Journal of Network and Computer Applications 67 (2016) 99–117

Contents lists available at ScienceDirect

Journal of Network and Computer Applications

journal homepage: www.elsevier.com/locate/jnca

Review

State-of-the-art, challenges, and open issues in the integration of Internet of things and cloud computing

Manuel Díaz, Cristian Martín *, Bartolomé Rubio

Department of Languages and Computer Science, University of Málaga, Boulevar Louis Pasteur 35, 29071 Málaga, Spain

ARTICLE INFO

Article history:
Received 25 September 2015
Received in revised form
21 December 2015
Accepted 15 January 2016
Available online 25 January 2016

Keywords:
Internet of things
Cloud computing
Infrastructure as a service
Cloud platforms
Iot middleware
Big data

ABSTRACT

The Internet of Things (IoT) is a paradigm based on the Internet that comprises many interconnected technologies like RFID (Radio Frequency IDentification) and WSAN (Wireless Sensor and Actor Networks) in order to exchange information. The current needs for better control, monitoring and management in many areas, and the ongoing research in this field, have originated the appearance and creation of multiple systems like smart-home, smart-city and smart-grid. However, the limitations of associated devices in the IoT in terms of storage, network and computing, and the requirements of complex analysis, scalability, and data access, require a technology like Cloud Computing to supplement this field. Moreover, the IoT can generate large amounts of varied data and quickly when there are millions of things feeding data to Cloud Computing. The latter is a clear example of Big Data, that Cloud Computing needs to take into account. This paper presents a survey of integration components: Cloud platforms, Cloud infrastructures and IoT Middleware. In addition, some integration proposals and data analytics techniques are surveyed as well as different challenges and open research issues are pointed out.

Contents

* Corresponding author.
E-mail addresses: mdr@lcc.uma.es (M. Díaz), cmf@lcc.uma.es (C. Martín), tolo@lcc.uma.es (B. Rubio).

http://dx.doi.org/10.1016/j.jnca.2016.01.010
1084-8045/

1. Introduction

The Internet of Things was probably introduced by Ashton (2009) in 1999. The IoT can be defined as a set of interconnected things (humans, tags, sensors, and so on) over the Internet, which have the ability to measure, communicate and act all over the world. The key idea of the IoT is to obtain information about our environment to understand and control and act on it. The IoT can help us in our daily life, e.g. Zaslavsky et al. (2012), where a smart home scenario adapts to the everyday user improving their quality of life and home consumption through a set of home sensors and data information city. Furthermore, the IoT is also suitable in Ambient-Assisted Living, Smart Unit, Monitoring, Tracking, Control systems, Safer Mining Production and so on (Botta et al., 2014; Sindhanaiselvan and Mekala, 2014; Singh et al., 2014; Gubbi et al., 2013; Da Xu et al., 2014). However, the IoT usually coincides with sensors with low power, low memory and battery and network limitations, so there is a need of computing, storage and access and analysis of IoT data (Zaslavsky et al., 2012). Furthermore, there are large amounts of heterogeneity data and devices (Compton et al., 2012) which will grow (Zaslavsky et al., 2012), so a platform that can handle all of this is necessary.

Cloud Computing enables a convenient, on demand and scalable networks access to a pool of configurable computing resources (Zaslavsky et al., 2012). Cloud Computing has virtually unlimited capabilities in terms of storage and processing power (Botta et al., 2014), which are the main drawbacks of IoT. Therefore, by Cloud Computing, IoT can be abstracted of its limitations, heterogeneity, connectivity, identification and security of devices involved (Zorzi et al., 2010). There are different types of categories in Cloud Computing: IaaS (Infrastructure as a Service) which is the lowest layer in a Cloud Infrastructure and offers a poll of Virtual Machines for computing and storage, PaaS (Platform as a Service) is the middle layer which allows deploying applications, and SaaS (Software as a Service) the top layer, that offers accessible user applications like IBM Bluemix, OPENSHIFT, Google App Engine, HEROKU and Microsoft Azure.

The Cloud Computing and IoT integration, known as Cloud of Things (Aazam et al., 2014), solves such problems as IoT's limitations, data access, computing, data analysis, and can create new opportunities, like Smart Things, Things as a Service and SenaaS (Sensor as a Service) (Barbaran et al., 2014; Madria et al., 2014). Moreover, offering a PaaS, users can build applications which use and handle System's Things; even external applications can acquire semantic standard data through Linked Data (Le-Phuoc et al., 2012). The latter takes advantage of the data acquired and saves storage space. Due to the integration benefits and the proliferation of Cloud Computing in recent years, there are several projects and strong research efforts in this field. In the last few years, multiple platforms, protocols, and systems have emerged to tackle the challenges of Cloud Computing and IoT constrained devices.

This paper aims to present the state of the art of different levels of integration components, analyzing different existing proposals in this field and pointing out some challenges and open research issues. Previous research has surveyed Cloud Computing and IoT integration. A survey of Cloud Computing and Wireless Sensor Networks (WSN) overview some applications with both, known as Sensor-Cloud which is presented in Sindhanaiselvan and Mekala (2014). Botta et al. (2014) survey the need for integration, showing some applications thanks to this paradigm and mentioning some open issues and future directions. IoT challenges, visions and applications and the importance of cloud computing and semantics in this field is surveyed in Singh et al. (2014). The term the Cloud of Things and some key integration issues have been introduced by Aazam et al. (2014). Also, Gubbi et al. (2013) present the IoT as an emerging technology, show applications, tends, a cloud centric IoT approach, and mentioned Cloud Computing as an open challenge in the IoT. Our approach does not focus on scenarios where the IoT and Cloud Computing are suitable or limitations or needs as other work has. However, our approach attempts to offer a practical vision to integrate current components of Cloud Computing and the IoT.

Also we know the current limitations on IoT devices, especially on embedded devices, so although we have surveyed different cloud technologies to improve these, the software for embedded devices is a key challenge to achieve the desired integration. In addition to the limitations of the devices, IoT also requires applications in critical and real-time systems where low-latency and low-bandwidth-usage are key requirements. We have taken into account the latter and we have tried to survey an integration which addresses these requirements.

The rest of the paper is organized as follows. In Section 2 integration components are surveyed. Section 3 analyzes current proposals in multiple areas for this field. Section 4 shows data analytics techniques for optimizing such integration. In Section 5 some case studies are analyzed for discussing about the elements surveyed. Section 6 points out challenges and open research issues. Finally, conclusions are drawn in Section 7.

2. Integration components

We have classified the integration components into three categories taking into account the need for a seamless integration. On the one hand, we have surveyed multidisciplinary Cloud Platforms to satisfy IoT limitations and to offer new business opportunities and more scalability. For the deployment, management and monitoring of Cloud Platforms, we have surveyed different Cloud Infrastructures. And lastly, we have surveyed several IoT middleware to abstract the underlying heterogeneous IoT devices.

Cloud Computing and IoT integration provides new storage, processing, scalability and networking capabilities which are so far limited in the IoT due to its characteristics. Furthermore, new opportunities like complex analysis, data mining and real-time processing will be present on IoT, hitherto unthinkable in this field. Finally, through the IoT Middleware, the IoT devices will have a lightweight and interoperable mechanism for the communication among themselves and with the Cloud systems deployed. Fig. 1 shows the integration components surveyed in this paper.

2.1. Cloud platforms

Recently and in the future, the number of users and data from IoT will grow significantly as the number of connected devices increases (Zaslavsky et al., 2012). For a long time, DBMS (Database Management Systems) have been used to store and access data in a great number of applications. Nevertheless, the growth in users and data means that a large number of DBMS are unsuitable. Hence, a platform which can tackle these needs is required in order to offer high scalability, storage and even processing. In this subsection, we summarize different platforms for storing, processing and accessing large amounts of heterogeneity data, which has recently become known as Big Data (Zaslavsky et al., 2012).

2.1.1. Batch processing

Processing and analyzing large amounts of data is one of the requirements addressed in the integration proposal. Batch processing components are responsible for the execution of a series of jobs without manual intervention, allowing a greater distribution of these and high throughput.

An open source framework to manage large amounts data is approached by Shvachko et al. (2010). Hadoop is composed by

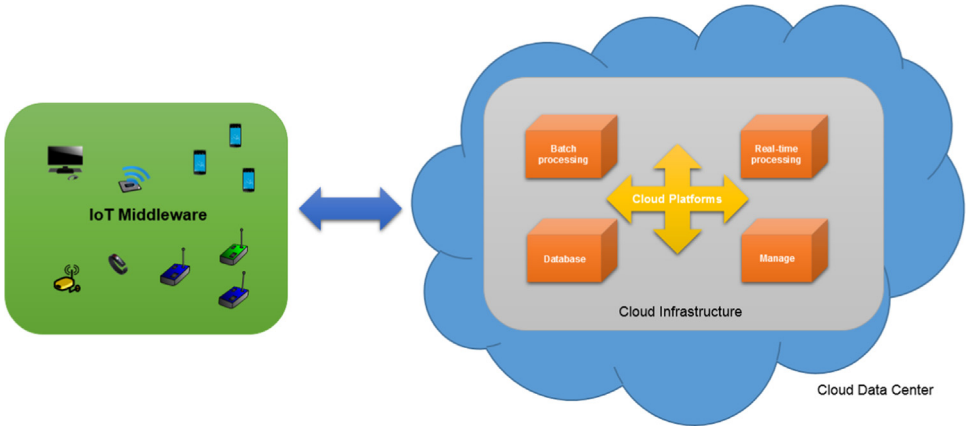

Fig. 1. Cloud and IoT integration.

three main components: HDFS, MapReduce and YARN. HDFS is a distributed file system responsible for storing distributed and replicated data through a server cluster, providing reliability, scalability and high bandwidth. HDFS also balances the disk space usage between servers with a master/slave architecture. MapReduce (http://hadoop.apache.org/docs/r1.2.1/mapred_tutorial.html) is also a master/slave framework used in Hadoop 1 to process and analyze large data sets, master job scheduling and cluster resource management on slaves. The coupling of a specific programming model like MapReduce with the cluster resource management, and concerns about centralized handling jobs, have forced the appearance of YARN (Vavilapalli et al., 2013) on Hadoop 2. YARN aims to decouple resource management functions from the programming model, incorporating a new layer which abstracts resource management at the same time as allowing new programming models. Presently, MapReduce is only an application running on top of YARN. However, the complexity of developing MapReduce programs has originated the appearance of new enriched systems that translate code into MapReduce, like Apache Hive (https://hive.apache.org/), an SQL-like data warehouse for managing and querying large datasets, and Apache Pig (https://pig.apache.org/), a platform for analyzing large datasets with a high-level language for expressing programs.

Although Hadoop and other platforms are suitable for processing and analyzing large amounts of data, in some situations such as machine learning algorithms and interactive ad-hoc on large datasets, latency is considerable due to having to reload continuous data from a disk (Zaharia et al., 2010). A good solution is a Map Reduce framework that keeps data in memory to reuse for multiple jobs and to reduce disk latency. Apache Spark (Zaharia et al., 2010; https://spark.apache.org/) is a fast and general framework which processes large amounts of data with low latency. Spark incorporates fault-tolerant and large data processing like Hadoop Map Reduce, but also introduces a novel abstraction, the RDD (Resilient Distributed Dataset). RDDs are fault-tolerant collections of elements distributed across nodes than can be created, parallelizing an existing collection or referencing a dataset in data store. The main advantages of RDDs are that they can persist in memory and can rebuild lost data without replication, executing 100 × faster than Hadoop Map Reduce in a logistic regression algorithm (Apache spark, 2015). Moreover, Spark has a driver-worked architecture unique to each application, allowing isolation between applications, but data sharing applications must be written in external storage. Spark is also compatible with any Hadoop data store, like HDFS, Apache Cassandra, Apache Hive, Apache Pig or Apache HBase, Chukwa and Amazon S3.

2.1.2. Distributed databases

Focusing more on storing and querying large amounts of data, the distributed databases offer mechanisms to enable some traditional features DBMS in a distributed environment with high availability and low latency.

Druid (Yang et al., 2014; http://druid.io/) is an open source distributed and column oriented platform for storing and accessing large data sets in real-time. The need for storage and computing have originated platforms that store large amounts of data like Hadoop. However, these platforms do not guarantee how quickly data can be accessed and stored nor do they query performance, which can be required in some systems. Druid aims to resolve these problems and offers a platform which is suitable for applications that require low latency on ingestion and query data. Druid architecture is composed by two main components: historical nodes, which store and query non-real-time data and real-time nodes, which ingest stream data and respond to queries with it. Moreover, other components such as the coordination nodes coordinate historical nodes to ensure that data is available and replicated, the brokers receive queries and forward them to real-time and historical nodes, and lastly, the indexer nodes ingest batch and real-time data in the system. The main functions of real-time nodes are to respond with real-time data and built segments for aged data that they send to the historical nodes. Historical nodes persist data in deep storage and download it in memory when the coordination node requires it. Druid also utilizes Apache Zookeeper—a well-known Hadoop component for synchronizing elements in a cluster—to manage the current cluster state, and MySQL for the maintenance of metadata about data segments.

Apache HBase (http://hbase.apache.org/) is another open source distributed database suitable for random and real-time read/write large amounts of data. Apache HBase emerged after Google's Bigtable, a distributed storage system for structured data provided by Google File System. Specifically, Apache HBase uses HDFS for its storage system, and can be seen as the Hadoop Database. In HBase, data is stored in tables like traditional Relational Database Management Systems (RDBMS). However, HBase tables are composed by multiple rows, where each row has a set of columns which can store mixed and indeterminate key-value sets unlike traditional RDBMS. The tables can also be stored in different namespaces to restrict resources, and offer different security levels and region groups. The rows are identified and stored lexicographically by row key, allowing related rows, or rows which can be read together. The HBase architecture is HMaster/RegionServer, where the HMaster monitors and manages all RegionServers, and each RegionServer serves and manages the underlying regions. The regions are the basic components to distribute and provide availability and fault-tolerant in Hbase tables. Initially,

if a pre-splitting policy has not been established, a table is composed by only one region. Then, when the table rows grow, a set of policies can be established to split the regions in the RegionsServers. If a table has been split into three regions, each region contains a sorted keyset chunk of the table. Regions' replication is provided by HDFS to enable fault-tolerant. The HMaster is responsible for assigning regions to RegionsServers, balancing regions between RegionServers and restoring regions if a RegionServer goes down. Furthermore, HBase has allowed interaction with MapReduce, so MapReduce jobs can interact with the HBase data store. HBase provides a set of operations to modify and query tables, but if you want to use a SQL functionality and JDBC connector with HBase, you can use Apache Phoenix (http://phoenix.apache.org/) or Cloudera Impala (loudera.com/content/cloudera/en/products-and-services/cdh/impala.html).

However, in some situations such as time series, the sorted key can imply a major issue because when a table is split into two regions, the new writers will be written in the last region due to sorting, so a hot region will always be present. This problem is solved by OpenTSDB (http://opentsdb.net/). OpenTSDB uses an HBase table and a key format composed by a metric type—a string composed by a timestamp and a set of key/value—to store all stream data across various regions in a table. OpenTSDB has a broker architecture formed by the Time Series Daemons (TSD). The TSDs are the key components in OpenTSDB since they are responsible for managing and querying all data saved on HBase. A set of aggregation operations over the metrics as max, min or avg can be applied to OpenTSDB. All the features of HBase such as fault-tolerant, replication, split into tables, and so on, are used by OpenTSDB. Furthermore, a Web UI and other tools have been integrated to query and graph all the time series stored. Nevertheless, a storage limitation is defined in OpenTSDB, since each metric and its types and values are assigned a unique id with 3 bytes, so $2^{24}-1$ metrics and $2^{24}-1$ types and values on each metric can be stored.

2.1.3. Real-time processing

On the other hand, there are situations that require not only access to data, but also processing it in real-time, like decision making in critical systems or trending topics in social media.

Apache Storm (Toshniwal et al., 2014) is a popular open source distributed system for processing data streams in real-time. Storm contains a nimbus/supervisor architecture coordinated through Zookeeper, and is based on directed graphs, where the vertices represent computational components, and the edges represent the data flow among components. Spouts are the Storm components that receive data in the topology and are transmitted to Bolts which are responsible for processing the data and transmitting it to the next set of Bolts or data storage as required. Moreover, Storm offers semantic guarantees about data processing like 'at least once' or 'at most once' in addition to fault-tolerance. The main drawback is that it is not able to dynamically re-optimize at runtime, between nodes, but it is considered as future work (Toshniwal et al., 2014).

An extension of Apache Spark for fault-tolerant streaming processing is known as Spark Streaming (Zaharia et al., 2012; https://spark.apache.org/streaming/). Spark Streaming follows a different philosophy than Apache Storm. In Spark Streaming, the data received is stored in memory for a specific interval or window, and then is processed and stored in a Spark RDD. A D-Stream (Discretized stream) is the representation by times series of RDD, and it lets users manipulate them through various operators in real-time. Spark Streaming is properly integrated with Apache Spark, as it uses the same data representation, allowing reuse code and a hybrid architecture through the combining of batch with stream processing with the same data. The main drawback is the length of the sliding windows, as if they are too large it can make the real-time disappear whereas if they are too small it can generate multiple RDDs. Furthermore, in most cases, stream data is received over the network, so to achieve data received fault-tolerance, Spark Streaming replicates data among the worker nodes.

2.1.4. Distributed queues

Albeit in multiple systems, data is obtained from internal data stores, in some systems, especially in IoT, data is dispatched by many devices or sub-systems. To deploy a large number of things in an IoT system requires large amounts of data received to be handled and eventually dispatched in real-time to the stakeholders. Distributed queues are summarized in this subsection in order to solve these problems.

Apache Kafka (Kreps et al., 2011; http://kafka.apache.org/) is a distributed messaging system that consumes and dispatches large amounts of data with low latency. Kafka is based on a publish/subscribe queue with 'at least once' semantics that guarantees its suitability for streaming data and performing offline analysis. Topics are the stream of messages in Kafka, wherein producers can publish messages and consumers can subscribe to receive them. Load balancing and fault-tolerance is performed by partitions of the topics, where each topic can be divided into multiple partitions, and each partition can have multiple replicas. Communication between producers and consumers to the server is through agnostic TCP, so it is available in many languages. When a message is sent by a producer to Kafka, it is stored on a disk in determined partitions and its replicas are consumed for a prefixed time. Moreover, Kafka allows group-consumers, a way to introduce load balancing ingestion data between consumers. However, Kafka does not contain a master node, but it does contain a set of brokers which locate the system partitions and are synchronized through Apache Zookeeper. The latter avoids concerns of master failures. Furthermore, producers and consumers can operate synchronous or asynchronous with batch data, but it originates a great dilemma between latency and throughput.

RabbitMQ (Dossot, 2014; https://www.rabbitmq.com/) is another open source messaging queue that contains an Erlang-based implementation of AMQP (Advanced Message Queuing Protocol) v0.9.1 protocol. AMQP is an open standard to exchange messages over TCP. RabbitMQ utilizes the Mnesia database, which is an in-memory persistent embedded database of Erlang for data persistence. RabbitMQ also uses brokers which are middleware applications of AMQP that receive messages from producers and send them to other brokers or consumers. Brokers can be replicated for high availability and send messages to other brokers in other clusters. When a message is sent by producers, it is received by a component of brokers, called exchange, that routes it, according to its routing rules—point-to-point, publish-subscribe, headers or multicast—and it is sent to a queue. When a consumer or consumer group receives a message, it is deleted from the queue. Moreover, RabbitMQ contains a pluggable system, where plugins can extend the deployment with new components like a web UI management and monitoring. Therefore, RabbitMQ follows a different philosophy from Kafka, it incorporates message confirmation and more forms of communication than a topic. Producers can also obtain the message acknowledgment of consumers, so RabbitMQ can behave as RPC (Remote Procedure Call) protocol. RabbitMQ can also be more flexible than Kafka, however throughput and latency in Kafka is more desirable.

2.1.5. Management, monitoring and deployment

Many systems like Apache Storm, Apache Hadoop or Apache Spark contain a web UI for monitoring and visualizing the cluster deployed. Nevertheless, in deployed clusters with multiple platforms

deployed, what is needed is a single platform that can deploy, monitor and manage all platforms.

In the Hadoop ecosystem, there is a platform for provisioning, monitoring and managing Hadoop clusters, called Apache Ambari (http://ambari.apache.org/). Apache Ambari provides an open source management web UI, enabling system administrators to easily deploy new Hadoop services in the cluster, add new host nodes, manage existing services deployed and monitor the heath and status of the Hadoop cluster. Ambari follows a master/agent architecture, where master contains a set of components responsible for monitoring the agent status and planning actions. Ambari agents send heartbeat event to master every few seconds with the agent status and actions and it receive actions to execute in the response. Moreover, Ambari utilizes a database to store the system status, so in case of master failure, the status is rebuilt. Ambari uses Ganglia—a distributed monitoring high-performance system —for metrics collections, and Nagios—a monitoring system for IT infrastructure—for system alerting and sending emails. Furthermore, Ambari also integrates all capabilities in other systems through the Ambari REST APIs. Ambari is therefore a powerful tool for managing and monitoring Hadoop components with only one platform, and it can be integrated easily into other systems.

Tables 1 and 2 summarize a comparison for the different Cloud Platforms analyzed. The components cover multiple Cloud Computing needs such as batch and stream processing, distributed storage and distributed queue which can be integrated independently or jointly. Most components follow a master/slave architecture and present a point of failure if the master goes down, but this can be mitigated with master replication or a broker architecture. Replication and load balancing are contained in most components, except load balancing in Apache Storm and Apache Ambari, and replication in Apache Ambari and Apache Spark that relies on the data store for this. Fault tolerance is one of the basic Cloud Computing features and is present in most components.

For batch processing, Apache Spark is recommended for low-latency applications with the same data, whereas Apache Hadoop is a fairly consolidated platform and is more integrated with external systems and compatible. Considering compatibility aspects, Spark Streaming is suitable if Apache Sparks is chosen as a batch processing component. Apache Storm does not necessarily require data storage as Spark Streaming does, and it contains more external integration, but nevertheless it does not contain load balancing so the correct design is necessary and not too many work peaks. For distributed queuing, it depends on the application, you may require confirmation messages and more options than a single topic, so RabbitMQ is the best solution in this case, but if you want a low-latency and throughput solution, Apache Kafka should be chosen.

Although HBase is an established solution, the need to design based on the data and the lack of real-time, mean that Druid will become the more promising solution. Apache Ambari has been designed to solve multi-platform deployment problems and is seemingly the solution for this. Also, Apache HDFS is the most compatible distributed file system, whilst components like Java Apache Thrift and Hadoop Streaming allow multi-language service development. On the other hand, Apache Hadoop has the largest number of official companies using it, more than 170 companies, in comparison with Druid and OpenTSDB with 10 and 30 companies, respectively. Lastly, all components surveyed have an open source license.

2.2. Cloud infrastructures

In a cloud datacenter, the IaaS is the key component to provide capabilities, since it is responsible for managing and provisioning and deploying virtual components. Virtualization has an important

Table 1
Cloud Platforms comparison I.

Cloud Platform	Finality	Architecture	Replication	Load balancing	Fault-tolerant	Memory	Programming Languages	Data store	External integration	Access	Documentation	Official companies that use it	Last update	License
Apache Hadoop MapReduce	Batch processing	Master/slave	Yes	Yes	Yes, if not master	Yes	Java, C++, Hadoop Streaming and REST API	HDFS		Web UI and CLI	★★★	+170	dec-15	Apache License Version 2.0
Apache Hadoop HDFS	Distributed file system	Master/slave	Yes	Yes	Yes, if not master	Yes, optionally	Java, Apache Thrift and REST API	HDFS		Web UI and CLI	★★★	+170	dec-15	Apache License Version 2.0
Druid	Real-time data store	Multiple nodes	Yes	Yes	Yes, if not coordinator	Yes, real-time nodes	Ruby, Python, R and Node.js	S3, HDFS, local mount and Cassandra	Hadoop, Storm and Kafka	CLI	★★	+10	dec-15	GNU General Public License Version 2.0
Apache HBase	Distributed data store	HMaster/ HServerRegions	Yes	Yes	Yes, if not Hmaster	Yes, optionally	Java, C/C++, Apache Thrift and REST API	HDFS, local filesystem	Apache Hadoop MapReduce	Web UI and CLI	★★★	+40	nov-15	Apache License Version 2.0
OpenTSDB	Distributed time series data store	Brokers	Yes	Yes	Yes, if not Hmaster	No	REST API, R Client, Erlang, Ruby, Go	HBase	RabbitMQ, Etsy Skyline and Apache Pig	Web UI and CLI	★★	+30	nov-15	GNU LGPLv2.1+

Table 2
Cloud platforms comparison II.

Cloud platform	Finality	Architecture	Replication	Load balancing	Fault-tolerant	Memory	Programming Languages	Data store	External integration	Access	Documenta- tion	Official companies that use it	Last update	License
Apache Spark	Batch processing	Driver/ worked	No in memory, implicit in storage	Yes	By application, if driver not fail.	Yes	Java, Python and Scala	HDFS, S3, Cassandra and HBase		Web UI and CLI	⋆⋆⋆	+80	nov-15	Apache License Version 2.0
Apache Kafka	Distributed queue	Brokers	Yes	Yes	Yes	No	Several languages		Several systems	CLI	⋆⋆	+70	N/A	Apache License Version 2.0
Apache Storm	Stream processing	Nimbus/ supervisor	Yes	No	Yes, if not nimbus	Yes	Java and Apache Thrift		Several systems	Web UI and CLI	⋆⋆	+70	nov-15	Apache License Version 2.0
Apache Spark Streaming	Stream processing	Driver/ worked	Yes, at reception	Yes	By application, if driver not fail.	Yes	Java, Python, Scala	HDFS, S3, Cassandra, Hbase		Web UI and CLI	⋆⋆⋆	N/A	nov-15	Apache License Version 2.0
RabbitMQ	Distributed queue	Brokers	Yes, with plugin	Yes	Yes	Yes	Several languages		MQTT and STOMP	Web UI and CLI	⋆⋆	N/A	dec-15	Mozilla Public License Version 1.1
Apache Ambari	Provisioning and monitoring and managing clusters	Master/agents	No	No	Yes, if not master	No	REST API	HDFS	Several Hadoop Components	Web UI, CLI	⋆⋆	N/A	dec-15	Apache License Version 2.0

role because of abstract computing, networking and storage platforms from the underlying physical hardware (Moreno-Vozmediano et al., 2012). Furthermore, through virtualization you can execute multiple OSs (Operation Systems) with a better exploitation of the hardware it facilitates the fault-tolerance through simultaneous deployment and provides high server utilization with saving costs and energy.

OpenNebula (http://opennebula.org/) is an example of a Cloud IaaS platform, which emerged from a research project undertaken at the Complutense University of Madrid in 2005. OpenNebula is an open source solution for management of virtualized data centers with private, public and hybrid IaaS clouds. The aim of the project is to provide an open, flexible, extensible, management layer to abstract networking, storage, virtualization, and monitoring and user management in different public clouds such as Amazon Web Services, Microsoft Azure and private clouds. All components in OpenNebula have been grouped in one key component, the cloud OS, which manages virtual and physical infrastructures and controls the provisioning of virtual resources according to the needs of services (Moreno-Vozmediano et al., 2012). The system is composed by a front-end/host architecture. The front-end is considered a master node in a common cloud platform, containing the OpenNebula services to manage and monitor the cluster, and each host can be considered a slave in a common cloud platform, that runs VMs (Virtual Machines). The different data stores—System, Image and File data stores—are shared through the network by front-end and host nodes.

Public clouds can also be configured in OpenNebula to allow access to external users or to sell capabilities through the Amazon EC2 Query Interface. Multiple OpenNebula instances can be configured in Federation mode, where each instance is configured as a slave and the master instance manages and shows everything as a single OpenNebula instance. The latter is suitable for organizations with several worldwide data centers. OpenNebula offers both a Data Center Virtualization Management allowing datacenters to be managed and consolidated through advanced features like capacity management and resource optimization, as Cloud management to enable VDC (Virtual Data Centers) provisioning that allows the data center to be portioned into multiple VDCs. The VDCs enable the isolation of users or workloads, defining different levels of security, high availability in each VDC, datacenter federation and hybrid cloud computing. Lastly, OpenNebula allows Multi-VM Applications and Auto-Scaling, where services which contain multiple VMs deployed are managed as a single entity, and an application insight through the reception of monitoring information of VM guest which can be used to detect problems in applications and trigger auto-scaling rules.

Another platform which follows the cloud OS paradigm is the well-known open source IaaS OpenStack (https://www.openstack.org/). Whereas OpenNebula has a centralized system with optional components, OpenStack is composed by a set of interrelated subcomponents with its own APIs which must be installed and integrated for OpenStack deployment. Therefore, OpenStack has a pluggable peer architecture where each component can be installed and controlled in a different node. Moreover, OpenStack is focused on private clouds, or public clouds which have an AWS EC2 compatible API. OpenStack have four main services: Dashboard, Compute, Networking and Storage. The Dashboard has been designed to manage OpenStack resources and services through a Web-based UI which can be customizable to adjust it to the user's needs. Computer service is maybe the most important component as it is responsible for managing and controlling the cloud platform. OpenStack supports two types of storage: Object and Block. Object Storage is suitable for redundant, fault-tolerant and scalable data stores, while Block Storage provides a persistent block level storage suitable for performance sensitive scenarios. OpenStack also has a VDC implemented through VLAN, permission and quotas over existing resources: volumes, instances, images, and so on. The networking service allows Network-Connectivity-as-a-Service for other OpenStack services, offering an API to manage networks, supporting many technologies and networking vendors through a pluggable architecture. The Load-Balancer-as-a-Service enables networking to distribute requests between instances, and the Firewall-as-a-service enables firewall policies for all networking routes.

The data processing service provides a mechanism to easily deploy data processing clusters like Apache Hadoop o Apache Spark, enabling the clusters' deployment in a few minutes by means of specified parameters like cluster topology and nodes hardware. For high availability, OpenStack distinguishes between two services: stateless and stateful services. The stateless services do not depend on their status, so for high availability it is only necessary to have redundant instances with load balancing between them. For stateful services, OpenStack offers two configurations: the active/passive configuration that provides additional services in case of failure through backup instances, and active/active configuration wherein main and backup instances are managed, so in the case of failure the backup instance should not be recovered and backup latency is lower. Finally, the telemetry service is another important component that enables billing and allows users to set alarms to provide Monitoring-as-a-Service. OpenStack is maybe the most popular IaaS service. It is supported by a large part of the community and is sponsored and funded by a wide range of companies.

Apache also has an IaaS project, called Apache CloudStack (http://cloudstack.apache.org/). CloudStack began as a project of a start-up in 2008, and was submitted to the Apache Software Foundation in 2012. CloudStack is an open source IaaS project of the Apache Software Foundation (ASF) for deploying public, private and hybrid IaaS clouds. Like OpenNebula and OpenStack, CloudStack has developed an AWS EC2 support for public IaaS cloud. The CloudStack architecture is also master/slave, where CloudStack Management Server runs in an Apache Tomcat container and manages and orchestrates resources in the cloud, and the hypervisor hosts which deploy VMs in each host node through the installed hypervisor. The CloudStack Management Server is also responsible for providing Web UI and API interfaces, managing storage and images, and it can be deployed in a multi-node mode for high availability and load balancing between management servers. CloudStack also supports zones, like OpenNebula, in order to manage geographically distributed nodes. Distributed geographical zones provide a higher level of fault-tolerance since a cloud system can recover from external disasters in a zone. CloudStack storage is divided into two types: Primary and Secondary Storage. Primary Storage is in charge of storing disk volumes of all VMs while Secondary Storage stores disk volumes, snapshots, ISO images and disk templates. Heterogeneous Secondary Storage is not supported in zones nonetheless CloudStack has plugins to enable OpenStack Object Storage and Amazon S3 storage as Secondary Storage. Moreover, CloudStack also provides VDC in zones, called Virtual Private Clouds (VPC), for isolation data center deployments. Besides, high availability, auto-scale rules and load balancing are all applicable to VMs.

The comparison results of Cloud Infrastructures are shown in Tables 3 and 4. The main difference is that whilst OpenStack is composed by a pluggable set of components, OpenNebula and CloudStack have a centralized architecture. A pluggable architecture like OpenStack can adapt better to the user's needs apart from offering a multitude of component-as-a-service, but the requirements and installation efforts of OpenStack are generally higher than CloudStack and OpenNebula. The main features of IaaS platforms like load balancing, auto-scaling, monitoring, accounting

Table 3
Cloud IaaS comparison I.

IaaS cloud	Cloud model	Hypervisor	External Cloud Connector	Cloud API Interfaces	Cloud Integrators	Access	Cloud Federation	Image repository	Monitoring	Storage
OpenNebula	Public, private and hybrid	KVM, Xen, Vmware ESX and Vcenter	AWS, SoftLayer and Azure	AWS EC2, EBS and OGF OCCI	XML-RPC, Ruby, Java and OpenNebula OneFlow RESTFul API	REST API, Web UI and CLI	Yes, cloud bustring	Yes	Virtual and physical resources	NFS, Lustre, GlusterFS, ZFS, GPFS, MooseFS, Vmware datastore, LVM datastore and Ceph
OpenStack	Private and public	Baremetal, Docker, Hyper-V, LXC, KVM, QEMU, UML, Vmware vShere, Xen		AWS EC2, S3 and OGF OCCI	Native API, Java, Node.js, Ruby, .NET and third-party tools (Euca2ools, Hybridfox, boto, fox, php-opencloud)	REST API, Web UI, CLI and third-party tools	No	Yes	Virtual and physical resources, with Ceilometer service	Object Storage (Swift, Ceph, Gluster, Sheepdog), Block Storage(LVM, Ceph, Gluster, NFS, ZFS, Sheepdog) and multiple commercial drivers
CloudStack	Public, private and hybrid	Baremetal, Hyper-V, KVM, LXC, vSphere, Xenserver and Xen Project		AWS EC2, S3, OpenStack Block Storage and OGF OCCI	Native API and Python	REST API, Web UI and CLI	No	Yes	Virtual and physical resources	Primary Storage (all standards-compliant iSCSI and NFS servers supported by the hypervisors) and Secondary Storage (zone-based NFS, Amazon S3, OpenStack Object Storage and SMB/CIFS)

Table 4
Cloud IaaS comparison II.

IaaS cloud	Accounting	Networking	Security	High avaliability	Load balancing	Auto-scaling	Database status support	Official companies that use it	Last update	License
OpenNebula	Yes	802.1Q VLAN, ebtables, Open vSwitch and Vmware networking	ACL, authentication (user/password, ssh, x509, LDAP) and authorization and various administration roles	Trigger to host failures and multi-cluster deployments	Load balancing in clusters	Yes, application auto-scaling based on metrics or a schedule	MySQL, SQLite	+140	nov-15	Apache License Version 2.0
OpenStack	Yes, with Telemetry service	Plugis for Open vSwitch, Cisco, Linux Bridge, Modular Layer 2, NVP, Ryu OpenFlow, Big Switch, Cloudbase Gyper-V, MidoNet, Brocade Neutron, PLUMgrid, Mellanox, Embrane, IBM SDN-VE, CPLANE, Nuage and OpenContrail	Authentication (username/password, external CA, HTTPD, trusts, ssl, x509, LDAP), authorization, groups, tenants, domains and roles	Redundant instances for stateless services and active/passive and active/active configurations for stateful services	Load balancing as a Service in Networking and load balancing in high availability	By Heat component	MySQL, MongoDB, Cassandra, and so on	+250	oct-15	Apache License Version 2.0
CloudStack	Yes, optional service	Plugins for MidoNet, VXLAN, and SDN	Authentication (username /password, LDAP, ssh keys), authorization, administration roles, domains and groups	Multimanagement replication servers, restart VMs automatically and MySQL replication	IP load balancing VM though rules, load balancing between multiple Management Servers, Citrix NetScaler and F5 support	Yes, back-end services and applications VMs though rules and monitoring	MySQL	+200	nov-15	Apache License Version 2.0

and high availability are present in all three platforms surveyed. Whereas OpenNebula and CloudStack are focused on public, private and hybrid cloud models, OpenStack is mainly focused on private clouds with compatibility with AWS EC2 for public cloud, as are others.

OpenNebula is the only in the cloud infrastructures surveyed that presents cloud federation to interconnect worldwide data centers, and is also the only one that contains external cloud connectors to establish a relationship with public clouds. This offers a high level of QoS (Quality of Service) with peak workloads. However, as mentioned in Tusa et al. (2012), OpenNebula tries to manage federated resources but the approach used for interacting with physical servers (SSH remote commands) it is quite hard to accomplish real federation achievements. The cloud middleware CLEVER (Tusa et al., 2010) takes into account that obstacle and proposes a scalable federated Cloud environment through single sign-on authentication. On the other hand, OpenStack is the platform with more support for storage, networking, database and hypervisor components, whilst OpenNebula is the platform with less component support. OpenStack is also the platform with the most cloud integrators, including native API, different programming languages and third-party tools. CloudStack is the platform with the fewest cloud integrators, with compatibility for just native API and Python. The majority of the platforms can be accessed by an API, a Web UI and a CLI, but OpenStack also supports access through third-party tools. With respect to security aspects, all components have authentication, administrative roles and domain division. OpenStack is the platform which has received the most funding and official use by external companies, over 250 in both cases, whilst OpenNebula contains the lowest number, just over 140 companies use it. Lastly, all components surveyed contain an open source Apache License Version 2.0.

2.3. Middleware for IoT

In IoT deployments there are usually many heterogeneous devices with different functionalities, capabilities, and multiple programming languages to access them. So an abstraction layer is necessary to abstract this heterogeneity in order to achieve a seamless integration with anything. Through a middleware, users and applications can access the data and devices from a set of interconnected things, hiding communication and low-level acquisition aspects (Calbimonte et al., 2014). We now summarize different IoT middleware in this context. In the comparison we have also added the Web Service (WS) protocols DPWS (Device Profile for Web Services) and CoAP (Constrained Application Protocol) that although they are not middleware as such, but they incorporate several middleware features and are important protocols to promote the IoT towards WoT.

GSN (Aberer et al., 2006; https://github.com/LSIR/gsn/wiki) is a middleware that originated in 2005 as a platform for processing data streams generated by WSN at EPFL which provides a platform for flexible integration and deployment of heterogeneous WSNs. The key concept in GNS is the virtual sensor, which describes sensors across XML files, it is able to abstract from implementation details of access to sensor data, and structures though different configurations of the data stream which the virtual sensor consumes and produces. GSN has a container-based architecture composed of several layers: the virtual sensor manager (VSM) which manages virtual sensors and underlying infrastructure; the query manager (QM) which parses and executes and planning SQL queries. It also has a configurable notification manager to configure alerts; and lastly, at the top, an interface layer for access through web services and via Web. VMS manages connection with devices though wrappers, currently there are wrappers for TinyOs platforms, several devices and generic wrappers. Moreover, different instances of GSN can establish a communication through remote wrappers.

Although GSN handles the heterogeneity devices and acquisition level problems, there is actually a large heterogeneity data problem when data need to be interpreted and understood (Le-Phuoc et al., 2012; Calbimonte et al., 2014). An extension of GSN middleware, called XGSN, is presented in Calbimonte et al. (2014) to address this problem facilitating the discovery and search tasks in an IoT environment. XGSN provides semantic annotation for GSN middleware, enriching virtual sensors with semantic data by means of an extension of the SSN ontology. The main semantic annotations of XGSN are about sensors, sensing devices and their capabilities, and the measurements produced which have not been described before in GSN. Moreover, in the same way as GSN, XGSN offers interfaces to query data and managing virtual sensors, and even integration with the LSM (Linked Sensor Middleware) for storing and processing stream data.

DPWS (Device Profile for Web Services) (Sleman and Moeller, 2008; Jammes et al., 2005) is an OASIS Web Services specification for embedded devices and devices with few resources. DPWS is based on SOA (Service-Oriented Architecture), where each device is abstracted as a set of services. DPWS defines a protocol stack built over Web Services standards: SOAP (Simple Object Access Protocol) 1.2, WSDL (Web Services Description Language) 1.1, XML Schema and WS-Addressing; and also defines several protocols for discovery, security, messaging and eventing. The DPWS services can be specified using XML Schema, WSDL and WS-Policy like a Web Service, and are discovered by the WS-Discovery protocol in a plug-and-play mode. Services are discovered, specifying the type of the device, the scope in which devices reside or both. The WS-Discovery protocol uses multicast with SOAP over UDP in order to reduce network traffic overhead and during the discovery process each device displays its metadata information like its EPR (End-Point Reference) or a set of messages that can be sent or received. Moreover, the WS-Eventing protocol defines a publish/subscribe protocol allowing devices to register to receive messages about other devices. Lastly, the WS-Security is responsible for providing different levels of security, such as authentication and confidentiality in the devices. Currently, there are several implantations of DPWS like the WS4D open source implementation (http://ws4d.org/) in C/C++ and different implementations in Java, and the adoption of Microsoft in their OS from Windows Vista and Windows Server 2008 (https://msdn.microsoft.com/en-us/library/windows/desktop/aa826001(v=vs.85).aspx). However, due to the DPWS protocol stack, the DPWS integration in embedded devices may be too heavy, so a solution like the DPWS-compliant gateway proposed by Cubo et al. (2014), can abstract the underlying IoT components while taking advantage of DPWS interoperability and semantics on embedded devices.

The OMG Data-Distribution Service for Real-Time Systems (DDS) (http://portals.omg.org/dds/) is a data-centric publish/subscribe OMG (http://www.omg.org) specification for real-time and embedded systems. The specification defines both the communication semantics (behavior and QoS) and the APIs for efficient delivery of information between producers and consumers. In the same way as Apache Kafka, data publishers in DDS publish typed data-flows over topics which consumers can subscribe to. However, DDS does not contain a server to connect publishers and producers rather it provides dynamic and extensible applications by means of dynamic discovery of publishers, subscribers and data types. DDS follows the DCPS (Data-Centric Publish-Subscribe) model, through typed interfaces, providing the information needed to tell the middleware how to manipulate the data and also providing a level of safety type. A domain is the way to abstract different entities (publishers, consumers, data types) in the same group. At the beginning of each application, it is necessary to

define the data types for the communication, although publishers and consumers have been designed to support multiple data types. Unlike other middleware, DDS defines QoS policies like data durability, latency maximum or data ownership. Currently, there are several official implementations of DDS which include multiple programming languages and SOs as well as different license types.

For communication between devices WSs are usually used. The WSs offer a way to communicate, share and access data information over Internet. Nowadays, there are many WSs offering a variety of functionalities over Internet, like current currency exchange, information on zip codes and cities, connecting with social networks, and so on. The main ones are RESTFul (Representation State Transfer) and SOAP WSs. However, WSs normally operate over HTTP, or in the case of SOAP, they work with exchange XML, so it is a major limitation for constrained devices. CoAP (Bormann et al., 2012; Shelby et al., 2015) is a transfer protocol designed for constrained devices. CoAP follows the same style of RESTFul architecture, but it uses UDP thereby reducing TCP connection and transfer overheads. CoAP define GET, POST, DELETE and PUT operations and responses like normal RESTFul WSs, allowing a seamless integration with HTTP platforms. However, in HTTP the transactions are usually initiated by clients with pull mode, so this scenario is too expensive for devices with power limitations, battery and network. CoAP, in contrast to HTTP, uses an asynchronous mode to push information from servers to clients following the observer design pattern. Moreover, to allow interoperability between CoAP end points, CoAP includes a technique for adverting and discovering resource descriptions, and depending on the situation, CoAP allows several types of confirmation messages.

Nonetheless, HTTP is widespread across the Internet in many applications and systems, unlike CoAP. A proxy design to leverage the compatibility with HTTP and the seamless adaptation of CoAP on embedded devices is presented in Ludovici and Calveras (2015). The proxy shows that the WebSocket protocol is more suitable than HTTP for long-lived communications, whilst the final CoAP devices are not overloaded since they are only connected with the proxy following the observer pattern, handling the proxy all consumers' connection. The proxy also provides HTTP compatibility, and leveraging the observer pattern, uses a cache to reduce the number of requests to CoAP devices. On the other hand, Castro et al. (2014) avoid to use external intermediate application servers for interconnecting clients with end devices, and they propose CoAP Embedded Java Library for interconnecting Web browsers directly with end devices. There are different implementations of CoAP both with private and public licenses as in several programming languages (http://coap.technology/impls.html). CoAP can also considerably reduce power consumption with respect to HTTP in many situations as show in Levä et al. (2014).

LinkSmart (https://linksmart.eu/) emerged from a European research project to develop a middleware based on a Service-oriented architecture for network embedded systems. LinkSmart has been designed, taking into account the common problem of compatibility between proprietary protocols and devices. The result of the project is a software gateway which acts as a bridge between the digital world and the underlying IoT devices. Loosely coupled OSGi based web services are powered by LinkSmart in order to allow applications to control, observe and manage physical devices through the LinkSmart gateway. The use of OSGi provides a components system which allows the deployment and control of components without requiring a reboot. LinkSmart is so far the only one which allows dynamic deployment and control of components during execution, a key issue in critical devices which cannot be rebooted. By default, LinkSmart has components installed, like the Network Manager, which can be used to register and discovery services in the LinkSmart Network and for direct communication between LinkSmart nodes, in addition to optional components to provide a topic based publish-subscribe service, device discovery and secure identity and encrypted communications. LinkSmart requires a JVM and a minimum memory of 256 Mb so it is too high for embedded devices.

For the latter, a proxy component has been incorporated to abstract different communication protocols and interfaces like ZigBee, Bluetooth and USB. The publish/subscribe mechanism provided by the Event Manager component is another key component which exchanges XML messages between senders and receivers, being suitable for asynchronous environments. Moreover, IOS, Android and Windows tools have been incorporated to control and manage the IoT resources deployed.

The component infrastructure middleware introduces a novel paradigm in IoT, where the underlying devices can be reconfigured as well as offering common mechanisms for deploying and managing components in runtime. However, the requirements of JVM in OSGi make it unsuitable for constrained devices. LooCI (Hughes et al., 2012) is a middleware for building component-based applications in WSN which takes into account this restriction. LooCI provides interoperability across various platforms: a Java micro edition for constrained devices known as Squawk, the open source OS for IoT Contiki, and the OSGi. Like LinkSmart, LooCI has a distributed event bus to allow publish/subscribe in addition to direct bindings for communication between all the components deployed. All the communication is based on type events, so it leads to type-safe communications and services and binding discoveries. In contrast to LinkSmart, LooCI instead of having a proxy component to abstract different communication protocols or interfaces, leaves the implementation of the device communication in each component and standardizes the networking services through a Network framework. Recently (Maerien et al., 2015), a new middleware known as SecLooCI has been proposed to expand the LooCI middleware in order to enable a secure sharing of resource-constrained WSN devices. However, the SecLooCI middleware's implementation is not available for download as it has not yet been released

Table 5 shows the results of the IoT Middleware comparison. All middleware surveyed have a mechanism to discover other systems and services. Most middleware contain a peer-to-peer model communication which uncouples the main point of failure in client–server models. On the other hand, DPWS has a client–server model and CoAP has a client–server asynchronous model since the information is pushed from server to clients asynchronously. Several QoS mechanisms are only addressed by DDSs which, together with the real-time support, is maybe the main reason for the adoption of DDS in critical and governmental systems. The device virtualization is addressed by GSN and LinkSmart through the proxy component. GSN, LinkSmart and LooCI are maybe the most versatile middleware, since GSN and LinkSmart contain support for several protocols and devices, and LooCI can implement any device communication through its corresponding interface. For the device requirements, CoAP is the most lightweight middleware, while the different instantiations of the different wrappers as the middleware instantiation itself mark the requirements in GSN, LinkSmart and LooCI. GSN is the weightiest middleware due to its protocol stack, and DDS does not define the communication stack so the weight of DDS may vary in each implementation.

DDS, DPWS and LinkSmart contain secure implementations, including security services like authentication and data confidentiality. On the other hand, CoAP can use DTLS (Datagram Transport Layer Security) for security, GSN does not mention anything about security and LooCI has released a new middleware for security but it is not yet available. DDS, LinkSmart and LooCI require that all data applications must be defined at the start, but

Table 5
IoT Middleware comparison.

Middleware	Communication model	Discovery	Connection with devices	Real-time	Devices requirements	Virtual devices	QoS supported	Security	License
DDS	Peer-to-peer	Yes	Protocol instantiation	★★★	High/ Medium/ Low	No	Yes	DDS-Security (Authentication, Privacy, Access Control, Logging and Data Tagging)	Different implementations (enterprise and open source)
GSN	Peer-to-peer	Yes	Wrappers for TinyOs platforms, HTTP generic, several devices, and generic UDP and serial	★	Medium/ Low	Yes	No	N/A	GNU General Public License Version 3 (or later)
DPWS	Client-server	Yes	Protocol instantiation	★	High	No	No	WS-Security (Integrity, Confidentiality, Authentication)	Different implementations (enterprise and open source)
CoAP	Client-server asynchronous	Yes	Protocol instantiation	★★★	Low	No	No	DTLS (Authentication, Privacy) Different implementations (enterprise and open source)	Different implementations (enterprise and open source)
LinkSmart	Peer-to-peer	Yes	Protocol instantiation or connection through the proxy	★	High/Low	Yes	No	Crypto Manager (Authentication, Privacy)	GNU Affero GPL v3
LooCI	Peer-to-peer	Yes	Protocol instantiation or wrapper with devices	★	High/ Medium /Low	No	No	Specification on SecLooci	GNU General Public License version 3

at the same time this offers a safety type which is not present in the remaining middleware. Moreover, LinkSmart and LooCI are good choices when dynamic management and deployment of components are required and DDS and CoAP are better for real-time support. Lastly, GSN, LinkSmart and LooCI have open source licenses and the rest have both enterprises as open source implementations.

3. Integration existing proposals

In this section, different existing proposals for Cloud Computing and IoT integration are summarized. The proposals cover research projects, enterprise products and open source projects in multiple areas, so they form a multidisciplinary set of existing solutions in this field.

OpenIoT (https://github.com/OpenIotOrg/openiot) is an open source middleware, co-funded by the European Union's Seventh Framework Programme, for getting information about sensors, actuators and smart devices and offering utility-based IoT services in a cloud platform. OpenIoT leverages the state of the art on middleware frameworks of RFID/WSN and IoT like XGSN and AspireRFID. Virtual sensors are also promoted by OpenIoT, enabling the concept of Sensing-as-a-Service. The main inclusion is the semantics structure, using ontologies, enabling semantics interaction and interoperability between external systems and offering Open Linked Data interfaces. OpenIoT has also designed a set of applications over the cloud platform, to enable on-the-fly specification of services requests to the OpenIoT platform, data visualization, and configuration and monitoring components over the sensors and OpenIoT services.

The SENSEI project (http://www.ict-sensei.org/; Tsiatsis et al., 2010) is also an Integrated Project in the EU's Seventh Framework Programme, which aims to integrate heterogeneous WSAN (Wireless Sensor and Actor Networks) in an open business-drive architecture in order to offer services and applications on them. SENSEI's result is a secure marketplace where users can manage and access information about WSAN resources. The Resource is the main concept in the architecture, which abstracts sensors, actuators, processors and software components through a semantic specification at the same time that offers a set of services. SENSEI also offers an Open Service Interface and a management UI designed to enable machine processing which facilitates the dynamic composition, discovery and instantiation of new services. For communication and management with end points, SENSEI makes use of the Resource End Point (REP), which makes resources on embedded devices or sensors available to the SENSEI system through Restful interfaces. The REPs can be directly deployed on devices through TinyOS, Contiky, and Android implementations or by REP gateways which act as a resource proxy between the underlying WSAN and the SENSEI systems through the HTTP-CoAP, HTTP-Socket and HTTP-USB conversions.

Nimbits (http://www.nimbits.com/) is an open source platform for connecting things in the cloud, and one another. At present, Nimbits can be downloaded and installed privately in addition to using a public cloud Nimbits deployed in Google App Engine. Nimbits is composed of two main components: a web server which records and processes geo and time stamped data, and executes user rules on the data such as push notification, emails and XMPP messages; and a Java library for developing and connecting new applications on the platform. Moreover, Nimbits contains a library for Arduino support and an Android App to manage and visualize all connected data prints. The data is sent by clients using the JSON (JavaScript Object Notation) format. Users can configure data points to behave in many cascading ways when new data is recorded, generating different triggers and alerts in

each situation. Several parts of Nimbits are provided with an open source license, but the core components do not contain an open source license, though the latter can be downloaded for free now.

Xively (https://xively.com/) is an IoT Platform as a Service for developing applications over connected things. Xively offers different communication methods for connecting things, like RESTFul, HTTP, Sockets and MQTT (Message Queue Telemetry Transport). Different data formats such as JSON, XML and CSV, and official libraries for dozens of languages and platforms like Arduino, Android, Java and C. The Xively API has been built to read and write data and manage products and devices. Xively has been designed to support a development process in three stages: development for testing devices and applications, deployment for turning prototypes into products, and management for batching products and support real-time devices. This can be done by the web UI offered by Xively. Furthermore, the platform contains end-to-end security for protecting commutation channels and for fine-grained permissions. Xively has a proprietary license, allowing users and systems to connect with its platform.

Paraimpu (Pintus et al., 2012) introduces a novel paradigm, to share objects, data and functionalities with other people in order to attain a participative and collaborative use of a friend's things. Paraimpu is a web platform which allows the adding, use, interconnection and sharing of real smart objects and virtual things like services on the Web and social networks. Users can define connections between a sensor and an actuator through a sequence of rules and actions, and manage the thing's privacy. As a result, a user can publish on social media, like Facebook, an information message when a sensor's temperature exceeds a certain threshold. The platform promotes the concept of "Web of Thing", where multiple connected smart objects communicate using Web protocols. Currently, Paraimpu has a proprietary license and only allows things to be used or connected to its platform.

An integration with their own products has been released by Particle (https://www.particle.io/). Particle offers a set of development kits at low prices—starting from 19$ a microcontroller with Wi-Fi support—with a free hosted cloud platform for each device. The Particle team has obviously thought of the importance of IoT software and they have designed a set of software tools for developing, managing and monitoring your Particle devices. The software tools include a mobile app for remote control and monitoring, Web and local IDEs, a CLI Particle and programming support for Node.js, Arduino-like development and a REST API. However, the main advantage of Particle is that all their software contains an open source license and all are published on GitHub. Moreover, the solutions have also been integrated with several external systems such as a smart bed and a water monitoring system. The development kits have Wi-Fi and 2G/3G support, so together with their low price and the hosted cloud, they form a cheap choice to start with IoT and Cloud Computing.

As is Particle, Thinking Things (http://www.thinkingthings.telefonica.com/) is also a project founded by Telefonica with their own devices which aims to connect IoT devices and cloud technologies. Unlike Particle, Thinking Things has presented a set of modular development kits with integrated sensors—air temperature, air humidity and ambient light—making up a stack of connected blocks that you can put together like Lego pieces. Furthermore, a REST API and mechanisms to monitor and define action rules form part of the platform. Thinking Things also offers devices with Arduino-compatibility, with free cloud connection and utilization. A collaboration with a pizza company has concluded with the development of an embedded button to order pizzas in real-time. However, Thinking Things development kits are more expensive than Particle kits, they do not contain as many software tools as Particle and they only have connectivity support for GSM communications.

SensorCloud (http://www.sensorcloud.com/) leverages the Cloud Computing technologies to provide a data storage, visualization and management platform. For the IoT interconnection, the company MicroStrain offers a specific gateway which collects data from its sensors and pushes it to SensorCloud. Moreover, data from other IoT devices can be published on SensorCloud through the RESTful API offered. SensorCloud also incorporates the MathEngine—a set of software tools to process, analyze and monitor sensor data—a mechanism to upload you own scripts to process data and an alerts engine (as other integration platforms). Lastly, SensorCloud offers multiple plans to use its platform, from free ones with 25.000 transactions per month.

The concept of a secure and robust software agent is promoted by CloudPlugs (http://cloudplugs.com/). The agent, known as SmartPlug, connects with the CloudPlugs IoT platform in addition to having local intelligence for communication and to control other devices. CloudPlugs also has a pay-as-you-grow IoT platform with which you can manage and deploy the underlying IoT deployment and access your IoT data through a REST API. SmartPlug has also been designed to deploy applications over a Node.js server, offering communication support with local sensors through several local interfaces like ZigBee and Bluetooth and multi-operating system support. Final devices can connect directly to the CloudPlugs platform through several communication protocols and all things can communicate with each other independently from the protocol through a smart message bus in the platform. CloudPlugs is the only platform which besides the Cloud Computing integration offers mechanisms to support local intelligence. Although, CloudPlugs has a pay-as-you-grow platform, it also offers a free account to test the platform.

Stack4Things (Merlino et al., 2015) proposes an extension of the OpenStack platform in order to enable a cloud-oriented infrastructure for managing the IoT. Stack4Things relies on the OpenStack as a cloud platform as well as virtual infrastructure manager to provide higher level services applied to the IoT. The s4tProbe components constitute the end components in the system, which are deployed in Arduino YUN boards. The s4tProbe components connect the IoT with the OpenStack platform providing a pull-based design through the AQMP protocol and a push based design through CoAP. The OpenStack platform has been extended to support new UI functionalities, integrate s4tProbe components and provide complex event processing.

The comparison of integration proposals in Table 6 shows that all the surveyed proposals have REST API support and a management Web UI. The connection with devices is established in different ways. On the one hand Nimbits, Paraimpu and Sensor Cloud just rely on a RESTFul API whereas Particle and Stack4Things rely on the lightweight CoAP protocol. Stack4Things also relies on AQMP for a pull-based design. OpenIoT, SENSEI and CloudPlugs rely on, respectively, middleware, gateways and agents to abstract the underlying IoT deployment. In the case of SENSEI and CloudPlugs, they also offer support for other protocols. Lastly, Xively allows communications with several protocols and Thinking Things abstracts the communication through its own devices.

Most integration proposals are available online for use and deployment of an IoT infrastructure except for SENSEI, Stack4Things and OpenIoT, but the latter has the source code available online. In the case of Nimbits, Thinking Things and Particle, it is free to use the cloud, but the last two require the use of their own devices for this. Moreover, Particle goes beyond and apart from the free use of cloud, all source code of the platform's components contain open source licenses and are available online. Stack4Things leverages the cloud features of a well established solution as OpenStack to build the system. Nimbits also contains an open source license for its source, but in this case only for some portions. For security aspects, most components make use of tokens

Table 6
Integration proposals comparison.

Platform	REST API	Management Web UI	Connection with devices	Online for use	Available code	Security	License
OpenIoT	Yes	Yes	X-GSN	No	Yes	OAuth 2.0 authentication, permissions and roles.	GNU General Public License version 3
SENSEI	Yes	Yes	CoAP, Sockets, USB and REP instantiation	No	No	Token authentication, permissions, roles	N/A
Nimbits	Yes	Yes	RESTful API	Yes	Partially	Authentication with token and user/password	Apache License Version 2.0 (portions)
Xively	Yes	Yes	RESTful API, Sockets, WebSockets and MQTT	Yes	No	Authentication and permissions with OAuth 2.0 and API keys. User/password authentication	End User License Agreement
Paraimpu	Yes	Yes	RESTful API	Yes	No	Token authentication	End User License Agreement
Particle	Yes	Yes	CoAP	Yes	Yes	Authentication with user/password and OAuth 2.0	Several open source licenses
Thinking things	Yes	Yes	N/A	Yes	No	Authentication with token and user/password	End User License Agreement
Sensor cloud	Yes	Yes	RESTful API	Yes	No	All transactions occur over TLS	End User License Agreement
CloudPLugs	Yes	Yes	MQTT, RESTful API, SmartPlug instantiation and WebSockets	Yes	No	SSL for communications and AES and RSA to protect user sources	End User License Agreement
Stack4Things	Provided by OpenStack	Yes	AMQP and CoAP	No	No	N/A	N/A

in order to allow a secure form of authentication, identification and permissions over a set of users or devices. On the other hand, SensorCloud and CloudPlugs protect their communication through secure protocols, but they do not make use of tokens as it can be overly weighty for embedded devices.

4. Data analytics techniques

Although the aims of processing, storing and representing large amounts of data in worldwide IoT environments can be provided by the integration of the IoT and Cloud Computing, there are still open concerns such as verify, normalize, filter and analyze IoT data. The lack of open standards, the large diversity of technologies involved in the IoT and the large amount of data generated require techniques to improve and optimize such integration issues.

Despite the cloud features, there are still some concerns about data security and user privacy. Data security is one of the biggest reason why people are unwilling to use the Cloud (Liu et al., 2015, 2014). Through data integrity verification techniques the data owners can verify if their data is maintained intact at the same time that providing data security. Moreover, data integrity verification techniques must be aligned with the Cloud and Big Data and be efficient in storage, communication and computation. A wide review of integrity protection and verification techniques from external parties is presented in Liu et al. (2015). The better evaluated technique is MuR-DPA Liu et al. (2014), which presents a public auditing verification scheme which support dynamic data updates and efficient verification. On the other hand, for ensuring the privacy even with TPAs (Third Party Auditor), in Wang et al. (2010) is proposed a privacy-preserving public auditing system for data storage security which guarantee that TPAs would not learn any knowledge about the data content.

On the other hand, the data compression of the high volume of data generated by the IoT can lighten the storage, transmission and processing tasks in this field. Furthermore, information in some environments such as smart city are not as important in terms of privacy and data loss as in critical systems and connected health. The data compression techniques follows different paradigms while the traditional compression philosophy is focused on reducing redundant information (Ukil et al., 2015). SensCompr (Ukil et al., 2015) focuses on extracting useful information from IoT data with continuous adaptation and information loss. Normal and usual activities usually do not provide extra information, but anomalous IoT data. Therefore, SensCompr applies outlier detection method to capture unusual patterns. Also, SensCompr is released to be done during data capture, in-network processing and on sensor data storage. On the other hand, the work in Li et al. (2013) proposed a CS (Compressed Sensing) framework with a different approach than SensCompr. Although the CS framework also has as purpose acquiring events of interest, the CS framework focuses on compressing signal in data acquisition networks and future reconstruction in the data analysis system.

Another technique for saving data and making estimations is data filtering. In this field, there are different approaches and some of them are specified for some systems (non-linear and linear systems). One of the most widely used data filter for non-linear system is the extended Kalman filter (EKF) (Julier et al., 1995). The EKF linearizes about an estimate of the current mean and covariance providing a recursive linear estimator. The collaborative filtering is another approach that is based on the collaboration between multiple agents or data sources for filtering information. The collaborative filtering has been used on many areas such as recommendation systems in order to predict suitable recommendations for end users. An algorithmic framework for performing

collaborative filtering is presented in Herlocker et al. (1999). Furthermore, in environments where the user opinion is taken into account, data filtering can be helped in the decision process from users as realized in CrowdScreen (Parameswaran et al., 2012).

Even though semantic data and open standard middlewares can homogenize data, the great variety of data sources and the needs for integration data and making knowledge requires to unify them. Moreover, the variety is one of the main features of Big Data, so their processing and analysis processes as well as redundant and inconsistency issues can be improved with normalization techniques. Also as mentioned in Nayak et al. (2014) the data normalization is one of the fundamental data preprocessing steps for learning from data before feeding to the Artificial Neural Network (ANN). The min–max, z-score and decimal point normalizations are evaluated in Al Shalabi and Shaaban (2006). Besides the authors propose preference matrix to choose the best normalization method. The work in Nayak et al. (2014) also includes the above data normalizations techniques and includes the median, Sigmoid, Median and Median Absolute Deviation (MAD) and Tanh estimator normalizations techniques which have been evaluated in four ANN based forecasting models. The evaluation concludes that users must not adopt a single normalization technique, rather alternate methods should be considered to obtain better results.

The above techniques except verification are involved in the preprocessing step, which help in the next step, processing for extracting useful information. Analysis techniques are the main techniques in the IoT since they turn into raw data into useful information that can be used as knowledge. Data mining involves discovering of useful patterns from large amounts of data and applying algorithms for extracting useful and hidden information (Chen et al., 2015). The analysis techniques may contain different aims and may be useful for certain types of problems. In Chen et al. (2015) the data mining techniques are divided into five categories: classification, clustering, association analysis, time series analysis and outlier analysis. The classification techniques pretend to find models or functions that describe and distinguish data classes. Clustering analyzes data without consulting reference models. Association analysis aims for discovering association rules in data sets. Time series and outlier analysis pretend to analyze time series data and objects whose behavior changes over time, respectively. Depends on the problem type, a large amount of techniques can be chosen in Tsai et al. (2014) and Chen et al. (2015).

5. Case studies

The opportunities offered by the IoT are in continuous growth. Besides the large amount of systems and applications available for improving manufacturing and quality of our lives, a great amount of startups, hardware, and systems and so on related to the IoT are released every day. The great expansion of the IoT is due to the unbound possibilities which submits this field.

Smart home is probably the first application that comes to mind when thinking about IoT, and it is indeed a major field of research. Future homes will be fully monitored, this technology will make possible not just to know the average temperature of our home and the power and water consumption, but also the quality of air we breathe to automate the house ventilation, and the prediction of water or gas leaks or any structural failure.

Smart objects such as smart watches, activity trackers or smart glasses are also a hot trend. These objects are the key to make the IoT part of our daily life, in order to interact with other objects or to monitor ourselves. In the near future, the human body could be monitored making possible early detection of illnesses and its prediction, what's more internal sensors could be implanted to augment senses and capabilities.

Some cities also have bet for IoT to enhance quality and performance of urban services. These Smart cities offer a wide range of applications, from transport and traffic management to water distribution, including healthcare, waste management, energy, security or environmental monitoring (air and noise pollution monitoring or forest fire detection).

Therefore, the IoT is now one of the most interesting fields of research that will be present to help us every day. The following subsections show several case studies of some of the most relevant IoT applications. The case studies have been selected based on the work of IoT Analytics in the most popular IoT applications (http://iot-analytics.com/10-internet-of-things-applications/) shown in Fig. 2. Although, many applications share requirements and aims, we have divided the IoT applications in three categories taking into account its main requirements: real-time, for applications which contains time restrictions; data analysis, for applications focused in analyzing data; and device interaction, for applications focus on devices relations. On the one hand, the Connected Health and Smart Farming requires real-time monitoring of vital signs and the Smart Supply Chain needs real-time for an efficient trading. In Smart Home, Wearable and Industrial Internet device interaction is a key aim. Lastly, Smart Retail, Smart City and Smart Grid rely on data analysis to optimize business, cities and the electrical grids, respectively.

In the case studies, we have chosen one of the most relevant applications in each category, with an introduction of the case study and a discussion of the technologies surveyed for a properly deployment in each situation.

5.1. Connected health

Nowadays, there is a trend in many countries to reduce hospital resources and moving healthcare services like medical checks to home (Yang et al., 2014). Besides more work needs to be done on models and algorithms to utilize data for the decision-making activities of health care diagnosis and medical treatment (Yan et al., 2015). The Connected Health would lead to a reduction in the financial burden, a personal comfortable place for patients and a quick release of hospital resources in case of emergency. However, a new paradigm shift of health resources and financial burden would need to be rethought for achieving it in society. Establishing the mentioned environment requires real-time monitoring of vital signs as a necessary requirement for actuating as soon as possible to save lives. Moreover, analyzing normal vital signs could lead to early detection and the prediction of abnormal situations. Also checking medical records of all patient illnesses would prevent possible illnesses before it is too late. The system would also propose personal treatments based on the patient history and genetic information. The Connected Health scenario proposed is composed by a large deployment of smart units with vital signs sensors for monitoring the patients health.

Respect to IaaS platforms, the Connected Health needs high availability to ensure 24/7 availability, security to protect sensible data and auto-scaling to ensure that all data is processed and received in the system. The three IaaS platforms surveyed provide the requirements of high availability, security and auto-scaling. OpenStack can adapt better to the user's needs installing only the required components with its pluggable architecture meanwhile offering a multitude of component-as-a-service. Also OpenStack provides, in addition to the largest support for hypervisor, storage and networking components adjusting better to the user's needs, support with NoSQL databases providing a high level of high availability, in contrast to OpenNebula and CloudStack. On the other hand, OpenNebula can adapt better to the peak workloads thanks to the cloud federation without investing in hardware.

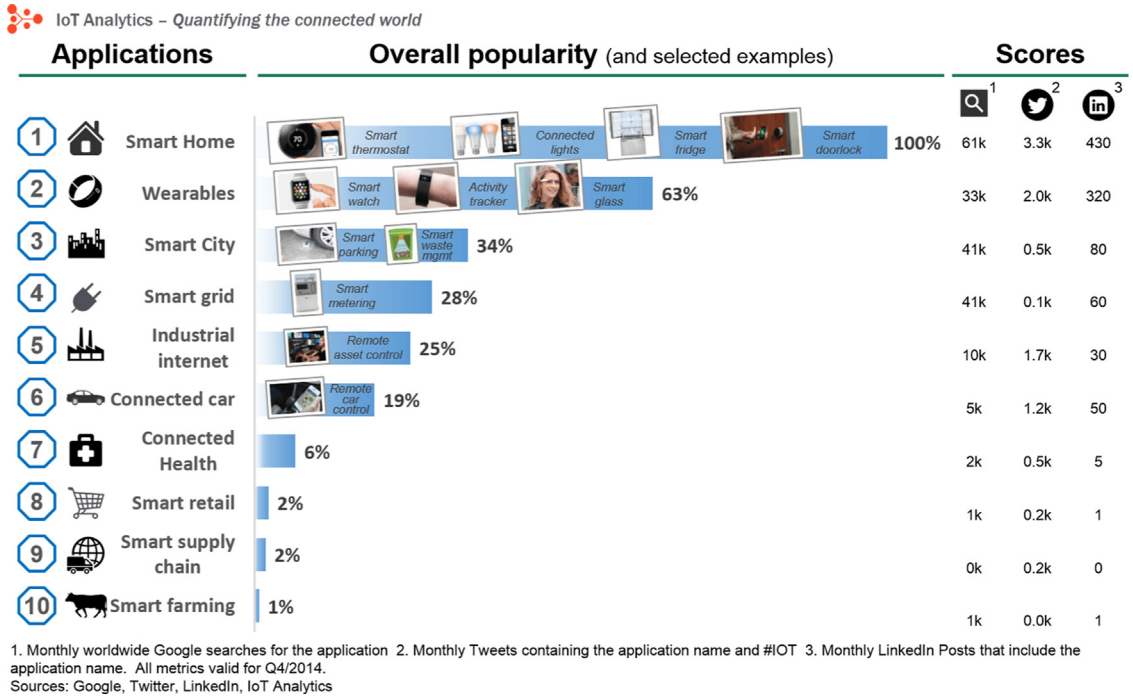

Fig. 2. The most popular Internet of Things applications right now.
Source: Permission authorized for publication by IoT Analytics.

Furthermore, the great amount of data stream received in the system requires real-time processing to detect abnormal situations. On the other hand, batch processing becomes established as a needed component for analyzing common data and preventing possible illnesses. Also all data in the system needs to be stored to be queried and accessed by the personal corresponding. A possible solution could be constituted by Apache Hadoop for achieving bath processing, Apache Storm for real-time processing and Apache Kafka to distribute the data between the cloud platforms with low latency. RabbitMQ can provide a high level of confidentiality through message confirmation, however the latency in Kafka is more desirable for real-time. Apache Hadoop could also be used as data storage, but other components like Apache HBase can be used for that and providing a SQL-like querying interface like the actual health systems. A solution for reducing managing and deployments costs, would be to use Apache Spark and its component Apache Spark Streaming for real-time and batch processing. Moreover, Apache Ambari can be used to orchestrate the deployment and management of cloud platforms, but in its current version does not include support for Apache Spark.

The IoT also needs to be abstracted by a middleware with QoS and real-time requirements. DDS grows stronger as suitable middleware for that purpose. However, some of the requirements of DDS can be too heavy for some of resource constrained devices. For that purpose, CoAP with connection confirmation would be a promising candidate even though does not provide QoS.

5.2. Smart home

As previously stated, Smart Home is one of the outstanding examples of the IoT. Nowadays, homes contain more and more smart objects and it will continue to grow. A properly orchestration between all objects connected at home could save money for the end users, adapt to their needs in addition to react from abnormal situations. Also the home consumption is one of the main household expenses and the Smart Home would lead to optimize it. For instance, temperature limits of the air conditioner and central heating can considerably increase the home consumption. However, when we get home we want a pleasant temperature both in summer as in winter and the home consumption is affected. Also the room temperature may change when the number of people increases or when the oven is turned on. Moreover, others aspects such as lighting and access control or human actions could also be connected. So the Smart Home can help us daily to save money with household expenses and adapt to our needs. The proposed scenario is composed of multiple Smart Homes with objects connected locally and support of Cloud Computing for monitoring and managing each case.

In this scenario high availability and security are not as important as Smart Healhtcare, but could be important for aspects like access control. On the contrary, the accounting may be a required requirement for monitoring the users consumption. Like the above scenario, the three surveyed provide such requirements. However, thanks to the large amount of component-as-a-service offered, OpenStack provides an extra level of service which can be offered to the end users.

Batch processing and querying large amounts of data are present in this scenario. Apache HBase and Druid can be used as data storage taking advantage of its integration with Apache Hadoop for batch processing. Druid also provides real-time requirements on ingesting and querying data. On the other hand, OpenTSDB can store and query large amount of time series data—solving the sorted key issue of HBase—as long as achieving data set processing with Apache Pig. Furthermore, Apache Hadoop and Apache Spark can be used as data storage with solutions such as Apache Pig and Apache Hive or Apache Spark SQL for providing high layers for accessing, respectively. Finally, as the Connected Health scenario, Apache Ambari can orchestrate the deployment and management of cloud platforms.

On the other hand, the middleware is the main component in the scenario. Discovery and peer-to-peer communication are desirable requirements for achieving device interaction. DDS, GSN, LinkSmart and LooCI satisfy such requirements. LooCI and LinkSmart would be suitable for dynamic environments thanks to their

dynamic configuration. On the other hand, DDS provides high levels of QoS and security for scenarios where such aspects are more important. GSN does not include security support, but offers communication with remote middlewares.

A different approach could integrate a Smart Home with humidity, temperature, lighting and physical presence with Thinking Things, or custom behaviors with Particle devices, both with free cloud utilization. Moreover, CloudPlugs with local intelligence, Xively or SensorCloud could be used for that.

5.3. Smart city

The increase of world population in urban environments, the shortage of natural resources, and the concerns about the climate change and the environment, have led to the cities to make its services and infrastructure more accessible, interactive and efficient (Pellicer et al., 2013) to tackle them. The Smart Cities pretend to improve the everyday life of a large amount of areas such as transportation (He et al., 2014), public safety, urban consumption, tourism, urban planning and so on. The large amount of subareas belong to Smart City have originated that many Smart City approach addressed have different aims. For instance, the initial efforts of Málaga and Amsterdam were focused on promoting renewable energy generation, Madrid and Stockholm were focused on traffic management, and public safety and Santander and Göteborg on comprehensive communications infrastructure. Therefore, Smart City is one of the fields with the largest scope and diversity of the IoT. Moreover, variations of one of the factors involved in Smart Cities may affect to the rest, so a strong orchestration of the process involved optimize the aggregation. The proposed scenario in Smart City is composed by a strong orchestration of a large amount of service in order to optimize the city life.

The initiative of many Smart Cities to offer some collected data as open data, requires providing a QoS level. The requirements of high availability, auto-scaling and load balancing are provided by the infrastructures surveyed as already stated. Moreover, both OpenStack, CloudStack and OpenNebula support the Open Cloud Computing Interface (OCCI) (http://occi-wg.org/). OCCI enables an open specification and a flexible API for cloud task such as integration, portability and interoperability. So OCCI offers mechanisms to make easier integrations or migrations between cloud infrastructures.

Analyzing and making intelligence with an orchestration between the services deployed are the key requirements in this scenario. Moreover, new paradigms need to be addressed in order to allow semantic and open data. Apache Spark provides a batch processing framework which can process data 100 × faster than Apache Apache Hadoop. Fast processing in machine learning can generate knowledge more trained in less time so it will adjust itself better to prevent events in others areas. Moreover, many machine learning algorithms are available for Apache Spark through Apache Mahout.

In this Smart City scenario a middleware which facilities the task for adding context information would help to provide semantic and open data. Although many solutions incorporate a gateway for facilitating that process, the gateway inclusion in Smart City can assume a bottleneck due to components fails or human attacks. DDS, XGSN and DPWS provides high layers to facilitate the inclusion of semantic data. XGSN, in fact, enriches virtual sensors with semantic data by means of an extension of the SSN ontology. DPWS could require more devices requirements than XGSN and DDS. On the other hand, DDS and DPWS require a data format communication definition, so it provides a level of safety type. DDS also is the only one that takes into account QoS and real-time requirements.

In the case of an OpenStack deployment that has already been done, another solution could consist in the proposed scenario in Stack4Things in order to leverage the OpenStack cloud features.

6. Challenges and open research issues

Security and Privacy are key challenges in the deployment of IoT infrastructures. IoT devices are normally associated with constrained devices, so they are more vulnerable to attacks and threats. On the other hand, in many situations IoT systems use sensitive information like personal information or critical infrastructures, thus privacy with devices, cloud and network are key aspects. Roman et al. (2013) mentioned the importance of security and privacy to push the Internet of Things distributed approach into real world. They enumerated the security mechanisms that can be integrated in the IoT: protocol and network security that offers end-to-end secure communication mechanism; identity management to achieve authentication and authorization to assure that the data is produced by a certain entity and to restrict the access control; privacy over data generated; trust between entities and users interaction and governance to support political decisions and stability; and fault tolerance to prevent and to detect attacks. Lastly, and the most difficult to avoid are attack models in the IoT such as DoS (Denial of service), physical damage, eavesdropping, node capture to extract information and controlling entities. Moreover, security and privacy are one of the main concerns on the Cloud adoption.

Ipv6: The Internet is the main component of the Internet of Things, and everyone knows its addressable limitations on IPv4. Moreover, the adoption of technologies like CoAP that allow interaction with embedded devices directly from the Internet, and the continued growth of the latter over the coming years, represent a greater effort to get rid of the Network Address translation (NAT) mechanisms and to address each thing or service in the world with a unique IP address. IPv6 has been designed to solve this problem through an IP 128 bit address, bringing with it several advantages such as a native integration with the Internet, end-to-end connectivity and a compliance with open REST interfaces (Ziegler et al., 2014). In order to adopt the IPv6 in the embedded devices of the IoT, 6LoWPAN and ZigBee IP are suitable specifications. Nevertheless, at the present time there are not too many commercial platforms that implant these specifications. With the adoption of IoT, we are moving from networks with human initiated activities towards networks in which machine-to-human and machine-to-machine communications will be more numerous and IPv6 will pave the way.

Fog Computing: Even though Cloud Computing can help avoid some IoT limitations, there are situations like mobility support, geo-distribution, location awareness and low latency that need to be addressed and Cloud Computing lacks means to tackle them. A new platform, called Fog Computing (Aazam and Huh, 2014), wants to provide storage, computing, and networking services between Cloud Computing and end devices. It is named Fog because fog is a cloud close to the ground and its main purpose is to extend Cloud Computing to bring it closer to IoT devices. In certain situations, data is not required for the Cloud or must be processed with very low latency and mobility, so Fog Computing can provide the necessary requirements in IoT through a distributed and collaborative platform in collaboration with IoT devices. Notwithstanding, due to IoT limitations, Fog Computing cannot provide functionalities such as complex analysis, data access to large numbers of users and storing historical data, which is complemented with Cloud Computing.

Lambda Architecture: The Lambda Architecture (LA) (Marz and Warren, 2015) is a paradigm composed by cloud platforms—a

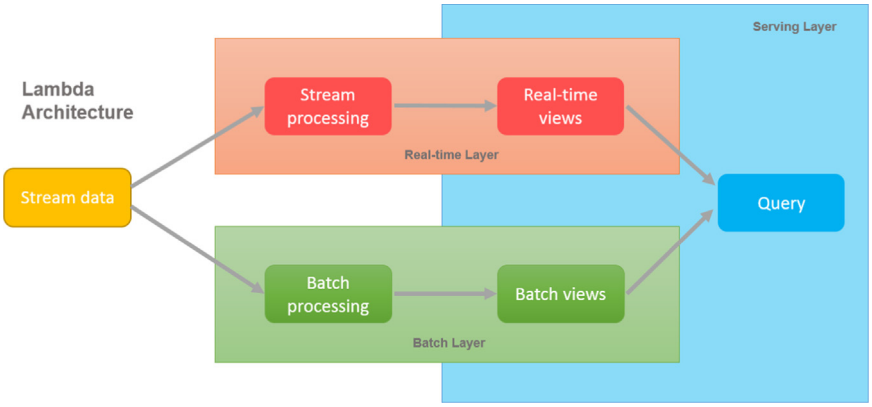

Fig. 3. Lambda architecture.

distributed queue, batch and stream processing and distributed data stores—designed to offer arbitrary queries over arbitrary real-time data. Due to the temporal characteristics of IoT data, and the need to extract knowledge and predictions of historical data, even in some situations with real-time, we believe that the LA is a right paradigm for IoT, which will bring the following benefits: storage, real-time processing, scalability and machine learning. Real-time processing is a necessary requirement for many situations, for example, critical infrastructures and health systems. Moreover, in other scenarios, the true knowledge resides in the data, since it can be used to prevent certain situations from arising and to act in advance, and the LA stimulates these scenarios. Furthermore, the LA abilities for large-scale smart environment management and Big Data storage/analytics are show in Villari et al. (2014). Apart from these benefits, LA is not centered on any particular technology, so each user can choose the necessary technology based on his/her needs. Fig. 3 shows the surveyed Lambda Architecture.

Interoperability: The large number of devices and technologies in IoT makes interoperability a key issue in this field. The IoT and Cloud Computing integration can partly solve this problem, but in some situations like the aforementioned Fog Computing, the interoperability is a necessary requirement. Moreover, many companies use the interoperability to obtain unique products that are only compatible with each other. The research in this field for a greater number of standards, and their adoption by enterprise components will enable the creation of smart scenarios with heterogeneity devices without having to worry about interoperability.

Context-Aware Computing: Context-Aware computing has been acquired importance thanks to the growth and the IoT and its potential of it. Context-aware computing allows us to store context information linked to sensor data allowing an easy and meaningful interpretation (Perera et al., 2014). Moreover, the European Union has identified context awareness as an important research area for context-aware IoT computing (Da Xu et al., 2014). Context-Aware computing would help the IoT with new information that can be used for new applications and obtaining a knowledge better founded. Context-Aware computing also reopens the original use of ontologies as sources of knowledge, remaining a pending issue its properly integration with cloud computing.

7. Conclusions

The IoT is an emerging technology that is gradually moving towards forming part of many facets of our lives. The multifaceted IoT, and the increasingly large number of devices, technologies and platforms in this field, have led IoT to be a global and extended technology in many areas. However, due to the limitations of the IoT as presented in this paper and the need for complex features to address existing demands, current technologies, like Cloud Computing, are appropriate as a complement in this field.

We have thought of an integration based on several components: Cloud platforms, Cloud infrastructures and IoT middleware. The Cloud platforms are the components responsible for providing the IoT with the necessary and current requirements such as real-time processing, scalable storage and global access, as well as expansion towards others opportunities like machine learning. The Cloud platforms have different aims and have been divided in several categories such as batch processing, distributed databases, distributed queues, real-time processing and platforms for helping in the management, monitoring and deployment tasks. However, the Cloud platforms are not discriminatory among them and can be orchestrated together such as the LA. In order to monitor, manage, and offer an infrastructure to deploy Cloud platforms, different Cloud infrastructures have been surveyed. The Cloud infrastructures provides the storage, networking and computing resources required by the IoT and the Cloud platforms. And lastly, the IoT middleware provides an abstraction layer for the underlying IoT devices, as well as mechanisms for interacting with Cloud Computing.

There exists different integration proposals in the literature. We have analyzed both research projects, enterprise products as open source projects in multiple areas in order to provide an overview of existing solutions. On the other hand, preprocessing data techniques and data mining algorithms also have been surveyed in order to complement such integration. The techniques includes both preprocessing method and data mining algorithms in order to normalize, verify, filter, compress and analyze the large amount of data from the IoT.

Furthermore, we have taken into account the requirements of real-time and critical applications for the desired integration and we have surveyed components like real-time processing Cloud platforms, high-availability and auto-scaling Cloud infrastructures, and real-time and lightweight IoT middleware. Some case studies have been analyzed in order to discuss the components surveyed. Nonetheless, the components surveyed are not the only requirements which should be taken into consideration, since security, networking and interoperability aspects are key challenges in an IoT deployment, but many of the components surveyed have been designed with this in mind.

The IoT and Cloud Computing still have concerns about privacy and security, and the lack of interoperability is presented in both areas. Approaches such as Fog Computing, the Lambda Architecture and Ipv6 could be considered for such integration. This paper offers an insight into the proposed components for integrating the Internet of Things with Cloud Computing. Based on the comparisons made and the elements surveyed, users can select the necessary elements based on their own needs in order to obtain a

seamless integration. Most of the surveyed components have open source licenses, so any user can develop a platform with these components and survey new components to improve this paper and the research in this field.

Author contributions

All authors contributed to the conception of the proposal, defined the methodology and contributed to write and revise the article.

Acknowledgements

This work was funded by the Spanish projects TIC-1572 ("MIsTIca: Critical Infrastructures Monitoring based on Wireless Technologies") and TIN2014-52034-R ("An MDE Framework for the Design and integration of Critical Infrastructure Management Systems").

References

Aazam M, Huh E-N. Fog computing and smart gateway based communication for cloud of things. In: Proceedings of the 2nd international conference on future internet of things and cloud (FiCloud-2014), Barcelona, Spain; August 2014. p. 27–9.

Aazam M, Khan I, Alsaffar AA, Huh E-N. Cloud of things: integrating internet of things and cloud computing and the issues involved. In: Applied sciences and technology (IBCAST), 2014 11th International Bhurban conference on Anchorage, Alaska, USA: IEEE; June 2014. p. 414–9.

Aberer K, Hauswirth M, Salehi A. The global sensor networks middleware for efficient and flexible deployment and interconnection of sensor networks. Technical Report, Submitted to ACM/IFIP/USENIX 7th International Middleware Conference; 2006.

Al Shalabi L, Shaaban Z. Normalization as a preprocessing engine for data mining and the approach of preference matrix. In: Dependability of computer systems, 2006. DepCos-RELCOMEX'06. International Conference on Szklarska Poreba, Poland, IEEE; 2006. p. 207–14.

Apache ambari, Available online: ⟨http://ambari.apache.org/⟩, (accessed on 20 December 2015).

Apache cloudstack, Available online: ⟨http://cloudstack.apache.org/ ⟩, (accessed on 20 December 2015).

Apache hbase, Available online: ⟨http://hbase.apache.org/⟩, (accessed on 20 December 2015).

Apache hive, Available online: ⟨https://hive.apache.org/⟩, (accessed on 20 December 2015).

Apache kafka, Available online: ⟨http://kafka.apache.org/⟩, (accessed on 20 December 2015).

Apache phoenix, Available online: ⟨http://phoenix.apache.org/ ⟩, (accessed on 20 December 2015).

Apache pig, Available online: ⟨https://pig.apache.org/⟩, (20 December 2015).

Apache spark, Available online: ⟨https://spark.apache.org/⟩, (accessed on 20 December 2015).

Ashton K. That 'internet of things' thing. RFiD J. 2009;22(7):97–114.

Barbaran J, Diaz M, Rubio B. A virtual channel-based framework for the integration of wireless sensor networks in the cloud. In: Proceedings of the 2nd international conference on future internet of things and cloud (FiCloud-2014). Barcelona, Spain; Aug 2014. p. 334–9. http://dx.doi.org/10.1109/FiCloud.2014.59.

Bormann C, Castellani AP, Shelby Z. Coap: an application protocol for billions of tiny internet nodes. IEEE Internet Comput 2012;16(2):62–7.

Botta A, de Donato W, Persico V, Pescapé A. On the integration of cloud computing and internet of things. In: Proceedings of the 2nd international conference on future internet of things and cloud (FiCloud-2014), Barcelona, Spain; August 2014. p. 27–9.

Calbimonte J-P, Sarni S, Eberle J, Aberer K. Xgsn: an open-source semantic sensing middleware for the web of things. In: 7th international workshop on semantic sensor networks. Riva del Garda, Trentino, Italy, no. EPFL-CONF-200926; October 2014.

Castro M, Jara AJ, Skarmeta AF. Enabling end-to-end coap-based communications for the web of things. J Netw Comput Appl 2014, http://dxdoi.org/10.1016/j.jnca.2014.09.019.

Chen F, Deng P, Wan J, Zhang D, Vasilakos AV, Rong X. Data mining for the internet of things: literature review and challenges. Int J Distrib Sensor Netw 2015;501:431047.

Cloudera impala, Available online: ⟨http://www.cloudera.com/content/cloudera/en/products-and-services/cdh/impala.html⟩, (accessed on 20 December 2015).

Cloudplugs, Available online: ⟨http://cloudplugs.com/ ⟩, (accessed on 20 December 2015).

Coap implementations, Available online: ⟨http://coap.technology/impls.html⟩, (accessed on 20 December 2015).

Compton M, Barnaghi P, Bermudez L, GarcíA-Castro R, Corcho O, Cox S, Graybeal J, Hauswirth M, Henson C, Herzog A, et al. The ssn ontology of the w3c semantic sensor network incubator group. Web Semant Sci Serv Agents World Wide Web 2012;17:25–32.

Cubo J, Nieto A, Pimentel E. A cloud-based internet of things platform for ambient assisted living. Sensors 2014;14(8):14070–105.

Da Xu L, He W, Li S. Internet of things in industries: a survey. IEEE Trans Ind Inform 2014;10(4):2233–43.

The omg data-distribution service for real-time systems (dds), Available online: ⟨http://portals.omg.org/dds/⟩, (accessed on 20 December 2015).

Dossot D. RabbitMQ essentials. Birmingham, United Kingdom: Packt Publishing Ltd; 2014.

Druid, Available online: ⟨http://druid.io/⟩, (accessed on 20 December 2015).

Web services for devices, Available online: ⟨http://ws4d.org/ ⟩, (accessed on 20 December 2015).

Global sensor network, Available online: ⟨https://github.com/LSIR/gsn/wiki⟩, (accessed on 20 December 2015).

Gubbi J, Buyya R, Marusic S, Palaniswami M. Internet of things (iot): a vision, architectural elements, and future directions. Futur Gener Comput Syst 2013;29(7):1645–60.

He W, Yan G, Da Xu L. Developing vehicular data cloud services in the iot environment. IEEE Trans Ind Inform 2014;10(2):1587–95.

Herlocker JL, Konstan JA, Borchers A, Riedl J. An algorithmic framework for performing collaborative filtering. In: Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval. Berkeley, CA, USA: ACM; 1999. p. 230–7.

Hughes D, Man KL, Shen Z, Kim KK. A loosely-coupled binding model for wireless sensor networks. In: SoC design conference (ISOCC), 2012 international. Jeju Island, Korea (South): IEEE; Nov 2012. p. 273–6.

The 10 most popular internet of things applications right now, Available online: ⟨http://iot-analytics.com/10-internet-of-things-applications/ ⟩, (accessed on 20 Dec 2015).

Jammes F, Mensch A, Smit H. Service-oriented device communications using the devices profile for web services. In: Proceedings of the 3rd international workshop on middleware for pervasive and ad-hoc computing. Grenoble, France: ACM; 2005. p. 1–8.

Julier SJ, Uhlmann JK, Durrant-Whyte HF. A new approach for filtering nonlinear systems. In: American control conference, proceedings of the 1995. Seattle, WA, USA, vol. 3, IEEE; 1995. p. 1628–32.

Kreps J, Narkhede N, Rao J, et al. Kafka: a distributed messaging system for log processing. In: Proceedings of 6th international workshop on networking meets databases (NetDB). Athens, Greece; 2011.

Le-Phuoc D, Nguyen-Mau HQ, Parreira JX, Hauswirth M. A middleware framework for scalable management of linked streams. Web Semant Sci Serv Agents World Wide Web 2012;16:42–51.

Levä T, Mazhelis O, Suomi H. Comparing the cost-efficiency of coap and http in web of things applications. Decis Support Syst 2014;63:23–38.

Li S, Da Xu L, Wang X. Compressed sensing signal and data acquisition in wireless sensor networks and internet of things. IEEE Trans Ind Inform 2013;9(4):2177–86.

Linksmart, Available online: ⟨https://linksmart.eu/⟩, (accessed on 20 December 2015).

Liu C, Ranjan R, Yang C, Zhang X, Wang L, Chen J. Mur-dpa: top-down levelled multi-replica merkle hash tree based secure public auditing for dynamic big data storage on cloud. IEEE Trans Comput (1) 2014:1–1.

Liu C, Yang C, Zhang X, Chen J. External integrity verification for outsourced big data in cloud and iot: a big picture. Future Gener Comput Syst 2015;49:58–67.

Ludovici A, Calveras A. A proxy design to leverage the interconnection of coap wireless sensor networks with web applications. Sensors 2015;15(1):1217–44.

Madria S, Kumar V, Dalvi R. Sensor cloud: a cloud of virtual sensors. IEEE Softw 2014;31(2):70–7.

Maerien J, Michiels S, Hughes D, Huygens C, Joosen W. Seclooci: a comprehensive security middleware architecture for shared wireless sensor networks. Ad Hoc Netw 2015;25:141–69.

Mapreduce tutorial, Available online: ⟨http://hadoop.apache.org/docs/r1.2.1/mapred_tutorial.html ⟩, (accessed on 20 December 2015).

Marz N, Warren J. Big data: principles and best practices of scalable realtime data systems. Cherry Hill, NJ, 08003 United States: Manning Publications Co.; 2015.

Merlino G, Bruneo D, Distefano S, Longo F, Puliafito A, Al-Anbuky A. A smart city lighting case study on an openstack-powered infrastructure. Sensors 2015;15(7):16314–35.

Microsoft web services on devices, Available online: ⟨https://msdn.microsoft.com/en-us/library/windows/desktop/aa826001(v=vs.85).aspx, (accessed on 20 December 2015).

Moreno-Vozmediano R, Montero RS, Llorente IM. Iaas cloud architecture: from virtualized datacenters to federated cloud infrastructures. Computer 2012;12:65–72.

Nayak S, Misra B, Behera H. Impact of data normalization on stock index forecasting. Int J Comp Inf Syst Ind Manag Appl 2014;6:357–69.

Nimbits, Available online: ⟨http://www.nimbits.com/⟩, (accessed on 20 December 2015).

Open cloud computing interface, Available online: ⟨http://occi-wg.org/ ⟩, (accessed on 20 Dec 2015).

Openiot, Available online: ⟨https://github.com/OpenIotOrg/openiot ⟩, (accessed on 20 December 2015).
Opennebula, Available online: ⟨http://opennebula.org/⟩, (accessed on 20 December 2015).
Openstack, Available online: https://www.openstack.org/ ⟩, (accessed on 20 December 2015).
Opentsdb, Available online: ⟨http://opentsdb.net/⟩, (accessed on 20 December 2015).
Parameswaran AG, Garcia-Molina H, Park H, Polyzotis N, Ramesh A, Widom J. Crowdscreen: algorithms for filtering data with humans. In: Proceedings of the 2012 ACM SIGMOD international conference on management of data. Scottsdale, AZ, USA: ACM; 2012. p. 361–72.
Particle, Available online: ⟨https://www.particle.io/, (accessed on 20 December 2015).
Pellicer S, Santa G, Bleda AL, Maestre R, Jara AJ, Gomez Skarmeta A. A global perspective of smart cities: a survey. In: Innovative mobile and internet services in ubiquitous computing (IMIS), 2013 Seventh International Conference on, Taichung, Taiwan, IEEE; 2013. p. 439–44.
Perera C, Zaslavsky A, Christen P, Georgakopoulos D. Context aware computing for the internet of things: a survey. IEEE Commun Surv Tutor 2014;16(1):414–54.
Pintus A, Carboni D, Piras A. Paraimpu: a platform for a social web of things. In: Proceedings of the 21st international conference companion on World Wide Web. Lyon, France: ACM; April 2012. p. 401–4.
Rabbitmq, Available online: ⟨https://www.rabbitmq.com/⟩, (accessed on 20 December 2015).
Roman R, Zhou J, Lopez J. On the features and challenges of security and privacy in distributed internet of things. Comput Netw 2013;57(10):2266–79.
Sensei, Available online: ⟨http://www.ict-sensei.org/ ⟩, (accessed on 28 May 2015).
Sensorcloud, Available online: ⟨http://www.sensorcloud.com/⟩, (accessed on 20 December 2015).
Shelby KH, Zach, Bormann C. The constrained application protocol (coap), Available online: ⟨https://tools.ietf.org/html/rfc7252/⟩, (accessed on 20 December 2015).
Shvachko K, Kuang H, Radia S, Chansler R. The hadoop distributed file system. In: Mass storage systems and technologies (MSST). 2010 IEEE 26th Symposium on Incline Villiage, Nevada, USA; May 2010. p. 1–10. http://dx.doi.org/10.1109/MSST.2010.5496972.
Sindhanaiselvan K, Mekala T. A survey on sensor cloud: architecture and applications. In: International journal of P2P network trends and technology (IJPTT). p. 1–6.
Singh D, Tripathi G, Jara AJ. A survey of internet-of-things: future vision, architecture, challenges and services. In: 2014 IEEE world forum on Internet of things (WF-IoT), Seoul, Korea (South): IEEE; March 2014. p. 287–92.
Sleman A, Moeller R. Integration of wireless sensor network services into other home and industrial networks; using device profile for web services (dpws). In: 3rd International Conference on Information and communication technologies: from theory to applications, 2008. ICTTA 2008, Damascus, Syria: IEEE; April 2008. p. 1–5.
Apache spark streaming, Available online: ⟨https://spark.apache.org/streaming/⟩, (accessed on 20 December 2015).
Thinking things, Available online: ⟨http://www.thinkingthings.telefonica.com/, (accessed on 20 December 2015).
Toshniwal A, Taneja S, Shukla A, Ramasamy K, Patel JM, Kulkarni S, Jackson J, Gade K, Fu M, Donham J, et al. Storm@ twitter. In: Proceedings of the 2014 ACM SIGMOD international conference on management of data. Snowbird, Utah, USA: ACM; 2014. p. 147–56.
Tsai C-W, Lai C-F, Chiang M-C, Yang LT. Data mining for internet of things: a survey. IEEE Commun Surv Tutor 2014;16(1):77–97.
Tsiatsis V, Gluhak A, Bauge T, Montagut F, Bernat J, Bauer M, Villalonga C, Barnaghi PM, Krco S. The sensor real world internet architecture. In: Future internet assembly. Ghent, Belgium; Dec 2010. p. 247–56.
Tusa F, Paone M, Villari M, Puliafito A. Clever: a cloud-enabled virtual environment. In: 2010 IEEE Symposium on Computers and communications (ISCC), Riccione, Italy: IEEE; 2010. p. 477–82.
Tusa F, Celesti A, Villari M, Puliafito A. Federation between clever clouds through sasl/shibboleth authentication. In: The fourth international conference on evolving internet. Venice, Italy, no. 4; 2012. p. 12–7.
Ukil A, Bandyopadhyay S, Pal A. Iot data compression: sensor-agnostic approach. In: Data compression conference (DCC), 2015, Snowbird, UT, USA: IEEE; 2015. p. 303–12.
Vavilapalli VK, Murthy AC, Douglas C, Agarwal S, Konar M, Evans R, Graves T, Lowe J, Shah H, Seth S, et al. Apache hadoop yarn: yet another resource negotiator. In: Proceedings of the 4th annual symposium on cloud computing. Santa Clara, CA, USA: ACM; Oct 2013. p. 5.
Villari M, Celesti A, Fazio M, Puliafito A. Alljoyn lambda: an architecture for the management of smart environments in iot. In: 2014 International Conference on Smart computing workshops (SMARTCOMP Workshops), Hong Kong, China, IEEE; 2014. p. 9–14.
Wang C, Wang Q, Ren K, Lou W. Privacy-preserving public auditing for data storage security in cloud computing. In INFOCOM, 2010 Proceedings IEEE, San Diego, California, USA: IEEE; 2010. p. 1–9.
Xively-public cloud for internet of things, Available online: ⟨https://xively.com/⟩, (accessed on 20 December 2015).
Yan H, Xu LD, Bi Z, Pang Z, Zhang J, Chen Y. An emerging technology-wearable wireless sensor networks with applications in human health condition monitoring. J Manag Anal 2015;2:121–37. http://dx.doi.org/10.1080/23270012.2015.1029550.
Yang F, Tschetter E, Léauté X, Ray N, Merlino G, Ganguli D. Druid: a real-time analytical data store. In: Proceedings of the 2014 ACM SIGMOD international conference on management of data. ACM; 2014. p. 157–68.
Yang G, Xie L, Mantysalo M, Zhou X, Pang Z, Da Xu L, Kao-Walter S, Chen Q, Zheng L-R. A health-iot platform based on the integration of intelligent packaging, unobtrusive bio-sensor, and intelligent medicine box. IEEE Trans Ind Inform 2014;10(4):2180–91.
Zaharia M, Chowdhury M, Franklin MJ, Shenker S, Stoica I. Spark: cluster computing with working sets. In: Proceedings of the 2nd USENIX conference on Hot topics in cloud computing. Boston, MA, USA; June 2010. p. 10–10.
Zaharia M, Das T, Li H, Shenker S, Stoica I. Discretized streams: an efficient and fault-tolerant model for stream processing on large clusters. In: Proceedings of the 4th USENIX conference on Hot Topics in Cloud Computing. Boston, MA, USA, USENIX Association; June 2012. p. 10–10.
Zaslavsky AB, Perera C, Georgakopoulos D. Sensing as a service and big data. In: International conference on advances in cloud computing (ACC-2012). Bangalore, India; July 2012. p. 21–9.
Ziegler S, Crettaz C, Thomas I. Ipv6 as a global addressing scheme and integrator for the internet of things and the cloud. In: 2014 28th International Conference on Advanced information networking and applications workshops (WAINA), Victoria, Canada: IEEE; May 2014. p. 797–802.
Zorzi M, Gluhak A, Lange S, Bassi A. From today's intranet of things to a future internet of things: a wireless-and mobility-related view. IEEE Wirel Commun 2010;17(6):44–51.