# Osmotic Flow: Osmotic Computing + IoT Workflow

**Matteo Nardelli**
University of Rome Tor Vergata

**Stefan Nastic and Schahram Dustdar**
TU Wien

**Massimo Villari**
University of Messina

**Rajiv Ranjan**
Newcastle University

The rapid evolution of Internet of Things (IoT) devices (e.g., sensors and gateways) and the almost ubiquitous connectivity (e.g., 4G, Wi-Fi, RFID/NFC, Bluetooth, IEEE 802.15.4) are forcing us to radically rethink how to effectively deal with massive volume, velocity, and variety of big data produced by such IoT devices. There are currently 6.4 billion IoT devices in use around the world and their number, capabilities, as well as the scope of their use, keeps growing rapidly. According to Gartner (http://www.gartner.com/newsroom/id/3165317) the number of IoT devices will reach 20.8 billion by 2020, and, by then, IoT service spending will reach $1,534 billion and hardware spending $1,477 billion.

As IoT expands into various application domains such as healthcare, utility grids, cities, agriculture, transportation, industry 4.0, and disaster management, need for investigating on-the-fly computation over the IoT data streams is ever more pressing. Indeed, most IoT applications are modeled as data transformation workflows that consists of: i) multiple interdependent, heterogeneous data analysis computational and programming models that realise various data transformation tasks from data ingestion to analysis, ii) virtualised/non-virtualised computational and network infrastructure, iii) communication media of various kinds (including wireless). Currently, powerful Cloud Datacentres (CDCs, e.g. AWS[1]) provide computation and data storage resources for IoT workflows, but they suffer from limited bandwidth and network latency, and support neither latency-sensitive applications nor applications that rely heavily on the data streaming from IoT data sources for computing intelligence in real-time (in the form of data ingestion and data analysis).

A possible solution to augment the scalability of CDCs lies in taking advantage of the ever-increasing computational and storage capabilities available at the network edge.[2,3,4] We note in the previous instalment of "Blue Skies" sensing and networking devices available at the network edge constitute a new type of computing infrastructure, the Edge Datacentre (EDC).[5] An EDC may vary in scope and capability, including gateways (Raspberry Pi 3, UDOO board, esp8266, Meshlium Xtreme, Arduino), software defined network solutions (e.g. Cisco IOx), or smart phones equipped with sensors. To facilitate highly distributed and federated computing environments, we proposed Osmotic Computing paradigm[5] that enables the automatic deployment of microservices over inter-connected EDC and CDC. The benefits of integrating EDC and CDC has already been recognised by several companies and open source initiatives, in-

2325-6095/17/$33.00 © 2017 IEEE

cluding CISCO, AWS[1], and Google[3], and the OpenFog Consortium.[4] For example, AWS has enriched its CDC offerings with near-edge computing and storage capabilities (i.e., Snowball Edge, Greengrass).

Nevertheless, the usage of an Osmotic Computing infrastructure (CDC+EDC) poses new challenges for IoT workflow application developers and operations managers as they need the awareness of resource/device (CDC server vs. IoT gateway) heterogeneity, virtualisation software heterogeneity (e.g., hypervisor vs. container), data analytic programming model heterogeneity (stream processing vs. batch processing), geographic distribution, and network performance uncertainties.

Existing streaming data analysis platforms including (e.g., Spark[6], Heron[7], Google Dataflow[8], AWS, Kinesis[1], StreamCloud, Apache Storm), are CDC-centric, hence they do not meet the resource management and scheduling requirements for IoT workflows that require coordinated mapping for data analysis activities to both CDC and EDC. Many workflow application management platforms such as Pegasus, Triana, Taverna, Galaxy, e-Science Central, and Kepler support the development, deployment and execution of scientific workflow applications on CDC without considering newly evolved EDC capabilities. Apache Oozie and Linkedin Azkaban support a Hadoop workflow, but in a rather rigid manner that works well for only batch processing activities. Data analytics platforms such as YARN, Mesos  Amazon IoT and Google Cloud Dataflow can support manual provisioning of multiple data transformation tasks on CDC resources, but only in a performance-agnostic way.

## The Osmotic Flow Model

We propose *Osmotic Flow*, a new model for holistically programming, mapping and executing IoT data transformation workflow applications on a distributed infrastructure combining both EDC and CDC resources. In the Osmotic Flow model, an IoT workflow application is modelled as a directed (potentially cyclic) graph with data transformation tasks as its nodes, and dataflow dependencies (or control flow dependencies for computational synchronization, if/as needed) between data transformation tasks as its vertices. Osmotic Flow model permits data transformation tasks to be distributed, managed, and executed across any available CDC and EDC provider. A data transformation task encapsulates a microservice (e.g., Docker, Unikernel), a computational model (e.g. statistics, clustering, classification, anomaly detection, accumulation), and a data analysis programming model (e.g., stream processing, batch processing, SQL, NoSQL, data ingestion).

### Motivation

Let us consider a contemporary Smart City, where a plethora of IoT sensing devices with Internet connectivity are disseminated all over the urban environment. Buses, trains, and taxis continuously communicate their position; vehicles notify congested routes; citizens often geo-locate their position in messages, photos, videos, or accessing specific services.

All these IoT data sources continuously produce ever-increasing streams of data that can be collected and processed to get the so-called “pulse of the city”, thus fostering awareness and capacity of taking informed decisions. Intelligent services aimed at improving the citizens’ quality of life can be built by merging, filtering, correlating, and transforming these diverse data streams.

For example, a smart traffic light IoT application (see Figure 1) can identify traffic congestions and proactively change traffic light priorities and speed limits, so to reduce ripple effects and relieve the environmental impact. Let us focus on a single road divided in multiple segments and managed by traffic light. The traffic light is instrumented with appropriate IoT sensor (e.g., light state sensor, CCTV) and actuator. The traffic light sensor produce data streams about their current state (i.e., color of light turned on and light change timings) while the CCTV (traffic congestion indicator) produces visual evidence of congestion. The smart traffic light IoT application assumes that traffic congestion is directly proportional to the number of cars queued at each intersection and inversely proportional to the average speed per road segment, each road segment is equipped with above IoT sensor and actuator. The First analytic task merges the data streams from the light sensors and CCTV sensors to develop awareness on traffic congestion across the dependent road segments. The second analytic task combines output from first analytic tasks with appropriate traffic flow computational model to develop aggregated knowledge of traffic flow and congestion across the segments. The traffic

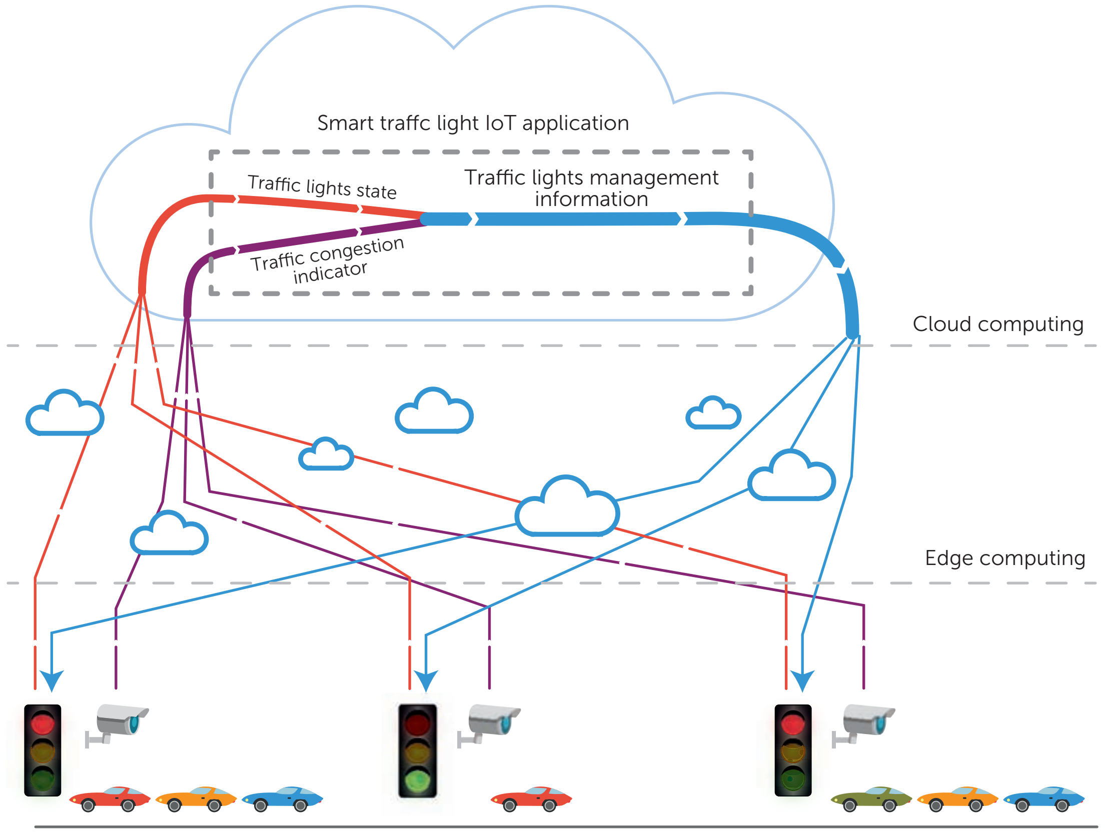
Smart traffc light IoT application
Traffic lights state
Traffic lights management information
Traffic congestion indicator
Cloud computing
Edge computing

FIGURE 1. A high-level description of the smart traffic light IoT workflow application

flow computational model dynamically emits the traffic light control commands to the actuators about the switching off/on of the traffic lights across the segments such as that it leads to optimal traffic flow.

The classic approach for realising this kind of IoT workflow application (see Figure 1) relies exclusively on CDC resources, which could be distant from data sources, hence leading to excessive event detection (e.g., traffic congestion) delay. For example, the first analytic step of the traffic light sensor data aggregation should be mapped to nearby EDC resource, while resource-intensive second analytic task should be mapped to CDC resource, as it needs to execute complex traffic flow computational model.

## Design Goals of the Osmotic Flow model

We identify the key requirements that drive the design of Osmotic Flow programming model.

**Scalability and Elasticity.** Due to the huge amount of data that will be processed in a real-time fashion, scalability represent a key design requirement for IoT workflow applications. For example, a recent analysis of a (single) healthcare-related IoT workflow application (with 30 million of users) showed data flows up to 25,000 tuples per second.[10] The Osmotic Flow model should consider scalability and elasticity by design, so that applications can automatically grow and shrink based on data volume and velocity.

**Focus on data transformation.** The IoT application providers, who wants to realise a data transformation workflow, desires to focus only on data transformation, without spending too much time on management or configuration operations. Hence, the Osmotic Flow model should enable an easy deployment interface where data transformations should be easily defined and, at the same time, each transformation should be seamlessly mapped to either EDC or CDC based on performance needs. As a consequence, the deployment process is transparently performed by the underlying run-time engine. However, the application providers can customize the framework behavior so to better address his/her specific performance needs.

**Efficient composition of data transformations.** The Osmotic Flow model should support the composition of cross-workflow data transformations and linking, so to easily realise complex workflows. To this end, Osmotic Flow should consider by design the possibility of composing data streams coming from multiple, public IoT devices and applications, thus promoting the principle of sharing and reusability. Our Osmotic Flow model should allow the application provider to easily define new streams, which extract high value information from raw data, without worrying about low level concerns related to their runtime execution, such as resource allocation, streams deployment, elasticity, and governance.

**Network Awareness.** The emerging IoT environment calls for strong network awareness. The Osmotic Flow model should not neglect the presence of communication delays while performing the deployment of data transformation tasks to CDC and/or EDC.

### Main Entities in the Osmotic Flow model

In Osmotic Flow, as depicted in Figure 2, includes one or more input endpoint that receives data from an external data sources and one or more output endpoints, which emit processed or transformed data towards sinks or other streams. A workflow is characterised by the following elements:

- one or more *data sources*: a data source is an entity, potentially external to the system, that continuously generates events or data. For example, a data source can be an IoT device emitting temperature measurements or traffic congestion conditions.
- one or more *sinks*: a sink is a final endpoint in the data transformation flow (or pipelne). Interested parties can subscribe to the sink for receiving notification information (e.g., traffic congestion information requested by drivers in Figure 1).

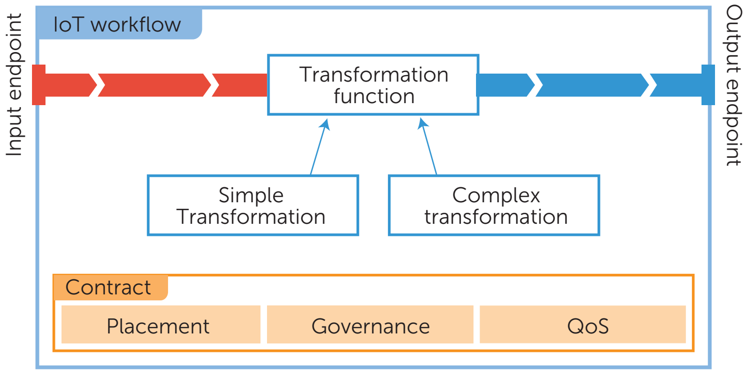

FIGURE 2. Depiction of IoT Transformation Functions in the Osmotic Flow model

- a *transformation function or task*: it encapsulates the user-defined analytics logic which transforms (e.g., combines, filters, splits) incoming data streams and passes the results to next transformation functions or final sinks of the workflow.
- a *contract*: it is a high-level configuration and performance requirement descriptor of the transformation functions or tasks.

### Types of IoT Data Streams

An IoT data stream can be ephemeral or public. An *ephemeral stream* is a special kind of stream that exists only if a sink is (directly or indirectly) interested to incoming flow. An indirect interest is manifest when one or more streams lie in between the stream and the final destination is the workflow endpoint (i.e., sink). Being ephemeral, the existence of the stream depends on the presence of (direct or indirect) interested sinks and its scope is restricted within the same application, i.e., it can be used only by user-defined transformations running within the same application that contains the stream. A public stream is a globally available stream and, as such, can be part of more than one IoT workflow application (for example traffic pattern stream data can be used for smart traffic management applications as well as air pollution monitoring applications).

### Types of Transformation Tasks

Transformation functions are the only piece of code that has to be defined by the application providers. A transformation function encapsulates the data analytics logic. For an efficient execution, the stream model requires transformation functions to be as scalable as possible, therefore the latter are either stateless or provide an explicit definition of their state. We distinguish between two kind of transformations: simple and batch.

A *simple transformation* is applied to every incoming data in parallel and produces zero, one, or more outgoing data. For example, a simple transformation can be realised using only one type of data analysis programming model. An example of simple transformation (in context of Figure 1) could include tracking vehicles that exceed speed limit.

A *complex transformation* fuses one or more data streams before applying a transformation using one or more data analysis programming models (e.g., stream processing, batch processing, NoSQL). Complex transformation can produce zero, one, or more outgoing data streams. The group of incoming data streams fully determine the function state, which can then be manipulated by the transformation. A flow of data can be determined according to two modes: window and window-and-key. A window-based transformation creates a time-based or count-based window of events that have to be combined before running the transformation. For example, a window-based transformation can computes statistics on traffic patterns (see Figure 1) on a road segment in the last 30 seconds. On the other hand, a window-and-key transformation is a special case of windowed transformation that has a finer granularity in selecting the data streams for fusion. A classic example for a window-and-key transformation is the implementation of the vehicle counter task, which computes statistics on how many times a particular vehicle has travelled across a road-segment in last hour/week/month. Hence, the transformation across both historical and real-time data is dependent on the same key (i.e., vehicle registration number).

Simple transformations are stateless function, whereas complex transformations provide an explicit definition of their current state (real-time data) as well as it depends on the past state (historical data).

### Contract

The contract provides a high-level description of the IoT workflow application's configuration including:

- Placement constraints: these constraints guide the placement of transformation tasks over the distributed CDC+EDC infrastructure. If no restrictions are provided, the run-time execution engine can deploy a transformation task either on CDCs or on EDCs. Placement constraints be included to, e.g., maximize the utilization of nearby EDC resources or exploit centralized Cloud resources.
- Governance constraints: together with the previous one, the governance rules enable to specify further restrictions regarding the transformation task deployment and adaptation. These restrictions are often related to security, privacy, or law concerns. For example, a governance rule can exclude every edge resource belonging to a specific geographical region or can require to encrypt the exchanged data, so to meet stringent law restrictions.
- QoS or performance constraints: these ones express non-functional properties that should be met during the stream execution, so to obtain a desired quality level. For example, constrains can bound the maximum stream latency or minimum stream throughput or the event detection accuracy.

## Osmotic Flow Scheduling Architecture and Research Issues

Figure 3 provides a system-level description of the Osmotic Flow model. Whenever an IoT application provider wants to execute a workflow of data transformations, he/she submits the application code to the nearest Osmotic Resource Manager using a submission client. Then, the Osmotic Resource Manager allocates a new Node Manager, which, in turn, first determines the application placement, governance, and QoS constraints, and then distributes the data transformation tasks to appropriate EDC and/or CDC resources. The main software components include

**Osmotic Resource Manager (ORM).** An Osmotic Resource Manager coordinates Edge and Cloud resources and supports the Node Manager in determining the placement of Osmotic Flow transformation functions. In the proposed model, multiple

ORM can coexist and cooperate, where each one coordinates a pool of nearby CDC/EDC resources. The federation of multiple ORM enables the deployment of applications on the combined (EDC+CDC) infrastructure.

To ensure scalable communication and coordination between ORMs, future research should focus on developing self-healing load coordination protocols that can cope with changes in the infrastructures and IoT device state, and that can dynamically adapt to failures, connections, and detachments of ORMs and EDC/CDC resources. Another research thread could be to develop cooperative and opportunistic workload coordination protocol such that ORMs are able to balance their workload with each other in order to make sure that no CDC/EDC resource are wasted due to redundant data streams. Several workload coordination solutions already exist for CDC environments (e.g., Quincy, Omega, Sparrow, Mercury),[11] nevertheless the features of this new environment (CDC+EDC), as well as the characteristics of the Osmotic Flow model, foster the development of new load coordination policies and protocols, tailored for the specific setting where a significant heterogeneity of resources as well as multiple of data transformation tasks has to be management.

**Universal Stream Repository (USR).** To enable the sharing and reuse of high-value streams, Osmotic Flow includes a Universal Stream Repository (USR), which collects and provides the descriptor of every public stream available in the Osmotic Flow ecosystem. An IoT application provider relies on the descriptor to discovery existing streams and reuse them within his/her application.

Therefore the future research should focus on developing holistic data model for expressing EDC/CDC resources and data stream characteristics using an ontology-based representation, which enables encoding both dynamic (e.g., performance, status of stream) and static (e.g., functionality provided, types of events) QoS parameters. The ontology will thus guide decisions made on the types of data transformation tasks that are deemed most suitable for the deployment in the CDC or at the network Edge. An open access USR should be built. Existing ontologies such as Semanrtic Sensor Network (SSN) can de-

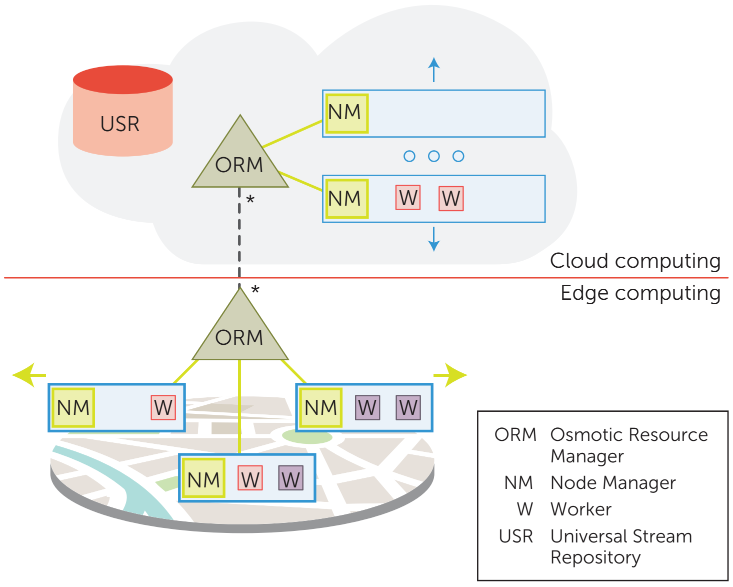

FIGURE 3. Scheduling Architecture of Osmotic Flow Model

scribe concepts as IoT sensor characteristics, or data formats; nevertheless, they are not suitable to capture characteristics relevant to CDC/EDC resources.

**Node Manager.** The Node Manager is a per-machine agent that supports the ORM in controlling the available resources of the EDC/CDC infrastructure. Besides launching and terminating the execution of workers, the Node Manager monitors and reports statistical information about resource utilization (i.e., CPU, memory, network) to the ORM. Moreover, the Node Manager provides information to the ORM for determining the communication delays of the node with the other components of the infrastructure. Observe that communication delays can be obtained by means of either active/passive measurements (e.g., with a network coordinate system[12]), or with some network support (e.g., SDN).

Though Simple Network Management Protocol (SNMP), using the Management Information Base (MIB), has been highly adopted for monitoring resources in CDC, it lacks the ability to monitor EDC resources (as identified by the Internet Engineering Task Force[13]) due to huge computational overhead and the constrained nature of Edge resources. Hence, future research will need to investigate modeling of a novel Edge resource monitoring agent that harnesses lightweight IoT protocols such as Constrained Application Protocol and a new Edge resource-specific MIB interface provided by the Internet Engineering Task Force.[13]

**Worker.** The Worker is in charge of executing one or more transformation functions. To this end, a worker collects data from the stream data source (i.e., another worker or an external data source), runs the user-defined transformation function, and emits outgoing streams. In other words, the worker takes care of the distributed execution of the user code, that defines only how to manipulate input data to obtain output data. Since streams are executed by workers, they directly communicate to transfer data up to the final consumers. Being stateless or with a window-based state, multiple transformation functions can run concurrently in a worker, and multiple workers can run concurrently on the infrastructure. Moreover, since a transformation function is defined with a fine granularity (i.e., per event or per window), it can be transparently scaled as the number of incoming events increases or decreases, up to—theoretically—creating an instance per each event.

As Osmotic Flow model thrives to support multiple type and mix of data transformation tasks on shared EDC+CDC infrastructures, the Worker need to be equipped with scheduling intelligence to automatically discover and resolve contention between co-deployed data transformation tasks. During deployment of data transformation tasks, the Worker must consider which data transformation tasks should be combined on an EDC and/or CDC resource, to minimize resource contention due to workload interference. Workload resource consumption and QoS are not additive, so understanding the nature of their composition is critical to deciding which transformation tasks can be deployed together. Existing content detection approaches such as Paragon[14] that applies collaborative filtering techniques for resolving contention between co-deployed, hypervisor-based application workloads on CDC are agnostic to the new hardware (e.g. Raspberry, Pi 3, UDOO board, Cisco IOx) and virtualisation features (e.g., Containers, Unikernels) of EDC resources.

Till data, several data analytic programming models and frameworks have been proposed. Nevertheless, most of them are designed to run in CDC, thus neglecting the presence of EDC resources. Osmotic Flow builds on the strengths of existing solutions and creates a novel approach for executing data analytics in a Cloud-supported Edge environment. Similar to Google Cloud Dataflow[8] and Apache Spark, Osmotic Flow defines a very simple and scalable programming model that enables to automatically deploy transformations with a high degree of parallelism. However, differently from existing approaches, Osmotic Flow focuses mainly on processing continuous and unbounded streams of data on decentralized CDC+EDC resources. Moreover, since the Osmotic Flow includes resource management capabilities, it can optimize how Edge and Cloud nodes are allocated among multiple and concurrent data transformation functions.

## References

1. Amazon Web Services, https://aws.amazon.com
2. F. Bonomi, R. Milito, J. Zhu, and S. Addepalli. Fog computing and its role in the Internet of Things. In *Proc. of MCC '12*, pages 13–16. ACM, 2012.
3. Google Edge Network, https://peering.google.com.
4. OpenFog Consortium, https://www.openfog consortium.org.
5. M. Villari, M. Fazio, S. Dustdar, O. Rana and R. Ranjan, "Osmotic Computing: A New Paradigm for Edge/Cloud Integration," in *IEEE Cloud Computing*, vol. 3, no. 6, pp. 76-83, Nov.-Dec. 2016. doi: 10.1109/MCC.2016.124.
6. M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica. "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing". In *Proc. of USENIX NSDI '12*, 2012.
7. S. Kulkarni, N. Bhagat, M. Fu, V. Kedigehalli, C. Kellogg, S. Mittal, J. M. Patel, K. Ramasamy, and S. Taneja. "Twitter Heron: Stream Processing at Scale". In *Proc. of SIGMOD '15*, 2015.
8. Google Cloud Dataflow, https://cloud.google.com/dataflow/
9. V. Gulisano, R. Jiménez-Peris, M. Patiño-Martínez, C. Soriente and P. Valduriez, "StreamCloud: An Elastic and Scalable Data Streaming

System," in *IEEE Transactions on Parallel and Distributed Systems*, vol. 23, no. 12, pp. 2351-2365, Dec. 2012.

10. R. Cortés, X. Bonnaire, O. Marin, and P. Sens, "Stream processing of healthcare sensor data: studying user traces to identify challenges from a big data perspective", *Procedia Computer Science*, vol. 52, 2015, pp. 1004-1009, 2015.
11. P. Pietzuch, J. Ledlie, J. Shneidman, M. Roussopoulos, M. Welsh, and M. Seltzer "Network-aware operator placement for stream-processing systems". In *Proc. of IEEE ICDE '06*, 2006.
12. F. Dabek, R. Cox, F. Kaashoek, and R. Morris. "Vivaldi: A decentralized network coordinate system". *SIGCOMM Comput. Commun.* Rev., 34(4), 2004.
13. P. V. D. Stok et al.,"CoAP Management Interfaces," October 2016, IETF Internet-Draft Work-in-Progress, https://datatracker.ietf.org/doc/draft-vanderstok-core-comi/.
14. C. Delimitrou and C. Kozyrakis. 2013. Paragon: QoS-aware scheduling for heterogeneous datacenters. *SIGPLAN Not.* 48, 4 (March 2013), 77-88. DOI=http://dx.doi.org/10.1145/2499368.2451125.
15. D. Puthal, S. Nepal, R. Ranjan and J. Chen, "Threats to Networking Cloud and Edge Datacenters in the Internet of Things", *IEEE Cloud Computing*, vol. 3, no. 3, pp. 64–71, 2016.

**MATTEO NARDELLI** *is a PhD student of computer science at the University of Rome Tor Vergata, Italy. His research interests are in the field of distributed computer systems, with a focus on the execution of data stream applications in geographically distributed environments. Contact him at nardelli@ing.uniroma2.it.*

**STEFAN NASTIC** *is a Postdoctoral Research Assistant at the Distributed Systems Group at TU Wien, Austria. His research interests include: Internet of Things and Edge Computing; Cloud Computing; Big Data Analytics; and Smart Cities. He received a PhD in software engineering and Internet technologies from TU Wien. Nastic has been involved in several EU-funded research projects such as SMART-FI, U-Test and SM4ALL, as well as, large industrial projects such as Pacific Controls Cloud Computing Lab (PC3L). Contact him at snastic@infosys.tuwien.ac.at.*

**SCHAHRAM DUSTDAR** *is a full professor of computer science heading the Distributed Systems Group at TU Wien, Austria. His work focuses on Internet technologies. He is an IEEE Fellow, a member of the Academy Europeana, and an ACM Distinguished Scientist. Contact him at dustdar@dsg.tuwien.ac.at or dsg.tuwien.ac.at.*

**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*

**RAJIV RANJAN** *is a reader in the School of Computing Science at Newcastle University, UK; chair professor in the School of Camputer, Chinese University of Geoscience, 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.*