Towards IoT Processes on the Edge

Schahram Dustdar(✉) and Ilir Murturi

Distributed Systems Group, TU Wien, Vienna, Austria
{dustdar,imurturi}@dsg.tuwien.ac.at

Abstract. Edge computing is a fundamental enabler for the proliferation of the Internet of Things (IoT). Resources, including compute and storage, are increasingly located at the edge of the network and bridge the gap between the cloud and IoT entities. Edge computing enables low-latency, privacy-awareness, and resilient applications. Many of the applications are in fact business process-based and are distributed over edge as well as cloud resources. Edge devices can be used to analyze high-volume IoT data streams for on-premises monitoring and alerting, and at the same time to aggregate and forward these data to other premises or the cloud for long-term storage and analytics. Many operational and business challenges can be solved by running applications on edge resources or on-premises of Edge-Cloud infrastructure. However, the broad range of IoT application requirements concerning latency, QoS, or fault-tolerance, combined with the heterogeneous and dynamic nature of edge networks, make it particularly challenging to develop, configure, deploy, and operate such applications. In this paper, we discuss some of the research issues with span the domains of business processes engineering and edge computing.

Keywords: Edge-Cloud continuum · Distributed processes

1 Introduction

In recent years, the Internet of Things (IoT) has been diffused into the society, and many services are constructed on the top of IoT technologies in various industries such as Industrial Manufacturing, Healthcare, Lifestyle, Automotive, and Smart Building, just to name a few. At the same time, it is well accepted that a centralized architecture does not scale well regarding the enormous number of devices, although the system infrastructure of central computers processing those data have improved by cloud computing technologies. In contrast to a fully distributed and decentralized architecture (e.g., peer-to-peer network), many IoT services need to maintain a partially centralized design to operate the service. Nonetheless, the significant portion of decentralization is often achieved by delegating functions from central servers to edge computing devices [1]. Edge devices are essentially computers close to the edge of the network, hence, closer to the sensors, which create the data streams to be processed later. Edge devices, therefore, can be utilized to process data streams pumped into an IoT system

M. Aiello et al. (Eds.): Papazoglou Festschrift, LNCS 12521, pp. 167–178, 2021.
https://doi.org/10.1007/978-3-030-73203-5_13

by providing analytics capabilities, providing decision functions as to which data has to be forwarded to a cloud infrastructure or not and possibly also providing decisions based on modeled business processes as well as business models.

Another important aspect in the age of the IoT is the heterogeneity of IoT components. Several vendors provide different products not only across different layers but also for the same type of components. Different products may have different operating systems, available support for programming languages, resource constraints, and so on. In contrast to our expectations, each device's functions are hard-coded in most of the current implementations. This causes inflexibility and limited extensibility for future changes. This leads to a plethora of point-to-point solutions being developed based on proprietary protocols. Any change being made to one IoT component leads to many possible changes in many other components thus leading to a situation which resembles the state of software infrastructure in the age of enterprise computing before the emergence of Web services and Service Oriented Architectures and their respective middleware infrastructure such as Enterprise Service Bus (ESB). This insight led us to (re-)design IoT systems into a distributed and decentralized architecture that is scalable not only in numbers but also concerning its heterogeneity as well as its notion of processes being supported.

A similar problem to managing heterogeneous entities with different functionalities in highly distributed yet still partially centralized settings has been addressed by business process management (BPM) for several years. Before the establishment of business process engines (BPE), or workflow engines, processes were directly implemented in information systems. In other words, tasks and their sequences were hard-coded altogether in the source code of the information system. This style of design causes inflexibility and low extensibility against changes demanded from ones in business requirements. As a solution to this problem, BPEs enable overseeing the execution and maintenance of workflows and their enactments with high flexibility and extensibility by integrating different data sources and execution entities spread across IT applications and services.

1.1 Background

A business process consists of a set of related structured events, tasks, decisions, inputs, and outputs. Events happen in the environment and trigger tasks in a business process. Tasks are the smallest units, which can be performed by various entities such as people, machines, or software within the business process. A process may contain decisions based on certain business rules apart from the process itself. Inputs can be physical goods or in-materialistic (such as filling out a form and inputting data) required to complete a business process in question, including information. A process produces outputs, which is the goal of the process, to the environment.

A business process engine (BPE) is a software middleware overseeing the technical instantiations of business processes and their associated activities. A BPE copes with more than one application and services regardless of their layers

(e.g., front-end, back-end, or middleware), boundaries (e.g., internal or external services), or vendors. A BPE becomes involved in interlinking and interprocessing between activities by routing and transforming data across different components, rather than executing each activity itself. Those interlinks are automated based on the process specification often given by a business process modeling language, which the BPE supports. Users can change how activities are interlinked by updating the process model.

Among others, BPE's benefits in the context of BPM that are relevant to the rest of this article can be summarized with two aspects: flexibility and monitoring. Since a BPE separates each activity's actual technical execution and its coordination (orchestration), we can change the process specification without changing the technical implementation, thus acquiring flexibility for future changes, which might be referred to as extensibility or agility depending on the context. Monitoring is a fundamental property of BPEs as it allows an understanding of utilized resources and their time as well as involved actors and additional auditing information.

1.2 Distributed Process Execution

Several approaches have been proposed to enable the distributed execution of business processes. Essentially, the BPE engine is replicated among computation entities (i.e., distributed clouds), and process activities are distributed among available entities. Muthusamy et al. [2] presents a Service Level Agreement (SLA) driven approach to BPM for service-oriented applications in environments such as cloud computing platforms. Initially, the proposed approach decomposes a business process into a set of dependent activities. Afterward, each activity is mapped into a set of distributed computation entities (i.e., execution engines) closer to the data sources. The coordination among activities (i.e., communication/messaging) is achieved by emitting and consuming events over the PADRES distributed publish/subscribe platform. The proposed approach aims to minimize the overall communication costs by executing process activities as closely as possible to the data source. A comprehensive survey on the distributed execution of business processes is provided by Wutke [3].

A similar approach can be adopted in the context of Edge-Cloud continuum systems. One the one hand, various mechanisms for resource coordination [4], service deployment [5], controlling elasticity in application [6], IoT system deployment [7], or monitoring [8] can be dynamically placed at the edge. On the other hand, a service deployment mechanism provides functionality to accept requests to deploy an IoT application (i.e., service) comprised of a set of small tasks (i.e., microservices [9]) and distribute them into the Edge-Cloud infrastructure. When placing decision mechanisms at the edge, a challenging aspect is each edge device's resource capabilities. For instance, the service deployment mechanism's main prerequisite is the real-time monitoring of available resources in the infrastructure. Placing both functions on the same edge device is computationally and network demanding; therefore, some functions should be delegated

to available edge devices or other computation entities that meet their Quality of Service (QoS). Thereby, distributing functionalities and dynamically place them in the heterogeneous and dynamic edge networks enable relieving complex computations to occur on a single computation entity. The same appears when considering distributing IoT application components over edge resources or on-premises of Edge-Cloud infrastructure.

2 An Overview of Edge-Cloud Continuum

Edge computing is positioned as one important architectural layer in addition to fog and cloud. It can be considered as paramount to systems including (but not limited to) IoT deployments and the cloud, providing data and control facilities to participating IoT devices. Up to now, several comprehensive surveys have been published, describing the edge computing paradigm and its challenges [10, 11]. According to [12], the Edge-Cloud architecture is divided into three layers: the cloud layer, the fog layer, and the edge layer (as illustrated in Fig. 1). A comprehensive description for each layer is given below.

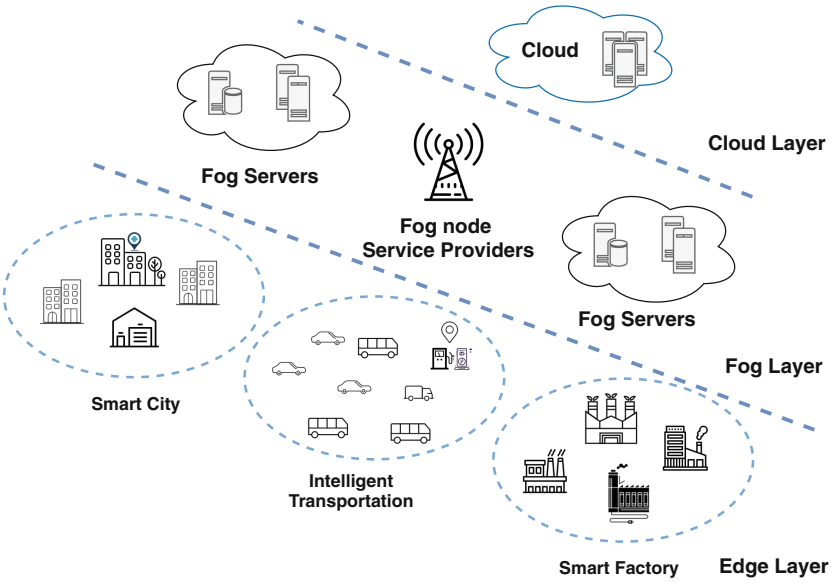

Fig. 1. An overview of Edge-Cloud architecture.

2.1 The Edge Layer

The edge layer is the lowest layer of the architecture, which represents the edge of the network. As can be denoted in Fig. 1, various domains such as smart city,

intelligent transportation, or smart factories can benefit from available computation devices closer to the IoT domain. Essentially, this leads to having various edge networks formed for different contexts. Generally speaking, edge networks are highly dynamic, heterogeneous, and resource-constrained environments. Such environments are composed of a set of low-powered end devices with various computation capabilities (e.g., smartphones, smartwatches, Raspberry Pis, etc.) and IoT resources (e.g., sensors, actuators, etc.).

Edge devices at this layer provide their computation and storing capabilities to process the IoT data streams generated by IoT resources. Essentially, these devices enable processing data streams as close as possible to the data sources and handling the most substantial network traffic that may occur. Nevertheless, resource-constrained edge devices do not pose the ability to process vast amounts of data. Consequently, processing such data on a single edge device may cause high-latency and poor overall performance. To overcome such challenges introduced by resource-constrained devices, distributing processes become a crucial aspect. Recent developments suggest adding more computation devices in edge networks. Concretely, through forming peer-to-peer edge networks, the scope of resources is extended, and the opportunity to distribute processes among devices becomes feasible.

Edge-to-edge collaboration, respectively, an *edge network*, provides a seamless opportunity for placing control mechanisms closer to the end-users, which results in creating more *autonomous environments* and less dependent on the external environments (i.e., cloud, or fog). For instance, in a smart home, residents should be able to control their devices and process data locally without depending on the cloud resources. However, it is worth noting that even though extending resource scope through edge-to-edge collaboration provides many benefits, there are still many complex tasks that cannot be processed at this layer. Thus, in such situations, the edge devices must forward their processing to the upper layers (i.e., fog or cloud systems).

2.2 The Fog Layer

The fog layer includes a set of powerful devices in charge of managing, connecting, and enabling sharing resources among different edge networks. This layer essentially represents the external environment where several fog nodes are connected, offering computation and storage resources for edge networks and roaming end-devices in proximity. Similarly, fog devices (e.g., servers and base stations) are connected, forming fog infrastructure. In contrast to the edge networks, the fog infrastructure tends to be composed of stationary and powerful devices (typically maintained and provided by Telco operators). Several tasks can be deployed at this layer, for example, processing data streams, caching, device management, and privacy protection.

At first glance, one can note that both layers provide similar features. Generally speaking, edge and fog paradigms foresee enabling more computation resources closer to the end-users and the IoT/sensor domain - at the edge of the network, respectively. However, the most significant difference between the

two layers is administrative differences and responsibilities. Furthermore, fog nodes (e.g., deployed in base stations) may provide their services for larger geographical areas. For instance, intelligent transportation systems may benefit from connecting and processing vehicle data in fog infrastructure [13].

Even though fog infrastructure provides more powerful devices, long-term data storage is impractical—similarly, processing tasks with heavy computation requirements are infeasible. Thus, in such situations, fog devices must forward their data and heavy computation tasks to the upper layer. Nonetheless, both layers provide low-latency services since the end-devices are closer to the source where the data is produced and consumed.

2.3 The Cloud Layer

The cloud layer provides "unlimited" computational and storage resources. This layer includes cloud servers that are deployed far away from the end devices and the IoT domain. Essentially, the cloud-based servers perform computationally intensive operations received from lower-layers of the architecture. Such environments provide advanced features for both end-users and service providers, such as performance configurations and security controls. Moreover, this computing utility has been seen as a critical component for the development, deployment, and execution of IoT platforms promising to meet the general community's everyday needs.

Despite the numerous resources available, this paradigm faces increasing challenges in meeting new IoT applications' stringent requirements. At present, geographically distributed IoT devices with intensive data generation cannot efficiently utilize resources available in Cloud environments [12]. Transferring intense and large amounts of data to a centralized cloud over Wide Area Networks (WAN) generates latencies and poses risks of service unavailability due to the non-persistent connectivity or eventually scheduled system maintenance. On the other hand, real-time distributed applications require fast response time, high-availability, and increased privacy, which centralized environments such as clouds often fail to fulfill.

3 Use Cases of Edge-Cloud Systems

3.1 Emergency Situations

To motivate our subsequent discussion, we consider emergencies such as natural disasters (e.g., earthquakes, fires, floods) in the city. Emergencies like earthquakes may affect various city zones, which can damage infrastructure, cause injury or loss of life, and trap people under buildings. In such situations, time is valuable, and drones may be used to analyze the entire situation and help rescue teams find and communicate with victims under a collapsed building. In this scenario, we consider multiple connected drones (i.e., form an edge neighborhood) flying over the city's affected areas (i.e., neighborhoods) aiming to provide

services for the rescue teams in finding victims under a collapsed building. Each drone (i.e., edge node) is equipped with various computation capabilities and integrated sensors (e.g., radar sensor, infrared cameras, etc.). We consider that drones are multi-purpose devices where the rescue teams may request to deploy various services depending on the emergency. Meanwhile, base stations may provide computational and storage capabilities (i.e., fog nodes) and provide docker charge stations for charging drones. At the same time, cloud capabilities may be used to store data for long terms.

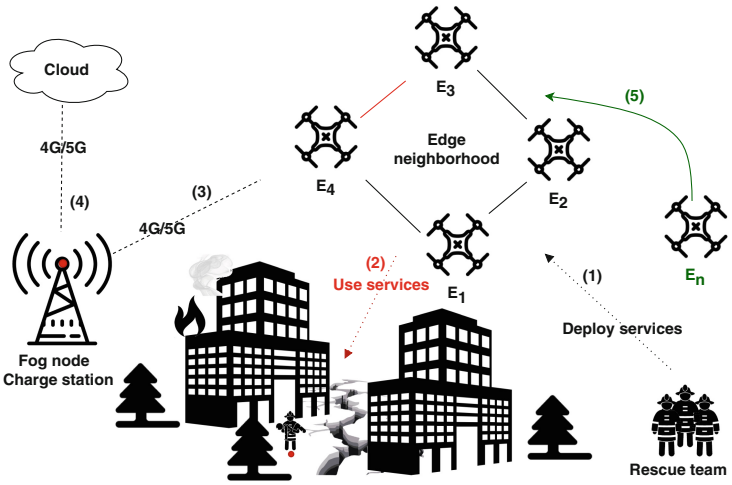

Fig. 2. IoT public safety service.

We assume that a rescue team deploys (1) a public safety IoT service that detects a dangerous zone in the affected area (i.e., discovering cracks, smoke, hazardous gases, etc.) (as illustrated in Fig. 2). Such service aims at assisting rescue teams (2) in finding a safe path and avoiding dangerous zones. The service is dependent on various resources such as multiple infrared cameras, radar sensors, and an electronic nose that are integrated into numerous drones. Since each drone is a potential candidate to run the service, it is evident that each node should be able to automatically discover resources in a decentralized manner and make them available at runtime. In such use case scenarios, we cannot depend on physically static entities to provide their services (3–4). Additionally, the edge neighborhood cannot be built as a centralized edge-system and dependent on particular nodes. This due to the network dynamicity (i.e., drones may join (5) and leave often), connectivity and latency issues, and the master drone may run out of energy or get out of the connection range. As a result, shifting various processes closer to the edge and dynamically place them in the most suitable nodes is crucial. Thus, deploying decentralized decision mechanisms and dynamically placing them makes edge networks *autonomous environments* and less dependent on centralized nodes that are located far away.

3.2 Smart Building Evacuation

The building evacuation is another example that can benefit from enabling the executing of various processes on edge. We consider a smart building, equipped with an infrastructure that supports inhabitants to evacuate the building safely in case of fire. On each floor, several IoT sensors are deployed and connected to servers (e.g., fog nodes) through edge gateways. We consider a goal-driven IoT system (GDS) [7], which is composed of a set of devices with individual functionalities that connect and cooperate temporally to achieve the user goal. For instance, in a smart meeting room, a GDS could be dynamically formed by connecting the motion detection sensor, smart screen, and speakers to achieve the goal (e.g., *safe evacuation*). The motion detection sensor detects the presence of the people. The smart screen displays the location(s) of the fire and possible safe paths for evacuation. Speakers also play a voice message asking people to evacuate the building.

The servers are responsible for dynamically forming and enacting GDSs that facilitate the evacuation of the building. Essentially, each computation entity may become responsible for forming and enacting GDSs seamlessly. Assume that communication between the edge gateways on one floor and the on-premise server is lost, e.g., due to the fire. The edge gateway decides whether to connect to the cloud or collaborate with other edge devices to form and enact GDSs automatically. In case the connection to the cloud can not be established, or due to the high-latency, the edge gateways collaborate to form and enact GDSs at the edge network. Generally speaking, such GDSs include fewer IoT sensors and should require less computational resources compared to those formed by the fog or the cloud. As a result, deploying IoT systems at the edge overcomes many undesired and critical situations.

4 Distributed Processes on the Edge

In the past few years, researchers in edge and fog computing have been mostly focused on proposing multiple techniques for resource allocation problems aiming to minimize various trade-offs such as latency, bandwidth, energy consumption, or maximizing the utilization of resources at the edge. In general, IoT applications and their associated business process models are deployed according to the following models [14]: i) *everything in the cloud*, ii) *everything in the edge*, and iii) *hybrid edge-cloud model*. Essentially, allocation techniques may deploy software components entirely on a single environment (e.g., cloud or edge) [5], or components are deployed and executed in both cloud and edge.

Recent developments within distributed systems have led to emerging commercial cloud-based IoT and cloud/edge integration solutions. Edge computing platforms such as EdgeX Foundry[1], AWS IoT Greengrass[2], or Google IoT Edge[3]

[1] https://www.edgexfoundry.org/.
[2] https://aws.amazon.com/greengrass/.
[3] https://cloud.google.com/solutions/iot.

promise to bridge the gap between the IoT and the cloud by providing a flexible runtime for applications running at the edge. However, these systems are extremely limited in their operational capabilities, missing elasticity features, and lack of self-adaptive mechanisms required in dynamic edge and IoT settings.

In general, the proposed solutions assumed that application demands remain static and do not change over time. This implies that hardware resources are reserved more than needed to guarantee application functionality when the workload is increased. However, over-provisioning in resource-constrained edge infrastructures is highly impractical, resource-expensive, and decreases system performance considerably. Furthermore, application developers or end-users cannot specify QoS and elastic requirements of the application, as well as there is no mechanism support to enact them. To fill this gap, we identify challenges and potential solutions that the IoT applications deployed in Edge-Cloud architecture must overcome to fulfill their full potential.

4.1 IoT Application Requirements

Future IoT systems for the described Edge-Cloud continuum architectures need to hide their operational complexity from application developers. In particular, programmers should not have to deal with the heterogeneity of the edge network setting. For instance, developers should be able to express the context in which IoT application components are allowed to run and their requirements (e.g., QoS, elastic requirements, etc.) in a high-level way [15]. In particular, identifying the current and future demands of IoT applications from various areas is decisive for any contemporary IoT system's success. However, this necessitates that the programming model is intuitive for developers but expressive enough to help the execution system perform runtime decisions on (re)scheduling and scaling operations.

To overcome such challenges, as a potential candidate for defining these requirements, we consider a declarative language called *Simple Yet Beautiful Language* (SYBL) [6] and its runtime mechanism for controlling elasticity in applications. SYBL enables the user to specify elastic requirements at different granularities and enables applications to scale in elasticity space (*cost*, *resources*, and *quality*). In essence, SYBL allows the user to define: i) *monitoring* (i.e., specifying metrics to be monitored), ii) *constraints* (i.e., specifying the limits in which the monitored metrics are allowed to oscillate), iii) *strategies* (i.e., specifying actions to be taken when a constraint is violated), and iv) *priorities* (i.e., specifying constraints priority to be executed first). Furthermore, SYBL provides features such as enabling the user to achieve various trade-offs, such as specifying demands on the relation between cost, resources, and quality. For instance, when the cost is high, the IoT application needs to scale up to achieve higher service quality. Or, the IoT application should scale down in order to optimize the usage and the cost. Nevertheless, an extension to the language, as well as to the runtime mechanism, is required. Moreover, developing novel high-level constraints and enforcement strategies related to the IoT applications and Edge-Cloud architecture is crucial.

4.2 IoT System Deployment and Resilient Application Runtime

As we explore new IoT systems, IoT applications, and the heterogeneous edge networks, distributing system components and application processes among various computation entities becomes increasingly apparent. For instance, an IoT system can comprise a set of dependent software components (i.e., controlling module, scheduler, resource manager, etc.) that can be deployed individually on multiple edge devices. Thus, we analyze and discuss the pros and cons of three main IoT system architectures that enable distributing application tasks among edge devices, such as *centralized*, *distributed*, and *decentralized*.

In the centralized architecture, a single edge device acts as a master device responsible for monitoring and distributing tasks among other available computation entities in the Edge-Cloud continuum. In essence, placing a set of functionalities on a single edge device may be feasible in the context of small and non-dynamic edge networks (e.g., smart homes). As mentioned previously, centralized architecture does not scale easily, while a master edge device may be overwhelmed quickly due to resource limitations. In contrast, the distributed solution treats all edge devices equally in terms of system responsibilities. In an environment without a master device, nodes in proximity are consensually coordinated and distribute processes to some SLA agreement. In practice, distributed solutions may face latency issues when nodes need to find consensus to distribute processes, and the number of nodes in topology is limited. However, regardless of the approach, both solutions may have plenty of advantages in various IoT scenarios.

In the decentralized architecture, the master device functionalities may be placed *statically* (i.e., at design time) or *dynamically* (i.e., with self-adaptive capabilities). However, the dynamic nature of edge networks requires the continuous re-evaluation of placement decisions for such functionalities. Thus, a possible solution is considering election based algorithms. For instance, through initiating an *election* between edge devices, the most suitable node (e.g., in terms of computation power) is elected as a master device. In fact, through exchanging election results, nodes in the network come to the same result independently of each other. Such a solution denotes a very decentralized approach to automatically electing the master device and succeeding over the challenges introduced by mobile devices and possible failures in edge networks. However, to overcome the obstacles with resource-constrained edge devices, further improvements are required. For instance, the master device must delegate various functionalities to other nodes to handle the computation and network overheads.

To overcome such aforementioned challenges, a possible solution is to elect new coordinators (i.e., superpeers [16]) in the system. Each coordinator is responsible for managing a set of nodes in the edge network. As the network size grows, new coordinators are introduced on the system as well. In particular, coordinators provide similar functionalities as the master device. However, the master device becomes a supervising node responsible for managing coordinators, monitoring, and distributing applications among coordinators. For instance, a user can submit a request to the master device for application deployment. The master

device examines his/her geographically location and asks the closest coordinators to determine if their group can meet the application requirements. Afterward, the coordinator who fulfills application demands gets the application and distributes it to the group's edge devices. Nevertheless, various scheduling algorithms, communication protocols, self-adaption techniques, and monitoring tools are required to deploy IoT systems in a decentralized manner.

Another important aspect is executing software components on heterogeneous environments (i.e., edge layer, fog layer, or in the cloud layer). To this end, we require a homogeneous runtime platform that follows the "*run once, run anywhere*" model. Thus, to overcome such challenges, as a potential candidate for executing IoT applications, we consider Docker[4] or Java-based OSGi[5]. Nevertheless, an extension is required for the runtime mechanism to monitor the edge device's real-time internal resources (e.g., CPU, memory, storage), network QoS, and application performance (e.g., application responsiveness).

5 Summary and Conclusions

In recent years, processing IoT data streams closer to the end-users and IoT domain has received significant attention from the research community and industry stakeholders. Enabling to process data closer to the end-user can solve several operational and business challenges. Since then, we have seen a rapidly increasing number of available resources in IoT infrastructures. Essentially, resources with computation and storage capabilities (i.e., perceived as *edge devices*) promise to offer various services and enable processing data with low-latency, high-availability and increased privacy. However, with acquainting new IoT scenarios and their stringent requirements (i.e., latency, QoS, dynamicity, or fault-tolerance), deploying IoT systems and processing data on a single edge device becomes impractical. To that end, in this paper, we discussed some of the research issues and the necessity of decentralizing IoT systems and distributing processes among devices at the edge.

Acknowledgment. Research supported by the Research Cluster "Smart Communities and Technologies (Smart CT)" at TU Vienna.

References

1. Shi, W., Dustdar, S.: The promise of edge computing. Computer **49**(5), 78–81 (2016)
2. Muthusamy, V., Jacobsen, H.-A.: BPM in cloud architectures: business process management with SLAs and events. In: Hull, R., Mendling, J., Tai, S. (eds.) BPM 2010. LNCS, vol. 6336, pp. 5–10. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-15618-2_2

[4] https://www.docker.com/.
[5] https://www.osgi.org/.

3. Wutke, D.: Eine infrastruktur für die dezentrale ausführung von bpel-prozessen (2010)
4. Tsigkanos, C., Murturi, I., Dustdar, S.: Dependable resource coordination on the edge at runtime. Proc. IEEE **107**(8), 1520–1536 (2019)
5. Avasalcai, C., Dustdar, S.: Latency-aware distributed resource provisioning for deploying IoT applications at the edge of the network. In: Arai, K., Bhatia, R. (eds.) FICC 2019. LNNS, vol. 69, pp. 377–391. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-12388-8_27
6. Copil, G., Moldovan, D., Truong, H.L., Dustdar, S.: SYBL: an extensible language for controlling elasticity in cloud applications. In: 2013 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing, pp. 112–119. IEEE (2013)
7. Alkhabbas, F., Murturi, I., Spalazzese, R., Davidsson, P., Dustdar, S.: A goal-driven approach for deploying self-adaptive IoT systems. In: IEEE International Conference on Software Architecture (ICSA 2020), pp. 1–11. IEEE (2020)
8. Bajrami, X., Murturi, I.: An efficient approach to monitoring environmental conditions using a wireless sensor network and NodeMCU. e & i Elektrotechnik und Informationstechnik, **135**(3), 294–301 (2018)
9. Bouguettaya, A., et al.: A service computing manifesto: the next 10 years. Commun. ACM **60**(4), 64–72 (2017)
10. Shi, W., Cao, J., Zhang, Q., Li, Y., Lanyu, X.: Edge computing: vision and challenges. IEEE Internet of Things J. **3**(5), 637–646 (2016)
11. Avasalcai, C., Murturi, I., Dustdar, S.: Edge and fog: a survey, use cases, and future challenges. Fog Comput. Theory Pract. 43–65 (2020)
12. Dustdar, S., Avasalcai, C., Murturi, I.: Edge and fog computing: vision and research challenges. In: 2019 IEEE International Conference on Service-Oriented System Engineering (SOSE), pp. 96–9609. IEEE (2019)
13. Hussain, M.M., Alam, M.S., Sufyan Beg, M.M.: Fog computing model for evolving smart transportation applications. Fog Edge Comput. Principles Paradigms **22**(4), 347–372 (2019)
14. Ashouri, M., Davidsson, P., Spalazzese, R.: Cloud, edge, or both? towards decision support for designing IoT applications. In: 2018 Fifth International Conference on Internet of Things: Systems, Management and Security, pp. 155–162. IEEE (2018)
15. Dustdar, S., Guo, Y., Satzger, B., Truong, H.-L.: Principles of elastic processes. IEEE Internet Comput. **15**(5), 66–71 (2011)
16. Jesi, G.P., Montresor, A., Babaoglu, O.: Proximity-aware superpeer overlay topologies. In: Keller, A., Martin-Flatin, J.-P. (eds.) SelfMan 2006. LNCS, vol. 3996, pp. 43–57. Springer, Heidelberg (2006). https://doi.org/10.1007/11767886_4