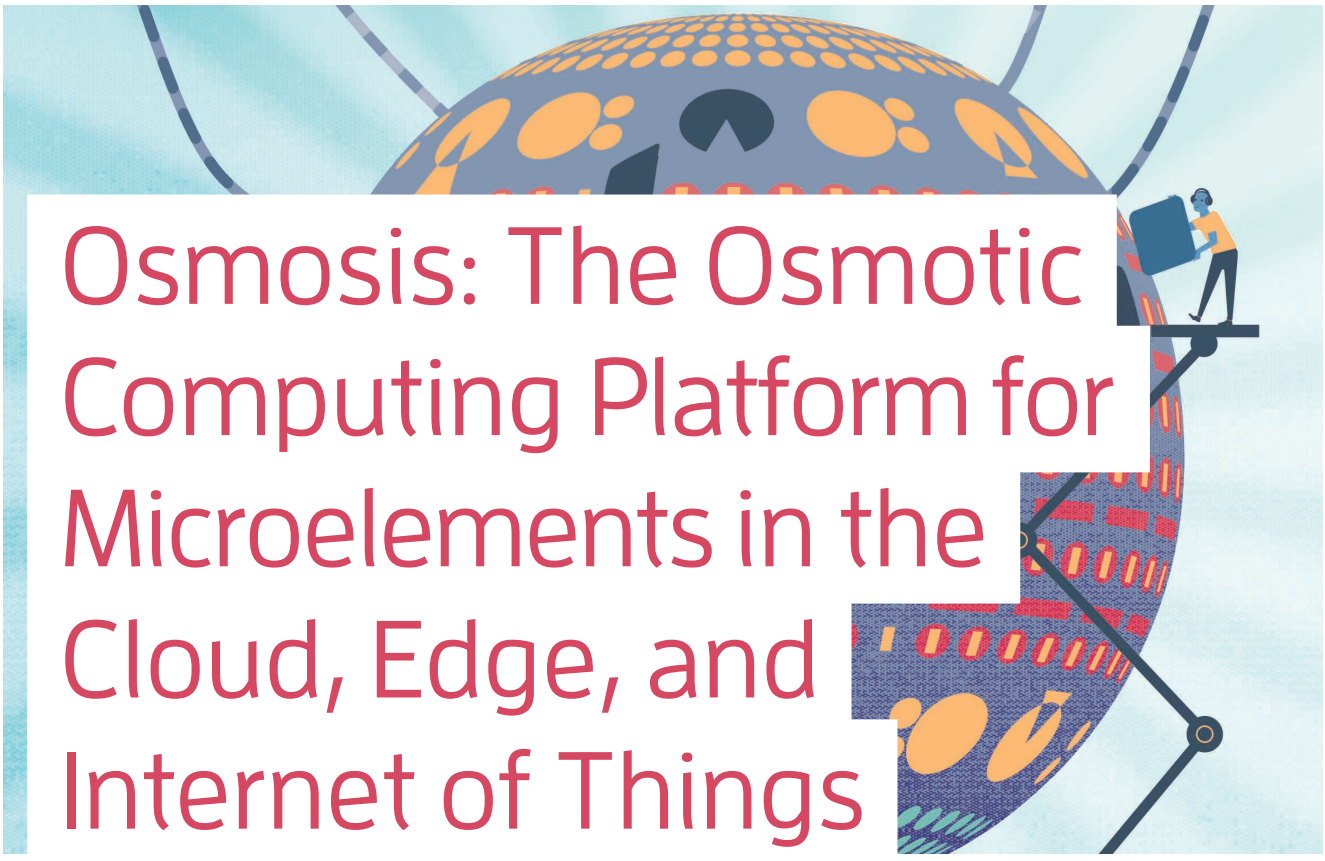

Osmosis: The Osmotic Computing Platform for Microelements in the Cloud, Edge, and Internet of Things

Massimo Villari and Maria Fazio, University of Messina
Schahram Dustdar, Technische Universität Wien
Omer Rana, Cardiff University
Devki Nandan Jha, Newcastle University
Rajiv Ranjan, Chinese University of Geosciences and Newcastle University

Rapid growth and evolution in the number, functionalities, and scope of Internet of Things devices is evident. We present osmotic computing as a new paradigm able to respond to resource heterogeneity, secure data exchange, and efficient microservices deployment across federated cloud systems.

Internet of Things (IoT) and edge devices (such as smart sensors and actuators, smartphones, closed-circuit television cameras, switches, gateways, and routers) monitor events in the cyber and physical worlds, providing the potential to collect raw data and ensuring the timely processing of such data. An intelligent approach is required to reduce the processing cost and network latency while guaranteeing a high degree of security and data privacy. The advantages of integrating different computing paradigms, such as cloud and edge computing, have already been acknowledged by numerous industry- and academia-based initiatives, including the OpenFog Consortium, Amazon Web Services (AWS) (e.g., Snowball Edge and Greengrass), and Cisco.

In this article, we outline a more generic approach that can, conceptually, combine these different industry

Digital Object Identifier 10.1109/MC.2018.2888767
Date of publication: 30 July 2019

0018-9162/19©2019IEEE

initiatives to provide support for IoT–cloud integration. The proposed approach is referred to as *osmotic computing*, and it provides a new computing model that builds upon and extends existing approaches. In particular, the proposed Osmosis platform supports an opportunistic management of IoT microelements (MELs), an abstraction defined and used in this framework to compose and deploy IoT applications across a number of different resource types. Such MELs can also be migrated across different resources within the system.

MOTIVATION

Cloud computing offers both data storage and computation for IoT data processing. However, effectively exploiting cloud technologies for IoT applications can be challenging since the remote management of resources can cause unacceptable delay for latency-sensitive IoT applications. Thus, new computing paradigms are needed, such as edge and fog computing technologies.

A key concern with using computing models to support IoT applications is the management of different physical and virtual infrastructures (e.g., data centers, edge devices, and IoT devices) according to specific application and service requirements (for instance, latency, data volume, responsivity, and processing delays). However, these models address specific application issues and often coexist or need to cooperate. The coexistence of these computing paradigms in the same application scenario can be hard to manage, and it requires additional services to support interoperability and service management. Osmotic computing aims to overcome such constraints by integrating services offered across (close to user) edge devices and cloud systems, supporting a service migration paradigm that enables services to move from a data center to a device close to the user (or the data generation source).

MELs FOR IoT APPLICATIONS

Osmotic computing provides an abstraction referred to as an *MEL*, encapsulating resources, services, and data. In particular, IoT applications can be organized as a graph of MELs [Figure 1(a)] and migrated across different infrastructures, following different triggers (cost, security and privacy, or performance). An MEL encapsulates one of the following categories: 1) microservices (MS), which offer particular functionality and can be easily deployed or migrated, 2) microdata (MD), which represent information flow to and from a sensor or actuator, 3) microcomputing (MC), which executes different type of computational tasks (e.g., statistical analysis, error checking, or machine learning) using a mixture of real-time and historic MD data, or 4) the microactuator (MA), which implements programming interfaces (e.g., for sending some commands) using actuators that alter or control the state of a physical resource at the network edge.

Each IoT application can be decomposed into cooperating subprograms and services to improve deployability and scalability. In osmotic computing, IoT applications are decomposed into a number of interacting MELs, which are atomic entities providing simple functionalities (for example, data collection from a specific sensor, identification of data types or associated semantic annotations, pushing of information toward a remote server, or credential and access control). In general, a graph of MELs can include several MS and MD combined to provide specific behaviors in the reference domain (such as smart building or traffic management). Due to their simple behavior, some of these MELs can be usefully exploited across different contexts through independent instances at the same time. MELs can also be compared with the implementation of specialist functions (e.g., AWS or Azure Functions) that are triggered on the availability of particular events. We assume in this instance that an MEL exists for a longer time frame than such serverless functions.

In Osmosis, virtual components or containers (for instance, Linux Containers, Docker, Preboot Execution Environment, Amazon Elastic Container Service, and Google Kubernetes Engine) executing on cloud or edge resources are used to deploy MELs [Figure 1(b)]. MELs can be implemented using the uPython virtual machine (VM), the uLUA VM, and JavaScript (Node.js), all lightweight alternatives to the hypervisor-based approach (such as Xen, Kernel-based Virtual Machine, and Hyper-V). Only well-defined software components (e.g., database servers) are permitted to be encapsulated in a container, leading to a remarkable reduction in execution time and a high density of instances on a single device as compared to hypervisor-based approaches. MELs may be deployed and orchestrated across both cloud and edge resources [Figure 1(b)].

The Osmosis framework exploits and extends existing container-based approaches for the dynamic deployment and migration of MELs across heterogeneous systems. An open-access MEL repository is needed to support public consumption and collaboration of services. MELs encapsulate simple entities in virtual

environments (e.g., a container). This guarantees the following:

1. *Isolation among MELs*: This offers the opportunity to execute MELs across different applications on the same physical device.
2. *Execution of MELs on heterogeneous infrastructures*: MELs can be deployed and executed on different physical devices, even on resource-constrained devices, depending on specific application scenarios.
3. *Application independence from MEL graph composition and the orchestration engine (OE)*: MELs can be easily applied in any application context. This approach extends workflow engines that can enable the deployment of edge services.

MEL composition can also be supported through existing composition tools, such as Fabric8 (http://fabric8.io), if Kubernetes is being used as an underlying resource management system. With an application represented as a group of multiple self-contained MELs, Osmosis addresses the needs of emerging IoT applications and systems in terms of scalability and dynamicity by developing orchestration techniques and an underlying platform. More generally, the Osmosis orchestrator enables the enactment of MELs across heterogeneous infrastructures. Unlike traditional cloud orchestrators, the Osmosis orchestrator merges activities and information at two logical layers: the application

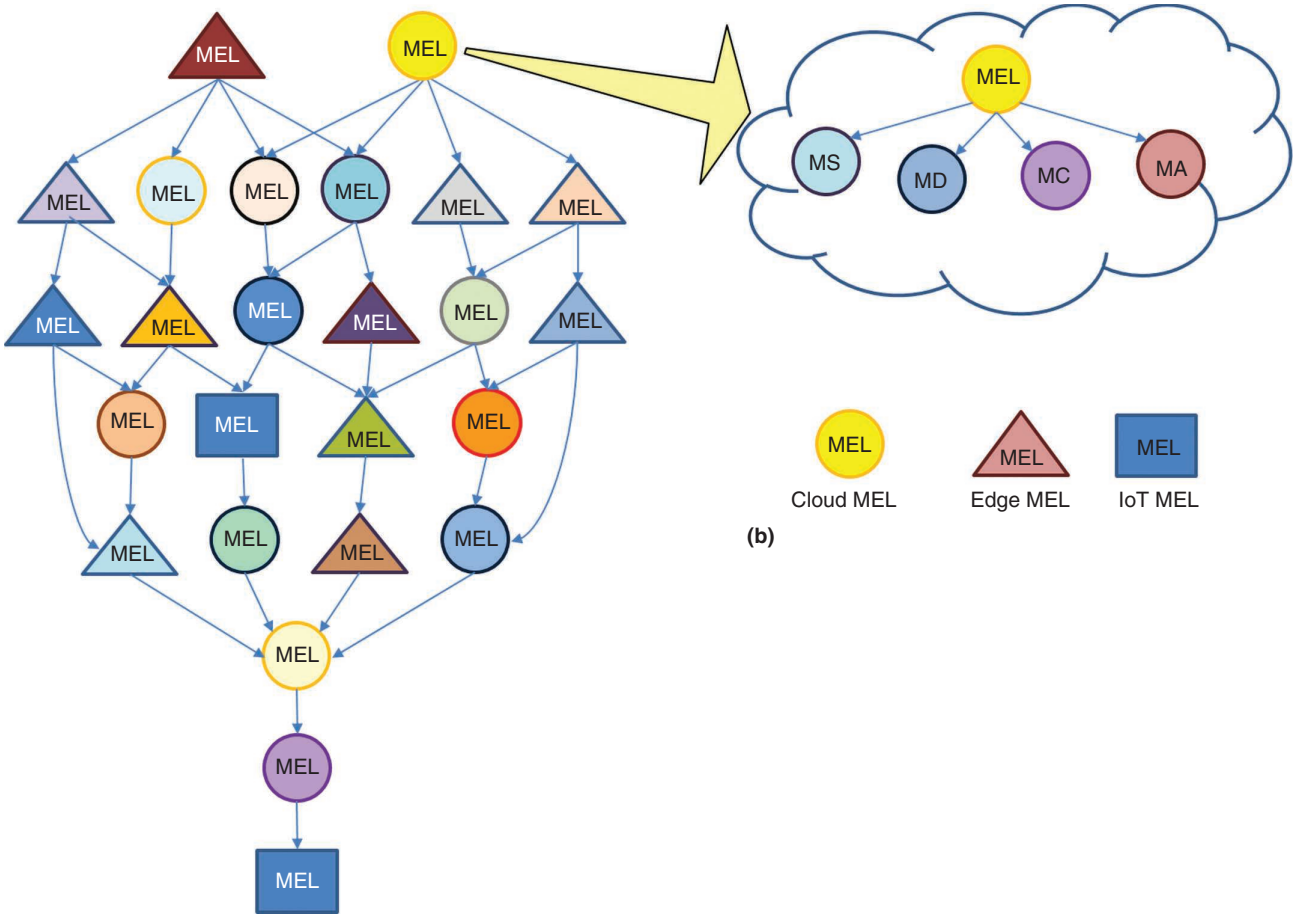

FIGURE 1. A tree representation for MELs. (a) A graphical representation of MELs. (b) Cloud or edge resources used to deploy MELs.

layer and the infrastructure layer. This means that new functionalities in both monitoring activities and scheduling algorithms need to be investigated. MEL deployment, therefore, must take into account not only resource availability to execute single MELs with a specific quality-of-service (QoS) level but also the performance of the entire application or service. Thus, the orchestrator has to monitor the infrastructure's MS execution to detect issues on physical hardware, software and network resources, and the performance of the application or service to guarantee the promised service level agreement (SLA). In the scheduling of MEL deployment and migration, based on dependencies identified in the MEL graph, it is necessary to manage virtual networking among MELs and identify highly coupled MELs that need to be deployed according to specific application and service constraints. For example, some MELs need to be in the proximity of a user to support real-time interactions and data sharing.

Osmosis leads to a software-defined federated ecosystem supporting different stakeholders working at the IoT, edge, and cloud layers. The adoption of network function virtualization (NFV) and software function chaining (SFC) solutions (such as HP OpenFlow and middlebox technologies) enables MELs deployed on edge and cloud data centers to be interconnected to provide a secure service. NFV and SFC, due to their flexibility, are adopted as new software-defined network management services in Osmosis to mitigate threats observed in network management. Moreover, additional security-oriented solutions intrinsic to the network function software—for example, centralized security management and hypervisor introspection—are considered. Osmosis could benefit other IoT application domains (such as through the use of public IFTTT.com interfaces to Google, Twitter, and Fitbit). It is also able to demonstrate how a holistic IoT application management platform can be supported by federating heterogeneous computing systems and infrastructures. Inspired by the growth of the Docker and Kubernetes communities, the MEL repository and container-based approach are used to demonstrate the potential for IoT providers to develop and advertise sophisticated container-hosted MELs.

Figure S1 in "Osmotic Computing" shows how MELs are deployed in embedded devices at different layers in the osmotic computing environment.[1] To manage flows of data and composed services, MEL orchestration must leverage networking functionalities. For example, it can make use of the open source framework Open Virtual Network, originally launched by the Open vSwitch team at Nicira (currently part of VMware), which can be usefully adapted to manage the abstraction of MEL networks and data-, device-, and network-centric security features, thanks to its high flexibility in configuration isolation capability.

A COMPARISON OF OSMOTIC COMPUTING WITH DISTRIBUTED COMPUTING MODELS

Osmotic computing[1] is an innovative paradigm that merges and extends several distributed computing technologies. In this section, we discuss the main advantages of using this new paradigm, comparing it with well-known distributed computing models. On-demand computing and storage resources and services are provided by *cloud computing*. Osmotic computing extends cloud resources over edge and IoT infrastructures and devices, providing seamless service. This is achieved through the use of the MEL abstraction that introduces microentities deployable on heterogeneous and resource-constrained systems.

By providing additional resources when required, *elastic computing* supports resource expansion and contraction to increase productivity. Resources can be easily scaled up or out, depending on the runtime resource requirements, without creating any disruption to services. Both cloud and osmotic computing implement elastic management of resources but take care of both low-level metrics (such as resource availability, load balancing, and QoS) and application requirements (for example, application and service response time and the SLA). Although cloud computing focuses on resource elasticity within a data center, osmotic computing attempts to support elasticity by also including resources at the network edge.

Edge computing manages constrained resources close to end users and the physical world. In contrast, osmotic computing extends edge resources throughout the cloud data centers. The Osmosis orchestrator effectively manages the different computing resources. The edge computing paradigm is improved by *mobile edge computing*, or *multiaccess edge computing* (*MEC*), which supports mobile end users and embraces different access technologies for mobile devices (not only cellular networks). Osmotic computing naturally supports MEC due to its dynamic management of resources and services in runtime and exploitation of heterogeneous computing and networking technologies.

OSMOTIC COMPUTING

Borrowed from chemistry, the term *osmosis* represents the seamless diffusion of microelements (MELs) across heterogeneous infrastructures. To better describe the osmotic management of computing, we logically subdivide computing environments into three main layers (see Figure S1): layer 1 (L1), cloud data centers; layer 2 (L2), edge systems and microdata centers; and layer 3 (L3), Internet of Things (IoT) devices. At L3, various IoT devices, such as smartphones (with human input), sensors, and smart wearables, capture raw data from the environment with some defined frequency or, if specific events occur, based on the device's data collection and storage capacity along with the particular requirements of the system that need to be fulfilled. Recently, different standards have been proposed for L3, for example, the Constrained Application Protocol supported through specialist operating systems, such as the Riot operating system or Contiki, and the REST engine, such as Erbium. Multiple gateways are used by L2, which is commonly implemented, using routers and network switches (supported by OpenFlow protocols) or network processor–based hardware that enables network components to be accessed and managed remotely. Moreover, data from different L3 sensors can also be collected and aggregated by these gateways. Finally, L1 consists of large computing clusters, where storage and computational capacity is provided to application users, thus enabling complex, generally time-consuming operations to be performed. L2 devices can support the collection of raw data from numerous L3 devices and perform various computations (such as simple stream operations, for example, min, max, average, aggregation, and filtering, on a small data time frame), encryption, encoding, or transcoding operations on the incoming data stream before transferring these data to L1 for further analysis. Therefore, L2 devices not only retrieve data from L3 devices but also perform some fundamental analysis.

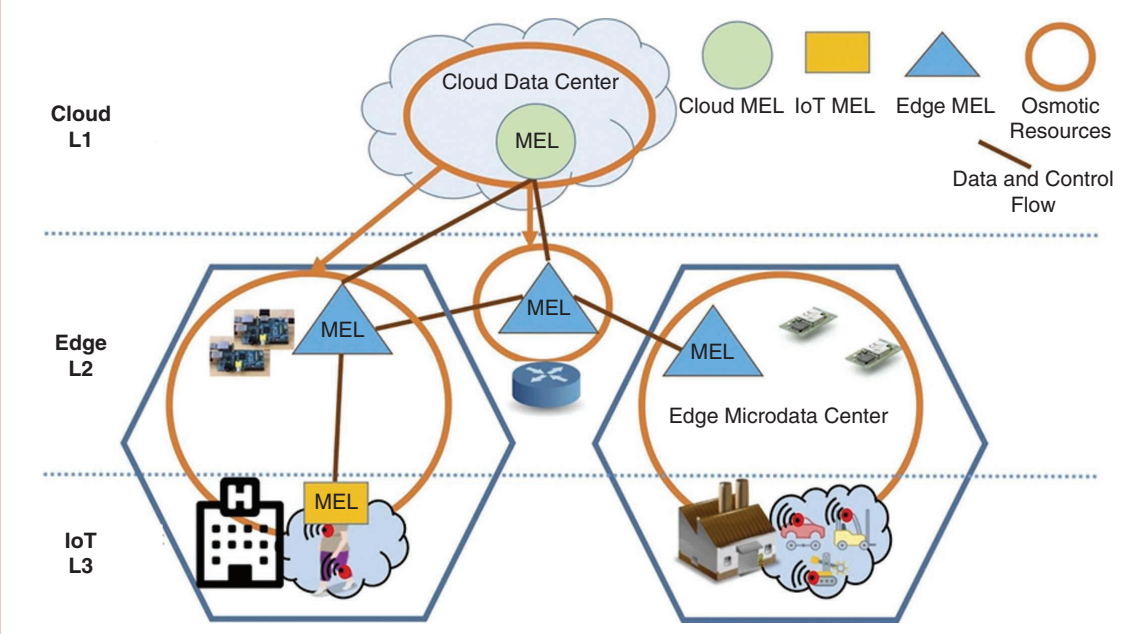

FIGURE S1. The osmotic computing model. To better describe the osmotic management of computing, the computing environments are divided into three main layers: layer 1 (L1), cloud data centers; layer 2 (L2), edge systems and microdata centers; and layer 3 (L3), IoT devices.

Each type of infrastructure (cloud and edge) also has specific objective functions that influence different types of operations. For example, edge (L2) devices are generally resource constrained (with limited battery power, network range, etc.). Therefore, operations must be executed keeping these constraints in mind. Thus, multiple concurrent dataflows from the L3 layer share the storage and computation capability of the edge device, limiting analysis to the particular time constraints and number of flows. Cloud (L1) operations are based on predefined service level agreements between a cloud service provider and a client, for example, response time, throughput, availability, and cost. Understanding the interaction and coordination between the cloud (L1) with the IoT (L3) and edge (L2) for deployment of an application is a key research challenge, especially for real-time, stream-processing applications. The MEL can be allocated across edge and cloud resources based on the privacy and security and quality-of-service (QoS) constraints. This distribution of data analysis can support performance improvement, reducing overall core network load. Based on our proposed osmotic concept, one MEL can be implemented across different resource types and data centers having different levels of complexity.

Osmotic computing extends elastic resource management because infrastructure (for example, availability and load balancing) and applications (for example, sensing or actuation capability and user proximity) requirements affect deployment and migration strategies and can change over time (see Figure S2). Osmotic functionalities automatically configure or reconfigure the movement and deployment of MELs in response to QoS, security, and runtime perturbations (see Figure S2). With our proposed abstraction, it is possible to decouple infrastructure management from application deployment issues and make the flow of MELs from edge to cloud and vice versa possible. For example, consider a situation with numerous IoT devices at L3 collecting and generating large data volumes, the transfer of which to a cloud platform (L1) can consume significant bandwidth. If this scenario is detected, the osmotic computing orchestrator shifts some of the data processing to the edge (L2), leading to increased flexibility and system resilience. Osmosis offers a great opportunity to adopt FIWARE and its related generic enablers in a new context, such as edge computing and MELs.

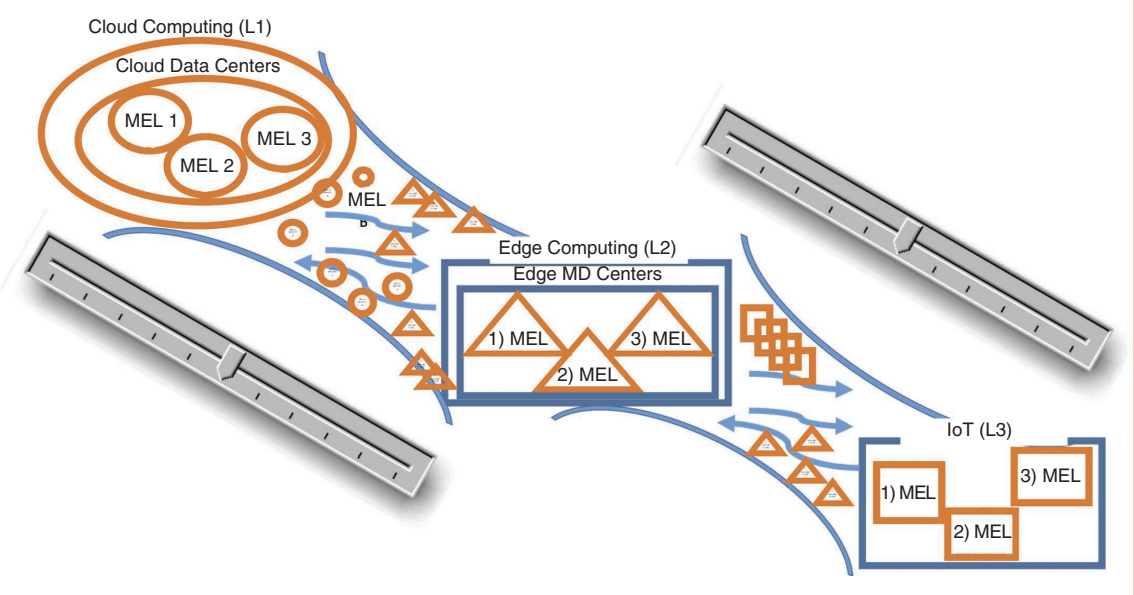

FIGURE S2. MELs' osmotic movement across the cloud, edge, and IoT.

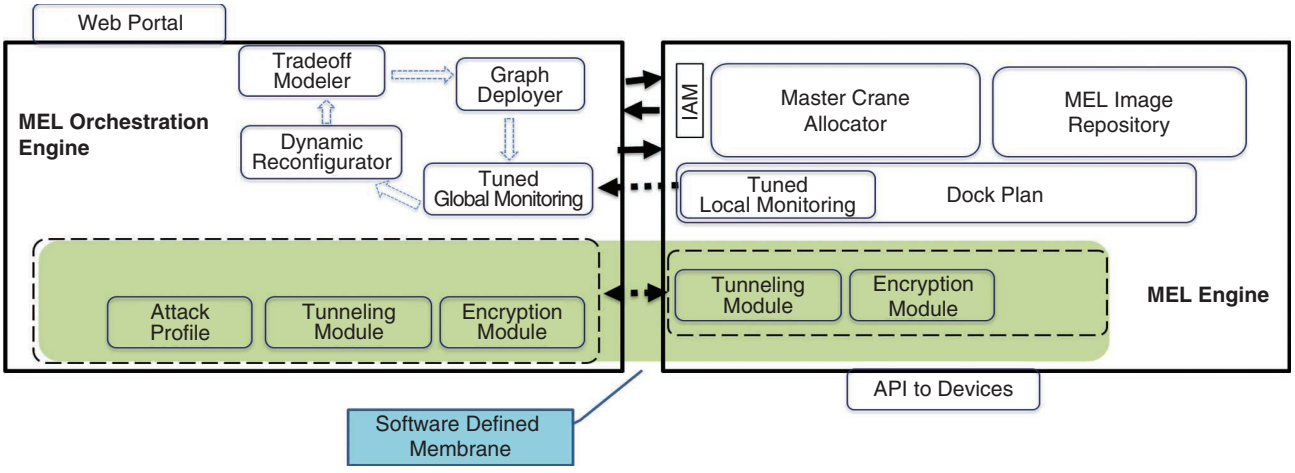

FIGURE 2. The Osmosis internal architecture: the MEL orchestrator engine and MEL engine. API: application programming interface; IAM: identity and access management.

Low-latency IoT applications are supported by *fog computing*, a distributed computing environment that connects edge and cloud resources. Fog computing focuses attention on IoT applications and their efficiency, performing real-time analysis of the data generated by these IoT objects and devices. In osmotic computing, IoT applications are abstracted and composed into connected MELs at a first stage and then deployed on available resources at the edge and in the cloud (and even in IoT devices, if possible). During the runtime stage of applications, the Osmosis orchestrator monitors physical and virtual resources, MEL behaviors, and IoT application performance to provide a dynamic and continuous reconfiguration of the whole system. Such reconfigurations can imply the movement of MELs through the system but can be performed at low cost in terms of overhead and delay and also with high flexibility.

OSMOSIS ARCHITECTURE

The Osmosis platform spawns MELs on cloud and edge devices, with continuous monitoring supporting feedback-driven orchestration during runtime. The main challenges that Osmosis addresses in this context can be summarized as follows:

- creating a new way to describe MELs for all cloud and edge domains, making it necessary to determine how to specify an *execution environment*—in particular, where an MEL can be hosted
- setting up an easy-to-use environment in which MELs are intrinsically fully equipped with security and able to evaluate the behavior of MELs during their executions and guarantee security and privacy
- understanding how MELs behave with external interactions and adjusting them accordingly, releasing new versions and new configurations.
- understanding how to wire virtual networks among MELs, given security constraints.

Figure 2 shows the two main parts of the Osmosis reference architecture: the MEL engine (ME) and the OE.

ME engine

The ME controls deployment of MELs across the edge and cloud infrastructure, supporting an execution environment that is configurable and adaptive and also independent of any underlying infrastructure. A pluggable infrastructure abstraction associated with the MEL deployment environment transparently supports and manages heterogeneous MEL deployment mechanisms (VM and container based). The ME includes the MEL image repository, which provides the basis for storing images, along with profiling and versioning data that use predefined interfaces to constantly monitor and detect the underlying runtime QoS anomaly status of resources (for example, disk or memory consumption) and security attacks. The versioning system allows the recovery of MEL instances after failure or security attacks. Based on this profiling information, the OE's tuned global monitoring module detects failure or derives and learns load patterns for dynamic reconfiguration. The master crane allocator module of the ME

is connected to the OE to push commands to and pull data from the OE. It establishes a secure connection with other modules and components. The dock plane is in charge of creating, executing, and monitoring MELs in interoperable ways. To aggregate data that need to be delivered to global monitoring, the tuned local monitoring module is used.

MEL OE

The MEL OE implements the following four modules: the tradeoff modeler, graph deployer, tuned local monitoring, and dynamic reconfigurator. The tradeoff modeler simplifies resource and device configuration selection across level 1 (L1), level 2 (L2), and level 3 (L3) by implementing novel tradeoff optimization models, fusing multicriteria optimization techniques with multicriteria decision-making techniques. The graph deployer implements algorithms to characterize the runtime performance of MELs deployed across heterogeneous infrastructures (e.g., L1 versus L2 versus L3).

The tuned local monitoring module is used for the orchestrator and supports the continuous monitoring and collection of the dataflow anomaly and security attack information from connected resources at L1, L2, and L3. Moreover, it offers techniques to manage collected data, events, and faults or failures. Using gathered logs and information about the underlying infrastructures, this module is capable of evaluating performance and fault analyses based on techniques, such as Bayesian networks. Furthermore, the dynamic reconfigurator uses the detailed analysis information for adapting MEL topologies so that it can respond to the defined requirements in terms of privacy, security, QoS, and so on.

The dynamic reconfigurator supports techniques, such as a mixture density network (a combined structure of neural networks and mixture models), for predicting the QoS of deployed MELs across L1, L2, and L3. The MEL OE offers an application programming interface (API) to interact with distributed modules of the framework for orchestrating MELs across L1, L2, and L3 infrastructure. Figure S3 in "How the Osmosis Orchestrator Differs From Traditional Cloud Orchestrators" captures the main architectural components of Osmosis and in addition considers a federated scenario, where several cloud, edge, and IoT providers cooperate to deploy applications and services.

The MEL engine accomplishes the adaptive deployment of MS within edge and cloud data centers and the runtime performance monitoring of containers in execution. The deployment task provides a customizable and adaptable environment without having any dependency on the underlying infrastructure. To manage and support different MS deployment operations, such as container- and VM-based deployments, an abstract, pluggable infrastructure is incorporated by the execution environment.

The OE runs within a cloud data center and implements selection, composition, deployment, high-level monitoring, and runtime management techniques. Continuous resource monitoring and data collection are supported by the OE using the monitoring framework of the underlying infrastructure or some software capabilities that collect different performance metrics (for example, tailored monitoring profilers for cloud and edge containers). In a federated scenario, each cloud provider implements its own OE (e.g., the providers of clouds A, B, and C in Figure S3), and the OEs cooperate for the osmotic management of MELs over the whole system.

ADVANCES BROUGHT BY OSMOSIS IN THE STATE OF THE ART

In this section, we summarize the key advances as compared with related work in each core topic relevant to the Osmosis framework, highlighting issues to be addressed and our proposed approaches to solve them.

Osmotic MELs deployment

Container-based virtualization is a useful alternative to VMs when implementing cloud-based solutions in IoT and edge devices. The application with its dependencies is encapsulated within a virtual container by some container engine, such as Docker, to provide resource control and isolation. This makes containers an appealing solution for deploying MELs. Rao et al.[3] present a system model for developing applications that process raw data gathered by IoT devices, leading to a processing model that runs on the cloud, whereas IoT devices are exclusively engaged in collecting sensed data. A container evaluation is given within the context of the edge environment by Lee and Pahl.[4] Multitier web application design is also enabled by containers because multitenancy can be built into cloud-enabled services, allowing the reengineering and refactoring of the traditional codebase system using container-based MS. Some solutions in the literature provide MS orchestration frameworks—for example, Docker Swarm supports a native orchestration framework for multi-Docker deployment environments, converting a collection

HOW THE OSMOSIS ORCHESTRATOR DIFFERS FROM TRADITIONAL CLOUD ORCHESTRATORS

A cloud orchestrator is a software component of the cloud management system that evaluates workloads on the cloud infrastructure and automates the deployment of services. In this regard, it continuously monitors hardware and software resource availability and services, and it schedules all activities for the efficient management of resources according to well-defined policies. Thus, the orchestrator allows the cloud system to ensure the desired quality of service (QoS) to detect possible failures or capacity shortage.

In osmotic computing, the orchestrator functionalities need to be extended, integrating new capacities in both monitoring activities and scheduling algorithms. Indeed, in this computing model, microelements (MELs) holding microdata or microservices (MS) are composed and networked to provide full applications and services. The deployment of such MELs must take into account not only resource availability to execute single MELs with a specific QoS level but also the performance of the entire application and service. Thus, the orchestrator must monitor the execution of MS in the infrastructure to detect issues on physical hardware, software, and network resources and monitor the performance of the application and service to guarantee an agreed service level agreement.

In scheduling MEL deployment and migration, because of the MEL graph composition, it is necessary to manage virtual networking among MELs but also to identify highly coupled MELs that need to be deployed according to specific application and service constraints (for example, some MELs need to be close for real-time interactions). In general, we can say that, in contrast to traditional cloud orchestrators, an osmotic orchestrator works to merge activities and information at two logical layers: the infrastructure layer and the application layer.

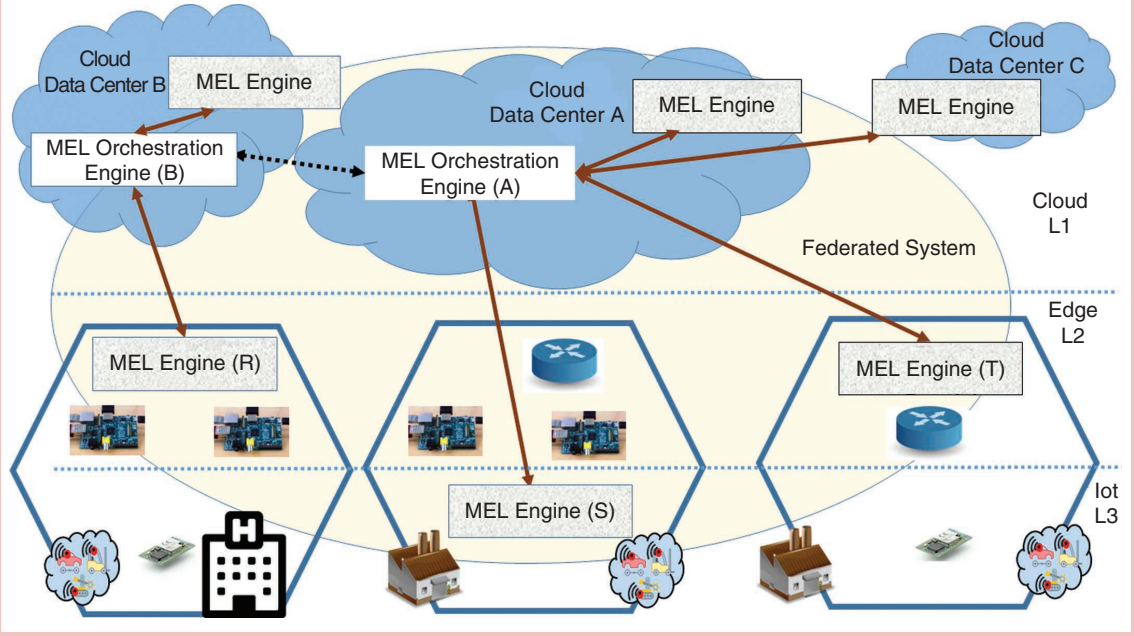

FIGURE S3. The deployment of Osmosis architectural components at different layers.

of Docker hosts into one large virtual Docker host. Kubernetes is an open source framework that automates the deployment, operations, and management of applications running in containers. It provides horizontal scaling of applications, implemented using manual or automatic methods based on the CPU usage (while ignoring QoS constraints). The Osmosis framework provides an MEL engine that allows users and developers to deploy containers running MELs on edge devices, enabling customization and migration of MELs. Software adaptation and versioning mechanisms will allow edge and cloud resource providers to deploy MELs across a heterogeneous pool of physical devices.

IoT computing

Significant interest from the academic community has been seen in the area of the IoT, as is indicated by recent efforts, such as IoTCloud or the European OpenIoT, an open source middleware-oriented framework providing sensors as a service. In parallel contexts, strong commercial interest in applications ranging from smart homes to smart traffic is indicated by REST- and HTTP-based APIs, such as Xively, ThinkSpeak, and OpenSense. There have also been attempts to develop an interoperability specification for the IoT, which, according to the National Institute of Standards and Technology, are "smart systems that include engineered interacting networks of physical and computational components."[13] Recent attempts by the IEEE to define an architectural framework for IoT, such as P2413,[14] indicate that current standardization activities are limited to specific areas and represent disjointed or sometimes redundant developments. The P2413 framework promotes functional compatibility and aids system interoperability and cross-domain interaction.

Another significant approach[5] to mobile offloading is focused toward the offloading of complex and time-consuming tasks from resource-constrained mobile devices to resource-abundant cloud data centers. To increase battery operation hours and reduce potential application overhead due to intermittent network connection, tasks are offloaded from mobile devices (considered to have lower computing and storage capability compared with a cloud data center) to cloud data centers, with regular synchronization between the cloud data center and the edge device. Similarly, a different but related approach gaining favor is MEC, which has primarily been driven by significant advances in 5G networks and the ability to support user-provisioned services within the network. MEC is driven by similar requirements for latency-sensitive service provisioning as well as the ability to offer application management and service orchestration at the network edge. Recent research conducted in the realm of MEC occurred in the context of mobile computing, where smartphones act as both IoT devices and gateways. However, most existing MEC approaches focus on infrastructure-level QoS constraints, such as energy minimization or resource utilization, while giving very little attention to the more complex and interdependent QoS requirements across MELs that must be choreographed and orchestrated in a coordinated manner to realize an IoT application. Our approach reverses the emphasis on mobile offloading: we transfer a cloud-intended computation to a resource-constrained mobile device. This reverse offloading facilitates computation closer to the activity being measured (data transfer costs and latency). Therefore, the Osmosis framework is focused on understanding the MEL types that would be better executed on the edge and IoT rather than on a cloud. Our work complements MEC because the Osmosis framework approach advocated here focuses on migration of MELs across MEC levels and investigates techniques that can be used to support and facilitate such migration based on application QoS and security requirements.

QoS and security tradeoff optimization across the IoT, edge, and cloud

Mapping an application (represented by a graph of MELs) requires the selection of customized cloud or edge resources or both. However, existing orchestration platforms (such as Kubernetes, Docker Swarm, Amazon EC2 Container Service, and OpenShift Origin) are not capable of handling conflicting QoS, privacy, and security requirements while undertaking container-mapping decisions. Various optimization and performance measurement techniques support selecting VM configurations using numerous QoS requirements (for instance, cost, throughput, latency, availability, reputation, etc.). However, the QoS constraints, configuration environments, and security and privacy requirements for mapping MELs to the edge (L2) are fundamentally different from choosing VM configurations on a cloud (L1). Given a set of conflicting QoS, security, and privacy requirements, as well as the configuration search space, existing deterministic optimization techniques, such as linear and mixed-integer nonlinear programming, although successful in optimizing web service composition, would not be effective for computing the optimal solution when mapping a

graph of MELs in the osmotic computing (L1, L2, and L3) environments.

The Osmosis framework requires multilevel (L1 versus L2 versus L3) optimization models based on fusing multicriteria optimization and decision-making techniques. A unique set of computationally tractable tradeoff optimization models is needed using cross-cutting concerns as parameters, such as resource configuration, QoS, and security and privacy requirements. The optimization models can be used to generate a Pareto-optimal curve whose points will represent a nondominated optimal configuration set (no other configuration set has higher utility without compromising one of the requirements in terms of QoS and security and privacy) for mapping an MEL either to the edge or to the cloud or to both.

benchmark kernels that are relevant for particular parts of the infrastructure and application classes (such as stream or real time versus batch). For instance, the following benchmark (kernels) may be considered:

- *Edge layer*: TPCx-IoT (for data aggregation, real-time analytics, and persistent storage), Google ROADEF, and Linear Road benchmarks (for stream processing).
- *Cloud layer*: TeraGen, TeraSort, TeraValidate, and BigDataBench (for batch-oriented processing)

However, creating a benchmark that can aggregate processing capability across these two layers and, more importantly, identify performance dependencies between these layers in a holistic manner remains a challenge.

OUR RESEARCH OUTCOMES WILL SIGNIFICANTLY IMPROVE IoT RECONFIGURATION AND RESILIENCE CAPABILITIES TO DEAL WITH RUNTIME UNCERTAINTIES.

MEL graph performance characterization

Performance characterization is undertaken by evaluating the relative performance of a computing resource (for example, processor capability, memory, storage, and network) via benchmarking. However, existing performance characterization research in the cloud context (L1) is limited for the IoT (L3) and edge (L2). Understanding performance bottlenecks of MELs within an IoT or cloud system remains a challenge, and it is useful to identify

Holistic QoS and dataflow-anomaly monitoring

Much of the difficulty in QoS and dataflow-anomaly monitoring (and detection) in Osmosis originates from the complexity of MELs and the integration of different computing environments. Monitoring tools—such as the Relational Grid Monitoring Architecture, Hawkeye, and CloudWatch—are concerned with monitoring QoS metrics related only to cloud resources, such as CPU percentage, TCP/IP performance, and available nonpaged memory. Similarly, cluster-monitoring tools, such as SequenceIQ, Ganglia, DMon, Sematext, Apache Chukwa, and Nagios, provide QoS parameter (CPU-, disk-, or network-bound, or memory) information of the virtualized resources belonging to private or public clouds. These tools are incompatible with monitoring any anomalies in QoS and dataflow in the osmotic computing environment.

QoS monitoring has been investigated extensively in the context of the cloud. Emeakaroha et al.[6] explore an approach to tackle QoS anomalies at the application level, but the computing model is fundamentally different from that of osmotic computing. A framework to monitor QoS anomalies for content-based routing and complex event-processing applications hosted on the cloud is presented by Romano et al.[7] To understand the behavior of cloud-hosted web and database applications, Moldovan et al.[8] propose analyzing consistent relationships among monitored metrics belonging to different software and hardware components; Copil et al.[9] suggest monitoring and analyzing cloud-hosted application behavior in terms of topology and software- and hardware-level QoS information. A monitoring platform that automatically improves the QoS of multicloud-hosted applications, MODAClouds is endorsed by Di Nitto et al.[10] Leitner et al.[11] describe an approach to monitor high-level QoS parameters of complex cloud-hosted event processing applications.

In summary, no existing QoS anomaly-monitoring tools and techniques can ubiquitously monitor QoS and dataflow across MELs mapped to an osmotic infrastructure or detect root causes of QoS and dataflow anomalies across the osmotic infrastructure. In Osmosis, a

monitoring technique is needed to give users insights into how their MELs are performing, where possible QoS bottlenecks may occur, and what the potential is for security threats.

Dynamic reconfiguration

In the Osmosis infrastructure, dynamic reconfiguration of MELs can lead to the following challenges: 1) estimating the behavior of MEL-specific dataflow in terms of analyzed data volume and potential input–output behavior and 2) making decisions about the types and scale of edge or cloud resources that should be provisioned across MELs without any knowledge about the runtime changes to the dataflow.

Edge orchestration tools, such as Kubernetes, Docker Swarm (based on the sFlow-RT monitoring engine), and OpenShift Origin, offer a container reconfiguration capability, scaling by monitoring CPU usage and network traffic. Apache Oozie, Quincy, Omega, Mercury, and LinkedIn's Azkaban support provisioning of Hadoop data-processing graphs on the cloud but in a restrictive manner; it works well only for MapReduce. Other big data application orchestration platforms, such as AWS IoT, Google's Cloud Dataflow, Mesos, and YARN, support manual reconfiguration of VM- and container-based data analytics graphs on cloud resources. Substantial theoretical work[12] on dynamic reconfiguration of traditional multitier web applications on the cloud has been undertaken. In Osmosis, we extend the traditional notion of dynamic reconfiguration, which only considers application components hosted on cloud resources, to application components (MELs) that can be deployed across edge and cloud resources. Our research outcomes will significantly improve IoT reconfiguration and resilience capabilities to deal with runtime uncertainties (for example, changing data volume and velocity or runtime failures). A degree of confidence will be associated with each MEL-specific prediction model. Along with QoS and security and privacy requirements, this will allow an application owner to assess risks and opportunities and plan the level of osmotic capacity needed to maximize

ABOUT THE AUTHORS

MASSIMO VILLARI is an associate professor of computer science at the University of Messina, Italy. His research interests include cloud computing, distributed systems, wireless networks, security systems, big data analytics, and the Internet of Things. Villari received a Ph.D. in computer engineering from the University of Messina, Italy. Contact him at mvillari@unime.it.

MARIA FAZIO is an assistant researcher in computer science at the University of Messina, Italy. Her research interests include distributed systems and wireless communications. Fazio received a Ph.D. in advanced technologies for information engineering from the University of Messina, Italy. Contact her at mfazio@unime.it.

SCHAHRAM DUSTDAR is a full professor of computer science heading the Distributed Systems Research Division at Technische Universität Wien, Austria. His research interests include the Internet of Things and edge computing. Dustdar received a Habilitation degree in computer science from Technische Universität Wien. He is an IEEE Fellow, an Association for Computing Machinery Distinguished Scientist, a member of the Academia Europaea—The Academy of Europe, and a member of the *Computer* Editorial Board. Contact him at dustdar@dsg.tuwien.ac.at.

OMER RANA is a full professor of performance engineering in the School of Computer Science and Informatics at Cardiff University, Scotland, where he also leads the Internet of Things laboratory. Rana received a Ph.D. from Imperial College of Science, Technology and Medicine, United Kingdom. Contact him at o.f.rana@cs.cardiff.ac.uk.

DEVKI NANDAN JHA is a Ph.D. student in the School of Computing at Newcastle University, United Kingdom. Jha received an M.Tech. in computer science and technology from Jawaharlal Nehru University, India. Contact him at d.n.jha2@ncl.ac.uk.

RAJIV RANJAN is full professor in the School of Computing at Newcastle University, United Kingdom, and a chair professor in the School of Computer at the China University of Geosciences. Ranjan received a Ph.D. in computer science and software engineering from the University of Melbourne, Australia. Contact him at raj.ranjan@ncl.ac.uk.

the value of an application for end users. Osmosis will also develop techniques to dynamically fine-tune performance prediction models based on real-time information (for example, QoS anomalies, security anomalies, and failures) available from the monitoring and anomaly detection engine. Fine-tuned analysis based on MEL-specific performance-prediction models can be used by the orchestrator to adapt MEL topologies' reaction react to defined QoS and security constraints. These performance prediction models will form the basis for the next generation of dependable osmotic orchestration algorithms and platforms, with complete understanding of dataflow and cloud, IoT, and edge resource demands, thereby improving resilience to uncertainties.

In this article, we presented the benefits and limitations of osmotic computing, supporting the deployment of applications and services over heterogeneous distributed infrastructures. We described how the use of osmotic computing can advance the current state of the art within the IoT–edge–cloud continuum. The Investigation of new technologies and solutions for osmotic computing will bring innovation in several research areas. Osmotic computing also represents the integration of a number of technologies and does not necessarily need to be developed as a separate platform (as outlined in this article). We distinguished between 1) a conceptual model and 2) an architectural realization of an osmotic computing model. The conceptual model has a number of potential mechanisms that enable realization using existing technologies.

REFERENCES

1. M. Villari, M. Fazio, S. Dustdar, O. Rana, and R. Ranjan, "Osmotic computing: A new paradigm for edge/cloud integration," *IEEE Cloud Computing*, vol. 3, no. 6, pp. 76–83, 2016. doi: 10.1109/MCC.2016.124.
2. R. Khalaf, A. Slominski, and V. Muthusamy, "Building a multi-tenant cloud service from legacy code with docker containers," in *Proc. 2015 IEEE Int. Conf. Cloud Eng.*, pp. 394–396.
3. B. B. P. Rao, P. Saluia, N. Sharma, A. Mittal, and S. V. Sharma, "Cloud computing for Internet of Things and sensing based applications," in *Proc. 2012 6th Int. Conf. Sensing Technol.*, pp. 374–380.
4. B. Lee and C. Pahl, "Containers and clusters for edge cloud architectures—A technology review," in *Proc. 2015 3rd Int. Conf. Future Internet of Things and Cloud*, pp. 379–386.
5. S. Abolfazli, Z. Sanaei, E. Ahmed, A. Gani, and R. Buyya, "Cloud-based augmentation for mobile devices: Motivation, taxonomies, and open challenges," *IEEE Commun. Surveys Tut.*, vol. 16, no. 1, pp. 337–368, 2014.
6. V. C. Emeakaroha, T. C. Ferreto, M. A. S. Netto, I. Brandic, and C. A. F. De Rose, "CASViD: Application level monitoring for SLA violation detection in clouds," in *Proc. 2012 IEEE 36th Annu. Comput. Software and Appl. Conf.*, pp. 499–508.
7. L. Romano, D. D. Mari, Z. Jerzak, and C. Fetzer, "A novel approach to QoS monitoring in the cloud," in *Proc. 2011 1st Int. Conf. Data Compression, Commun. and Process.*, pp. 45–51.
8. D. Moldovan, G. Copil, H. L. Truong, and S. Dustdar, "On analyzing elasticity relationships of cloud services," in *Proc. 2014 IEEE 6th Int. Conf. Cloud Computing Technol. and Sci.*, pp. 447–454.
9. G. Copil et al., "ADVISE—A framework for evaluating cloud service elasticity behavior," in *Proc. 11th Int. Conf. Service-Oriented Computing*, 2014, pp. 275–290.
10. E. Di Nitto et al., "Supporting the development and operation of multi-cloud applications: The MODAClouds approach," in *Proc. Int. Symp. Symbolic and Numeric Algorithms Scientific Computing*, 2013, pp. 417–423.
11. P. Leitner, C. Inzinger, W. Hummer, B. Satzger, and S. Dustdar. "Application-level performance monitoring of cloud services based on the complex event processing paradigm," in *Proc. Int. Conf. Service-Oriented Computing and Appl.*, 2012, pp. 1–8.
12. D. Ardagna, G. Casale, M. Ciavotta, J. F. Pérez, and W. Wang, "Quality-of-service in cloud computing: Modeling techniques and their applications," *J. Internet Services Appl.*, vol. 5, no. 1, pp. 11, 2014.
13. U.S. Department of Commerce, "Welcome to the NIST cyber-physical systems website," NIST Engineering Laboratory. Accessed on: Mar. 8, 2019. [Online]. Available: https://www.nist.gov/el/cyber-physical-systems
14. IEEE Standards Association, "Standard for an architectural framework for the Internet of Things (IoT)," IEEE. Accessed on: Mar. 8, 2019. [Online]. Available: http://grouper.ieee.org/groups/2413/