Cloud Computing in Smart Irrigation

1. Introduction

Cloud computing relies on sharing of resources to achieve coherence, reduced investments and proportional costs, increased scalability, increased availability and reliability similar to a utility (like the electricity grid) over a network. At the foundation of cloud computing is the broader concept of converged infrastructure and shared services.

Cloud computing focuses on maximizing the effectiveness of the shared resources. Cloud resources are usually not only shared by multiple users but are also dynamically reallocated per demand. This can work for allocating resources to users.

2. Role of Cloud Computing in Improvement of IoT

IoT can benefit from the virtually unlimited capabilities and resources of Cloud to compensate its technological constraints (e.g., storage, processing and energy). Specifically, the Cloud offers an effective solution to implement IoT service management and composition as well as applications that exploit the things or the data produced by them. On the other hand, the Cloud can benefit from IoT by extending its scope to deal with real world things in a more distributed and dynamic manner and for delivering new services in a large number of real life scenarios. Essentially, the Cloud acts as intermediate layer between the things and the applications, where it hides all the complexity and the functionalities necessary to implement the latter. This framework will impact future application development, where information gathering, processing, and transmission will produce new challenges to be addressed, also in a multi-cloud environment.

I. Storage Resources

IoT involves by definition a large amount of information sources (i.e., the things), which produce a huge amount of non-structured or semi-structured data having the three characteristics typical of Big Data: volume, variety, and velocity (i.e., data generation frequency). Hence it implies collecting, accessing, processing, visualizing, archiving, sharing, and searching large amounts of data. Offering virtually unlimited, low-cost, and on-demand storage capacity, Cloud is the most convenient and cost effective solution to deal with data produced by IoT. This integration realizes a new convergence scenario where new opportunities arise for data aggregation, integration, and sharing with third parties. Once into the Cloud, data can be treated in a homogeneous manner through standard APIs, can be protected by applying top-level security, and directly accessed and visualized from any place.

II. Computational Resources

IoT devices have limited processing resources that do not allow on-site data processing. Data collected is usually transmitted to more powerful nodes where aggregation and processing is possible, but scalability is challenging to achieve without a proper infrastructure. The unlimited processing capabilities of Cloud and its on-demand model allow IoT processing needs to be properly satisfied and enable analyses of unprecedented complexity. Data-driven decision making and prediction algorithms would be possible at low cost and would provide increasing revenues and reduced risks. Other perspectives would be to perform real-time processing (on-the-fly) to implement scalable, real-time, collaborative, sensor-centric applications, to manage complex events and to implement task offloading for energy saving.

III. Communication Resources

Cloud offers an effective and cheap solution to connect, track, and manage anything from anywhere at any time using customized portals and built-in apps. Thanks to the availability of high speed networks, it enables the monitoring and control of remote things, their coordination, their communications, and the real-time access to the produced data.

IV. New Capabilities

IoT is characterized by a very high heterogeneity of devices, technologies, and protocols. Therefore, scalability, interoperability, reliability, efficiency, availability, and security can be very difficult to obtain. The integration with the Cloud solves most of these problems also providing additional features such as ease-of-access, ease-of-use and reduced deployment costs.

V. New Paradigms

The adoption of the CloudIoT paradigm enables new scenarios for smart services and applications based on the extension of Cloud through the things.

- SaaS (Sensing as a Service), providing ubiquitous access to sensor data;
- SAaaS (Sensing and Actuation as a Service) enabling automatic control logics implemented in the Cloud;
- SEaaS (Sensor Event as a Service) dispatching messaging services triggered by sensor events;
- SenaaS (Sensor as a Service) enabling ubiquitous management of remote sensors;
- DBaaS (DataBase as a Service) enabling ubiquitous database management;
- DaaS (Data as a Service) providing ubiquitous access to any kind of data;
- EaaS (Ethernet as a Service) providing ubiquitous layer-2 connectivity to remote devices.

3. Research Projects

Platforms

There are several open source and commercial platforms for Cloud and IoT integration. Most of them are aimed at solving one of the main issues in this field that is related to the heterogeneity of things and Clouds. These platforms try to bridge this gap implementing a middleware towards the things and another towards the Cloud, and they typically provide an API towards the applications.

IoTCloud is an open source project aimed at integrating the things (smart phones, tablets, robots, Web pages, etc.) with backend for managing sensors and their messages and for providing an API to applications interested in these data.

OpenIoT is another open source effort fostered by a research project. The project aims at providing a middleware to configure and deploy algorithms for collection and filtering messages by things, while at the same time generating and processing events for interested applications. Among the main focuses of OpenIoT are the mobility aspects of IoT for energy-efficient orchestration of data collecting and transmission to the Cloud.

IoT Toolkit (run by a Silicon Valley based organization called OSIOT) aims at developing a toolkit that allows gluing the several protocols available both for the things, for the Cloud, and for the applications.

GSN(Global Sensor Networks)

Global Sensor Networks is a middleware which supports the rapid and simple deployment of a wide range of sensor network technologies, facilitates the flexible integration and discovery of sensor networks and sensor data, enables fast deployment and addition of new platforms, provides distributed querying, filtering, and combination of sensor data, and supports the dynamic adaption of the system con- figuration during operation.

4. Implementation of the Research Projects

I. IoTCloud Overview

The IoTCloud implements the publish/subscribe design pattern as shown in figure 1 to orchestrate communication between sensors and client applications which form an inherently distributed system.

IoTCloud Server creates Publisher-Subscribe Channels (Represented as a JMS Topic)

Sensors acting as publishers create TopicPublishers to send messages to a Topic

Client applications acting as subscribers create TopicSubscribers to receive messages on a topic

Apache ActiveMQ is used as the default underlying MOM (Message Oriented Middleware) and any other JMS (Java Message System) style broker can be used as well.

Sensors are deployed by the Grid Builder into logical domains; the data streams from these sensors are published as topics in the sensor grid to which client applications may subscribe.

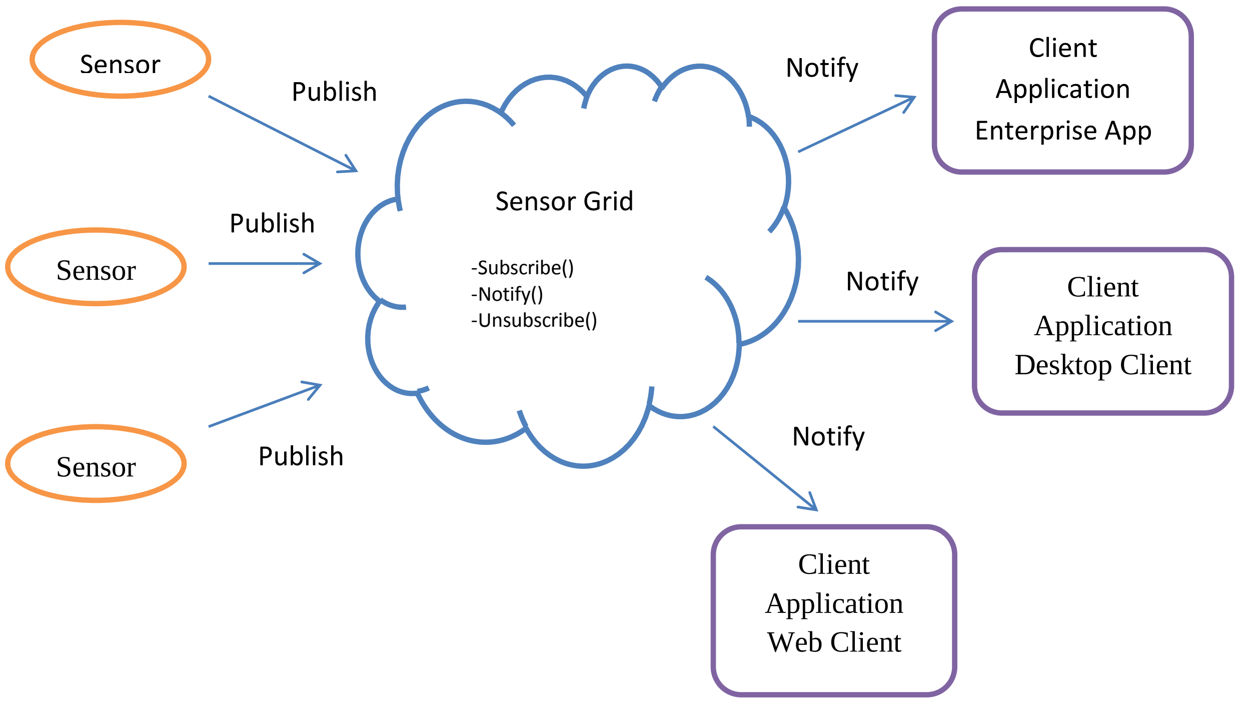

Figure 1 Schematic of the IoTCloud

Examples of physical devices already implemented include:

- Web Cameras
- Wii Remotes
- Lego MindStorm NXT Robots
- Bluetooth GPS Devices
- RFID Readers

However Sensors can be made from chat clients, Power Point presentations, web pages virtually anything which produces data in a time-dependent stream can be implemented as Sensor Grid.

IoTCloud Architecture

The main objective of the IoTCloud Project is to design and develop an enabling framework to support easy development, deployment, management, real-time visualization and presentation of collaborative sensor-centric applications. The IoTCloud Sensor Grid framework is based on an event-driven model that utilizes a pub/sub communication paradigm over a distributed message-based transport network.

The Sensor Grid is carefully designed to provide a seamless, user-friendly, scalable and fault-tolerant environment for the development of different applications which utilize information provided by the sensors. Application developers can obtain properties, characteristics and data from the sensor pool through the Sensor Grid API, while the technical difficulties of deploying sensors are abstracted away. At the same time, sensor developers can add new types of sensors and expose their services to application developers through Sensor Grid's Sensor Service Abstraction Layer (SSAL). ActiveMQ (NB) is the transport-level messaging layer for the Sensor Grid.

Sensor Grid Server (SG)

The SG mediates collaboration between sensors and applications. Primary function of SG is to manage and broker sensor message flows.

Sensor/SG flow - The SG keeps track of the status of all sensors when they are deployed or disconnected so that all applications using the sensors will be notified of changes. Sensor data normally does not pass through SG.

Application/SG flow - Applications communicate application API, which in turn communicates with SG internally. Applications can define their own filtering criteria, such as location, sensor id, and type to select which sensors they are interested in. These filters are sent to SG for discovering and linking appropriate sensors logically for that application and forwards messages among the relevant sensors and that application. SG must always check which sensors meet the selected filter criteria and update the list of relevant sensors accordingly. It then sends an update message to application if there are any changes of the relevant sensors.

Sensors' properties are defined by the sensors itself. Applications have to obtain this information through SG.

Application/Sensor flow – The SG provides each application with information of sensors they need according to the filtering criteria. The application then communicates with sensors through the application API for receiving data and sending control messages.

Application API

The Sensor Grid aims at supporting a large amount of applications for users and service providers of different industries (e.g. financial, military, logistics, aerospace etc.). The Sensor Grid provides a common interface which allows any kind of application to retrieve information from the sensor pool managed by SCGMMS. The API also provides filtering mechanism which provides application with sensors matching their querying criteria only.

Sensor

The definition of sensor is a time-dependent stream of information with a geo-spatial location. A sensor can be a hardware device (e.g. GPS, RFID reader), a composite device (e.g. Robot carrying light, sound and ultrasonic sensor), Web services (e.g. RSS, Web page) or task-oriented Computational Service (e.g. video processing service).

Sensor Client Program

A sensor needs a Sensor Client Program (SCP) to connect to the Sensor Grid. The SCP is the bridge for communication between actual sensors and SCGMMS. On the sensor side SCP communicates with the sensor through device-specific components such as device drivers. On the Sensor Grid side SCP communicates with the Sensor Grid through the Sensor Service Abstraction Layer.

II. OpenIoT Overview

The OpenIoT architecture is comprised by seven main elements that belong to three different logical planes. These planes are the Utility/Application Plane, the Virtualized Plane and the Physical Plane which include the following modules:

Utility/Application Plane

The Request Definition component enables on-the-fly specification of service requests to the OpenIoT platform by providing a Web 2.0 interface. It comprises a set of services for specifying and formulating such requests, while also submitting them to the Global Scheduler.

The Request Presentation component selects mashups from an appropriate library in order to facilitate a service presentation in a Web 2.0 interface. In order to visualize these services it communicates directly with the Service Delivery & Utility Manager so as to retrieve the relevant data.

The Configuration and Monitoring component enables the management and configuration of functionalities over the sensors and the (OpenIoT) services that are deployed within the OpenIoT platform. Moreover, it enables the user to monitor the health of the different deployed modules.

Virtualized Plane

The Scheduler processes all the requests for services from the Request Definition and ensures their proper access to the resources (e.g., data streams) that they require. This component undertakes the following tasks: it discovers the sensors and the associated data streams that can contribute to service setup; it manages a service and selects/enables the resources involved in service provision.

The Cloud Data Storage (Linked Stream Middleware Light, LSM-Light) enables the storage of data streams stemming from the sensor middleware thereby acting as a cloud database. The cloud infrastructure stores also the metadata required for the operation of the OpenIoT platform (functional data). The prototype implementation of the OpenIoT platform uses the LSM Middleware, which has been re-designed with push-pull data functionalities and cloud interfaces for enabling additional cloud-based streaming processing.

The Service Delivery & Utility Manager performs a dual role. On the one hand, it combines the data streams as indicated by service workflows within the OpenIoT system in order to deliver the requested service (with the help of the SPARQL query provided by the Scheduler) either to the Request presentation or a third-party application. To this end, this component makes use of the service description and resources identified and reserved by the Scheduler component. On the other hand, this component acts as a service metering facility, which keeps track of utility metrics for each individual service. This metering functionality will be accordingly used to drive functionalities such as accounting, billing, and utility-driven resource optimization. Such functionalities are essential in the scope of a utility (pay-as-you-go) computing paradigm, such as the one promoted by OpenIoT.

Physical Plane

The Sensor Middleware (Extended Global Sensor Network, X-GSN) collects, filters, combines, and semantically annotates data streams from virtual sensors or physical devices. It acts as a hub between the OpenIoT platform and the physical world. The Sensor Middleware is deployed on the basis of one or more distributed instances (nodes), which may belong to different administrative entities. The prototype implementation of the OpenIoT platform uses the GSN sensor middleware that has been extended and called X-GSN (Extended GSN).

III. IoT Toolkit Overview

The Internet of Things today consists of many different sensor networks and protocols, connected to dedicated cloud services, providing access through smartphone and browser apps. It is rare for these separate "silos" to cooperate or interact with each other.

The IoT Toolkit is an Open Source project to develop a set of tools for building multi-protocol Internet of Things Gateways and Service gateways that enable horizontal co-operation between multiple different protocols and cloud services. The project consists of the Smart Object API, gateway service, and related tools:

- Smart Object API gateway service reference implementation
- HTTP-to-CoAP Semantic mapping proxy
- Gateway-as-a-Service deployment
- Application framework, embedded software Agents
- Semantic discovery and linkage, Linked Data compatibility
- Tools for multiple sensor net clients
- Raspberry Pi and cloud micro-instance deployment images (Ubuntu)

IV. GSN(Global Sensors Network) Overview

GSN is a software middleware designed to facilitate the deployment and programming of sensor networks. GSN offers virtual sensors as a simple and powerful abstraction which enables the user to declaratively specify XML-based deployment descriptors in combination with the possibility to integrate sensor network data through plain SQL queries over local and remote sensor data sources.

GSN is a Java environment that runs on one or more computers composing the backbone of the acquisition network. A set of wrappers allow to feed live data into the system. Then, the data streams are processed according to XML specification files. The system is built upon a concept of sensors (real sensors or virtual sensors, that is a new data source created from live data) that are connected together in order to built the required processing path. For example, one can imagine an anemometer that would sent its data into GSN through a wrapper (various wrappers are already available and writing new ones is quick), then that data stream could be sent to an averaging mote, the output of this mote could then be split and sent for one part to a database for recording and to a web site for displaying the average measured wind in real time. All of this example could be done by editing only a few XML files in order to connect the various motes together.

Currently, each sensor network deployment has its own software system for gathering, analyzing (application dependent) and publishing the data coming from the sensor network. GSN is developed based on the observation that most of the logical requirements of the applications developed for sensor networks are similar. Having each sensor network deployment using its own custom software not only increases the costs and times of deployment and development (reinventing the wheel for each application), but also makes it difficult (almost impossible in some cases) to integrate several sensor networks together (especially when managed by different authorities).

GSN is designed to make the sensor network application development a pleasure. The applications based on GSN are hardware-independent making the sensor network changes invisible to the application, for instance you can change the underlying sensor network from the Mica2 nodes to the BTNodes (with compatible sensing boards) without ever touching a single line of code in the application.

Now you have all the common sensor network requirements in one package plus the support for dozens of well known sensing hardware. Download GSN today and let us know about your experience.

5. Choosing Appropriate Cloud

An important step is to decide which type of cloud our application needs, it can be either public, private or hybrid.
A “cloud” is essentially a pool of virtualized hardware that enables software applications to add and release resources as needed on the fly.

Public Cloud
An off-premises multi-tenant solution that enables a utility computing model. Example: a website hosted by a third party that can scale up or down to match fluctuating demand.

Private Cloud
A secure single-tenant solution hosted either on- or off-premises. Example: a customer database running on a set of virtualized servers behind a firewall.

Hybrid
Two or more clouds connected to support load balancing or cloud bursting between dedicated in-house resources and virtualized resources in the public cloud. Example:
A marketing portal hosted by a third party connected to an on-premises customer database. The website needs little security, while the database needs a dedicated environment. When demand

increases, the website can scale up by adding servers in the public cloud; meanwhile, the database remains secure in its private cloud.

Take a look at each workload to determine which kind of cloud it should be in. By asking the right questions around criteria such as availability, security and cost, the answers will push the workload to the public or private cloud.

6. References

http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=6984170&tag=1

http://iot-toolkit.com/

https://github.com/iotcloud/IoTCloud

https://github.com/LSIR/gsn/wiki