and powerful; this led to subsequent productive decades, leading to introduction of a number of new and refined concepts and terms, sometime over short periods of time.

Especially through the last decade, a number of keywords have appeared that imply different computing paradigms such as cloud, mobile cloud, fog, or edge, among others. However, the rapid proliferation of contributions on these paradigms, even prior to the real consolidation of a wide accepted definition for some of them, yielded to some confusion on their definitions. For example, the definition of *edge computing* diverges across a number of works. In [139], edge computing is defined as "any computing and network resources along the path between data sources and cloud data centers"; whereas [147] defines edge computing as a paradigm belonging to the sphere of the pure network infrastructure that connects the user devices (that it refers to as "edge nodes") to the cloud. This last vision of edge computing is also shared by

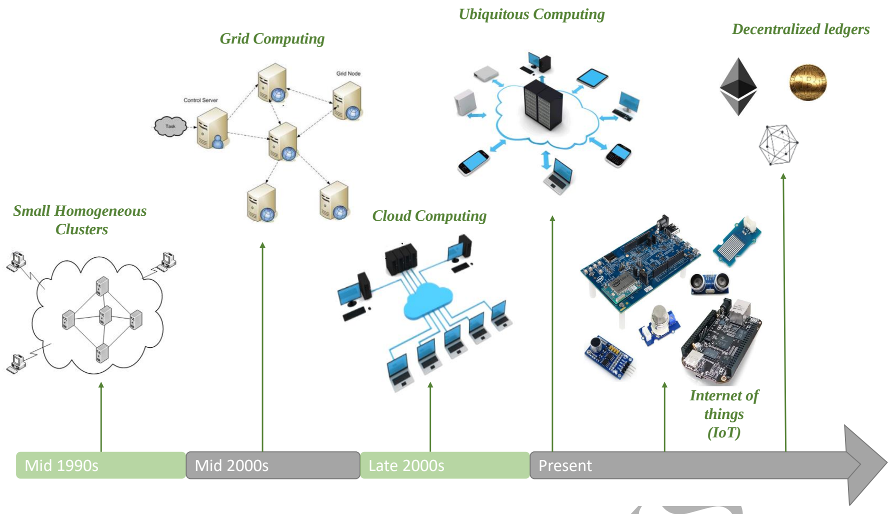


Figure 1: Evolution of Computing: A general view on the evolution of personal devices over the years

[53] although it refers to the user devices as "end nodes" in a more consistent manner.

All these concepts have led us to the point where we are ready to realize the potential of social computing using resources from either the cloud, the fog, or locally dispersed computing resources. Nevertheless, it is first important to clarify the terminology and, for this reason, we initially provide a comprehensive definition of key computing paradigms present in modern literature, with the aim to establish a common understanding. These definitions are based on the most accepted significations of the research community. The goal is to draw a clean separation (wherever possible) among the different computing paradigms also explaining their evolution, motivation, and purpose.

2.1 Cloud computing

Cloud computing (CC) is a service model where computing services that are available remotely allow users to access applications, data, and physical computation resources over a network, on demand, through access devices that can be highly heterogeneous.

In cloud computing [61], resources are rented in an on demand and pay-per-use fashion from cloud providers. Just as a huge hardware machine, cloud computing data centers deliver an infrastructure, platform, and software applications as services that are available to consumers. This facilitates offloading of highly consuming tasks to cloud servers.

The National Institute for Standard and Technology (NIST) is responsible for developing standards and guidelines for providing security to all assets. [111] provides an insight into the cloud computing infrastructure which consists of three service models, four deployment models, and five essential characteristics which are: on-demand self-service, broad network access, resource pooling, rapid elasticity, and measured service.

A cloud service model represents the packaging of IT resources required by the consumers as a service that is provided by the cloud vendor. The three cloud service models are:

- *Software as a service (SaaS)*: The consumers are only provided with the capability to run the applications of the provider, but they have no control over the cloud infrastructures like operating system, servers, or storage.

- *Platform as a service (PaaS)*: The consumers have the capability to deploy either own or acquired applications to the cloud. The consumer does not have any control on the cloud infrastructure, but has control over the deployed application.

- *Infrastructure as a service (IaaS)*: The consumers can use the applications provided on the cloud without the need to download the application to the consumer's computer. Consumers can manage the underlying infrastructure at the cloud such as virtual machines, the operating systems, and other resources.

Additionally, with the wide increase of data processing and storage in the cloud, larger data volumes circulate on the network also increasing their exposure to third parties and attackers. This brings in the need for data security and privacy mechanisms. Data security in particular is a vital challenge that has been given a thought in [117]. Here, the authors have taken a look at the security problem from the perspective of different stakeholders like cloud provider, service provider, and consumer. It also summarizes the security issues in each one of the service delivery models of IaaS, PaaS, and SaaS, where some of the identified problems are responsibility of the cloud vendor while the other issues are that of the consumers. The authors of this work have identified the various holes in the security loop of the cloud computing model, shedding light on which would make the whole model more secure.

By analyzing some of the existing works, it can be seen that these provide the formal definitions of cloud computing, precisely describing the model insights and some of the obstacles in implementing the cloud services. Other than the advantage of the large amount of data storage and analytics capabilities of the cloud, some of its disadvantages (e.g., unreliable data latency, immobility and lack of location awareness) are important drawbacks in some domains; and this has made way to other technologies like mobile cloud computing or fog computing.

2.2 Mobile Cloud Computing

The proliferation of mobile personal devices led to *Mobile Cloud Computing* (MCC). MCC appeared as a natural evolution and enhancement of cloud computing with the goal of offering specific services to mobile users with powerful computational and storage resources. As mobile devices are resource limited as compared to the cloud servers, task offloading strategies are one of the most studied problems. As explained in [58], MCC combines mobile computing, mobile Internet, and cloud computing for providing task offloading.

The literature gives different definitions for MCC as explained in [80]. *Infrastructure based MCC* refers to a model that uses the cloud data centers hardware to serve mobile users; and *ad-hoc MCC* defines the concept of mobile cloud as made up of nearby mobile nodes acting as a resource cloud that grants access to the Internet (including other cloud services) for other mobile users. Using the nearby mobile users has several advantages like the possibility of using a faster LAN network that is comparable to the available servers interconnection inside a cloud data center. Also, MCC is "cloudlet" based, which is defined below.

The paper [20] provides an overview of MCC along with its evolution from cloud computing, and its advantages and disadvantages, as well as its applications. Some of the mentioned noteworthy advantages of MCC are flexibility, storage, cost efficiency, mobility and availability, scalability and performance. Some discussed disadvantages are security, privacy, compliance, compatibility, and dependency. The authors also enumerate a few open challenges faced by MCC which are low bandwidth and QoS issues like congestion, network disconnection and interoperability.

MCC has non-negligible security and privacy challenges which arise due to the integration of mobile devices with cloud computing. Along with the similar security concerns of cloud computing, some new issues on security and privacy also arise as there is a wireless medium for transferring data between the mobile device and the cloud. In [116], the authors identify the main security and privacy challenges as data security, virtualization security, partitioning and offloading security, mobile cloud application security, mobile device security, data privacy, location privacy and identity privacy, and solutions to each of these issues have also been discussed by citing prior literature work. Given the increase in the number of mobile users and applications, security and privacy requirements are vital for MCC; dealing with them increases the computation and communication overhead which has to be dealt with by the users.

With the integration of mobile devices and cloud computing, MCC overcomes the limitation of immobility and lack of location awareness in cloud computing; also, it provides an attractive and convenient technology of moving all the data rich mobile computation to the cloud. However advantageous this idea of MCC may look, there are still open issues like the associated high network latency and power consumption of data transmission from the mobile devices to the cloud, which are not handled by MCC.

2.3 Cloudlet

Cloudlet is defined as resource rich and trusted computing devices near the vicinity of mobile users which can be used to process data sent to them over a local area network. It is a major technological enabler for MCC, defined at the convergence of MCC and cloud computing. It defines a virtualized architecture [135] as a computational resource accessible by mobile users at range, i.e., within their physical vicinity; this has the objective of empowering mobile devices providing them the capabilities to access computationally intensive services that could not be run by their own limited resources. Examples of such as services are speech recognition, processing of natural language, machine learning, or augmented reality.

As discussed in [135], even with the increased computation and storage capacity, mobile devices are not able to process rich media content locally with their own resources. MCC aimed at solving the above issue by offloading all the data from the mobile device to the cloud for computation. However, this is not feasible for applications with tight latency requirements (i.e., real time applications) that led to giving way to the concept of cloudlet.

Additionally, as discussed in [149], mobile users can utilize the cloudlets VM to run the required software applications closer to the mobile devices. Apparently, this idea of cloudlet aims to solve the latency issues by moving the virtual machine closer to the mobile devices; however, there is a notable drawback of mobile users being dependent on network service providers to deploy cloudlets into the LAN network for the mobile devices to utilize them. The authors in [149] also provide insight into the architecture of cloudlets where the applications are managed at the component level; also this paper classifies the architecture into two categories of ad hoc cloudlet and elastic cloudlet. The idea of the discussed architecture is evaluated by implementing it for a use case of augmented reality.

Effectively, [149] and [135] discuss the evolution of the cloudlet and place it between cloud computing and MCC paradigms. Briefly, in cloudlet, the jobs of the mobile users are not transmitted all the way to the cloud but to a nearby cloudlet; this fact tends to reduce the power consumption of mobile devices and also the transmission delay. Thus, cloudlet makes an advantageous evolution from MCC.

Additionally, [77] merges the concepts of MCC and cloudlet to reduce the mobile device power consumption and also the network communication latency. The proposed architecture merges both MCC and cloudlet, providing an advantage as real time processing can be run on the cloudlet and other non-real time data processing and storage could be run on the cloud. [77] provides the statistics to support the claim of reduced power consumption and transmission delay.

2.4 Internet of Things

Internet of Things (IoT, that includes IoE – Internet of Everything) is an extension of the classical sensor network paradigm, providing support for large scale sensor data aggregation and cloud based data processing and decision support systems.

The concept of *pervasive computing* emerged before IoT to refer to the provisioning of computation *anytime, anywhere*. One of the novelties of this concept was the fact that computation devices could be personal devices, among others. This idea was also expressed and referred to as *ambient intelligence* or *everywhere*.

IoT is a similar concept except that in IoT the emphasis is placed on the physical object. The range of possible devices in IoT was enlarged as compared to those considered in pervasive computing. As technology improved, IoT vision was to flood the market with computation nodes that were deeply immersed in the environment: from sensors to small embedded computers that could be connected to the Internet as direct and uniquely addressable end points.

The primary evolution in the IoT paradigm compared to the sensor networks is support for *complex event processing* (CEP) [47] which is typically executed on the integrated cloud platform. CEP engines can be run over the intermediate IoT node resources in the network and queries can be placed on the incoming continuous data streams from the end devices[1] like (e.g. sensors, RFID). As compared to the previous paradigm where end nodes sent data streams to the cloud that processes them, performing such processing on the available IoT nodes could reduce the latency and bandwidth requirements of the IoT network.

An overview of CEP for IoT and its applications is provided in [42], consisting of a deep insight into the distributed CEP architecture based on client-server model which can be realized on the IoT devices to perform queries like filtering, passing data, and placing windows on the incoming data. Some of the advantages of using CEP over IoT are: (1) distributed CEP in the network will balance the workload better; (2) ease of CEP engine deployment; and (3) the data traffic can be significantly reduced by removing unwanted data using queries of filtering, data passing, etc.

Additionally, there are other works like [66] which are dealing with the idea of distributing the data analytics between the IoT nodes and the cloud. For example, they use genetic algorithms to optimize the query mapping to the end devices. While the integration of IoT and CEP is a well studied concept, the challenges of security, privacy, adaptability, scalability and interoperability still remain.

To deal with the complexity and heterogeneity of IoT environments, a number of high level flexible layered architectures have been contributed. Heterogeneity has led to different sets of requirements, with different needs for complexity and varied performance, which has affected the design of architectures. This has led to a scenario in which solutions have not yet converged to a reference model, that causes limited interoperability across systems [87].

2.5 Cyber-physical Systems

Cyber-Physical Systems (CPS) are networked systems in which the computational (cyber) part is tightly integrated with the physical components. That is, the computational components sense the state of the system and then provide continuous feedback for controlling the system. Physical components include energy sources, transmission and distribution lines, loads, and control devices. Cyber components include energy management systems (EMS), supervisory control and data acquisition (SCADA) systems, and embedded implementations of control algorithms. The interplay of computational and physical systems yields new capabilities. The network is a

[1] By end devices, we refer to the nodes at the leaf position of the information flow graph, typically intelligent sensors, smartphones, embedded computers, etc.

key component in cyber-physical systems as it provides the backplane that guarantees timely transmission of the information (from the physical to the computational world) and of the commands (from the computation to the physical world).

Traditionally, these systems had been mostly self-contained in the sense that they have included all needed computational elements with little interaction with external elements. For example, the traditional architecture for the Smart Grid transfers all SCADA data to centralized utility servers [145]. An evolution of design of such systems arrived with the arrival of the cloud computing paradigm as many of the analytics functions were deployed in the cloud [142]. However, even with the availability of on-demand resources in the cloud, the critical CPS often are unable to transfer the time-critical control tasks to the cloud due to communication latencies [37][39]. This centralized SCADA architecture is changing with recent developments like Fog Computing [156][67], which have advertised the use of dual purpose sensing and computation nodes (i.e., end nodes) that are closer to the physical phenomenon that is observed or analyzed. For example, the SCALE-2 [30] platform provides the capability to run air-quality monitoring sensors, the Paradrop architecture [150] provides the capability to run containerized applications in network routers.

As a direct consequence of the evolution of the computing paradigm from 'central data-centers' to 'shared cloud computing resources' to 'distributed edge (end) computing resources plus shared cloud resources', critical CPS like Smart Grids can distribute the intelligence further down into the network, away from the centralized utility servers. For example, this capability provides us with the means to build energy management applications of the future that are both distributed and coordinated, with heavy reliance on communication and coordination among local sensing and control algorithms, while also obeying strategic energy management decisions made on a higher level of the control hierarchy. We discuss this concept of providing "scalable" and "extensible" computation services near the physical process — fog computing— next.

2.6 Fog Computing

Fog computing (FC) was introduced to solve the problem of having billions of IoT devices and cyber physical systems that cannot operate by simply having connectivity to servers in the cloud; instead, computations are pushed closer to these end nodes/devices. Unlike the traditional computation model, the fog computing model, pioneered by the Open Fog Consortium, suggests the use of shared computation servers, similar to the vision of cloudlet described by [135]. However, the key difference lies in the software as a service pioneered by fog computing. For example, instead of just providing the computation resources, a fog computing machine often provides machine learning stack as a service [7]. Also, a difference with respect to cloud is that fog computing supports user mobility. Nevertheless, fog and cloud are not independent paradigms as in a fog computing environment there is the need for interacting with the cloud to achieve coherent data management.

As mentioned in [155], the unresolved issues in cloud computing of latency and mobility have been overcome by providing services and elastic resources at the end of the information chain, close to the sensors. [155] defines fog computing and discusses the issues related to it like fog networking, quality of service (QoS), interfacing and programming model, computation offloading, accounting, billing and monitoring, provisioning and resource management, and security and privacy. Along with providing insights into the issues related to fog computing, it also mentions paradigmatic applications like augmented reality (AR) and Real-time video analytics, content delivery and caching, and mobile big data analytics which will promote fog computing.

All of the computation paradigms discussed have big security and privacy challenges. Some of the main security issues faced by FC [119] are trust, authentication, secure communication, end user's privacy, and malicious insider attacks. A number of papers have contributed to identifying the security and privacy concerns of FC, and similarly a number of solutions for each of the above stated security challenges have also been analyzed in the literature.

Similar to [119], the paper [144] mentions different security issues in FC. All smart appliances (e.g. fog computing nodes like smart meters) have an IP address and here authentication problems are a big threat; a malicious attacker may try to hack the device and tamper the data (e.g. in case of smart meter, provide false meter reading) associated with it. Similar to authentication problems, man-in-the-middle attack is also a prominent type of attack on fog computing nodes (FCN), where the devices may be compromised or replaced by fake ones. This problem arises due to the fact that the FCN under this type of attack utilize only a small amount of the processor and memory, and normal intrusion/ anomaly detection techniques will not be able to detect it. The authors also provide an insight into the solution to the man-in-the-middle attack. Privacy issues in fog computing and different solutions available in the literature have also been mentioned in this work.

Overall, it must be acknowledged that fog computing provides a number of advantages that are of key importance for most applications: low latency, location awareness, real time operation, heterogeneity, and end device mobility; all these make it an attractive computation paradigm. But, the security and privacy challenges of trust, authentication and man-in-the-middle attack discussed above makes it challenging to implement the FCN in daily life applications.

2.7 Edge Computing

Edge computing (EC) is an overloaded concept, defined differently across the literature. The most commonly mentioned meaning of *edge* is that of *end*, meaning that edge computing is carried out by the end devices (also called edge devices in the associated works). Although this concept pulls the focus away from the network elements and their associated challenges, it is probably the most extended definition up to the present time.

However, the networking community has started to use *edge* computing to refer to the computation performed by the network elements. If we view the Internet as a graph that connects computation nodes (computers), the edge meaning is assigned to the connecting line between the central nodes (cloud servers) and the leaf nodes (the devices at the end of the network). Here, edge computing refers to the computation done at the network backhaul.

After presenting both usages of *edge computing*, we use this term in the networking sense, so that we refer to end devices (or user devices) as the leaf nodes of the Internet and to edge computing as to the computation done at the network elements and backhaul that will support to off load and speed up the service time to end devices by partly performing heavy computations in the network segments.

In the first meaning of the term, the idea behind edge computing is to perform computation and storage locally within the resources available at the end devices. For this type of nodes, [139] targets at addressing the potential issues of response time requirements, battery life constraints, data security and privacy, and bandwidth reduction; this paper also discusses the evolution of the edge computing from the concepts of cloud computing and IoT, providing a definition for edge computing and several case studies that support this paradigm and show the inherent advantages that it offers.

Similarly, paper [53] provides an insight into EC along with the comparison among the different EC implementations of fog computing, mobile edge computing, and cloudlets. Some simple differences among the three are:

- Characteristics of nodes: Fog computing nodes use off-the-shelf devices and provide them with computation and storage capabilities which make them slower as compared to the dedicated devices of mobile edge computing and cloudlets.

- Proximity to end devices: Fog computing nodes may not be the first hop for end devices due to the use of off-the-shelf computing devices;

whereas for MEC and cloudlet, the devices can connect directly to the end nodes using WiFi for cloudlets and mobile base station for mobile edge computing.

- Access/communication mechanisms between the devices: Fog computing nodes can use WiFi, Bluetooth or even mobile networks; mobile edge computing devices can only utilize mobile networks; and Cloudlets use WiFi.

- Diversity and heterogeneity in the off-the-shelf devices: the Fog computing paradigm requires an abstraction layer; whereas mobile edge computing and cloudlets do not require this because of the dedicated connections that devices they use.

Additionally, the authors have also mentioned the use case based selection of the three edge computing implementations in terms of power consumption, access medium, context awareness, proximity, and computation times.

As a result from the literature analysis, it appears that the genesis of EC has made way to other EC implementations of fog computing, mobile edge computing, and cloudlets which tend to tackle the disadvantages of CC and MCC. However, there are several open issues [98] of EC which are security and privacy, resilience, robustness, openness in the network and, multi-services and operation.

2.8 Mobile Edge Computing

Mobile-Edge Computing (MEC) was motivated by the growth of the network traffic generated by the proliferation of smart phones and their applications that require intensive data exchange and processing. MEC intends to reduce the latency and to support location awareness in order to increase the capacity of the applications that run on mobile devices. MEC started development in 2014 led by ETSI[2] for achieving a sustainable business strategy [135]; for this, it brought together mobile operators, service providers, mobile users, and over-the-top (OTT) players. Different metrics can be improved by deploying services over MEC. On the functional side: latency, energy efficiency, throughput, goodput, packet loss, jitter, and QoS. On the non-functional side: service availability, reliability, service load, and number of invocation requests. MEC servers are located near the base stations. Smart devices offload activities and the cellular data and offloaded activities are

[2] European Telecommunications Standards Institute. www.etsi.org

processed on such servers; them, the edge servers decrease the traffic and congestion on the backhaul.

As discussed above, MEC aims at placing the computational and storage resources at the mobile base stations so that mobile users can widely use the additional features it has to provide. [27] provides technical insight into MEC along with its limitations by identifying the applications of MEC. Various applications and use cases of content scaling, offloading, augmentation, edge content delivery, aggregation and local connectivity are evaluated in terms of power consumption, delay, bandwidth and scalability. A few of the listed advantages of MEC for different stakeholders of end users, network operators and application service providers are: (1) End users benefit from reduced communication delay; (2) Network operators benefit with bandwidth reduction and scalability; (3) Application service providers benefit with faster service and scalability; and (4) Augmentation enables the application providers to integrate cellular network specific information into the application traffic.

A comprehensive overview of MEC is found in [107] that gives an introduction to features of MEC along with its paradigm shift from MCC. A comparison of MEC and MCC has been made to support the advantages of paradigm shift from MCC. The advantages of MEC like low latency, mobile energy saving, context awareness and, privacy/ security enhancement are discussed along with examples. Some of the mentioned technical challenges of MEC are: security, network integration, application portability, performance, regulatory and legal consideration, resilience and operation. The literature also mentions some use scenarios of MEC like video stream analysis, augmented reality, IoT, and connected vehicles.

In contrast to cloudlet which is available to specific users in the vicinity of the cloudlet, MEC is available to all mobile users as MEC servers are deployed in mobile base stations to deliver additional features such as location and mobility information.

Fog or cloudlet nodes are managed typically by individuals and can be deployed at any location that they judge convenient. MEC servers are owned by mobile operators; servers have to be deployed near the base stations to facilitate that users have access to the mobile network over the RAN [134]. MEC model has been prototyped on a few scenarios such as edge video orchestration in which users access live video streams enabled by an orchestration application running on a MEC server. MEC servers can be deployed at different locations on the networking infrastructure: an LTE base station [3], 3G Radio Network Controllers (RNC), or a mix of both.

[3] Long-Term Evolution (LTE) is a telecommunications standard –a registered trademark

Security and privacy issues are shared by fog, cloud, and MEC. Moreover, in MEC the congestion of a server may affect the service provided to a number of mobile users, resulting in high monetary costs. Therefore, increasing the computation power at the edge servers is a real need.

2.9 Mist Computing

Mist Computing is a concept explained in [128]. There is lack of consensus as to the precise definition of *mist computing*. In some works, mist computing is defined as the paradigm that takes advantage of every processing capacity available everywhere, from the end nodes (sensors and actuators) to the cloud servers. Some of these works also provide definitions for other concepts that collide with the mainstream trend, e.g., edge computing acquires fog computing capabilities [76]. As there is no clean definition of what mist actually provides, we are inclined to either use cloud, fog, or edge.

As in [129], fog computing performs the computation at the network using the gateway devices, but in mist computing this is performed by the actual end devices, i.e., sensors and actuators. We know that the closer the computation is to the end devices, the bigger is the decrease in the network latency and transmission delay, which improves the user experiences in real time applications.

2.10 Social Computing

Social Computing [81] is a paradigm for analyzing and modeling social behaviors of users on media and platforms to extract added value information and create intelligent and interactive applications and data. It involves a multi-disciplinary approach that encompases computing, sociology, social psychology, communication theory, computer-science, and human-machine interaction (HMI). For this purpose, social computing focuses essentially on studying the relations among people within a group to analyze how the information flows; the collaboration manner to extract positive and negative patterns; and how communities are built and how grouping is achieved. The target systems for analysis are social media, social networks, social games, social bookmarking and tagging systems, social news and knowledge sharing, among others.

of ETSI– for high-speed wireless communication in mobile devices and data terminals; it increases the capacity and speed by using a different radio interface together with core network improvements

Among these scenarios, *social computing* and social software are capable of providing big data that can be processed and analyzed with complex algorithms and computation techniques [81] capable of extracting essential social knowledge that creates high value for society, industry, or individuals. Social computing is a part of computer science at the confluence area between social behavior and computational systems. By means of using software systems and computer science technology, social computing recreates social conventions and social contexts. Software applications for social communication and interaction are the building block of social computing and illustrate this concept. Among these software elements, one may find public web based content, blogs, email, instant messaging, social network services, wikis, social tagging and bookmarking.

Since the wide availability of Internet and powerful personal computers, social computing took a phenomenal growth. This paradigm shifts the computing towards the end of the network for the end users to engage in social communities, share information and ideas, and collectively build and use new tools. Social communities with common ideas, tools and interests are formed which can improve the experience of using tools and sharing common problems and solutions. As an example, Wikipedia is an open source encyclopedia that works like an information sharing tool formed by collaborative authoring which can be reviewed and changed upon the feedback of users. Though this social tool helps the community in sharing information through a common platform called wiki, the credibility of information is at stake, as it is an open source tool with collaborative authorship. Some other notable examples of social computing platforms are YouTube, Word press, Tumblr, Facebook, Twitter, or LinkedIn.

2.11 Dispersed Computing

Dispersed computing [18] involves algorithms and protocols for mission-aware computation and communication across broad-scale, physically dispersed objects for designing scalable, secure, and robust decision systems that are collectively driven by a global mission. These systems can operate under highly variable and unpredictable (possibly also degraded) network connectivity conditions. For this reason, dispersed computing envisions opportunistic and convenient design of mobile code and data relations as needed by the users, the applications, or the mission.

For cloud computing and mobile computing, users offload the real time data on to the cloud for processing and data analytics. We have also discussed a few limitations of high network latency and transmission delay, that lead to the genesis of the different paradigms of edge computing, fog computing and MEC which is based on the idea of utilizing the computational resources of the edge devices in the network to process the data locally. Similar to this idea, dispersed computing seeks to provide a scalable and robust computing system which collectively uses heterogeneous computing platforms to process large data volumes. This paradigm is typically deployed in situations where there is degraded network connectivity that leads to higher data latency and transmission delay.

Among the first works on dispersed computing, we find [143] that defines the term as an alternative model derived from the consolidation of a number of contributions on data transmission, data storage and code execution. Still that work is very preliminary and very much targeted to surveying the existing distributed computing models according to various criteria and highly related to cloud.

Other meanings of disperse computing rather point at the edge computing elements, such as DARPA's definition [18] where NCPs (the network control points) are placed at the core of the computations.

Dispersed computing systems run software partly inside the programmable platforms within the network, the NCPs. As mentioned earlier, NCPs are capable of running code for both, users/applications and for the network protocol stack. For implementing the dispersed computing paradigm, the application-level logic will need resources available at the end points (the computation devices) and at the NCPs.

3 Social Dispersed Computing

In this paper, we coin the term ***social dispersed computing*** that is at the intersection of *social computing* and *dispersed computing*. On the one hand, *dispersed computing* [18] has the goal of providing scalable, secure, and robust decision systems that are collectively driven by a global mission. Dispersed computing is a computing paradigm for designing systems that can operate under highly variable and unpredictable (possibly also degraded) network connectivity conditions. For this, such a computing paradigm envisions opportunistic and convenient design of mobile code and data relations as needed by the users, the applications, or the mission. On the other hand, the *social dispersed computing* paradigm takes an agent or actor based approach, connecting the users with each other with messages, enabling them to obtain globally useful analysis, while performing local computations. Further, decisions on what users do are influenced not only by the users' personal preference and desire but also by what other users are doing.

These models demand complex, flexible, and adaptive systems, in which components cannot simply be passive nor can reactive entities be managed by only one organization [49]. Nevertheless, instead of being a solitary activity, *computation becomes* rather an inherently *social activity*, leading to new ways of conceiving, designing, developing, and handling computational systems [141]. Considering the emergence of distributed paradigms such as web services, service-oriented computing, grid computing, peer-to-peer technologies, autonomic computing, etc., large systems can be viewed as the services that are offered and consumed by different entities, enabling a transactive paradigm.

Formally, social dispersed computing applications can be approximated as multi agent systems. For example, they can be thought of as collections of service-provider and service-consumer components interlinked by dynamically defined workflows [106]. *Agents* are autonomous entities with given behaviours that interact with other agents that also have their own behaviours. As a result of these interactions, individual behaviours (or even objectives, preferences, etc.) may be affected, emerging a global (or aggregated) behaviour of the whole system. Intelligent software agents are a new class of software that act on behalf of the user to find and filter information, negotiate for services, easily automate complex tasks, or collaborate with other software agents to solve complex problems. This concept of *intelligent agent* provides support to build complex social dispersed computing systems as components with higher levels of intelligence, which demand complex ways of interaction and cooperation in order to solve specific problems and achieve the given objectives. However, while procedures , functions, methods and objects are familiar software abstractions that software developers use every day, Software agents, are a fundamentally new paradigm unfamiliar to many software developers. Thus, new platforms and programming abstractions are required. We describe some of these paradigms in the sections on market based approaches later in the paper.

3.1 Multi-Agent Systems

It should be noted that this concept of social dispersed systems borrows heavily from the paradigm of multi-agent systems and integrates social behaviors and incentives (to encourage participation) in to the mix. **Multi Agent Systems** are one of the most important and exciting research areas that have arisen in the field of Information Technologies in the last decade [105]. According to [152], an agent is defined by its flexibility, which implies that an agent is *reactive* as it must answer to its environment; *proactive* as it must try to fulfill its own plans or objectives; and *social* because an agent has to be able to communicate with other agents by means of some kind of language. A Multi Agent System consists of a number of agents that interact with one-another [151].

The most promising application of MAS technology is its use for supporting open distributed systems [105]. Open systems are characterized by the heterogeneity of their participants, non-trustworthy members, existence of conflicting individual goals and a high possibility of non-accordance with specifications [25]. The main feature of agents in these systems is autonomy. It is this autonomy that requires regulation, and norms are a solution for this requirement. In these types of systems, problems are solved by means of cooperation among several software agents [106]. Norms prescribe what is permitted, forbidden, and mandatory in societies. Thus, they define the benefits and responsibilities of the society members and, as a consequence, agents are able to plan their actions according to their expected behaviour.

When developing applications based on the new generation of distributed systems, developers and users require infrastructures and tools that support essential features in Multi Agent Systems (such as agent organizations, mobility, etc.) and that facilitate the system design, management, execution, and evaluation [50, 59]. Agent infrastructures are usually built using other technologies such as grid systems, service-oriented architectures, P2P networks, etc. In this sense, the integration and interoperability of such technologies in Multi Agent Systems is also a challenging issue in the area of both tools and infrastructures for Multi Agent Systems. What is more, agent technologies can provide concepts and tools that give possible answers to the challenges of practical development of such systems by taking into consideration issues such as decentralization and distribution of control, flexibility, adaptation, trust, security, and openness [36]. Finally, in order for Multi Agent Systems to be included in real domains such as media and Internet, logistics, e-commerce and health care, infrastructures and tools for Multi Agent Systems should provide efficiency, scalability, security, management, monitoring and other features related to building real applications.

3.2 Social dispersed computing illustration

Let us consider three examples: two from the transportation domain and one from the energy domain. to illustrate the concept of social dispersed computing.

3.2.1 Next Generation Electrical Energy Systems

Transactive energy systems (TES) [94, 83, 46, 112] have emerged in anticipation of a shift in the electricity industry, away from centralized, monolithic business models characterized by bulk generation and one-way delivery, toward a decentralized model in which end users play a more active role in both production and consumption [112]. The main actors of this system are the consumers, which are comprised primarily of residential loads and prosumers who operate *distributed energy resources* (DERs). Examples of such DERs include photovoltaics, batteries, and schedulable loads (electric vehicle charging, laundry, etc.). Additionally, a *distribution system operator* (DSO) manages the connection between the microgrid and the primary grid. Such installations are equipped with an advanced metering infrastructure, which consists of TE-enabled smart meters. In addition to the standard functionality of smart meters (i.e., the ability to measure line voltages, power consumption and production, and communicate these to the DSO), TE-enabled smart meters are capable of communicating with other smart meters, have substantial on-board computational resources, and are capable of accessing the Internet and cloud computing services as needed. Examples of such installations include the well-known Brooklyn Microgrid Project [17].

At its core, transactive energy systems are market based social applications that have to dynamically balance the demand and supply across the electrical infrastructure [112]. In this approach, customers on the same feeder (i.e., those sharing a power line link) can operate in an open market, trading and exchanging generated energy locally. Distribution System Operators can be the custodians of this market, while still meeting the net demand [48]. Implementing such systems requires either a centralized or decentralized market framework that is robust, resilient, and secure. Fog computing resources provide ideal opportunity to schedule the operation of the market activities in the community as most of the activity remains within the community and each home has access to a set of smart inverters and computers attached to the smart inverters that can be part of the fog computing layer.

3.2.2 Social Mobility

Social routing platforms address the problem of urban transportation and congestion by directly engaging individual commuters. Due to widespread use of smart devices, users are becoming active agents in the shared mobility economy. This enables the use of algorithms for designing active incentives that encourage users to take mobility decisions considering the overall system effect, rather than myopic individual utilities, focusing on what is best for each individual from his or her local perspective, as implemented by commercially available mobility solutions [133].

Such services require a platform to information sharing, and transactive platform that: (a) provides multimodal routing algorithms, which extend existing optimization techniques for solving the multimodal transit problem by incorporating probabilistic representations of events in cities, creating a near-optimal distributed algorithm by employing sub-modularity and folding incentive mechanisms into the optimization problem; (b) provide high-fidelity analytics and simulation capabilities for service providers, informing them about how users are consuming transportation resources, which enables them to develop mechanisms for improving services; and (c) provide an immutable and auditable record of all transactions in the system.

Again a market-based distributed system running across these agents will be able to create a dynamic offer with incentive-based route assignment logic that can ensure that transportation resources are shared efficiently without causing congestion. Clearly, such a platform is also an extension of the transaction management platform by: (1) making individual consumers the participants; and (2) making the apps running on their smart phones the transaction agents and the transaction management platform provided by the transportation agency.

A solution to this problem requires a social computing and information sharing platform that overcomes the incentive gap between individuals and municipalities. This platform must offer mixed-mode routing suggestions and general system information to travelers and—in turn—supply service providers with high-fidelity information about how users are consuming different transportation resources. At the same time, this system must also consider the investment required by the cities in the computing infrastructure required to solve the problem at scale. Alternatively, a social dispersed computing approach that utilizes the various edge computing resources available in the city, including the mobile devices of the commuters, can be employed by municipalities to improve efficiency within their cities with little investment.

However, this precisely leads to the problem of secure and trustworthy computing. Privacy of individuals is an important aspect of this solution; the usage of individuals' smart devices as both data sources and computational resources could expose the end-users to a risk of privacy breach. Seemingly innocuous data, such as transit mode or route choice, can lead to inferences of private information, such as real-time tracking of an individual's position [85], likelihood of affairs [118], and forecasting trip destinations [52]. Therefore again localized computing resources which are managed under the legal jurisdiction are more attractive to use for implementing the transaction management.

3.2.3 Distributed Traffic Congestion Analysis

Another example is traffic congestion analysis in cities. Traffic congestion in urban areas has become a significant issue in recent years. Because of traffic congestion, people in the United States traveled an extra 6.9 billion hours and purchased an extra 3.1 billion gallons of fuel in 2014. The extra time and fuel cost were valued up to 160 billion dollars [136]. Congestion that is caused by accidents, roadwork, special events, or adverse weather is called non-recurring congestion (NRC) [68]. Compared with the recurring congestion that happens repeatedly at particular times in the day, weekday and peak hours, NRC makes people unprepared and has a significant impact on urban mobility. For example, in the US, NRC accounts for two-thirds of the overall traffic delay in urban areas with a population of over one million [103].

Driven by the concepts of the Internet of Things (IoT) and smart cities, various traffic sensors have been deployed in urban environments on a large scale, and many techniques for knowledge discovery and data mining that integrate and utilize the sensor data have been also developed. Traffic data is widely available by using static sensors (e.g., loop detectors, radars, cameras, etc.) as well as mobile sensors (e.g., in-vehicle GPS and other crowdsensing techniques that use mobile phones). The fast development of sensor techniques enables the possibility of in-depth analysis of congestion and causes.

The problem of finding anomalous traffic patterns is called traffic anomaly detection. Understanding and analyzing traffic anomalies, especially congestion patterns, is critical to helping city planners make better decisions to optimize urban transportation systems and reduce congestion conditions. To identify faulty sensors, many data-driven and model-driven methods have been proposed to incorporate historical and real-time data [132, 104, 158, 64]. Some researchers [78, 148, 154, 84] have worked on detecting traffic events such as car accidents and congestion using videos, traffic, and vehicular ad hoc data. There are also researchers who have explored the root causes of anomalous traffic [102, 153, 45, 89, 90, 19].

Most existing work still mainly focuses on a road section or a small network region to identify traffic congestion, but few studies explore non-recurring congestion and its causes for a large urban area. Recently, deep learning techniques have gained great success in many research fields (including image processing, speech recognition, bioinformatics, etc.), and provide a great opportunity to potentially solve the NRC identification and classification problem. However, the state of the art still is to collate the data into a server and then perform the NRC classification periodically. The concept of Mobile Edge Computing and Fog Computing provide a new opportunity.

Consider, a network of micro-devices running on the transit buses, kiosks at the bus stops and the metro data center can be used to not only provide the transit schedule analysis services to the end customer but can also be used to provide analysis of non-recurring congestion (NRC). Compared with the recurring congestion that happens repeatedly at a particular time in the day, weekday and peak hours, NRC usually shows specific patterns associated with the causing events. It is important to identify and correlate the traffic data gathered by individual road sensors, including cameras and solve a coordinated analysis of traffic conditions across the region. Clearly, sending all the data in real-time to the cloud or the metro data center is inefficient and the data should be only sent when the likelihood of NRC is high. Detection of NRC events is important in communities as the local traffic operation centers and emergency responders can take proactive actions. Once an NRC event is detected, it is possible to do further analysis to identify if it can be explained due to an existing event or if it can be explained as a failure of one or more traffic sensors [65], which can then be repaired.

4 Enabling Social Dispersed Computing

While fog computing, edge computing, and mobile edge computing provide the required computation resources, the resilience, timeliness, and security requirements impose need of additional middleware technology. While traditionally middleware was thought of as the “networking” glue, these days middleware is often used as the term to also describe “ useful platform” services. These platform services provide reusable capabilities like distributed transaction, time synchronization, fault-tolerance, etc. This section describes some of these core computation services. The reader must think of them as core-enablers, which when combined appropriately with the underlying computation substrates enable useful social dispersed computing applications.

4.1 Distributed Transaction Management

At its core, agents in the social dispersed computing domain are executing a set of related operations. These operations and their sequence can be grouped into a transaction to enable fault tolerance, specially providing the capability of roll back.

A *distributed transaction* is a set of operations that involve two or more networked nodes that, in turn, provide resources that are used and probably updated by the operations. In a traditional transaction, there is the notion of the *transaction manager* that manages the execution of the constituent operations and their access to the distributed resources. Typical transaction systems such as [51, 114] use techniques for faster execution such as compensating transactions, optimism, and isolation without locking. However, the concept of centralized management will have to be revisited for social dispersed applications; these are highly distributed applications, potentially involving large numbers of participants with high mobility, that produce large data volumes, and that manage data selectively.

Social computing applications are transactive by nature because they often involve exchange of digital assets between participants. The state transition of the system also depends upon the confirmed past state of the system. Examples include transactive ride-share systems [157], transactive healthcare systems [26], and transactive energy systems [83, 46, 112]. Typically, there are three different kinds of subsystems required to settle the transactions in a social dispersed computing application.

The first subsystem is a distributed ledger (e.g. Blockchains), which is responsible for keeping track of (and log) all events of interest. For instance, in the energy domain these events are trades, energy transfers, and financial transactions. In the health care domain, the events record the time of access of the health care data. The data is not stored in the blockchain due to the size and privacy issues. Rather, the data is stored in the second layer, which can be implemented by either a cloud or a decentralized storage service like Storj [3] or IPFS [122]. The second subsystem is the IoT layer, which is responsible for sensing and control. The third subsystem is the communication layer and is typically implemented using messaging middlewares like MQTT [123] or DDS [124].

A new enabling technology for transaction management can be IPFS (InterPlanetary File System) that is a peer to peer distributed file system with the goal of connecting all computing devices through a single global file system. In IPFS, nodes do not need to trust each other: it uses a distributed hashtable and a self-certifying namespace, and has no single point of failure. IPFS is similar to the web but it tries to mimic the exchange of files through a Git type of repository for all devices by providing a content-addressed block storage model with content-addressed hyper links. This connection type will form a data structure (Merkle DAG) that can be used for providing blockchains, versioned file systems, or a permanent web.

4.2 Blockchain

Blockchains combine the storage of transaction information with advanced protocols in a way that ensures that there is a consensus on the operations that were executed. It is a public database where new data are stored in a container called a *block*. Each block is added to an immutable chain that has data added in the past. Data stored in blockchains can be of any type. The perfect illustration of this technology is inevitably related to Bitcoins, a cryptocurrency whose transactions are recorded chronologically and publicly on the database, where each block is a transaction.

The evolution of blockchain technology ancestors until today is depicted in figure 2.

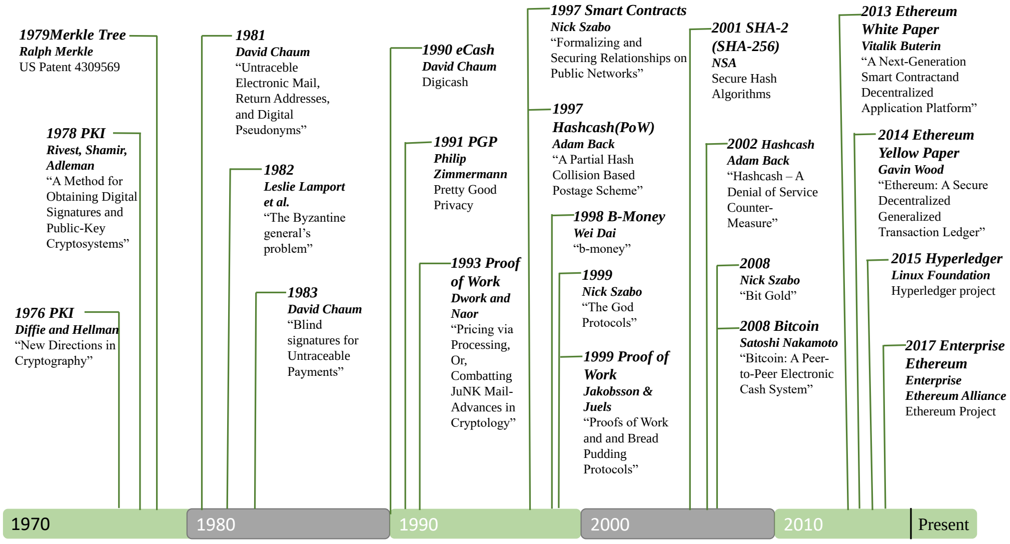

1979Merkle Tree
Ralph Merkle
US Patent 4309569
1978 PKI
Rivest, Shamir, Adleman
"A Method for Obtaining Digital Signatures and Public-Key Cryptosystems"
1976 PKI
Diffie and Hellman
"New Directions in Cryptography"
1981
David Chaum
"Untraceble Electronic Mail, Return Addresses, and Digital Pseudonyms"
1982
Leslie Lamport et al.
"The Byzantine general's problem"
1983
David Chaum
"Blind signatures for Untraceable Payments"
1990 eCash
David Chaum
Digicash
1991 PGP
Philip Zimmermann
Pretty Good Privacy
1993 Proof of Work
Dwork and Naor
"Pricing via Processing, Or, Combatting JuNK Mail-Advances in Cryptology"
1997 Smart Contracts
Nick Szabo
"Formalizing and Securing Relationships on Public Networks"
1997 Hashcash(PoW)
Adam Back
"A Partial Hash Collision Based Postage Scheme"
1998 B-Money
Wei Dai
"b-money"
1999
Nick Szabo
"The God Protocols"
1999 Proof of Work
Jakobsson & Juels
"Proofs of Work and and Bread Pudding Protocols"
2001 SHA-2 (SHA-256)
NSA
Secure Hash Algorithms
2002 Hashcash
Adam Back
"Hashcash – A Denial of Service Counter-Measure"
2008
Nick Szabo
"Bit Gold"
2008 Bitcoin
Satoshi Nakamoto
"Bitcoin: A Peer-to-Peer Electronic Cash System"
2013 Ethereum White Paper
Vitalik Buterin
"A Next-Generation Smart Contractand Decentralized Application Platform"
2014 Ethereum Yellow Paper
Gavin Wood
"Ethereum: A Secure Decentralized Generalized Transaction Ledger"
2015 Hyperledger
Linux Foundation
Hyperledger project
2017 Enterprise Ethereum
Enterprise Ethereum Alliance
Ethereum Project
1970
1980
1990
2000
2010
Present

Figure 2: Evolution of Blockchain

Current transactions require that people trust on a third party to complete the transaction. This can be a bank or a national authority for the case of transactions involving money.

Blockchain technology is radically challenging the current way of operating transactions. Blockchain relies on the use of mathematical tools and cryptography to provide an open decentralized database as a global and decentralized source of trust recording every transaction that involves value, money, goods, property, work, or even votes. Every transaction is recorded on a public and distributed ledger accessible by anyone who has an Internet connection. It consists of creating and managing a record whose authenticity can be verified by the entire user community. Distributed property and

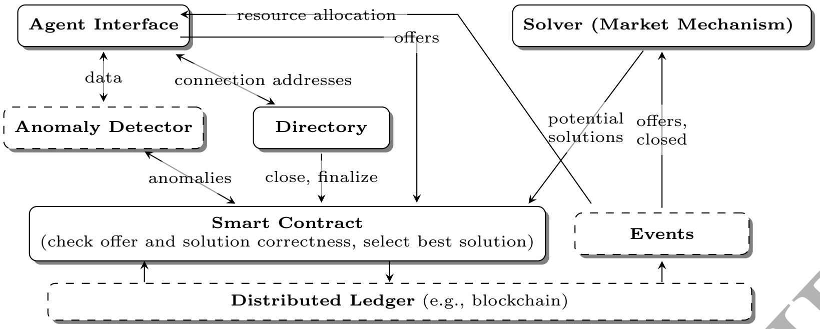


Figure 3: An example of a distributed market platform managing the interaction of the agents in a social computing setting described in [56].

trust can, then, be enabled in a way in which every user with access to the Internet can get involved in blockchain-based transactions, and third party trust organizations may no longer be necessary. Blockchain technology can be used in an endless number of applications: tax collection, money transfers without bank intervention, or health care management. How it work is explained in what follows.

When someone requests a transaction, such transaction is broadcast to a peer-to-peer network consisting of computation nodes, simply known as nodes, that form a completely decentralized network. The network of nodes validates the transaction and the user's status applying algorithms. When this transaction is verified, it is combined with other transactions to create a new block of data that is placed in the ledger. After, the new block is added to the existing blockchain permanently and inmutably.

Social dispersed applications are candidates for using blockchain technology given their highly distributed nature. Overall, the blockchain database is stored in a distributed way, and the records it keeps are public and easily verifiable. As no centralized version of such information exists, it is secured from hacker attacks.

4.3 Distributed Market Platform

As discussed in the earlier examples, there is a need for *incentives* to participate as a resource in the social dispersed computing as well as to be eager to provide information. A market based distributed framework can provide this foundation: one in which all interactions generated in the social computing application are safely stored. As mentioned previously, such interactions are found in other sharing economy driven applications [138], e.g., ride-sharing [82, 97], car-sharing [69] and transactive energy systems [95, 31, 88]. However, these exchange of data and resource raises the concerns of integrity, trust, and above all the need for fair and optimal solutions to the problem of resource allocation, motivating the requirement for a management platform.

Specifically, such a market based platform involves a number of self-interested agents that interact with each other by submitting *offers* to buy or sell the goods, while satisfying one or more of the following requirements: *i.)* anonymity of participant identities, i.e., individual agents shall not have the means to infer the identities of other agents, or who trades with whom; *ii.)* confidentiality of market information, which includes individual bids and transaction information, output of trade verification processes, and finalized trading data that are yet executed; *iii.)* market integrity and non-repudiation transactions; *iv.)* availability and auditability of all events and data which can take the form of encrypted or non-encrypted data.

Blockchains form a key component of such market based platforms because they enable participants to reach a consensus on the value of any state variable in the system, without relying on a trusted third party or trusting each other. Distributed consensus not only solves the trust issue, but also provides fault-tolerance since consensus is always reached on the correct state as long as the number of faulty nodes is below a threshold. Further, blockchains can also enable performing computation in a distributed and trustworthy manner in the form of smart contracts. However, while the distributed integrity of a blockchain ledger presents unique opportunities, it also introduces new assurance challenges that must be addressed before protocols and implementations can live up to their potential. For instance, smart contracts deployed in practice are riddled with bugs and security vulnerabilities. Another problem with blockchain based implementation is that the computation is relatively expensive on blockchain-based distributed platforms and solving the trading problem using a blockchain-based smart contract is not scalable in practice.

Figure 3 describes an example of such a market platform called SolidWorx [56]. It allows agents to post offers using predefined programming interfaces. A directory actor provides a mechanism to look up connection endpoints, including the address of a deployed smart contract. The smart contract functions check the correctness of each offer and then store it within the smart contract. Mixer services can be used to obfuscate the identity of the prosumers [32]. By generating new anonymous addresses at random periodically, prosumers can prevent other entities from linking the anonymous addresses to their actual identities [94, 32], thereby keeping their activities private. Solver actors, which are pre-configured with constraints and an objective function, can listen to smart-contract events, which provide the solvers with information about offers. Solvers run at a pre-configured intervals, compute a resource allocation, and submit the solution allocation to the smart contract. The directory, acting as a service director, can then finalize a solution by invoking a smart-contract function, which chooses the best solution from all the allocations that have been submitted. Once a solution has been finalized, the prosumers are notified using smart-contract events. To ensure correctness, the smart contract of SolidWorx is generated and verified using a finite-state machine (FSM) based language called FSolidM [109].

4.4 Time Synchronization

Satisfying time deterministic requirements during code execution on a node is crucial but not enough for a distributed system like social dispersed computing. In these applications, we sometimes need to establish a common synchronized time base and need to align each node's local clock(s) to this global reference. Even slight differences in each node's local clock—typically a few tens of parts per million (ppm)—accumulate fast and become apparent over time. Based on environmental factors (temperature, humidity, and voltage stability), the frequency differences are not constant. Thus, to provide an accurate globally synchronized time base, the supporting services need to periodically measure and compensate for these differences. The periodic adjustment of the local time on the node requires careful considerations to avoid disruption of the local event scheduler [54]. Fortunately, there are two well established technologies for solving this problem, both are supported by any modern Linux kernel.

The Network Time Protocol (NTP) [74] is a ubiquitous time synchronization service using heuristic software algorithms with no special requirements on the networking hardware and communication infrastructure. The Precision Time Protocol (PTP, IEEE-1588) on the other hand is built on accurate end-to-end hardware-level timestamping capabilities. It is no surprise that the attainable accuracy of the two methods differ by orders of magnitudes: tens of milliseconds with NTP vs. microseconds with PTP [120]. PTP has also been implemented over wireless [43].

The PTP protocol achieves excellent accuracy if used within a local area network and/or all network equipment in the packet forwarding path participate in the protocol. The basic building blocks of the protocol are: (1) a hierarchical master/slave clock tree strategy supported by a leader-election ('best master') protocol, (2) accurate time-of-flight measurement of network packets with the built-in assumption that these delays are symmetrical (3) support for measuring and compensating for intermediate delays across the communication medium (4) using level-2 LAN frames or IPv4/IPv6 UDP messages as the transport mechanism (5) support for co-existing independent PTP clock domains on the same LAN.

At its core, the master-slave clock synchronization mechanism is implemented by periodic beacon frames broadcast by the master and containing the master clock value at the beginning of the beacon message generation. If the networking hardware is not capable of inserting this time value during frame transmission, a second non time critical frame is sent by the master containing this value. With properly maintained estimates on frame transmission delays, each slave can adjust its local clock to the master. The delay estimation is based on periodic round-trip requests from the slaves to the master. The request message is transmit-timestamped by the slave and received-timestamped by the master. The server then replies with a non real time message which contains the received-timestamp for the slave to have a good estimate on the current network delay.

4.5 Distributed Coordination Services

Social dispersed computing applications will aggregate large numbers of users that will participate as sensing actors and will also receive and use data produced by the application. Interactions across these users will be possibly made on the basis of user groups that can change dynamically. Services for grouping/membership management and distributed coordination and consensus will have to be put in place to enable consistent inter-operation with coherent state management.

An application may be deployed on a variable number of nodes. Nodes can be added or removed from the network at any time, either by a controlling authority or unintentionally due to a fault condition. It is possible for an application to operate on a subset of nodes (or groups), while another application operates on another subset of nodes. It is possible for a node to migrate from one subset to another subset.

A distributed coordination service provides common services for coordination among actors that run on a network of nodes. The distributed coordination service includes (1) group membership, (2) leader election, (3) distributed consensus, and (4) time-synchronized coordinated action; these are explained below:

- **Group membership maintenance:** It is a basic building block that maintains the logical lists of components (i.e., users) that register with the service. All the distributed coordination features are available inside a logical group.

- **Leader election:** Choosing a leader is a process where a single node becomes designated as an organizer of tasks among several distributed nodes.

- **Distributed consensus:** A process where group members form agreement on some data value.

- **Time-synchronized coordinated action:** Time synchronized activities take the clock value as the trigger for their execution. In a distributed scenario, several nodes will have to agree on when to schedule a task of this kind, and for this, their clocks must be synchronized.

More in detail, coordination services are needed to maintain shared state in a consistent and fault-tolerant manner. Achieving fault tolerance is done by using replication that is typically based on running a quorum (majority) based protocol such as **Paxos** [92, 91]. Paxos manages the state updates history with acceptors, and each update is voted by a quorum of acceptors. The leader that manages the voting process is one acceptor. Paxos also has *learners* that are light weight services that get the notifications of updates after the quorum accepts them; learners do not participate in the voting. Different technologies have implemented this protocol; a few selected ones will be presented in the next section. A major criticism to Paxos is that is not an easy to understand protocol. Raft [125] is similar to Paxos, however, according to the authors it is more understandable, the implementation phase is shorter, and it is designed to have fewer states.

Often, distributed hash tables are also used to store the information that can be used for distributed coordination. For example [57] uses OpenDHT [131] to store, query, and disseminate details of publishers and subscribers across the network. OpenDHT is a fast, lightweight Distributed Hash Table (DHT) implementation. The dissemination does not mean full data replication on all nodes. OpenDHT stores the registered value locally and forwards it to a maximum of eight neighbours. The distributed hash table for service discovery does not distinguish the nodes, (i.e. there are no “server” or “client” nodes) – nodes are peers and each operates with the same rules. If a node disconnects from the network, the DHT service on the other nodes is still able to register new services or run queries.

4.6 Software technologies

4.6.1 Virtualization

Figure 1 provides a general view over computing and its evolution (q.v. figure 1) to the current virtualization technology.

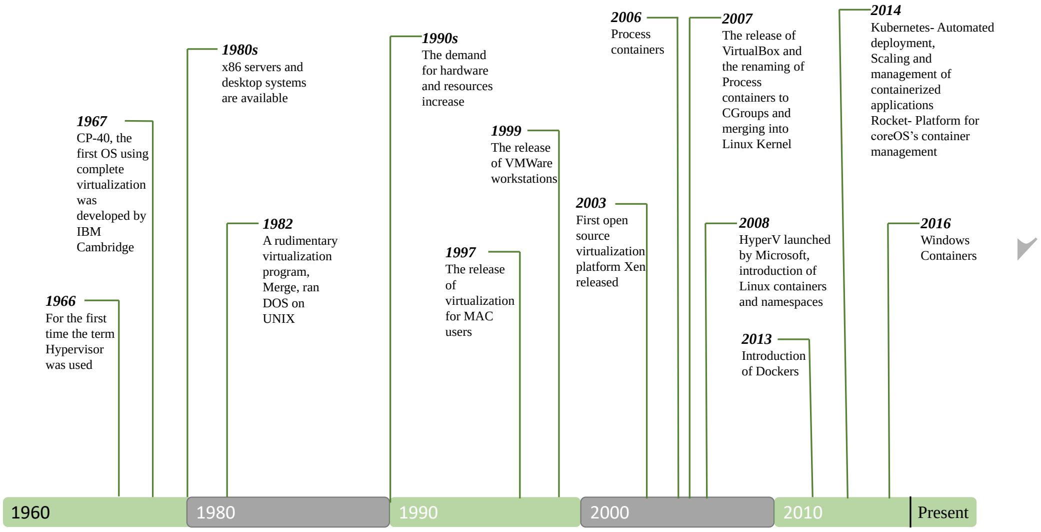

1966
For the first time the term Hypervisor was used
1967
CP-40, the first OS using complete virtualization was developed by IBM Cambridge
1980s
x86 servers and desktop systems are available
1982
A rudimentary virtualization program, Merge, ran DOS on UNIX
1990s
The demand for hardware and resources increase
1997
The release of virtualization for MAC users
1999
The release of VMWare workstations
2003
First open source virtualization platform Xen released
2006
Process containers
2007
The release of VirtualBox and the renaming of Process containers to CGroups and merging into Linux Kernel
2008
HyperV launched by Microsoft, introduction of Linux containers and namespaces
2013
Introduction of Dockers
2014
Kubernetes- Automated deployment, Scaling and management of containerized applications
Rocket- Platform for coreOS's container management
2016
Windows Containers
1960
1980
1990
2000
2010
Present

Figure 4: Looking back at 60 years of virtualization history.

Virtualization technology has been one of the key enablers of cloud computing [61], and will also play a major role in social dispersed computing. The partial computations from user groups will have to happen in servers in their vicinity that will aggregate the data received from users, possibly maintaining a state of the group, and communicate back to the users and to other neighboring servers and the cloud. These servers will have to run other applications besides the social computing application; in this way, virtualization can be used to isolate the execution of the different applications in the same physical node, avoiding interference and preserving performance. In a computer system, *virtualization* refers to the creation of a virtual (not actual) version of some other system; that includes processor, storage, or network virtualization. There are different types of virtualization. A few of them are provided in what follows.

Machine virtualization. It provides an abstraction of the real hardware resources or subsystems, mapping the virtual resource to the actual one, offering applications an abstract view through interfaces of the hardware platform and resources that are provided underneath. In this context, the *host machine* is used for referring to the physical machine on which virtualization occurs; and *guest machine* is the virtual machine that is created on the physical machine. The *hypervisor* or *virtual machine monitor* (VMM) is a program (whether software, firmware, or hardware) that creates virtual machines on an actual host machine.

Virtualization allows applications to be run in software environments that are separated from their underlying hardware infrastructure by a layer of abstraction. This enables different applications to be split into virtualized machines that can run over different operating systems running over the same hardware.

A virtual machine (VM) is an execution environment in its own: it is a software implementation of a physical execution platform, machine, or computer, capable of running the same programs that the physical machine can run. Virtual environments can be designed from either a hardware partitioning or hypervisor design side. Hardware partitioning does not support the benefits that resource sharing and emulation offered by hypervisors can provide.

There are two main types of hypervisors. On the one hand, *bare metal* (namely type 1) hypervisors execute directly on the physical hardware platform that virtualizes the critical hardware devices offering several independent isolated partitions. Examples of these are VMWare ESX, Xen, or Microsoft Hyper-V; and others such as WindRiver Hypervisor or XtratuM for real-time systems. These can also include network virtualization models like VMware NSX. On the other hand, *type 2* hypervisors are hosted ones as they run over a host operating system.

Containers. Containers are a different virtualization model in which different applications and services can run on a single operating system as a host, instead of virtual machines which allow to run different operating systems. The idea behind containers was providing software code in a way that can quickly be moved around to run on servers using Linux OS; such software form can even be connected together to run a distributed application in the cloud. The benefit is, then, maximized by the possibility of speeding up the building of large cloud applications that are scalable.

Containerization was originally developed as a way to separate namespaces in Linux for security reasons for protecting the kernel from the execution of applications that could have questionable security or authenticity. After this came the idea of making these “partitions” efficient and portable. LXC [10] was probably the first true container system, and it was developed as part of Linux. Additionally, Docker [4] was then developed as a system capable of deploying LXC containers on a PaaS platform.

The applications running with containers are virtualized. In the specific case of Docker’s native environment, there is no hypervisor. There is a daemon in the kernel that provides the isolation across containers and connects the existing workloads to the kernel. Modern containers usually include a minimal operating system (e.g. VMWare’s Photon OS) with the sole objective of providing basic local services for the hosted applications.

Microservices. The concept of microservices has a natural fit to containers, and it provides an alternative to the monolithic architecture pattern that is the traditional architectural style of enterprise applications. The microservice architecture structures applications as collections of loosely coupled, small, modular services that provide business capabilities and in which every service runs a unique process and communicates through well-defined, lightweight mechanism.

Microservices are functions that can operate for different applications like libraries, that contact them via an API to produce a discrete output. In monolithic applications, these functions would be instantiated redundantly: one per application. Netflix [121] streaming video service provider uses microservices. Modern containers include only the basic services needed for a given system. Orchestration services such as Kubernetes and Mesosphere Marathon manage the replication and removal of container images depending on the traffic patters to/from the workloads of microservices.

Different protocols are possible for communication across microservices like HTTP; however, DevOps professionals mostly choose REST (Representational State Transfer) given its lower complexity as compared to other protocols. Microservices support the continuous delivery/deployment of large, complex applications, that yields agile software provisioning. Given its scalability, it is considered a particularly interesting pattern when it is needed to support a broad range of platforms and devices.

4.6.2 Cloud deployment and management

There are various alternatives to designing and developing a cloud computing infrastructure and manage it such as Amazon Elastic Compute Cloud (Amazon EC2) [21], Microsoft Azure [113], CloudStack [22], OpenStack [12], OpenNebula [11], Eucalytus [6], or IBM Cloud [9], among others. They offer compute and storage services on the basis of an IaaS model, except for Google App Engine [8] and Azure; the latter offer a PaaS model on which it is possible to deploy web applications and scalable mobile backends.

The technologies that provide IaaS model are typically based on lower level virtual machine monitors (VMMs) that allow the construction of virtual execution environments or virtual machines. Most of the previous technologies are based on either Xen [15], VMware [14], or KVM [99] VMMs and have a native Linux host. This is true except for IBM Cloud that also uses the above virtualization.

On the other hand, the technologies that provide PaaS are based on lighter weight virtualization models such as application containers in the case of Google App Engine or OS virtualization for Microsoft Azure. Among the main benefits of this model is the maintenance cost as users do not have to configure nor fine tune any backend server. User applications deployed in this type of environments can use APIs to access a number of available services just as data base interfacing (through SQL queries, etc.) or user authentication. In addition, applications availability is also managed by the platform, and they are automatically scaled depending of the amount of incoming traffic so users only pay for the amount of resources used.

A number of problems have been addressed over the last decade for data center management. Precisely, virtual machine placement has been one of the most popular problems addressed by the scientific community that has produced many contributions such as [108]. Energy consumption has also received great attention; some researchers have contributed algorithms to optimize virtual machine placement and optimize energy consumption such as [44] through live migration based on values of usage thresholds considering multiple resources, therefore targeting two of the greatest problems of data centers.

Another research problem in cloud is quality of service aware data delivery to users. One of the bottlenecks in a data center that hinders performance is the networking across servers with kilometers of cables and terabytes of exchanged data across inhouse servers. Quality of service provisioning is concerned also with a number of very common activities such as effective resource management strategies [28] including virtual machine migration, service scaling, service migration, or on-the-fly hardware configuration changes. These may all affect the quality experience by data delivery to users.

One of todays' open problems in cloud computing management is managing the complexity introduced by geographically distributed data centers. Some authors have proposed the design of an integrated control plane [41] that integrates both computation resources and network protocols for managing the distribution of data centers. Timely traffic delivery is essential to guaranteeing quality of service to applications, services, and users. Traffic engineering relies on the appropriate networking mechanism over LSP (Label Switched Paths) that are set at core networks and are controlled by the control plane. Path computation is essential to achieving the goals of traffic engineering. Actually, the IETF (Internet Engineering Task Force) promoted the Path Computation Element (PCE) architecture as a means to overcome the inefficiencies encountered by the lack of visibility of some distributed network resources. The core idea of PCE is a dedicated network entity destined to path computation process. A number of initiatives for using the Path Computation Element (PCE) also for cloud provisioning has been further researched [127].

Predictable cloud computing technologies. The penetration of virtualization technology has paved the way for the integration of different functions over the same physical platform. This effect of virtualization technology has also arrived to the real-time systems area supporting the integration of a number of functions of heterogeneous criticality levels over the same physical platform. The design of mixed criticality systems (MCS) [40] is an important trend that supports the execution of various applications and functions of different criticality levels.

Real-time domains have improved the capacities of hypervisors to ensure full isolation across virtual machines that are called *partitions*. Partitions are fully independent and are scheduled by the hypervisor according to some scheduling policy. To comply with the real-time requirements, usually hierarchical scheduling is used due to its simplicity that favors timeliness; however, still the most complex point in this domain is the integration of the communication and distribution technology into partitioned systems. In [62], it is shown how a distributed partitioned system can be naturally integrated with a hierarchical scheduling mechanism to ensure timeliness of the communications when using distribution software under a number of restrictions.

4.6.3 Messaging Middleware

A large number of interacting entities in IoT are likely to be resource-constrained devices for whom some of the existing middleware solutions are not suitable. To overcome these restrictions, a variety of solutions have recently been developed and new ones are emerging. We survey a few of these solutions.

Message Queuing Telemetry Transport (MQTT) MQTT [73] is a connectivity protocol to support machine-to-machine (M2M) communications in IoT. Since the goal was to support the IoT resource-constrained devices, it is designed to be very lightweight. MQTT supports a publish/subscribe messaging transport. Example use cases include sensors communicating to a broker via satellite link, over occasional dial-up connections with health care providers, and in a range of home automation and small device scenarios. Even mobile applications can make use of MQTT because of its support for small size, low power usage, smaller data packet payloads, and efficient distribution of information to one or many receivers.

MQTT supports a publish/subscribe communication model and uses the term “client” to refer to entities that either publish topics or subscribe to topics, while the term “server” refers to mediators/brokers that relay messages between the clients. MQTT operates over TCP or any other transport protocol that supports ordered, lossless message communication. MQTT supports three levels of QoS for message deliver: (a) at-most-once, (b) at-least-once, and (c) exactly-once.

MQTT was originally developed in 1999 and has recently become an OASIS standard starting from version 3.1.1.

Message Brokers MQTT is in fact an example of a publish subscribe message broker. In addition to MQTT, a number of message brokers like Apache Kafka, AMQP (Advanced Message Queue Protocol), and Active MQ are finding applications in areas of IoT. Apache Kafka [23] is an open source distributed streaming platform used to build real time data pipelines between different systems or applications. They provide high throughput, low latency and fault tolerant pipeline for streaming data with a tradeoff between performance and reliability. They are deployed as a cluster of servers which handles the messaging system with the help of four core API's, namely, producers, consumers, streams, and connectors. The other important part of the Kafka architecture is the topic, broker and records. Here, data is divided into topics, which is further divided into partitions for the brokers to handle them. Apache Zookeeper is used to provide synchronization between multiple brokers. Among the most popular data buses is the data centric DDS (Data Distribution Service) [75] which has been extended in a number of ways such as [63] for supporting real-time reconfiguration.

Constrained Application Protocol (CoAP) The CoAP protocol [38], which is defined as an Internet Standard in RFC 7252, is a web transfer protocol for use by resource-constrained devices of IoT, e.g., 8-bit microcontrollers with small ROM and RAM. Like MQTT, CoAP is also meant to support M2M communications. CoAP provides a request/response interaction model in contrast to the publish/subscribe model between application endpoints. It supports built-in discovery of services and resources.

CoAP supports key concepts of the web such as URIs and Internet media types. It leverages the REST architectural pattern that has been highly successful in the traditional HTTP realm. Thus, in CoAP, servers make their resources available as URLs and clients can use commands such as GET, PUT, etc to avail of these resources. Due to the use of the REST architectural pattern, it is seamless to combine HTTP with CoAP thereby allowing traditional web clients to access an IoT sensor device.

CoAP uses UDP as its transport layer. Other protocols like DTLS are also applicable. Like HTTP, CoAP allows payloads of multiple different types, e.g., XML, JavaScript Object Notation (JSON), Concise Binary Object Representation (CBOR).

Node-RED. Node-RED [35] is technically not a middleware but rather a browser-based model-driven tool to wire the flows between IoT devices. The tool then allows a one-click approach to deploy the capabilities in the runtime environment. Node-RED uses Node.js behind the scenes. The flows are stored as JSON objects. Thus, we can consider Node-RED as a model-driven middleware capability.

Akka. [1] Akka is an open-source event-driven middleware framework that uses Actor Model [70] to provide a better platform to build scalable, resilient, and responsive distributed and concurrent applications. Akka runs on a Java virtual machine (JVM) and supports actors written in Java and Scala. Actors in Akka are very lightweight event-driven processes that provide abstractions for concurrency and parallelism. Akka follows "let it crash" model for fault-tolerance in order to support applications that self-heal and never stop.

Distributed applications in Akka will constitute of multiple actors distributed amongst a cluster of member nodes. Cluster membership is maintained using *Gossip Protocol*, where the current state of a cluster is randomly propagated through the cluster with preference to members who have not seen the latest state. Actors within a cluster can communicate with each other using *mediators* that facilitate point-to-point as well as pub/sub interaction patterns. Each node can host single mediator in which case discovery becomes decentralized, or we can designate particular nodes of a cluster to host mediator in which case discovery becomes centralized. Akka's message delivery semantics facilitates three different QoS policies - (a) at-most-once, (b) at-least-once, and (c) exactly-once.

Robot Operating System (ROS). ROS [13] is a framework that provides a collection of tools, libraries, and conventions to write robust, general-purpose robot software. It is designed to work with various robotic platforms. ROS nodes are processes that perform computation and these nodes combined together form a network (graph) of nodes that communicate with each other using pub/sub interaction pattern or request/response interaction pattern.

Pub/sub interaction is facilitated via *topics*. Multiple publishers and subscribers can be associated with a topic. Request/response interaction, on the other hand, is done via a *service*. A node that provides a service, offers its service under a string *name*, and a client calls a provided service by sending the request message and awaiting the reply. Both, topics and services, are monitored by the ROS *Master*. Therefore, the Master is a single point of failure that performs the task of matching nodes that needs to communicate with each other, regardless of the interaction pattern.

4.6.4 Complex Event Processing (CEP)

CEP is used in multiple points for IoT analytics (e.g. Edge, Cloud etc). In general, event processing is a method for tracking and analyzing streams of data and deriving a conclusion from them, while the data is in motion. A number of CEP engines like Siddhi, Apache flink and Esper are available for stream processing. These CEP tools allow the users to write queries over the arriving stream of data which can be utilized to determine anomalies, sequences, and patterns of interest. For example, Siddhi [146] is an open source CEP server with a very powerful SQL query like language for event stream processing. It allows the users to integrate the data from any input system like Kafka, MQTT, file, and websocket with data in different formats like XML, JSON or Plain text. After the data has been received at the input adapters, queries like patterns, filters, sequences, windows and pass through can be applied on the data at the even stream to perform some real time event processing. The data obtained after processing can be published over web based analytics dashboard to monitor the meaningful processed data.

4.6.5 Transaction management

Hyperledger[4] is a Linux implementation for blockchain. Hyperledger (or the Hyperledger project) is an umbrella project of open source blockchains and related tools [55] that started in December 2015 by the Linux Foundation [16]. Hyperledger's goal is to develop blockchain-based distributed ledgers following the Linux philosophy of collaborative development.

During 2016, several open-source platforms for the financial services industry have appeared, e.g. Hyperledger, Chain Core, or Corda, besides other open-source platforms such as Ethereum and Monax that were released in precedent years.

The Hyperledger project is partitipated by a large number of partners contributing different tools individually or in collaboration. Burrow[5] is a blockchain client that includes a virtual machine (Ethereum). Fabric[6], is an architecture that defines the execution of smart contracts (namely chaincode in Fabric); the processes for consensus and membership, and the roles of the participating nodes. Iroha[7] is another Hyperledger tool similar to Fabric but targeted at mobile aplications. Lastly, Sawtooth[8] is a tool that provides the

[4] http://www.hyperledger.org

[5] Burrow was contributed by Monax

[6] Fabric was originally contributed by IBM and Digital Asset

[7] Contributed by Soramitsu

[8] Contributed by Intel

Proof of Elapsed Time consensus protocol based on a lottery-design consensus protocol; this tool is based on trusted execution environments such as SGX[9].

4.6.6 Service configuration and deployment technologies

Kubernetes is an open source platform that facilitates the task of running applications in clouds, whether private or public. It supports the automatic deployment and operation of application containers. Applications can be scaled on the fly, and the usage of hardware can be limited to required resources only. Whenever an application need be released, Kubernetes allows generating container images; it can schedule and run application containers on clusters of physical or virtual machines. One of the most interesting characteristics is that it supports continuous development, integration, and deployment with quick rollbacks. Also, it raises the level of abstraction as compared to running an operating system on a virtualized hardware; now, it is an application that is run on an operating system that uses logical resources. In Kubernetes, applications are composed of smaller microservices that are independent pieces of code that can be deployed and managed dynamically.

Paradrop [101] is a platform that offers computing and storage resources over the end nodes supporting the development of services. A key element is the WiFi access point as it has all information about its end devices and manages all the traffic across them. Paradrop provides an API for third party developers to create and manage their services across different access points, that are isolated in containers (called chutes). Also, it provides a cloud backend to install dynamically the access points and the third party containers, and to instantiate and revoke them. Paradrop uses lightweight Linux containers [100] instead of virtual machines as the virtualization mechanism to deploy services on the network routers.

4.6.7 Distributed service coordination

Zookeeper[24] is an open source technology that provides key services for large scale systems containing large numbers of distributed processes; these services are configuration, synchronization, group services, and naming registry. Typically, these services can be highly complex to design and implement and they are used by the vast majority of distributed applications.

Zookeeper has a simple architecture in the form of a shared hierarchical namespace to facilitate process coordination. Also, it is a reliable system that

[9] Software Guard Extensions by Intel

can continue to run in the presence of a node failure; it provides redundant services for ensuring high availability.

Data storage is performed in a hierarchical name space such as a file system or a tree data structure. It supports data updates in a totally ordered manner as in an atomic broadcast system.

Fault tolerance and security is an important characteristic in coordination services that must be well supported by not considering only simple faults (crashes) or attacks (invalid access) that they should cover fault tolerance and security. **DeepSpace** [34] is a distributed coordination service that provides Byzantine fault tolerance [93] in a tuple space abstraction. It provides secure, reliable, and available operation in the presence of less than a third of faulty service replicas. Also, it has a content-addressable confidentiality scheme that allows to store critical data. The maturity level, community, services, and penetration of Zookeeper is, however, not comparable.

Girafe [140] is a scalable coordination service for cloud services. It organizes the coordination of servers by means of interior-node-disjoint trees; it uses a Paxos protocol for strong consistency and fault-tolerance; and it supports hierarchical data organization for high throughput and low latency.

ZooNet [96] is a coordination service idea that addresses the problems of coordination of applications running in multiple geographic regions; these applications need to trade-off between performance and consistency, and ZooNet provides a modular composition design for this purpose.

Consul [2] is a system to enable service discovery and configuration in a distributed infrastructure. Consul clients provide services (e.g. MySQL) and other clients can discover the providers of such given service. Health checks for given services are also enabled with respect to specific characteristics such as if a service is up and running or if it is using a certain memory size. Health checks can be used to route traffic avoiding unhealthy hosts. It also provides multi-region datacenters.

Consul is based on *agents*. Each node that is part of Consul (i.e., that provides services to it) runs a Consul agent that is responsible for health checking the services on the node as well as the node itself. Agents interact with Consul *servers* that store data and replicate it. Servers elect a *leader*. Components that need to locate a service query any of the servers or any of the agents; agents automatically forward requests to the servers. Location of services residing in remote data centers is performed by the local servers that forward the queries to the remote data center.

etcd [115] is a key value store, which internally uses raft [126] consensus algorithm. Etcd can be used to build a discovery service. However, it is primarily used to store information across a set of nodes. Kubernetes uses etcd for managing the configuration data across the cluster.

4.6.8 Fine grain resource management

Mesos [71] is a thin software acting as a resource manager that enables fine-grained sharing across different and highly diverse cluster computing frameworks by providing them with a common interface to access the cluster resources. Control of task scheduling and execution is taken by the frameworks; this allows each framework to decide on execution of activities according to its specific needs and better supports the independent evolution of frameworks.

Mesos consists of a *master* and *slave* daemons, *frameworks*, and *tasks*. The master process manages the slave daemons running on each cluster node. Moreover, frameworks run tasks on these slave daemons. Each framework running on Mesos has two components: a *scheduler* and a *executor*. The scheduler registers with the master in order to be offered resources; the executor process is launched on the slave daemons to run the tasks.

Fine-grained resource sharing across the frameworks is implemented using resource offers, that are lists of free resources on multiple slaves. The organizational policies (priority or fair sharing) determine how the master decides on how many resources to offer to each framework. Mesos defines a plugable allocation module to let organizations define their own allocation policies.

An important characteristic is that Mesos provides performance isolation between framework executors running on the same slave by leveraging existing isolation mechanisms of operating systems.

4.6.9 Edge computing and networking technologies

Software defined networks (SDN). Social dispersed computing applications require flexible network connections to support the dynamic geographic distribution of end users. Although the advances in network technology and bandwidth increase have been impressive, still IP networks have until recently been structured in an manner that did not achieve sufficient flexibility.

Actually, the boost of Internet has occurred over IP networks that are vertically integrated [137] in which control and data planes are bundled together [60] inside the network devices. However, this design makes it hard to reconfigure in the event of adverse load conditions, faults, etc. The control plane is the logic that decides how to handle the network packets; whereas the data plane is the logic in charge of forwarding the packets as indicated by the control plane. Network operators configure each network device individually using low-level (and sometimes vendor specific) logic; all data packets are treated the same by the switch that starts sending every packet going to the same destination along the same path. Originally, SDN focused exclusively on the separation of the control and data planes.

Software defined networking brings in the promise for solving the above limitations in a flexible way by providing the needed mechanisms for a network that will be programmable.

[86] provides a comprehensive survey of the technologies towards SDN and its adoption. It presents the main differences of the conventional networking as compared to SDN, describing the role of the SDN controller over which a number of network applications (like MAC learning, routing algorithms, intrusion detection system, and load balancer) run.

The above is the classic SDN scenario in which a controller (that is an application running somewhere on some server) sends the switch the rules for handling the packets; then, switches that are the data plane devices request guidance to the controller whenever needed and provide the controller with information about the traffic that they handle. The communication between the controllers and the switches happens through well defined interfaces. The interface that enables communication between the SDN controller and the network nodes (that are physical and virtual switches and routers) is called the controller's southbound interface, and it is typically the OpenFlow [110] specification. OpenFlow has become the most important architecture for managing large scale complex networks and has, therefore, become the major bet for SDN. This is a specification that need be applied in matured systems through implementations. [72] provides a survey of the target applications, the language abstraction, the controller functions and inner workings, the virtualization that is achieved, quality of service properties, security issues and its integration in different networks. OpenFlow security issues are very relevant especially in large scale deployments. [79] describes the two types of denial of service (DoS) attacks that are specific to OpenFlow SDN networks discovering some key configurations (like the timeout value of a flow rule and the control plane bandwidth) that directly affect the capability of a switch and it identifies mitigation actions for them.

The research in SDN proceeds in parallel with the improvement of the control plane algorithms searching for better and more efficient ways to route traffic. Especially cloud services with soft real-time requirements experience the delays of wide area IP network interconnects across geographically distributed locations. To address this problem, [33] proposes a routing mechanism for providing latency and reliability assurances for control traffic in wide-area IP networks with a just in time routing that routes deadline constrained messages that are control messages at the application layer with the goal of achieving a non-intrusive solution for achieving timely and reliable communication.

5 Challenges in Social Dispersed Computing

Having explained the different computation technologies that cover the range from utility cloud computing to edge computing, we can now revisit the concept of social dispersed computing and identify the key challenges that still exist. For researchers these points also serve as a summary of current research interests and opportunities for the community.

The primary challenge of social dispersed computing is mobility. Consider that nodes in the social routing application described earlier are mobile, the system must be cognizant of intermittent connectivity caused due to high mobility. Thus, new mechanisms have to be built for implementing handover mechanisms that account for multi-tenancy on a local cloud in which multiple service providers can be present to ensure backup. Additionally, given the high mobility of users, managing volatile group formation may play a key role in the efficient collection of data and in the transmission of only the needed data that is relevant for particular groups. For this, it will be needed to incorporate dynamic transaction management functionality.

The second challenge emanates from the resource constraints of the system, which suggests that only required applications should be running on the computation platforms. However, this leads to an interesting question of what are the required applications. In the past "goal-driven" computing has been used in high criticality, but mostly static systems [130]. However, a social dispersed application implies that the end nodes or user nodes act in a social way; they will exchange information, sharing part of their computations among the participant users, the local fog nodes, and partly with the cloud nodes. A number of different services may run at these three layers: user/end, fog, and cloud. Also, some services may be split across the different layers. As all participant nodes are driven by a shared goal, they will have to share part of their data and computations in a synchronized way and the exchanged data will have to be appropriately tagged in the temporal domain to meet the global goal. Thus goal-driven service orchestration is a challenge in these systems.

Another challenge includes service synchronization and orchestration. In cloud model, services are provided to clients in a client-server interaction type. In social dispersed computing, end nodes come into play, requiring interaction not only with the cloud servers. End nodes will interact with other end nodes for fast information exchange; with the fog nodes for data bulk exchange and for low latency gathering of information derived from heavy processing; and with the cloud servers for obtaining results derived from more complex data intensive computations like machine learning technology for longer term prediction. Social dispersed computing applications will need that supporting architectures add an abstraction layer that meets the coordination and orchestration requirements by providing smooth cooperation through the end nodes. This layer will contain the required logic to orchestrate the interaction between fog servers and the central cloud, as well as the interaction across fogs.

Timely operation and stringent quality of service demands is yet another challenge. Some social dispersed applications need to provide real-time services to users. This requires to put in place a number of physical resource management policies that ensure time bounded operation. Fog servers will have high consolidation, so virtualization techniques will have to be properly applied in conjunction with scheduling policies that ensure timely operation for those real-time services and avoidance of execution interference among applications in the presence of possibly computationally greedy functions.

Understanding that failures are going to be more common in social dispersed computing applications is important. Thus, we must manage the soft state of applications. End nodes may interact heavily in social dispersed applications. Interactions across end nodes may not assume that data nor the infrastructure are available at all times. There is a noticeable difference with respect to the cloud model that handles hard state and persistent data. Considering soft state brings in much more complex scenarios in which fall back operations will need to be considered for the user execution of recovery actions.

The focus of social dispersed computing shifts towards the *service* and the *data*, and other characteristics such as the location become less important. A service may reside on a number of fog servers as well as partly in the cloud. Then, the traditional client-server structure falls short as IP based operations become inappropriate for handling service and data centric interactions across nodes (mainly the fog and end nodes). A service centric design that relies on data centric interaction and information exchange better adjusts to this level of complexity.

Therefore, service offloading strategies and target infrastructure processing point selection is going to be a difficult problem. In a social dispersed application, it will be beneficial to draw a clever server processing hierarchy. Where to process, whether at the edge or at the fog, and why are decisions that will have to be taken based on a per application basis. We believe that the target point for running a specific service should be selected according to the computational complexity of the service itself (e.g. online video streaming probably at the edge servers, face recognition probably at the fog). There is strong need for designing efficient service partitioning schemes that make use of the end, fog, edge, and cloud infrastructures as a complementary overall execution platform that will speed up the dispersed computations for the social interactions.

Lastly, autonomy, interaction, mobility and openness are the characteristics that the Multi Agent System (MAS) paradigm covers from a theoretical and practical perspective. MAS technology provides models, frameworks, methods and algorithms for constructing large-scale open distributed computer systems and allows to cope with the (high) dynamicity of the systems topology and with semantic mismatches in the interaction, both natural consequences of the distributed and autonomous nature of the components. Open distributed systems are going to be the norm in the software development industry of the future, and the interoperation of the software entities will need to rely on a declarative concept of agreement that is autonomously signed and executed by the entities themselves. The generation of agreements between entities will need to integrate semantic, normative, organization, negotiation and trust techniques (namely agreement technologies).

As evidenced by the partial list of technical problems given above, there is a complex technical challenge in the design and development of social dispersed computing applications that is multi-faceted. Addressing some of these problems simultaneously may result in the appearance of emerging problems that have still not been envisioned.

6 Discussion and Conclusions

A number of computing paradigms have appeared through the years that are, currently, applied simultaneously to develop a number of systems across different application domains. Newer computing paradigms coexist with other more classical ones and each has brought into scene their accompanying set of tools and technologies in their support. This large number of computation design options allows us to implement systems of a complexity that was not previously imagined and that increases day by day. Some of the more recent paradigms have still not been sufficiently applied in practice and even create confusion as to what they mean to different scientific communities. This paper situates the computing paradigms from the times where they were used as a utility to provide practical solutions to given problems that could be solved in a faster and more efficient manner; up to today where users are progressively accustomed to having commodity solutions at reach which go some steps beyond the mere practicality of automating tasks to a point in which they even consent to share information and knowledge online to ease their lives in some way or to obtain some other non primary benefit in exchange.

In this paper, we presented a review of core computing paradigms that have appeared in the distributed system community in the last two decades, focusing specifically on cloud computing, edge computing, and fog computing. Then, we described a set of computing technologies or services, which when augmented with the computing paradigms can enable interesting *social dispersed computing* applications. We described three example applications, two from the transportation domain and one from the energy domain. These applications can run successfully on both edge and fog computing devices. However, as we imagine more complex and integrated applications, we must start considering the challenges we mentioned in §5. Current computing technologies only partially meet these challenges, giving the community a great opportunity to explore this broad research space.

References

[1] akka. akka.io.

[2] Consul. https://www.consul.io. Last accessed February 2018.

[3] Distributed Cloud Storage. https://storj.io. Last accessed February 2018.

[4] Docker. https://www.docker.com.

[5] Ethereum block chain app platform. https://www.ethereum.org/. Last accessed February 2018.

[6] Eucalyptus cloud computing platform. https://github.com/eucalyptus/eucalyptus. Accessed Jan 2018.

[7] Foghorn. https://www.foghorn.io. Last accessed February 2018.

[8] Google cloud platform - app engine. https://cloud.google.com/appengine/. Accessed Jan 2018.

[9] Ibm cloud. https://www.ibm.com/cloud/. Accessed Jan 2018.

[10] LinuX Containers. https://linuxcontainers.org.

[11] Opennebula. http://opennebula.org. Accessed Jan 2018.

[12] Openstack. https://www.openstack.org/. Accessed Jan 2018.

[13] ROS. www.ros.org.

[14] Vmware. http://www.vmware.com. Accessed Jan 2018.

[15] The xen project 4.8.1 version april 2017. http://www.xenproject.org. Accessed Jan 2018.

[16] The linux foundation. Linux Foundation unites industry leaders to advance Blockchain technology, 2015. Last accessed: Dec. 2017.

[17] Brooklyn microgrid, 2017.

[18] DARPA Defense Advanced Research Projects Agency. Dispersed Computing, 2017. Accessed on October 2017.

[19] Ranwa Al Mallah, Alejandro Quintero, and Bilal Farooq. Distributed classification of urban congestion using vanet. *IEEE Transactions on Intelligent Transportation Systems*, 2017.

[20] A. Alzahrani, A. Alalwan, and M. Sarrab. Mobile cloud computing: Advantage, disadvantage and open challenge. In *in Proceedings of the 7th Euro American on Telematics and Information Systems, (EATIS 2014)*, pages 1–4, Valparaiso, Chile, 2014.

[21] Amazon. Amazon elastic compute cloud (amazon ec2). https://aws.amazon.com/es/ec2/. Accessed Jan 2018.

[22] Apache. Apache cloudstack. https://cloudstack.apache.org/. Accessed Jan 2018.

[23] Apache. Apache Kafka v0.8.2.0, February 2015. Last accessed Dec. 2017.

[24] Apache. Apache Zookeeper v3.5.3, April 2017. Last accessed Dec. 2017.

[25] Alexander Artikis and Jeremy Pitt. A formal model of open agent societies. In *Proceedings of the Fifth International Conference on Autonomous Agents*, AGENTS '01, pages 192–193, New York, NY, USA, 2001. ACM.

[26] Asaph Azaria, Ariel Ekblaw, Thiago Vieira, and Andrew Lippman. Medrec: Using blockchain for medical data access and permission management. In *Open and Big Data (OBD), International Conference on*, pages 25–30. IEEE, 2016.

[27] M. Beck, M. Werner, S. Feld, and S. Schimper. Mobile edge computing: A taxonomy. In *Proc. of the Sixth International Conference on Advances in Future Internet*, 2014.

[28] Anton Beloglazov and Rajkumar Buyya. Managing overloaded hosts for dynamic consolidation of virtual machines in cloud data centers under quality of service constraints. *IEEE Trans. Parallel Distrib. Syst.*, 24(7):1366–1379, 2013.

[29] Juan Benet. IPFS - content addressed, versioned, p2p file system (draft 3). https://ipfs.io. Last accessed Feb 2018.

[30] K. Benson, C. Fracchia, G. Wang, Q. Zhu, S. Almomen, J. Cohn, L. D'arcy, D. Hoffman, M. Makai, J. Stamatakis, and N. Venkatasubramanian. Scale: Safe community awareness and alerting leveraging the internet of things. *IEEE Communications Magazine*, 53(12):27–34, Dec 2015.

[31] Jonatan Bergquist, Aron Laszka, Monika Sturm, and Abhishek Dubey. On the design of communication and transaction anonymity in blockchain-based transactive microgrids. *CoRR*, abs/1709.09601, 2017.

[32] Jonatan Bergquist, Aron Laszka, Monika Sturm, and Abhishek Dubey. On the design of communication and transaction anonymity in blockchain-based transactive microgrids. In *Proceedings of the 1st Workshop on Scalable and Resilient Infrastructures for Distributed Ledgers (SERIAL)*, pages 3:1–3:6. ACM, 2017.

[33] Alysson Bessani, Nuno F. Neves, Paulo Veríssimo, Wagner Dantas, Alexandre Fonseca, Rui Silva, Pedro Luz, and Miguel Correia. Jiter: Just-in-time application-layer routing. *Computer Networks*, 104:122 – 136, 2016.

[34] Alysson Neves Bessani, Eduardo Pelison Alchieri, Miguel Correia, and Joni Silva Fraga. Depspace: A byzantine fault-tolerant coordination service. *SIGOPS Oper. Syst. Rev.*, 42(4):163–176, April 2008.

[35] Michael Blackstock and Rodger Lea. Toward a distributed data flow platform for the web of things (distributed node-red). In *Proceedings of the 5th International Workshop on Web of Things*, pages 34–39. ACM, 2014.

[36] Olivier Boissier, Rafael H. Bordini, Jomi F. Hübner, Alessandro Ricci, and Andrea Santi. Multi-agent oriented programming with jacamo. *Sci. Comput. Program.*, 78(6):747–761, June 2013.

[37] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. Fog computing and its role in the internet of things. In *Proceedings of*

the First Edition of the MCC Workshop on Mobile Cloud Computing, MCC '12, pages 13–16, New York, NY, USA, 2012. ACM.

[38] Carsten Bormann, Angelo P Castellani, and Zach Shelby. Coap: An application protocol for billions of tiny internet nodes. *IEEE Internet Computing*, 16(2):62–67, 2012.

[39] Alessio Botta, Walter De Donato, Valerio Persico, and Antonio Pescapé. On the integration of cloud computing and internet of things. In *Future Internet of Things and Cloud (FiCloud), 2014 International Conference on*, pages 23–30. IEEE, 2014.

[40] Alan Burns and Robert Davis. Mixed criticality systems - a review. Report. University of York, 2016.

[41] J. Buysse, M. De Leenheer, L. M. Contreras, J. I. Aznar, J. R. Martinez, G. Landi, and C. Develder. Ncp+: An integrated network and it control plane for cloud computing. *Optical Switching and Networking*, 11:137–152, January 2014.

[42] C.Y. Chen, J.H. Fu, T. Sung, P.F. Wang, E. Jou, and M.W. Feng. Complex event processing for the internet of things and its applications. In *IEEE International Conference on Automation Science and Engineering*, August 2014.

[43] H. Cho, J. Jung, B. Cho, Y. Jin, S. W. Lee, and Y. Baek. Precision time synchronization using ieee 1588 for wireless sensor networks. In *2009 International Conference on Computational Science and Engineering*, volume 2, pages 579–586, Aug 2009.

[44] A. Choudhary, S. Rana, and K. J. Matahai. A critical analysis of energy efficient virtual machine placement techniques and its optimization in a cloud computing environment. *Procedia Computer Science*, 78:132–138, 2016.

[45] Andy HF Chow, Alex Santacreu, Ioannis Tsapakis, Garavig Tanasaranond, and Tao Cheng. Empirical assessment of urban traffic congestion. *Journal of advanced transportation*, 48(8):1000–1016, 2014.

[46] William Cox and Toby Considine. Structured energy: Microgrids and autonomous transactive operation. In *Innovative Smart Grid Technologies (ISGT), 2013 IEEE PES*, pages 1–6. IEEE, 2013.

[47] Gianpaolo Cugola and Alessandro Margara. Processing flows of information: From data stream to complex event processing. *ACM Computing Surveys (CSUR)*, 44(3):15, 2012.

[48] O. Dag and B. Mirafzal. On stability of islanded low-inertia microgrids. In *Proc. of 2016 Clemson University Power Systems Conference (PSC)*, pages 1–7, March 2016.

[49] E. del Val, M. Rebollo, and V. Botti. Enhancing decentralized service discovery in open service-oriented multi-agent systems. *Autonomous Agents and Multi-Agent Systems*, 28:1–30, January 2014.

[50] E. Denti, A. Omicini, and A. Ricci. Coordination tools for mas development and deployment. *Applied Artificial Intelligence*, 16:721–752, 2002.

[51] JBOSS developer. Compensating transactions: When acid is too much. developer.jboss.org. Last accessed Feb 2018.

[52] Rinku Dewri, Prasad Annadata, Wisam Eltarjaman, and Ramakrishna Thurimella. Inferring trip destinations from driving habits data. In *Proceedings of the 12th ACM Workshop on Workshop on Privacy in the Electronic Society*, WPES '13, pages 267–272, New York, NY, USA, 2013. ACM.

[53] K. Dolui and S. K. Datta. Comparison of edge computing implementations: Fog computing, cloudlet and mobile edge computing. In *Proceeding Global Internet of Things Summit*, pages 1–6, Geneva, Switzerland, 2017.

[54] A. Dubey, G. Karsai, and S. Abdelwahed. Compensating for timing jitter in computing systems with general-purpose operating systems. In *2009 IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing*, pages 55–62, March 2009.

[55] Farzam Ehsani. Blockchain in finance: From buzzword to watchword in 2016, December 2016. Last accessed: Dec. 2017.

[56] S. Eisele, A. Laszka, A. Mavridou, and A. Dubey. SolidWorx: A resilient and trustworthy transactive platform for smart and connected communities. *ArXiv e-prints*, April 2018.

[57] S. Eisele, I. Mardari, A. Dubey, and G. Karsai. Riaps: Resilient information architecture platform for decentralized smart systems. In

2017 IEEE 20th International Symposium on Real-Time Distributed Computing (ISORC), pages 125–132, May 2017.

[58] Keke Gai, Meikang Qiu, Hui Zhao, Lixin Tao, and Ziliang Zong. Dynamic energy-aware cloudlet-based mobile cloud computing model for green computing. *Journal of Network and Computer Applications*, 59(Supplement C):46 – 54, 2016.

[59] Ana García-Fornes, Jomi Fred Hübner, Andrea Omicini, Juan A. Rodríguez-Aguilar, and Vicente J. Botti. Infrastructures and tools for multiagent systems for the new generation of distributed systems. *Eng. Appl. of AI*, 24(7):1095–1097, 2011.

[60] Marisol García-Valls and Roberto Baldoni. Adaptive middleware design for cps: Considerations on the os, resource managers, and the network run-time. In *Proceedings of the 14th International Workshop on Adaptive and Reflective Middleware*, ARM 2015, pages 3:1–3:6, 2015.

[61] Marisol García-Valls, Tommaso Cucinotta, and Chenyang Lu. Challenges in real-time virtualization and predictable cloud computing. *Journal of Systems Architecture*, 60(9):726 – 740, 2014.

[62] Marisol Garc{ia-Valls, Jorge Domínguez-Poblete, Imad Eddine Touahria, and Chenyang Lu. Integration of data distribution service and distributed partitioned systems. *Journal of Systems Architecture*, 83:23 – 31, 2018.

[63] Marisol García-Valls, Iago Rodríguez Lopez, and Laura Fernández-Villar. iland: An enhanced middleware for real-time reconfiguration of service oriented distributed real-time systems. *IEEE Trans. Industrial Informatics*, 9(1):228–236, 2013.

[64] Amin Ghafouri, Aron Laszka, Abhishek Dubey, and Xenofon Koutsoukos. Optimal detection of faulty traffic sensors used in route planning. In *Proceedings of the 2nd International Workshop on Science of Smart City Operations and Platforms Engineering*, pages 1–6. ACM, 2017.

[65] Amin Ghafouri, Aron Laszka, Abhishek Dubey, and Xenofon D. Koutsoukos. Optimal detection of faulty traffic sensors used in route planning. *CoRR*, abs/1702.02628, 2017.

[66] Rajrup Ghosh and Yogesh Simmhan. Distributed scheduling of event analytics across edge and cloud. In *CoRR, (1608.01537)*, 2016.

[67] OpenFog Consortium Architecture Working Group et al. Openfog architecture overview. *White Paper, February*, 2016.

[68] Randolph W Hall. Non-recurrent congestion: How big is the problem? are traveler information systems the solution? *Transportation Research Part C: Emerging Technologies*, 1(1):89–103, 1993.

[69] Yusuke Hara and Eiji Hato. A car sharing auction with temporal-spatial od connection conditions. *Transportation Research Part B: Methodological*, 2017.

[70] Carl Hewitt, Peter Bishop, and Richard Steiger. A universal modular actor formalism for artificial intelligence. In *Proceedings of the 3rd international joint conference on Artificial intelligence*, pages 235–245. Morgan Kaufmann Publishers Inc., 1973.

[71] Benjamin Hindman, Andy Konwinski, Matei Zaharia, Ali Ghodsi, Anthony D. Joseph, Randy Katz, Scott Shenker, and Ion Stoica. Mesos: A platform for fine-grained resource sharing in the data center. In *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation*, NSDI'11, pages 295–308, Berkeley, CA, USA, 2011. USENIX Association.

[72] F. Hu, Q. Hao, and K. Bao. A survey on software-defined network and openflow: From concept to implementation. *IEEE Communications Surveys Tutorials*, 16(4):2181–2206, 2014.

[73] Urs Hunkeler, Hong Linh Truong, and Andy Stanford-Clark. Mqtt-s—a publish/subscribe protocol for wireless sensor networks. In *Communication systems software and middleware and workshops, 2008. comsware 2008. 3rd international conference on*, pages 791–798. IEEE, 2008.

[74] IETF. Rfc 5905. network Time Protocol (NTP) version 4. https://www.ietf.org/rfc/rfc5905.txt, February 2018.

[75] Real-Time Innovations. RTI Data Distribution Service. http://www.rti.com/products/dds/index.html.

[76] Intellinium. Fog, edge, cloud and mist computing, 2017. Last accessed Nov 2017.

[77] Y. Jararweh, L. Tawalbeh, F. Ababneh, and F. Dosari. Resource efficient mobile computing using cloudlet infrastructure. In *in IEEE*

Ninth International Conference on Mobile Ad-hoc and Sensor Networks (MSN), pages 373–377, December 2013.

[78] Shunsuke Kamijo, Yasuyuki Matsushita, Katsushi Ikeuchi, and Masao Sakauchi. Traffic monitoring and accident detection at intersections. *IEEE transactions on Intelligent transportation systems*, 1(2):108–118, 2000.

[79] R. Kandoi and M. Antikainen. Denial-of-service attacks in openflow sdn networks. In *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, pages 1322–1326, May 2015.

[80] A. R. Khan, M. Othman, S. A. Madani, and S. U. Khan. A survey of mobile cloud computing application models. *IEEE Communications Surveys Tutorials*, 16(1):393–413, First 2014.

[81] Irwin King, Jiexing Li, and Kam Tong Chan. A brief survey of computational approaches in social computing. In *Proceedings of the 2009 International Joint Conference on Neural Networks*, pages 2699–2706, 2009.

[82] Alexander Kleiner, Bernhard Nebel, and Vittorio A Ziparo. A mechanism for dynamic ride sharing based on parallel auctions. In *IJCAI*, volume 11, pages 266–272, 2011.

[83] Koen Kok and Steve Widergren. A society of devices: Integrating intelligent distributed resources with transactive energy. *IEEE Power and Energy Magazine*, 14(3):34–45, 2016.

[84] Xiangjie Kong, Ximeng Song, Feng Xia, Haochen Guo, Jinzhong Wang, and Amr Tolba. Lotad: long-term traffic anomaly detection based on crowdsourced bus trajectory data. *World Wide Web*, pages 1–23, 2017.

[85] Fragkiskos Koufogiannis and George Pappas. Diffusing private data over networks. 2015.

[86] Diego Kreutz, Fernando M. V. Ramos, Paulo Jorge Esteves Veríssimo, Christian Esteve Rothenberg, Siamak Azodolmolky, and Steve Uhlig. Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*, 103(1):14–76, 2015.

[87] S. Krčo, B. Pokrić, and F. Carrez. Designing iot architecture(s): A european perspective. In *2014 IEEE World Forum on Internet of Things (WF-IoT)*, pages 79–84, March 2014.

[88] Karla Kvaternik, Aron Laszka, Michael Walker, Douglas C. Schmidt, Monika Sturm, Martin Lehofer, and Abhishek Dubey. Privacy-preserving platform for transactive energy systems. *CoRR*, abs/1709.09597, 2017.

[89] Simon Kwoczek, Sergio Di Martino, and Wolfgang Nejdl. Predicting and visualizing traffic congestion in the presence of planned special events. *Journal of Visual Languages & Computing*, 25(6):973–980, 2014.

[90] Simon Kwoczek, Sergio Di Martino, and Wolfgang Nejdl. Stuck around the stadium? an approach to identify road segments affected by planned special events. In *Intelligent Transportation Systems (ITSC), 2015 IEEE 18th International Conference on*, pages 1255–1260. IEEE, 2015.

[91] Leslie Lamport. The part-time parliament. *ACM Trans. Comput. Syst.*, 16(2):133–169, May 1998.

[92] Leslie Lamport. Paxos made simple. *ACM Sigact News*, 32(4):18–25, 2001.

[93] Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 4(3):382–401, 1982.

[94] Aron Laszka, Abhishek Dubey, Michael Walker, and Douglas C. Schmidt. Providing privacy, safety, and security in iot-based transactive energy systems using distributed ledgers. *CoRR*, abs/1709.09614, 2017.

[95] Aron Laszka, Abhishek Dubey, Michael Walker, and Douglas C. Schmidt. Providing privacy, safety, and security in iot-based transactive energy systems using distributed ledgers. *CoRR*, abs/1709.09614, 2017.

[96] Kfir Lev-Ari, Edward Bortnikov, Idit Keidar, and Alexander Shraer. Modular composition of coordination services. In *Proceedings of the 2016 USENIX Conference on Usenix Annual Technical Conference*, USENIX ATC '16, pages 251–264, Berkeley, CA, USA, 2016.

[97] Michael W Levin, Kara M Kockelman, Stephen D Boyles, and Tianxin Li. A general framework for modeling shared autonomous vehicles

with dynamic network-loading and dynamic ride-sharing application. *Computers, Environment and Urban Systems*, 64:373–383, 2017.

[98] H. Li, G. Shou, Y. Hu, and Z. Guo. Mobile edge computing: progress and challenges. In *Mobile Cloud Computing, Services, and Engineering (MobileCloud), 2016 4th IEEE International Conference on. IEEE*, pages 83–84, 2016.

[99] Linux. Kvm - kernel based virtual machine. http://www.linux-kvm.com. Accessed Jan 2018.

[100] Linux Containers. Infrastructure for container projects. www.linuxcontainers.org. Last accessed February 2018.

[101] P. Liu, D. Willis, and S. Banerjee. Paradrop: Enabling lightweight multi-tenancy at the network's extreme edge. In *2016 IEEE/ACM Symposium on Edge Computing (SEC)*, pages 1–13, Oct 2016.

[102] Wei Liu, Yu Zheng, Sanjay Chawla, Jing Yuan, and Xie Xing. Discovering spatio-temporal causal interactions in traffic data streams. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1010–1018. ACM, 2011.

[103] S Lockwood. The 21st century operation oriented state dots, nchrp project 20–24. *Transportation research board, American Association of State Highway and Transportation Officials, Washington, DC*, 2006.

[104] Xiao-Yun Lu, Pravin Varaiya, Roberto Horowitz, and Joe Palen. Faulty loop data analysis/correction and loop fault detection. In *15th World Congress on Intelligent Transport Systems and ITS America's 2008 Annual Meeting*, 2008.

[105] M. Luck and P. McBurney. Computing as interaction: Agent and agreement technologies, 2008.

[106] M. Luck, P. McBurney, O. Shehory, and S. Willmott. *Agent Technology: Computing as Interaction (A Roadmap for Agent Based Computing)*. AgentLink, 2005.

[107] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief. A survey on mobile edge computing: The communication perspective. In *IEEE Communications Surveys and Tutorials*, volume 19, pages 2322–2358, August 2017.

[108] M. Masdari, S. S. Nabavi, and V. Ahmadi. An overview of virtual machine placement schemes in cloud computing. *Journal of Network and Computer Applications*, 66:106–127, May 2016.

[109] Anastasia Mavridou and Aron Laszka. Designing secure Ethereum smart contracts: A finite state machine based approach. In *Proceedings of the 22nd International Conference on Financial Cryptography and Data Security (FC)*, February 2018.

[110] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, and J. Rexford. Openflow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*, pages 69–74, 2008.

[111] Peter Mell and Tim Grance. The NIST definition of cloud computing,v15, NIST. 2009.

[112] Ronald B Melton. Gridwise transactive energy framework (draft version). Technical report, Pacific Northwest National Laboratory, Richland, WA, 2013.

[113] Microsoft. Microsoft azure. http://azure.microsoft.com/Azure. Accessed Jan 2018.

[114] Sun microsystems. Java Transaction API (JTA). http://java.sun.com:80/javaee/technologies/jta/. Last accessed Feb 2018.

[115] Rimantas Mocevicius. *CoreOS Essentials*. Packt Publishing Ltd, 2015.

[116] M. B. Mollah, M. A. K. Azad, and A. Vasilakos. Security and privacy challenges in mobile cloud computing: Survey and way ahead. In *J. Netw. Comput. Appl.*, volume 84, pages 38–54, 2017.

[117] Mohamed Al Morsy, John Grundy, and Ingo Müller. An analysis of the cloud computing security problem. In *in Proceedings of APSEC 2010 Cloud Workshop*, Sydney, Australia, November 2010.

[118] Kurt Mueffelmann. Uber's privacy woes should serve as a cautionary tale for all companies. Wired Magazine, Jan. 2015.

[119] Mithun Mukherjee, Rakesh Matam, Lei Shu, Leandros Maglaras, Mohamed Amine Ferrag, Nikumani Choudhury, and Vikas Kumar. Security and privacy in fog computing: Challenges. In *IEEE Access*, volume 5, pages 19293–19304, September 2017.

[120] T. Neagoe, V. Cristea, and L. Banica. Ntp versus ptp in com puter networks clock synchronization. In *2006 IEEE International Symposium on Industrial Electronics*, volume 1, pages 317–362, July 2006.

[121] Netflix. Netflix video streaming. https://www.netflix.com/.

[122] P. B. Nichols. The permanent web for healthcare with ipfs and blockchain. https://www.cio.com/article/3174144/innovation/the-permanent-web-for-healthcare-with-ipfs-and-blockchain.html, February 2017.

[123] OASIS. Message queue telemetry transport (mqtt) v3.1.1. http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html. Last accessed Feb 2018.

[124] OMG. The Data Distribution Service specification, v1.2. http://www.omg.org/spec/DDS/1.2, 2007.

[125] Diego Ongaro and John Ousterhout. In search of an understandable consensus algorithm. In *Proceedings of the 2014 USENIX Conference on USENIX Annual Technical Conference*, USENIX ATC'14, pages 305–320, Berkeley, CA, USA, 2014. USENIX Association.

[126] Diego Ongaro and John Ousterhout. In search of an understandable consensus algorithm. In *Proc. USENIX Annual Technical Conference*, pages 305–320, 2014.

[127] F. Paolucci, F. Cugini, A. Giorgetti, and P. Castoldi N. Sambo. A survey on the path computation element (pce) architecture. *IEEE Communications Surveys & Tutorials*, 15(4):1819–1841, January 2013.

[128] J. S. Preden, K. Tammemäe, A. Jantsch, M. Leier, A. Riid, and E. Calis. The benefits of self-awareness and attention in fog and mist computing. *Computer*, 48(7):37–45, July 2015.

[129] J. S. Preden, K. Tammemäe, A. Jantsch, M. Leier, A. Riid, and E. Calis. The benefits of self-awareness and attention in fog and mist computing. *Computer*, 48(7):37–45, July 2015.

[130] Robert D Rasmussen. Goal-based fault tolerance for space systems using the mission data system. In *Aerospace Conference, 2001, IEEE Proceedings.*, volume 5, pages 2401–2410. IEEE, 2001.

[131] Sean Rhea, Brighten Godfrey, Brad Karp, John Kubiatowicz, Sylvia Ratnasamy, Scott Shenker, Ion Stoica, and Harlan Yu. Opendht: a public dht service and its uses. In *ACM SIGCOMM Computer Communication Review*, volume 35, pages 73–84. ACM, 2005.

[132] Stephen Peter Robinson. *The development and application of an urban link travel time model using data derived from inductive loop detectors*. PhD thesis, University of London, 2006.

[133] C. Samal, L. Zheng, F. Sun, L. J. Ratliff, and A. Dubey. Towards a Socially Optimal Multi-Modal Routing Platform. *ArXiv e-prints*, February 2018.

[134] M. Sapienza, E. Guardo, M. Cavallo, G. La Torre, G. Leombruno, and O. Tomarchio. Solving critical events through mobile edge computing: An approach for smart cities. In *2016 IEEE International Conference on Smart Computing (SMARTCOMP)*, pages 1–5, May 2016.

[135] Mahadev Satyanarayanan, Pieter Simoens, Yu Xiao, Padmanabhan Pillai, Zhuo Chen, Kiryong Ha, Wenlu Hu, and Brandon Amos. Edge analytics in the internet of things. *IEEE Pervasive Computing*, 14, April 2015.

[136] David Schrank, Bill Eisele, Tim Lomax, and Jim Bak. 2015 urban mobility scorecard. 2015.

[137] Scott Shenker. The future of netowrking and the past of network protocols. http://www.opennetsummit.org/archives/oct11/shenker-tue.pd, 2011. Open Network Summit.

[138] Amit Sheth, Pramod Anantharam, and Cory Henson. Physical-cyber-social computing: An early 21st century approach. *IEEE Intelligent Systems*, 28(1):78–82, 2013.

[139] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu. Edge computing: Vision and challenges. In *IEEE Internet of Things Journal*, volume 3, pages 637–646, October 2016.

[140] X. Shi, H. Lin, H. Jin, B. B. Zhou, Z. Yin, S. Di, and S. Wu. Giraffe: A scalable distributed coordination service for large-scale systems. In *2014 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 38–47, Sept 2014.

[141] C. Sierra, V. Botti, and S. Ossowski. Agreement computing. *KI-Knstliche Intelligenz*, 25:57–61, 2011.

[142] Y. Simmhan, S. Aman, A. Kumbhare, R. Liu, S. Stevens, Q. Zhou, and V. Prasanna. Cloud-based software platform for big data analytics in smart grids. *Computing in Science Engineering*, 15(4):38–47, July 2013.

[143] J. Spillner and A. Schill. Towards dispersed cloud computing. In *2014 IEEE International Black Sea Conference on Communications and Networking (BlackSeaCom)*, pages 170–174, May 2014.

[144] Ivan Stojmenovic and Sheng Wen. The fog computing paradigm: Scenarios and security issues. In *Computer Science and Information Systems (FedCSIS), 2014 Federated Conference on*, volume 2, pages 1–8, Warsaw, Poland, September 2014.

[145] H. L. Storey. Implementing an integrated centralized model-based distribution management system. In *2011 IEEE Power and Energy Society General Meeting*, pages 1–2, July 2011.

[146] Sriskandarajah Suhothayan, Kasun Gajasinghe, Isuru Loku Narangoda, Subash Chaturanga, Srinath Perera, and Vishaka Nanayakkara. Siddhi: a second look at complex event processing architectures. *ACM workshop on Gateway computing environments*, Nov 2011.

[147] B. Varghese, N. Wang, S. Barbhuiya, P. Kilpatrick, and D. S. Nikolopoulos. Challenges and opportunities in edge computing. In *2016 IEEE International Conference on Smart Cloud (SmartCloud)*, pages 20–26, Nov 2016.

[148] Harini Veeraraghavan, Paul Schrater, and Nikolaos Papanikolopoulos. Switching kalman filter-based approach for tracking and event detection at traffic intersections. In *Intelligent Control, 2005. Proceedings of the 2005 IEEE International Symposium on, Mediterrean Conference on Control and Automation*, pages 1167–1172. IEEE, 2005.

[149] T. Verbelen, P. Simoens, F.D. Turck, and B. Dhoedt. Cloudlets: bringing the cloud to the mobile user. In *in Proceedings of the 3rd ACM workshop on mobile cloud computing and services*, pages 29–36, Low Wood Bay, UK, 2012.

[150] Dale Willis, Arkodeb Dasgupta, and Suman Banerjee. ParaDrop: A Multi-tenant Platform to Dynamically Install Third Party Services on Wireless Gateways. In *Proceedings of the 9th ACM workshop on Mobility in the evolving internet architecture*, pages 43–48. ACM, 2014.

[151] Michael Wooldridge. *An Introduction to MultiAgent Systems*. Wiley Publishing, 2nd edition, 2009.

[152] Michael Wooldridge and Nicholas R. Jennings. Intelligent agents: Theory and practice. *Knowledge Engineering Review*, 10:115–152, 1995.

[153] Lin Xu, Yang Yue, and Qingquan Li. Identifying urban traffic congestion pattern from historical floating car data. *Procedia-Social and Behavioral Sciences*, 96:2084–2095, 2013.

[154] Shiming Yang, Konstantinos Kalpakis, and Alain Biem. Detecting road traffic events by coupling multiple timeseries with a nonparametric bayesian method. *IEEE Transactions on Intelligent Transportation Systems*, 15(5):1936–1946, 2014.

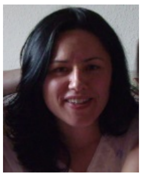
[155] S. Yi, C. Li, and Q. Li. A survey of fog computing: Concepts, applications and issues. In *Proceedings of the 2015 Workshop on Mobile Big Data. ACM*, 2015.

[156] Shanhe Yi, Cheng Li, and Qun Li. A survey of fog computing: Concepts, applications and issues. In *Proceedings of the 2015 Workshop on Mobile Big Data*, Mobidata '15, pages 37–42, New York, NY, USA, 2015. ACM.

[157] Yong Yuan and Fei-Yue Wang. Towards blockchain-based intelligent transportation systems. In *Intelligent Transportation Systems (ITSC), 2016 IEEE 19th International Conference on*, pages 2663–2668. IEEE, 2016.

[158] Nikolaos Zygouras, Nikolaos Panagiotou, Nikos Zacheilas, Ioannis Boutsis, Vana Kalogeraki, Ioannis Katakis, and Dimitrios Gunopulos. Towards detection of faulty traffic sensors in real-time. In *MUD@ ICML*, pages 53–62, 2015.

Bios



Marisol García-Valls is associate professor in the *Department of Telematics Engineering of Universidad Carlos III de Madrid, Spain.* She has been the leader of the Distributed Real Time Systems Lab for more than 10 years. Her research interests are IoT systems and distribution software design and performance for fog computing. She is also involved in Cyber-physical systems, quality of service, operating systems, predictable cloud computing and virtualization and time sensitive systems (soft real-time). http://www.it.uc3m.es/mvalls/

Abhishek Dubey is associate professor at Vanderbilt University, USA. His research goals are to develop tools, platforms and analytical techniques required for dynamic and resilient cyber-physical platforms. He is especially interested in applying his research to smart grid. His current research focuses on developing hierarchical failure propagation models for understanding failure dynamics in smart grid and using that information for online fault diagnostics and prognostics. https://www.engineering.vanderbilt.edu/bio/abhishek-dubey

Vicent Botti is full professor of computer systems at Universitat Politécnica de Valéncia, Spain. His current research activities include the following interdisciplinary areas: Agreement Technologies; Virtual organizations, automatic negotiation, argumentation, trust, reputation and privacy; Multiagent Systems; and architectures and platforms, development Technologies, agent based social simulation, agent based intelligent manufacturing Systems, multiagent adaptive Systems, agreement Networks, decentralized services management. http://www.users.dsic.upv.es/vbotti/