Demonstration of Cloud GNSS Signal Processing

V. Lucas-Sabola, G. Seco-Granados, J. A. López-Salcedo, Universitat Autònoma de Barcelona (UAB), Spain;
J.A. Garcia-Molina, European Space Agency (ESA/ESTEC), TEC-ETN, and HE Space, The Netherlands
M. Crisci, ESA, The Netherlands

BIOGRAPHY

Vicente Lucas-Sabola received the B.Sc. in Telecommunication Systems Engineering in 2015 from Universitat Autònoma de Barcelona (UAB). Currently, he is a M.Sc. student in Telecommunication Engineering and research assistant at the SPCOMNAV group of UAB.

Gonzalo Seco-Granados holds a PhD degree from Univ. Politecnica de Catalunya (UPC) and an MBA from IESE, Universidad de Navarra. Since 2006, he is associate prof. at the Dept of Telecom. Eng. of UAB and head of the SPCOMNAV group.

José A. López-Salcedo received the M.Sc. and Ph.D. degrees in Telecommunication Engineering in 2001 and 2007 from UPC. He joined UAB and the SPCOMNAV group as an assistant prof. in 2006, and since 2013 he is an associate prof.

José A. García-Molina is Radio Navigation engineer at ESA/ESTEC in Noordwijk, The Netherlands, where he leads several R&D activities on GNSS receiver technology for ground and space applications. His main research interests include signal processing, estimation theory, GNSS receivers and signals, and cloud positioning applications.

Massimo Crisci is the head of Radio Navigation Systems and Techniques Section at the ESA. He is the technical domain responsible for the field of radionavigation and the head of a team of engineers providing radionavigation expert support to the various ESA programs (EGNOS and Galileo included). He holds a Ph.D. in automatics and operations research from the University of Bologna and a Master's degree in electronics engineering from University of Ferrara.

ABSTRACT

Nowadays, Global Navigation Satellite Systems (GNSS) receivers are used in applications in which size, power or computational constraints are gradually becoming of paramount importance. Furthermore, new GNSS will be fully operational in the coming years, which will considerably increase the amount of data to be processed by the user receiver. Because of the constraints of current user GNSS receivers, the employment of Cloud computing has become an alternative for migrating the GNSS signal processing tasks into a distributed, scalable and high-performance computing platform. Therefore, the Cloud paradigm facilitates the possibility of developing innovative applications where their particularities (e.g. massive processing of data, cooperation among users, security-related applications, etc.) make them suitable for implementation using a cloud-based infrastructure.

In this context, the purpose of this work is to introduce the concept of Cloud GNSS signal processing, based on the Cloud GNSS receiver proof-of-concept developed by the authors in collaboration with ESA. The focus will be placed on the Cloud GNSS receiver architecture, as well as on the performance evaluation of Elastic Compute Cloud (EC2) instances offered by Amazon Web Services (AWS). To do so, different tests on GNSS signal processing will be carried out along with the corresponding the cost of the EC2 service.

INTRODUCTION

In recent years the GNSS world is involved in an epoch of big changes with the future operability of new GNSS systems such as Galileo and BeiDou, in addition to already operable GNSS systems such as GPS or Glonass. These multiple GNSS will provide more than 40 visible satellites at a time (multi-constellation), which may solve different problems like harsh urban environments positioning or Geometric Dilution of Precision (GDOP). Nonetheless, the possibility of working with large number of satellites at the same time has the drawback of requiring the processing of a huge and overwhelming amount of data. To do so, an improvement of the computational requirements of GNSS receivers must be achieved. In addition, higher computational resources are translated in a higher power consumption receiver.

On the other hand, all previously mentioned changes are in opposite direction with current user applications. This is the case of Internet of Things (IoT), Smart City and Machine-To-Machine (M2M) applications, which demand low cost, low power consumption and miniaturized devices. With these paramount

characteristics, devices are tending to have limited computational resources, which are not sufficient to perform GNSS signal processing tasks oriented to multi-constellation. Furthermore, some user applications require advanced analyses that can exceed the receiver capabilities: authentication using regulated signals, liability-critical applications, crowdsourcing GNSS signal processing [1].

Due to the unfeasibility of implementing low cost, low power consumption and miniaturized devices with high computing resources, this paper presents the Cloud GNSS receiver paradigm (Figure 1). In this concept, all the GNSS signal processing workload is migrated to high-scalable and high-performance Cloud servers, which can provide nearly unlimited computing resources. In this manner, user terminals do not have to carry out any kind of GNSS signal processing tasks in the device itself. Instead, user terminals only need to gather the GNSS RF samples and send them to the Cloud. The Cloud GNSS receiver paradigm therefore allows the implementation of massive amount of data and sophisticated GNSS signal processing techniques without significantly increase the computational workload and energy consumption of the user terminal. Since the software receiver is allocated in Cloud servers, all user terminals can be easily upgraded at the same time applying all the required changes in the Cloud side.

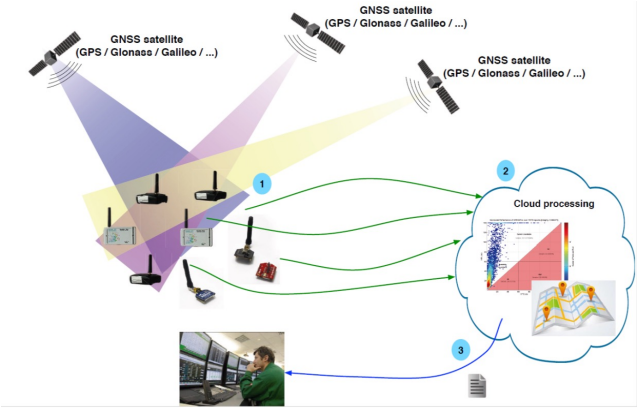

Figure 1 - Cloud-based GNSS receiver paradigm.

The objective of the present work is to set forth the Cloud GNSS paradigm based on the Cloud GNSS receiver proof-of-concept developed by the authors in collaboration with ESA. In the Cloud computing section, an introduction to Cloud computing and the services used under the scope of this work offered by Amazon Web Services is made. Such services will set the basis to develop all necessary modules for the implementation of the Cloud GNSS receiver. Then, the three parts in which the Cloud GNSS concept is based, i.e. user terminal, front-end, and back-end, are described in the Cloud-based GNSS receiver architecture, emphasizing on the uplink and channel bandwidth matters. Next, the performance of manifold instance families will be tested under different workloads, helping the reader to be aware of which EC2 instance is better for GNSS signal processing tasks and the price of using the service. Finally, in the last section conclusions are exposed.

CLOUD COMPUTING

Cloud computing is an internet-based computing platform which delivers on-demand IT resources and applications allowing users to pay per use: pay only for the resources and workload they are using. In the present years, Cloud computing technologies are becoming a reliable, secure, scalable and cheap way of using the newest IT technologies and migrating high-demanding tasks into high-performance computing platforms. Such Cloud platforms offer virtually unlimited computational power at low cost, making it an ideal solution for applications with massive amount users (or sensors). In that sense, Cloud technologies fit perfectly to be used as the back-end part of a Cloud-based GNSS receiver as the one presented in this work, where all the GNSS signal processing tasks are carried out in the Cloud.

There are three models of Cloud computing services: Software as a Service (SaaS), Platform as a Service (PaaS) and Infrastructure as a Service (IaaS) [2]. In the SaaS, software is hosted in the Cloud and accessible by internet and users do not have to worry about which resources or infrastructure is necessary to run the service they are using. That is the case presented in this work: the Cloud-based GNSS receiver. PaaS provides a development environment in the cloud. Thus, users do not need to install or configure any development software or environment to work with. Finally, the IaaS provides Virtual Machines (a machine that acts as a physical computing machine) to the users. With IaaS, users can create large computing clusters demanding all the IT resources as needed.

On the other hand, the Cloud computing infrastructure of a SaaS can be divided in two sections: front-end and back-end. The front-end section is the side which is visible to the user, i.e. user interface, and can be accessed through a user terminal (computer or smartphone usually). Throughout this interface, the user can interact with the Cloud platform, service or application and attach all the required inputs, e.g. configuration, files, user credentials, etc. The back-end section is the side which is not visible to the user. In this section of the cloud infrastructure is where all the IT resources are allocated and the computational tasks are carried out.

Cloud infrastructures can also be divided in four models depending on how the service model is implemented: private, public, community and hybrid Cloud. The private Cloud is a Cloud service dedicated for one single

organization. Public Cloud is completely open for general public and it is shared between all the public users. Then, community Cloud is an infrastructure shared between multiple organizations. Finally, hybrid Cloud is composed by multiple Cloud models, and can be public or private.

These days, there is a wide variety of enterprises offering Cloud platform services such as Amazon Web Services, Google Cloud Platform, Stratosphere, Microsoft Azure, Oracle, and so on. Such enterprises are offering Cloud services for organizations and individual users as well. In the context of this work, the goal of this paper is to open the door of Cloud computing to the reader, reviewing the main services and configurations offered, and in particular, for the development of a Cloud-based GNSS receiver. Nonetheless, the selection of one commercial Cloud platform in front of another is not further discussed in this paper. For the scope of this work, some of the services offered by AWS will be discussed and analyzed.

Amazon Web Services

Amazon Web Services (AWS) offers more than 70 cloud-computing services (e.g. compute, storage, database, networking) that operates from multiple geographical locations, composed of regions and Availability Zones (AZ). These AZ are designed to be isolated from others, thus achieving better fault tolerance and stability. Resources can be allocated in multiple locations, which mean that in case of failure in an AZ, another one can handle the corresponding request without interrupting the Cloud service. Services can be controlled and configured through the Management Console or using a Software Development Kit (SDK) developed by AWS (e.g. Python, Ruby, JavaScript). It should be noticed that to automatize the system, it is preferable to program the infrastructure using the SDK tools.

Some of these services can be of high interest for Cloud-based positioning applications and in particular for implementing a Cloud-based GNSS receiver as the case study.

Amazon Web Services: Elastic Compute Cloud

The Elastic Compute Cloud (EC2) service forms the main part of AWS. It is a virtual computing environment that allows users to launch instances (Virtual Machine) providing scalable and re-sizable computing capacity. These instances or Virtual Machines are launched using a wide range of operating systems and loading custom application environments if necessary. EC2 provides a

Table 1 - EC2 instance characteristics

Model	vCPU	Memory (GB)	Storage (GB)	Physical Processors	Clock Speed (GHz)	Price ($/hour)
t2.nano	1	0.5	EBS Only	Intel Xeon family	up to 3.3	0.0075
t2.micro	1	1	EBS Only	Intel Xeon family	up to 3.3	0.015
t2.small	1	2	EBS Only	Intel Xeon family	up to 3.3	0.03
t2.medim	2	4	EBS Only	Intel Xeon family	up to 3.3	0.06
t2.large	2	8	EBS Only	Intel Xeon family	up to 3.3	0.12
m4.large	2	8	EBS Only	Intel Xeon E5-2676 v3	2.4	0.143
m4.xlarge	4	16	EBS Only	Intel Xeon E5-2676 v3	2.4	0.285
m4.2xlarge	8	32	EBS Only	Intel Xeon E5-2676 v3	2.4	0.57
m4.4xlarge	16	64	EBS Only	Intel Xeon E5-2676 v3	2.4	1.14
m4.10xlarge	40	160	EBS Only	Intel Xeon E5-2676 v3	2.4	2.85
m3.medium	1	3.75	1x4 SSD	Intel Xeon E5-2670 v2	2.5	0.079
m3.large	2	7.5	1x32 SSD	Intel Xeon E5-2670 v2	2.5	0.158
m3.xlarge	4	15	2x40 SSD	Intel Xeon E5-2670 v2	2.5	0.315
m3.2xlarge	8	30	2x80 SSD	Intel Xeon E5-2670 v2	2.5	0.632
c4.large	2	3.75	EBS Only	Intel Xeon E5-2666 v3	2.9	0.134
c4.xlarge	4	7.8	EBS Only	Intel Xeon E5-2666 v3	2.9	0.267
c4.2xlarge	8	15	EBS Only	Intel Xeon E5-2666 v3	2.9	0.534
c4.4xlarge	16	30	EBS Only	Intel Xeon E5-2666 v3	2.9	1.069
c4.8xlarge	36	60	EBS Only	Intel Xeon E5-2666 v3	2.9	2.138
c3.large	2	3.75	2x16 SSD	Intel Xeon E5-2680 v2	2.8	0.129
c3.xlarge	4	7.5	2x40 SSD	Intel Xeon E5-2680 v2	2.8	0.258
c3.2xlarge	8	15	2x80 SSD	Intel Xeon E5-2680 v2	2.8	0.516
c3.4xlarge	16	30	2x160 SSD	Intel Xeon E5-2680 v2	2.8	1.032
c3.8xlarge	32	60	2x320 SSD	Intel Xeon E5-2680 v2	2.8	2.064
r3.large	2	16	1x32 SSD	Intel Xeon E5-2670 v2	2.5	0.2
r3.xlarge	4	30.5	1x80 SSD	Intel Xeon E5-2670 v2	2.5	0.4
r3.2xlarge	8	61	1x160 SSD	Intel Xeon E5-2670 v2	2.5	0.8
r3.4xlarge	16	122	1x320 SSD	Intel Xeon E5-2670 v2	2.5	1.6
r3.8xlarge	32	224	2x320 SSD	Intel Xeon E5-2670 v2	2.5	3.201

wide catalog of instances types and models comprising multiple combinations of CPU, RAM, storage and network capacity.

At high level, the AWS instances can be divided into two main types: burstable and fixed performance instances. The former provide a baseline level of CPU performance with the ability to burst above the baseline. That baseline performance and ability to burst are governed by CPU Credits [3], which are received continuously depending on the instance size. Burstable instances (e.g. *T2*) accumulate CPU credits while they are idle, and use CPU credits when they are active. Thus, burstable instances are suitable for applications or workloads which do not require the full CPU consistently (e.g. web servers, databases). In contrast, fixed performance instances (e.g. *M3, C3, R3, C4*) do offer consistent CPU performance. Therefore, fixed performance instances are better suited for workloads that require intensive and consistence CPU performances, protecting the user from the variable performance of the instance.

A summary of instances types that are offered by AWS [3] and are on the scope of this work is provided in Table 1. It should be noticed that the price per hour of the instances is not fixed and can vary from the ones showed in this work (e.g. the price vary depending on the Availabilty Zone where launching the instance). Furthermore, the price of the different EC2 instances depends on the region it is launched. In Table 1, the displayed prices are referred to the EU (Frankfurt) region. As it can be seen, a large range of CPU, memory and storage combinations is available for the user. Thus, the selection of the right instance depends on the computing requirements of the application or workload. For example, *T2* instances are suited for applications which do not require a consistent CPU performance such as web servers or small databases; *C4* instances are of high interest for multiple Cloud-based GNSS applications thank to their CPU cores, e.g. fast executions with low integration times; *R3* instances are suited for Cloud applications that require an extensive amount of data to be processed, e.g. executions which require long integration times in high-sensitivity techniques.

For launching a new instance the user must introduce some configuration parameters such as the selection of the Amazon Machine Image (AMI), the type of instance, the desired storage, security group and the key pair. An AMI is a packed-up environment that includes all the necessary OS and software to set up and boot the instance. There are multiple AMI templates (e.g. private, public, self-created, in the AWS marketplace) that are provided by AWS. Self-created AMIs are of high interest when creating clusters of cloned instances. They can be simply created by making and image or snapshot (that will be stored in the Elastic Block Storage service) of an instance from the Manager Console. Thus, when launching a new instance as users makes requests and more computational power is required, the instance will be a copy of the cloned instance, which means that they will be ready to process the incoming workloads just after booting the SO. Therefore, when launching an instance, the id of the desired AMI must be attached. Then, the type of the instance must be chosen (Table 1) together with the size of any additional storage if necessary. Afterward, the configuration of the Security Group is made. A security group is a set of firewall rules that control the traffic of your instance. Hence, rules can be added to allow any specific traffic to reach the instance (e.g. SSH, HTTP/S). These security groups can also be easily created through the Manager Console. Finally, a selection of an existing key pair or creation of a new one must be done. A key pair consists of a public key that AWS stores and a private key file the user stores. Together, both keys allow the user to connect to the instance securely (e.g. SSH). This process can also be carried out in the Manager Console.

Amazon Web Services: Elastic Bloc Storage

The Elastic Block Storage (EBS) provides persistent block level storage volumes for use with EC2 instances. That means that created storage volumes (e.g. snapshots of instances) can be attached to instances. Such snapshots, as explained before, can be used to initiate new instances, expand the volume size of a working instance, or move the volume across different AZ. On the other hand, some instances storage is based on EBS (e.g. *T2*, *M4*) instead of HDD or SDD. For the first snapshot of a volume, Amazon EBS saves a full copy of your data to Amazon Simple Storage Service. For this case, the price of the service in the EU (Frankfurt) region is of $0.054 per GB-month of data stored. Hence, the creation of AMIs and storage of the corresponding EBS volume allows the user to clone instances in a time and cost-effective way.

Amazon Web Services: Simple Storage Service

Amazon Simple Storage Service (S3) provides a secure, durable and high-scalable Cloud storage. With this service, data with a maximum size of 5 TB can be stored, written, read and deleted as resources, i.e. buckets. In the Cloud GNSS receiver, S3 is used to store the user's input data (e.g. raw GNSS samples file, JSON, RINEX). This process is carried out uploading the files from the EC2 instance that works as the front-end (web server) to the S3 repository. To do so, the user must be connected to the corresponding S3 repository and create a key including the name of the file. Afterwards, the selected file can be uploaded pointing to the generated key. The same process is carried out to download objects from S3 to EC2. The data transfer speed between EC2 instances and S3 (in both directions) is one of the most outstanding

characteristic this service offers: roughly 14 MB/s, together with a low rate of $0.032 per GB approximately.

Amazon Web Services: Simple Queue Service

The Simple Queue Service (SQS) is a fast, reliable, scalable, fully managed message queuing service, making easy to decouple the components of a Cloud applications, e.g. front-end and back-end. With this service, an unlimited number of queues with an unlimited number of messages can be created at low rate: first 1 million SQS requests per month are free, after which requests have a rate of $0.0000005. Request can include from 1 to a maximum of 10 messages and a maximum total payload of 256 KB. Each chunk of 64 KB of payload is billed as 1 request. That is to say, the maximum payload of 256 KB is formed by four requests. Furthermore, messages can also have attributes of different kind, i.e. string, number or binary, which is useful to provide structured metadata items about the message itself.

For the sake of clarity a brief summary of when and how use SQS is provided as follows. In the scope of this work, this service has been used to connect the front-end interface (web server) with the back-end (core of the Cloud GNSS receiver). Thereby, when a user sends an execution to the Cloud, a JSON message is generated with all the required input data and sent to a SQS queue. Then, a resource manager is in charge of read those and sent them to the corresponding back-end instance. Finally, the back-end instance read the message and performs the execution requested by the user. In order to read and send messages, the user must be connected to the SQS service through the SQS management console or an instance. Then, the queue to get connection with must be selected. Finally, the message is sent by adding the body, attributes if needed and the previous selected queue.

CLOUD-BASED GNSS RECEIVER ARCHITECTURE

In previous sections the basic blocks, i.e. EC2, SQS, S3, and EBS services, to develop a Cloud GNSS receiver have been explained. In this section we will describe the architecture of an experimental Cloud-based GNSS receiver developed in collaboration with ESA. This proof-of-concept receiver has been successfully tested with real GNSS data gathered with an USRP N200 and applying different types of workloads. In Figure 2 is displayed the block diagram of the developed Cloud GNSS receiver [1]. From a high-level perspective, it is composed of three main blocks: the Cloud user terminal, the Cloud front-end, i.e. user interface, and the cloud back-end.

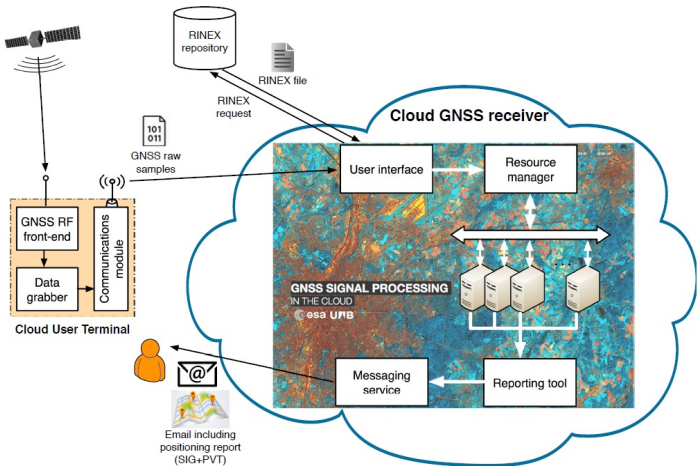

Figure 2 - Cloud-based GNSS receiver architecture [1].

Table 2 - SDRs comparison

SDR Model	Frequency range	ADC resolution	Bandwidth	Rx/Tx	Price
USRP N200	DC – 6 GHz	14 bits	100 MHz	Rx & Tx	$1795.10
LimeSDR	100 kHz – 3.8 GHz	12 bits	61.44 MHz	Rx & Tx	$289
RTL-SDR Donger	500 kHz – 1.7 GHz	8 bits	3.2 MHz	Rx	$20
HackRF One	1 MHz – 6 GHz	8 bits	20 MHz	Rx & Tx	$299
BladeRF x40	300 MHz – 3.8 GHz	12 bits	40 MHz	Rx & Tx	$420
Airspy R2	24 – 1800 MHz	12 bits	10 MHz	Rx	$199

Table 3 – Raw GNSS samples file size

GNSS	Band	F_s	Quantization	T_{signal}	Chunk size	Packets per hour (1GB/month)
GPS	L1 C/A	4 MHz	2	2 ms	1.95 Kbyte	746.85
GPS	L1 C/A	5 MHz	16	40 ms	390.6 Kbyte	3.73
Galileo	E1B	8 MHz	2	8 ms	15.6 Kbyte	93.36
Galileo	E1C	8 MHz	2	100 ms	195.3 Kbyte	7.46
Galileo	E5a data	100 MHz	2	20 ms	488.3 Kbyte	3

Cloud user terminal

One of the blocks that form the Cloud GNSS receiver architecture is the user terminal. It is composed of three main elements: a RF front-end (not to be confused with the Cloud front-end) that works at the GNSS frequency band of interest, a data grabber that digitizes the GNSS signals, and a communication module to interact and send the captured GNSS signal to the Cloud GNSS receiver.

Table 2 presents some of the Software Defined Radio (SDRs) products which are available in the market. Such SDRs can work as GNSS RF front-end and data grabber, and in some cases, they can even work as communication module.

Nevertheless, as explained in the Introduction section, the advent of Smart City, IoT and M2M applications will drive devices to be miniaturized and with low power consumption [4], where SDRs are not the best option for such applications. Instead, cheap, small and low consuming sensors should be used in order to capture, digitize and sent the raw GNSS sample file to the Cloud GNSS receiver where it would be processed with virtually unlimited computing capacity. In this manner, the creation of a sensor network of GNSS receivers would be very cheap because there is not GNSS signal processing in the sensor itself. The data transfer can be carried out through the communication module using different technologies. In that sense, 3GPP has developed different standards that address to the IoT market and is suitable for the Cloud GNSS receiver concept: eMTC, NB-IOT and EC-GSM-IoT [5]. The former supplies LTE in-band coverage of roughly 155.7 dB bandwidth of 1.08 MHz and a peak rate of 1 Mbps both in downlink and uplink. NB-IOT provides LTE in- and guard-band coverage of approximately 164 dB, bandwidth of 180 kHz and a peak rate of 250 kbps for downlink and uplink with multi-tone or roughly 20 kbps with single tone. The latter supplies GSM in-band coverage of 154-164 dB depending on the power class, a bandwidth of 200 kHz per channel, and a peak rate of 70 kbps both for uplink and downlink using GMSK or 240 kbps using 8PSK. The coverage of the mentioned 3GPP standards uses the Maximum Coupling Loss (MCL) methodology: the coupling loss is defined as the total long-term channel loss over the link between the UE antenna ports and the antenna ports, and includes in practice antenna gains, path loss, shadowing, body loss, etc. The MCL is the limit value of the coupling loss at which the service can be delivered, and therefore defines the coverage of the service. The MCL is independent of the carrier frequency [6].

Other information of paramount importance that the user should consider is the size of the raw GNSS samples file to be sent from the receiver to the Cloud, which is directly related to the channel bandwidth required for transmission. In Table 3 is shown the required chunk size to process different GNSS signals with the necessary sampling frequency (real samples), the quantization of the file and the signal time length. The chunk size can be obtained as:

$$Chunk\ size = T_{signal} \cdot F_s \cdot Quantization \qquad (1)$$

Nonetheless, the user has to face the tradeoff of choosing between a short signal length and a small chunk size. This is because as the signal length increases, so does the chunk size, as shown in Equation (1). Furthermore, as the signal length decreases, the sensitivity of the receiver worsens. Thus, the C/N0 must be higher in order to acquire the satellites [7]. For the sake of the clarity, in Table 3 is also shown for each of the cases the number of packets that could be sent per hour having a 1 GB/month flat rate, which is usual nowadays in many cellular plans. This rate of packets per hour would be suitable for the previously mentioned 3GPP standards eMTC, NB-IOT and EC-GSM-IoT. Therefore, the required channel bandwidth is not a potential showstopper for the transmission of raw GNSS samples files to the Cloud-based GNSS receiver, since it is just a matter of sending more or less packets per hour.

Cloud front-end

The front-end is the interface with which the user can interact with the Cloud GNSS receiver and the back-end. It is based on a Ruby on Rails webpage where users can log in and manage their new or current execution through their own private desktop using the HTTP/S service. When launching a new execution, the process is divided in three steps with which the user can configure the GNSS software receiver parameters as needed:

1) *Raw Samples Settings*. The first step is targeted to the raw GNSS samples file. The file can be uploaded to the Cloud through a file located at the user's terminal (e.g. computer, smartphone), selecting a previously uploaded file or giving an URL (e.g. Dropbox, Drive, FTP), in such case the file is automatically downloaded by the Cloud GNSS Receiver. As explained in Amazon Web Services: Simple Storage Service, files are automatically transferred to the S3 repository from the EC2 instance where the front-end is working. In addition, the user must attach information about the raw GNSS samples file in order to allow the Cloud-based GNSS receiver to work with the file, e.g. read, down-convert. Such information reflects how the raw GNSS samples file was captured and digitized: sampling frequency, front-end bandwidth, intermediate frequency, delay, sample format, encoding and quantization of the file. This information can also be attached within an ION SDR GNSS metadata standard XML file [8], that is

defined in terms of core metadata classes as shown in Figure 3. The objective of using such metadata files is to facilitate the interoperability between SDRs and GNSS receivers and to standardize all metadata related with the capture of raw GNSS sample file [9].

2) *Receiver Settings*. In the second step the configuration of the GNSS software receiver is made. Thanks to this, the user can select which kind of analysis wants to perform with parameters such as GNSS signal and band of interest, the integration time, number of snapshot, near-far detection, PVT computation, etc. These parameters will determine the required computing time and resources in the back-end.
3) *Assistance Settings*. The last step purpose is to give the receiver all the necessary assistance information. Hence, the user has to attach a list of satellites to be searched. In addition, a list with the Doppler of the satellites the user wants to search can be also attached. This information is of interest when the user wants to speed up the execution, as it will be seen in EC2 instance performance in Cloud GNSS signal processing. Finally, if the computation of the PVT was enabled in the previous step, a RINEX (Receiver Independent Exchange Format) file must be attached. Such RINEX file can be uploaded to the Cloud either using a local file, a URL or let the Cloud GNSS receiver download it automatically from a list of available GNSS Data Center (GDC) giving the date the raw GNSS samples file was created and a station close to the location it was taken.

After filling the required information in the above steps, the execution can be launched and managed through the private desktop of the user. Finally, the execution is sent to the Cloud back-end SQS queue as a JSON message where will be handled and processed.

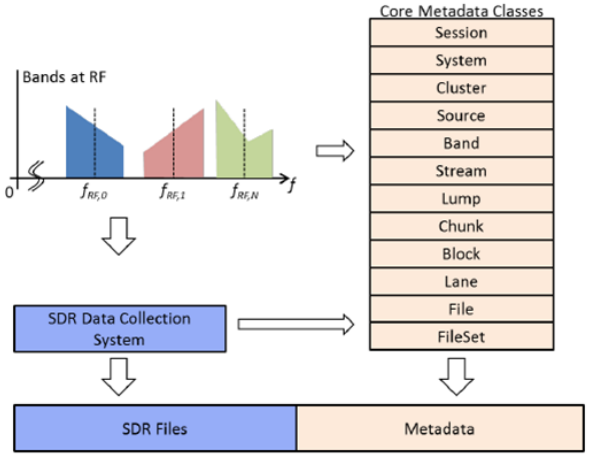

Figure 3 - Summary of the ION SDR metadata classes [8]**.**

Cloud back-end

Back-end EC2 instances are the bulk of the Cloud GNSS receiver architecture and where all the GNSS signal processing and computational tasks are carried out. Instances are based on the High-Sensitivity (HS) GNSS software receiver developed in the framework of the ESA funded project DINGPOS (Signal Processing Techniques and Demonstrator for Indoor GNSS Positioning) [10], [11]. It is a snapshot-based receiver compatible with GPS L1/L5 and Galileo E1C/E5a. One of the major assets of this software receiver is the capability of providing C/N0 sensitivity down to 15 dBHz, implementing long integration times using advanced non-coherent integrations, implementing near-far and multipath detection as well as interference mitigation techniques. Hence, the Cloud GNSS receiver is suitable for processing extremely weak signals using long integration times, acquiring signals under multipath or adverse signal conditions, e.g. indoor conditions [7].

Once the JSON message with all the configuration parameters of the previously mentioned steps is sent to the SQS queue, a resource manager is in charge of reading the SQS message and sending it to the corresponding back-end EC2 instance to be processed and executed. The choice of the back-end EC2 instance to perform the execution is determined by the workload of the current running instances, which can be managed by monitoring software such as Ganglia, Nagios or the Amazon CloudWatch Service. In case there is not enough room to launch an execution in the running instances, a new instance is initiated. Instances are booted with a pre-configured image (snapshot) previously generated using the Elastic Block Service of AWS with the HS-GNSS software receiver and all the other required software. Thus, new launched instances are ready to work just after being booted.

Lastly, after finishing the execution, a report in PDF format is generated with the obtained results by the software receiver and is sent to the user's email address. The report provides: results at signal level, i.e. correlation peak power, C/N0 and Doppler frequency; results at observable level if the PVT computing is enabled, i.e. estimated Time of Week (TOW), 2D projection onto north-east axis, map screenshot with the position with different zoom approaches and a polar plot of the visible satellites; multipath analysis, i.e. estimated C/N0 values and Slope Asymmetry Metric (SAM); and a data log with the obtained numeric results.

EC2 INSTANCE PERFORMANCE IN CLOUD GNSS SIGNAL PROCESSING

In previous sections, an introduction to the wide collection of EC2 instances offered by AWS has been

provided and related with the Cloud-based GNSS receiver architecture. Next, a performance assessment of the *m3.medium, c3.large, m4.large, r3.large, c4.xlarge, c4.8xlarge*, and *t2.small* instances according to relative execution time will be presented. *t2.small* instance results can differ depending on the number of running *t2* instances, due their burstable behavior. In this sense, this paper aims at presenting the performance results of different instances under three workload cases: different number of frequency bins when searching the Doppler frequency of the satellites, different integration times, and different number of satellites to be acquired. Thanks to this, we can get the notion of which instance, whose characteristics are presented in Table 1, is better to use depending on the kind of workload to be carried out. Based on these results, in EC2 service cost section a study of the most cost-effective instance for a typical execution is made.

The test results have been performed with the following raw GNSS samples file parameters:

- GNSS signal: GPS L1 C/A
- Chunk size: 400 Kbyte
- Sampling frequency: 5 MHz
- Quantization: 16 bits
- Signal length: 40 ms
- Number of snapshots: 1

In the first test case, the integration time is set to 20 ms, a search of 5 satellites is made, and the number of frequency bins to be performed when searching the Doppler frequency of the satellites to be acquired is varied from -5 kHz to +5 kHz in steps of 1 kHz with a frequency resolution of ±500 Hz, a mandatory step when there is no assistance data available. The number of frequency search (Nf) can be obtained as shown in Equation (2), which depends on the maximum frequency error the user wants to search (F_{max}). In case of attaching the assistance GNSS data, the software receiver implements a single search only, at the proper frequency indicated by the assistance data. In that case, Nf is equal to 1. The reader should be aware that as the number of frequency bins decreases, also does the possibility to acquire satellites due the lower range of the Doppler frequency search.

$$Nf = round\left(\frac{F_{max}}{1\ kHz}\right) \cdot 2 + 1 \qquad (2)$$

In Figure 4 we show the obtained results for this test, all of them normalized at the faster execution of 3.57 seconds. It can be seen that the faster instances are the *c4.8xlarge* and the *c4.xlarge*, and the slower one is the *m3.medium,* a condition that is repeated in the different performed workloads during this section. The execution time of the process of this study of case can be obtained as:

$$T_{exec} \cong T_{500Hz}\ \frac{F_{max}\ [Hz]}{500} \qquad (3)$$

Where T_{500Hz} is the time needed for processing 5 satellites with a frequency search of $F_{max} = 500\ Hz$. Meaning that approximately every time the number of frequency bins is doubled, so does the execution time (i.e. processing a frequency search of 4000 Hz needs twice the time than processing a 2000 Hz search).

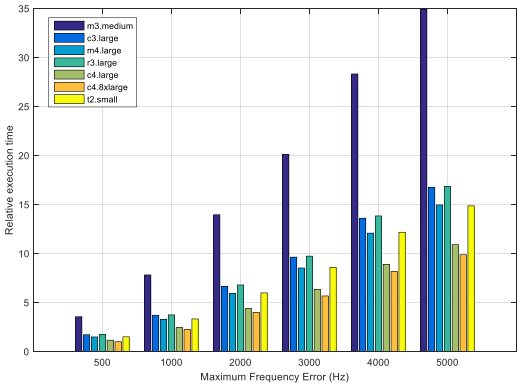

Figure 4- Relative execution time with different frequency bins.

Then, in the second study of case, the number of satellites to be searched is equal to 5, and the integration time is varied in three steps: 20, 40 and 100 ms. In addition, the same tests have been performed enabling (red part of the bar) and disabling (colored part of the bar) the assistance GNSS data, in such case the frequency bin search has been set from -4000 Hz to +4000 Hz. At first sight, looking at Figure 5, where the relative execution time has been normalized at 1.75 seconds, it is spotted that using assistance information reduces the execution time considerably: roughly between 15-20 times depending on the instance family. Consequently, in order to speed up the execution it is highly recommended to give the assistance information. In case of not having assistance GNSS information, the increase of the execution time can be obtained as in Equation (4), where $T_{20ms-NoAGNSS}$ is the time needed for an execution with 20 ms of integration time without using assistance information. If the user attaches the assistance GNSS information, the increase can be calculated with the Equation (5), where $T_{20ms-AGNSS}$ is the time needed for an execution with 20 ms of integration time using assistance GNSS data.

$$T_{exec} \cong T_{20ms-NoAGNSS} + T_{20ms-NoAGNSS} \cdot 0.47\left(\frac{T_{int}\ [ms]}{20} - 1\right) \qquad (4)$$

$$T_{exec} \cong T_{20ms-AGNSS} + T_{20ms-AGNSS} \cdot 0.37\left(\frac{T_{int}\ [ms]}{20} - 1\right) \qquad (5)$$

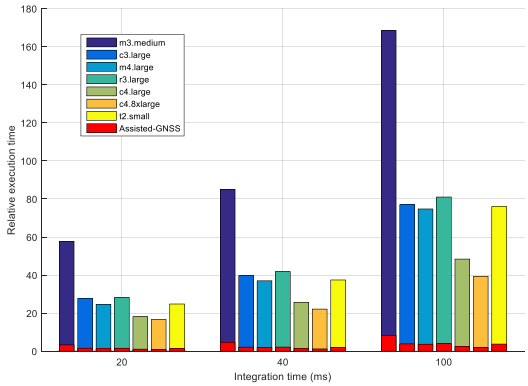

Figure 5 - Relative execution time with different integration time.

Finally, in the third and last case of study, the integration time is fixed to 20 ms and the execution has been implemented searching 1, 2 and 5 satellites. In addition, as in the previous case, the test has been carried out enabling the assistance GNSS data (red bar) and disabling the assistance information (colored bar), in which case the frequency bin search has been set up from -4000 Hz to +4000 Hz. The obtained results are shown in Figure 6, where we can clearly see how the use of assistance GNSS data reduces the execution time from roughly 10 to 20 times as the number of satellites to be searched increases. When the user does not use any assistance information, the execution time can be obtained through Equation (6),

$$T_{exec} \cong T_{1_{sat-NoAGNSS}} \cdot n_{sat} \qquad (6)$$

where $T_{1sat-NoAGNSS}$ is the execution time needed for processing 1 satellite without using assistance GNSS data. That is to say, the execution time can be calculated multiplying the time needed to search one satellite by the total number of satellites to be searcher. On the other hand, when the user attaches assistance GNSS data, the execution time depending on the number of satellites can be calculated using Equation (7), where $T_{1sat-AGNSS}$ is the execution time for processing 1 satellite using assistance information.

$$T_{exec} \cong T_{1_{sat-AGNSS}} + T_{1_{sat-AGNSS}} \cdot 0.35\ (n_{sat} - 1) \qquad (7)$$

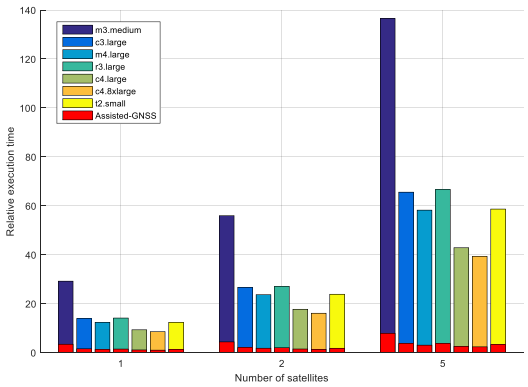

Figure 6 - Relative execution time with different number of satellites.

In the three cases of study performed so far, we have seen that the fastest instances are the *c4.xlarge* and the *c4.8xlarge*, while the slowest one seems to be the *m3.medium*. Moreover, we have also seen that using assistance data reduces the execution time considerably, which is translated in a lower cost per execution.

EC2 service cost

In EC2 instance performance in Cloud GNSS signal processing, we have seen which instances are faster and better suited to perform GNSS signal processing tasks. In this section, we will study which is the more cost-effective instance and what is the cost of using these instances as part of the Cloud GNSS receiver.

Table 4 - EC2 instances cost.

Model	vCPU	Memory (GB)	Price per hour (EU-Frankfurt)	Executions per hour	Price per execution	Price per month
m3.medium	1	3.75	$0.079	618.56	$0.000128	$0.34
c3.large	2	3.75	$0.129	1814.74	$0.000071	$0.19
m4.large	2	8	$0.143	2147.65	$0.000067	$0.18
r3.large	2	15	$0.2	2060.44	$0.000097	$0.26
c4.xlarge	4	7.5	$0.267	5971.04	$0.000045	$0.12
c4.8xlarge	36	60	$2.138	44720.50	$0.000048	$0.13
t2.small	1	2	$0.03	1490.31	$0.000020	$0.05

Table 4 depicts the EC2 service cost with the following receiver configuration:

- GNSS signal: GPS L1 C/A
- Chunk size: 400 Kbyte
- Sampling frequency: 5 MHz
- Quantization: 16 bits

- Signal length: 40 ms
- Number of snapshots: 1
- Integration time: 20 ms
- Assistance-GNSS: Enabled
- Number of satellites: 5
- Snapshots per hour: 3.73

T2.small offers the best price per execution, but the burstable behavior of the *T2* instance family makes it not appropriate for GNSS signal processing workloads. Even with their higher price per hour, *c4.xlarge* and *c4.8xlarge* instances provides the lower price per execution and price per month (for a single user terminal with 3.73 packets per hour to be processed, including the computation of the PVT for each snapshot) for fixed performance instances thanks to their computing capabilities.

In this manner, we can see that if the resource manager optimizes the use of the EC2 instances, the price per month of the overall service is quite low: ~ $0.1 - $0.3 depending on the instance type. That means that the price for the use of the Cloud GNSS receiver is low enough to be properly considered to be implemented in miniaturized, low powered and low energy consumption devices (e.g. IoT, Smart City).

CONCLUSION

This paper has given an introduction to the use of Cloud computing infrastructures such as Amazon Web Services for the development of a Cloud GNSS receiver. We have described some of the services offered by AWS which are convenient for GNSS signal processing, highlighting the use of the EC2 service. The notion of migrating GNSS signal processing workloads to the Cloud is of special interest for many applications which require of computational demanding techniques. Then, the architecture of a Cloud GNSS receiver developed by the authors in collaboration with ESA has been presented. A summary of SDRs products to be used as user terminal together with different 3GPP standards for IoT applications has been provided. Thanks to this, we have seen that the necessary channel bandwidth for the communication between the user terminal and the Cloud is feasible with current 3GPP standards. Next, the front-end and back-end elements of the Cloud GNSS receiver have been described. Then, a study of the EC2 instance performance for GNSS signal processing tasks has been made with three different study cases, obtaining quantitative expressions for the calculus of the required execution time depending on the execution configuration. Finally, the EC2 service cost has been calculated to demonstrate the feasibility of developing and using a Cloud GNSS receiver as the solution for miniaturized, low powered and low energy consumption devices.

ACKNOWLEDGMENTS

The views presented in this paper represent solely the opinion of the authors and not necessarily the view of ESA. This work was partly supported by the Spanish Government under grant TEC2014-53656-R and by the European Space Agency (ESA) under contract No. 4000113891/15/NL/HK.

REFERENCES

[1] V. Lucas-Sabola, G. Seco-Granados, J. A. López-Salcedo, J. A. García-Molina, and M. Crisci, "Cloud GNSS receivers: New advanced applications made possible," in *Proc. International Conference on Localization and GNSS (ICL-GNSS)*, 2016.

[2] A. Gajbhiye and K. M. P. D. Shrivastva, "Cloud computing: Need, enabling technology, architecture, advantages and challenges," *Proc. 5th Int. Conf. Conflu. 2014 Next Gener. Inf. Technol. Summit*, pp. 1–7, 2014.

[3] Amazon Web Services, "Amazon Web Services Cloud Products," in *https://aws.amazon.com*.

[4] O. Vermesan and P. Friess, *Internet of Things: Converging Technologies for Smart Environments and Integrated Ecosystems*. 2013.

[5] D. Fiore and Qualcomm Technologies Inc., "3GPP Standards for the Internet-of-Things," in *GSMA IoT*, 2016.

[6] 3rd generation Partnership Project, "3gpp tr 36.824," 2012.

[7] G. Seco-Granados, J. a López-Salcedo, D. Jiménez-Baños, and G. López-Risueño, "Challenges in Indoor Global Navigation Satellite Systems," *IEEE Signal Process. Mag.*, no. February, pp. 108–131, 2012.

[8] ION GNSS SDR Metadata Working Group, "Global Navigation Satellite Systems Software Defined Radio sampled data metadata standard," *The Institute of Navigation, Revision 0.1*, 2015.

[9] S. Gunawardena and T. Pnay, "GNSS SDR Metadata Standard Working Group Report," in *Proc. ION GNSS+*, 2015, pp. 3218–3221.

[10] J. López-Salcedo, Y. Capelle, M. Toledo, G. Seco, J. López Vicario, D. Kubrak, M. Monnerat, A. Mark, and D. Jiménez, "DINGPOS: a hybrid indoor navigation platform for GPS and GALILEO," *21st Int. Tech. Meet. Satell. Div. Inst. Navig. (ION GNSS 2008)*, vol. 2, pp. 1780–1791, 2008.

[11] J. A. López-Salcedo and G. Seco-Granados, "Datasheet of the DINGPOS HS-GNSS Software Receiver," in *SPCOMNAV-UAB, http://spcomnav.uab.es/docs/projects/datasheet_HSGNSS-SPCOMNAV.pdf*.