EdgeMQ: Towards a Message Queuing Processing System for Cloud-Edge Computing

(Use Cases on Water and Forest Monitoring)

Ion Dorinel Filip*, Bogdan Ghita*, Florin Pop*† George-Valentin Iordache*, Catalin Negru*, and Ciprian Dobre*†
*Computer Science Department, University Politehnica of Bucharest, Romania
†National Institute for Research and Development in Informatics (ICI), Bucharest, Romania

***Abstract*—With the increase of computational enabled devices spread around us, improving the efficiency of our Cloud-based software solutions frequently led us to use the closest available resources and many real-time processes cannot rely on traditional cloud architectures for all of theirs processing tasks anymore. This paper proposes a Cloud-Edge data processing architecture covering both real-time and batch processing models. We design our solution as a general architecture and propose an implementation schema using RabbitMQ as a queuing engine. Two use-cases are considered: one refers to monitoring a forest and the other consider water monitoring and management. Those scenarios include both batch and real-time processing components and they can be mapped on the proposed architecture. We describe a methodology for performance evaluation and give experimental results for using RabbitMQ in a Set-Top Box environment. Using Docker containers with limited resources, we obtained a message rate of 4k messages per second for 1KB sized messages.**

***Index Terms*—Edge Computing, Cloud Computing, Message Queuing System, Real-Time Processing, Batch Processing**

I. Introduction

Edge computing is a new computing paradigm where computation is performed at the edge of the network on devices with limited storage and computing capabilities, called set-top boxes (STB) [1]. These STBs are easy to be installed, configured and used in different environments, but the problem of data processing using tools deployed in Clouds is still an open research issue.

An important challenge comes from data streams produced by mobile devices and sensors used in environmental monitoring (like water resources, forests), traffic management or video surveillance [2]. It is oriented on how to improve response time for dedicated applications by reducing latency and network traffic. One solution is to process data locally, instead of transferring it to the Cloud, in a synchronized mode with global storage.

It is forecast that by 2018, 1/3 of population will have a mobile device. Also, by 2020, 43 trillion gigabytes of data will be generated at global level [3]. The digital transformation leads to an explosion of data traffic to Cloud data-centers. So, using machines that are one hop away in the network, data traffic can be reduced significantly or it can be distributed optimally.

For distributed applications, with real-time constraints, the Cloud Computing model is not efficient anymore [4]. In order to improve the quality of service for these applications, processing and storage of data needs to be close to the data source, to be aware of context and to deal with complex event handling. Moreover, real-time applications need a response time in the order of milliseconds (25ms to 50ms) [5], which is an important challenge for SLA negotiation. Cloud processing adds a significant latency because of data transfer. The round trip time (RTT) can exceed by far the response time for real-time applications in case of large distances. For instance, RTT between Canberra and Berkley is around 175ms [6].

In this paper we discuss about a novel hybrid system for cloud-edge computing model.

This paper is structured as follows: Section I contains a short introduction related to the edge computing paradigm and message queuing processing. In Section II we present relevant papers related to the field of edge computing. Section III presents our proposed solution. Section IV contains two use-cases and finally in Section V we present the conclusions of this paper and future work.

II. Related-Work

Considering the evolution of Software Defined Networks (SDNs) and Internet of Things (IoT), the problem of increasing the efficiency of using available local or cloud resources frequently asks to reach the closest available computational resources and different solutions aim to increase the level of usage for the less visible resources, including Set-Top boxes and other devices.

A solution for increasing the load balancing in heterogeneous systems is [7] which proposes a new model for scheduling microservices over Cloud-Edge environments and present a microservice oriented scheduling architecture for IoT applications that emphasis the importance of using less general purpose devices. They also offer an evaluation of different scheduling algorithms for this type of scenarios.

A good compendium about Cloud-Edge is [1] which presents the problems that should be taken into account for a general Cloud-Edge system and provides insights for different services. The set of discussed problems includes programmability, naming, data abstraction, service management and security and it gets applied for multiple use-cases like cloud-offloading, video-analytics and smart homes / cities [8].

In [9] we find the description of a cloud-based message broker system called FogMQ that aims to overcome the limitations of conventional systems. It enables autonomous discovery, self-deployment, and on-line migration of message brokers across heterogeneous cloud platforms.

RabbitMQ [10] is a message broker software that allows distributed applications to share data using common protocols or to queue jobs for processing by distributed workers. As an Open Source solution it is very flexible and convenient to use.

Water and forest monitoring are subjects of great importance that might take advantages from using computational resources with different (far-close) locality.

Important aspects for designing a water quality monitoring system are presented in [11], which reviews current remote real-time monitoring application from different points of view, some of them being concentrated to link between their design and advancements in telemetry and computing technologies.

III. Proposed Solution

A. Required Services

In this section we describe the main goals in designing our Cloud-Edge architecture which should offer multiple services for our use-cases. These services can be split in multiple categories:

- **Queuing services** - the services that allow a user-agent to push, check and manage offloaded tasks. This kind of services can be described as a system that offers a reliable way of managing message-passing. Sometimes they can be extended to offer messaging to processing linking features like RPC;
- **Networking services** - the features that enable a device to communicate with and register itself for discovering, managing and roaming purposes;
- **Task Scheduling** - refers to finding a task $\rightarrow$ machine assignation that optimizes different goal functions (e.g. power consumption, latency, cost). If we consider a dynamic physical architecture this feature also includes a provisioning part that deals with the deployment of different resources on demand;
- **Network Storage** - includes the hardware (storage devices) and software components (protocols and drivers) that allow an user-agent to remotely store and retrieve data to/from a network attached device;
- **Remote Processing Services** - the services that manage execution of the offloaded tasks.

Considering different problems from the categories above, we aim to provide a solution that is able to: (i) connect different devices to the network; (ii) offer task offloading support based on an hierarchical queuing model; (iii) reduce networking latency for mobile devices using an access network based on set-top boxes; (iv) optimize processing costs and performance using an hierarchical queuing model that uses a mix of local, Edge (based on set-top-boxes) and Cloud computational resources.

As required by our use-case scenarios we consider the following architecture for a collection of heterogeneous user-agents and a wide variety of applications that consist in both batch and real-time processing tasks. A batch job is a type of task that might require a significant amount of time and/or resources in order to be performed and it is not required to be finished immediately. In contrast, the result of a real-time job is expected to be received in a very short amount of time in order to be relevant.

B. Architecture

In Figure 1 we present our proposed architecture that is organized in 3 hierarchical levels which provide services to applications.

In the left side of Figure 1, we present the components of the solution, while the right side presents the corresponding services for each of these levels. For the application level, the figure shows some important components of the application (on the left side) and some examples of applications (on the right side).

The first level from the bottom is the **User-Agent level** which consists in user-agents (e.g. devices, terminal, external services) that are connected to the processing architecture. They have both generic and use-case specific capabilities.

For example, considering the use-case of creating an architecture for smart-robots, user-agents are single board computers that connect a set of sensors and actuators to an embedded platform. Even though they might be considered IoT devices, we construct our solution considering that they might be directly connected to the Internet, but get connected to the rest of the architecture by local communication to the next level.

The second level of our architecture is represented by the **Edge-Processing Level**. This level contains a heterogeneous set of set-top boxes. They are computational enabled devices that are likely to be connected to the Internet and offer a considerable amount of processing power. This level serves as network access level for devices and offers task offloading (remote processing power) for the connected devices. Multiple queues are hosted by this level. Depending on the kind of task that it gets or on the scheduling decision, the software on this level may decide to send the tasks to the public cloud on the next level or process them on the Edge.

The third level is the one of Public/Traditional Cloud that is available through the Internet. Disregarding the networking role, the **Cloud Level** mainly differentiates itself from the Edge level by communication latency. We expect that the cost of communication to be more important when trying to reach remote resources, but we aim to take advantage of more powerful hardware that may be available on public clouds.

As presented on the right-side of Figure 1, each of these 3 levels offers certain services/features. Some of them are common between different levels, while others are less likely to be offered by some specific levels. We map those 2 types of resources to the kinds of tasks that the user-agents can generate. We expect the real-time processing task to be more

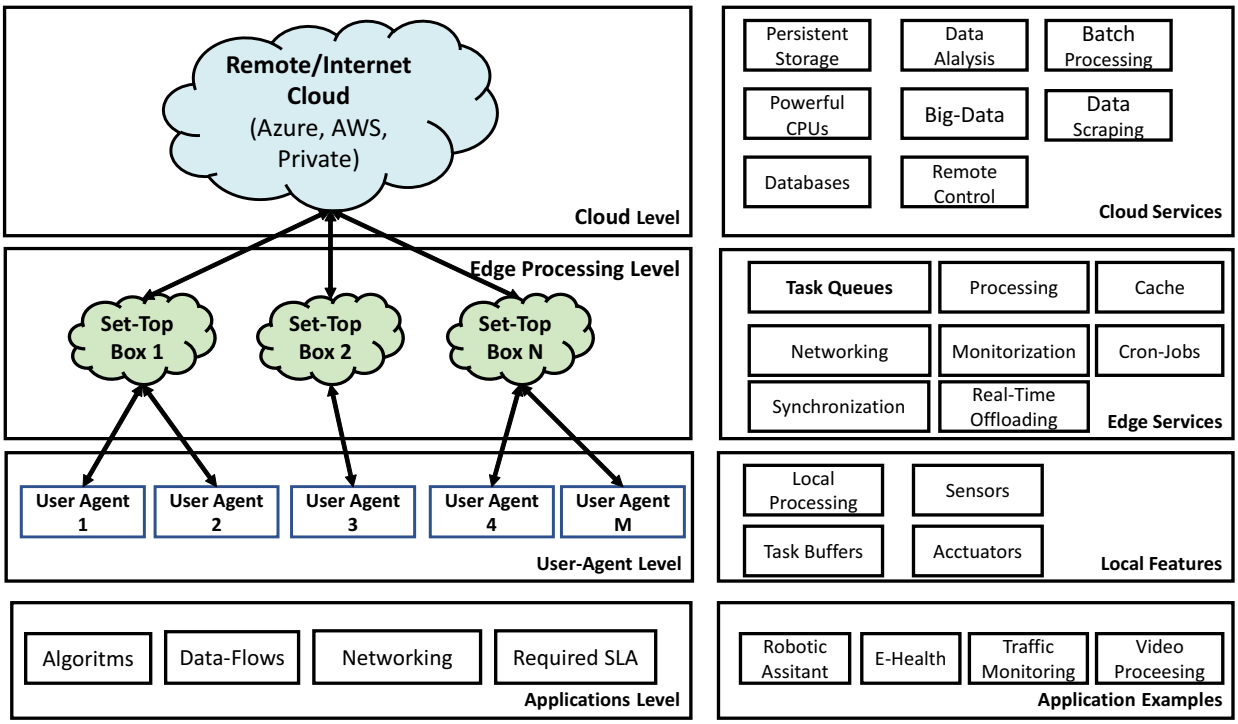

Fig. 1. Generic real-time scheduling system.

frequently scheduled on the Fog/Edge levels while the batch ones are more likely to better fit on the public cloud.

C. Service Level Agreement (SLA)

Edge infrastructures connect to different devices such as data acquisition systems, wireless sensors or smart meters producing data in different formats and scales and most of them refers to large volume of data that needs to be processed. For example, it is necessary to transmit and analyze large amounts of data efficiently to respond in real-time to the devices connected to the edge computing platform.

A Service Level Agreement (SLA) is represented by a contract between a customer and a provider that is designed and it is functional when the customer requests and receives from a given provider as a service of an operating edge computing infrastructure to use or analyze the results provided by some devices that connect to the edge computing infrastructure. Based on this definition we can define a SLA in edge computing as a common contract between the customer and the provider that has the purpose of verifying the functional aspects (or parameters) of the communication between the edge computing infrastructure and the devices connected to it. The parameters of the SLA specify various characteristics of the service with the purpose of assuring customer and provider satisfaction.

Subsequently, we will describe the life-cycle of a SLA and the parameters that characterize this type of SLA.

1) SLA Life-cycle: We can consider a SLA as being a contract between the customer and the provider that defines the functioning (both in terms of measuring and monitoring) of the edge computing infrastructure. We can adapt the four steps from [12], [13] to the case of a SLA contract for an edge computing infrastructure as follows:

- SLA design. In this step, the service customers describe the requirements they have in terms of SLA contracts characteristics. Additionally, edge computing infrastructure service providers offer definitions of the SLA contracts parameters.
- SLA selection. In this step, based on existing SLA offerings by the edge computing infrastructure providers, the edge computing infrastructure customer chooses the SLA(s) that is (are) the most appropriate to the service providers in terms of their characteristics. Additionally, a negotiation between the service provider(s) and the service customer(s) takes place with the purpose of having a unique SLA.
- SLA monitoring. Now the service becomes operational (it is started and provided to the customer). In this step the edge computing infrastructure customer monitors, measures and validates the edge computing infrastructure parameters.
- SLA termination. In this step, one of the parts of the contract decides to terminate the agreement because it expires or it was violated by one of the parts [14].

2) SLA parameters: When referring to SLA between the edge computing infrastructure customers and the edge computing infrastructure providers, several parameters need to be considered (see Table I).

In Table I [15] we define various SLA parameters that

TABLE I
SLA PARAMETERS THAT MEASURE THE PERFORMANCE OF AN EDGE COMPUTING INFRASTRUCTURE SERVICE.

No	Name	Unit
1	Service availability	Time
2	(Maximum) down-time	Hours
3	Edge computing infrastructure failure rate	Number
4	Periods of operation	Time
5	Latency times	ms
6	Accessibility in case of problems	Yes/No
7	Number and types of nodes	Number and type

are very important when discussing about the contract that characterizes the functionality of an edge computing infrastructure both from the perspective of a provider and from the perspective of a customer. Next, each parameter of the edge computing infrastructure that belongs to the SLA is presented.

One very important parameter when considering an edge computing infrastructure service is the **service availability**. In an environment prone to failures like an edge computing infrastructure, different incidents might occur such as packet loss, communication delay, very volatile bandwidth and so on. During the malfunctioning of the edge computing infrastructure service we can consider the service unavailable to the customer.

Another functional parameter when referring to an edge computing infrastructure is the **maximum down-time**. A down-time parameter is defined as planned or unplanned. The planned down-time is defined as a period when usual maintenance operations take place. In the case of an unplanned down-time, several situations need to be considered such as when the service is stopped due to the incorrect functioning of the edge computing infrastructure because of system failures (hardware failures) or of the communication failures (network failures). These failures can be detected by using predictive monitoring.

Edge computing infrastructure failure rate parameter defines the malfunctioning rate of the edge computing infrastructure and is defined by the number of failures per unit of time.

The periods of operation parameter defines the operational period such as during a day, a month or even a year.

The latency times parameter represents a period of the edge computing infrastructure defined by the time it takes a bit of data to travel from a **source node** to **destination** or **sink** [16] node.

The accessibility parameter has two possible values: "Yes" or "No". A "Yes" value denotes the fact that the provider allows the customer to access and modify the edge computing infrastructure. The "No" value denotes the fact that the customer doesn't have access to the edge computing infrastructure.

The number and type of nodes parameters defines the type and the number of monitoring nodes that were specified in the SLA contract.

IV. EXPERIMENTAL EVALUATION

A. Implementation proposal

In this section we describe a proof of concept implementation proposal for our model, based on RabbitMQ as a queuing engine for task processing. Figure 2 shows the high level architecture schema for our system.

RabbitMQ serves as the main communication interface between the three levels of our architecture. User-agents generate and offload tasks by publishing messages to the queues in the Edge Processing Level. Set-top boxes consume tasks from these queues and decide whether to process them or further offload to the Cloud by publishing messages to the queues in the Cloud Level. Finally, processing units in the Cloud consume the messages from the queues and process the tasks.

Our points of interest are the Application and Edge Processing levels. As these two levels are split into multiple local area networks, the user-agents offload tasks to the set-top boxes in the same LAN.

Each set-top box runs a RabbitMQ service and therefore it becomes a RabbitMQ node. One or more set-top boxes that are in the same LAN form a RabbitMQ cluster, thus each LAN has its own RabbitMQ cluster. Being in a cluster, the set-top boxes in the same LAN share the same queues, which means that they have the same image of each queue. The Cloud Level it can either be a single RabbitMQ cluster or multiple geographically distributed clusters.

A user-agent is a RabbitMQ client. It connects to any set-top box (RabbitMQ node) in the LAN and offloads tasks the the Edge Processing Level by publishing messages to the appropriate queue. There are two types of queues: batch queues and real-time queues. The user agents publish tasks in the corresponding queue, depending on the type of the task.

In order to be able to process the tasks sent by user agents, each set-top Box also runs a RabbitMQ client which consumes messages from the queues. When a set-top box receives a message from the real-time queue it can either process it or publish it to the Cloud RabbitMQ cluster. The tasks that are published to the Cloud Level queues get processed in the Cloud. If the result needs to get back to the user-agent, it is received by the set-top box which published it and then forwarded to the user-agent.

The user-agents are free to connect to any set-top box. Since queues are shared in the LAN cluster, if one set-top box becomes unavailable, the user-agents can connect to other set-top boxes. The tasks published on the affected set-top box are not lost and get processed by the other nodes, as allowed by RabbitMQ shared queues.

In respect to load balancing, the set-top boxes processing units are balanced in terms of job allocation. This is done by making use of RabbitMQ direct exchanges. The RabbitMQ broker distributes jobs to the set-top boxes consumers in a round robin manner, based on their prefetch count (i.e. here, the number of jobs that the set-top box can process simultaneously). For the RabbitMQ nodes we obtain high availability by mirroring the queues. As nodes are grouped

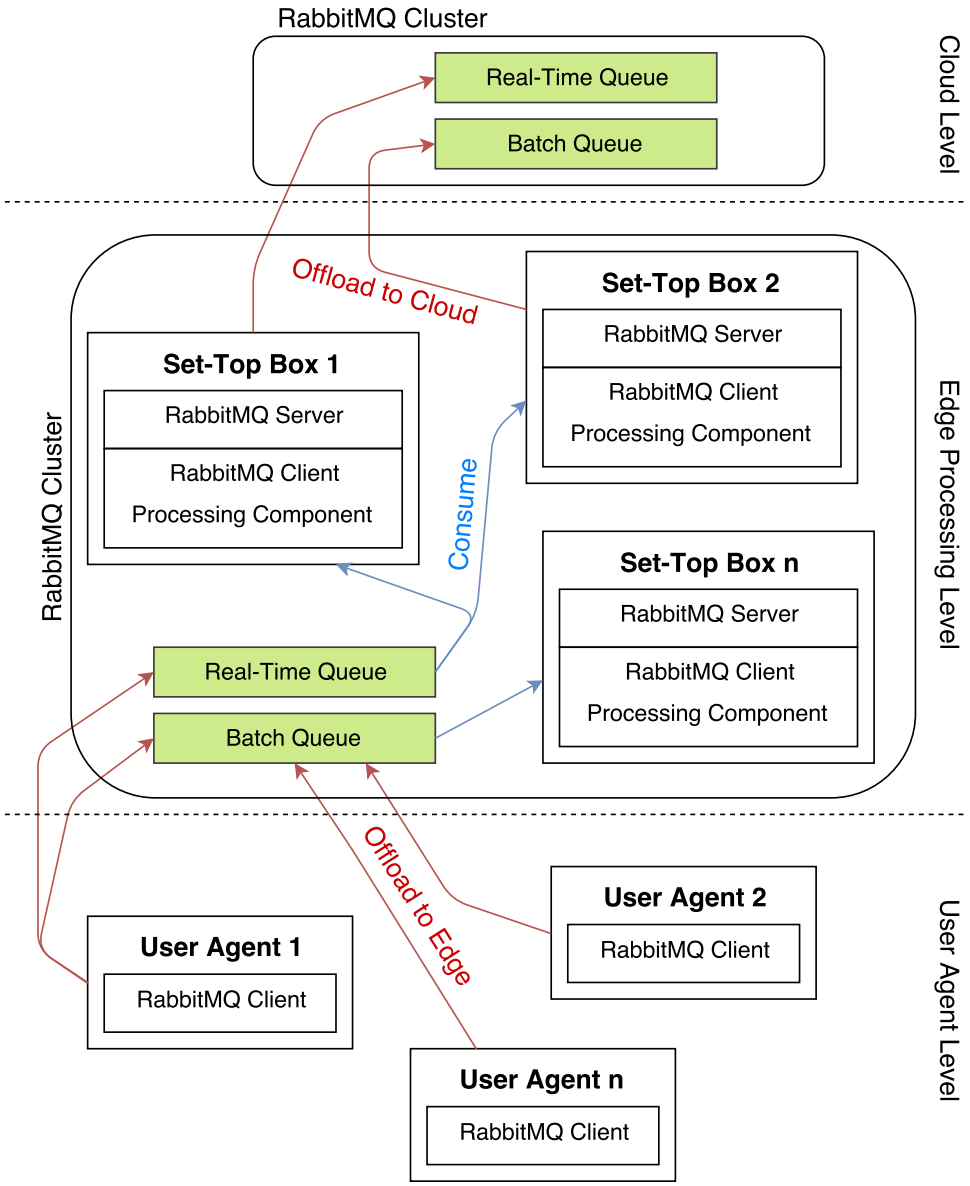

Fig. 2. Architecture schema using RabbitMQ.

in a RabbitMQ cluster, all requests (publish/consume) for a specific queue are directed to the node on which the queue resides. If that node fails, a new master for the queue will be elected.

B. RabbitMQ Evaluation

In order to test how the RabbitMQ system would behave on set-top boxes and what performance levels it could reach, we simulate a scenario close to our architecture.

Considering the hardware capabilities of a set-top box, we deploy a RabbitMQ instance inside a Docker container with resources limited to 1 CPU and 512MB of RAM. This would be the equivalent of a set-top box. The user agents being RabbitMQ consumers and producers, are simulated by Python processes that publish and consume messages, also running in Docker containers. In order to measure the performance of the system we use a monitoring component which gathers system resources statistics using the API provided by Docker and RabbitMQ metrics by using the RabbitMQ management plug-in.

The main metric that we are interested in is the throughput of the RabbitMQ node given the limited resources of a set-top box. The throughput is measured in number of published and delivered messages per unit of time. To determine the limit of the system we create the following scenario: the producers publish an increasing number of messages (each of size 1KB) per second, starting from 10, 20, 30 and so on, while the consumers consume messages as they are delivered

by RabbitMQ.

The results of the simulation are presented in Figure 3. The RabbitMQ system manages to receive and deliver messages at a maximum rate of $\tilde{4}$K messages per second, being limited by the CPU, while memory usage remain rather constant at 40% of its capacity.

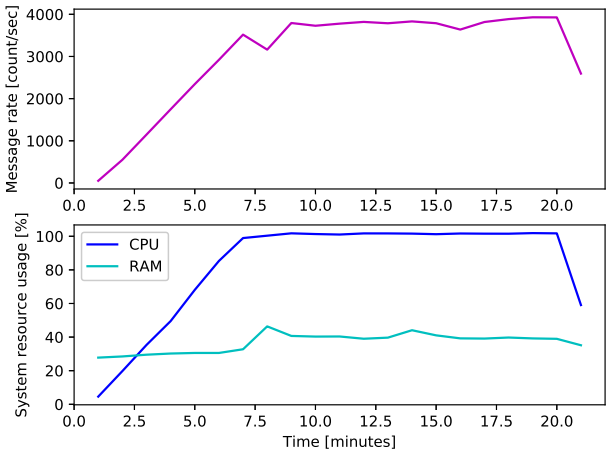

Fig. 3. Experimental results of the proposed system.

In addition to these promising results, we can leverage RabbitMQ for solving issues like scalability, load-balancing, fault tolerance and redundancy by making use of features that ensure reliability: persistent messages, durable queues and exchanges, message acknowledgments and clustering.

V. Use-cases

A. Water monitoring

Monitoring of Water resources such as water distribution systems in smart cities implies large volume of heterogeneous information (e.g. spatial, sensor and multimedia data) with temporal dimension. In order to respond efficiently and to alert possible affected population in case of pollution accidents or to damages in the distribution system, we need to acquire, store, transmit and analyze data with small response times. Also, we need to perform simple data analytics on the data [17].

Usually, the monitoring systems for water distribution consist of a large base of sensors that record and transmit data for storage and processing. Using a traditional centralized processing model where data is processed in a central location is not efficient anymore due to the increased processing times. So, these systems need fast, scalable, reliable and efficient processing model, in order to achieve cost reduction, faster and better decision making in case of accidents or failures in the system.

By using a hybrid model such as the cloud-edge computing model, we can achieve better response times. Data processing tasks can be done locally on set-top boxes. Furthermore, long-term archive data can be stored in the Cloud and can be processed for complex analytics task such as system optimization.

B. Forest monitoring

Another use case that is suitable for cloud-edge computing model is forest monitoring. Monitoring areas covered with woody vegetation is a topic of high relevance both in Romania and at European/global level. The development of efficient and trustworthy services and applications is of great interest, triggering real-time processing challenges, efficient storage and fast retrieval of data.

Romania's forestry fund has an area of 6,529 thousand hectares, representing 27.3% of the country's territory. The total forest footprint is estimated at more than 1.340 million m3. Remote monitoring of forests, using satellite imagery, is an alternative that has been used more often in recent years for ground and air monitoring. Ground monitoring is limited by its discrete / discontinuous character (measurements or observations can be obtained in a small number of points) and the difficulty and cost of reaching hard-to-reach or remote areas [18] [19]. In turn, aerial monitoring (using airplanes, drones), although it allows quasi-continuous data (in the sense that it can take pictures along flight paths, which then have to be assembled in a larger scene), increases the degree of fragmentation of the spatial puzzle and produces an increased asynchrony between fragments / areas. If the superiority of remote monitoring can be questionable at individual scene levels, it is incontestable to process and analyze time series. The ability to analyze sets of large-scale spatial images taken at regular intervals from the same or more sensors, usually at the same time of day, over a long period of time, creates the premises for observing subtle phenomena, which manifests on large areas or in multiple local outbreaks [20] [21].

VI. Conclusions and Future Work

In this paper we presented the design of Cloud-Edge data processing architecture covering both real-time and batch processing scenarios.

Our work also covers the primary design for implementing such a system using RabbitMQ as message passing engine and offers some an objective evaluation of performance limits on running RabbitMQ on a User-Agent to Set-top box environment.

In Section III-C2 we presented some SLA parameters that should be used for early risk analysis of our proposed solution.

As future work we aim to:

- Implement a one or more of the described use-cases using RabbitMQ as a queuing engine for task processing;
- Evaluate different performance metrics against each of the proposes use-case;
- Integrate the idea presented in this paper in one (or more) of our supporting projects.

In the context of the current SI, our paper has added value in giving an overview of the requirements around designing a processing architecture for a Cloud-Edge infrastructure and offers a multiple-paradigm solution for designing a reliable task-scheduling architecture that uses both Edge and Cloud

resources and linking it to an existing technology that mights cover important aspects of the system.

ACKNOWLEDGMENT

The research presented in this paper is supported by the following projects: Data4Water - H2020 Twinning project (H2020-TWINN-2015,CSA-690900), MONROE - Toff project (H2020 No 644399), NETIO-ForestMon (53/05.09.2016, SMIS2014+ 105976), SPERO (PN-III-P2-2.1-SOL-2016-03-0046, 3Sol/2017) and ROBIN (PN-III-P1-1.2-PCCDI-2017-0734).

We would like to thank the reviewers for their time and expertise, constructive comments and valuable insight.

REFERENCES

[1] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, 2016.

[2] C. Negru, F. Pop, M. Mocanu, and V. Cristea, "A unified approach to data modeling and management in big data era," in *Data Science and Big Data Computing*. Springer, 2016, pp. 95–116.

[3] S. Curtis, "Quarter of the world will be using smartphones in 2016," *The Telegraph*, vol. 11, 2014.

[4] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica *et al.*, "A view of cloud computing," *Communications of the ACM*, vol. 53, no. 4, pp. 50–58, 2010.

[5] A. Biondi, M. Di Natale, and G. Buttazzo, "Response-time analysis for real-time tasks in engine control applications," in *Proceedings of the ACM/IEEE Sixth International Conference on Cyber-Physical Systems*. ACM, 2015, pp. 120–129.

[6] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The case for vm-based cloudlets in mobile computing," *IEEE pervasive Computing*, vol. 8, no. 4, 2009.

[7] I. D. Filip, F. Pop, C. Serbanescu, and C. Choi, "Microservices scheduling model over heterogeneous cloud-edge environments as support for iot applications," *IEEE Internet of Things Journal*, vol. PP, no. 99, pp. 1–1, 2018.

[8] R.-I. Ciobanu, C. Negru, F. Pop, C. Dobre, C. X. Mavromoustakis, and G. Mastorakis, "Drop computing: Ad-hoc dynamic collaborative computing," *Future Generation Computer Systems*, 2018.

[9] S. Abdelwahab and B. Hamdaoui, "Fogmq: A message broker system for enabling distributed, internet-scale iot applications over heterogeneous cloud platforms," *arXiv preprint arXiv:1610.00620*, 2016.

[10] A. Videla and J. J. Williams, *RabbitMQ in action: distributed messaging for everyone*. Manning, 2012.

[11] H. B. Glasgow, J. M. Burkholder, R. E. Reed, A. J. Lewitus, and J. E. Kleinman, "Real-time remote monitoring of water quality: a review of current applications, and advancements in sensor, telemetry, and computing technologies," *Journal of Experimental Marine Biology and Ecology*, vol. 300, no. 1-2, pp. 409–448, 2004.

[12] G.-V. Iordache, F. Pop, C. Esposito, and A. Castiglione, "Selection-based scheduling algorithms under service level agreement constraints," in *Control Systems and Computer Science (CSCS), 2017 21st International Conference on*. IEEE, 2017, pp. 134–140.

[13] G. V. Iordache, M. S. Boboila, F. Pop, C. Stratan, and V. Cristea, "A decentralized strategy for genetic scheduling in heterogeneous environments," in *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*. Springer, 2006, pp. 1234–1251.

[14] L. Wu, R. Buyya *et al.*, "Service level agreement (sla) in utility computing systems," *IGI Global*, vol. 15, 2012.

[15] G. IORDACHE, A. PASCHKE, M. MOCANU, and C. NEGRU, "Service level agreement characteristics of monitoring wireless sensor networks for water resource management (slas4water)," *Studies in Informatics and Control*, vol. 26, no. 4, pp. 379–386, 2017.

[16] "Network delay (wikipedia webiste)," https://en.wikipedia.org/wiki/Network_delay, (Accessed on 01/13/2018).

[17] C. Negru, F. Pop, M. Mocanu, and V. Cristea, "Storage solution of spatial-temporal data for water monitoring infrastructures used in smart cities," in *Control Systems and Computer Science (CSCS), 2017 21st International Conference on*. IEEE, 2017, pp. 617–621.

[18] O. Muresan, F. Pop, D. Gorgan, and V. Cristea, "Satellite image processing applications in mediogrid," in *Parallel and Distributed Computing, 2006. ISPDC'06. The Fifth International Symposium on*. IEEE, 2006, pp. 253–262.

[19] K. Shimizu, T. Ota, N. Mizoue, and S. Yoshida, "Forest monitoring using time series satellite images and its applications to tropical forests," *Nihon Ringakkai Shi/Journal of the Japanese Forestry Society*, vol. 98, no. 2, pp. 79–89, 2016.

[20] S. Amin and M. P. Goldstein, *Data against natural disasters: establishing effective systems for relief, recovery, and reconstruction*. World Bank Publications, 2008.

[21] F. B. Bektas, D. Mihon, V. Colceriu, K. Allenbach, C. Goksel, A. O. Dogru, G. Giuliani, and D. Gorgan, "Remotely sensed data processing on grids by using greenland web based platform," *International Journal of Advanced Computer Science and Applications*, vol. 3, no. 3, pp. 58–65, 2013.