Coordinating Data Analysis & Management in Multi-Layered Clouds

Ioan Petri[1], Javier Diaz-Montes[2], Omer Rana[1],
Yacine Rezgui[1], Manish Parashar[2], and Luiz F. Bittencourt[3]

[1] School of Computer Science & Informatics, Cardiff University, UK
petrii@cardiff.ac.uk
[2] Rutgers Discovery Informatics Institute, Rutgers University, USA
javidiaz@rdi2.rutgers.edu
[3] Institute of Computing, University of Campinas, Brazil
bit@ic.unicamp.br

Abstract. We introduce an architecture for undertaking data processing across multiple layers of a distributed computing infrastructure, composed of edge devices (making use of Internet-of-Things (IoT) based protocols), intermediate gateway nodes and large scale data centres. In this way, data processing that is intended to be carried out in the data centre can be pushed to the edges of the network – enabling more efficient use of data centre and in-network resources. We suggest the need for specialist data analysis and management algorithms that are *resource-aware*, and are able to split computation across these different layers. We propose a coordination mechanism that is able to combine different types of data processing capability, such as *in-transit* and *in-situ*. An application scenario is used to illustrate the concepts, subsequently evaluated through a multi-site deployment.

Key words: Distributed Clouds, Cloud Computing, Data Analytics, CometCloud

1 Introduction

With increasing deployment of sensors to measure physical phenomenon, there has been interest in recent years in standardising sensor device types, communication protocols and their data exchange formats. This has resulted in various attempts to define interoperability specifications for Internet-of-Things (IoT) – which according to NIST (as part of their "Cyber-Physical Systems" programme), is "a global network infrastructure, linking physical and virtual objects through the exploitation of data capture and communication capabilities" [7]. Recent efforts at the IEEE, such as P2413 [8], also attempt to define an architectural framework for IoT, indicating that "most current standardization activities are confined to very specific verticals and represent islands of disjointed and often redundant development" in the IoT area. The P2413 architectural framework "will promote cross-domain interaction, aid system interoperability and functional compatibility."

Understanding how data collected from IoT-based devices can be channelled for analysis into a Cloud-based system remains an important research area. Although significant work exists in offloading computation from mobile devices to Cloud-based systems [15], better understanding how to divide data processing across IoT-based devices (which can have on-board computational capability, e.g. through the use of Arduino or Raspberry Pi-based deployments) and Cloud-based infrastructure has received limited attention. Recent efforts in creating an open source "IoTCloud" (providing sensors-as-a-service) [13] and middleware oriented efforts in European Open IoT project [14] indicate significant interest in this area from the academic community. In the same context, HTTP/REST-based APIs, such as Xively (previously Pachube) [9], Open Sen.se [10], Think Speak [11] and Pacific Control Gateways [12], indicate strong commercial interest, in applications ranging from smart cities to intelligent homes.

We describe how IoT-based devices and Clouds can be integrated using a multi-layered architecture. The basis of this comes from the observation that not all data collected through IoT-based devices needs to be channeled to a Cloud platform. Current practice is primarily to stream or batch-collect all data from devices and carry out subsequent analysis via a Cloud platform. However, this is often unnecessary (and may involve costly data transfers across networks with varying characteristics, in terms of bandwidth, cost of access, availability and latency) as only a subset of the data may actually contribute to the analysis being performed. Similarly, partial data processing may be carried out directly on the devices or through intermediate collection gateways (that are situated between the devices and the Cloud platform). We therefore propose a coordination model where the Cloud platform, intermediate gateway devices and IoT-based devices need to work collectively to carry out data processing. Such coordination takes account of constraints of the devices (e.g. limited network and battery power) and optimisation criteria of Cloud platforms (e.g. improve throughput and reduce execution time). Section 2 describes the overall systems architecture, and the various layers involved from data collection to processing. Section 3 outlines a coordination mechanism that enables the data processing to be split across multiple layers, followed by an example scenario in section 4 and evaluation in section 5. We conclude with a general discussion in section 6.

2 Approach and Architecture

The distributed system architecture presented in Figure 1 consists of three main layers: (i) L3: data capture point, (ii) L2: gateway nodes (in practice, multiple levels may exist) and (iii) L1: data centre/computing cluster. At L3 various data capture devices, such as sensors, mobile phones (with human input) record values based on an observed phenomena. These devices capture data with a pre-defined frequency (often dictated by the rate of change of the phenomenon being observed), depending on the capacity of the device to record/collect data and also based on specific system requirements that need to be satisfied. L2 involves the use of multiple gateways, which may be realised in practice using network switches and routers, fronted by OpenFlow software (for instance) or network processor-based hardware, which enables such network components to be remotely managed. However, such gateways may also be computational devices that aggregate data from a variety of L3 sensors. Finally, L1 contains more complex computing clusters, where greater computational and storage capability is made available to application users, enabling more complex, generally long running, simulations to be carried out on the data.

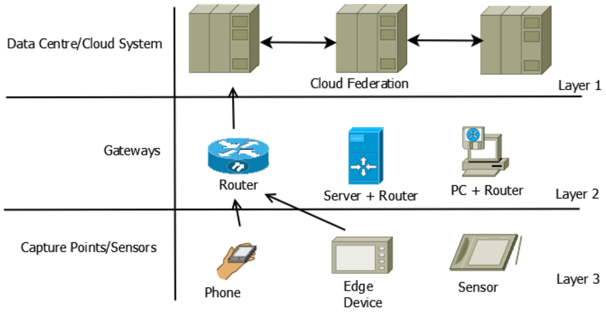


Fig. 1: Conceptual (system) architecture

Devices at L2 can carry out various operations on the raw data collected at L3 – such as performing stream operations (average, min, max, filtering, aggregation etc) on a time/sample window of data, carrying out encryption of an incoming data stream or a variety of other data encoding/transcoding operations before forwarding this data for subsequent analysis to L1. Hence, devices at L2 retrieve data but can also perform some preliminary analysis. We envision a distributed Cloud to be composed of devices at all of these levels, and with a need to coordinate work across these levels to achieve particular data analysis and performance targets. Each level also has its own objective function which influences the types of operations carried out. For instance, L3 generally consists of resource constrained devices (i.e. limited battery power, network range, etc) which must carry out operations in the context of these constraints. Similarly, L2 consists of various network elements or computing nodes that need to be shared across multiple concurrent data flows, requiring any analysis to be constrained by the number of flows and time constraints in carrying out the filtering/pre-analysis. Operations at L1 are based on pre-agreed targets between a client and a data centre provider, such as throughput, response time, cost, etc. Understanding how an application hosted on a Cloud at L1 can interact and coordinate with L3 and L2 (either directly or via L2) is a key research challenge in such systems, particularly for real time, streaming applications.

Distributing analysis of data across these different levels can improve the overall system performance and reduce the load on L1 infrastructure and the core network. We also observe that raw data collected at L3 may not necessarily be needed (in its entirety) at L1 – and aggregate operations on the data (e.g. average, summation/fusion, etc) may be enough for the type of analysis required at

L1. It is therefore not necessary to transfer all the collected data to the data centre (as often undertaken currently – even with the availability of recent systems such as Amazon Kinesis [3], Google BigQuery or Apache Flink for streaming data), wasting network bandwidth and buffer/ storage space at levels L2 and L1. We identify the following classes of data analysis: In-situ analysis: is carried out at L1, on a pre-agreed number of computing resources. This is the current mode of operation with many Cloud systems – whereby data is aggregated at a central site prior to analysis. In streaming systems (e.g. Amazon Kinesis), data sharding is carried out prior to transfer of this to Amazon VM instances hosted at a particular data centre. This approach can have major disadvantages in terms of load and response time, as collection at a central server can be time consuming (and sometimes not necessary). Data-drop analysis: After data values are collected by edge devices, and sent over the network, the actual data analysis process starts when the data sets are dropped into a specific folder. Data-drop analysis is the ability to trigger on-demand analysis making use of elastic computing resources available at L1 (at the data centre). A key challenge in this type of analysis is to predict the number of computing resources needed (as data is dynamically made available) based on heuristics or prior execution history. This type of analysis can suffer from the same QoS limitations as In-situ analysis, as it still requires data to be shipped over the network from L3 to L1 infrastructure. In-transit data analysis: Identifies the type of distributed analysis carried out at L3 and (more generally) L2. In-transit analysis makes use of capability available in software defined networks to undertake partial analysis while the data is in transit from source (L3) to the data processing engine (generally L1). This approach can significantly improve overall analysis time (and limit use of resources at L1), as pre-analysis can help identify what needs to be carried out at L1. In-transit analysis therefore makes more effective use of computing capability available at L2.

3 Multi-Layered Coordination

A coordination mechanism should enable selection of the type of analysis (as discussed in section 2) to be carried out at a particular level (sensor, gateway or data centre). The coordination mechanism also needs to take account of application specific constraints (hosted at the data centre). We consider an overall quality of service metric – associated with an application – to be composed of three individual layer metrics:

$$QoS_T = QoS_T^{L1} \bigoplus QoS_T^{L2} \bigoplus QoS_T^{L3} \tag{1}$$

where QoS_T represents the total quality of service that the system needs to support to meet application requirements, $\bigoplus$ represents the aggregation operator (and may be min or max, depending on the QoS parameter being considered), QoS_T^{L1} represents the quality of service for the clouds/data centre layer, QoS_T^{L2} is the quality of service at the gateway layer and QoS_T^{L3}, the quality of service at the sensors layer, respectively. Each QOS_T^x is influenced by constraints within that layer, for instance:

1. Sensor/device Level (L3): battery power, network coverage, on-board memory available, type of sensing (for a multi-purpose sensor) etc.
2. Gateway Level (L2): data storage, network bandwidth, operations supported (influenced by window or sample size for incoming data), number of concurrent streams processed, sample rate, etc.
3. Cloud/data centre Level (L1): throughput, response time, execution time per application, number of concurrent applications (for multi-tenancy), cost of access etc.

The coordination mechanism is, given particular constraints, attempting to improve QoS_T over a given time frame. Such a mechanism could be realised in practice by using a controller at each layer in figure 1, which aims to learn potential control actions. For instance:

- QoS_T – minimise response time for a particular application job running at L3, which could be achieved by: (i) reduce the size of data transfered from L2 to L1, (ii) reduce sampling interval at L3, (iii) increase number of VMs at L1. The same outcome could be realised by: (i) pre-process data at L2 and L3; (ii) increase number of VMs at L1.
- QoS_T – increase accuracy of analysis at a particular budget, which would lead to: (i) identify number of VMs within budget constraints at L1; (ii) identify data size needed from L2 to maximise VM utilisation at L1; (ii) vary sampling rate at L3 based on network capability between L2, L3 and L1, L2, etc.

Each of these application requirements could therefore be expressed as a set of min/max constraints, leading to potential control actions carried out to realise the outcome.

4 Application Scenario

To demonstrate the use of our multi-layered approach, we consider a scenario in the construction/built environments domain focusing on energy flow analysis within a building using EnergyPlus [1]. Consider a user job to be defined as: $[input, obj, deadline]$, where $input$ data is represented as $[IDF, W, [param]]$, where IDF represents the building model to be simulated, W represents the weather file required for the simulation, $[param]$ defines the parameter ranges associated with the IDF file that need to be optimised $[param] = [r_i \rightarrow (x_m, x_n)]$. We consider an optimisation $objective$: $[outVarName, min/max]$, defining the name of the output variable to be optimised $outVarName$ and the target of the optimisation process min/max, min:minimising the $outVarName$ or max:maximising the $outVarName.Deadline$ is a parameter defining the time interval associated with the job submitted. We make use of CometCloud [2] as our Cloud platform.

A job contains a set of tasks $N = \{t_1, t_2, t_3, ..., t_n\}$ mapped into tuples within the CometCloud tuple-space. Each task t_i is characterised by two parameters $t_i \rightarrow [ID, data]$ with the first parameter being a task identifier and $data$ represents one set of results (given a particular parameter range). The simulation output represents an optimum setpoint to be implemented within the building using suitable actuation mechanisms. We use sensor data from the $SportE^2$ project pilot called FIDIA[1] and EMTE[2] – both public sports buildings in Rome, Italy and Bilbao, Spain, respectively.

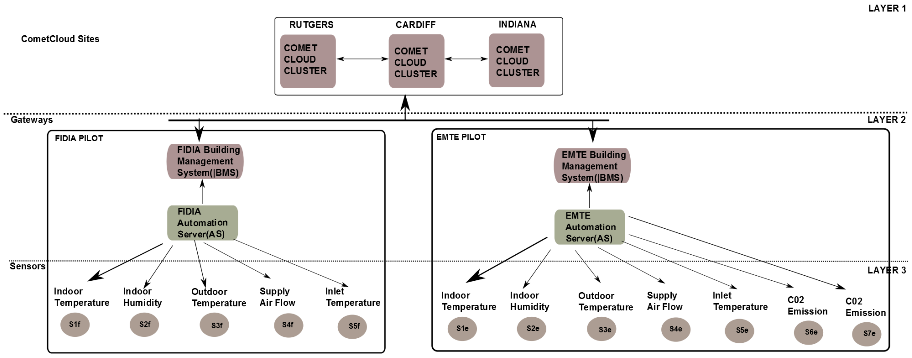


Fig. 2: Application Scenario

Based on the layers in figure 1, at layer 3, each sensor in our pilot can either connect via a gateway or directly to an Automation Server (AS). Sensors are generally battery powered meters which can measure: (i) indoor temperature and air temperature inlet – via a Modbus IP protocol connected to the AS gateway; (ii) water temperature using a regular I/O operation to the AS gateway; (iii) indoor humidity – communicating to the AS gateway; (iv) supplied air flow rate measured with a velocity sensor and using I/O operations to the AS gateway. Additional details of the sensors can be found in [5].

4.1 Level 2: Building Management System & Automation Server

There are two distinct gateways: (i) Building (Energy) Management System (BMS) and (ii) Automation Server (AS) – each acting as an autonomous system. The BMS gateway is a server machine that controls the activities and spaces within the building. In addition to controlling the building's internal environment, BMS systems are sometimes linked to access control (turnstiles and access doors controlling who is allowed access to the building) or other security systems such as closed-circuit television (CCTV) and motion detectors. The AS

[1] http://www.asfidia.it

[2] http://www.emtesport.com/

gateway is a hardware-based server that is factory programmed with StruxureWare Building Operation software (for instance). In small installations, an AS may act as a stand-alone server, mounted with its I/O modules. In medium and large installations, functionality is distributed over multiple Automation Servers (ASs) that communicate over TCP/IP. An AS can deliver data directly to an analysis system or to other servers throughout the site. The AS can run multiple control programs, manage local I/O, alarms, and users, handle scheduling and logging, and communicate using a variety of protocols.

4.2 Level 1 : CometCloud Sites Level

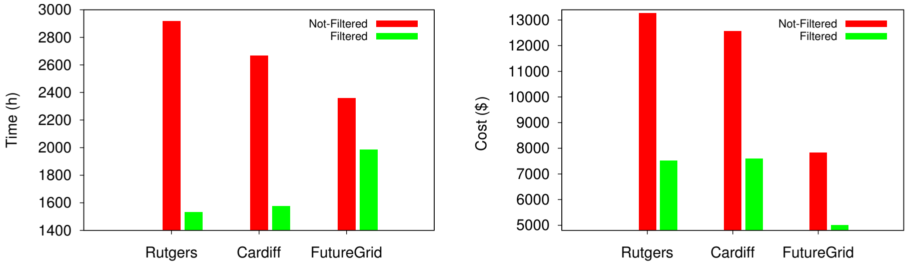
At this level, we have a CometCloud-based federation of resources [4, 6], where each site has access to a set of heterogeneous and dynamic resources, such as public/private clouds, supercomputers, etc. Each site decides on the type computation it runs, as well as the prices based on various decision functions that include factors such as availability of resources, computational cost, etc. This federation is dynamically created at runtime where sites can join or leave at any given time. Notably, this requires a minimal configuration at each site that amounts to specifying the available resources and access credentials. We consider three sites in this scenario: at Cardiff , Rutgers, and Indiana Universities. Each site provides the following resources: Cardiff: has a virtualized cluster-based infrastructure with 12 dedicated physical machines. Each machine has 12 CPU cores at 3.2 GHz. Each virtual machine (VM) uses one core with 1GB of memory. The networking infrastructure is 1Gbps Ethernet with a measured latency of 0.706 ms on average. Rutgers: has a cluster-based infrastructure with 32 nodes. Each node has 8 CPU cores at 2.6 GHz, 24 GB memory, and 1Gbps Ethernet connection. The measured latency on the network is 0.227 ms on average. FutureGrid: make use of an OpenStack cloud deployment at Indiana University. We have used instances of type medium, where each instance has 2 cores and 4 GB of memory. The measured latency of the cloud virtual network is 0.706 ms on average. Based on the use of CometCloud [2], each site has a master process that receives task requests from other sites, and is able to forward requests to other sites. Each site also has multiple worker processes that carry out actual task executions on locally available resources. In this application scenario, each worker is responsible for executing an EnergyPlus [1] simulation with a different input parameter range.

5 Evaluation

In our experiments we use two different configurations – (a) Cloud level analysis where the tasks are executed exclusively at the cloud level with two configurations: (i) single cloud context where all the tasks have to be processed locally (within the local site) and (ii) federated cloud context where the sites have the option of outsourcing tasks to remote sites and (b) distributed Cloud analysis where the tasks are executed on a multi-cloud infrastructure – i.e. making use of gateway nodes alongside the CometCloud deployment.

5.1 Distributed clouds analysis

In use case (b) above, information collected by sensors will be processed in-transit in the Gateway layer (i.e. L2) to filter out various sensor information (e.g. values out of range or certain combination of parameters that cannot lead to reasonable results) and then create jobs to be sent to sites at L1. An example of filtering at gateway layer is the average of the temperature values that are recorded at various zones of a building(north, south, etc.). Often, although significant to record these value across all the zones of a building for maintaining the optimization accuracy, it is useful to use an average of these temperature values not only to reduce the total number of EnergyPlus simulations at cloud layer but also to have a more comprehensive view of the overall building behaviours. Through such pre-filtering, we are able to reduce the computational requirement at L1. We explore the benefit of in-transit data analysis by comparing differences between these two scenarios in terms of the total cost for each site to compute all jobs, the overall time spent and number of jobs completed successfully.



(a) Execution time

(b) Execution Cost

Fig. 3: Summary of experimental results for use cases with filter and without filter – 3a shows total execution time and 3b the total cost spent on computing all jobs

We consider sensors in two geographically distributed buildings that are collecting information about the status of the building and sending this information to gateways (at L2). In order to better explore the behavior of in-transit data analysis and task distribution, we emulate the execution of the tasks and use a Poisson distribution to periodically generate sensor collected information every 100 minutes. A job is generated after the gateway has received data from sensors. One job will produce multiple EnergyPlus computation sub-tasks. All three sites bid for computing those jobs based on their available resources and the number of sub-tasks they can finish before the deadline. No single winner

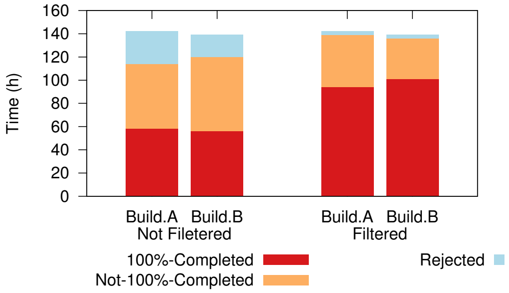


Fig. 4: Shows the number of rejected, fully & partially completed jobs

will get all the sub-tasks. Instead, these sub-tasks will be distributed to all bidder sites based on their estimation of job completion deadline. Each site will get $bidNum/allSitesTotalBidNum$ sub-tasks to compute. We allocate two local and two external workers to each site. Once a site consumes a list of sub-tasks, these tasks will be sent to workers to finish computation. *No filtering:* sensor outputs include four types of parameters which then gives a combination of 16 EnergyPlus sub-tasks per job. Each EnergyPlus sub-task takes 30 minutes to compute on all three sites. In Figure 4, we observe that due to resource limitation, some jobs are rejected because these sub-tasks cannot be completed before the deadline by these three sites. Among the accepted jobs, *Not 100% Completed* jobs are those whose sub-tasks were not completed within the given deadline. This may occur due to the availability of limited network bandwdith, scheduling constraints, placing multiple bids without knowing results of previous auctions, etc. Conversely, *100% Completed* jobs have all sub-tasks completed on time. *With filtering:* After analysis of sensor data, we can filter out the data received from sensors, with the number of sub-tasks for each job being reduced to eight.

In order to better compare this use case with the previous one, in this experiment we assume that jobs are generated following the same process as the previous experiment. This means the total number of jobs are the same, only the number of sub-tasks per job is smaller. Figure 4 shows the number of rejected jobs, reduced after filtering. From Figure 3a, the total execution time for completing all jobs also decreases, along with reduction in total execution cost in Figure 3b. This is mainly due to the number of sub-tasks per job, being reduced from 16 to 8 after filtering. This demonstrates the benefit of undertaking in-transit analysis via L2.

6 Conclusions

We demonstrate the benefit of supporting multi-layered Clouds, whereby computation can be distributed across multiple layers of a data capture and computation infrastructure. Each layer offers specific capabilities and constraints, and we discuss the need to support resource-aware computation to be combined across multiple layers. With increasing availability of devices that support standardised access protocols, as proposed in recent developments towards specifying IoT standards, we discuss the benefit of combining these devices with resources at federated data centres, enabling data analysis to be split across multiple layers based on a coordination mechanism. A building energy simulation scenario is used to illustrate the concept. Edge devices can range in capability – from sensing devices within limited battery power and on-board memory, to large scale scientific instruments that have significant computational capability. The current pratice of migrating all data to a centralised data centre for analysis may be inefficient in how the network capacity and data centre capability is utlised. In the experiments section we demonstrate that multi-cloud analysis and coordination can reduce the total execution time with tasks and can greatly lead to reducing the execution cost.

References

1. Fumo, N.; Mago, P.; Luck, R. "Methodology to Estimate Building Energy Consumption Using EnergyPlus Benchmark Models.", Energy and Buildings 42:12, pp. 2331-2337, 2010. Elsevier.
2. CometCloud Project. `http://nsfcac.rutgers.edu/CometCloud/`. Last accessed: July 2015.
3. Amazon Kinesis. Available at: `http://aws.amazon.com/kinesis/`. Last accessed: March 2014.
4. I. Petri, T. Beach, M. Zou, and et. al. Exploring models and mechanisms for exchanging resources in a federated cloud. In Intl. Conf. on cloud engineering (IC2E 2014),EEE Computer Society, pp. 215-224, Boston, 2013, ISBN: 978-1-4799-3766-0.
5. Ioan Petri, Omer Rana, Rezgui Yacine, Li Haijiang, Beach Tom, Zou Mengsong, Diaz-Montes Javier, Parashar Manish, "Cloud Supported Building Data Analytics", 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), 2014 , pp.641-650, 26-29 May 2014, doi: 10.1109/CCGrid.2014.29
6. J. Diaz-Montes, Y. Xie, I. Rodero, J. Zola, B. Ganapathysubramanian, and M. Parashar. Exploring the use of elastic resource federations for enabling large-scale scientific workflows. In Proc. of Workshop on Many-Task Computing on Clouds, Grids, and Supercomputers (MTAGS), pages 1–10, 2013.
7. National Institute of Standards and Technology (NIST), "Cyber Physical Systems", available at: `http://www.nist.gov/cps/`. Last accessed: July 2015.
8. IEEE, "P2413 IoT Architectural Framework", available at: `https://standards.ieee.org/develop/project/2413.html`. Last accessed: July 2015.
9. Xively, `http://xively.com`. Last accessed: July 2015.
10. Open Sen.se/Internet of Everything, `http://open.sen.se/`. Last accessed: July 2015.
11. Think Speak, `https://thingspeak.com/`. Last accessed: July 2015.
12. Pacific Controls Gateway, `http://pacificcontrols.net/products/galaxy.html`. Last accessed: July 2015.
13. IoT Cloud, `http://sites.google.com/site/opensourceiotcloud/`. Last accessed: July 2015.
14. Open IoT, `http://www.openiot.eu/`. Last accessed: July 2015.
15. N. Fernando, S. W. Loke, and W. Rahayu. "Mobile cloud computing". Future Gener. Comput. Syst. 29, 1 (January 2013), 84-106. Elsevier Science.