A Graph-Based Cloud Architecture for Big Stream Real-Time Applications in the Internet of Things

Laura Belli(✉), Simone Cirani, Gianluigi Ferrari, Lorenzo Melegari, and Marco Picone

Wireless Ad-Hoc and Sensor Network Laboratory, Department of Information Engineering, University of Parma, Viale G.P. Usberti, 181/A, 43124 Parma, Italy
laura.belli1@studenti.unipr.it,
{simone.cirani,gianluigi.ferrari,marco.picone}@unipr.it,
lorenzo.melegari@tlc.unipr.it

Abstract. The Internet of Things (IoT) will consist of billions of interconnected heterogeneous devices denoted as *"smart objects."* Smart objects are generally sensor/actuator-equipped and have constrained resources in terms of: (i) processing capabilities; (ii) available ROM/RAM; and (iii) communication reliability. To meet low-latency requirements, real-time IoT applications must rely on specific architectures designed in order to handle and process gigantic (in terms of number of sources of information and rate of received data) streams of data coming from smart objects. We refer to this smart object-generated data stream as *"Big Stream,"* in contrast to traditional "Big Data" scenarios, where real-time constraints are not considered. In this paper, we propose a novel Cloud architecture for Big Stream applications that can efficiently handle data coming from deployed smart objects through a graph-based processing platform and deliver processed data to consumer applications with lowest latency.

Keywords: Internet of Things · Cloud · Real-time applications · Processing graph

1 Introduction

The actors involved in IoT scenarios will have extremely heterogeneous characteristics, (in terms of processing and communication capabilities, energy supply and consumption, availability, and mobility), spanning from constrained devices, also denoted as "smart objects," to smartphones and other personal devices, Internet hosts, and the Cloud. Smart objects are typically equipped with sensors and/or actuators and are thus capable to perceive and act on the environment where they are deployed. Billions of smart objects are expected to be deployed in urban, home, industrial, and rural scenarios, in order to collect relevant information, which may be used to build new applications. Shared and interoperable communication mechanisms and protocols are currently being defined and standardized, allowing heterogeneous nodes to efficiently communicate with each other and with existing Internet actors. The most prominent driver for interoperability in the IoT is the adoption of the Internet Protocol (IP), namely IPv6 [1,2]. An IP-based IoT will be able to extend and interoperate seamlessly with the existing Internet. Standardization institutions, such as the Internet Engineering Task Force (IETF) [3], and several research projects [4] are in the process of defining mechanisms to bring IP to smart objects, due to the need to adapt higher-layer protocols to constrained environments. However, not all objects will be supporting IP, as there will always be tiny devices that will be organized in closed/proprietary networks and rely on very simple and application-specific communication protocols. These networks will eventually connect to the Internet through a gateway/border router.

Sensed data are typically collected and sent uplink, namely from IoT networks, where smart objects are deployed, to collection environments (server or cloud), possibly through an intermediate local network element, which may perform some processing tasks, such as data aggregation and protocol translation. Processing and storing data at the edge of networks (e.g., on set-top-boxes or access points) is the basis for the evolution of Fog Computing [5] in IoT scenarios. Fog Computing is a novel paradigm that aims at extending Cloud computing and services to the edge of the network and, therefore, leverages on its proximity to end-users, its dense geographical distribution, and its support for mobility. The wide geographical distribution makes the Fog Computing paradigm particularly suited to real-time big data analytics. Densely distributed data collection points allow to add a fourth axis, low-latency, to the typical Big Data dimensions (volume, variety, and velocity). Figure 1 shows the hierarchy of layers involved in data collection, processing, and distribution in IoT scenarios.

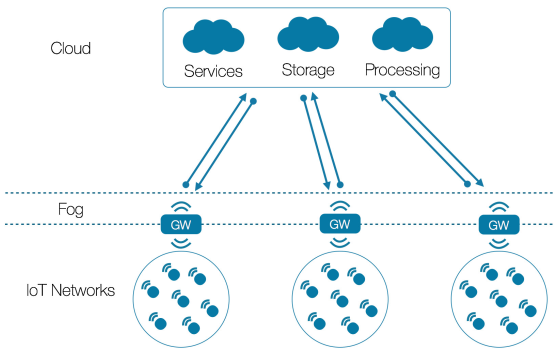

Fig. 1. The hierarchy of layers involved in IoT scenarios: the Fog works as an extension of the Cloud to the network edge to support data collection, processing, and distribution.

With billions of nodes capable of gathering data and generating information, the availability of efficient and scalable mechanisms for collecting, processing, and storing data is crucial. Big Data techniques, which were developed in the last few years (and became popular due to the evolution of online and social/crowd services), address the need to process extremely large amounts of heterogeneous data for multiple purposes. These techniques have been designed mainly to deal with huge volumes (focusing on the amount of data itself), rather than to provide real-time processing and dispatching. Cloud computing has found a direct application with Big Data analysis due to its scalability, robustness, and cost-effectiveness. The number of data sources, on one side, and the subsequent frequency of incoming data, on the other side, create a new need for Cloud architectures to handle such massive flows of information, thus shifting the Big Data paradigm to the Big Stream paradigm. Moreover, the processing and storage functions implemented by remote Cloud-based collectors are the enablers for their core business, which involves providing services based on the collected/processed data to external consumers. Several relevant IoT scenarios, (such as industrial automation, transportation, networks of sensors and actuators), require real-time/predictable latency and could even change their requirements (e.g., in terms of data sources) dynamically and abruptly. Big Stream-oriented systems could react effectively to changes and provide smart behavior for allocating resources, thus implementing scalable and cost-effective Cloud services. Dynamism and real-time requirements are another reason why Big Data approaches, due to their intrinsic inertia (i.e., Big Data typically works with batch-based processing), are not suitable for many IoT scenarios.

The main differences between Big Data and Big Stream paradigms are (i) the nature of data sources and (ii) the real-time/latency requirements of consumers. The Big Stream paradigm allows to perform real-time and ad-hoc processing in order to link incoming streams of data to consumers, with a high degree of scalability, fine-grained and dynamic configuration, and management of heterogeneous data formats. In brief, while both Big Data and Big Stream deal with massive amounts of data, the former focuses on the analysis of data, while the latter focuses on the management of flows of data, as shown in Fig. 2.

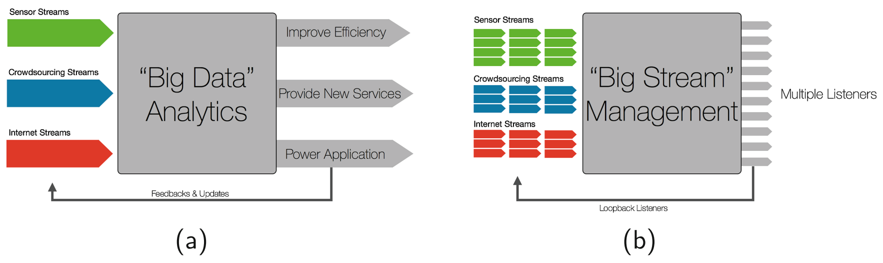

Fig. 2. (a) Data sources in Big Data systems. (b) The multiple data sources and listeners management in Big Stream system.

The difference is in the meaning of the term "Big," which refers to volume of data for Big Data and to global information generation rate of the data sources for Big Stream. This has an impact also on the data that are considered relevant to consumer applications. For instance, while for Big Data applications it is important to keep all sensed data in order to be able to perform any required computation, Big Stream applications might decide to perform data aggregation or pruning in order to minimize the latency in conveying the results of computation to consumers, with no need for persistence. Note that, as a generalization, Big Data applications might be consumers of Big Stream data flows.

For these reasons, in this paper we propose an architecture targeting Cloud-based applications with real-time constraints, i.e., Big Stream applications, for IoT scenarios. The proposed architecture relies on the concepts of *data listener* and *data-oriented processing graph* in order to implement a scalable, highly configurable, and dynamic chain of computations on incoming Big Streams and to dispatch data with a push-based approach, thus providing the lowest delay between the generation of information and its consumption.

The rest of this work is organized as follows. In Sect. 2, an overview of related works is presented. In Sect. 3, the proposed architecture is presented and detailed. Finally, in Sect. 4 we draw our conclusions and discuss future research directions.

2 Related Work

The IoT paradigm consists of a huge number of different and heterogeneous smart objects connected in a "Network of Networks." These nodes are envisioned as collectors of information from the environment in order to provide useful services to users. This ubiquitous sensing, enabled by the IoT in most areas of modern living, has led to information and communication systems invisibly embedded in the environment, thus making the technology disappear from the consciousness of the users. The outcome of this trend is the generation of a huge amount of data that, depending on the application scenario, should be aggregated, processed, stored, transformed, and delivered to the final users of the system, in an efficient and effective way, with traditional commodity services.

2.1 IoT Architectures

A large number of architectures for IoT scenarios have been proposed in the literature. Among these, many consider Cloud computing as the infrastructure for data and service management for IoT client applications. For instance, most of the outgoing projects on IoT architecture address relevant challenges, particularly from a Wireless Sensor Networks (WSN) perspective. Some examples are given by a few European Union (EU) 7th Framework Program (FP7) projects such as SENSEI [6] and Internet of Things-Architecture (IoT-A) [7].

The purpose of the SENSEI project is to create an open and business-driven architecture that addresses the scalability problems for a large number of globally distributed Wireless Sensor and Actuator (WS&A) networks devices. It provides necessary network and information management services to enable reliable and accurate context information retrieval and interaction with the physical environment. By adding mechanisms for accounting, security, privacy, and trust, SENSEI aims at enabling an open and secure market space for context-awareness and real world interaction.

The IoT-A project team has focused on the design of a IoT architecture, creating an Architectural Reference Model (ARM) with the definition of an initial set of key building blocks. The IoT-A ARM's aim is to connect vertically closed systems, architectures, and application areas in order to create open interoperable systems and integrated environments or platforms. It constitutes a foundation from which software companies can capitalize on the benefits of developing consumer-oriented platforms including hardware, software, and services.

"Connect All IP-based Smart Objects!" (CAL*IP*SO) [4] is another EU FP7 project with relevant implications on the design of an IoT architecture. The main purpose of CAL*IP*SO is to build IoT systems with IPv6-connected smart objects and to look for novel methods to achieve very low power consumption, thus providing both high interoperability and long lifetime. CAL*IP*SO leans on the significant body of work on sensor networks to integrate radio duty cycling and data-centric mechanisms with the IPv6 [2]. The project entails three layers (network, routing, and application) using Contiki [8] open source Operating System (OS), Europe's leading OS for smart objects, as the target development environment for prototyping and experimental evaluation. CAL*IP*SO focuses on three main use cases to drive its research activities: (i) Smart Infrastructures; (ii) Smart Cities; and (iii) Smart Toys applications.

In the work of Gubbi et al. [9], the success of IoT is attributed to three factors:

- shared understanding of the situation of its users and their appliances;
- software architectures and pervasive communication networks to process and convey the contextual information where it is relevant;
- IoT analytics tools aiming at autonomous and smart behaviors.

For the above reasons, the IoT architecture proposed therein is not based on WSNs and the focus is instead on the user and the Cloud. The consumer is the "center" and drives the use of data and infrastructure to develop new applications. The rest of the work discusses the key enabling technologies and the different future applications domains, describes a Cloud-centric architecture for IoT, and presents an implementation based on Aneka [10], a .NET-based application development Platform-as-a-Service (PaaS), which can utilize storage and computing resources of both public and private clouds.

Another framework for (IoT-based) Cloud systems is OpenStack [11], a global collaboration of developers producing an ubiquitous open source cloud computing platform for public and private clouds. The project aims to deliver solutions for all types of clouds by being simple to implement, massively scalable, and feature-rich. The technology consists of a series of interrelated projects delivering various components for a cloud infrastructure solution. The result is an open cloud OS that controls large pools of computing, storage, and networking resources. OpenStack can be used as a background to build other platforms. An example of this approach is given by the FI-WARE project [12], an open cloud-based infrastructure for the cost-effective creation and delivery of Internet applications and services. FI-WARE API specifications are public, royalty-free, and OCCI-compliant, driven by the development of an open source reference implementation which allows developers, service providers, enterprises, and other organizations to develop innovative products based on FI-WARE technologies.

While OpenStack can be seen as a framework with a vendor-driven model, OpenNebula [13] is an open-source Cloud platform that focuses efforts on the user. This project aims at delivering a simple feature-rich and flexible solution to build and manage enterprise clouds and virtualized data centers.

2.2 Big Data Processing Pattern

Big Data architectures generally use traditional processing patterns with a pipeline approach [14]. These architectures are typically based on a processing approach where the data flow goes downstream from input to output, to perform specific tasks or reach the target goal. Typically, the information follows a pipeline where data are sequentially handled, with tightly coupled pre-defined processing sub-units (static data routing). The described paradigm can be defined as "process-oriented:" a central coordination point manages the execution of sub-units in a certain order and each sub-unit provides a specific processing output, which is created to be used only within the scope of its own process without the possibility to be shared among different processes. This approach represents a major deviation from traditional Service Oriented Architectures (SOAs), where the sub-units are external web services invoked by a coordinator process rather than internal services [15].

2.3 Fog Computing

In the area of user-driven and Cloud IoT architectures, in [5] Fog computing is proposed as the novel and appropriate paradigm for a variety of IoT services and applications that require mobility support, low latency, and location awareness. The Fog can be described as a highly virtualized platform that provides computing, storage, and networking services between end-devices and traditional Cloud Computing. In other words, the Fog is meant to act as an extension of the Cloud, operating at the edge of the network to support endpoints by providing rich services that can fulfill real-time and low-latency consumers requirements. The Fog paradigm has specific characteristics, which can be summarized as follows:

– geographical distribution, in contrast with the centralization envisioned by the Cloud;
– subscriber model employed by the players in the Fog;
– support for mobility.

The architecture described by Bonomi et al. [5] is based on the Fog and Cloud interplay: the former provides localization, low latency, and context awareness to endpoints; the latter provides global centralization functionalities. In the presented IoT Fog scenario, collectors at the edge of the network ingest the data generated by sensors and devices: the portion of these data that require real-time processing (from milliseconds to tenths of seconds) are consumed locally by the first tier of the Fog. The rest is sent to the higher tiers for operations with less stringent time constraints (from seconds to minutes). The higher the tier, the wider is the geographical coverage and the longer the time scale. As a result, the Fog must support several types of storage: from ephemeral, at the lowest tier, to semi-permanent, at the highest tier. The ultimate and global coverage is provided by the Cloud, which is used as repository for data for a duration of months or years.

2.4 IoT Protocols and Models

As previously stated in Sect. 1, the most prominent driver to provide interoperability in the IoT is IPv6. Referring to the IP stack, at the application layer, the IoT scenario brings a variety of possible protocols that can be employed according to the specific applications requirements. The following are relevant options:

- HyperText Transfer Protocol (HTTP) [16]: mainly used for the communication with the consumer's devices;
- Constrained Application Protocol (CoAP) [17]: explicitly designed to work with a large number of constrained devices operating in LLNs. CoAP is built on the top of User Datagram Protocol (UDP) and follows a request-response paradigm;
- Extensible Messaging and Presence Protocol (XMPP) [18];
- MQ Telemetry Transport (MQTT) protocol [19]: a lightweight publish/subscribe protocol flowing over TCP/IP for remote sensors and control devices through low bandwidth, unreliable, or intermittent communications;
- Constrained Session Initiation Protocol (CoSIP) [20,21]: a version of the Session Initiation Protocol (SIP) [22] aiming at allowing constrained devices to instantiate communication sessions in a lightweight and standard fashion. Session instantiation can include a negotiation phase of some parameters which will be used for all subsequent communication.

Regardless of the selected application-layer protocol, most IoT/M2M applications will follow the REpresentational State Transfer Protocol (REST) architectural model [23], because (i) it provides simple and uniform interfaces and (ii) it is designed to build long-lasting and robust applications, resilient to changes that might occur over time.

3 Architecture

As stated in Sect. 1, a major difference between Big Data and Big Stream approaches resides in the real-time/low-latency requirements of consumers. The gigantic amount of data sources in IoT applications has mistakenly made Cloud-service implementors believe that re-using Big Data-driven architectures would be the right solution for all applications, rather than designing new paradigms specific for IoT scenarios. IoT application scenarios are characterized by a huge number of data sources sending small amounts of information to a collector service at a typically limited rate. Many services can be built upon these data, such as: environmental monitoring, building automation, and smart cities applications. These applications typically have real-time or low-latency requirements in order to provide efficient reactive/proactive behaviors, which could effectively be implemented especially in an IP-based IoT, where smart objects can be directly addressed.

3.1 Big Stream-Oriented Architecture

Applying a traditional Big Data approach for IoT application scenarios might bring to higher or even unpredictable latencies between data generation and its availability to a consumer, since this was not among the main objectives behind the design of Big Data systems. Figure 3 illustrates the time contributions introduced when data pushed by smart objects need to be processed, stored, and then polled by consumers. The total time required by any data to be delivered to a consumer can be expressed as $T = t_0 + t_1 + t_2$, where:

1. t_0 is the time elapsed from the moment a data source sends information, through an available API, to the Cloud service (1) and the service dispatches the data to an appropriate queue, where it can wait for an unpredictable time (2), in order to decouple data acquisition from processing;
2. t_1 is the time needed for data, extracted by the queue, to be pre-processed and stored into a Data Warehouse (DW) (3); this time depends on the number of concurrent processes that need to be executed and get access the common DW and the current size of the DW;
3. t_2 is the data consumption time, which depends on the remaining time that a polling consumer needs to wait before performing the next fetch (4), the time for a request to be sent to the Cloud service (5), the time required for lookup in the DW and post-process the fetched data (6), and the time for the response to be delivered back to the consumer (7).

It can be observed that the architecture described is not optimized to minimize the latency and, therefore, to feed (possibly a large number of) real-time applications, but, rather, to perform data collection and batch processing. Moreover, it is important to understand that data significant for Big Stream applications might be short-lived, since they are to be consumed immediately, while Big Data applications tend to collect and store massive amounts of data for an unpredictable time.

In this work, we propose a novel architecture explicitly designed for the management of Big Stream applications targeting IoT scenarios. The main design criteria of the proposed architecture are: (i) the minimization of the latency in data dispatching to consumers and (ii) the optimization of resource allocation. The main novelty in the proposed architecture is that the data flow is

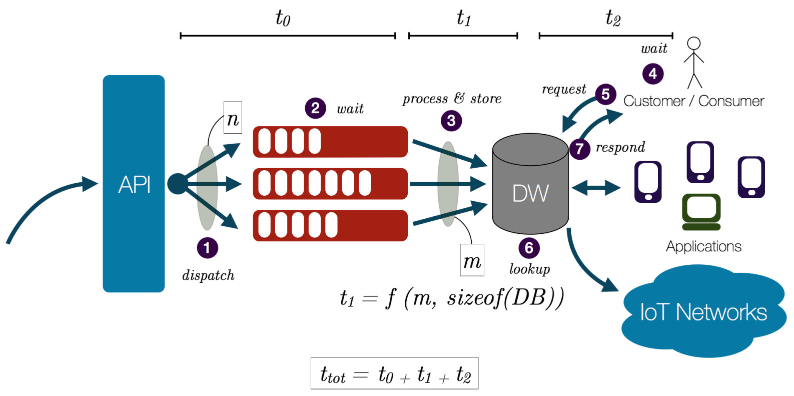

Fig. 3. Traditional Big Data architecture for IoT and delay contributions from data generation to applications information delivery.

"consumer-oriented", rather than being based on the knowledge of collection points (repositories) where data can be retrieved. The data being generated by a deployed smart object might be of interest for some consumer application, denoted as *listener*. A listener registers its interest in receiving updates (either in the form of raw or processed data) coming from a streaming endpoint (i.e., Cloud service). On the basis of application-specific needs, each listener defines a set of rules, which specify what type of data should be selected and the associated filtering operations. For example, in a smart parking application, a mobile app might be interested in receiving contents related only to specific events (e.g., parking sensors status updates, the positions of other cars, weather conditions, etc.) that occur in a given geographical area, in order to accomplish relevant tasks (e.g., find a covered free parking spot). The pseudo-code that can be used to express the set of rules for the smart parking application is shown in the following listing:

```
when
    $temperatureEvent = {@type:http://schema.org/Weather#temperature}
    $humidityEvent = {@type:http://schema.org/Weather#humidity}
    $carPositionEvent = {@type:http://schema.org/SmartCar#travelPosition}
    $parkingStatusEvent = {@type:http://schema.org/SmartParking#status}

    @filter: {
      location: { @type:"http://schema.org/GeoShape#polygon",
      coordinates: [ [
        [41.3983, 2.1729], [41.3986, 2.1729], [41.3986, 2.1734],
        [41.3983, 2.1734], [41.3983, 2.1729]
      ] ]
    }
then
    <application logic>
```

[Pseudo-code to express the set of rules for a smart parking application.]

The set of rules specifies (i) which kinds of events are of interest for the application and (ii) a geography-based filter to apply in order to receive only events related to a specific area. Besides the final listener (end-user), at the same time, the Cloud service might act as a listener and process the same event data stream, but with different rules, in order to provide a new stream (e.g., providing real-time traffic information), which can be consumed by other listeners. An illustrative pseudo-code for a real-time traffic information application is presented in the following listing:

```
when
    $cityZone = {@type:http://schema.org/SmartCity#zone}
    $carPositionEvents = collect({
         @type: http://schema.org/SmartCar#travelPosition,
         @filter: {
                    location: cityZone.coordinates
         }
    }) over window:time(30s)
then
    emit {
         @type: http://schema.org/SmartCity#trafficDensity,
         city_zone: $cityZone,
         density: $carPositionEvents.size,
     }
```

[Pseudo-code to express the set of rules for a real-time traffic information application.]

The proposed Big Stream architecture guarantees that, as soon as data are available, they will be dispatched to the listener, which is thus no longer responsible to poll data, thus minimizing latencies and possibly avoiding network traffic.

The information flow in a listener-based Cloud architecture is shown in Fig. 4. With the new paradigm, the total time required by any data to be delivered to a consumer can be expressed as:

$$T = t_0 + t_1 \tag{1}$$

where:

1. t_0 is the time elapsed from the moment a data source sends information, through an available API, to the Cloud service (1) and the service dispatches the data to an appropriate queue, where it can wait for an unpredictable time (2), in order to decouple the data acquisition from processing;
2. t_1 is the time needed to process data extracted from the queue and be processed (according to the needs of the listener, e.g., to perform format translation) and then deliver it to registered listeners.

It is clear that the inverse of perspective introduced by a listener-oriented communication is optimal in terms of minimization of the time that a listener must wait before it receives data of interest. In order to highlight the benefits brought by the Big Stream approach, with respect to a Big Data approach, consider an alerting application, where an event should be notified to one or more consumers in the shortest possible time. The traditional Big Data approach would require an unnecessary pre-processing/storage/post-processing cycle to be executed before the event could be made available to consumers, which would be responsible to retrieve data by polling. The listener-oriented approach, instead, guarantees that only the needed processing will be performed before data are being delivered directly to the listener, thus providing an effective real-time solution.

This general discussion proves that a consumer-oriented paradigm may be better suited to real-time Big Stream applications, rather than simply reusing existing Big Data architectures, which fit best applications that do not have critical real-time requirements.

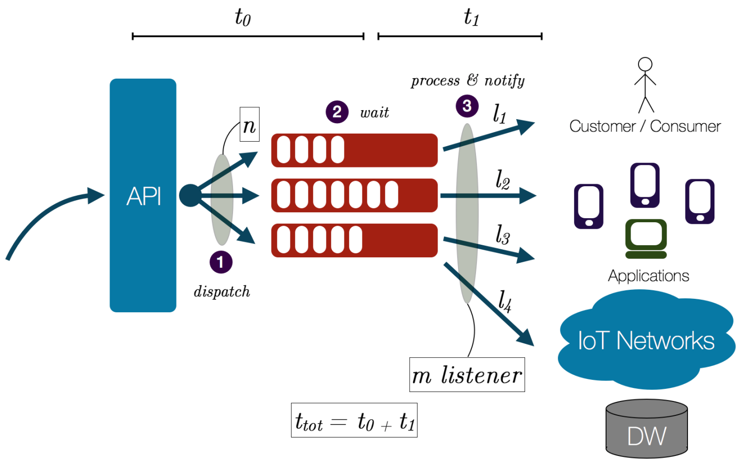

Fig. 4. The proposed listener-based architecture for IoT delay contributions from data generation to consumers information delivery are explicitly indicated.

3.2 Graph-Based Processing

In order to overcome the limitations of the "process-oriented" approach described in Sect. 2, and fit the proposed Big Stream Architecture, we have envisioned and designed a new Cloud Graph-based architecture built on top of basic building blocks that are self-consistent and perform "atomic" processing on data, but that are not directly linked to a specific task. In such systems, the data flows are based on dynamic graph-routing rules determined only by the nature of the data itself and not by a centralized coordination unit. This new approach allows the platform to be "consumer-oriented" and to implement an optimal resource allocation. Without the need of a coordination process, the data streams can be dynamically routed in the network by following the edges of the graph and allowing the possibility to automatically switch-off nodes when some processing units are not required at a certain point and transparently replicate nodes if some processing entities is consumed by a significant amount of concurrent consumers.

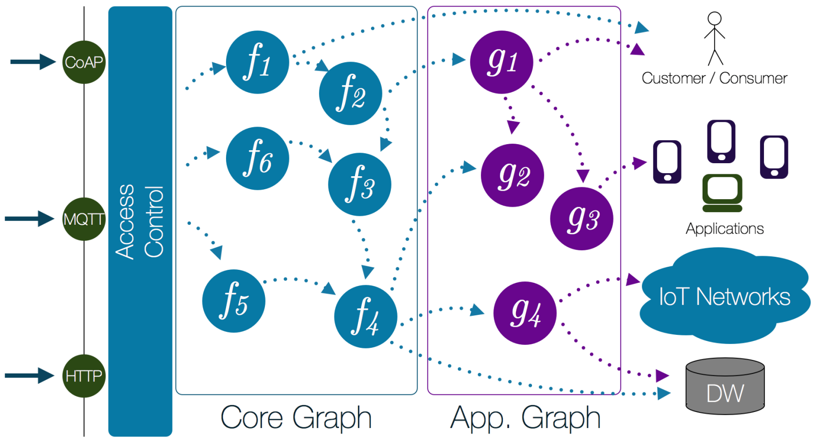

Fig. 5. The proposed listener-based Graph architecture: the nodes of the graph are listeners; the edges refer to the dynamic flow of information data streams.

Figure 5 illustrates the proposed directed graph-based processing architecture and the concept of listener. A listener is an entity (e.g., a processing unit in the graph or an external consumer) interested in the raw data stream or in the output provided by a different node in the graph. Each listener represents a node in the topology and the presence and combination of multiple listeners, across all processing units, defines the routing of data streams from producers to consumers. In this architectural approach:

– nodes are processing units (processes), performing some kind of computation on incoming data;
– edges represent flows of informations linking together various processing unit, which are thus able to implement some complex behavior as a whole;
– nodes of the graph are listeners for incoming data or outputs of other nodes of the graph.

The designed graph-based approach allows to optimize resource allocation in terms of *efficiency*, by switching off processing units that have no listeners registered to them (enabling cost-effectiveness), and *scalability*, by replicating those processing units which have a large number of registered listeners. The combination of these two functionalities and the concept of listener allow the platform and the overall system to adapt itself to dynamic and heterogeneous scenarios, by properly routing data streams to the consumers, and to add new processing units and functionalities on demand.

In order to provide a set of commonly available functionalities, while allowing to dynamically extend the capabilities of the system, the graph is composed by concentric levels:

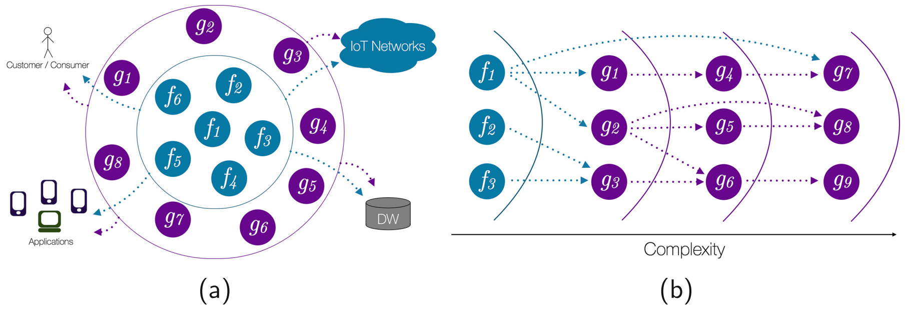

Fig. 6. (a) The concentric linked Core and Application Graphs. (b) Basic processing nodes build the Core Graph: the outer nodes have increasing complexity.

- *Core Graph*: basic processing provided by the architecture (e.g., format translation, normalization, aggregation, data correlation, and other transformations);
- one or more *Application Graphs*: listeners that require data coming from an inner graph level in order to perform custom processing on already processed data.

The complexity of processing is directly proportional to the number of levels crossed by the data. This means that data at an outer graph level must not be processed again at an inner level. From an architectural viewpoint, as shown in a representative scheme in Fig. 6, nodes at inner graph levels cannot be listeners of nodes of outer graph levels. In other words, there can be no link from an outer graph node to an inner graph node, but only vice versa. Same-level graph nodes can be linked together if there is a need to do so. Figure 7 illustrates incoming and outgoing listener flows between Core and Application graphs units. In particular,

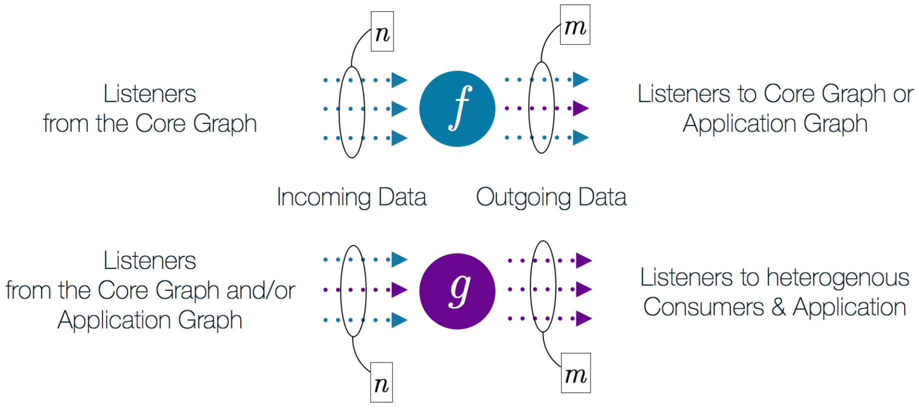
Listeners from the Core Graph
n
f
m
Listeners to Core Graph or Application Graph
Incoming Data
Outgoing Data
Listeners from the Core Graph and/or Application Graph
g
Listeners to heterogenous Consumers & Application
n
m

Fig. 7. Allowed input and output flows for Core Graph nodes and Application Graph nodes.

a processing unit of the Core graph can be a listener only for other nodes of the same level (n incoming streams) and a source both for other Core or Application graph nodes (m outgoing streams). A node of an Application graph can be, at the same time, (i) a listener of n incoming flows from Core and/or App graph and (ii) a data source only for other (m) nodes of the application graph or heterogeneous external consumers.

4 Conclusions & Future Works

In this paper, we have presented a novel Cloud architecture for the management of Big Stream applications in IoT scenarios. After defining the Big Stream paradigm and highlighting the main differences with the traditional Big Data paradigm, in terms of data sources and real-time requirements, we have described how a consumer-oriented architecture might lead to the minimization of the latency between the time instant of data generation and the time instant at which the processed data can be delivered to a consumer application. The proposed architecture is based on a data-driven processing graph, where data flows between processing units (graph nodes), denoted as *"listeners,"* which, collectively, perform consumer-oriented processing, thus implementing an architecture-wide push-based communication. The listener-oriented approach has to several benefits, such as: (i) lowest latency, as the push-based approach guarantees that no delay, due to polling and batch processing, is introduced; (ii) fine-grained self-configuration, as listeners can dynamically "plug" to those that output data of interest; (iii) optimal resource allocation, as processing units that have no listeners can be switched off, while those with many listeners can be replicated, thus leading to cost-effectiveness from a Cloud service perspective. As a next step of our work, we plan to start the implementation of the presented architecture modules with open-source technologies, and test them with data from real IoT scenarios.

Acknowledgments. The work of Simone Cirani and Gianluigi Ferrari is funded by the European Community's Seventh Framework Programme, area "Internetconnected Objects", under Grant no. 288879, CALIPSO project - Connect All IP-based Smart Objects. The work reflects only the authors views; the European Community is not liable for any use that may be made of the information contained herein. The work of Laura Belli is supported by Multitraccia SC. The work of Marco Picone is supported by Guglielmo srl.

References

1. Internet Engineering Task Force: RFC 791 Internet Protocol - DARPA Inernet Programm, Protocol Specification, September 1981
2. Deering, S., Hinden, R.: RFC 2460 Internet Protocol, Version 6 (IPv6) Specification. Internet Engineering Task Force, December 1998
3. IETF: The Internet Engineering Task Force. http://www.ietf.org/

4. EC FP7 Project: CALIPSO - Connect All IP-based Smart Objects. http://www.ict-calipso.eu/
5. Bonomi, F., Milito, R., Zhu, J., Addepalli, S.: Fog computing and its role in the internet of things. In: Proceedings of the First Edition of the ACM Workshop on Mobile Cloud Computing, New York, pp. 13–16 (2012)
6. EU FP7 project: SENSEI. http://www.ict-sensei.org/index.php
7. EU FP7 project: Internet of Things - Architecture (IoT - A) (2012–2015). http://www.iot-a.eu/
8. Contiki Operating System. http://www.contiki-os.org/
9. Gubbi, J., Buyya, R., Marusic, S., Palaniswami, M.: Internet of Things (IoT): a vision, architectural elements, and future directions. Future Gener. Comput. Syst. **29**(7), 1645–1660 (2013)
10. Manjrasoft: Aneka. http://www.manjrasoft.com/aneka_architecture.html
11. Rackspace, NASA: OpenStack Cloud Software - Open source software for building private and public clouds. https://www.openstack.org/
12. EU FP7 project: FI-Ware (2011). http://www.fi-ware.org/
13. Open Source Project: OpenNebula. http://opennebula.org/
14. Hohpe, G., Woolf, B.: Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions. Addison-Wesley Longman Publishing Co., Inc., Boston (2003)
15. Isaacson, C.: Software Pipelines and SOA: Releasing the Power of Multi-Core Processing, 1st edn. Addison-Wesley Professional, Upper Saddle River (2009)
16. Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T.: Hypertext transfer protocol - http/1.1 (1999)
17. Shelby, Z., Hartke, K., Bormann, C., Frank, B.: Constrained Application Protocol (CoAP). Technical report, IETF Secretariat, Fremont, CA, USA (2011)
18. Saint-Andre, P.: Extensible messaging and presence protocol (xmpp): Instant messaging and presence. Internet RFC 3921, October 2004
19. MQTT: MQ Telemetry Transport. http://mqtt.org/
20. Cirani, S., Picone, M., Veltri, L.: CoSIP: a constrained session initiation protocol for the internet of things. In: Canal, C., Villari, M. (eds.) ESOCC 2013. CCIS, vol. 393, pp. 13–24. Springer, Heidelberg (2013)
21. Cirani, S., Picone, M., Veltri, L.: A session initiation protocol for the internet of things. Scalable Comput. Pract. Exp. **14**(4), 249–263 (2014)
22. Roach, A.B., Memo, S.O.T.: Session Initiation Protocol (SIP)-Specific Event Notification, RFC 3265 (2002)
23. Fielding, R.T.: Architectural styles and the design of network-based software architectures. Ph.D. thesis (2000)