Building a Big Data Platform for Smart Cities: Experience and Lessons from Santander

Bin Cheng, Salvatore Longo, Flavio Cirillo, Martin Bauer, Ernoe Kovacs
NEC Laboratories Europe, Heidelberg, Germany

***Abstract*—The Internet of Things (IoT) is now shaping our cities to make them more connected, convenient, and intelligent. However, this change will highly rely on extracted values and insights from the big data generated by our cities via sensors, devices, and human activities. Many existing studies and projects have been done to make our cities smart, focusing more on how to deploy various sensors and devices and then collect data from them. However, this is just the first step towards smart cities and next step will be to make good use of the collected data and enable context-awareness and intelligence into all kinds of applications and services via a flexible big data platform. In this paper, we introduce the system architecture and the major design issues of a live City Data and Analytics Platform, namely *CiDAP*. More importantly, we share our experience and lessons learned from building this practical system for a large scale running smart city testbed, SmartSantander. Our work provides a valuable example to future Smart City platform designers so that they can foresee some practice issues and refer to our solution when building their own smart city data platforms.**

I. Introduction

Nowadays more than half of the world's population lives in cities and this proportion is still increasing day by day. As urban environments are becoming more densely populated and more complex, modern cities especially big cities are facing challenges in various areas: congested transport infrastructure, energy efficiency, water and waste management, air pollution and climate change. On the other hand, we see cities are becoming much more digital and connected than ever before. Many cities, like Santander in Spain and Songdo in South Korea, have deployed lots of sensors to monitor the real-time situations of their cities, such as status of bridges, road traffic, parking places, noise, light, air quality etc. Also, people in the cities are reporting contextual information about what they see and what they feel via social networks or participatory sensing. All of these data come together and become the pulse of our cities, providing us new opportunities to solve the challenges facing cities worldwide.

There have been lots of efforts made in both industry and academia to turn our cities into smart cities, but many of them focus on connecting cities by installing a large number of sensors and devices. For example, in the European project of SmartSantander [1], more than 15,000 sensors have been installed around an area of approximately 13.4 square miles in the Spanish city of Santander. This project has made big effort to deploy a citywide testbed, with regard to wireless network topology, reliable data transmission, battery lifetime and programmability of deployed sensor nodes. However, making cities connected for collecting data is just the first step towards smart cities. The second step is to turn big data into valuable and actionable insights and open them for various applications to use in different business domains.

Building up a flexible and efficient big data analytics platform between data sources and applications is the key to achieve this second step. A few studies have been carried out to explore big data platforms for smart cities, but most of them just discuss functional requirements and architecture designs at the abstract level. When we start to build a real big data platform for smart cities, there are few practical running systems that we can refer to. Meanwhile, since so many different tools and platforms with similar functionalities exist in the big data community, we are often overwhelmed and confused by their features and capabilities. Therefore, there is still a gap between what a big data platform for smart cities looks like at the high level and how it should be properly realized. To fill this gap, this paper presents a concrete and valueable example by introducing our city data and analytics platform named *CiDAP*.

The CiDAP platform has been deployed and integrated with a running IoT experimental testbed *SmartSantander*, one of the largest smart city testbed in the world. Importantly, in this paper we share our experience and lessons that we have learn from building such a live big data platform. Our paper can shed some light on the practical issues that the future designers must have to consider when building their own big data platforms towards smart cities. The main contributions of this paper are summarized as follows.

First, it presents the system architecture overview and major design issues of a live smart city big data platform called *CiDAP*, which is able to deal with historical data, near-time data, and also real-time data. The CiDAP platform is also architectually scalable, flexible, and extendable to be integrated with different scales of smart city infrastructures.

Second, it reports our experience and lessons learned from a 3-month deployment of the CiDAP platform that has been integrated with a running smart city testbed *SmartSantander*, one of the largest smart city testbeds. Our experience shed the light on what we did well in *CiDAP* and what needs to be further considered in the future. For example, we identify that: a naming mechanism must be seriously taken into account when dealing with multiple data sources; it is better to keep the data paths for both real time data and near time data since we have to balance real time requirements of some applications and power consumption of deployed sensor nodes.

Third, it presents some preliminary measurement results about the performance of the CiDAP platform via a microbenchmark and also introduces some analytics results based on the sensor data collected from the SmartSantander testbed. For example, our results indicate that detecting anomalies of

978-1-4673-7278-7/15 $31.00 © 2015 European Union
DOI 10.1109/BigDataCongress.2015.91

sensor nodes and sensor data must be considered as a building block of smart city data platforms.

II. Background and Motivation

In this section we briefly introduce the SmartSantander testbed and the motivation of our work. More detailed information on SmartSantander can be referred to the relevant papers in [1] [2] [3] [4].

A. SmartSantander

SmartSantander is one of the largest smart city experimental testbeds in the world. This testbed has been deployed at the city of Santander, located in the north of Spain. Around 180,000 people live in the city with its beaches, leisure facilities and history. Using the grant from the European Union, a consortium of 25 partners has been able to turn Santander into a living experimental laboratory, which is now used as an experimental test facility for the research and experimentation of architectures, key enabling technologies, services and applications for the Internet of Things in the context of smart cities.

Within the SmartSantander testbed, more than 15,000 sensors (attached with around 1,200 sensor nodes) have been installed around an area of approximately 13.4 square miles in the city. A large proportion of the sensor nodes are hidden inside white boxes and attached to street infrastructure such as street lamps, buildings and utility poles, while others are buried into the pavement, e.g., parking sensors. Not all of the sensors are static; some are placed on the city's public transport network, including buses, taxis and police cars. The deployed sensors provide real-time information regarding different environmental parameters (light, temperature, noise, CO_2), as well as other parameters like occupancy of parking slots in some downtown areas.

The SmartSantander architecture has been conceived in a 3-tiered architecture: Sensor nodes, Repeaters, and Gateways. Both sensor nodes and repeaters are responsible for sensing determined parameters, but the sensor nodes are not able to forward information while the repeaters can forward the information they receive to the next hop. Communication among sensor nodes, repeaters and gateways carries out through IEEE 802.15.4 interface, while gateways use Wi-Fi, GPRS/UMTS or Ethernet interfaces to connect with the SmartSantander backbone.

B. Remaining Challenges

The focus of the SmartSantander project was to connect the Santander city via widely deployed sensors. The SmartSantander testbed provides us the underlying infrastructure to collect data but it does not provide any capability of storing, processing, and analyzing generated data. Since the testbed itself does not keep historical data, applications have to manage the collected data and perform big data processing and analytics with their own resources. Therefore, to build a big data platform on top of the SmartSantander testbed for all smart city applications to share and reuse processing and analytics of both real-time and historical data still remains a big challenge.

To build such a big data platform, the following issues must be considered. First, *how to efficiently store various unstructured or semi-structured data*? Second, as data are continuously generated by smart cities over time, *how to process both historical data and real-time data and keep aggregated results updated in a scalable and incremental manner*? Also the designed platform must be flexible to extend its data processing capability. Third, *how to share processing and analytics of big data across applications via flexible APIs*? In the following section, we will introduce how we tackle these issues within a practical big data platform designed for the SmartSantander testbed.

C. Related Work

Lots of efforts from both industry and academia have been made towards smart cities, but most of them focus on infrastructure construction, data collection, testbed deployment, or specific services/applications development. In this paper we focused more on smart city data platforms that can deal with both historical data and real time data. We also consider its interfaces with regard to the requirements from external applications.

There are a few studies already exploring big data platforms for smart cities, but they only present some high level platform architecture designs [5] [6]. Examples include SCOPE [7] which is a Smart-city Cloud-based Open Platform and Ecosystem from Boston University, and FIWARE [8] which provides a framework for the development of intelligent applications in the Future Internet. Several industrial companies advertise their smart citiy data platforms, like IBM [9], AGT [10], Microsoft [11], but no details are open to the future designers. Meanwhile, there are some ongoing projects trying to explore the opportunities and challenges of big data for smart cities at the platform level, such as CityPulse [12], an ongoing European project exploring real-time stream processing and large scale data analytics for smart city applications. In addition, Singapore is building a new smart city platform called Smart Nation [13] to enable greater pervasive connectivity, better situational awareness through data collection, and efficient sharing of collected sensor data. The Infocomm Development Authority of Singapore (IDA) said some 1,000 sensors will be deployed at six high-traffic areas by the end of 2015 in the first phase of Singapore's Smart Nation Platform (SNP). Our experience and lessons from the CiDAP platfrom could be a timely and valuable input to the future smart city designs like Smart Nation.

III. CiDAP: Big Data Platform for SmartSantander

To solve the challenges mentioned above, we build up a big data platform named *CiDAP* on top of the SmartSantander testbed. The CiDAP platform is able to process both historical and real-time data, meanwhile exposing results to various applications. This section describes its overall architecture and major design issues, including how various data are collected from different data sources, how big data are saved and processed, and how external data analytics and applications can be supported using its interfaces.

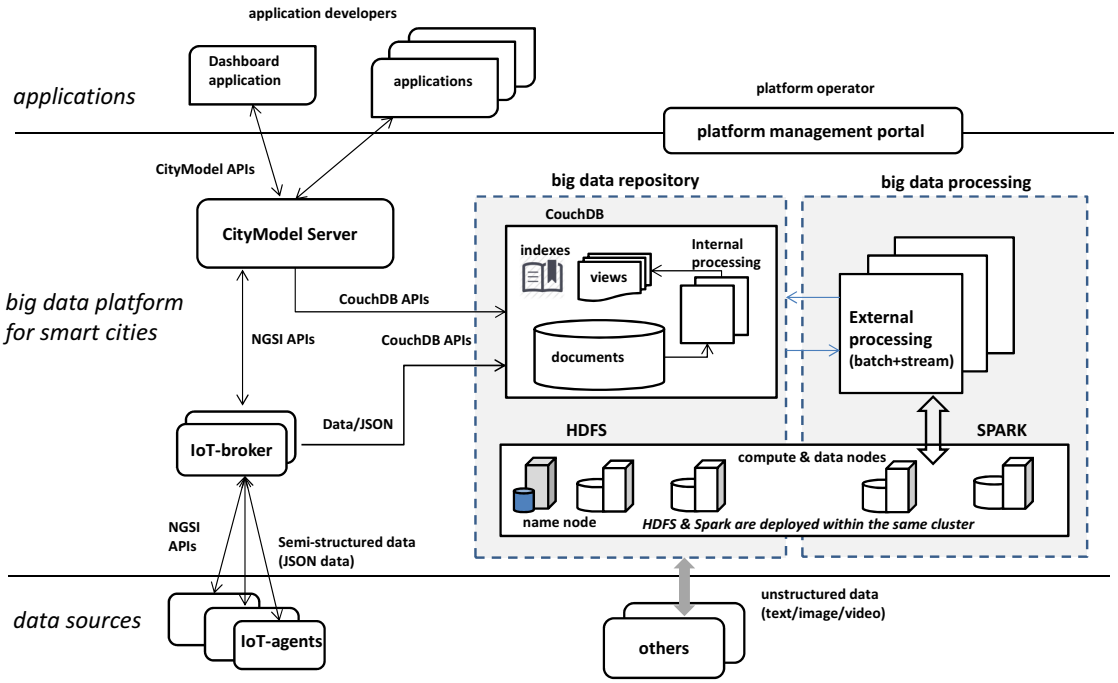

Fig. 1: System architecture of the CiDAP platform

A. Architecture Overview

Figure 1 illustrates the overall architecture of the CiDAP platform, which is the middle layer between data sources and smart city applications, including four main modules: *IoT-broker* and *IoT-agents* for data collection, *big data repository* for data storage, *big data processing* for intensive data processing and analytics, and *CityModel server* for communicating with external applications.

Here is how the CiDAP platform works at the high level. First, data with different formats are collected via the IoT-broker [14] from multiple data sources, then forwarded to the big data repository to be stored. The collected data are then processed and aggregated by a set of pre-defined or newly launched processing tasks. The simple processing tasks can be performed by the big data repository internally, such as transforming data into new formats or creating new structured views/tables to index data. Any complex or intensive processing tasks, such as aggregating or mining data via advanced data analytics, must be separated from the big data repository so they can be efficiently and externally executed over the big data processing module, which provides more flexible and scalable computation resource based on a Spark [15] cluster with a large number of compute nodes. Since the big data repository can already handle lightweight processing tasks in a scalable and incremental manner, the big data processing module can be optional if we do not need intensive data processing or analytics. By fetching generated results from the big data repository or forwarding messages directly from data sources in the smart city testbed, a CityModel server is designed to serve queries and subscriptions from external applications based on pre-defined CityModel APIs. Meanwhile, a web-based platform management portal is provided to the platform operator to monitor the status of the entire big data platform. More design details are discussed below.

B. Data Collection

In a smart city data can be collected from various sources, for example, IoT testbeds, social networks, video surveillance systems, or other shared open data like city maps, bus time schedule information, location of restaurants etc. The biggest challenge for data collection is to deal with the format diversity of different data sources. In order to provide a unified interface for collecting data, we design two components namely IoT-broker and IoT-agent. An IoT-agent has the wire or wireless connection with an amount of sensors and responds to requests and subscriptions from the IoT-broker on behalf of real sensor nodes. The IoT-broker is responsible for forwarding requests and subscriptions to IoT-agents and pushing returned results back to either the big data repository or the CityModel server.

Two layers of abstraction are introduced by our design for data collection. First, each IoT-agent works as gateway for a data source, hiding the details of all physical sensor nodes for the IoT-broker, for example, network connections to all sensor nodes and communication protocols and interfaces with all different sensors. Second, the IoT-broker further unifies the communication with IoT-agents, allowing other components like the CityModel server to issue a single request to all deployed IoT-agents. The communication protocol between the IoT-broker and all IoT-agents follows standardized NGSI specification [16]. Where the collected data should be reported is up to the destination path in the data requests/subscriptions. To avoid the IoT-broker becomes a bottleneck, the collected data can be directly pushed into the big data repository by IoT-agents. On the other hand, the CityModel server can ask IoT-agents to return the collected real-time data via IoT-broker as well, in order to check the real time status of sensor nodes.

In general, two types of data are collected from different data sources: semi-structured data and unstructured data. The semi-structured data are sensor data in JSON format, mainly collected from the SmartSantander testbed, for example, fixed sensor like parking sensors, traffic sensors, and light sensors, as well as moving sensors attached with cars and buses. The unstructured data can be texts collected from social networks and images/video collected from video surveillance systems deployed in cities.

C. Data Repository

All sensor data collected via IoT-agents from city IoT testbeds are saved into NoSQL database as JSON documents, including the historical data and the latest real-time data. The decision to go for NoSQL database was made regarding the volume of big data generated from IoT testbeds. In the early phase we investigated many NoSQL databases and then the following well-known NoSQL databases remains in our candidate list: CouchDB, CouchBase, MongoDB, and HBase. In the end we select CouchDB after a comprehensive comparison analysis of those databases. More details will be discussed in Section VI to justify this design decision.

With regard to the features of CouchDB and the requirements of our big data platform for smart cities, the following features of CouchDB are well appreciated. First, CouchDB supports map-reduce based view generation and incremental update on the views. Based on this feature, we can easily do real-time and incremental data analytics over the collected raw data to generate statistical results for applications to query. Second, CouchDB is able to notify external components whenever there is a change into the database, such as when a new document is created or some existing document is updated. A filter function can be further defined to watch out certain changes in the database. This allows us to turn a NoSQL database into a pub-and-sub message system as well. Third, CouchDB provides a Built-in HTTP REST API, which allows other components of the big data platform to read and write data directly. This provides convenient and flexible interfaces for external processing modules to do data analytics outside of the big data repository.

The unstructured data are supposed to be very big and usually require intensive processing or analytics to generate well-formatted results, which can be further saved into the NoSQL database. Within the CiDAP platform, all unstructured data are saved as files into HDFS, the Hadoop Distributed File System. Within HDFS, big files are divided into small chunks to be evenly distributed across a set of data nodes and the locations of all chunks are indexed by a name node. To benefit from data locality when doing data processing, we set up a SPARK system and the HDFS system in the same cluster, as shown in Figure 1.

D. Data Processing

Data processing tasks in the CiDAP platform are categorized into two types: *internal processing* and *external processing*, with regard to where data processing is executed. Internal processing is done by the CouchDB database internally while external processing is executed out of the CouchDB database.

Internal processing mainly includes the map and reduce functions to generate indexed views in CouchDB. The CouchDB map-reduce framework can support incremental updating for pre-defined views, which means only the documents that have changed since the last update will be reindexed and reflected into the new view. However, the updating process is not done by CouchDB automatically and it must be triggered by a query. The reindexing and caculation can be done before or after the query result is returned, up to the *stale* parameter in the query. In our implementation, we allow queries to be answered immediately without waiting for the updating to be finished. Therefore, the returned query results are based on slightly outdated views. However, a view could become too stale if there is no query for a long time. To avoid this problem, the CityModel server will issue a query to update all CouchDB views whenever N new documents have been appended into the database or a M seconds timer is timeout (currently N and M are set to 10) .

External processing is more flexible and less limited than internal processing in terms of computation resource and programming language. Logically an external process task fetches semi-structured data from CouchDB or unstructured data from HDFS, performs some data analytics and then outputs derived results back to CouchDB for saving. It can be implemented in any language, as long as it can communicate with big data repository via CouchDB REST API. In principle, internal processing is mainly responsible for indexing views and doing certain simple data aggregation like min, max, avg, grouping, while any intensive and complex data processing and analytics, such as clustering, classification, or data mining, must be done as external processing out of the big data repository. They can be executed on top of the SPARK cluster for doing both batch and stream processing.

Figure 2 illustrates the bottom-up approach to data processing in our platform, showing how raw data can be transformed from unstructured/semi-structured data to structured data (e.g, views, and tables) and aggregated results. At the bottom layer, unstructured raw data saved in HDFS as big files can be transformed by customized external processing tasks into semi-structured data, which will be then stored in the NoSQL database as JSON documents. On the other hand, some external processing tasks can also read documents from the NoSQL database, perform some batch or stream processing, and then put generated results as documents back to the database again. Within the NoSQL database CouchDB, internal processing tasks are made in advance or on the fly to generate indexed views. In the end, through the REST based CityModel API, applications can query aggregated results based on the indexed views in an interactive and flexible manner.

E. CityModel API

All external applications communicate with the CiDAP platform via the CityModel server based on a REST based API, called *CityModel API*. The CityModel API allows application to do *simple query*, *complex query*, and *subscription*. A simple query requests aggregated results over the latest status of all sensors, which represent the latest and real-time snapshot of the entire city testbed, while a complex query can request aggregated results over the historical data collected within a specified time range. Subscription is the mechanism to keep

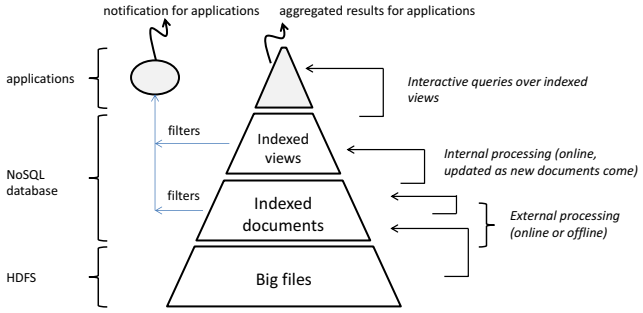

Fig. 2: bottom-up approach of data processing

applications always notified with the latest results so that the application do not have to query the data all the time. There are two types of subscriptions, *CacheDataSub* and *DeviceDataSub*, as illustrated by Figure 3. The difference is CacheDataSub goes to the data repository CouchDB while DeviceDataSub goes directly to physical devices in the IoT testbed through IoT-broker and IoT-agents. Both of them are designed to notify applications with real-time changes, but with different expected latency. The notification latency for CacheDataSub is relatively longer than the one for DeviceDataSub, because devices will fire notifications immediately after the requested changes happen, without waiting for the next report period. Unfortunately, the DeviceDataSub is not fully working in the integrated system with the SmartSantander testbed because the sensor nodes in this testbed can only report updates in a passive and periodic way.

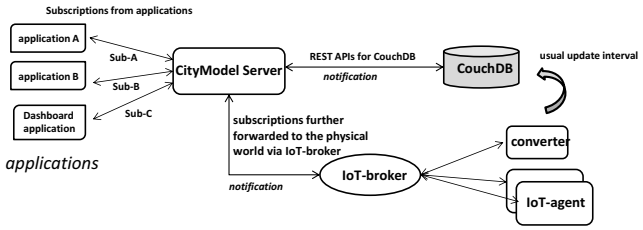

Fig. 3: subscription mechanism to get real-time notifications

IV. System Implementation and Deployment

The entire CiDAP platform has been implemented based on the design presented in Section III. The IoT-broker is reused from NEC's open source project Aeron [14]. A dedicated IoT-agent is developed to integrate the SmartSantander testbed for collecting data. The CityModel server and platform management portal are implemented in JavaScript based on the NodeJS platform. As an example of external processing, an anomaly detection for traffic sensors has been implemented to find out when and which traffic sensors become abnormal for a Smart Maintenance Service of the SmartSantander testbed. All internal processing tasks are implemented in JavaScript to support the CityModel API for external applications. Currently, a comprehensive Dashboard service has been integrated with the CiDAP platform based the implemented CityModel API.

The CiDAP platform has been deployed over the SmartSantander testbed to showcase how collected data can be utilized to improve transportation, parking management, and environment monitoring within a comprehensive City Dashboard service. The CouchDB server is deployed on a dedicated server and the CityModel server and IoT-broker are running on another server. External processing tasks are submitted to a deployed SPARK cluster with 8 nodes, in which Hadoop YARN and HDFS are also deployed. In terms of scalability, the data repository can be scaled up with a cluster of CouchDB server instances, which has been supported by CouchDB 2.0. The bottleneck and single point of failure problem with CityModel Server and IoT-broker can be overcome by scheduling their workload to multiple running instances.

Until this paper is finished, the deployed system has been running for three months in a cloud environment. Within three months about 50GB data has been collected from 1112 sensor nodes. Totally, there are 9 different types of sensor nodes reporting their update values to the CiDAP platform at various time intervals. The time interval is pre-configured for each type of sensor nodes with regard to their power saving and battery lifetime. For example, the time interval for the light and temperature sensors is 60 seconds while for the parking sensors it is 300 seconds. This is because the light and temperature sensors are installed at streetlights with connected power suppliers while the parking sensors are installed underground with only batteries. During each time interval the parking sensors will switch into a sleep mode to save power so that they can be used longer. Each update message is a JSON document in CouchDB. The data size of update messages is 536 bytes on average. In total, about 40 millions unique documents are saved in CouchDB, resulting in about 22GB data. The rest disk space is used by indexes and views as additional overhead.

V. Preliminary Performance Evaluation

The deployed CiDAP system has been working well in the past three months with the current workloads from the SmartSantander testbed and a few applications, but without reaching the limit of the current setup. In order to test the bottleneck and scalability of the CiDAP system, we perform a microbenckmark in our lab based on a similar setup to what has been deployed for SmartSantander. More specifically, this setup consists of 3 machines connected on the same local network with 1Gb/s capacity. Two machines are normal blade servers with the same hardware configuration (2 Intel Xeon E5606 4-cores processors 2.13GHz, 32GB memory): one is assigned to CouchDB and the other is assigned to the CityModel server and the IoT-broker instances. The third machine has 2 Intel Xeon Dual-core processors 3.6GHz and 8GB memory, running a tester to issue simulated queries and updates to the CityModel server.

To keep the test more realistic, we simulate the workloads based on the sensor data collected from the SmartSantander testbed. We mainly evaluate the performance of the big data repository in terms of how many simultaneous updates from sensors can be supported and how many queries from applications can be answered. This is based on the following considerations: 1) these results can help us roughly estimate how much initial resources are required to deploy the CiDAP platform for a given scale of Smart City Testbed; 2) all the other modules, like the IoT-broker, and the CityModel server, are more or less stateless and can be easily scaled up by adding more working instances on the fly without involving any data migration or data loss. Our preliminary results are reported below.

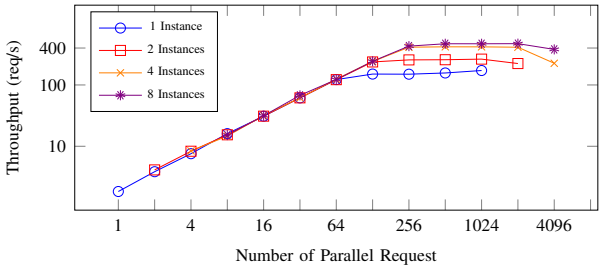

Fig. 4: Throughput of simple queries

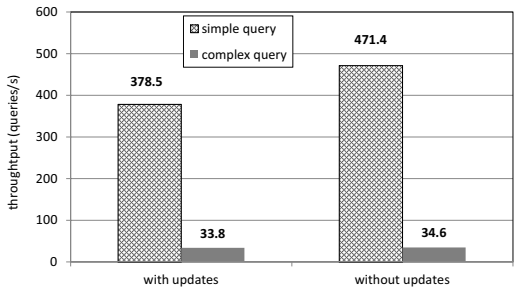

Fig. 5: simple queries vs. complex queries

We first test the throughput of the CityModel server for handling simple queries in different cases. Currently the CityModel server is implemented based on NodeJS, which wraps up all application logics into a single thread, therefore the throughput of the CityModel server is first limited by the number of server instances, rather than the resource of the server machine. To overcome this limit, we launch multiple CityModle server instances to serve application queries. Figure 4 shows the throughput of the CityModel server over a various number of parallel requests when handling simple queries from applications based on a static database, which means we stop collecting live updates from the Santander testbed. The x-axis is the number of parallel requests that the tester issues simultaneously, meaning how many simultaneous simple queries the CityModel server is handling at the same time. As the number of parallel requests increases, more workload will be generated on both the CityModel server and the CouchDB server. Meanwhile, the throughput of the CityModel server increases linearly until we approaching the maximal throughputs. If there is only one CityModel server instance, we can reach about 150 queries/s at most, which is the maximal throughput of a single CityModel server instance. This limit can be extended to about 470 queries/s when we launch 8 parallel CityModel instances. At that time, the CityModel server with multiple instances is no longer the bottleneck. The big data repository becomes the new bottleneck because of its saturated disk I/O. Of course, this limit can be further extended if we deploy multiple CouchDB server instances for the big data repository.

We also perform similar tests to measure the throughput of the CityModel server for handling complex queries. Within the big data repository, different types of views are defined to support simple queries and complex queries. The views for simple queries are only based on the lastest information of each sensor node, much smaller than the views for complex queries, which are generated from all historical data. Therefore, handling a complex query will cause more overhead on CouchDB than a simple query. This can be seen from the

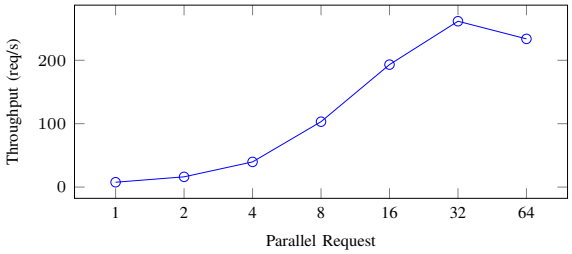

Fig. 6: Throughput of live updates to CouchDB

results in Figure 5. When the live updates to CouchDB are not allowed, the maximal throughput for complex queries is about 35 queries/s, less than 10% of the throughtput of simple queries, which is about 470 queries/s. When the live updates to CouchDB are allowed, the throughputs of both simple queries and complex queries will decrease, but the decrease on simple queries is much larger, about 20%. This is because the large portion of the views for simply queries will be affected by the incoming updates.

We then evaluate how many updates from sensor nodes can be collected by a CouchDB server instance. Figure 6 shows the throughput of updates to the big data repository with a single CouchDB instance. During the test, we send the sensor updates collected from 1007 sensors of Santander in 20 minutes, totally 12927 data samples. By sending more simultaneous updates, we see the throughput of the big data repository increases and then reaches its limit about 300 updates/s when the number of parallel request becomes 32. However, this throughput is measured with an empty CouchDB database. As the data in the database grows over time, the update throughput slowly decreases because the views are becoming bigger and bigger, which triggers more internal disk IO and further affects the update performance. Adding more CouchDB instances will definitely help.

VI. Experience and Lessons Learned

This section discusses the experience and lessons we gain from the *CiDAP* platform as we deploy and integrate it with the SmartSantander testbed and a production SmartCity Dashboard application service.

A. How to Deal with Multiple Data Sources

Data for a smart city are usually collected from multiple data sources. Very often different data sources are provided and managed by different organizations and therefore it is unavoidable that some sensor nodes from different data sources use the same identification occasionally. A duplicated ID for different sensor nodes will cause lots of problems for data saving and aggregation in CiDAP. *To achieve the uniqueness of each sensor node from different data sources, a naming mechanism is needed*. In CiDAP, a uniform resource name (URN) is implemented by all IoT-agents to rename all sensor nodes according to their original IDs. We can add some predefined prefix to express the name of a data source, for example, ”urn:x-iot:smartsantander:1:3323”, which means the sensor node comes from the first source in the SmartSantander IoT testbed with an original ID 3323. To reduce the overhead of saving all historical data, it is better to keep all prefixes short. That is why we use a unique number to identify different

sources in the SmartSantander testbed, rather than their direct URLs. In addition, we maintain a mapping between data source URLs to their unique IDs.

B. Which NoSQL Database to Use

One of the big decision points for our big data platform is to choose a suitable NoSQL database to save collected sensor data. According to our investigation, there are mainly four different types of NoSQL databases to be considered for handling big data: key-value based, wide column based, document based, and graph based. Since all data from the SmartSantander testbed are scheme-less and wrapped up as an JSON object, we narrowed down our choice to document based databases with regard to efficiency and flexibility. Then MongoDB, CouchDB, and CouchBase were seriously considered by us as three main candidates.

CouchDB has been selected due to the following reasons. First, CouchDB can support incremental map-reduce for real-time processing, therefore view results in CouchDB can be updated incrementally as the database changes. But MongoDB does not support dynamically-updated views in the way that CouchDB does. With MongoDB, MapReduce results are written to a collection on disk and/or returned in the query; they are not updated unless the associated jobs get executed again. Second, CouchDB has the unique feature, so called changes notifications, which allows external components to be notified with changed documents in real time. These two features are important to our architecture design and we can only get them from CouchDB.

With these two important features from CouchDB mentioned above, we are able to deal with both real-time data and historical data in the same unified architecture shown in Figure 7. As compared with the existing architectures, e.g., Lambda architecture proposed by Nathan Marz in [17], Kappa architecture designed by Jay Kreps in [18], and Liquid architecture presented by Fernandez et al. in [19], our architecture has the following advantages: First, CouchDB is acting as both the storage layer that saves all received new data and historical data and the message layer that can actively serve external queries and subscriptions to push out data for any further processing. This reduces system complexity and the latency for applications to reach real-time data. Second, CouchDB is also acting as a real-time processing layer which can do simple data processing for real-time data internally. Therefore, the SPARK based external processing layer is optional, up to whether we need to do more intensive and scalable data processing. Third, both CouchDB and Spark can accept python-based map/reduce functions to do data processing. When dealing with a new task, Spark can reuse the map/reduce functions from CouchDB and apply them directly into external data processing.

Of course, there are some other aspects that CouchDB is relatively weak. For example, CouchDB does not support rich queries like MongoDB and was not supporting cluster deployment. But these weakness have been addressed by its latest version.

C. How to Support Data Semantics

In general data semantics increase the interoperability of linked city data, helping us share knowledge across different

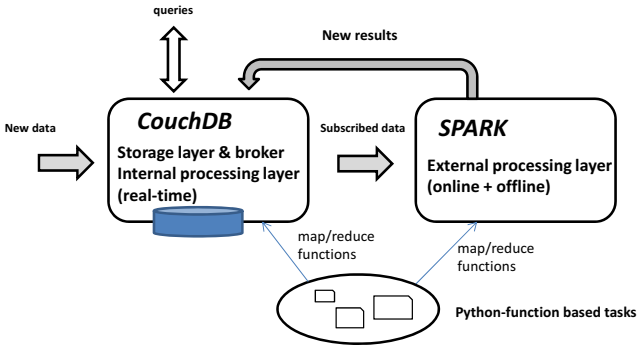

Fig. 7: Architecture to enable online and offline processing

data sources, applications, and services in different domains and cities. Within the project we had lots of discussions on whether and how we need to support data semantics due to the following considerations. On the one hand, we did see some need of data semantics from the Santander City Dashboard service. For example, the Dashboard service needs to query all latest temperature data for a temperature service, but the temperature data come from several types of sensor nodes,which report messages to CouchDB with different data formats, e.g., temperature nodes, light nodes, noise nodes, and env_station nodes. In this case the association between services and node types must be defined at somewhere. On the other hand, however, the requirements of data semantics are still quite limited in the current phase of our project, since most of our data are collected from the same IoT testbed and only a few applications are currently supported, and also data semantics will introduce more complexity and performance overhead if we support them within the big data repository. In the current implementation of CiDAP the CityModel server is taking care of the mapping between services and IoT nodes. We do see this is just a temporary solution. A better approach will be to separate it from the CityModel server and then implement it as a dedicated semantic data service based on a triple store or a graph-based database.

D. Should We Consider Anomalies

For smart cities real time insights rely on the real-time data generated from sensors. The reliability and stability of installed sensors and nodes is important to be taken into account when we analyze the data collected from a city wide IoT testbed, because the collected sensor data might be quite noisy and even affect the correctness of derived results. This is especially true for an experimental testbed. We have observed serious failures of sensor nodes in the SmartSantander testbed. Based on the data set collected between 26th of September and 2th of December in 2014 from the SmartSantander testbed, we observe that about 200 nodes were already abnormal because the data received from them are out of date. Even during the two months when our platform was running, about 100 nodes also became abnormal. There are also some sensor nodes which still send update messages but their reported values are not in the normal range, for example, some temperature sensor nodes report extremely high temperature values. There are various reasons why a sensor node becomes abnormal, for example, a parking sensor might run out of battery, or a repeater might be broken then several associated sensor nodes stop reporting at the same time. Since we do not have the full

knowledge of the testbed, it is not possible for us to identify the concrete reasons why those sensor become abnormal. But this observation indicates for a smart city with a large number of deployed sensors, detecting anomalies in collected data must be seriously considered. That is why we implemented some anomaly detection algorithms as external processing tasks to label the status of each sensor node.

E. Real-time Data Vs. Near-time Data

In the SmartSantander testbed, sensor nodes report update messages in a periodic manner due to the concern of their power consumption. Statistically, a sensor node consumes 9 mA in ON mode, 62 μA in sleep mode, and 0.7 μA in hibernate mode. Within each time interval, a sensor node wakes up, takes readings from the sensors, transmits the data, and returns to sleep mode. This way the sensor node sleeps most of the time to conserve battery life. For different types of sensor nodes, the intervals have been set differently in the SmartSantander testbed. For example, many of the sensor nodes have a more than 60 seconds reporting time period, which means the collected latest sensor data in the big data repository will have at least 60 seconds delay. However, this causes another type of problem because 60 seconds do not fit the requirement of many real-time applications, for example, if an ambulance car wants to check the status of the traffic lights at 100 meters ahead, 60 seconds delay will too much. Therefore, it is important to keep a separate data path from applications to sensor nodes so that applications are able to actively subscribe or request the real-time data from sensor nodes. The sensor nodes should not only report updates at certain time intervals, but also work at the request/subscription driven mode, meaning if some applications need to request or subscribe the latest real-time data the sensor nodes can immediately and directly respond them with the latest results. Of course, this requires more complex processing logic on the sensor nodes and will introduce some extra power consumption. But enabling this direct data path is extremely important for real-time applications that demand less than a few seconds delay. In CiDAP, the data path is established by IoT-broker and IoT-agents and the CityModel server is switching applications requests to real-time data or near-time data.

VII. Conclusion and Open Issues

In this paper we present the architecture overview of a working big data platform called CiDAP and discuss its major design issues. Since October 2014, the CiDAP platform has been deployed and integrated with SmartSantander, one of the largest smart city testbeds in Europe, to collect city data and then serve results to a few real applications. Based on our three-month experience of running this platform, we introduce our observations, experience, and some lessons, which could be timely and valuable inputs for future smart city data platform designers to consider in their own platforms. Of course, our system architecture is designed for certain goals, such as being able to deal with both historical data and real-time data and being flexible to handle different scales/types of data. As we already indicate in the paper, some issues need to be considered for a more advanced smart city big data platform, for example, how to enhance the support of semantic data? how to share knowledge between different applications? how to ensure data security and system security? what are business models? These problems still remain unsovled and we will explore them in our future work.

VIII. Acknowledgement

The work presented in the paper has been partially funded by the European Union's Horizon 2020 Programme under Grant Agreement No. CNECT-ICT-643943 (FIESTA-IOT).

References

[1] L. Sanchez, L. Muñoz, J. A. Galache, P. Sotres, J. R. Santana, V. Gutierrez, R. Ramdhany, A. Gluhak, S. Krco, E. Theodoridis, and D. Pfisterer, "Smartsantander: Iot experimentation over a smart city testbed," *Journal Computer Networks: The International Journal of Computer and Telecommunications Networking*, vol. 61, pp. 217–238, Mar. 2014.

[2] (2014) The smartsantander project. [Online]. Available: http://www.smartsantander.eu

[3] L. Sanchez, J. Galache, V. Gutierrez, J. Hernandez, J. Bernat, A. Gluhak, and T. Garcia, "Smartsantander: The meeting point between future internet research and experimentation and the smart cities," in *Future Network Mobile Summit*, Warsaw, Poland, Jun. 2011.

[4] J. Bernat, J. M. Hernandez, L. A. Hernandez, and A. Tierno, "An experimental platform for large-scale research facing fi-iot scenarios," in *Future Network Mobile Summit*, Warsaw, Poland, Jun. 2011.

[5] I. Vilajosana, J. Llosa, B. Martinez, M. Domingo-Prieto, A. Angles, and X. Vilajosana, "Bootstrapping smart cities through a self-sustainable model based on big data flows," *Communications Magazine, IEEE*, vol. 51, no. 6, pp. 128–134, June 2013.

[6] N. Walravens and P. Ballon, "Platform business models for smart cities: from control and value to governance and public value," *Communications Magazine, IEEE*, vol. 51, no. 6, pp. 72–79, June 2013.

[7] "SCOPE: A Smart-city Cloud-based Open Platform and Ecosystem," http://www.bu.edu/hic/research/scope/.

[8] "FIWARE Open Source Platform," http://www.fi-ware.org/.

[9] M. K. P. Fritz and J. Kwan, "IBM Smarter City Solutions on Cloud," 2012.

[10] M. Strohbach, H. Ziekow, V. Gazis, and N. Akiva, "Towards a big data analytics framework for iot and smart city applications," in *Modeling and Processing for Next-Generation Big-Data Technologies*. Springer, 2015, pp. 257–282.

[11] "Microsoft CityNext Solution," http://www.microsoft.com/global/en-us/citynext/RichMedia/Safer_Cities/CityNext_Brochure_Safer_Cities_SML_FY15.pdf.

[12] (2014) The citypulse project. [Online]. Available: http://www.ict-citypulse.eu

[13] (2014) Singapore smart nation platform. [Online]. Available: http://www.ida.gov.sg/Infocomm-Landscape/Smart-Nation-Vision

[14] "Aeron open source project," https://github.com/Aeronbroker/Aeron/.

[15] (2014) The apache spark project. [Online]. Available: https://spark.apache.org

[16] "NGSI 9/10 information model," http://technical.openmobilealliance.org/Technical/Release_Program/docs/NGSI/V1_0-20101207-C/OMA-TS-NGSI_Context_Management-V1_0-20100803-C.pdf.

[17] "Lambda Architecture," http://lambda-architecture.net/, 2014.

[18] "Kappa Architecture," http://radar.oreilly.com/2014/07/questioning-the-lambda-architecture.html, 2014.

[19] R. C. Fernandez, P. Pietzuch, J. Kreps, N. Narkhede, J. Rao, J. Koshy, D. Lin, C. Riccomini, and G. Wang, "Liquid: Unifying nearline and offline big data integration," in *The biennial Conference on Innovative Data Systems Research (CIDR'15)*, Jan. 2015.