A Data-Centric Framework for Development and Deployment of Internet of Things Applications in Clouds

Farzad Khodadadi, Rodrigo N. Calheiros, Rajkumar Buyya
Cloud Computing and **D**istributed **S**ystems (CLOUDS) Laboratory
Department of Computing and Information Systems
The University of Melbourne, Australia
Email: {fkhodadadi@student., rnc@, rbuyya@}unimelb.edu.au

Abstract—**As technology develops, more human-made devices are able to make use of Internet to communicate with each other, thus enabling the Internet of Things (IoT) era to emerge. The amount of data IoT entities generate can overwhelm computer infrastructures not prepared for such data deluge and consequent need for more CPU power. Cloud computing offers a solution at infrastructure level that alleviates such a problem by offering highly scalable computing platforms that can be configured on demand to meet constant changes of applications requirements in a pay-per-use mode. Current approaches enabling IoT applications are domain-specific or focus only on interaction with sensors, thus they cannot be easily ported to other domains nor provide means of interactions with data sources other than sensors. To address this issue, in this paper we introduce a data-centric framework for development of IoT applications executed in clouds. The framework handles connection to data sources, data filtering, and utilization of cloud resources including provisioning, load balancing, and scheduling, enabling developers to focus on the application logic and therefore facilitating the development of IoT applications. We present a prototype application executing on AWS, built on top of the Aneka Cloud Application Platform and present experiments demonstrating the use of our framework for building an application capable of processing input from different sources and analysing them using easy-to-use APIs provided by Aneka.**

I. Introduction

We are witnessing the rise of new data intensive technologies, such as Internet of Things (IoT), that give raise to the "Big Data" problem. This is because, as technology develops, human-made devices such as sensors, surveillance cameras, and scientific instruments are continually generating huge amounts of data, and making sense of all this data is becoming harder. To make things worse, the nature of this machine-generated data is changing substantially, and nowadays most of the generated data is unstructured or semi-structured, what causes previous techniques for storing and processing the data obsolete.

To counter this, technologies such as NOSQL databases and the MapReduce programming model started being heavily utilized in what is called "Big Data analytics". Although these technologies provide the tools necessary for processing huge amounts of data, the amount of data generate can still overwhelm storage systems and applications trying to make use of them. Furthermore, in some situations like surveillance and disaster detection and prevention, the applications need to operate under strict deadlines so the output information can be used by response teams to optimize their work, what in turn imposes a massive burden over the computational infrastructures that need to cope with this data and processing power requirements.

Cloud computing [1], by offering distributed computational resources as utilities with a pay-as-you-go model and with Quality of Service (QoS) assurance via Service Level Agreements (SLAs), offers a solution at the infrastructure level that can support processing large volumes of data by means of high scalability and elasticity of computing resources. It enables solutions to be built on top of highly scalable computing platforms that can be configured on demand to meet constant changes of applications requirements in a pay-per-use mode, reducing the investment necessary to build applications.

To address this issue, in this paper we introduce a cloud-based framework for development and deployment of IoT applications. The framework enables IoT application developers to focus on the application logic, while the framework handles not only connection to data sources and query and filtering, but also utilization of cloud resources including provisioning, load balancing, and scheduling. The proposed framework facilitates data acquisition and sharing, while allowing sophisticated analytic methods to be applied on the data. Since there is plenty of existing cloud solutions that can handle the latter aspects, we developed a prototype that focuses on the former aspects. To this end, we built our prototype on top of the Aneka Cloud Application Platform [2], which manages all the aspects related to interaction with the cloud platform. This facilitates the development of IoT applications and reduces the development time of such applications, contributing for the establishment of the streaming computing paradigm and cloud computing as key enablers of solutions for Big Data analytics and other complex problems. Furthermore, our integrated and modular solution can be easily extended to support other data sources and different analytic algorithms.

II. Related Work

Frameworks for facilitating interaction between applications and cloud computing infrastructures have been proposed for many domain-specific scenarios, such as e-Learning [3], Massively Multiplayer Online Games [4], emergency call centers [5], agriculture and forestry [6], urban traffic control systems and vehicular clouds [7], [8], and tailing disposal from mining [9]. However, these existing frameworks lack the seamless access to multiple data sources that is addressed by our approach in order to meet the demand of emerging complex applications such as disaster management and prediction, social media, and environment monitoring.

Substantial effort towards IoT application development in clouds has been carried out so far. Alam et al. [10] and Zhu et al. [11] concurrently and independently proposed frameworks focusing in aggregating sensors in the context of cloud-enabled IoT. Li et al. [12] proposed a Platform as a Service (PaaS) solution for deployment of IoT applications. The solution is multi-tenant and focuses on the support for such tenants, as well as managing and metering the utilization of the platform by different tenants.

Nastic et al. [13] proposed PatRICIA, a framework for deployment of IoT applications that provides its own programming model for development of IoT applications to be deployed in the cloud. It differs from our work mainly on the abstractions offered for application developers. Whereas we focus in conventional object-oriented programming constructs, PatRICIA proposes new abstractions that belong to the proposed concept of *Intent-based programming*.

Parwekar [14] discussed the importance of identity detection devices in IoT and proposed a service layer to demonstrate how a sample tag-based acquisition service can be defined in the cloud. A simple architecture for integrating machine to machine (M2M) platform, network, and data layers has also been proposed, but the architecture fails to comprehensively address details and requirements of implementing the proposed integration method.

All these works focus mainly in supporting processing of data derived from sensors, ignoring the issue of supporting the execution of general applications accessing other streaming data sources such as social networks, microblogs, databases, and websites, which are supported by our proposed approach.

III. Application Scenario and System Requirements

The target application scenario of our framework is IoT applications hosted in cloud environments. From the perspective of a user of our framework willing to deploy its application, the framework can be seen as a PaaS resource, because it enables users to focus on their applications (in this case, IoT applications) while ignoring details about the underlying infrastructure actually supporting the application.

The framework provides interfaces to enable the easy extraction of data from data sources to be delivered for applications. Examples of possible data sources for the applications include sensors and sensor networks; surveillance cameras; application logs; social media; microblogs; cloud storage; and NOSQL and relational databases. It is the users' responsibility to write the business logic of the application, i.e., the code that processes the data from data sources and produces the expected output to users. The framework does not enforce any specific programming paradigm (MapReduce, workflow, BoT, etc.); therefore, users are free to adopt the paradigm that is more suitable for their applications.

In order to support the above scenario, the following requirements were identified:

- Programming paradigm-independent API;
- Support for multiple types of data sources, potentially supporting data of different nature (structured, unstructured, streaming, video, and so on);
- Support for many users, with performance isolation between them, thus ensuring framework scalability;
- Security and confidentiality, as data accessed, or operation performed by users may be sensitive;
- Support for many types of infrastructure resources, including public clouds, private clouds, LANs, supercomputers, and grids;
- Fault tolerance via replication of components;
- QoS assurance via intense monitoring of framework components and applications;
- Extensibility, so new types of data sources and programming paradigms, yet to be developed, could be integrated in the future;
- Complete isolation between user executing the application and the underlying cloud resources processing the user application;
- Automatic and transparent (to users) selection of cloud providers and instance types for automatic resource scaling; and
- High utilization and load balance of cloud resources in order to optimize the cost for use of cloud resources.

Based on the above requirements, we developed a general architecture for development and deployment of IoT applications in clouds, which is detailed in the next section.

IV. Framework Components

Figure 1 depicts the overall architecture of the framework. It is composed of three main components: *Application Manager*, *Cloud Manager*, and *Data Source Manager*. Each of these elements can run in a single cloud resource or they can be decoupled and executed in different cloud resources. In particular, the Data Source Managers should preferably run closely to the data sources, in order to reduce network delays. Similarly, Data Source Filters should run preferentially in the same cloud infrastructure than the corresponding data sources in order to reduce latency and data transfer costs. Other elements can be deployed based on location of users and availability of cloud resources.

1) Application Composer: The Application Composer is the entry point for users submitting applications for execution in the framework. Through an API, it enables users to select one particular programming model to be used for the

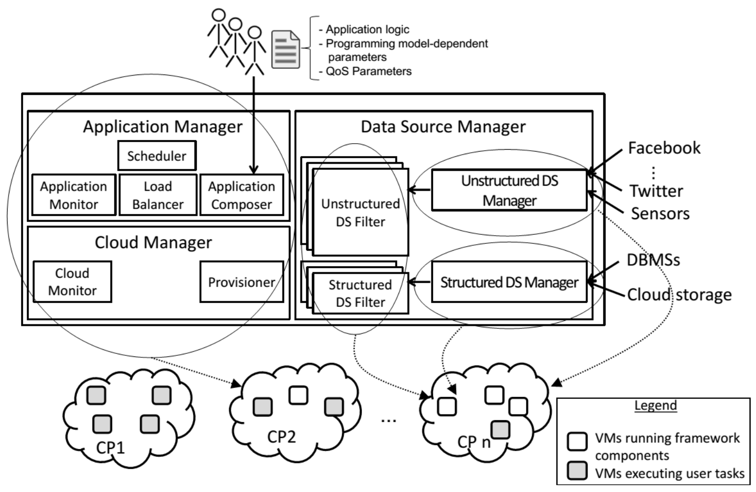

Fig. 1. Framework architecture.

application logic and to tailor the application for the selected programming model. The APIs also enable users to describe how the data being received from multiple data sources will be used by the application.

The framework is more suitable for execution of loosely-coupled applications, which include widely adopted application models such as Parameter Sweep, Bag-of-Tasks, Workflows, and MapReduce. Nevertheless, transactional applications composed of a high number of requests that require some transactional processing to be performed, such as multi-tier web applications, are also supported by the framework.

Optionally, the Application Composer also allows users to give hints on the granularity of the application. For example, in case of BoT applications, a hint on the maximum number of tasks to be created can be supplied. Although this requires a lower level knowledge about the application and the underlying system, it also enables a better performance to be obtained for application execution. This is because knowledge on parameters such as the I/O-computation ratio may cause the application to reach a performance optimal for a given number of tasks and performance degradation after a certain number of tasks. If this is known beforehand, application execution can be optimized.

The Application Monitor, after a certain amount of time monitoring the application execution, could eventually detect that the number of tasks is not optimal because of rates of I/O and CPU processing. Nevertheless, a previous knowledge on the issue enables the application to run at an optimum rate since its start, whereas reactive correction requires some time until it can be effectively applied.

2) Scheduler: The Scheduler module is responsible for planning the execution of non-transactional tasks. It determines the placement of tasks that compose one application as well as the execution order of the tasks. The latter is performed not only in the context of a single application, but also on the context of previous applications whose tasks are already scheduled but not yet executed. Another responsibility of this module regards the admission control of new requests for execution of non-transactional applications, depending on the deadline set for the application, availability of resources for execution, and deadlines of running applications, the module can opt for rejecting the new request.

3) Load Balancer: The Load Balancer component is responsible for keeping the load of machines processing transactional applications balanced through an appropriate routing of new requests to the most suitable machine. This component can be seen as the counterpart of the Scheduler acting for web requests arriving to the system. Similar to the Scheduler, the Load Balancer also rejects requests and issues a warning to the Provisioner when QoS for new requests cannot be guaranteed.

4) Application Monitor: The Application Monitor is responsible for keeping track of resource utilization and execution time of the tasks and requests submitted to the system. Relevant performance data is delivered to appropriate modules in order to enable scaling actions on the system. For example, task execution times are delivered to the Scheduler so, based on the tasks deadlines, actions can be taken. For the Load Balancer, aggregated metrics, such as response time of the 90th percentile, are reported, as this are typically the metrics of interest for performance evaluation of transactional requests.

5) Provisioner: The Provisioner is responsible for transforming a request for resources received from the Application Manager into actual allocation of cloud resources. This is performed with consideration of multiple sources of information, as follows: (i) warnings from Load Balancer and/or the Scheduler that new requests are being rejected because of lack of resources; (ii) response time of requests, execution time of tasks, and deadline violations, obtained from the Application Monitor; (iii) performance metrics and utilization rate of cloud resources obtained from the Cloud Monitor. Based on the above information, decision is made on whether to maintain the current amount of resources, scaling down, or scaling up cloud resources.

Activities performed by the Provisioner include selection of cloud providers to provide the required resources; selection of specific resources from the selected provider; allocation of selected resources on the selected provider; and resource decommission.

To avoid underperformance of accepted application requests, a timeout mechanism has to be in place, so in case no response is obtained from the provider after a given time, alternative solutions can be sought by the module. Similarly, if the provider is unable to provide the required resources with the required performance, alternatives are explored.

6) Cloud Monitor: The Cloud Monitor is responsible for keeping track of the status and performance of allocated cloud resources, as well as observed utilization of such resources. This information is used to update the Provisioner about performance so that it can decide to allocate further resources or to migrate resources to another provider if performance is not satisfactory.

A. Data Source Manager

The Data Source Manager is the component that acts as the interface between the framework and the multiple sources of data for applications. It contains components that can interact with different sources, and can "understand" the data that is received from such sources. It also contains components that can perform queries and filter data from data sources so only data of interest for each user is delivered to the user's application.

1) Structured Data Source Manager: The Structured Data Source Manager is the interface between the framework and relational databases (and other sources of structured data, such as some types of cloud storage). It can handle connection/disconnection with the data source (in case of data streaming sources) and submit queries and retrieve results for related data sources.

2) Unstructured Data Source Manager: The Unstructured Data Source Manager links the framework with external entities that generate unstructured data, such as microblogs data or time-series data aggregated from a thermal sensor. For each different type of sensors and instruments, this components is able to connect and disconnect to it, and interpret the incoming data so framework-level filters are applied in order to eliminate meaningless information before user-specific filters (detailed later on this section) are applied.

3) Structured/Unstructured Data Source Filter: The Structured Data Source Filter accepts queries applied by users and applies them on the data being received from the Structured Data Source Manager, while the Unstructured Data Source Filter applies user-defined filtering rules to the unstructured data sources that the user is registered to. Given that one user can be registered to receive data from multiple structured data sources, this component also consolidates the information before applying the query. The resulting, "filtered" data is then delivered to the user application via specific API calls. In order to achieve an acceptable performance for this operation, queries run on separate resources than the rest of the framework, so more resources for query processing can be added if more user-specific queries are necessary (i.e., if more users submitted applications with customized queries).

V. Prototype Design and Development

In the previous sections, we discussed requirements of a framework for development and deployment of IoT applications in clouds. The discussion did not focus on any particular technologies that could be employed for achieving the framework goals. Thus, as previously stated, the framework could be built from scratch, by reusing one or more existing technologies, where each technology performs some activities, or by reusing technologies to some goals and writing functionalities for other tasks.

Since many of the framework requirements are already met by the Aneka Cloud Application Platform [2], we designed the framework to take advantage of the existing modules and functionalities of Aneka, and focused on the new features and modules required for enabling execution of IoT applications in clouds. Aneka, as a PaaS middleware, enables users to write applications in many programming models, some built-in in the platform, and others defined by the users. The underlying infrastructure is composed of any computational resource that supports the .NET framework (or its Open Source implementation Mono).

A. Aneka Integration

Following the architecture introduced on Figure 1, the Application Manager and Cloud Manager are replaced by native features of Aneka. Thus, features that needed to be developed and/or integrated concerned application development and data source management. A high-level view of the architecture of the prototype is depicted in Figure 2.

Regarding application development, Aneka also offers options for developers to integrate their own application models. Prior to our contribution, three major types of applications were supported by Aneka, namely MapReduce, Task, and Thread. Thus, to enable IoT application development, we implemented a new programming model for Aneka, called IoT model. This model is built on top of the existing Aneka Task model, which enables users to execute arbitrary applications on the Aneka platform. Hence, we implemented low level management of data and data source management, and provided an API that application developers use to access such features.

We also implemented the integration of data sources with the entry-point for the user's application logic. Internally, the supplied application logic, along with the logic for data source management, is encapsulated as a task that is executed by the platform, thus it leverages the capabilities of Aneka regarding load balancing, scheduling algorithms, and dynamic provisioning.

Regarding data source management, the requirements of the proposed framework demand the use of a lightweight protocol as a means of communication between different available data sources and the Aneka platform. MQ Telemetry Transport (MQTT)[1] is a lightweight protocol originally developed by

[1]http://mqtt.org/

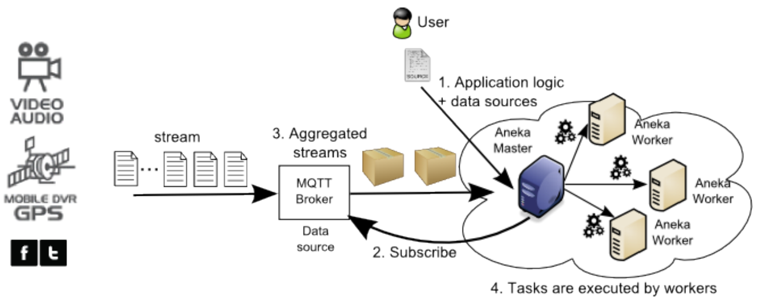

Fig. 2. High-level architecture of the prototype implemented on top of Aneka.

IBM for unreliable networks with low-bandwidth and devices with memory constraints. Since this protocol is based on the publish/subscribe model and is also vastly used in the context of IoT, we adopted this protocol in our framework.

In our proposed prototype, every data source is created by extending a base class that handles communication between the data source and Aneka using the MQTT protocol. Since this protocol uses a publish/subscribe model, we defined a set of suitable event-oriented API functions to simplify application development using Aneka. Any defined data source in Aneka connects to a MQTT broker that is specified using an IP address. The broker is responsible for registering data sources using the MQTT protocol and assigning unique identifiers for them. The broker also manages topics and requests corresponding to the MQTT standard. A topic is a hierarchical data structure defined in MQTT that allows the publishers to specify exactly in which topic their published data should reside and consequently, any subscriber who has subscribed to that specific topic, or pool of topics, will receive the message when it is received by the broker entity.

B. API Features

In order to write an IoT application, the only action required from developers is the creation of a class implementing the required logic. The class implementing the logic extends `IoTEntity`, so the framework is aware that this class implements the logic to be deployed and scaled in the cloud. For example, an application that receives streaming data from Twitter can be written as:

```
01 class TwitterStream extends IoTEntity
02 {

03  public override void Start(string[] topics,
                                   IoTQoS QoS)
04  {
05       base.ConnectAndRegister(topics, QoS);
06  }
07 }
```

The particular task logic is written in the Start() method (Line 3). Initially, the developer connects to the data sources to be used (Line 5). The QoS parameter is used by the MQTT protocol for prioritizing messages received by broker. This allows developers to establish the priority on which data from different sources should be given when the data is sent to the application. It also allows developers to prioritize applications registering with the broker.

The framework is responsible for determining how many data sources are available and, based on the specified QoS, decides the number of tasks to be created task. Data sources are balanced between the task instances. The continuing monitoring ensures that if the number of tasks is not enough to meet QoS, new instances are executed on new VMs and the data sources are redistributed among the new tasks. Similarly, if there are an excessive number of tasks, some of the VMs running the task are terminated and the data sources are assigned to the remaining tasks.

VI. Performance Evaluation

In order to evaluate the performance of the proposed framework, we performed experiments deploying the prototype described in the previous section on a public cloud provider (Amazon AWS on Sydney, Australia). The testbed is composed of five virtual machines of type `t2.small` (1 vCPU and 2 GB of RAM on an Intel Xeon 2.5 GHz CPU) running Windows Server 2012. One of the VMs served as Aneka master, coordinating the execution of the application and scheduling tasks, and the other four VMs were configured to play the role of Aneka workers. As a broker implementing the MQTT protocol, we used Mosquitto[2], an open source tool that contains a Python API that we used to integrate the message broker with our test application.

Our application performs sentiment analysis on streaming data received from Twitter using a Raspberry PI[3] tweet harvester bot. We used Raspberry PI to demonstrate how different data formats originating from sensor devices to social media streams can be easily mapped, converted and processed using our framework. In this scenario, Aneka serves as a message subscriber with the Mosquitto broker and, after receiving tweet events, creates a task and the sends it to the master node for scheduling and execution on Aneka workers. The application logic required for running the experiments on this section was written with 67 lines of code. 16 of these lines concerned connecting to the data source, querying the data, and sending it to the main application. The sentiment analysis application itself was written with 51 lines of code.

[2]http://mosquitto.org/
[3]http://www.raspberrypi.org

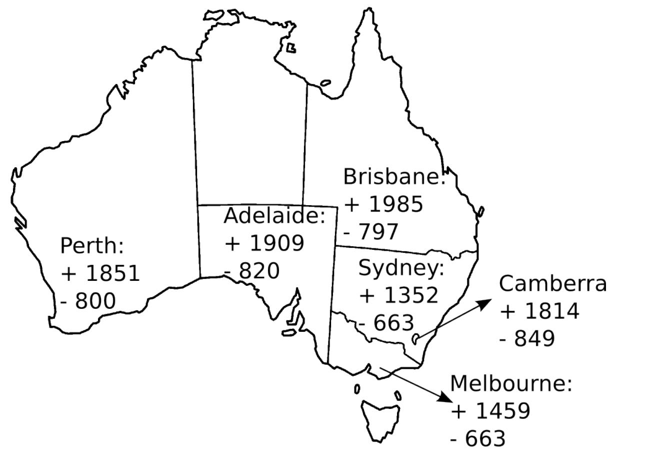

Fig. 3. Results of sentiment analysis of tweets from the biggest capital cities of Australia.

To demonstrate the application in action, we executed the application to compute the sentiment analysis of the six biggest capital cities of Australia. We used 20,000 tweets from each city obtained via the Twitter API on August 15, 2014. The score of each valid word in a tweet is fetch from the AFINN[4] word list. The total weight of each tweet is the sum of weights for each word in that tweet. If a tweet's total weight is bigger than 1, it is considered a positive tweet and if the total weight is smaller than -1, it is considered a negative tweet. Figure 3 shows the number of positive and negative tweets obtained for the studied capital cities (presented in their respective states on the map). It shows that, when neutral ones are disregarded, Brisbane, Adelaide, and Perth are the cities with the biggest number of positive tweets, while Australia's biggest cities—Melbourne and Sydney—have the smallest balances in favor of positive tweets.

One main feature of our framework is the ability to quickly and easily change the programming paradigm used for processing the data. To demonstrate this capability, instead of streaming harvested tweets, we use a micro-batch model to process our data. To properly implement this, arriving tweets are received and buffered by the broker and a task is created when the number of harvested tweets reaches a threshold, in our case 100. Considering the facts that tweets are very small in size and performing sentiment analysis on them requires low computational power, results show that overall execution time when using micro-batch approach is lower than corresponding time when using streaming approach. The difference is relative to number of workers and their computational power in terms of CPU frequency and amount of RAM.

VII. Conclusions and Future Directions

Current approaches integrating IoT applications and cloud computing are either manual or semi-automatic, and require software developers to understand the underlying computing and data infrastructures. This consumes valuable time and energy from developers, as it derails them from the focus on the application logic. To alleviate such a burden from developers, we proposed a framework for development of IoT applications executed in clouds. The framework handles all the aspects of interaction between the application, cloud, and data sources. We detailed the framework requirements, its components, and API. We presented a prototype executing on AWS and built on top of the Aneka Cloud Application Platform and presented experiments demonstrating the use of our proposed framework.

As future work, we plan to develop efficient and scalable algorithms for each of the activities performed by the framework. This includes data filtering, provisioning (including algorithms for resource selection), and scheduling. Algorithms supporting interactions with Inter-clouds (what may include resource negotiation) will also be researched, along with energy-efficient algorithms for interaction with private clouds.

[4]http://www2.compute.dtu.dk/~faan/data/AFINN.zip

References

[1] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, "Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility," *Future Generation Computer Systems*, vol. 25, no. 6, pp. 599–616, Jun. 2009.

[2] R. N. Calheiros, C. Vecchiola, D. Karunamoorthy, and R. Buyya, "The aneka platform and qos-driven resource provisioning for elastic applications on hybrid clouds," *Future Generation Computer Systems*, vol. 28, no. 6, pp. 861–870, June 2012.

[3] B. Dong, Q. Zheng, M. Qiao, J. Shu, and J. Yang, "BlueSky cloud framework: An e-learning framework embracing cloud computing," in *Proceedings of the 1st International Conference on Cloud Computing (CloudCom)*, 2009.

[4] A. Iosup, "CAMEO: Continuous analytics for massively multiplayer online games on cloud resources," in *Proceedings of the 2nd International Workshop on Real Time Online Interactive Applications on the Grid (ROIA)*, 2009.

[5] S. Venugopal, H. Li, and P. Ray, "Auto-scaling emergency call centres using cloud resources to handle disasters," in *Proceedings of the 19th International Workshop on Quality of Service (IWQoS)*, 2011.

[6] Y. Bo and H. Wang, "The application of cloud computing and the internet of things in agriculture and forestry," in *Proceedings of the 2011 International Joint Conference on Service Sciences (IJCSS)*, 2011.

[7] W. He, G. Yan, and L. Xu, "Developing vehicular data cloud services in the iot environment," 2014.

[8] P. Jaworski, T. Edwards, J. Moore, and K. Burnham, "Cloud computing concept for intelligent transportation systems," in *Intelligent Transportation Systems (ITSC), 2011 14th International IEEE Conference on.* IEEE, 2011, pp. 391–936.

[9] E. Sun, X. Zhang, and Z. Li, "The internet of things (IOT) and cloud computing (CC) based tailings dam monitoring and pre-alarm system in mines," *Safety Science*, vol. 50, no. 4, pp. 811–815, Apr. 2012.

[10] S. Alam, M. M. R. Chowdhury, and J. Noll, "SenaaS: An event-driven sensor virtualization approach for internet of things cloud," in *Proceedings of the 2010 IEEE International Conference on Networked Embedded Systems for Enterprise Applications (NESEA)*, 2010.

[11] Q. Zhu, R. Wang, Q. Chen, Y. Liu, and W. Qin, "IOT Gateway: Bridging wireless sensor networks into internet of things," in *Proceedings of the 8th International Conference on Embedded and Ubiquitous Computing (EUC)*, 2010.

[12] F. Li, M. Vogler, M. Claessens, and S. Dustdar, "Efficient and scalable IoT service delivery on cloud," in *Proceedings of the 6th International Conference on Cloud Computing (CLOUD)*, 2013.

[13] S. Nastic, S. Sehic, M. Vogler, H.-L. Truong, and S. Dustdar, "PatRICIA — a novel programming model for IoT applications on cloud platforms," in *Proceedings of the 6th nternational Conference on Service-Oriented Computing and Applications (SOCA)*, 2013.

[14] P. Parwekar, "From internet of things towards cloud of things," in *Computer and Communication Technology (ICCCT), 2011 2nd International Conference on.* IEEE, 2011, pp. 329–333.