CERN Storage Systems for Large-Scale Wireless

September 2013

Author:
Andrea Azzarà

Supervisors:
Xavier Espinal
Massimo Lamanna

CERN openlab Summer Student Report 2013

Abstract

The project aims at evaluating the use of CERN computing infrastructure for next generation sensor networks data analysis. The proposed system allows the simulation of a large-scale sensor array for traffic analysis, streaming data to CERN storage systems in an efficient way. The data are made available for offline and quasi-online analysis, enabling both long term planning and fast reaction on the environment.

Sommario

1 Introduction

1.1 Large Scale Wireless Sensor Networks

A Wireless Sensor Network (WSN) consists of spatially distributed autonomous devices which cooperate, by means of wireless communication, in order to fulfil their tasks, which are generally based on the monitoring of physical or environmental conditions, or most recently in acquiring and processing multimedia information. The sensor network is a network consisting of spatially distributed devices equipped with sensors which are used to collect physical data and monitor environmental condition at different locations. In a typical sensor network, nodes cooperate to complete the task of collecting raw data and return to the application back-end. In essence, the nodes within such a network have at least computation, wireless communication, and sensing functionality, all characteristics which candidate WSNs to become the key technology for pervasive sensing and computing.

1.1.1 Network protocols for WSN

Over the past few years the research community has focused on designing protocols to enable the integration of WSN nodes into the Internet world, following the Internet of Things (IoT) concept. Indeed, protocols for the traditional Internet are often unsuited for embedded devices that are usually constrained in terms of computing power, memory, and network interfaces. The main outcome of this research effort is an adaptation layer for IPv6 allowing to transmit IPv6 packets over IEEE 802.15.4 networks. 6LoWPAN, i.e., an adaptation of IPv6 for low-power devices, have proven to be a valid alternative to traditional WSNs employing proprietary protocols[1]. Another important step towards the IoT is the on-going drafting of the Constrained Application Protocol (CoAP), an HTTP-like protocol especially designed for constrained devices. CoAP permits to create embedded web services running on IoT nodes[2], thus extending the successful web architecture, based on the REST paradigm, to the IoT. The basic idea is that IoT nodes are responsible for one or more resources that may represent sensors, actuators, combinations of values or other information. Internet devices can send messages to change and query resources on IoT nodes. IoT nodes can also send notifications about changed resource values to devices that have subscribed to receive notification about changes.

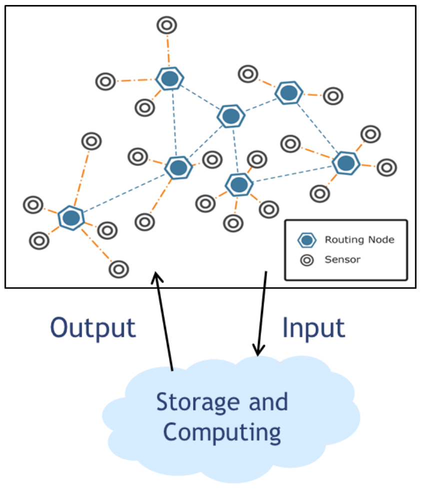
1.2 Motivation

Wireless Sensor Networks are increasingly used in various application domains like home-automation, agriculture, industries and infrastructure monitoring. Potential worldwide deployment of WSNs for, e.g., environmental monitoring purposes could yield data in amounts of petabytes each year. Deployments of Large-scale WSNs (LS-WSNs) have potential to shed light on a number of environmental phenomena, such as progression of climate change, pollution levels, effects of urbanization, water management, and so on. Large Scale WSN deployment require development of solutions for storing and reasoning about the potentially huge amounts of data they would generate.

Different application may have different demands on data sources and data processing, and different requirements in terms of timeliness. We Consider the case study of a large scale Intelligent Transportation System (ITS) in which a large number of small sensor are deployed as road side units all over the road network. There are many possible applications for such network, with different timeliness requirements:

1. Vehicle tracking; Vehicle can be tracked in different ways, using smart camera sensors, RFID. Vehicle tracking is a hard real-time application.
2. Traffic dispatching; Traffic dispatching need real-time data of traffic flow, weather data and so on. Traffic scheduling is a soft real-time application, as several minutes lag will not cause suffering result. If all the traffic related data are sent to the data centres of the cloud, the inflowing data may cause network congestion. If the scheduling algorithm is only based on the real-data, there is no need to store the collected data in the cloud, but if it is a prediction algorithm based on historical data, the collected data must be stored in the cloud.
3. Expressway planning. Unlike in the scenario of traffic dispatching, expressway planning does not need real-time data, but need huge historical data. The cloud data centres can collect statistical data from local servers, and complete the computation using the cloud computing resources.

From the analysis of these case studies it is clear that a common requirement is to run complex analysis on large data coming from distributed sensors, and possibly use the output of the computation to inject a feedback to the environment. In this project we consider the problem of interconnecting a large-scale WSN infrastructure to a "cloud" of storage and computing systems. In particular the project has the goal to provide a preliminary evaluation of the requirements and issues of a control loop between the sensor system and the computing infrastructure, in order to provide a feedback on the environment. Both "real-time" and "long-term" analysis use cases has been considered, with different pattern of data analysis: data-intensive (map-reduce) and CPU-intensive (batch processing).



2 Related Work

In recent years the research community started to consider the problem of interconnecting large-scale WSN to storage and computing infrastructures. In [3] the adoption of the Map-reduce programming model for the processing of sensor data is evaluated. The high costs of deploying WSNs in harsh environments providing no infrastructure result into a non-uniform geographical distribution of sensor nodes. The spatial interpolation of a phenomenon in such regions is highly important. In the proposed solution data from VLS WSN are collected in tables HBase, then statistical geospatial algorithms (dealing with missing measurements) are applied. Spatial-temporal analysis is based on data windowing, i.e. selecting a geographical region and applying the analyses on the window-contained nodes alone. The Map-Reduce model is then applied on tables representing large matrices.

In [4] a data processing model based on Cloud computing for very large scale sensor network is proposed. Specifically the authors deal with the problems of how to store the data and where to process it; if all the data are send to the data centres of the cloud at the sampling time, the inflow of massive data to the wide-area networks may cause network congestion. Moreover If all the data of the sensors are processing at the cloud, the communication latency may make the applications requiring for real-time demand intolerable. The authors propose a multilevel data processing model based on cloud computing. The architecture of the system is shown in Figure 1. In this model the massive sensor data and node information are stored in multilevel data storage including local data bases of different devices and distributed data bases of cloud; Different kinds of computations are decomposed and distributed on different nodes mainly considering their differences on computation ability and power supply.

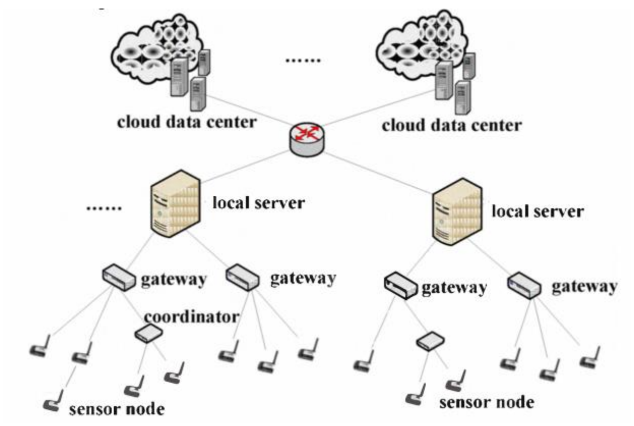


Figure 1 Multilevel data storing architecture

3 Scalable WSN Emulation

It is rather complex, or even unfeasible, to model analytically a WSN and it usually leads to oversimplified analysis with limited confidence. Besides, deploying test-beds supposes a huge effort. Therefore, simulation is essential to study WSN. However, it requires a suitable model based on solid assumptions and an appropriate framework to ease implementation. In addition, simulation results rely on the particular scenario under study (environment), hardware and physical layer assumptions, which are not usually accurate enough to capture the real behaviour of a WSN, thus, jeopardizing the credibility of results. However, detailed models yields to scalability and performance issues, due to the large number of nodes, that depending on application, have to be simulated. Therefore, the trade-off between scalability and accuracy becomes a major issue when simulating WSN.

In this project a mechanism to support large-scale WSN emulation has been developed. With WSN Network emulation the virtual simulation environment is integrated with a small number of real hardware devices to facilitate performance evaluation of real-life devices in a large-scale but well-controlled environment. Network emulation is an inexpensive approach to testing, validating, and evaluating protocols and techniques in a realistic but well controlled network environment[5].

The first part of the project consisted in the development of an infrastructure supporting large-scale WSN emulation. In the proposed system the WSNs are executed in the well-controlled virtual environment while the other components of the system run on the real environment. The next section will describe the system's architecture and modules.

3.1 Modules and Architecture

In this section the main the architecture of the system will be described starting from an overview of the its sub modules.

3.1.1 Cooja

Cooja is the Contiki[1] network simulator. Contiki is an open source operating system for the Internet of Things connecting tiny, low-cost, battery-operated and low-power systems to the Internet. Cooja allows large and small networks of Contiki motes to be simulated. Motes can be emulated at the hardware level, which is slower but allows precise inspection of the system behaviour, or at a less detailed level, which is faster and allows simulation of larger networks. Figure 2 shows a TMote Sky, the mote selected for emulation in Cooja. TMote Sky is a platform for extremely low power, high data-rate, sensor network applications. TMote Sky is equipped with a MSP430 MCU with 10Kb of on-chip RAM and a IEEE 802.15.4 radio transceiver.

[1] www.contiki-os.org

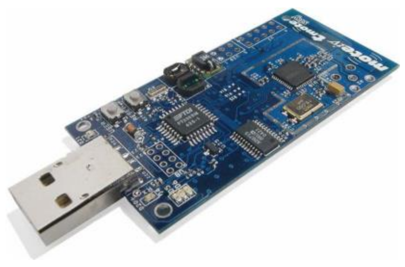


Figure 2 TMote Sky

3.1.2 PyoT

PyoT[2] is a system for macro-programming and managing IoT-based WSNs. PyoT abstracts the WSN as a set of resources (i.e., sensors and actuators) that can be manipulated and combined in order to perform complex tasks. Specifically, PyoT allows the user to discover available resources, monitor sensor data, handle its storage, control actuators, define events and the actions to be performed when they are detected, and interact with resources using a scripting language (macro-programming). The high-level abstraction provided by PyoT completely hides the nodes and the network, letting users and application developers focus on the sensing and actuation capability of the system. PyoT provides two ways for interacting with resources: a convenient web-based user interface (WUI) and a powerful macro-programming mechanism. The WUI is designed as a virtual control room that allows for easy execution of basic operations such as resource listing, sensor monitoring, actuator control, event detection and reaction, and access to historical data. The macro-programming mechanism allows the use of a high-level language, specifically Python, for defining more complex operations (e.g., operations involving group of sensors or having an elaborate logic). Indeed, PyoT provides a set of Python APIs for interacting with resources, which are abstracted as Python objects. Using the macro-programming mechanism, the user can programmatically access all the basic operations provided by PyoT's WUI and combine them to build IoT applications. PyoT also includes a shell interface that allows experienced users to use the macro-programming mechanism interactively.

Subcomponents

Figure 6 presents the architecture of PyoT. PyoT is designed to be a distributed system running in an IP network (possibly the Internet) and having at least four main components (excluding the WSNs it manages): a Virtual Control Room (VCR), a Shell, a Storage Element (SE), and a PyoT Worker Node (PWN).

[2] http://github.com/tecip-nes/pyot

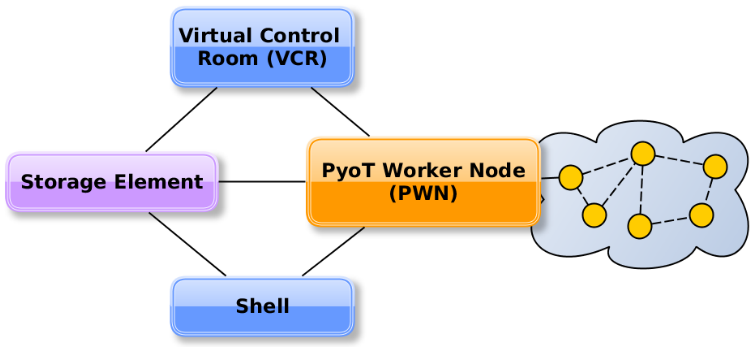


Figure 3 PyoT Architecture

The PWN is the point of service to the IoT-based WSNs. The PWN keeps track of the nodes and resources available in the WSN and updates the related information stored in the SE. Moreover, it manages the sensor data collection and the event detection. If the user subscribes to some sensor data, the PWN monitors the proper resources for changes and adds new sensor values to the SE every time they become available. The basic operations that can be performed on resources (e.g., resource retrieval, resource monitoring, etc.) are implemented as tasks that can be allocated to different PWNs, each one managing a different WSN. PyoT task distribution systems is depicted in Figure 4 and is based on The Advanced Message Queuing Protocol (AMQP), an open standard application layer protocol for message-oriented middleware.

During the project, PyoT has been extended to support a variable number of WSNs providing a scalable solution for the realization of a large-scale WSN infrastructure. Every network has been associated to a different IPv6 network address. Tasks performing interaction with WSN nodes are dispatched to the right PWN based on the IPv6 address of the nodes using AMQP routing mechanism.

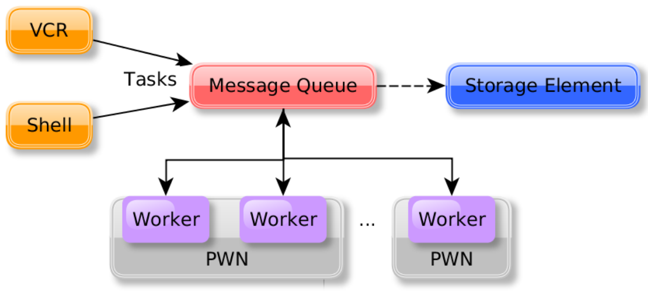


Figure 4 PyoT task distribution

3.1.3 OpenStack

OpenStack[3] is an open source infrastructure as a service (IaaS) initiative for creating and managing large groups of virtual private servers in a cloud computing environment. The

[3] http://www.openstack.org/

technology consists of a series of interrelated projects that control pools of processing, storage, and networking resources throughout a datacentre, all managed through a dashboard. OpenStack has a modular architecture that currently has three components: compute, storage and image service.

- OpenStack Compute - a cloud computing fabric controller for provisioning and managing large networks of virtual machines (VMs).
- OpenStack Object Storage - a scalable storage system that provides support for both object storage and block storage.
- Image Service - a delivery service that provides discovery and registration for virtual disk images.

3.1.4 Monitoring

Monitoring is realized using Celery Flower[4], a tool allowing real-time monitoring of tasks progress and history. Flower is able to show task details (arguments, start time, runtime, and more), plotting live graphs and statistics. Moreover Flower allows remote control of tasks, shutdown and restart of worker instances, control worker pool size and auto-scale settings, apply time and rate limits etc. Figure 5. shows a screenshot of Flower interface.

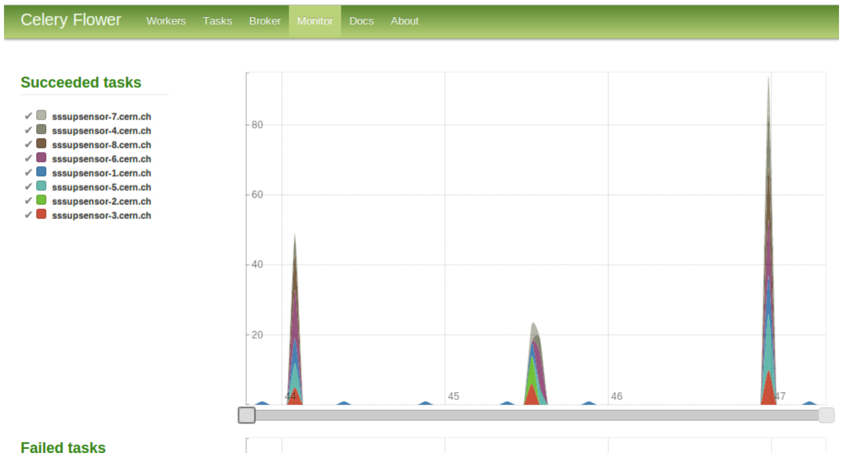


Figure 5 Celery monitoring interface

[4] https://github.com/mher/flower

3.1.5 Global Architecture

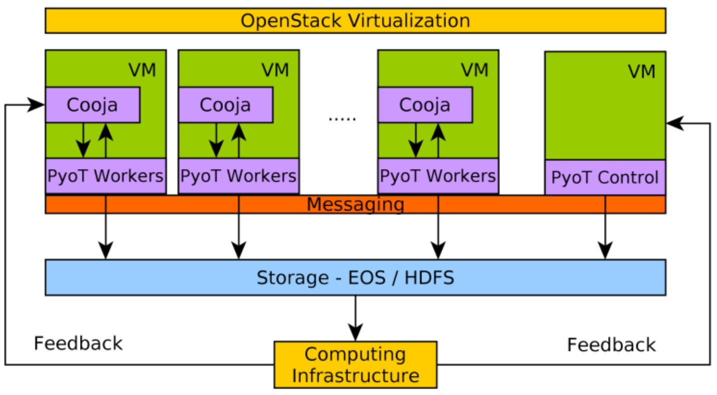


Figure 6 PyoT Architecture

The architecture of the system is represented in Figure 6. A number of Virtual Machines are instantiated using the OpenStack Virtualization framework. Each VM hosts an instance of a WSN simulator (Cooja), connected to the external world through a PyoT Worker Node. A messaging layer is implemented to manage the communication between the various PyoT Worker nodes. One specific VM (PyoT Control node) hosts the applications and servers which are in charge of managing the system. More specifically the PyoT Control node runs:

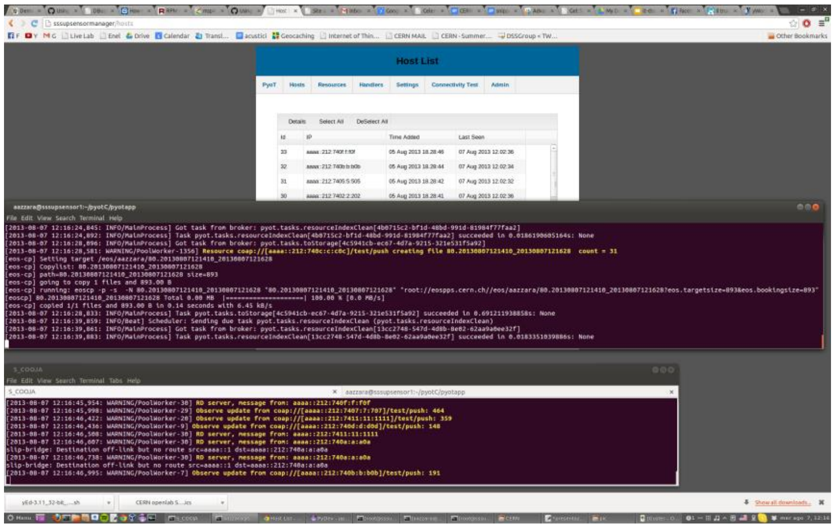
- an Apache Web Server;
- a MySQL Database;
- the RabbitMQ AMQP broker.

The simulated WSN network is composed by a set of CoAP Server nodes, each one hosting virtual sensors and virtual actuators. The PyoT Workers are in charge of communicating with the Storage Layer, which is realized by CERN EOS system and by the Hadoop File System (HDFS). The storage operation is handled by EOS and HDFS clients respectively. The emulator is instantiated when the Cooja simulators are started inside each VM. Then a macro-programming script is executed inside the PyoT Shell in order to start the interaction with the nodes: specifically the scripts sets up a subscription with each node of the WSN using the Observe mechanism of CoAP. The subscription is handled by a CoAP client instance running inside a PyoT worker. When the resource monitored by the WSN node changes the node sends a notification message to the client, containing the current representation of the resource. The client usually stores each message in a primary instance of the Storage Element, local to the PWN.

The experiments have been run using 8 virtual machines, each one simulating a WSN composed by 16 nodes, plus one VM executing the PyoT Control Centre. In order to demonstrate the data flow, the virtual sensors generate random readings (integers) with a fixed frequency of two seconds. More realistic readings may be produced for example simulating the tracking of some object inside the network. The number of WSN nodes inside a single network is generally limited by a number of factors. In this case the most important ones are the limited bandwidth available for the exchange of internal routing messages and the overload on the gateway node, which is the bottleneck of the communication from the WSN to the external network.

3.2 EOS and HDFS Storage

The secondary storage (to EOS and HDFS systems) operation is performed by some specific tasks instantiated on PyoT Worker nodes. The screenshot in Figure 7 shows a storage operation performed on a worker node, sending a file to the EOS system. Storage has been implemented following three alternative patterns: Sensor, WSN and Global levels. The patterns respond to different requirements in terms of latency and efficiency of the storage operation. In this context latency of storage is the interval of time between the reception of the notification from the sensor and the moment in which the data is present in the storage infrastructure. Latency is an important factor if a prompt reaction, based on the result of the computation, is required.



Id	IP	Time Added	Last Seen
33	aaaa::212:740f:f:f0f	05 Aug 2013 18.28:46	07 Aug 2013 12.02:36
32	aaaa::212:740b:b:b0b	05 Aug 2013 18.28:44	07 Aug 2013 12.02:34
31	aaaa::212:7405:5:505	05 Aug 2013 18.28:42	07 Aug 2013 12.02:32
30	aaaa::212:7402:2:202	05 Aug 2013 18.28:41	07 Aug 2013 12.02:36

```
aazzara@sssupsensor1:~/pyotC/pyotapp
[2013-08-07 12:16:24,845: INFO/MainProcess] Got task from broker: pyot.tasks.resourceIndexClean[4b0715c2-bf1d-48bd-991d-81984f77faa2]
[2013-08-07 12:16:24,892: INFO/MainProcess] Task pyot.tasks.resourceIndexClean[4b0715c2-bf1d-48bd-991d-81984f77faa2] succeeded in 0.0186190605164s: None
[2013-08-07 12:16:28,096: INFO/MainProcess] Got task from broker: pyot.tasks.toStorage[4c5941cb-ec67-4d7a-9215-321e531f5a92]
[2013-08-07 12:16:28,581: WARNING/PoolWorker-1356] Resource coap://[aaaa::212:740c:c:c0c]/test/push creating file 80.20130807121410_20130807121628  count = 31
[eos-cp] Setting target /eos/aazzara/80.20130807121410_20130807121628
[eos-cp] Copylist: 80.20130807121410_20130807121628
[eos-cp] path=80.20130807121410_20130807121628 size=893
[eos-cp] going to copy 1 files and 893.00 B
[eos-cp] running: eoscp -p -s  -N 80.20130807121410_20130807121628 "80.20130807121410_20130807121628" "root://eospps.cern.ch//eos/aazzara/80.20130807121410_20130807121628?eos.targetsize=893&eos.bookingsize=893"
[eoscp] 80.20130807121410_20130807121628 Total 0.00 MB  |====================| 100.00 % [0.0 MB/s]
[eos-cp] copied 1/1 files and 893.00 B in 0.14 seconds with 6.45 kB/s
[2013-08-07 12:16:28,833: INFO/MainProcess] Task pyot.tasks.toStorage[4c5941cb-ec67-4d7a-9215-321e531f5a92] succeeded in 0.691211938858s: None
[2013-08-07 12:16:39,859: INFO/Beat] Scheduler: Sending due task pyot.tasks.resourceIndexClean (pyot.tasks.resourceIndexClean)
[2013-08-07 12:16:39,861: INFO/MainProcess] Got task from broker: pyot.tasks.resourceIndexClean[13cc2748-547d-4d8b-8e02-62aa9a0ee32f]
[2013-08-07 12:16:39,883: INFO/MainProcess] Task pyot.tasks.resourceIndexClean[13cc2748-547d-4d8b-8e02-62aa9a0ee32f] succeeded in 0.0183351039886s: None
```

```
S_COOJA
[2013-08-07 12:16:45,954: WARNING/PoolWorker-30] RD server, message from: aaaa::212:740f:f:f0f
[2013-08-07 12:16:45,998: WARNING/PoolWorker-29] Observe update from coap://[aaaa::212:7407:7:707]/test/push: 464
[2013-08-07 12:16:46,422: WARNING/PoolWorker-28] Observe update from coap://[aaaa::212:7411:11:1111]/test/push: 359
[2013-08-07 12:16:46,436: WARNING/PoolWorker-9] Observe update from coap://[aaaa::212:740d:d:d0d]/test/push: 148
[2013-08-07 12:16:46,508: WARNING/PoolWorker-30] RD server, message from: aaaa::212:7411:11:1111
[2013-08-07 12:16:46,607: WARNING/PoolWorker-30] RD server, message from: aaaa::212:740a:a:a0a
slip-bridge: Destination off-link but no route src=aaaa::1 dst=aaaa::212:740a:a:a0a
[2013-08-07 12:16:46,738: WARNING/PoolWorker-30] RD server, message from: aaaa::212:740a:a:a0a
slip-bridge: Destination off-link but no route src=aaaa::1 dst=aaaa::212:740a:a:a0a
[2013-08-07 12:16:46,995: WARNING/PoolWorker-7] Observe update from coap://[aaaa::212:740b:b:b0b]/test/push: 191
```

Figure 7 PyoT storing data on EOS

3.2.1 Sensor level

When the system is running, a CoAP client is active listening for notifications from each sensor node in the WSN. In this scenario the CoAP client itself is responsible for storing the information received from the sensor on the Storage system. When a certain number of records is reached a temporary file is created and the storage client is called (EOS or HDFS). In this configuration a file is periodically stored for every sensor resource in the network (see Figure 8). This pattern can generate a large overhead if the number of sensors is large. The latency depends mainly on the desired chunk size, that is the number of records to save temporarily on the worker node before the storage. This pattern is effective if a fine control on the behaviour of some specific node is required.

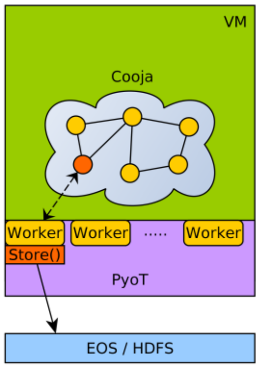


Figure 8 Sensor level storage

3.2.2 WSN level

In this scenario data from sensors are stored in a temporary database on each PyoT worker node. A periodic task is activated on each Worker node, checks the local database and collects all the messages received from the WSN nodes in the previous period (see Figure 9). Then a temporary file is created and uploaded on the storage system. This approach reduces significantly the number of files stored in the system (one for each WSN in each period). A database on each worker node is used in order to guarantee the persistency of the collected data within the period.

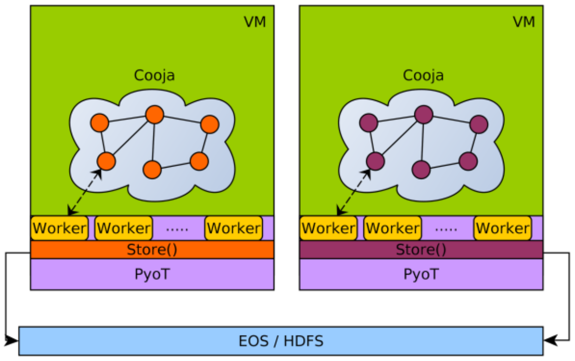


Figure 9 WSN level storage

3.2.3 Global level

In this case one periodic task for the whole network of WSNs (see Figure 10) collects all the messages received in the previous period and stores them. The task can be executed on any of the PyoT worker nodes: if the job fails, the execution can be retried on another node. In the global level case a common database is used to temporarily store the data. Since the database is accessed by every worker process on all the workers node, it may easily become the bottleneck; redundancy and clustering of the database may be considered to improve reliability and availability of the system.

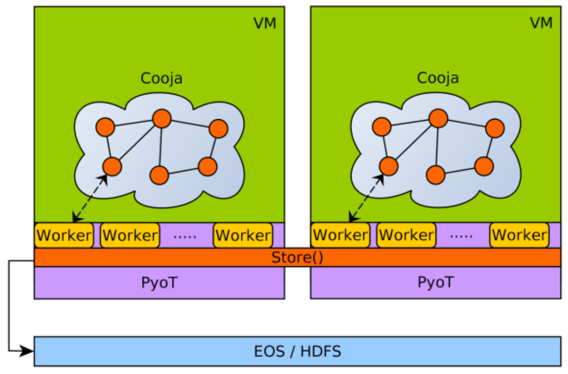


Figure 10 Global level storage

3.3 Workflow

Figure 11 represents workflow implemented in the project. The large-scale WSN emulator streams data to the storage systems (EOS and HDFS). Data is then analysed using both LSF batch jobs and Hadoop Map-reduce. The result of the computation is used to send a feedback to the WSN. In order to validate the workflow a simple algorithm has been employed. The algorithm simply counts the number of messages received by each sensor node. The feedback function has been implemented switching a set of virtual actuators inside the WSN, using PyoT's macro-programming feature.

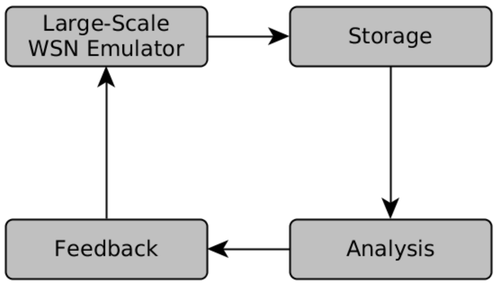


Figure 11 WSN Emulation workflow

3.3.1 LSF batch processing workflow

Typically the user sends a shell script containing the processing algorithm to the LSF batch system. The script is responsible for transferring the data set from EOS using the *xroot* protocol. Then the script executes the algorithm, possibly calling an external program, and finally stores the results on EOS.

3.3.2 Hadoop Map-Reduce workflow

Unlike LSF workflow, Map-Reduce does not require an additional data transfer. The framework is responsible for sending the processing function to the right data node. At the end of the computation the results are available on HDFS. The experiments have been performed using Hadoop Streaming API which helps passing data between Map and Reduce functions via standard input and standard output.

3.3.3 Results

The experiment has been run using 8 Virtual Machines, hosting a total of 128 WSN nodes. The resulting rate of the data collection is around 151KRecords/h. A total of ~30K Files has been stored on both EOS and HDFS. The analysis of ~250MB of data on Hadoop required ~1 minute to complete. During the experiments a large number of small files has been created on EOS and HDFS. To mitigate the overhead of the storage a mechanism to compact small files is recommended. This can be easily realized with a periodic job aggregating the small files.

For both LSF and Hadoop systems the speed of the analysis is depending on many factors and is generally unpredictable. The main factor is the availability of the computing resources. Though many techniques can be adopted to provide guarantees to the execution, these were outside the scope of the project and have not been adopted.

4 Data Analysis

The second part of the project focused on the study of analysis techniques on data coming from large-scale sensor networks. The case study is that of city-scale ITS system equipped with a pervasive network of vehicle detectors. More specifically the problem of estimating the Origin-Destination-matrix from traffic count has been considered [6]. A traffic simulator has been used in order to generate realistic data collected by the detectors.

4.1 SUMO

SUMO, "Simulation of Urban MObility", is an open source, microscopic, multi-modal traffic simulation. It allows to simulate how a given traffic demand which consists of single vehicles moves through a given road network. The simulation allows to address a large set of traffic management topics. It is purely microscopic: each vehicle is modelled explicitly, has an own route, and moves individually through the network. SUMO includes all applications needed to prepare and perform a traffic simulation, supports different vehicle types, multi-lane streets with lane changing, traffic lights, loop detectors etc. Moreover SUMO is capable of supporting large-scale simulations (up to 500.000 vehicles). Figure 12 shows a road intersection in SUMO simulator.

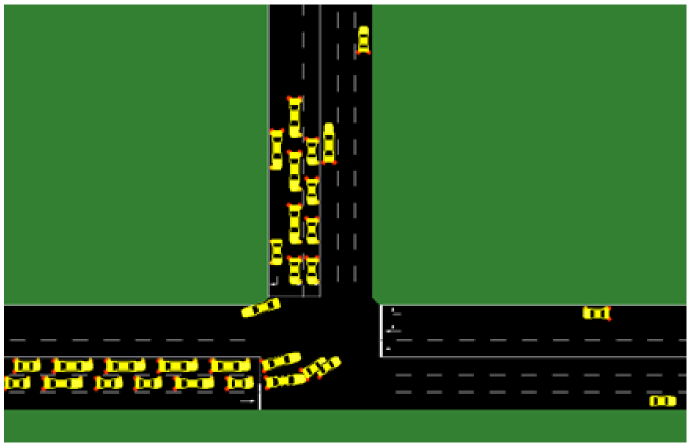


Figure 12 Road intersection in SUMO

A SUMO simulation is generated from different components: a road network, a traffic demand, and a definition of induction loop detectors.

Road Network A SUMO network file describes the traffic-related part of a map, the roads and intersections the simulated vehicles run along or across.

Traffic Demand The traffic demand describes the information about the vehicles. Specifically a trip is a vehicle movement from one place to another defined by the starting edge (street), the destination edge, and the departure time. A route is an expanded trip, that means, that a route definition contains not only the first and the last edge, but all edges the vehicle will pass. There are several ways to generate routes for SUMO. The choice depends on the available input data. Traffic demand is often defined using Origin-Destination-Matrices (or OD-matrices), which are often available from traffic authorities. OD-matrix matches trip makers' origins and destinations to develop a "trip table", a matrix that displays the number of trips going from each origin to each destination, describing traffic pattern between various zones.

Inductive loops Inductive loops are wire loops buried in pavement, connected to an inductive loop detector in the controller cabinet. Detector is "tuned" to normal inductance of loop, and when a vehicle passes over the loop, the loop inductance changes, and the detector sends an impulse to the controller. Single loop detector stations (one per lane) are typically deployed on freeways for real time traffic monitoring. SUMO supports both common loop detectors and instantaneous loop detectors. While common loop detectors aggregate the number of vehicles per time unit, the output of instantaneous loop detectors is an instantaneous measure for every vehicle detected. In the second case a vehicle identificator is part of the measure, enabling more complex applications like vehicle tracking. The reconstruction of the OD-matrix for an urban scenario is only possible when instantaneous loop inductor detectors are used. Conversely the use of common induction loop detectors (providing an aggregated measure) is only effective in highway scenario, for which entry and exit points can be monitored accurately.

SUMO provides a set of tools to create the routes starting from a definition of an OD-matrix. The procedure to create the routes is as follows.

1. Network creation. The network can be created starting from a real map, importing a road network from Open Street Map[5] or defining an abstract network (using the random network generator) for testing purpose.
2. OD-matrix definition. A set of traffic assignment zones (TAZ) - defines the districts. Each district is defined by a set of edges, for which input and output probabilities are assigned. OD-matrix is then associated with TAZs, defining the count of vehicles entering and leaving the zone.
3. Trips definition. Using the OD2TRIPS tool a set of trips is created starting from the network definition and the OD-matrix.
4. Routes definition. Starting from network and trip files the DUAROUTERS software creates the routes.

4.2 Objective

The main goal is to create a set of traces, obtained from SUMO traffic simulator, describing the passage of vehicles through the network (routes). The traces simulate the data collected by a large-scale sensor network. Following the workflow depicted in Figure 13 the collected data is analysed to reconstruct the OD-matrix and compared with the initial input data. The analysis is performed offline, using both batch jobs and Map-reduce.

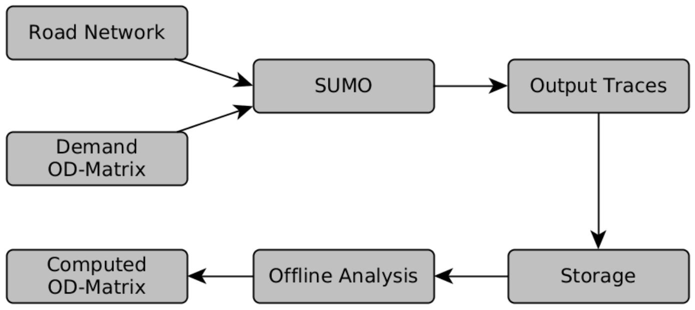


Figure 13 Data analysis workflow

4.3 Evaluation

In order to evaluate the effectiveness of the workflow a simple road network has been manually defined. The network includes a set of 16 Instantaneous loop detectors. The virtual detectors are placed in a simple road network, shown in Figure 14, composed by 8 nodes, 16 edges (lanes), 4 intersections, and 4 TAZs (named A, B, C, D). A simple OD-matrix has been defined in order to generate a sustained traffic in the network while preventing excessive congestions.

[5] http://www.openstreetmap.org

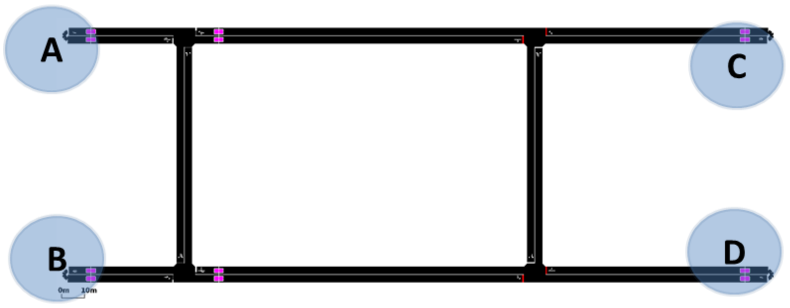


Figure 14 Simple road network

The simulation has been run for 8 days of simulated time. The raw output, containing the measurements from the instantaneous induction loop detectors, has approximate size of 100MB. In order to have a more realistic idea of the time needed to process the data the dataset has been expanded, replicating the output 100 times, creating a "virtual" output of 100 simple road networks.

Considering the analysis performed using LFS batch job system, a script has been created in order to compute the OD-matrix starting from the output of the simulator. Specifically the script:

1. fetches the data from EOS;
2. searches for all the routes travelled by vehicles, for every route it computes the number of vehicles that have travelled through it;
3. computes the Origin/Destination matrix for the network;
4. writes the results to the standard output.

The number of routes per trip is shown in Figure 15. The output O/D matrix, represented in the bi-dimensional histogram in Fig. YYY, reflects the input matrices, confirming the correctness of the calculation. A small difference between the original matrix and the computed one is possible because of some vehicles teleporting (moving across non adjacent edges) in SUMO simulation.

The histogram in Figure 16 shows the computed number of trips for all the possible routes. The analysis of ~10GB of data on Hadoop required ~2 minute to complete. Anyhow the results are approximate because the experiments could not be executed in a well-controlled environment.

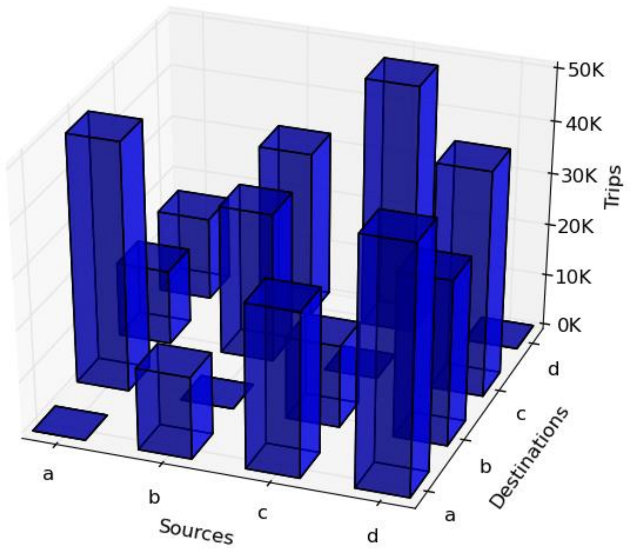


Figure 15 Computed OD-matrix

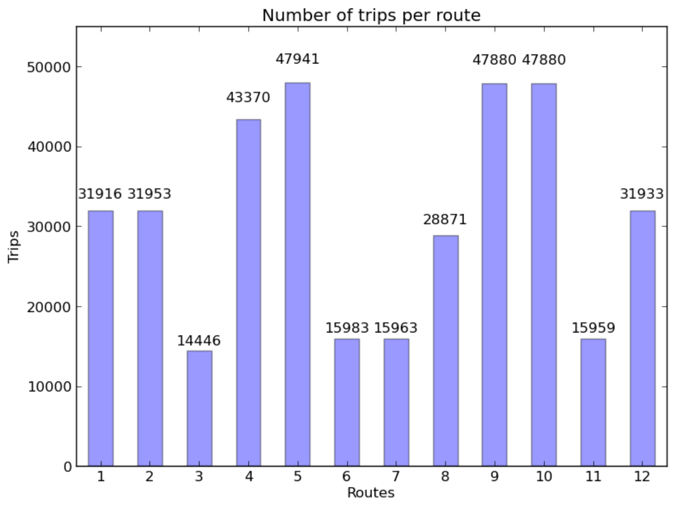


Figure 16 Computed number of trips per route

5 Conclusions and future works

In this project a system for connecting a large-scale WSN to CERN storage and computing system has been evaluated. The workflow, which comprises data collection, storage, online, and offline analysis, has been implemented and validated. The first part of the project focused on the development of an infrastructure enabling large-scale WSN emulation, supporting the storage of large amount of data on scalable systems such as EOS and HDFS. The infrastructure is capable of supporting a complete control loop, including data collection, online analysis and feedback in the environment. The second part regarded the study of data analysis pattern for data coming from large-scale WSNs, with a focus on the offline computation of ITS-related data. The project demonstrated the effectiveness of the usage of tools such as Hadoop Map-reduce for the offline analysis of sensor-related data. Anyway further studies are needed to better evaluate the feasibility of quasi-online computation on sensor data. Specific mechanisms should be adopted in order to provide guarantees in terms of timeliness to the system. Moreover a direct connection between the traffic simulator (SUMO) and a WSN simulator should be considered. An interface should be in charge of synchronization and information exchange between the two simulators. The resulting simulator would allow to evaluate the impact of a large-scale cooperative intelligent transport system more realistically.

In the system evaluated the nodes are in charge of simply sensing the environment and forwarding the data to the distributed storage system. Anyway in WSN the largest source of energy consumption is due to wireless transmission. Since motes have local computational power and support remote reprogramming, a system distributing the data processing inside the network should be considered to enhance the energy efficiency of the system. If we consider every node of the WSN as having a local collection of measurements of type key-value the WSN can be seen as a distributed file system and the map-reduce computing model in which every node of the network is a worker node, could be applied. Anyway porting the map-reduce runtime environment in a resource-constrained world is challenging. Besides, the model should be adapted to a lossy network, in which nodes may fail or disappear frequently.

6 References

1. Mulligan, G., *The 6LoWPAN architecture*, in *Proceedings of the 4th workshop on Embedded networked sensors*2007, ACM: Cork, Ireland. p. 78-82.
2. Shelby, Z., *Embedded web services*. Wireless Communications, IEEE, 2010. **17**(6): p. 52-57.
3. Jardak, C., et al. *Parallel processing of data from very large-scale wireless sensor networks*. in *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*. 2010. ACM.
4. Liu, J., et al. *An open, flexible and multilevel data storing and processing platform for very large scale sensor network*. in *Advanced Communication Technology (ICACT), 2012 14th International Conference on*. 2012. IEEE.
5. Sobeih, A., et al., *J-Sim: a simulation and emulation environment for wireless sensor networks*. Wireless Communications, IEEE, 2006. **13**(4): p. 104-119.
6. Abrahamsson, T., *Estimation of origin-destination matrices using traffic counts–a literature survey*. IIASA Interim Report IR-98-021/May, 1998. **27**: p. 76.