Open Issues in Scheduling Microservices in the Cloud

Maria Fazio and Antonio Celesti
University of Messina

Rajiv Ranjan
Newcastle University

Lydia Chen
IBM Research

Chang Liu
Newcastle University

Massimo Villari
University of Messina

The adoption of container-based microservice architectures is revolutionizing application design. By adopting a microservice architecture, developers can engineer applications that are composed of multiple lightweight, self-contained, and portable runtime components deployed across a large number of geodistributed servers.

A microservices-based cloud application involves the interoperation of multiple microservices, each developed separately, that can be deployed, updated, and redeployed independently without compromising the application's ecosystem's integrity. The ability to independently update and redeploy the code base of one or more microservices increases applications' scalability, portability, updatability, and availability, but at the cost of expensive remote calls (instead of in-process calls) and increased overhead for cross-component synchronization.

The microservices approach is in contrast to the traditional "monolithic" development of applications, where each application is a single, autonomous unit. For example, in a client-server application, the server is a monolithic entity that handles HTTP requests, executes logic, and retrieves or updates its data. The problem with such monolithic architectures is that even a small modification of the application's logic requires the deployment of a new running version of the entire code base. A microservice architecture is lightweight and easily shipped and updated. Hence, it's ideal for engineering applications where we cannot fully anticipate functionalities in advance (for example, the types of devices that might one day access the application). Microservice architectures are a part of a larger shift in IT departments towards a DevOps culture, in which development and operations teams work closely together to support an application over its lifecycle, and go through a rapid or even continuous release cycle.

Microservices act as standalone application subunits or components, implementing specific communication protocols for sending and receiving messages. In microservices, data flows through smart endpoints, which also process incoming information. Using well-defined interfaces and protocols, application developers can deploy different microservices on heterogeneous infrastructures without a specific integration framework. Generally, microservice communication uses a REST approach based on HTTP and TCP protocols, XMPP, or JavaScript Object Notation (JSON). However,

2325-6095/16/$33.00 © 2016 IEEE

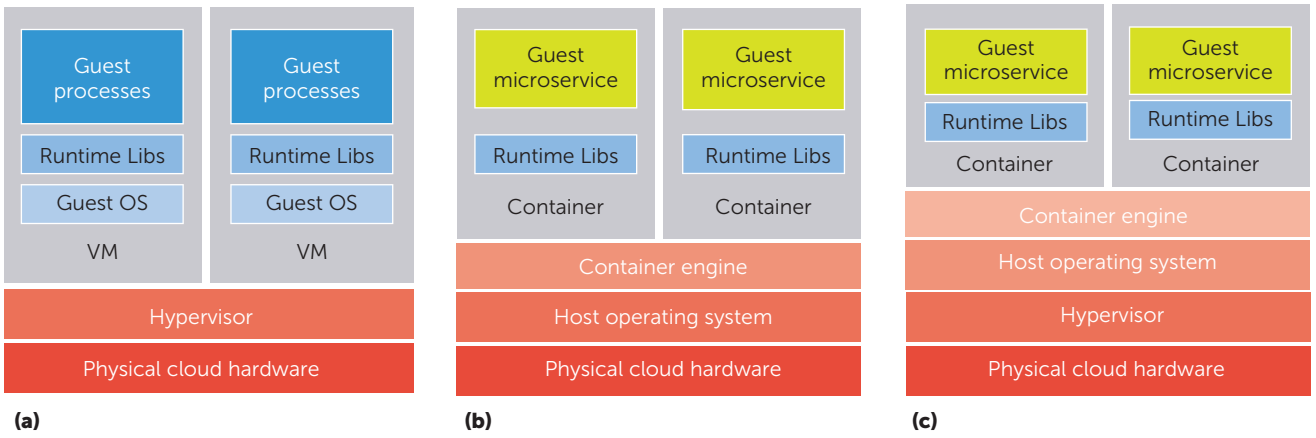

FIGURE 1. Comparison of cloud architectures: (a) hypervisor-based application deployment, (b) hypervisor-free containerized microservice, and (c) containerized microservice within a hypervisor-managed physical host.

currently, there are no widely adopted standardized protocols or data formats for microservice communication.[1] Microservice deployment and execution also leads to various networking issues. To this end, application developers currently adopt various software-defined networking (SDN) and network function virtualization (NFV) solutions for networking microservices.

Overview of Virtualization Technologies

Hypervisor-based resource virtualization (such as that used by Xen and VMware) is a key concept in cloud computing. Hypervisor-based virtualization enables cloud providers to create unique virtual machines (VMs) that share a set of physical hardware resources (CPU, memory, network, and disk). Each VM executes distinct operating system instances (ranging from proprietary to open source), which supports fault-tolerant and isolated security context behavior.

Container-based virtualization can be used to create microservices.[2] A container is a collection of operating system kernel utilities configured to manage the physical hardware resources used by a particular application component.[3] Containerization allows cloud providers to instantiate, relocate, and optimize hardware resources in a more flexible way while providing near-native performance (if deployed in "hypervisor-free" mode). Because the containers share a single operating system kernel, they incur lower overhead.[3] However, container-based virtualization leads to weaker isolation and introduces greater security vulnerabilities than hypervisor-based virtualization.[4]

From the user viewpoint, each container looks and executes exactly like a standalone operating system. Additionally, in a cloud computing scenario, developers can deploy a higher density of containers (compared to VM density in hypervisor-managed datacenters) on the same physical hardware. Linux container virtualization (LCV) is the most well-known container-based virtualization technology. Popular LCV solutions include Docker, LXC, lmctfy, and OpenVZ.

Figure 1 shows the key architectural differences between hypervisor-based and container-based virtualization. Figure 1a shows application components deployed within a hypervisor-based VM that provides abstraction for full guest operating systems (one per VM). Figure 1b shows microservice deployment within a hypervisor-free containerized environment. Finally, Figure 1c shows microservice deployment within a containerized environment on a physical hardware managed by a hypervisor-based VM. After physical hardware (for example, a server or appliance), a downward-facing hypervisor is more suitable for managing infrastructure-as-a-service (IaaS) clouds, whereas containers are more suited for managing platform-as-a-service (PaaS) clouds. Having said that, hypervisor-free containerization isn't a replacement for traditional hypervisor technologies;

the two technologies complement each other and must be carefully analyzed during the application architecture design phase in terms of performance isolation, overhead, and security requirements.

Container Engines for Microservices Scheduling and Management

Several tools can instantiate and manage containers in clouds. Docker Swarm, for example, provides native clustering for Docker containers. It turns a pool of Docker hosts into a single virtual Docker host. Because Docker Swarm serves the standard Docker API, any tool that already communicates with a Docker daemon can use Swarm to transparently scale to multiple hosts. A Docker container manager represents the basic container-oriented technology.

Kubernetes is an open-source technology for automating deployment, operations, and scaling of containerized applications. It groups the containers making up an application into logical units for easy management and discovery—for example, based on their resource requirements and other constraints. Kubernetes also provides horizontal scaling of applications, which can be performed manually or automatically based on CPU load. Finally, it provides automated rollouts and rollbacks and self-healing features.

Magnum is the OpenStack API service that makes container orchestration engines such as Docker Swarm and Kubernetes available as first-class resources in the OpenStack managed datacenter. Magnum uses the Heat service to schedule an operating system image, which contains Docker and Kubernetes, and runs this image on either VMs or a bare metal cluster.

The Google Container Engine provides a commercial service that relies on Docker and Kubernetes for cluster management and orchestration. Similarly, the Amazon Elastic Compute Cloud (EC2) container service supports Docker containers to be deployed on a managed cluster of Amazon EC2 instances. Rackspace is slightly behind with respect to container-based offerings. Its beta service, Carina, is based on Docker Swarm and doesn't provide any elasticity features.

Openstack Neutron supports the management of virtual LANs in cloud datacenters by creating ad hoc NFV. NFV uses virtualization technologies to manage core networking functions via software instead of relying on hardware to handle these functions.

Creating NFVs using Open Virtual Network (OVN) technology guarantees an efficient and secure use of the network. OVN complements existing SDN capabilities, adding native support for virtual network abstractions, such as virtual L1 and L2 overlays and security groups. OVN also supports the security inspection of data transfer inside virtual networks (for example, packet inspection); hence it provides extra features useful for increasing customer security and privacy

Open Issues in Scheduling and Resource Management

Despite the clear technological advances in container and hypervisor-based virtualization technologies, we are yet to realize a standard large-scale, performance-optimized scheduling platform for managing an ecosystem of microservices networked together to create a specialized application stack, such as a multitier Web application and Internet of Things (IoT) application. Future efforts will focus on solving the following research challenges.

Configuration Selection and Management

A cloud application (for example, a multitier Web application) must typically combine multiple interdependent microservices that provide diverse functionalities—for example, load balancer, webserver, and database server. Moreover, these microservices have both control and dataflow dependencies. The challenges exist in dealing with heterogeneous configurations of microservices and cloud datacenter resources driven by heterogeneous performance requirements. With the increase in microservice application functionality types (encryption, compression, SQL/NSQL server, virtual private network, and so on) and the heterogeneity of container engines (LXC, Docker, Google, and Amazon) and underlying cloud datacenter resources, the mapping of microservices to datacenters demands selecting bespoke configurations from an abundance of possibilities,[5] which is impossible to resolve manually.

Branded price calculators, available from public cloud providers (Amazon and Azure, for example) and academic projects (Cloudrado), allow comparison of

hardware resource leasing costs. However, these calculators can't recommend or compare configurations across microservices and datacenter resources.

We therefore need new research that focuses on developing techniques for accurately modeling, representing, and querying configurations of microservices and datacenter resources. In addition, we need general-purpose decision-making techniques, driven by heterogeneous performance requirements, to automate the selection of microservice configurations and their mapping to heterogeneous datacenter resources.[5]

Application Topology Specification and Composition

To compose a microservices-based application topology, you need to describe the microservices using a well-known standard. For example, you can base microservice descriptions on the Topology and Orchestration Specification for Cloud Applications (Tosca)/YAML along with the usual image representation. Moreover, workloads pertaining to different microservices depend on each other, and changes in one microservice's execution and dataflow will influence those of others. Overall, the topology specification and composition needs to cover the whole life cycle—that is, deploy, patch, monitor, reconfigure, and shutdown driven by the performance objectives of each microservice as well as the application as a whole.

The Business Process Execution Language (BPEL) and Web Service Choreography Interface (WSCI) are examples of Web service composition (agnostic to microservices) languages used in SOAs. The Resource and Application Description Language (RADL) is designed for composing and deploying VM images to different cloud providers.[6] Some application topology composition and specification tools found in literature (Crane, Fig, and Maestro, for example) can't deploy microservices across distributed datacenter hosts.[2] Although Tosca supports topology pattern specification, it lacks support for describing data and control flow dependencies between microservices, with a specific focus on identifying event coordination and dataflow mechanisms; properties of microservices in terms of workload features (such as data format, query rate, and runtime I/O dependency); and performance objectives and measures relevant to microservices.

Hence, an important research direction is to investigate a microservices composition framework, which will facilitate knowledge reuse and make it simpler for application engineers to interact with a complex computing platform.

Performance Characterization and Isolation

In a datacenter, microservices can be deployed inside hypervisor-based VMs or on nonvirtualized physical hardware. A recent study found that deployment within VMs imposes additional performance overhead while giving no extra benefit compared to deploying microservice containers on a virtualized physical server.[7] As noted earlier, single containers, such as Docker, can support multiple and heterogeneous microservices that provide various application-specific features in a containerized environment. In this environment, unexpected interference and contention can occur. For some microservices (such as a compression server) storage requirements dominate, whereas for others (for example, transactional query processing by database server) computational requirements dominate, and for still others (for example, a VPN server) communication requirements dominate. Hence, container engines (Kubernetes, Docker Swarm, and so on) must consider which microservices to combine to minimize workload interference and contention. Balancing resource consumption and performance is critical in deciding where to deploy microservices.

Some recent work has investigated performance isolation and interference detection. New hardware design techniques change processor cache architecture partitioning[8] or integrate novel insertion policies to pseudo-partition caches to reduce contention.[9]

Hardware-based approaches add complexity to the processor architecture and are difficult to manage over time. Sriram Govindan and his colleagues developed a scheme to quantify the effects of cache contention between consolidated workloads.[10] However, they limit their discussion to cache contention issues, ignoring other hardware resource types. Ripal Nathuji and Aman Kansal present a control theory-based consolidation approach that mitigates the effects of cache, memory, and hardware prefetching contention of coexisting workloads.[11] However, their focus is CPU-bound or compute-intensive applications.

Several new research topics are worthy of investigation: performance isolation and characterization techniques when multiple microservices run in the same container or on the same physical host; live migration of containers to reduce interference and contention; and tradeoffs between live migration and restarting.

Microservice Monitoring

Guaranteed application performance requires clear and real-time understanding of performance metrics across microservices and datacenter resources. However, variations in performance metrics across different microservices and datacenter resources complicate this problem. For example, key performance metrics for SDN resources are throughput and latency; for CPU resources, they're utilization and throughput; and for SQL and NoSQL database microservices, it's query response time. Therefore, how to define and formulate performance metrics coherently across microservices to give a holistic view of data and control flows remains an open issue.

Monitoring tools that were popular in the grid and cluster computing era (for example, R-GMA and Hawkeye) were concerned only with monitoring performance metrics at the datacenter resource level (such as CPU percentage and TCP/IP performance), but not at the microservice level (such as end-to-end request processing latency and communication overhead). Cluster-wide monitoring frameworks (Nagios, Ganglia, Apache Hadoop, and Apache Spark) provide information about hardware metrics (cluster, CPU, and memory utilization, and so on) of cluster resources that might belong to public or private cloud datacenter.[12,13] Monitoring frameworks used by the Amazon EC2 Container Service (Amazon CloudWatch) and Kubernetes (Heapster) typically monitor CPU, memory, filesystem, and network usage statistics, so they can't monitor microservice-level performance metrics.

This leads to several new research topics, including development of holistic techniques[13] for collecting and integrating monitoring data from all microservices and datacenter resources so administrators or a scheduler (a computer program) can track and understand the impact of runtime uncertainities (for example, failure, load-balancing efficiency, and overloading) on performance without understanding the whole platform's complexity.

Elastic Scheduling and Runtime Adaptation

The elastic scheduling of microservices is a complex research problem due to several runtime uncertainties.

First, it's difficult to estimate microservice workload behavior in terms of request arrival rate, type, and processing time distributions; I/O system behavior; and number of users connecting to different types and mix of microservices. The real challenge in devising microservice-specific workload models is to accurately learn and fit statistical functions to the monitored distributions such as request arrival pattern, CPU usage patterns, memory usage patterns, I/O system behaviors, request processing time distributions, and network usage patterns.

Without knowing the workload behaviors of microservices, it's difficult to make decisions about the types and scale of datacenter resources to be provisioned to microservices at any given time. Furthermore, the availability, load, and throughput of datacenter resources can vary in unpredictable ways, due to failure or congestion of network links.

Kubernetes offers a microservice container reconfiguration feature, which scales by observing CPU usage ("elasticity is agnostic to the workload behavior and performance targets of microservice." Amazon's autoscaling service employs simple threshold-based rules or scheduled actions based on a timetable to regulate infrastructural resources (for example, if the average CPU usage is above 40 percent, add another microservice container). Other cloud providers have implemented similar simple rule-based reactive runtime scheduling techniques: Google's Cloud Platform autoscaler, Rackspace's Auto Scale, Microsoft Azure's Fabric Controller, and IBM's Softlayer autoscale.

To the best of our knowledge, no prior work has developed workload and resource performance prediction models to enable reconfiguration (scaling, descaling, and migration) of microservices on cloud datacenters while ensuring microservice-specific performance targets. Hence, important new research is investigating predictive workload and performance models to forecast workload input and performance metrics across multiple, collocated microservices deployed on cloud datacenter resources.

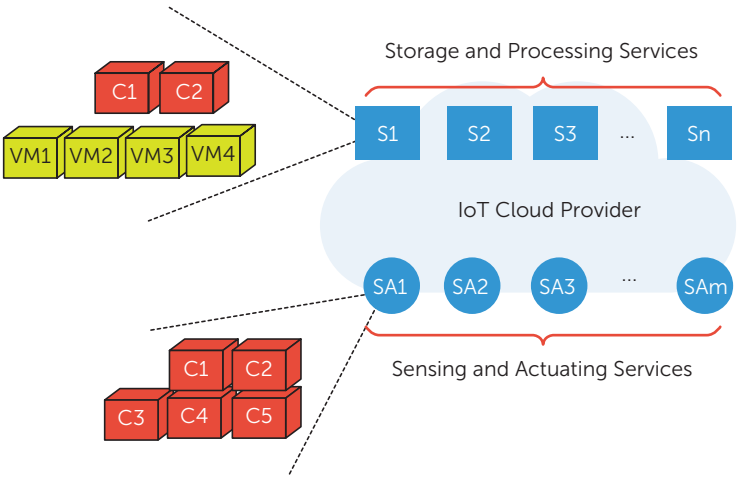

FIGURE 2. A microservice as the enabler for the IoT application cloud. IoT application are decomposed into collection of microservices which are distributed across physical hardware resources available in the cloud and on the network edge.

Evolution of Microservice-Powered Cloud Paradigms

Wide-scale adoption of containerization technologies and microservices architectures will strongly influence other emerging computing paradigms.

Cloud Computing and Internet of Things

The combination of cloud computing and the IoT is presenting new opportunities for delivering new types of application services (see Figure 2). For example, private, public, and hybrid cloud providers are looking to integrate their datacenters' software and hardware stacks with embedded devices (including sensors and actuators) to provide IoT as a service (IoTaaS).

Typically, IoT devices run customized software developed with a particular programming language and/or development framework. Minimal processing and storage tasks can be performed in IoT devices (for example, a sensor gateway or SDN virtualization) by deploying lightweight, containerized microservices.[14,15] Meanwhile, the massive data storage and processing tasks (data mining and big data analytics) are performed in cloud datacenters that exploit virtualization (both hypervisor and container-based) to elastically scale up/down storage and processing capabilities.

Federated Clouds

The cloud services market has been growing in recent years, a trend that's confirmed by the number of cloud providers that have appeared on the market. Currently, small and medium cloud providers can't directly compete with the big players (such as Google, Amazon, and Microsoft), so they must implement new business strategies to penetrate the market.[16,17]

In particular, small and medium providers can establish stronger partnerships to share resources according to the rules of the cloud federation ecosystem they belong to. Small providers can federate with large providers to gain economies of scale, optimize their assets, scale their capabilities, and share resources to establish new forms of collaboration. If a small provider's cloud runs out of capacity, it can migrate its microservices to federated datacenters to ensure business continuity (see Figure 3).

However, federated clouds need to respond to high heterogeneity across independent cloud systems, efficient and secure data exchange among clouds, and the ability to efficiently deploy resources and services across such federated systems. Indeed, the dynamism of a federation with incoming and outgoing providers and variable resource availability makes microservices and containers the best solution to quickly adapt to changes in the federated system.

Microservices will simplify orchestration of networked applications across heterogeneous cloud datacenters and emerging microdatacenters (on the network edge). However, the creation of such applications (for example, smart city and smart healthcare IoT clouds) requires new research into scheduling and resource management algorithms and platforms for managing highly distributed and networked microservices.

References

1. A Sill, "The Design and Architecture of Microservices," *IEEE Cloud Computing*, vol. 3, no. 5, 2016, pp. 76–80.
2. C. Pahl and B. Lee, "Containers and Clusters for Edge Cloud Architectures: A Technology Review," *Proc. 3rd Int'l Conf. Future Internet of Things and Cloud* (FiCloud), 2015, pp. 379–386.
3. M. Xavier et al., "Performance Evaluation of

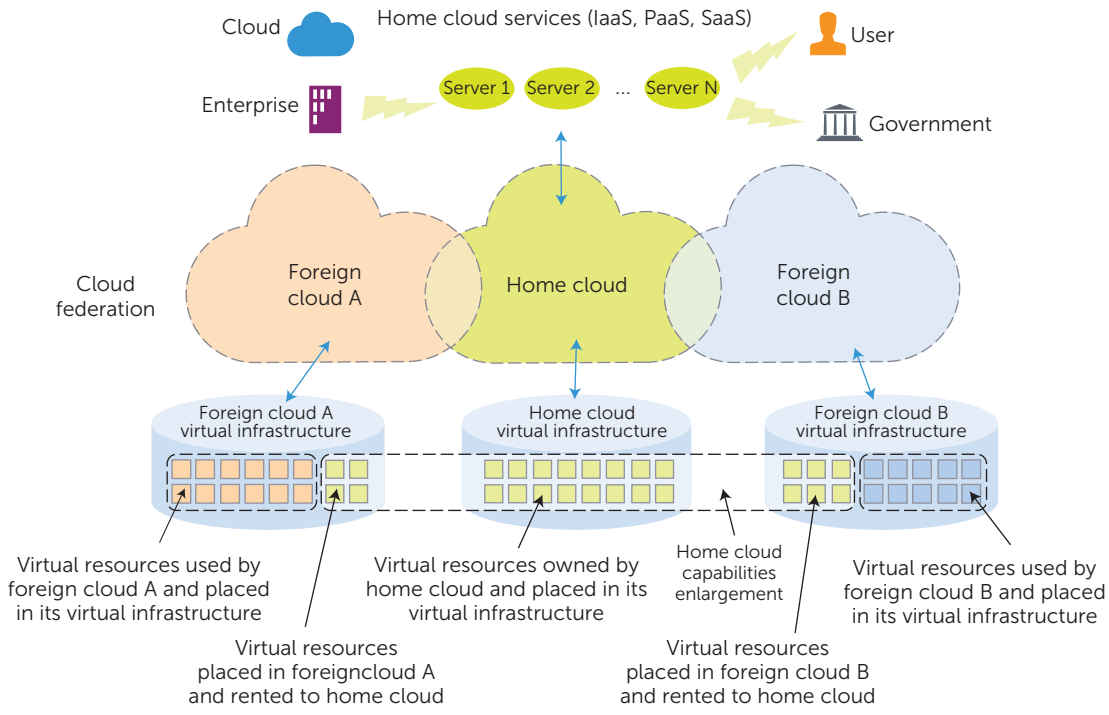

FIGURE 3. Microservice as the basis of federating multiple cloud datacenters as part of cohesive federation, where datacenter providers can meet the performance requirements of client applications through optimal placement and migration of microservices across datacenters.

Container-Based Virtualization for High Performance Computing Environments," *Proc. 21st Euromicro Int'l Conf. Parallel, Distributed, and Network-Based Processing* (PDP), 2013, pp. 233–240.

4. C. Esposito, A. Castiglione, and K.-K.R. Choo, "Challenges in Delivering Software in the Cloud as Microservices," *IEEE Cloud Computing*, Vol. 3, no. 5, 2016, pp. 10–14.
5. R. Ranjan et al., "Cross-Layer Cloud Resource Configuration Selection in the Big Data Era," *IEEE Cloud Computing*, vol. 2, no. 3, 2015, pp. 16–22.
6. M. Caballer et al., "Dynamic Management of Virtual Infrastructures," *J. Grid Computing*, vol. 13, Mar. 2015, pp. 53–70.
7. W. Felter et al., "An Updated Performance Comparison of Virtual Machines and Linux Containers," *Proc. IEEE Int'l Symp. Performance Analysis of Systems and Software* (ISPASS), 2015, pp. 171–172.
8. M.K. Qureshi and Y.N. Patt, "Utility-Based Cache Partitioning: A Low-Overhead, High-Performance, Runtime Mechanism to Partition Shared Caches," *Proc. 39th Ann. IEEE/ACM Int'l Symp. Microarchitecture* (Micro 06), 2006, pp. 423–432.
9. Y. Xie and G.H. Loh, "Pipp: Promotion/Insertion Pseudo-Partitioning of Multi-Core Shared Caches," *Proc. 36th Ann. Int'l Symp. Computer Architecture* (ISCA 09), 2009, pp. 174–183.
10. S. Govindan et al., "Cuanta: Quantifying Effects of Shared On-Chip Resource Interference for Consolidated Virtual Machines," *Proc. 2nd ACM Symp. Cloud Computing* (SOCC 11), 2011, article 22.
11. R. Nathuji and A. Kansal, "Q-Clouds: Managing Performance Interference Effects for QoS-Aware Clouds," *Proc. 5th European Conf. Computer Systems* (EuroSys 10), 2010, pp. 237–250.
12. R. Ranjan, "Streaming Big Data Processing in

Datacenter Clouds," *IEEE Cloud Computing*, vol. 1, no. 1, 2014, pp. 78–83.
13. M. Natu et al., "Holistic Performance Monitoring of Hybrid Clouds: Complexities and Future Directions," *IEEE Cloud Computing*, vol. 3, no. 1, 2016, pp. 72–81.
14. A. Celesti et al., "Exploring Container Virtualization in IoT Clouds," *Proc. 2016 IEEE Int'l Conf. Smart Computing* (SmartComp), 2016, pp. 1–6.
15. M. Fazio and A. Puliafito, "Cloud4sens: A Cloud-Based Architecture for Sensor Controlling and Monitoring," *IEEE Comm*, vol. 53, Mar. 2015, pp. 41–47.
16. M. Assis and L. Bittencourt, "A Survey on Cloud Federation Architectures: Identifying Functional and Non-functional Properties," J. Network and Computer Applications, vol. 72, 2016, pp. 51–71.
17. A. Celesti et al., "Characterizing Cloud Federation in IoT," *Proc. 30th Int'l Conf. Advanced Information Networking and Applications Workshops* (WAINA), 2016, pp. 93–98.

MARIA FAZIA *is an assistant researcher of computer science at the University of Messina. Her research interests include distributed systems and wireless communications, especially with regard to the design and development of cloud solutions for IoT services and applications. Fazia has a PhD in advanced technologies for information engineering from the University of Messina. Contact her at mfazio@unime.it.*

ANTONIO CELESTI *is a postdoctoral researcher at University of Messina. His research interests include distributed systems and cloud computing, with particular regard to federation, storage, security, energy efficiency; and assistive technology. Celesti has a PhD in advanced technology for information engineering from the University of Messina, Italy. Contact him at acelesti@unime.it.*

RAJIV RANJAN *is a reader in the School of Computing Science at Newcastle University, UK; chair professor in the School of Computer, Chinese University of Geosciences, Wuhan, China; and a visiting scientist at Data61, CSIRO, Australia. His research interests include grid computing, peer-to-peer networks, cloud computing, Internet of Things, and big data analytics. Ranjan has a PhD in computer science and software engineering from the University of Melbourne (2009). Contact him at raj.ranjan@ncl.ac.uk or http://rajivranjan.net.*

LYDIA Y. CHEN *is a research staff member at the IBM Zurich Research Lab, Zurich, Switzerland. Her research interests include modeling, optimizing performance and dependability for big data applications and highly virtualized datacenters. She received a PhD in operations research from the Pennsylvania State University. Contact her at yic@zurich.ibm.com.*

CHANG LIU *is a research fellow (assistant professor) at Newcastle University, UK. His research interests include cloud computing, big data, distributed systems, Internet of Things, and information security and privacy. Liu has a PhD in information technology from the University of Technology, Sydney, Australia. Contact him at: changliu.it@gmail.com.*

MASSIMO VILLARI *is an associate professor of computer science at the University of Messina. His research interests include cloud computing, Internet of Things, big data analytics, and security systems. Villari has a PhD in computer engineering from the University of Messina. He's a member of IEEE and IARIA boards. Contact him at mvillari@unime.it.*