I/Ocloud: Adding an IoT Dimension to Cloud Infrastructures

Dario Bruneo, Salvatore Distefano, Francesco Longo, Giovanni Merlino, and Antonio Puliafito, University of Messina

Infrastructure as a service is a successful cloud-based utility paradigm. Its effectiveness stems from employing virtualization and enabling cloud providers to leverage elastic datacenter capabilities. To expand the reach and benefits of IaaS, the cloud must embrace the Internet of Things infrastructure as virtualized I/O resources and first-class objects at the edge.

A growing interest in the Internet of Things (IoT) stems from the pervasiveness of sensor- and actuator-hosting devices that act as a programmable gateway to the physical world. The approaches in literature to fully exploit the IoT ecosystem mainly rely on the adoption of the cloud paradigm.[1] However, the cloud is typically leveraged as is, and IoT is considered a mere data provider (see Figure 1a) or at most as a bidirectional remote interface (see Figure 1b), usually not reconfigurable.

We believe that technologies are now mature enough to challenge the mainstream consensus on the relationship between the cloud and IoT. Thus, we aim to stretch the infrastructure-as-a-service (IaaS) paradigm by adapting cloud metaphors to the IoT infrastructure. We propose a cloud infrastructure merged with IoT facilities at the broadest level; that translates into viewing IoT as a natural extension of the datacenter, as in Figure 1c. This way, it becomes possible to pool a diverse range of geographically dispersed devices as infrastructure resources, together with standard IaaS computing and storage facilities.

This perspective might also be formulated by looking at IoT as a third dimension of the cloud—that is, its input/output—a paradigm we call *I/Ocloud*. Indeed, the virtual infrastructure, which typically consists of computing and storage resources, would be complemented by standardized access to sensors and actuators, without resorting to ad hoc, application-level, remote APIs. Moreover, this approach addresses board management and runtime customization by virtue of mechanisms already built into cloud platforms.

In this article, we present our I/Ocloud approach, which offers a twofold benefit of providing standardized and generalized programming capabilities on top of IoT resources, irrespective of the layout of the underlying infrastructure. At the same time, it preserves the ability to exploit the unique features of an IoT-enhanced distributed datacenter, such as the availability of nodes at the edge, that might then act as pods for incoming tasks. We also discuss design aspects as well as technologies, tradeoffs, use cases, and experimental results, highlighting the ongoing development of the OpenStack-based Stack4Things framework for I/Oclouds.

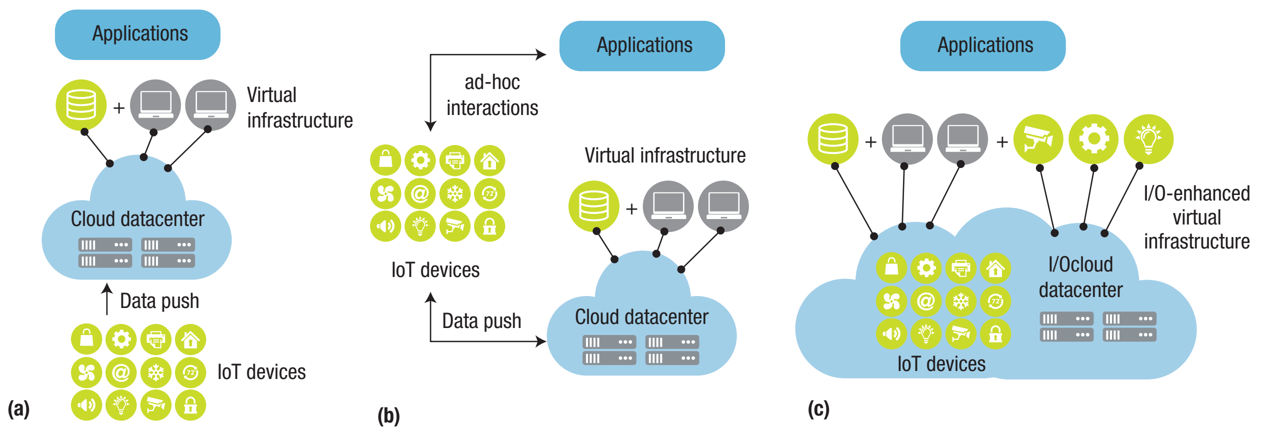

FIGURE 1. Internet of Things cloud architectures: (a) IoT as a data source for cloud-hosted data stores, (b) IoT as remote interface to cloud-hosted applications, and (c) IoT as an extension of the datacenter infrastructure hosting the cloud instance.

I/OCLOUD

An I/Ocloud provides IoT virtualization features, alongside plain IaaS (computing and storage) virtualization. Moreover, it exposes interfaces and mechanisms to direct an infrastructure that also includes IoT devices.

Nodes and resources

For our purposes, an IoT infrastructure is made up of a set of heterogeneous IoT nodes, exposing IoT resources.

An *IoT resource* is an entity (such as a sensor) that can be exported and attached, but might not itself be programmed.

Examples of IoT resources include sensors directly connected to general-purpose I/O (GPIO) pins of a single-board computer (SBC), an accelerometer hosted within an iOS smartphone, a (proprietary) home automation object, and a wireless sensor node.

We then define an *IoT node* as any computing unit that hosts physical IoT resources as well as some logic. Examples of IoT nodes are SBCs, smartphones, and so on. IoT nodes are usually characterized by relatively limited computational resources, minimal primary and secondary storage, and low bandwidth. Typically, these are deployed at the edge of the Internet, usually behind a firewall or Network address translation (NAT) gateway. Most are characterized by unstable network connectivity.

An essential role for an I/Ocloud is ensuring that IoT systems get engaged as active elements of the infrastructure, while still being bound to their unique attributes. A core approach in I/Oclouds indeed lies in redefining virtualization to include node-hosted IoT resources as entities. This leads us to I/O virtualization (virtIO).

Virtual entities

The I/Ocloud approach extends the concept of virtualization to the IoT world by providing mechanisms to abstract IoT resources into virtual counterparts with standardized interfaces.

We define an *I/O resource* as an instance of a developer-friendly interface abstracting I/O primitives from the underlying IoT resource, either in its entirety or as a subset or even superset (that is, a logical grouping of several IoT resources).

To enable this kind of scheme, we based our design for the virtualization of IoT resources in the form of a file system abstraction, based on the Unix philosophy that "everything is a file." This is inspired by the notion that many common instances of IoT nodes (such as Arduino YÚN and Raspberry Pi) already employ this kind of approach in their system software (such as the GPIO pseudo–file system[2]) to provide access to the physical (pin) interface. Our approach can therefore be formulated as an IoT-focused generalization of file system–based abstractions for I/O.

This approach might also be viewed at a high level as extending virtualization concepts to the IoT world.

A *virtual IoT node* (VN) is a self-contained and isolated environment that might be instantiated on top of either the datacenter-level infrastructure or a physical IoT node at the edge. A VN can host logic and have attached I/O resources, akin to an actual node. A VN example is a multiarch OpenWRT-based container (openwrt.org) acting as a virtual instance of a YÚN.

In terms of typical cloud facilities, we might differentiate between plain virtual machines (VMs) and VNs using the notion of instance flavors. Once IoT resources are virtualized as I/O, VMs and VNs can be instantiated and I/O attached to them as if they were physically connected. This can be done even if physical IoT resources are remote, despite network connectivity issues, while also addressing contention among concurrent requests acting on the same IoT resources. In doing so, we enable IoT developers to design their IoT applications as if they were expected to run on top of a single IoT node, hiding the intrinsic distributed nature of the IoT infrastructure.

Figure 2 depicts a combined edge/virtIO scenario conceived according to the I/Ocloud approach.

An I/O resource might correspond to an IoT resource, locally available to a certain IoT node, that is chosen to be exposed by the I/Ocloud. An IoT resource might be available exclusively to the hosting node, when not contemplated for pooling. In that case, no corresponding I/O resource gets instantiated, as it happens for three out of six IoT resources depicted in Figure 2. A plain VM or a VN running on the I/Ocloud might be attached to I/O resources, possibly exposed by several IoT nodes at once. For example, see the VM in Figure 2 attached to one I/O resource for each of the two IoT nodes.

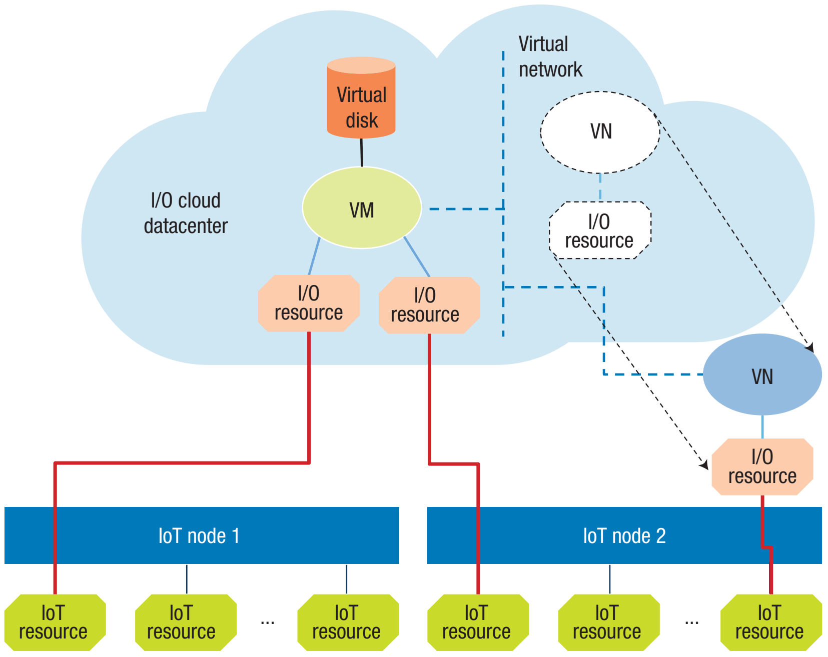

FIGURE 2. I/Ocloud edge/virtIO (edge and I/O virtualization) capabilities. I/O resources can correspond to IoT resources, locally available to certain IoT nodes exposed by the I/Ocloud.

I/O virtualization in IoT

In classic IaaS implementations, resources are made available to the pool as long as they belong to the datacenter. Typical IaaS mechanisms are provided to manage and virtualize them. Resources that may be exposed as logical entities by the VM monitor (VMM, or hypervisor) include CPUs, memory, network interfaces, and less frequently, I/O peripherals. The underlying physical resources are characterized by the multiplexing capability—for example, multiple virtual CPUs might be scheduled over the same physical CPU. A single VM is usually characterized by a certain number of virtual CPUs, a certain amount of virtual memory, a set of virtual disks, and one or more virtual network interfaces. Virtualization technologies are thus exploited to assemble pools of VMs hosted on top of the physical machines in the datacenter.

With regard to I/O peripherals, the hypervisor might be able to export one or more of their functions to any VM, but this often requires assistance from hardware, such as an I/O memory management unit, and is based on memory-remapping facilities and limited to directly attached resources, thus yielding limited flexibility.

In the IoT world, nodes are characterized by their direct interaction with the physical world through their resources. Thus, we believe that virtualization technologies need to be extended to enable the hosting of detachable instances of IoT resources, to be attached on-demand to machine instances of any kind.

One of the main differences between classic VM resources (CPU, memory, storage, network, peripherals, and so on) and IoT resources is that the former can be easily multiplexed or partitioned among a set of VMs. IoT resources, on the other hand, might consist of sensors, actuators, resource-constrained devices, and other closed smart objects that possibly do not always reveal the same multiplexing nature. It's important to manage carefully the contention among virtual IoT nodes insisting on the same IoT resource—for instance, conflicting directives issued by different IoT nodes on the same actuator could lead to incoherent, unpredictable situations. Even in the case of sensing resources, such contention could become an issue if unarbitrated configuration actions are issued by different IoT nodes on the same device.

Edge/virtIO duality

The dual nature of this perspective on IoT and cloud integration emerges as soon as we recognize that, on the one hand, I/O resources provide an interface to the physical world and, on the other, the underlying things might also represent a pool of distributed machines at the edge.

Pushing computing to the periphery of the network means decentralizing business logic as desired, scheduling its execution on the boards themselves, whenever conceivable. This means possibly engaging the cloud to mainly support communication below the application level[3] when direct interaction among nodes is impaired as a result of network-imposed constraints. In any case, this kind of approach is still limited, implying applications that run exclusively on nodes that have directly attached IoT resources.

We aim for mechanisms to let logic, when operating over geographically distributed resources, possibly run at the edge of the network (not in the cloud), regardless of the transducers' physical wiring.

The chief logical step to achieve such an objective lies in enabling the instantiation of VNs with remote I/O resources attached directly, as if those were local. One example scenario is where both a VM and a VN have I/O resources attached, and a VN has been migrated off the datacenter to the IoT node hosting the VN-attached I/O resource.

VNs may be instantiated as isolated and portable environments (lightweight containers) straight within an I/Ocloud datacenter and migrated to other infrastructure (possibly at the edge; see Figure 2). Conversely, they could be first instantiated at the edge and then offloaded elsewhere (for instance, to the datacenter). Moreover, virtual networks could be set up by the cloud management framework and instantiated among any combination of VMs and VNs, spanning the underlying datacenter datapaths as well as wide area networks (see Figure 2 on the right).

On these premises, a number of significant advantages can be achieved, including

- fully decoupling business logic from IoT infrastructure,
- empowering the notion of IoT infrastructure as code (IaC),
- enabling an extreme style of edge computing,
- exposing the lowest level of abstraction for IoT resources, and
- handling IoT resources with a high level of granularity.

IMPLEMENTATION

Stack4Things (S4T)[4] is an open source research effort aimed at extending OpenStack—a successful open source middleware for IaaS clouds—to implement the I/Ocloud vision. On the IoT node side, the S4T Lightning-Rod interacts with the node's OS tools and services and with the IoT resources directly attached to it. It represents the point of contact with the cloud infrastructure, allowing administrators to manage IoT resources, regardless of the configuration and level of connectivity of the nodes hosting them. This is ensured by Web Application Messaging Protocol (WAMP)-based communication[5] between the S4T Lightning-Rod and its cloud counterpart—namely, the S4T IoTronic service. Virtual networking services and node-side service forwarding to the cloud are implemented by tunneling mechanisms over WebSockets.[6]

As we mentioned earlier, one of the core mechanisms enabling our vision lies in treating IoT resources as virtual I/O. S4T allows the IoT infrastructure administrator to inject logic into IoT nodes at runtime in the form of S4T drivers for the interaction with local IoT resources. In this regard, S4T provides a virtual file system abstraction of the IoT resources dispersed in the physical world. We use the FUSE (File System in User Space) subsystem, originally developed for Linux, to implement such a file system.

The driver consists of a JavaScript Object Notation configuration file and a Node.js module, the former associating each method in the latter with a Posix-compliant file system primitive performed on one of the files in the corresponding virtual file system. The running instance of a driver, on top of an IoT node or even in the cloud, represents an I/O resource. The system associates a virtual file system with each driver instance, where the file system acts as a Unix-like interface for any interaction with the corresponding IoT resource. Lightning-Rod translates read and write operations into invocations of driver methods through FUSE libraries.

Figure 3 depicts the S4T architecture for drivers. On an IoT node (the left side of Figure 3), the Lightning-Rod loads drivers and, via FUSE, creates a virtual file system (exposing I/O resources) that can then be attached to local containers running custom applications. In addition, on a compute node in the datacenter (the right side of Figure 3), the IoTronic service can also populate the same file system structure, which a local VM/VN might then mount even if the resource is not local. When an app operates on such a file system, IoTronic will contact (through WAMP) its counterpart on the IoT node physically hosting the corresponding IoT resource, obtain the result of the call, and return it to the app. The same kind of reasoning can be applied to VNs running elsewhere, such as on another IoT node.

PERFORMANCE AND RELIABILITY CONSIDERATIONS

The WAMP protocol implements routed remote procedure calls (RPCs), meaning that the caller and callee are decoupled at the application level and it is up to a WAMP router to forward a call originating from the caller to the callee and relay results or errors back, and vice versa.

Even in terms of uneven connectivity, WAMP-enabled routed RPCs support the graceful management of disconnection given that logic can be injected to deal with the callee's disconnection, both on the WAMP router and caller side. A number of strategies might be devised to handle these kinds of exceptions. For example, the router might store the latest result of the issued call and return it if the callee gets disconnected, making it completely transparent for the caller. Another approach consists of delegating the caller to deal with such a situation, forwarding the error at the file system level with a standard Posix error code, to be captured and processed by the application running on top of the virtual IoT node.

In an I/Ocloud-enabled scenario, different conflicting requirements can be highlighted depending on the specific application. In particular, we identified a major tension between networking latency and computational power. Different policies might be implemented on top of edge-computing facilities, such as forcing the virtual IoT node hosting an interactive application to be deployed near the data sources, together with a publish/subscribe router exploited for internode communication, when latency must be minimized. On the other hand, nodes hosting CPU-bound applications could be hosted on the cloud to maximize the amount of computing resources that can be assigned to the corresponding containers.

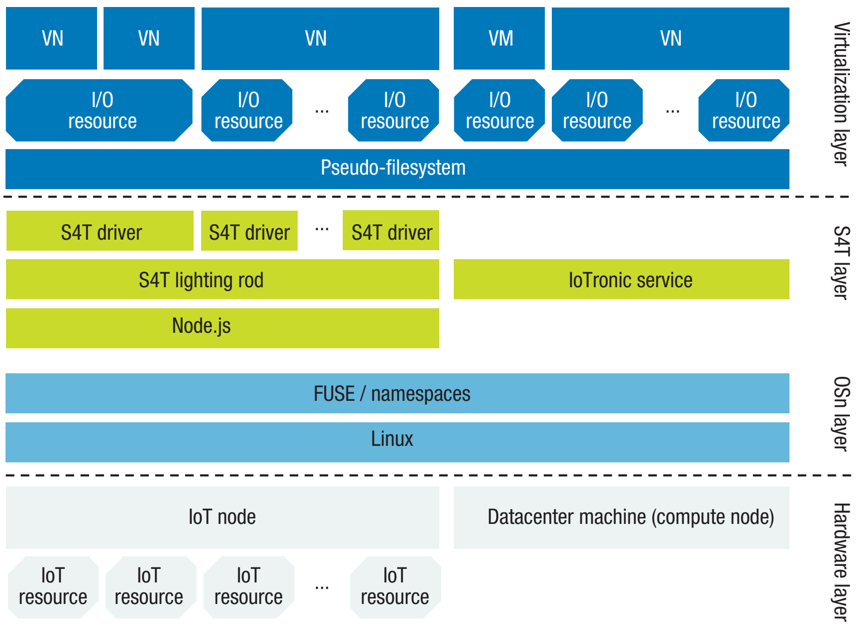

FIGURE 3. Stack4Things (S4T) file system drivers. The layered architecture includes an IoT node (left side) and a compute node (right side).

USE-CASE EXAMPLE

Building on the concept of a software-defined infrastructure, an I/Ocloud can transform complex IoT ecosystems in simple and programmable environments, where companies, scientists, and citizens can easily collaborate to develop innovative smart services.

Most IoT applications are designed from scratch—with a mix and match of APIs from different infrastructure/middleware vendors—to interface with multiple devices and design the back-end logic for the composition of data (and, where available, command) streams. In any case, applications must be aware of the hardware resources upfront and be developed against the software developers' kits that expose the infrastructure to be exploited.

Using the I/Ocloud APIs for orchestration, we propose shaping and composing a dispersed infrastructure by instantiating required network topologies to let back-end machines (such as the VM in Figure 2) access distributed resources as local I/O. That is, the goal is to load the right drivers and thus mount all the I/Ocloud-enabled file systems of choice.

As long as the IoT application we envision is designed to invoke these high-level APIs at the initialization phase, then the core logic could be reused from any existing legacy application code base. This might go so far as having ad hoc code replaced with invocations to functions belonging to specialized open source libraries, without any hard requirement to write glue code to interface with RESTful APIs.

A simple example of code structure for applications featuring sensors and actuators, possibly establishing feedback loops over those for automation, is

TABLE 1. Development options and benefits of the virtIO approach.

Application	I/Ocloud	Other Internet of Things clouds
Legacy	Can be used as is, and coded in any language	Requires extensive porting efforts or rewriting from scratch
Monolithic	Migration-transparent business logic, cleanly separated from setup code	Mix and match of API-level interactions only
Distributed	I/O can be rewired according to available resources	I/O hardwired

a typical maker-friendly Arduino-style "setup and loop" combination. As long as the setup routine is designed to take care of the aforementioned initialization phase in an IaC fashion, the loop routine just needs to cover the business logic, requiring only the inclusion of headers with regard to file-handling libraries, and selected off-the-shelf libraries for any specialized tasks at hand, such as data transformation, sound processing, or computer vision.

In the context of an I/Ocloud, developers could choose the language and tooling from framework-heavy object-oriented programming down to rapid script-based prototyping, positively impacting development productivity, mostly in terms of freedom of choice and broad reuse. For example, the Kaa IoT platform for smart-city applications (www.kaaproject.org) restricts the choice of languages to four (Java, C++, C, and Objective-C), whereas S4T choices are potentially unlimited because it is up to the developer to choose I/O libraries, typically robust and core to the chosen language and runtime. Moreover, in the case of similar platforms, business logic is entangled with setup code (subscriptions, REST invocations, and such), making code harder to follow.

Thanks to the possibility of attaching pins to any VN, further advantages include support for different software architecture paradigms—not only coding a distributed application, with business logic scattered on several boards, and the explicit coordination this approach entails, but also developing a monolithic application where preferred, thus favoring implicit coordination and natural exploitation of asynchronous patterns. In the latter case, if the developer already has a working application running locally, writing just a file system driver for access to GPIO, for instance, is enough to make the application work as is, in a distributed, multinode scenario.

Table 1 summarizes the aforementioned benefits in terms of development options, especially in light of the virtIO approach.

EXPERIMENTAL RESULTS

To further describe the potential of a framework for I/Ocloud, here we provide a board-to-cloud VN migration scenario, highlighting the virtues of the virtIO approach and providing a preliminary quantitative evaluation. We implemented a simple Node.js webserver that exposes a temperature sensor as a REST resource. The application was deployed in a VN (within a container) that, in a first stage of the experimental workflow, was hosted on a board (YÚN). A virtual GPIO file system was attached to the container, exposing access to the board's actual GPIO pins.

In a second stage of our experiment, the container was migrated to a S4T compute node, together with its virtual GPIO file system counterpart. Thanks to our I/Ocloud approach, in particular the virtIO file system–based abstraction, the application logic does not need to be adapted to the recipient environment because the migration to the cloud does not affect the way the application interacts with the temperature sensor.

To provide a quantitative evaluation, we used the Apache JMeter tool (jmeter.apache.org) to generate workloads for the webserver in both configurations. Figure 4 shows the experimental results we obtained, plotting a comparison between the Node.js webserver when it is running on the board and on the cloud.

Specifically, Figure 4a shows response times in the presence of 100 concurrent users with respect to varying computational load due to data preprocessing, and Figure 4b shows the corresponding error percentage. We emulated data preprocessing by performing a set of consecutive fast Fourier transforms (FFTs) on sensor data readings. The horizontal axis reports the number of FFTs performed

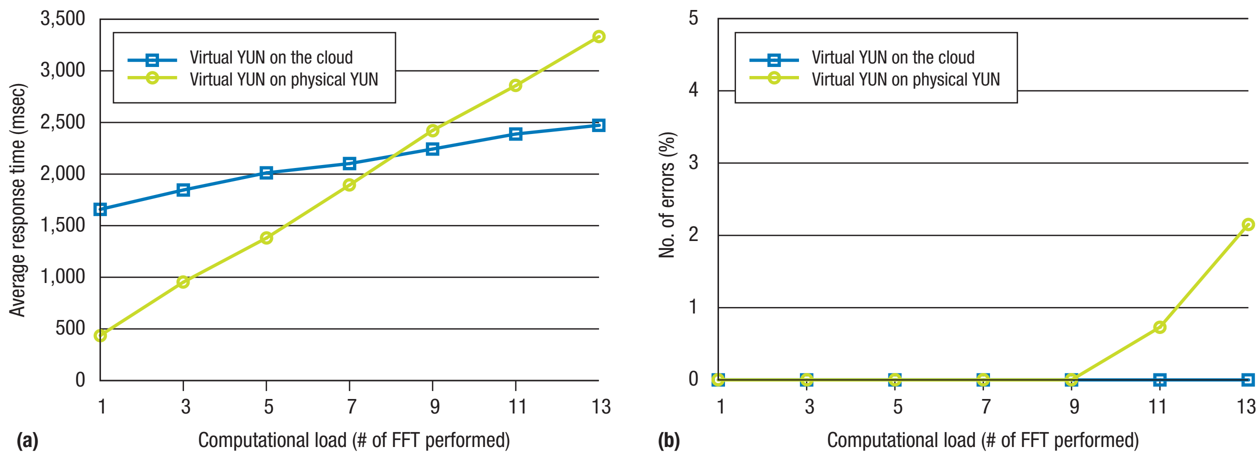

FIGURE 4. Experimental results: (a) response time and (b) error percentage. A client interacts with a virtIO-enabled virtual IoT node (VN)-hosted app through a REST interface exposing temperature readings.

per request. The curve profiles for the VNs running in the cloud compared with those running on the boards clearly highlight the advantages of moving certain workloads to the cloud. At the same time, the cloud framework ensures desirable properties, such as development in a familiar environment and choosing where to dynamically deploy the application logic as a function of the computational load, preferably at runtime as well.

RELATED WORK

Table 2 presents a feature matrix for the most relevant and current technologies and platforms with regard to IoT and cloud integration. The comparison spans numerous features and their scope, such as the cloud layer at which the integration is achieved, the support for node-side updates, interfaces, and licensing.

Further details and references complementing this overview from a methodological viewpoint are available in earlier studies that survey the main IoT–cloud approaches and solutions.[7–9]

We expect the I/Ocloud vision to take hold and become more mainstream in the near future, especially in the context of smart city environments and other heterogeneous cyber-physical systems scenarios, where the cloud approach coupled with a software-defined infrastructure might bring immediate benefits and provide more degrees of freedom in the development and deployment of IoT-enabled distributed applications.

Generalized, remote, virtualized I/O resources, on the one hand, and (extreme) edge computing, on the other, might empower all sorts of use cases, at different levels of abstraction, leaving most options available and squarely in the hands of DevOps personnel.

Moreover, fitting IaaS to IoT and taking advantage of the IaC approach enables the same cloud-related skillset and workforce to take over the IoT domain.

In terms of research directions, in this light most topics that are relevant for cloud infrastructure management and service-oriented architectures can thus be investigated in the IoT domain as well and reevaluated under this close, yet novel perspective.

REFERENCES

1. J. Mineraud et al., "A Gap Analysis of Internet-of-Things Platforms," *Computer Comm.*, vols. 89–90, 2016, pp. 5–16.
2. "GPIO Sysfs Interface for Userspace," Sysfs documentation, www.kernel.org/doc/Documentation/gpio/sysfs.txt.
3. D. Bruneo et al., "Stack4Things as a Fog Computing Platform for Smart City Applications," *Proc. IEEE Conf. Computer Comm. Workshops*

TABLE 2. IoT and cloud integration platform feature matrix.

IoT–cloud	Layers*	Updates (node)	Interfaces	Transports	Languages	Licensing
Carriots (carriots.com)	PaaS	Monolithic (over the air [OTA])	REST	HTTP	Groovy	Proprietary
Exosite (exosite.com)	SaaS	Monolithic (OTA)	REST	HTTP	-	Mixed
GroveStreams (grovestreams.com)	PaaS	–	REST	HTTP	Java, Python	Proprietary
realTime.io (realtime.io)	SaaS	Monolithic (OTA)	REST	HTTP	-	Proprietary
SensorCloud (sensorcloud.com)	PaaS	–	REST	HTTP	C#, Java	Proprietary
TempoIQ (tempoiq.com)	PaaS	–	REST	HTTP	Many	Proprietary
ThingWorx (thingworx .com)	PaaS	Monolithic (patches)	REST	HTTP, MQ Telemetry Transport (MQTT)	Java, C	Proprietary
WoTkit (wotkit. readthedocs.io)	SaaS	–	REST	HTTP	–	Proprietary
Xively (xively.com)	PaaS	Monolithic (OTA)	REST	HTTP	Many	Mixed
Lelylan (lelylan.com)	PaaS	–	REST, MQTT	HTTP, WebSockets	Ruby, Node.js	Open source
Amazon AWS IoT (aws .amazon.com/iot)	PaaS	Monolithic (OTA)	REST, MQTT	HTTP, WebSockets	Many	Proprietary
IBM Bluemix/Watson IoT (internetofthings.ibmcloud .com)	PaaS	Monolithic	REST, MQTT	HTTP	Node.js	Mixed
Google IoT (cloud.google .com/solutions/iot)	PaaS	Monolithic (OTA)	REST, gRPC (Google Remote Procedure Call)	HTTP	Many	Proprietary
FaceBook Parse (parseplatform.github.io)	PaaS	–	REST	HTTP	Many	Open source
Azure IoT Hub (azure.microsoft.com)	PaaS	Monolithic (OTA)	REST, MQTT, Advanced Message Queuing Protocol (AMQP)	HTTP	Many	Proprietary
Kinvey (kinvey.com)	PaaS	Software developer's kit (runtime)	REST	HTTP	Node.js	Proprietary
DreamFactory (dreamfactory.com)	PaaS	–	REST, MQTT	HTTP	Many	Open source
CloudFoundry (cloudfoundry.org)	PaaS	–	REST	HTTP	Many	Open source
Heroku (heroku.com)	PaaS	–	MQTT	TCP	Many	Proprietary
Stack4Things (stack4things.unime.it)	IaaS/ PaaS	Drivers (runtime)	REST, Web Application Messaging Protocol, file	HTTP, WebSockets	Any	Open source

* PaaS: platform as a service, SaaS: software as a service, IaaS: infrastructure as a service

(INFOCOM WKSHPS 16), 2016, pp. 848–853.
4. F. Longo et al., "Stack4Things: An OpenStack-Based Framework for IoT," *Proc. 3rd Int'l Conf. Future Internet of Things and Cloud* (FiCloud 15), 2015, pp. 205–211.
5. T. Oberstein and A. Goedde, "The Web Application Messaging Protocol," IETF working draft, Oct. 2015; tools.ietf.org/id/draft-oberstet-hybi-tavendo-wamp-02.html.
6. I. Fette and A. Melnikov, *The WebSocket Protocol*, IETF RFC 6455, Dec. 2011; www.rfc-editor.org/rfc/rfc6455.txt.
7. T. Pflanzner and A. Kertesz, "A Survey of IoT Cloud Providers," *Proc. 39th Int'l Convention on Information and Communication Technology, Electronics and Microelectronics* (MIPRO

16), 2016, pp. 730–735.
8. M. Dìaz, C. Martín, and B. Rubio, “State-of-the-Art, Challenges, and Open Issues in the Integration of Internet of Things and Cloud Computing,” *J. Network and Computer Applications*, vol. 67, May 2016, pp. 99–117.
9. A. Botta et al., “Integration of Cloud Computing and Internet of Things: A Survey,” *Future Generation Computer Systems*, vol. 56, Mar. 2016, pp. 684–700.

ABOUT THE AUTHORS

DARIO BRUNEO is associate professor of embedded systems at the University of Messina. His research interests include the Internet of Things (IoT), smart cities, cloud computing, and computer system performance evaluations. Bruneo received a PhD in computer engineering from the University of Messina. He is a member of the IEEE Computer Society. Contact him at dbruneo@unime.it.

SALVATORE DISTEFANO is associate professor at the University of Messina and fellow professor at Kazan Federal University. His research interests include cloud computing, IoT, crowdsourcing, big data, and quality of service. Distefano received a PhD in computer engineering from the University of Messina. Contact him at sdistefano@unime.it.

FRANCESCO LONGO is assistant professor at the University of Messina. His research interests include distributed system performance evaluations, virtualization of IoT systems, and machine learning applied to smart environments. Longo received a PhD in computer engineering from the University of Messina. Contact him at flongo@unime.it.

GIOVANNI MERLINO is research fellow in computer engineering at the University of Messina. His research interests include edge computing, IoT, network virtualization, smart sensing environments, and crowdsensing. Merlino received an international PhD in computer and telecommunications engineering from the University of Catania. He is a member of the IEEE Computer Society. Contact him at gmerlino@unime.it.

ANTONIO PULIAFITO is full professor of computer engineering and director of the Center for Information Technologies at the University of Messina. His research interests include parallel and distributed systems, networking, and wireless and cloud computing. Puliafito received a PhD in electronics and computer science from the University of Palermo. He is a member of the IEEE Computer Society. Contact him at apuliafito@unime.it.