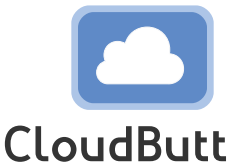

HORIZON 2020 FRAMEWORK PROGRAMME

CloudButton

(grant agreement No 825184)

Serverless Data Analytics Platform

D2.1 Experiments and Initial Specifications

Due date of deliverable: 30-06-2019
Actual submission date: 24-07-2019

Start date of project: 01-01-2019 Duration: 36 months

Summary of the document

Document Type	Report
Dissemination level	Public
State	v1.0
Number of pages	52
WP/Task related to this document	WP2 / T2.1, T2.3, T2.4, T2.5, T2.6
WP/Task responsible	URV
Leader	Pedro García (URV)
Technical Manager	David Breitgand (IBM)
Quality Manager	Gil Vernik (IBM)
Author(s)	Pedro García (URV), Juan José Lozano (ANSWARE), Victoria Moreno (ANSWARE), Juan Manuel Cintas (MATRIX), Theodore Alexandrov (EMBL), Paolo Ribeca (TPI), Gerard París (URV).
Partner(s) Contributing	URV, IBM, ATOS, EMBL, MATRIX, ANSWARE, TPI, RHAT, Imperial, IMT.
Document ID	CloudButton_D2.1_Public.pdf
Abstract	Description of use case scenarios, experiments and benchmarking framework. Initial specifications of the architecture.
Keywords	use cases, experiments, architecture, state of the art.

History of changes

Version	Date	Author	Summary of changes
0.1	03-06-2019	Pedro García	First draft with collaborations from all partners.
0.2	12-06-2019	Juan José Lozano, Victoria Moreno	Inputs from ANSWARE
0.3	12-06-2019	Juan Manuel Cintas	Inputs from MATRIX
0.4	12-06-2019	Theodore Alexandrov	Inputs from EMBL
0.5	08-07-2019	Paolo Ribeca	Inputs from TPI
1.0	24-07-2019	Victoria Moreno, Juan Manuel Cintas, Theodore Alexandrov, Gerard París	Review comments addressed. Final version.

Table of Contents

List of Abbreviations and Acronyms

ACDP	Advisory Committee on Dangerous Pathogens
AEMET	Agencia Estatal de Meteorología (*State Meteorological Agency (Spain)*)
AKS	Azure Kubernetes Service
API	Application Programming Interface
AWS	Amazon Web Services
BBSRC	Biotechnology and Biological Sciences Research Council
CLI	Command-line interface
CNCF	Cloud Native Computing Foundation
CNIG	Centro Nacional de Información Geográfica (*National Center for Geographic Information Systems (Spain)*)
COS	Cloud Object Storage
CSV	Comma-separated values
CUDA	Compute Unified Device Architecture
DAG	Direct Acyclic Graph
DBMS	Database Management System
DEM	Digital Elevation Model
DSL	Domain-specific language
DSM	Digital Surface Model
EBI	European Bioinformatics Institute
EC2	Amazon Elastic Compute Cloud
EKS	Amazon Elastic Container Service for Kubernetes
EMBL	European Molecular Biology Laboratory
ESA	European Space Agency
FAANG	Functional Anotation of ANimal Genomes
FaaS	Function as a Service
GIS	Geographic information system
GKE	Google Kubernetes Engine
GML	Geography Markup Language
GPU	Graphic Processing Unit
HPC	High Performance Computing
ICGC	International Cancer Genome Consortium

IGN	Instituto Geográfico Nacional (*National Geographic Institute (Spain)*)
IMT	Institut Mines-Télécom
IR	Intermediate representation
JMX	Java Management Extensions
JPA	Java Persistence API
K8s	Kubernetes
LiDAR	Light detection and ranging
ML	Machine Learning
MPI	Message Passing Interface
MTN	Mapa Topográfico Nacional (*National Topographic Map (Spain)*)
NDVI	Normalized difference vegetation index
NUTS	Nomenclature des unités territoriales statistiques (*Nomenclature of Territorial Units for Statistics*)
PRAM	Parallel random-access machine
QoS	Quality of Service
REST	Representational state transfer
RNA	Ribonucleic acid
SAPO	Specified Animal Pathogens Order
SFI	Software Fault Isolation
SIAM	Sistema de Información Agraria de Murcia (*Murcia Agricultural Information System*)
SIGPAC	Sistema de Información Geográfica de parcelas agrícolas (*Spanish Land-parcel identification system*)
SLA	Service Level Agreement
SLO	Service-level objective
SQS	Amazon Simple Queue Service
TCGA	The Cancer Genome Atlas
TIFF	Tagged Image File Format
URV	Universitat Rovira i Virgili
vCPU	Virtual central processing unit
VM	Virtual Machine

1 Executive summary

The deliverable D2.1 "Experiments and Initial Specifications" aims to present in detail the use cases that will be used to validate the outcomes of the CloudButton project. Specifically, this deliverable contains a description of each of the different experiments and an initial assessment on how these workloads can transition to serverless functions. The document also presents an initial specification of the CloudButton architecture that describes how the various components and building blocks fit together and suitably communicate and interact.

The document is structured as follows. Section 2 provides a brief introduction to the deliverable whereas sections 3-5 are devoted to the description of the use cases. Section 6 offers an overall project vision and reviews the state of the art in Serverless Data Analytics. The general objectives of the project are summarized in section 7 while the initial specifications of the architecture are described in section 8. Finally, section 9 contains the conclusions of this document.

2 Introduction

The main goal of the CloudButton project is to create a Serverless Data Analytics Platform that aims to "democratize big data" by overly simplifying the overall life cycle and programming model thanks to serverless technologies. The idea is to tap into stateless functions to enable radically-simpler, more user-friendly data processing systems. The CloudButton platform will seamlessly integrate a Serverless Compute Engine, a Mutable Shared Data Middleware, and new Serverless Cloud Programming Abstractions on top.

Although speed of deployment and ease of use of a serverless computing approach is a key enabler for increased productivity, the new radical approach behind CloudButton must be accompanied by a similar performance, if not better, in comparison to state-of-the-art analysis methods.

Therefore, validating a project like CloudButton and demonstrating its impact requires of a consistent evaluation with strong uses cases and benchmarks. To this end, we target three settings with large data volumes: genomics, metabolomics and geospatial data (LiDAR, satellite). For each of these three domains, we present three different experiments that cover a highly diverse array of analytics: from data-intensive Spark tasks, machine learning methods, genomics pipelines to pixel-based classifiers. With such a diversity of computations, our aim is to ascertain the promising potential of serverless computing for big data analytics.

Together with the description of the use cases, this document also provides the general vision of the project and the initial specifications of the platform architecture. By comparing our project vision with the state of the art, we identify the tradeoffs involved in the design of such a platform (disaggregation between computing and storage, isolation, and simplified scheduling). We also realize that most research works in Serverless Data Analytics combine serverless components with auxiliary dedicated services, a hybrid approach we identify as ServerMix. We explicitly advocate for ServerMix model as a good fit for Data Analytics applications.

3 Metabolomics use case

3.1 Experiments description

3.1.1 Experiments

The key experiment in the metabolomics use case is the metabolite annotation for imaging mass spectrometry according to (Palmer et al, Nature Methods 2017). Imaging mass spectrometry is a technique for detecting metabolites, lipids, and other small molecules in tissue sections from animal models and human samples. Metabolite annotation is finding signals corresponding to molecules in the data, normally of size of 1-100 GB per tissue section. Currently EMBL performs annotation and providing free services to more than 100 users across the world by using an Apache Spark implementation developed in the EU Horizon2020 project METASPACE[1]. The aim of this CloudButton experiment is to develop a serverless alternative to the Apache Spark implementation with the aim to enable interactive analysis, critically increase scalability, avoid current delays due to deployment. At the same time, we would like to simplify the current development process as well as achieve reduction of development and computing costs. This will not only provide a novel cost-effective implementation in a critical field of metabolomics but also will enable novel applications in particular by making possible interactive analysis.

Figure 1 highlights the state of art put forward by METASPACE for spatial metabolomics.

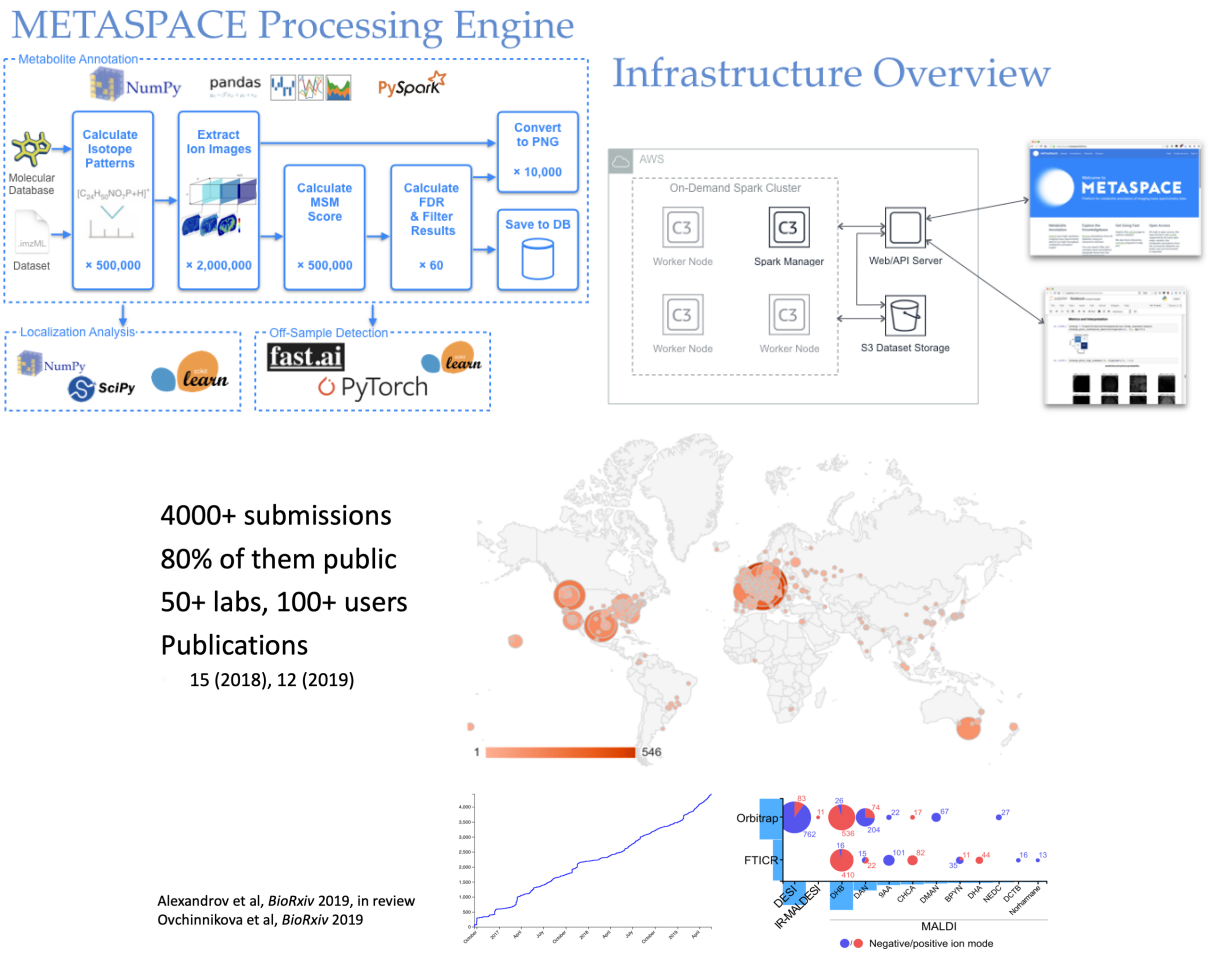

Figure 1: The state of the art of the METASPACE engine for spatial metabolomics: the principle of the engine and the molecular annotation, the current serverful instrastructure of our Apache Spark implementation[2], and the usage of the engine.

[1] `https://metaspace2020.eu`

Table 1: Metabolomics use case datasets

Dataset	Owner	Access	Volume	Growth	Format
ds1	University of Notre Dame	Public	35MiB	N/A	imzML
ds2	EMBL	Public	37MiB	N/A	imzML
ds3	EMBL	Public	571MiB	N/A	imzML
ds4	EMBL	Public	1285MiB	N/A	imzML
ds5	EMBL	Public	104MiB	N/A	imzML
ds6	EMBL	Public	152MiB	N/A	imzML
ds7	EMBL	Public	256MiB	N/A	imzML

Please note that although the provided datasets are of fixed sizes, the number of datasets in the whole METASPACE is growing approximately twice a year. As of July 2019, METASPACE includes over 4800 datasets similar to those included in Table 1. Approximately 10 datasets are being submitted and processed every day.

3.1.2 Data

In CloudButton, for the metabolomics use-case we will use preselected datasets from METASPACE. Currently METASPACE hosts more than 4500 datasets, of them 80% public data, with our team (EMBL) being the largest contributor (18%). For CloudButton, we have pre-selected datasets (and will keep providing new ones throughout the project) that satisfy the requirements of the partners (e.g., no human data), can be shared within the consortium (either public data from us or other submitters or private data from our team), and are representative in terms of size, noise, richness, and other parameters. The datasets are provided in the imzML format, the main open format in the field of imaging mass spectrometry [3]. For parsing the data, we provide a Python library[4].

3.2 Pipelines

An essential part of work in this reporting period was to formulate a Benchmarking Roadmap that includes the criteria for assessment of the future serverless implementation. The Benchmarking Roadmap includes the following aims:

- Match or exceed the performance of current METASPACE (Apache Spark) for typical workloads
- Match or exceed the dataset size limits that METASPACE can handle
- Match or exceed the molecular database limits that METASPACE can handle
- Achieve cost less than or the same amount as METASPACE
- Significantly shorten development time for infrastructure maintenance compared to METASPACE
- Efficiently run the pipeline against small workloads (e.g. ds1, ds2)

We have formulated specific scenarios for performing work in the Metabolomics Use-Case (see Figure 2).

[2] `https://github.com/metaspace2020`
[3] `https://ms-imaging.org/wp/imzml/`
[4] `https://github.com/alexandrovteam/pyimzML`

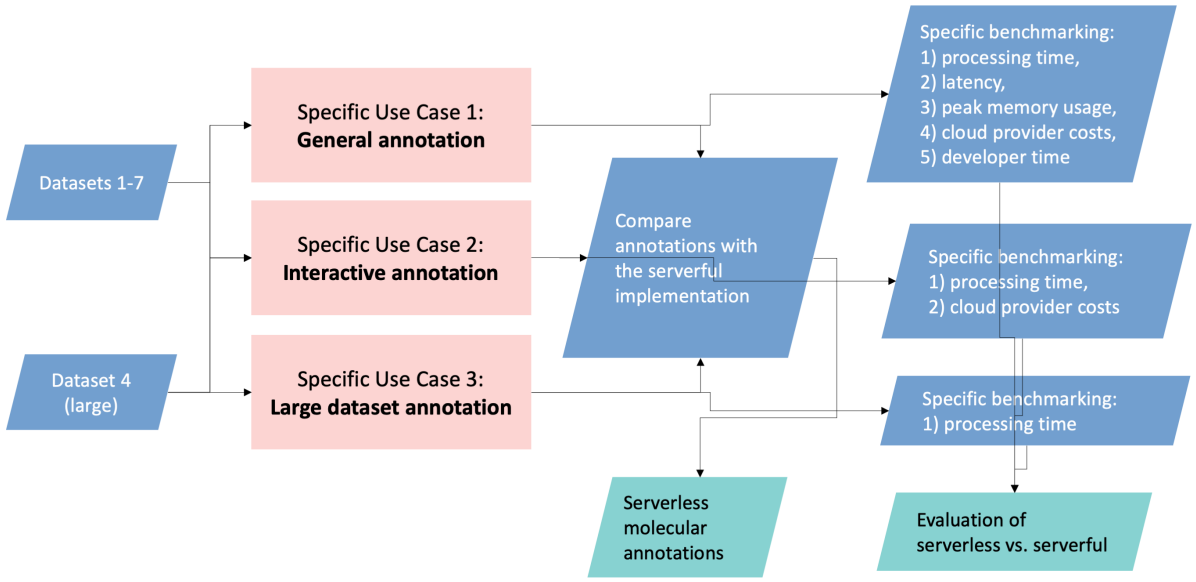

Figure 2: Diagram illustrating the formulated specific use cases for the serverless implementation of the molecular annotation in the Metabolomics Use Case as well as the benchmarking metrics specific for each use case. Please note that although all benchmarking metrics will be applied in all three Specific Use Cases, the current figure highlights those Specific benchmarking which are of particular importance for a regular metabolomics user.

3.2.1 Metabolomics Specific Use Case 1: General annotation

This specific use case covers the typical use of METASPACE for annotation (i.e. finding molecules) of a typical-size spatial metabolomics dataset (e.g. ds3) in non-interactive manner. This is the most common use case in METASPACE. In addition to the cluster execution, we consider also the execution of the engine on lower-specs computers or laptops that is hardly possible with the current serverful implementation.

The key metrics for benchmarking are: 1) total processing time with the goal to achieve a shorter processing time than when using the serverful METASPACE, 2) latency for retrieving all images of target ions with the aim to achieve similar-or-lower time than when using the current METASPACE Python client, 3) peak memory usage on client with the goal to be capable of running on low-spec PC with 8GB ram, 4) cloud provider cost with the goal to achieve similar price or cheaper than the serverful METASPACE (including or excluding cluster start time), 5) developer time with the goal to achieve less annual time required to manage cloud infrastructure than when using the serverful METASPACE.

3.2.2 Metabolomics Specific Use Case 2: Interactive annotation

This specific use case covers an interactive scenario of a scientist working interactively with a dataset of small-to-medium size (e.g. ds1, ds2, ds3). In this use case, the scientist wants to annotate one or a few molecules which were not included in the original molecular database. This use case is currently not implemented in METASPACE due to the long deployment time. In this use case some computational parts are skipped (e.g. FDR calculation) which are either not required or can be sacrificed to provide near-to-instantaneous results that is essential for the interactive analysis.

The main metrics for benchmarking are 1) the total processing time with the goal to perform calculations at least within 60 seconds and 2) the total cost with the goal to have it signficantly cheaper

compared to a full annotation (against a full molecular database).

3.2.3 Metabolomics Specific Use Case 3: Large dataset annotation

This is a use case to test the performance but also corner cases while annotating a large dataset of the size of 1.8 GB after pre-processing (ds4).

The main metric for benchmarking for this specific use case is the performance or the total processing time including the cluster starting time.

3.3 Transition to serverless

The metabolite annotation as it is performed in METASPACE (current Apache Spark implementation) can be highly parallelized and suits perfectly for a serverless implementation. Specifically, annotation of a dataset is performed against a database of molecules (a large list of 10K-100K molecules). Annotation against each molecule can be performed independently that makes this task embarrassingly parallelizable. Moreover, annotation against each molecule by itself involves search against many possible transformations of a molecule into possible ions, each ion being produced in a combinatorial manner (several possible ion adducts times several potential neutral losses) and on its own generating 4 signals. This provides another level of potential for parallelization. The results can be obtained in an independent manner for each molecule and only later needs to be merged to pinpoint potential ambiguity on the level of molecular isomers or isobars.

The transition to the serverless will not only enhance our ability annotating datasets again the current databases, but also will open the search space towards bigger databases (PubChem, including millions of molecules) as well as potential various chemical modifications of the molecules that are know to potentially happening either in the cells or during the ionization while acquiring the data. This will allow us to push the boundaries of the annotation and get a deeper insight into the currently not annotated "dark matter" that currently represents 95% or more of the data.

4 Genomics use case

4.1 Introduction

4.1.1 Current situation

The Pirbright Institute researches and surveils virus diseases of farm animals, and viruses spreading from animals to humans. It receives strategic funding from the Biotechnology and Biological Sciences Research Council (BBSRC), and works to enhance capability to contain, control and eliminate economically and medically important diseases through innovative fundamental and applied bioscience. Although main targets are viruses relevant to livestock, the model adopted is that of One Health – all stages of the viral life cycle and their direct and indirect effect on all hosts and eco-systems are taken into account as a whole. Hence possible interaction with vectors, different reservoirs, possible zoonotic spillage of infection from animal to human hosts, and consequences on agriculture and human development, are all subjects of study. Pirbright acts as Reference Laboratory for several animal diseases, notably Foot-and-Mouth Disease.

Pirbright is equipped with a world-class National Capability high-containment lab, where a number of pathogens falling under SAPO4/ACDP3 classification can be studied. An unusual feature is that several high-throughput sequencers, based on different Illumina technologies, are hosted within containment; this allows scientists to study virus-host interactions and cell response for biological systems involving dangerous pathogens, since biosafety regulations make it difficult for samples to be sent away to external sequencing providers.

4.1.2 Motivation for serverless

Pirbright has recently adopted a centralised model for providing bioinformatics support. It is based on a HPC cluster with 1000 CPUs and 0.5 PB of storage, which is adequate to support the existing user base at the Institute (more than 30 active users at the moment). However bioinformatics is rapidly becoming a central activity, with more and more sequencing protocols and applications being continuously added to the core set of tools that researchers rely upon. Typical applications involving sequencing experiments at Pirbright include: phylogenetic and phylogenomics of viral evolution; exploration of virus-host interaction based on RNA-sequencing and other protocols; immunological studies that focus on the reaction of the host to the virus, which includes the characterisation of the genetic make-up of livestock species.

While local computing power and storage are usually sufficient, some use cases would require a different approach and would greatly benefit from serverless platforms. It is sometimes necessary – in particular when analysing the genetics of livestock viral hosts, or when exploring the system biology of how a host cell reacts to external stimuli – to consider and reprocess large datasets not produced at the Institute in order to compare them with locally available data. Such analyses would be performed on-the-fly for exploratory purposes, with no intention of keeping the results stored locally for a long time. This consideration, and the transient need for large computational resources that are not available locally, would be an ideal use case for the technology explored within CloudButton.

As a first step, the Pirbright pipeline for the analysis of RNA-sequencing data will be ported to serverless technology. Subsequently, Pirbright will provide data coming from its own sequencing machines in containment. This data will be compared with a large number of publicly available similar RNA-sequencing datasets obtained elsewhere. Pirbright will then test the serverless version of the pipeline produced within CloudButton on both its own data and the publicly available data, comparing the results with the ones obtained with the version of the pipeline installed on the local HPC cluster. This comparison, and the ability to externally process datasets that cannot be stored locally, will hopefully demonstrate that the infrastructure provided by CloudButton represents an ideal way of solving analysis bottlenecks.

4.2 Description of datasets

In order to be able to assess such goals we will perform re-analysis of a number of publicly available datasets in the domains of:

1. **Livestock genomics** Typically they are functional studies, focused on understanding if and how the cells of species interesting for farming (cattle, pig, goat, sheep, horse, etc.) work differently from those of model organisms, and how they respond to stress and infection.

2. **Virus-host interaction**. That is obviously a core topic for Pirbright and livestock industry, in that understanding the mechanisms by which some cells are susceptible to a given virus and how others are not is essential to decipher and control animal diseases.

3. **Immunologic studies involving cancer cells**. Several diseases relevant to animals centre on viral infections of components of the immune system, which eventually turn into tumours. Comparing with similar human diseases can shed light on common mechanisms and possible solutions for both animals and humans.

The results obtained from these datasets will then be compared to the ones derived from similar datasets produced internally – Pirbright routinely conducts sequencing experiments in each of the three domains just described. Most of the internal datasets are confidential and cannot in general be shared with the consortium or publicly, but the comparison will provide useful information on the potential offered by the ability to process virtually unlimited amounts of data on serverless platforms.

More details on the datasets selected for each of the three experiments follow.

4.2.1 FAANG (Functional Annotation of ANimal Genomes) datasets

These are publicly accessible datasets [1] produced by the FAANG (Functional Annotation of ANimal Genomes) consortium [2]. They explore genomic and functional information on livestock species (see [3]). Each file considered for those datasets is typically in the range of 10-30GB uncompressed, and during the last 6 months several thousand datasets have been added. The datasets Pirbright will re-process will be a superset of the list in Table 2.

We will reprocess them through our in-house bioinformatics pipeline (an evolution of the RNA-sequencing analysis pipeline used in [4]) re-implemented on the top of the CloudButton toolkit, and compare them to the results obtained by running the pipeline locally on the Pirbright HPC cluster. In parallel we will process similar data sequenced at Pirbright, both locally on the Pirbright HPC cluster and on the CloudButton testbed.

4.2.2 Virus-host interaction from ENA

These are publicly available datasets (from [5]) produced by a variety of sources. They explore several aspects of virus-host interaction (genome evolution, functional changes, technical/protocol-related). Each file considered for those datasets is typically in the range of a few GB uncompressed. The datasets Pirbright will re-process will be a superset of the ones in Table 3.

We will reprocess them through our in-house bioinformatics pipeline (an evolution of the RNA-sequencing analysis pipeline used in [4]) re-implemented on the top of the CloudButton toolkit, and compare them to the results obtained by running the pipeline locally on the Pirbright HPC cluster. In parallel we will process similar data sequenced at Pirbright, both locally on the Pirbright HPC cluster and on the CloudButton testbed.

4.2.3 Immune system cancers from ICGC

These are non-public datasets generated by, or donated to, the International Cancer Genome Consortium [6] that can be obtained by PIs upon nominal request [7]. Each file considered for those datasets is typically in the range of 10-30GB uncompressed. In this case Pirbright will re-process 100 whole-genome sequencing samples from patients with blood cancers (such as Acute Myeloid Leukemia and Chronic Lymphocytic Leukemia) in order to call genomic variants.

Table 2: FAANG datasets

Accession	Title	Species	Files
PRJEB26787	FAANG transcriptome analysis of the horse	Equus caballus	208
PRJEB19268	FAANG data from three boars	Sus scrofa	34
PRJEB24166	Cattle transcriptome of macrophages	Bos taurus	72
PRJEB28219	Bovine small and micro RNA expression atlas	Bos taurus	136
PRJEB19199	RNA sequencing of tissues and cell types from Texel & Scottish Blackface sheep for transcriptome annotation and expression analysis.	Ovis aries	7904 [a]
PRJEB25677	Bovine gene expression atlas	Bos taurus	876
PRJEB23119	Duroc pig macrophages (+/-) LPS	Sus scrofa	84
PRJEB27337	Multi-tissue transcriptome associated with feed efficiency in Nellore cattle	Bos indicus	172
PRJEB28653	Effect of maternal nutrition in late gestation in the blood and skeletal muscle transcriptome of calves	Bos taurus	93
PRJEB23196	Goat gene expression atlas	Capra hircus	336
PRJEB19386	Pig transcriptome and gene expression atlas	Sus scrofa	610
PRJEB27455	Transcriptome profiling of liver and T cells in 4 livestock species by the FAANG pilot project	Capra hircus Gallus gallus Sus scrofa Bos taurus	82
PRJEB25226	Water buffalo gene expression atlas	Bubalus bubalis	4902
PRJEB24920	Horse bone marrow-derived macrophages (+/-) LPS	Equus caballus	84

[a] Only the files containing RNA-sequencing data will be processed.

Table 3: ENA datasets

Accession	Title
ERP104372	Host response to Newcastle disease virus challenge in the Harderian gland transcriptome
ERP004390	The role of viral and host microRNAs in the Aujeszkys disease virus during the infection process
SRP042295	Deep sequencing of the 5′ ends of viral mRNAs from all genome segments transcribed in both human (A549) and mouse (M-1) cells infected with the A/HongKong/1/1968 (H3N2) virus
SRP051574	Analysis of genetic variation and diversity of Rice stripe virus populations
SRP069043	Deep-sea hydrothermal vent virus compensate for microbial metabolism in virus-host interactions
SRP012102	Host RNAs, including transposons, are encapsidated by a eukaryotic single-stranded RNA virus
SRP075180	Multi-Host Evolution of Tobacco etch virus carrying eGFP: Raw sequence reads
SRP076509	Investigating intra-host and intra-herd sequence diversity of foot-and-mouth disease virus
SRP055968	Mutational Bias of Turnip Yellow Mosaic Virus in the Context of Host Anti-viral Gene Silencing
SRP082191	Rice Stripe Virus

We will reprocess them through our in-house bioinformatics pipeline (an evolution of the RNA-sequencing analysis pipeline used in [4]) re-implemented on the top of the CloudButton toolkit, and compare them to the results obtained by running the pipeline locally on the Pirbright HPC cluster. In parallel we will process data for similar animal diseases sequenced at Pirbright, both locally on the Pirbright HPC cluster and on the CloudButton testbed.

4.2.4 Formats

In general all the files listed are in FASTQ format, which stores sequencing reads as a record composed by:

1. A name produced by the machine that uniquely identifies the sequencing read
2. The sequence as a string of nucleotide bases (`A`, `C`, `G`, `T`, and `N` when the base is unknown)
3. A string of qualities. That is a vector of numbers, one per base, which express the probability for the corresponding base to have been incorrectly determined, as estimated by the sequencing machine.

The expected results will be in SAM format [8] which is a way to encode sequencing information aligned to the genome.

4.3 Description of experiments

The experiments aim to demonstrate a number of facts:

- Prove that our workflows can be successfully ported to serverless architectures
- Demonstrate that the serverless implementations can scale up to very large external datasets that we would not necessarily be able to store locally, or we would be unwilling to store locally long-term
- Prove that the scientific results generated by our analysis workflows on the local HPC cluster and on CloudButton platforms are the same
- Demonstrate that the technology developed within CloudButton allows us to gather new biological insights. They would arise from the comparison between our local datasets sequenced at Pirbright and larger datasets publicly available; the latter would require the re-analysis of a large amount of data and hence take a long time, or too much space, if we were to process them locally
- Evaluate efficiency/cost-effectiveness of the cloud solution versus local computing.

4.3.1 Analysis pipeline

The proposed experiments revolve around an existing pipeline for the analysis of RNA-sequencing data which has been developed by the PI and its group along the years. It is based on the GEM mapper [9] and it is an update of the workflow originally used in [4]. The pipeline offers several advantages in terms of accuracy and consistency of the results it produces. In particular, the quality of the alignments generated depends only weakly on the knowledge of a precise annotation for the organism being studied, which is an extremely important feature to have whenever a non-model organism (such as most of the species considered at Pirbright) is studied.

Schematically, the pipeline can be described as follows:

1. A basic alignment block (see Figure 3). It takes as input a sequence of single-end RNA-sequencing reads, and processes them by performing several alignment steps. In particular, each read is:
   (a) First, aligned continuously, from its beginning to its end, to the reference genome

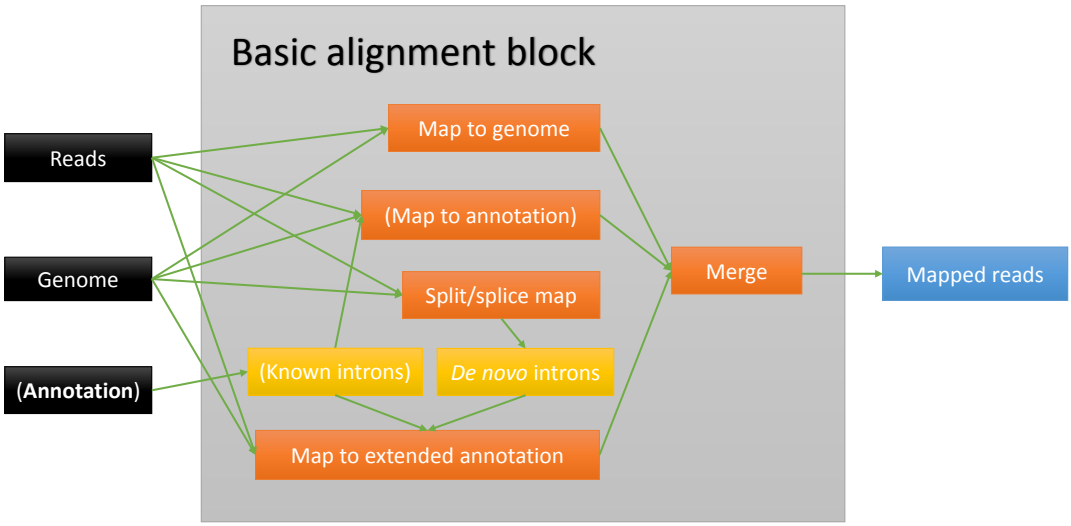

Figure 3: The basic alignment block of the RNA-sequencing alignment pipeline

(b) Second, continuously aligned to the transcripts present in the annotation, if such annotation has been provided

(c) Third, aligned non-continuously, i.e. in two or more chunks, to the reference genome. That is because, due to the biological mechanism of splicing, a continuous sequence in transcript space can originate from different non-contiguous blocks in the genome (the exons) being transcribed together after the intervening sequences (the introns) have been skipped. As a result, some RNA-sequencing reads will not align to the genome as a continuous sequence. This stage of the pipeline is also called de-novo split mapping; it allows the unbiased detection of most introns even though they are not present in the original annotation or the annotation is unknown

(d) Fourth, all the introns present in the annotation (if an annotation has been supplied) and those detected at the previous stage are collected. An extended annotation (i.e., an extended set of transcripts) is generated, and the read is aligned to it

(e) Finally, all the alignment generated at stages (a), (b) and (d) are collected, merged to eliminate redundancies, and scored by their quality (the more errors with respect to the reference sequence and the less unique the sequence, the less the final score).

2. Once the mechanics of the basic alignment block have been described, one can define the full pipeline in terms of the basic block (see Figure 4). Essentially, reads are mapped through the basic block first to a host reference genome (might be human, or that of some livestock species) and then to one or more viral genomes. In principle the alignment to host and virus(es) could be performed together in the same step, but due to practical reasons (such as imprecise knowledge of the virus, or the possible presence of more than one virus) it is often done separately. One way or another alignment to the different references are merged in the end. The same procedure is repeated separately for the two ends of each read if, as it usually happens, one is processing paired-end reads. Once that has been done, the alignments for the two ends are paired and scored.

Finally, the alignment obtained so far are projected onto the annotation, in order to derive counts

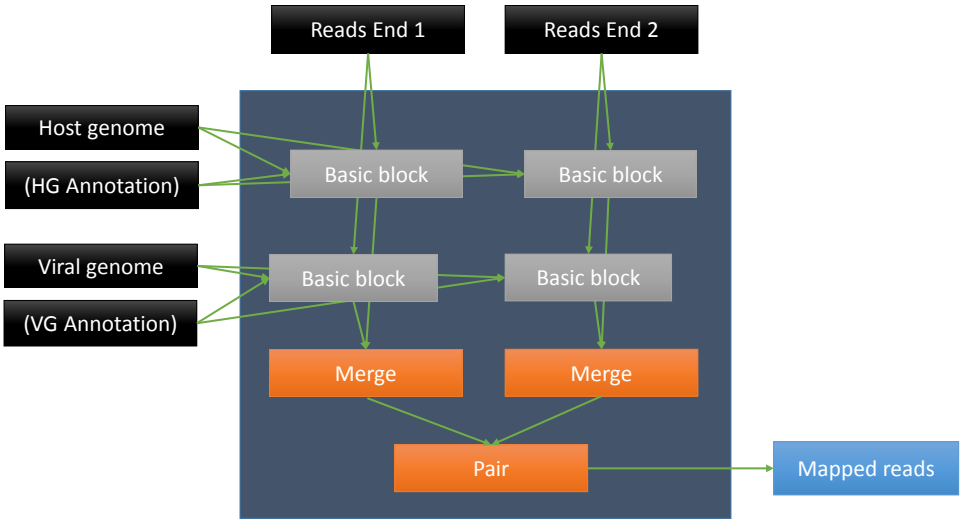

Figure 4: A general view of the pipeline in terms of its basic alignment block

(i.e. expression levels) for each transcript or gene present in the annotation (this step is not shown in the figure).

4.3.2 Transition to serverless

Not all the components of the analysis pipeline translate straightforwardly to serverless architectures. One example is the need for alignment programs based on the Burrows-Wheeler transform, such as the GEM mapper, to generate and store into memory a binary data structure known as an index. The index is a transformed version of the reference genome; thanks to its design, it allows to quickly find exact queries in the reference, ultimately making possible the implementation of fast error-tolerant alignment algorithms for sequencing reads. Mammalian genomes such as the human one are relatively large (about 3 billion nucleotides) and as a result the index for a whole human genome can be bulky, ranging from several hundred MB to several GB, depending on the implementation. While such indices are usually pre-computed once and for all when the analysis is being set up, unfortunately the need to load a bulky index in memory before each alignment run does not cope well with the concept of serverless platform.

However there are many relatively simple ways to circumvent the problem, one of those being that of splitting the reference genome into smaller chunks. That introduces a degree of inefficiency, as one is forced to re-align the same read to all the chunks, and one will need to collect and merge all the alignments at the end of the computation. Nonetheless, the overall gain in computing power acquired by moving to serverless should be amply sufficient to offset the additional cost, and lead to a significant reduction in the overall wall-clock time. Once the problem of the index is solved, aligning sequencing reads to a reference genome can be trivially parallelized – in essence, each read can be aligned independently. We are currently working with Imperial College in order to define a WebAssembly-based strategy and port.

4.3.3 Evaluation criteria

We have identified five main tests that we will perform:

1. **Identical results.** The data processing performed locally on the Pirbright HPC cluster and on the CloudButton-based pipeline should produce the same results. That is a relatively simple

test (identical output files) to be conducted if we suppose that the port will generate identical alignment output. However, the need for an increased degree of parallelism and the porting of the computation to cloud technology might need the introduction of some degree of non-determinism into the algorithm. Here we examine two typical cases:

(a) **Loss of the original order.** Sequencing reads are produced in a certain order, typically dictated by the position at which they originate in the flow cell used by the machine. The order is not by itself strictly significant – in the end the information coming from all the reads is gathered together and distributed according to the localisation of the reads on the genome, through what is called the creation of a read pileup. However, the files being considered are bulky (typically 30GB each uncompressed, but possibly larger), and having to re-sort a large number of them in order to be able to compare them might need the development of some ad-hoc procedure.

(b) **Not identical alignments.** As previously depicted when examining the design of the RNA-sequencing pipeline, the final alignments for each read are selected after several rounds of mapping, as the best in a list of several possible ones. The main criterion for selection is mapping quality – i.e., how much of the sequence of the read is mapped by an alignment stage, and how many errors are needed in order to do so. Some cost function has been implemented that is able to score alignments according to these criteria. However alignments are inherently ambiguous, i.e. there are sometimes several ways to express the same result, and hence it is possible that different alignment stages would produce slightly different results. The problem might be exacerbated by the need to split the computation into separate blocks in order to increase parallelism. In particular, if the computation on the cloud is executed as a number of separate steps whose results are then brought together in the end, the results might be slightly different from those produced by the local pipeline. Again, that would likely need a specific comparison procedure to be developed.

2. **Wall-clock time.** In situations where the analysis could be performed locally (i.e., the data and the intermediates of the computation are not too big to be stored locally), one of the main purposes of porting the analysis to the cloud is typically that of decreasing the overall wall-clock time of the computation – as there is an overhead implied by moving the data to the cloud, there should be clear advantages for doing so, and the desire to reduce the waiting (wall-clock) time is likely to be one of those. Usually one can decrease wall-clock time at the expense of increasing parallelism, but increasing parallelism will eventually lead to a more and more inefficient process (see next point). The typical result will be a sweet spot, i.e. an optimal amount of parallelism that leads to the best possible wall-clock time. Such a sweet spot will need to be found by conducting suitable computational experiments.

3. **Accumulated time, i.e. scalability in computing time.** Even if an analysis can be performed very quickly (i.e., it takes very little wall-clock time to be executed) that does not necessarily mean that the overall computing time required is low. In fact, what happens very often is that the more parallel the analysis is the more inefficient its basic process becomes. That is due to a variety of reasons such as inter-process communication, or the need for merging of partial results, or the need to replicate computation to some extent in order to increase parallelism. So one of the main questions to be answered is: How does the accumulated time taken by the analysis across all the computing agents scale with the size of the data to be analysed and the number of agents? Although in general serverless platforms seem able to provide large amounts of computing power at low costs, depending on the computation model selected the answer to the question might have implications in terms of the viability of the different possible solutions, and in general it would provide important suggestions about how to optimise the degree of parallelism of the computation.

4. **Scalability in size.** The other big advantage provided by porting analysis workflows to the

cloud is given by the possibility of analysing on the cloud datasets that would be too bulky to process locally. One of the key questions to answer is then: How big are such datasets? How do the wall-clock time, the accumulated time and the cost scale on a serverless platform with respect to the size of the dataset considered? Again, the answer to this question is extremely relevant to understand the scope of the technology, its range of applicability and its limitations.

5. **Cost effectiveness.** Once information has been generated for the previous points, it becomes possible to formulate precise guidelines, and assess the precise range in wall-clock time, accumulated time and size for which the porting to serverless of the original pipeline are cost effective. That will be essential to formulate informed decision for the potential and future of the technology.

5 GeoSpatial use case

Nowadays, the use of GeoSpatial technology and their related extensive databases is swiftly growing and it will continue to increase at a fast rate as new applications emerge and new data become available. A bulk of these services and point cloud data applications is devoted to land and urban planning, fire risk assessment mapping, ecosystem modeling, water use mapping, or infrastructure network design.

The new developments of the GeoSpatial technology have made available vast amounts of high resolution imagery and cloud point data which offer the potential to cover huge extensions of territory and allow to analyze in detail many variables simultaneously for each geographical coordinate

However, storage and memory limitations of conventional IT systems only allow analyses for either low to medium spatial resolution data calculations in large territories or highly detailed results in smaller areas, but do not allow integrating at the same time different data sources and reaching high accuracy over large regions. In particular, huge LiDAR datasets from large areas cannot be handled appropriately by standard software. This fact limits its usability due to the high cost associated to data storage and GeoSpatial information extraction of forest metrics, for example. In addition, these calculations frequently entail long periods of processing time [10].

The serverless platform developed in the CloudButton project will be used to develop high-resolution hybrid land-cover mapping and 3D fuel modeling for forest fire risk assessment combining satellite imagery and LiDAR data. Besides, we will map water use footprint combining satellite data and crop GeoSpatial information.

5.1 Experiments description

This use case comprises three experiments:

- **EX1: High-resolution hybrid land-cover mapping**: Through multi-temporal imagery from Sentinel 2 MSI sensor and LiDAR point cloud data from PNOA (National Plan of Aerial Ortophography), we will carry out a hybrid land-cover classification of Peninsular Spain by means of advanced analytical remote sensing fusion techniques. In particular, supervised classification of image objects, OBIA, for 12 Sentinel-2 images, one per each month of the year, will be performed through CloudButton and, after that, the land-cover discrimination outcome will be refined by LiDAR-derived 3D metrics using this toolkit. The expected result will be a clear improvement of map quality in comparison with current CLC mapping [11]. The same method will be applied using other computer facilities to test the same experiment in a selected sample of small areas of Spain, including photo-interpretation verification procedures. Cloud versus local processing comparison will be used to assess the incremental performance improvement (Figure 10).

- **EX2: 3D fuel mapping for forest risk assessment:**. We will use the same source of satellite and LiDAR big data described in the first experiment. We will generate a fuel map of three large regions of Peninsular Spain, which include protected areas, using remote sensing fusion techniques and following an object-based classification model. Segmented and supervised classification of Sentinel-2 images and 3D LiDAR-derived metrics as input data shrub and tree canopy structure of two dates (reference years) will be used as input data in spatial fire risk modeling, that also benefits from high resolution elevation models and other topographic input variables. The expected result is a high-resolution forest fire risk map that is useful for decision-making in forest planning and management (a meaningful increase in map accuracy and better land-cover discrimination). Also, we will make a performance comparison assessment of the results obtained through CloudButton toolkit and conventional software and computer abilities (Figure 10).

- **EX3: Mapping water use footprint:**. By comparing water use estimates obtained from two different databases over extended regions of irrigated crop fields, we will map differences in the water use footprint of irrigated arable lands in representative large areas of Peninsular Spain.

On the one side, with high-resolution NDVI index (derived from the same satellite imagery than previous experiments), we will identify actual irrigated crop areas and will estimate water consumption using multi-date imagery data along the growing season. Mapping these variations along a certain period of time with frequent updates will be only possible due to the continuous update of open-access databases and the utilization of the CloudButton capabilities. These results will be combined with LiDAR data from PNOA to discard tree vegetation areas and refine the model. On the other side, we will estimate and map water consumption indicators considering the officially declared and georeferenced irrigated arable land area which is available from SIGPAC, the Agricultural Common Policy open access database and specific correction factors (irrigated land area and crop water consumption volume). The comparison of both results will identify non-coincident areas which would help to monitor water use efficiency and funding resource allocation.

5.1.1 Data

Regarding **data volumes**, at following it is provided a description of the different datasets involved in the three experiments of the GeoSpatial use case:

- **Administrative regions:** This data contains administrative regions that will conform the experimentation zone. The administrative limits will be obtained from the official cartography available at the National Geographic Institute (CNIG-IGN), namely: municipal, provincial and autonomous precincts and the municipal, provincial and regional limit lines registered in the Database of Jurisdictional Limits of Spain (BDLJE). The reference geodetic system will be ETRS89 for the Iberian Peninsula and the Balearic Islands and REGCAN95, compatible with WGS84, for the Canary Islands. This geometry of the boundary lines has, in the best of cases and with the exception of those boundary lines that have been redefined on the ground, the precision of the 1/25,000 scale, conditioned by the topographical methods and instruments used to obtain them and subsequent cartographic edition. This data set is available in GML format through the WFS and ATOM download services and in shapefile format in the CNIG Download Center [12].

- **Sentinel-2.** This satellite is part of the constellation of satellites belonging to the Copernicus program. Copernicus is an initiative of the European Commission in collaboration with the European Space Agency (ESA) and is described as the most ambitious program developed to date in terms of Earth observation, allowing great temporal and spatial precision. The instruments included are multispectral cameras with spatial resolutions of 10 m^2, 20 m^2 and 60 m^2 per pixel that capture in 13 different bands with wavelengths from 443 nm to 2190 nm. Due to this experiment is referring to the calculation of the spectral indices, it is essential to use multispectral information, which, in this case, are only capable of capturing Sentinel-2 and Sentinel-3 OLCI (it is noteworthy that it is the Sentinel-2 satellite sensors that offer the highest spatial resolution). The Copernicus Sentinel-2 mission comprises a constellation of two orbiting satellites placed in the same synchronous solar orbit, in 180° phases with each other. Its objective is to monitor the variability in the conditions of the earth's surface and its wide strip width (290 km) together with its high revision time (10 days at the equator with a satellite, and 5 days with 2 satellites in cloudless conditions that result in 2-3 days in latitudes media) monitoring the changes in the surface of the Earth. The coverage limits are between latitudes 56° south and 84° north. Data can be downloaded from Copernicus Open Access Hub [13]. Regarding the volume of data, it must be taken into account that it is the data referred to the entire national territory. The casuistry of these experiments means that the data must be downloaded with a frequency of between 15 days and a month, so that the storage space must be multiplied, as well as the processing of the different layers.

- **SIGPAC.** SIGPAC is the spatial information contained in the Geographic Information System of the Common Agricultural Policy (SIGPAC in Spanish), which is the responsibility of the

Figure 5: Footprint size of a Sentinel-2 tile

Ministry of Agriculture, Fisheries and Food. This cartography allows to identify the use that is given to the agricultural and livestock parcels throughout the Spanish territory. The data must be requested from the competent administration, since they are not available for direct download. It is public information and can be consulted in the SIGPAC Viewer (without identification, due to the data protection law, of the owner/s).

Figure 6: SIGPAC screenshot

- **PNOA - LiDAR point cloud.** LiDAR technology, acronym for *Laser Imaging Detection and Ranging*, is a remote-sensing method based on discrete laser pulses and time intervals aimed to measure ranges (variables distances) between an air-borne transmitter and targets on the Earth surface. These laser pulses, combined with triangulation information, generate datasets of the

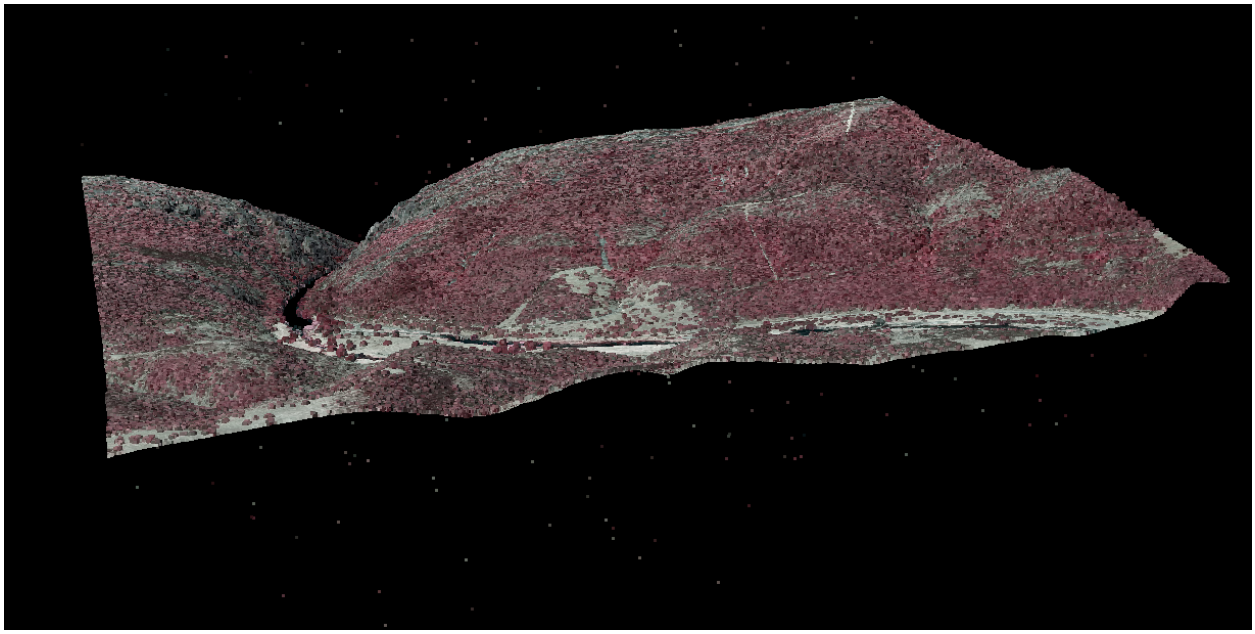

Figure 7: LiDAR screenshot through plas.io

Earth surface and its characteristics called LiDAR point cloud data. PNOA offers noise-free point clouds categorized according to the American Society for Photogrammetry and Remote Sensing (ASPRS) classification with a horizontal resolution of 0.5 points/m^2 and a vertical accuracy of 0.5 m. These point clouds were obtained from several LiDAR-equiped flights, with swathes covering the whole Spanish extension during two time periods: the first period from 2008 to 2015 and the second period, which started in 2015, is still recording data. Thus, PNOA is a program currently in progress that will provide updated data in the close future. LiDAR point clouds and are usually compressed in .las or .laz format and, because their size can be extremely large, its computation can be time consuming.

- **Meteorological information.** Data of the different meteorological stations located in the study area. The data of all weather stations use common parameters such as maximum daily temperature, minimum daily temperature, temperature, speed, wind speed, etc. The sources used to obtain this information are:
  - Agricultural Information Service of Murcia (SIAM) [14]
  - Spanish Meteorological Agency (AEMET) [15]
- **Irrigation communities.** In order to know the data referring to the irrigation of crops and to know the geographical extension of each community of irrigators, they will be resorted to. This information can be offered in different formats depending on each community and must adapt to current regulations. It will only be done once in the whole process and it will consist of defining a polygon of each one of the areas that each community manages.
- **Natura 2000**. It is defined by the EU as *a network of core breeding and resting sites for rare and threatened species, and some rare natural habitat types which are protected in their own right. It stretches across all 28 EU countries, both on land and at sea.* Since the forest fuel experiment will be located in several Spanish National Parks, the extent of the forest fuel experiment will be coincident with the boundaries of some of these protected areas. Such information is available through the platform of the Ministry for the Ecological Transition. [16]

Table 4: GeoSpatial use case datasets

Dataset	Owner	Access	Volume	Format
Administrative areas	National Geographic Institute (CNIG-IGN)	Public	31 MB for all of Spain region	Shapefile[a]
Sentinel-2	European Commission	Public	300 GB	TIFF[b]
SIGPAC	Ministry of Agriculture, Fisheries and Food	Public	200 GB	Shapefile
LiDAR	National Geographic Institute (CNIG-IGN)	Public	8 TB	laz[c] and las[d]
Meteorologic Information	Agrarian information service of the Murcia Region (SIAM) and Spanish Agency of Meteorology (AEMET)	Public	9 KB for one meteorological station and for one day	CSV[e]
Irrigation communities	Irrigation community of Murcia Region	On demand	65 KB for one year	Shapefile
Natura 2000	European Comission - Environment	On demand	80 MB for Spain	Geopackage[f]

[a]It is a digital vector storage format for storing geometric location and associated attribute information
[b]It is a computer file format for storing raster graphics images
[c]LAZ is a compressed light detection and localization (LIDAR) data format often used to transfer large amounts of LIDAR data.
[d]It is an industry standard binary format for storing air LIDAR data..
[e]It is a delimited text file that uses a comma to separate values
[f]Standard of the Open Geospatial Consortium

5.1.2 Processes

This subsection shows a list of geoprocesses (processes with geospatial information) involved in the three experiments. All these geoprocesses are developed using Python [17]. The software used to model them will be QGIS [18], using for this the PyQGIS [19] framework. For each one of the experiments of this use case, Figures 10 and 11 show the inputs, the geoprocesses and outputs involved

Download Sentinel-2 images

For this geoprocess a Python script will be generated. This script allows to select data by dates. Later it will be supported by the MTN (National Topographic Map) sheets for the definition of the areas of previous study, as it shows in the figure 8 .

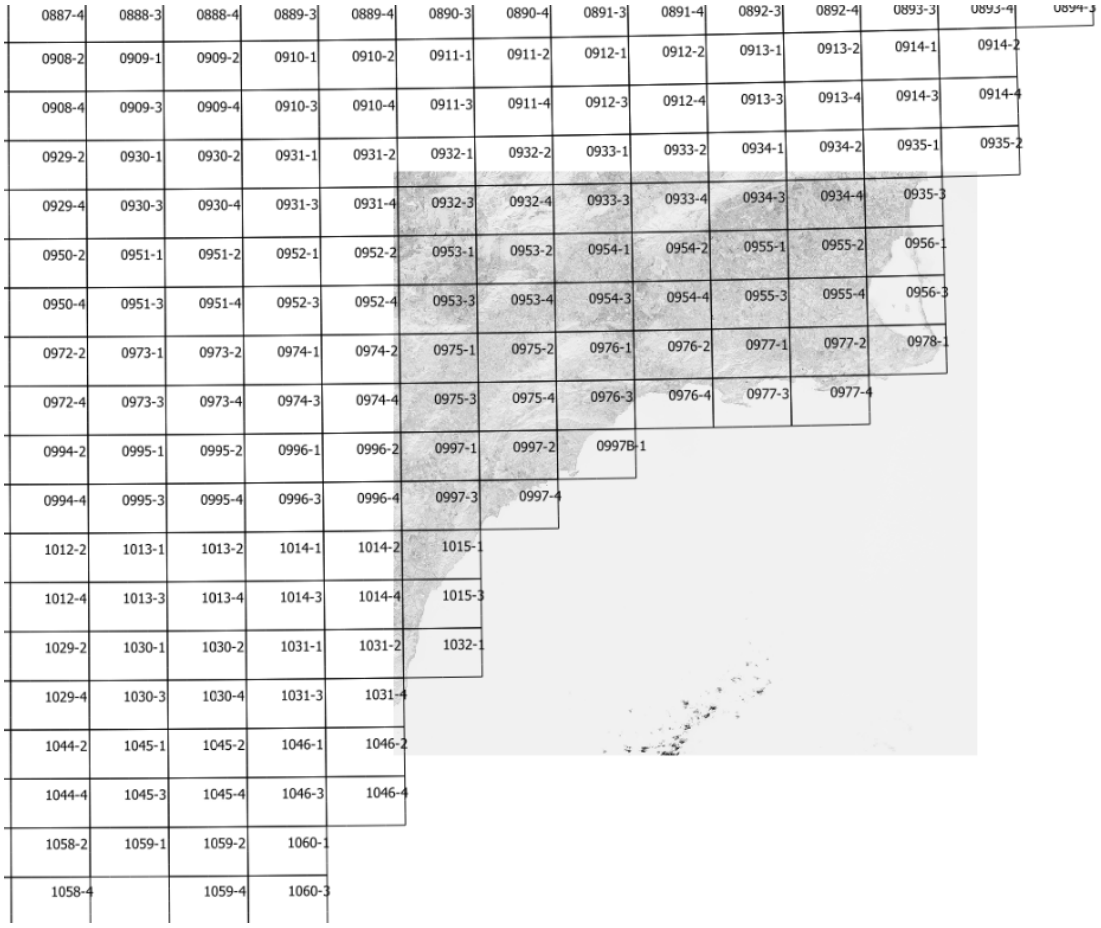

Figure 8: MTN sheet over Sentinel-2 tile

Radiometric correction

Once downloaded the satellite images, it is necessary to carry out a process of corrections (defects in the image produced in some cases, by faults of the sensor itself or alterations in the movement of the platform that transports it, together with errors of the interaction of the atmosphere in the process of information acquisition), which consists of correcting the values of the pixels of the image by applying various techniques and specific analyzes.

Geometric correction

The objective is to adjust the image perfectly to the position it should occupy in the territory. For that there are different techniques such as the use of Digital Elevation Models (DEM). This step is of vital importance since it allows the comparison between satellite images or other spatial information referring to the same territory.

Application of corrections

The downloaded Sentinel-2 images will present two possible treatment levels: L1C and L2A. Those referred to the first case need to be subjected to an atmospheric correction ("Top of Atmosphere") to be at the same level as the L2A images, which already include this correction.

Object-based images analysis (OBIA)
This technique analyses the set of pixels of an image and provides an output equivalent or similar to the so-called objects. The process of OBIA analysis consists of the following steps:

- Segmentation: This step separates an image into primary objects, which are the units which will used to classify the image.
- Classification: This step consists of defining the attribute for each objects. For example: object 1 –> buildings, object 2 –> water bodies, etc..To improve this classification, is mandatory to use a learning algorithm. In this case it is going to use Random Forest algorithm [20]. For this, the training areas will be defined based on the sheets that MTN described previously, giving a greater homogeneity to the whole. The selection of these training areas will be based on geographic criteria that are capable of covering the maximum of the territory's variability in terms of the diversity of elements that may occur in it.

LiDAR products and metrics
It refers to the different products and metrics derived from the LiDAR point cloud:

- Digital Elevation Model (DEM). Raster image that represents the continuous variation of topography in the space, without including heights associated with biological (i.e. trees or shrubs) or anthropic (i.e. buildings) elements.
- Digital Surface Model (DSM). Raster image that represents the continuous variation of the surface, including biological and anthropic elements.
- Canopy Height Model (CHM). Raster image that represents the height variation of biological elements (i.e. trees or shrubs). It is obtained by the difference of an DEM and an DSM of biological objects.
- Tree Canopy Height (TCH). Raster image that represents the mean tree height in each pixel.
- Shrub Canopy Height (SCH). Raster image that represents the mean shrub height in each pixel.
- Fraction of Canopy Cover (FCC). Percentage of the area covered by the vertical projection of the vegetation in each pixel of a raster image.
- Tree Canopy Cover fraction (TCC). Percentage of the area covered by the vertical projection of the tree canopy in each pixel of a raster image. The value of this indicator strongly depends on the search radius.
- Shrub Canopy Cover fraction (SCC). Percentage of the area covered by the vertical projection of the shrub canopy in each pixel of a raster image. The value of this indicator strongly depends on the search radius.
- Slope. Raster image that represents the maximum altitudinal variation in each pixel of a raster image in relation to the surrounding pixels. The units can be radians, degrees or percentage.
- Aspect. Raster image that represents the angle that a slope faces with respect to the north. The units can be radians, degrees or percentage.
- Standard deviation. Standard deviation of the vegetation height in each pixel of a raster image.
- Count. Mean number of returns in each pixel of a raster image.
- Maximum. Maximum vegetation height in each pixel of a raster image.
- Minimum. Minimum vegetation height in each pixel o a raster image.

Select region
It will be the first geoprocess that will allow to obtain the study area.

Tiling
Due to the large size of the satellite images and, especially, of the uncompressed LiDAR point clouds, it is necessary to split the spatial information into squared blocks of a smaller size and weight than the original source to be able to use them. This process is called tiling and it is usually used when improved performance and reduced computation times are required (often by means of parallelizing scripts).

Note here that this use case is amenable to serververless parallelization because of being embarassingly parallel.

Stackering
Raster images can be composed of several bands (i.e. Sentinel-2 images) which can be interpreted as an array of three dimensions, x, y and z (with as many z-coordinates as raster bands). Stackering is a technique that stores data in this form to keep diverse information under the same spatial coordinates x and y within a single object or file. Consequently, analyses that require the use of different raster bands referred to the same spatial location, such as segmentation, can be carried out in stacked files.

Mosaic
The download of Sentinel-2 images do not exactly match the administrative limits of any region, since the data is collected in different passes. Therefore, to complete an administrative area such as the Region of Murcia, several images must be collected (this is the mosaic process).
The result of this geoprocess is a set of multispectral bands, i.e, mosaics are obtained from each of the bands of the satellite's channels

Mask
Through this action, the previous mosaic is trimmed according to the limits of the desired administrative area.
The result of this process is a multispectral image of the perfectly bounded study area.
In addition, as many bands as there are channels are obtained from the satellite (13). These bands cover the geographical area that has been delimited.

Merge
In those cases where the information does not have the required extent (either because it has been tiled or because the final product has a larger extent than the input data), the *Merge* process can spatially join all parts in a single file to get the desired extent. The *Merge* process relies on the coordinates of each part to find the adjacent components and join them in order to build an object of the desired extent. In addition, this process can be applied to both spatial information data types: vector and raster.

Remote sensing indexes
Different algorithms will be used to know the parameters needed for each experiment. This technique is based on map algebra (figure 9), which allows to cross the pixel values of different bands located in the same geographical space from previously defined algorithms.

Select irrigation crops parcels
Based on the SIGPAC data, the parcels declared as agricultural and, among them, those of irrigated crops will be discriminated.

Zonal statistics
Using the geometry of the plots and the data corresponding to the different obtained remote sensing

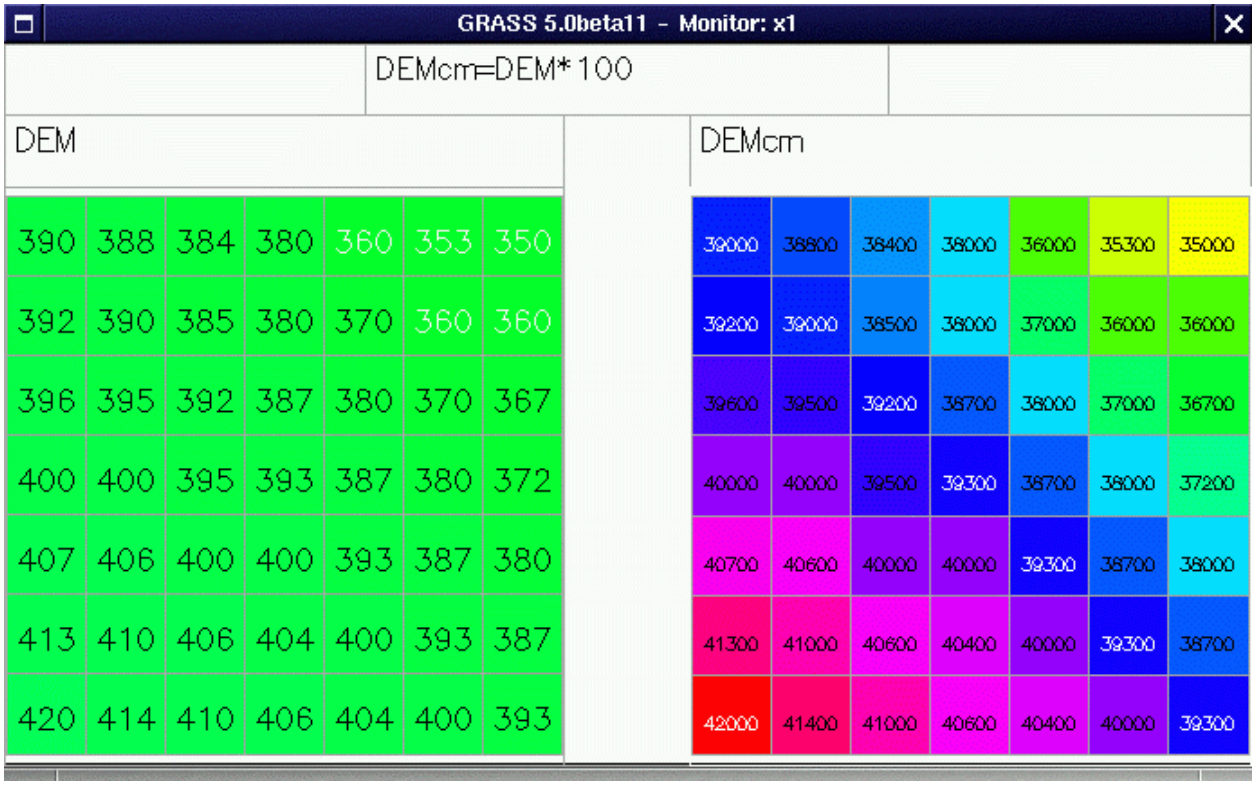

Figure 9: Map Algebra example

indexes, the statistical values of the crop corresponding to each one of the plots can be calculated.

Temperature calculation
The temperatures are modeled from point data measured in the meteorological stations of the networks previously described. For this, a mixed method is used in which Machine Learning is used with the Splines and Random Forest algorithms.

Humidity
Humidity will be spatially modelled with the *Inverse Distance Weighted* (IDW) algorithm from weather stations distributed through peninsular Spain. As with temperature, humidity is a required variable for forest fuel modelling.

Topology
A series of topological standards will be established to be implemented in the PostGIS geographic database, in order to avoid distortions in vector cartography (mainly SIGPAC), such as duplicity of borders, repeated plots, etc., that cause errors in the data processing and gross errors in the calculation of surfaces.

5.1.3 Outputs

The results of these geoprocesses are described next:

1. High-resolution hybrid land-cover mapping (1st experiment)
    - Land cover map: segmented classification of the type of coverture of the land based on SIOSE land cover map.

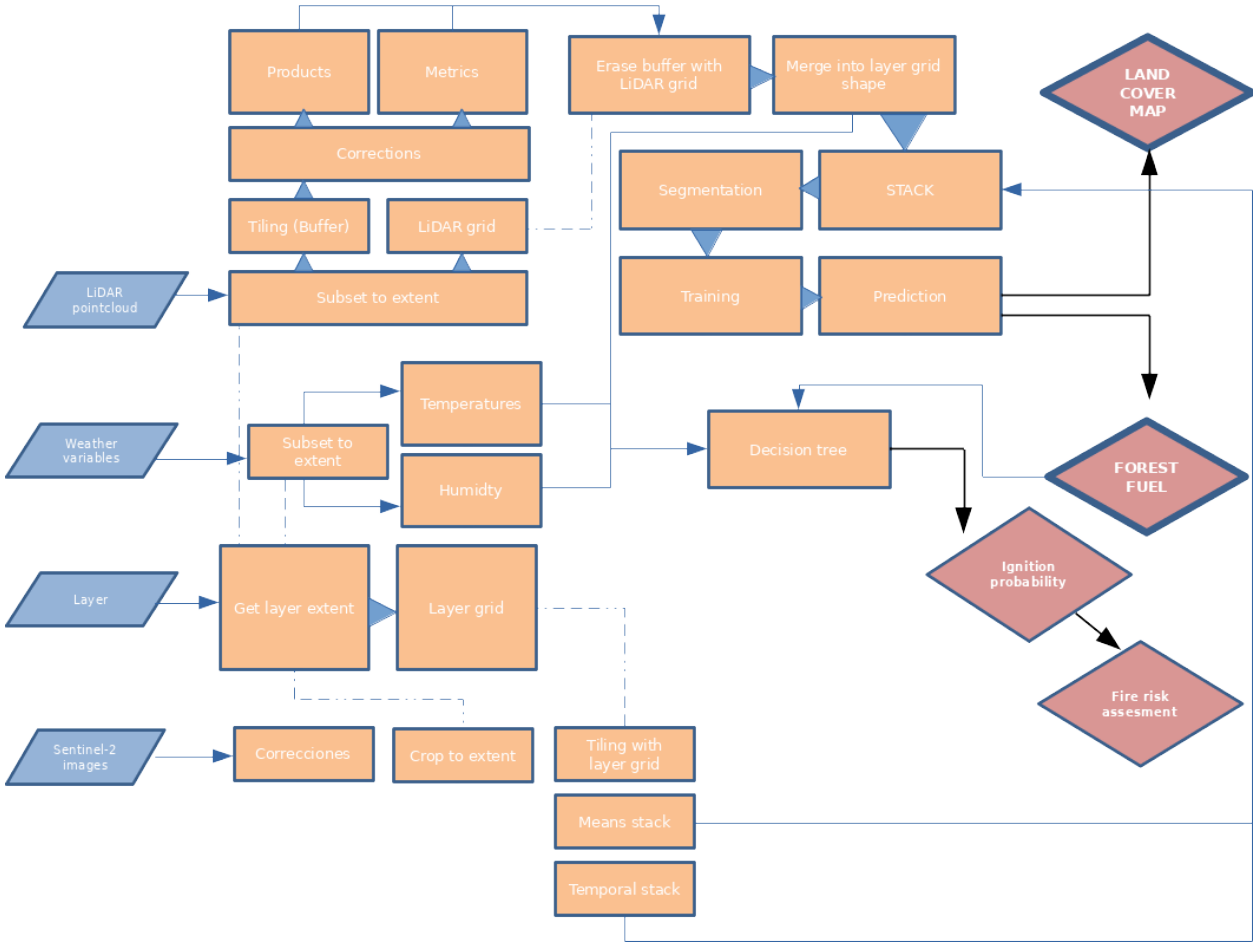

Figure 10: 1st and 2nd experiment pipelines

2. 3D fuel mapping for forest risk assessment (2nd experiment)

   - Forest fuel map: one of the essential variables to estimate fire risk is forest fuel, which can be modeled using spectral information obtained from satellite-derived data, spectral indices, and clouds of LiDAR points. The most common approach is to assign a categorical value to each vegetation class based on the canopy cover type, including that of the understory when it is present, and the height of the vegetation.
   - Ignition probability map: ignition probability is the probability of a spark causing a fire when it lands on a certain forest fuel. Ignition probability modeling requires information regarding temperature, relative humidity, inclination of slopes, orientation of slopes and vegetation class.
   - Fire risk assessment map: the assessment of fire risk requires a multidisciplinary and heterogeneous approach that should consider both natural (i.e. probability of ignition) and socio-economic agents (i.e. prevention policies).

3. Mapping water use footprint (3rd experiment):

   - Crop conditions: it is the first result that is obtained from the calculation of the remote sensing indexes, allowing knowing the state of the crops on a given date.
   - Irrigation crops and changes in land use: crossing the data with LiDAR, you can know the land uses, as well as the possibility of discriminating between herbaceous and arboreal crops.

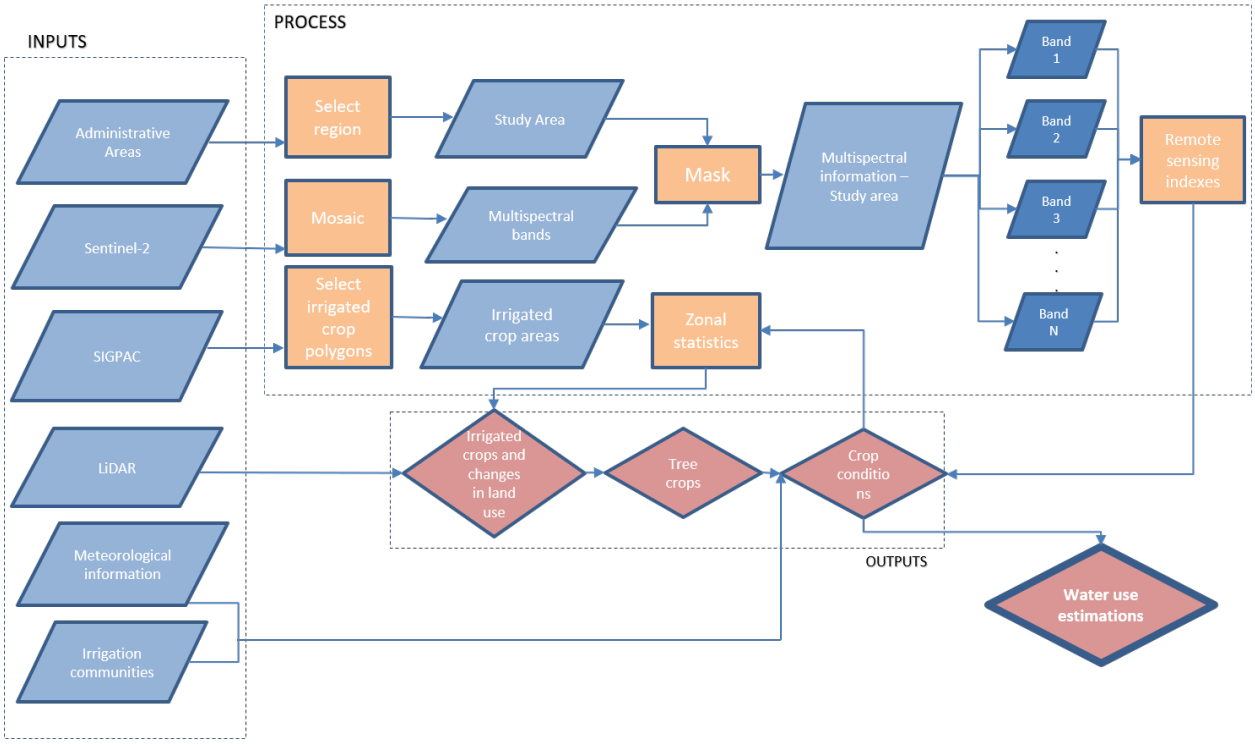

Figure 11: 3rd experiment pipelines

- Tree crops: only the part corresponding to tree crops can be extracted from the previous result.
- Water consumption estimations (final result): as a final result, crossing all the results obtained along the process with the data from the community of irrigators and meteorological data sources, an estimation of the water consumption necessary for all the crops of an Autonomous Community can be carried out like that of the Region of Murcia, where agriculture has a great importance in its economy.

The processes developed for these three experiments will be repeated every 15 days during the first year and depending on the results obtained, the required periodicity will be determined. Therefore, the information referred to Sentinel-2 will be downloaded twice a month in that first year and at least once a month in the following.

5.2 Pipelines

Figure 10 depicts a scheme of Experiments 1 and 2 processes. In these two experiments there are four type of inputs: LiDAR point clouds, weather variables, a layer, and Sentinel-2 images. They are defined by the user in each case depending on the final product desired. Usually, only temporal Sentinel-2 images, a LiDAR point cloud, and the study area extent will be required. In other cases, additional variables will be required. For instance, when ignition probability and fire risk assessment want to be computed, local temperatures and humidity data in each point will be needed along with spatial information regarding the location of roads, cities and leisure areas.

The layer, which determines the study area extent, will frame the data analyses. It serves two objectives: first, to subset the spatial data to the desired extent and, second, to define the spatial area of the tiles. Once the extent of the study area has been defined, LiDAR point clouds and Sentinel-2 imagery start being processed. On the one hand, LiDAR point clouds subsets are tiled with a buffer and corrected to compute different product and metrics. Then, the buffers are removed and the LiDAR tiles are merged up and co-registered, (that is, stacked) with the Sentinel-2 imagery. On the other hand, the Sentinel-2 imagery are corrected, cropped to the study area extent, and tiled.

As a result, two different stacks are produced: first, a Means stack containing annual temperature average values and, second, a Temporal stack containing all the temperature values of the time period defined by the user. With the Means stack and a DSM generated with the LiDAR point cloud data, a segmentation of data is computed. The resulting objects of the segmentation are classified according to their shape indexes and different spatial statistics (*e.g.* maximum, minimum, average) obtained from the Temporal stack of Sentinel-2 imagery and several LiDAR products and metrics.

The resulting output is a Land Cover map wherein forest cover can be extracted and used as an input to generate a Forest Fuel map through a decision tree process. The Forest Fuel map requires two additional products apart from forest cover: an ignition probability map and a forest risk assessment map. In order to generate these two products, information regarding the weather conditions are needed (temperature and humidity). Apart from temperature and humidity, slope, aspect, and shadow percentage obtained from predefined tables are also required to generate the ignition probability map. Besides, the fire risk assessment map uses distances from roads, cities and leisure areas as variables. Both products are computed with regression kriging and simple kriging approaches.

Figure 11 depicts a scheme of Experiment 3 processes. We can see that six are the inputs of the geoprocesses described previously for the 3rd experiment: *Administrative Areas* is used to select the region to work with it; *Sentinel-2* images are used to extract their multispectral bands; Then, using these two inputs is possible to calculate some indexes which are used later to estimate the water requirements of the crops, using also for this calculation the data obtained from *SIGPAC* (which is used to estimate the kind of crop in the selected area) and the meteorological data obtained from the SIAM sensors network. The rest of inputs (*LIDAR* and *Irrigation communities*) are used to evaluate the water estimations that the experiment provides as output.

5.3 Transition to serverless

5.3.1 Data partition

With regard to geospatial data parallelization, tiling appears as a key process, since each spatial point is related with its sorroundings. Indeed, this phenomenon is called spatial autocorrelation and it's well described in literature.

As far as the 1st and 2nd experiments are concerned, there are two different tiling approaches depending on the data type: Sentinel-2 images and LiDAR point clouds.

- **Sentinel-2 images**. A user-defined grid is created for each image, whose spacing size must be a divisor of the total extent of the image (the extent of the image appears in its metadata). Afterwards, the grid is used to crop the image in as many tiles as grid cells exist, naming them after their top left corner coordinates.

- **LiDAR point clouds**. Point clouds are downloaded already tiled and with top left corner coordinates specified in their names, although their spacing is still too big to ensure reasonable computation times. In addition, a buffer distance is required for the computation of certain geoprocesses as the slope and aspect algorithms, to avoid the edge effect. For all the above reasons, it is necessary to create tiles with smaller spacing and buffer distance. After, once all geoprocesses and statistics are computed, buffers are removed from the tile, by means of a grid that only considers the spacing value used in the tiling process and the coordinates of each LiDAR tile. Next step consists in merging all the information generated from LiDAR tiles up to the Sentinel-2 images tile spacing, enabling them to be co-registered through the stacking process.

Both tiling approaches depend on the creation of a user-defined grid, which implies that the size of tiling can be different in each case. In both cases, the size of the tile should be constant through the whole process.

As indicated above, for the third experiment of the GeoSpatial use case, two sources of data are used to estimate water consumption in a given area: Sentinel-2 raster layers provided by the ESA and the MTN sheets provided by the IGN. The first one divides the territory into zones called Tiles, while

the second classifies the geographical area into sheets. The different geoprocesses involved in the experiment take as input a tile, a sheet or both, so that to calculate the water consumption of a region, we must take into account all the tiles/sheets that comprise this region. This is fundamental when establishing the parallelization architecture, since to estimate the water consumption of a region, the geoprocesses can be launched in parallel for each tile/sheet.

5.3.2 Parallel tasks for geoprocesses

Different parallel tasks have been identified so far for each one of the experiments developed in the GeoSpatial use case. Each one of these tasks is associated to a set of different technical specifications in sequential and implements some of the geoprocesses described above. These tasks are summarized in Tables 5 and 6, and tested on test datasets in a single machine. In the case of the two first experiments, these tests have been carried out on a virtual machine with Ubuntu OS: Intel (R) Core (TM) i7-7200k CPU 4.20 GHz, 2 cores, 10405 MB of RAM and a storage capacity of 100 GB. In the case of the 3rd experiment, the tests have been carried out in a physical machine with Windows 10 64-bits OS: Intel(R) Core(TM) i7-7700HQ CPU @ 2.80GHz, 16GB of RAM, 4 cores, 8 threads and a storage capacity of 1 TB.

Parallel task	**Sequential time**	**RAM**	**Space**	**Geoprocess**
Subset LiDAR point clouds	0.12 sec	< 60 MB	500 MB	Get subset of LiDAR tiles based on their names
Tile LiDAR point clouds	147 sec	< 60 MB	500 MB	Compute tiling process over 15 tiles of 2000 m
Remove LiDAR overlay points	432 sec	< 5 MB	1.8 GB	Computed over 1470 tiles of 250 m
Remove noise from LiDAR point cloud	4867 sec	< 5 MB	1.8 GB	Removing noise from 1470 tiles of 250 m
Surfaces from LiDAR	3088 sec	1.2 MB	1 GB	Compute DEM, DSM, CHM, TCH and SCH for 1470 tiles of 250 m
Slope and Aspect	143.7 sec	20.1 KB	37.68 MB	Computing slope and aspect. 1920 tiles of 250 m.
FFC to folders	141 sec	< 5MB	1.8 GB	Compute TCC, SCC and FCC over 1470 tiles of 250 m
LiDAR metrics	293 sec	< 5MB	1.8 GB	Compute stdandard deviation, minimum height, maximum height and the count of returns over 1470 tiles of 250 m
Mask LiDAR files	180 sec	75.6 KB	108.5 MB	Eliminates buffers from 1470 rasters of 250 m
Create images tiles	1575 sec sec	600 MB	2.4 GB	Create 2000 m spacing tiles
Mosaic LiDAR rasters	3.63 sec	75.6 KB	108.5 MB	Merge LiDAR tiles up to images tiles spacing (1470 tiles of 250 m)

Table 5: Parallel tasks identified in the 1st and 2nd experiments, it continues in the next page

Parallel task	**Sequential time**	**RAM**	**Space**	**Geoprocess**
Temporal means images	9.76 sec	78.6 KB	442.1 MB	Create temporal means of bands 2, 3, 4 and 8 over 5760 tiles
Temporal stacking	23.9 sec	78.6 KB	442.1 MB	Create temporal stacks of bands 2, 3, 4 and 8 over 5760 tiles
Stack spatial information	2.73 sec	313 KB	963 MB	Co-register all spatial information based on its coordinates (617 tiles)
Stack for segmentation	0.329 sec	313 KB	226.9 MB	Co-register temporal means and DSM on its coordinates for segmentation (376 tiles)
Segmentation	5.71 sec	4.3 MB	68.6 MB	Segmentation of 16 tiles
Getting stats segments	824 sec	235 KB	3.8 MB	Getting stats from the temporal and lidar information stack (64 tiles)
Join training and stats	28.8 sec	800 KB	3.8 MB	Joining statistical information to training dataset
Land Cover Map (prediction)	TBD	568 KB	9.4 MB	None
Forest Fuel classification	4.70 sec	488 KB	6.9 MB	Decision tree over vegetation classes of a Land Cover classification (16 tiles)
Ignition probability	TBD	TBD	TBD	None
Wildfire risk	TBD	TBD	TBD	None

Table 5: Parallel tasks identified in the 1st and 2nd experiments

Parallel task	Sequential time	RAM	Space	Geoprocess
Downloading meteorological data from the AEMET by station	1 sec.	19 MB	10 KB	none
Radiometric correction for each tile downloaded from Sentinel-2 (results by tile)	2011 sec.	6 GB	Aprox 907 MB/tile	radiometric correction
Geometric correction for each tile downloaded from Sentinel-2 (results by tile)	2011 sec.	6 GB	Aprox 907 MB/tile	geometric correction
Obtain the study areas from an array of identifiers and Obtain the rasters from the 4th and 8th bands of each study areas (results by sheet)	135 sec.	50 MB	352 KB	Select region, study area and mosaic
Obtain agricultural areas from a list of study areas (results by sheet)	0.3 sec.	43 MB	215 KB	Select region and study area
Obtain irrigated areas from a list of study areas (results by sheet)	0.3 sec	43 MB	215 KB	Select region, study area and select irrigation crop parcels
Calculation of the NDVI of each one of the study areas	14 sec.	1 GB	350 KB	Mask, multispectral information and remote sensing indexes
Cultivation areas and changes in land use (results by sheet)	0.8 sec.	45 MB	88 KB	Remote sensing indexes and temperature calculation
Arboreal crops identification (results by sheet)	0.3 sec.	45 MB	44 KB	Select region and study area

Table 6: Parallel tasks identified in the 3rd experiment

6 Project Vision and State of the Art

In 2017, two relevant research articles [21] [22] demonstrated that Serverless Function as a Service (FaaS) could sustain massively parallel computations in the Cloud. The former presented ExCamera, providing on-the-fly video encoding over thousands of Amazon Lambda Functions. ExCamera proved to be 60% faster and 6x cheaper than using VM instances. Another relevant work is the "Occupy the Cloud" paper, showcasing simple MapReduce jobs executed over Lambda Functions in their PyWren prototype. In this case, PyWren was 17% slower than PySpark running on r3.xlarge VM instances. But the authors claimed that the simplicity of configuration and inherent elasticity of Lambdas justified the performance penalty. In this paper, they did not compare the costs between their lambdas and the VM experiments.

Both research works demonstrated the enormous potential of serverless data analytics. The two major advantages are clearly the simplicity and the massive scalability and elasticity of the model. On the one hand, the scaling, deployment, provisioning, fault-tolerance, and monitoring of functions is delegated to the cloud provider. Furthermore, the programming simplicity of functions clearly paves the way to a smooth Cloud transition. On the other hand, the transparent and almost infinite elasticity boosts the analysis of huge data volumes accessible in Cloud Object Stores.

But Serverless Computing is nowadays not adequate for many data analytics tasks due to two fundamental problems: high cost and lack of performance compared to Cluster Computing or even VMs running Spark. Two recent articles have outlined the major limitations of the Serverless model in general: [23] and [24]. In the latter, they review the performance and cost of several data analytics applications. They show that MapReduce Sort (100TB) was 1% faster than VMs, but costing 15% higher; Linear Algebra (NumPyWren) was 3x slower than an MPI implementation in a dedicated cluster, but only valid for large problem sizes; and Machine Learning pipelines (Cirrus) were 3x-5x faster than VM instances, but up to 7x higher total cost.

Furthermore, existing approaches must rely on auxiliary Serverful services to circumvent the limitations of the stateless serverless model. PyWren uses Amazon S3 for storage, coordination, and communication, Locus uses Redis ElastiCache In-memory system, ExCamera relies on a external rendezvous and communication service, or Cirrus relies on disaggregated in-memory servers.

In this deliverable, we identify as **ServerMix**, the hybrid applications combining Serverless and Serverful services. We will review how most related work can be classified under the ServerMix umbrella term. We will also show how a ServerMix application can still provide transparent provisioning to users, while ensuring fault-tolerance, and optimizing both cost and performance.

We will first describe the existing design tradeoffs involved in creating ServerMix data analytics systems. We will show that it is possible to modify core principles like disaggregation, isolation, and simple scheduling to increase performance. But these changes may compromise the elasticity, security, and even the pricing model and fast startup time of Serverless Functions. For example:

1. Relaxing disaggregation: Using locality in memory or function placement could boost performance. Moving from a serverless data-shipping model to benefit from computation close to the data could easily achieve performance improvements. But disaggregation is the fundamentall pillar of elasticity in the Cloud.

2. Relaxing isolation: Co-locating related functions (namespaces) in the same containers, and reusing containers could also improve performance. Providing direct communication between functions could also facilitate shared replicated memory models. Leveraging lightweight containers or even using language-level constructs would also reduce cold starts and boost inter-function communications. But isolation is the basis for multi-tenancy and security.

3. Flexible QoS and scheduling: To ensure SLAs it could be possible to implement more sophisticated scheduling algorithms that can reserve resources or entire nodes to functions, or even execute them in specialized hardware like GPUs. But simple location-agnostic scheduling is the basis for reduced start times and increased cloud resource utilization.

It is clear that these approaches would obtain significant performance improvements. But, depending on the changes, such systems would be closer to Cluster computing or Cloud dedicated resources (Serverful model) and they could be in direct opposition to the essence of serverless computing.

In fact, we claim in this deliverable that the so-called "limitations" of the serverless model are indeed its defining traits. When applications should require aggregation (computation close to the data), relaxing isolation (co-location, direct communication), or tunable scheduling (predictable performance, hardware acceleration) a suitable solution is to build a ServerMix.

We will demonstrate how the ServerMix model may be a good fit for Data Analytics applications. We advocate for (i) Smart Scheduling as a mechanism for providing transparent provisioning to applications while optimizing the cost-performance tuple in the Cloud, (ii) Fine-grained State Disaggregation thanks to Serverless Mutable Consistent State services, and (ii) Lightweight and Polyglot Serverful Isolation: novel lightweight Serverful FaaS runtimes based on WebAssembly as universal multi-language substrate.

CloudButton aims at the widest possible traction with the community. According to [25] only 20% of enterprise workloads are in the public cloud. While more enterprise workloads are expected to move to the public cloud in the coming years, it is reasonable to assume that a very large part of them will either be deployed on premises or in a, so called, *hybrid cloud* deployment. For enterprise, the defining transformation happening now is the cloud-native movement powered by the portable containers (e.g., Docker) and scalable intent-based container orchestrators (e.g., Kubernetes). A hybrid cloud typically comprises a number of federated clusters on premises where the physical infrastructure is owned by the enterprise, complemented with an outreach to the public cloud(s). In the hybrid cloud, especially, in its massive swaths of capacity, which is fully controlled by the enterprise itself, the aforementioned principles of disaggregation, isolation, and simple scheduling can be relaxed to a larger degree than what would be possible with the public cloud alone. In particular, a specialized data intensive FaaS oriented scheduling enforcing specific KPI goals can be deployed alongside default Kubernetes scheduler, disaggregation can be mitigated by local caching, specialized runtimes relaxing isolation for internal users can be explored, and locality can be improved through the cluster-wide and node-specific caching.

In CloudButton we intend to devise a flexible, agile and extensible architecture that would allow to leverage the benefits of the hybrid cloud as well as adhering to the defining traits mentioned above in a more strict manner when deployed in a public cloud. More details about our architectural approach to achieve this twofold goal are provided in D3.1 (Section 4).

6.1 Tradeoffs of Serverless Computing

We will first outline three important tradeoffs of the Serverless model. Previous proposals [23] [24] hinted that these tradeoffs could be relaxed to obtain more performance, but we will explain why this could compromise essential aspects of the serverless model.

According to Amazon AWS [26], the four defining features of a serverless system are: no server management, flexible scaling, pay for value, and automated high availability. No server management implies that users do not need to provision or maintain any servers. Flexible scaling entails that the application can be scaled automatically through units of consumption (throughput, memory) rather than units of individual servers. Pay for value is to pay for the use of consumption units rather than server units. And finally, automated high availability ensures that the system must provide built-in availability and fault tolerance.

Let's see how the tradeoffs may affect some of the defining features of serverless computing.

Dissaggregation is an essential element of cloud computing and the cornerstone for elasticity and scalability. Disaggregation means separating the stateless computation services from the stateful storage services, so that they can scale in an independent manner. And modern high speed networks permit sub millisecond latencies between the compute and storage layers (even allowing memory disaggregation like in InfiniSwap [27]). Serverless FaaS follows the same principle: stateless function computation which can rely on scalable disaggregated storage services like Amazon S3.

Three important papers already propose to relax disaggregation to increase performance. Hellerstein et al. [23] even consider that one of the limitations of Serverless Computing is its Data Shipping architecture that requires moving data to the functions. They propose the so called "Fluid Code and Data Placement" where the infrastructure should be able to physically colocate certain code and data. In a similar trend, [28] also proposed Fluid Multi-Resource Disaggregation which in fact consisted of allowing movement (i.e.,fluidity) between physical resources to enhance proximity and thus performance. Finally, [24], propose to allow functions to be co-located in the same VMs, and to share data among them. At least, they recognize that this can go against the spirit of serverless computing, since it would reduce the flexibility of cloud providers to place cloud functions, and thus resource utilization.

But data locality and computation close to the data are old friends that are always at odds with scalability and multi-tenancy. Computation close to the data, like Active Storage has not been adopted in the Cloud because of its lack of scalability. Recent works like [29] demonstrate that active storage computations may provoke resource contention and interference with the storage service. In a multi-tenant scenario, computations from one user could harm the storage service to other users. Finally, storage services are more difficult to scale up and down since they may need data movements like resharding.

Ensuring locality for serverless functions would mean for example locating them in the same node, and enabling fast shared memory between them. This would clearly improve performance in applications such as machine learning, OpenMP, and PRAM algorithms. But this decision can compromise flexible scaling, elasticity and fault-tolerance, and it would clearly require reservation of dedicated resources to the experiment. This would require changes in the scheduling of functions that would also compromise the model.

Simple Scheduling is another essential pillar of Serverless Computing. Cloud providers can ensure Quality of Service (QoS) and SLAs (Service Level Agreements) to different tenants by scheduling the reserved resources and bill them appropriately. The goal of cloud scheduling algorithms is to maximize the utilization of the Cloud resources while matching the requirements of the different tenants.

In Serverless Faas, the tenant only specifies the memory size, while the function execution time is severely limited. These reduced constraints considerably simplify the best effort scheduling of functions, helping Cloud providers to maximize utilization and even place them in old equipment. It is also well known by users that functions may suffer interferences from other services and functions, and even suffer cold starts.

To improve performance, a clear candidate would be to work on more sophisticated scheduling algorithms that would guarantee dedicated resources (i.e. hardware acceleration, GPUs), enforce locality, or optimize the placement of functions. In this case, tenants could detail resource requirements for their experiments or applications, and the scheduler would optimize the resource allocation for each tenant.

Again, the three aforementioned papers, propose similar ideas to provide predictable performance in serverless settings. In [28] they propose Fine-grained Live Orchestration, involving sophisticated allocation of resources as well as orchestrating the fluidity of resources (like the decision to move computation between nodes). In [23], they advocate for heterogeneous hardware support for functions where developers could specify their requirements in DSLs, and the cloud providers would then calculate the most cost-effective matchings to meet user SLOs. This would guarantee the use of specialized hardware for functions. In [24], they also support this claim for serverless computing to support hardware heterogeneity. They propose that serverless could embrace multiple instance types (with prices according to hardware specs), or that cloud providers may select the hardware automatically depending on the code (like GPU hardware for CUDA code and TPU hardware for TensorFlow code).

Again, tuning scheduling to improve performance could compromise flexible scaling, elasticity and even complicate the high utilization of resources to the Cloud provider. Putting more constraints on scheduling of functions could then lead to a change in the pricing model of such kind of functions

requiring dedicated resources.

Isolation is another pillar of multi-tenant clouds services. Isolation guarantees security, privacy, and fair use of resources among the different tenants. Existing multi-tenant resource allocation mechanisms are based either on per-VM allocations or hard rate limits that rely on uniform workloads to obtain high utilization. Isolation is also an important mechanism to avoid interferences between the different tenants accessing the shared resources.

Let us review what previous works are proposing here:

In [28], they propose a concept named coordinated isolation, in which isolation is extended across multiple servers (where the functions of the same user are executed). This goes beyond the current model where isolation is guaranteed for single function executions. They clearly see this concept as the basis for fluid dissaggregation and for fine grained orchestration of resources.

In [23], they are proposing two ways of relaxing isolation: (i) the previously explained fluid code and data placement, and (ii) direct communication and addressing. In particular, they claim that Serverless stymies distributed computing due to this lack of direct communication among entities. They advocate for long-Running, Addressable Virtual Agents offering public "ports" to other entities.

To increase performance, and reduce the overheads of creating and configuring isolated environments (cold starts), there are numerous efforts to offer lightweight isolation mechanisms. Several examples are Amazon Firecracker, Google gVisor [30], CloudFlare Workers with WebAssembly [31] or optimized containers like SOCK [32].

Another technique to increase performance is to relax isolation and co-locate functions in the the same VMs or containers. Or even to provide very lightweight language-level constructs to reuse containers as much as possible. This can make sense for functions belonging to the same tenant, since it would heavily reduce cold starts and function compositions.

Finally, it could also be possible to enable direct communication between functions of the same tenant. In this case, direct communication would permit a variety of distributed communication models, allowing for example the construction of replicated shared memory between functions.

But again, relaxing isolation may compromise security and privacy, but also resource management and scheduling between different tenants. And things can be even more entangled. Locality (relaxing disaggregation) could require relaxed isolation with in turn may compromise elasticity and fault tolerance.

As a summary, we refer to Figure 12 as a global view of the overall tradeoffs. Disaggregation, isolation, and simplified scheduling guarantee respectively the flexible scaling, multi-tenancy, and reduced startup time of functions.

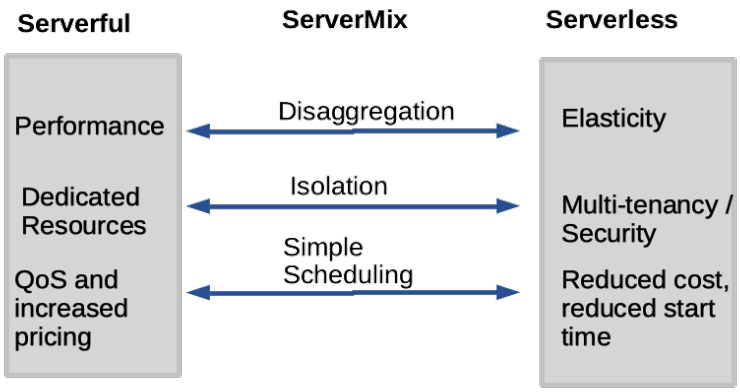

Figure 12: Tradeoffs

Reducing disaggregation means locality and computation close to the data, which can entail clear performance improvements. But then we will be sacrificing elasticity and complicating scheduling

and also probably reducing isolation. The more you move to the left, the closer you are to serverful computing or running VMs or clusters in the Cloud.

With isolation the effect is similar. Since isolation is the key to multi-tenancy, sacrificing isolation completely leads you to dedicated resources. In your dedicated VMs, containers, or clusters (serverful), you can run functions very fast without caring about sandboxing and security. But this also entails more complex scheduling and different pricing.

Finally, simple scheduling and agnostic function placement is also inherent to serverless computing. But if you require Quality of Service, SLAs or specialized hardware, the scheduling and resource allocation gets more complex. Again, moved to the extreme, you end up in serverful settings which already exist (dedicated resources, VMs, or clusters). Serverless cost efficiency is further discussed in Deliverable D3.1.

Probably the most interesting conclusion of this figure is the region in the middle, which we call ServerMix computing. The zone in the middle involves applications that will be constructed combining both serverless and serverful computing models. In fact, as we will review in the next section, many existing so-called serverless applications may be considered servermix according to our definition.

6.2 State of the Art

We can consider that most research in Serverless Data Analytics can be classified as ServerMix, since we will see that those systems combine both Serveless and Serverful components.

In general, most systems rely on a external (serverful) provisioner component that is in charge of launching and orchestrating serverless functions using the APIs of the Cloud provider. Sometimes the provisioner is called coordinator or scheduler, but the task is the same, orchestrate functions and provide fault-tolerance.

As we will see, some systems require additional serverful components to overcome the limitations of the Serverless model. For example, disaggregated in-memory systems like Redis ElastiCache to overcome the throughput and speed of disk-based storage like S3 (see Locus). Or even external communication or coordination services to enable the communication among functions through a disaggregated intermediary (see ExCamera).

Table 7: ServerMix applications

Systems	**Components**	
	Serverful	***Serverless***
Locus	Scheduler, Redis	Lambda Functions, S3
PyWren	Scheduler	Lambda Functions, S3
IBM PyWren	Scheduler	IBM Cloud Functions, COS, RabbitMQ
ExCamera	Coordinator, Rendezvous	Lambda Functions, S3
Flint	Scheduler	Lambda Functions, S3, SQS
NumPyWren	Provisioner	Lambda Functions, S3
Cirrus	Scheduler, Parameter Servers	Lambda Functions, S3

PyWren [22] is a proof of concept that MapReduce tasks can be executed as serverless functions. PyWren is combining a serverful function scheduler (client application) that orchestrates functions and launch the required mappers as Lambda functions. PyWren uses S3 for storing intermediate

results, but also for function coordination since they actively pull S3 to detect that all results have been uploaded.

IBM-PyWren [33] is a PyWren derived project which adapts and extends PyWren for IBM Cloud services. It includes a a number of new features, like broader MapReduce support, automatic data discovery and partitioning, integration with notebooks, and simple function composition, among others. IBM-PyWren also boosts PyWren performance by using RabbitMQ to avoid unnecessary polling to the Object Storage service.

ExCamera [21] performs digital video encoding leveraging the parallelism of thousands of Lambda functions. Again, ExCamera is using serverless components (Lambda functions, S3) and serverful ones (coordinator, rendezvous). In this case, apart from a coordinator/scheduler component that launches functions, they also need a rendezvous service for communication and coordination between functions.

Flint [34] implements a serverless version of the PySpark MapReduce framework. It is similar to PyWren in the use of an external scheduler (serverful) to orchestrate lambda functions, and S3 to store intermediate results. But Flint also uses SQS service for coordination.

The same pattern is again followed by NumPyWren Linear Algebra library [35] since it is based in PyWren. Again it relies on a scheduler and S3 for storing results.

Cirrus Serverless Machine Learning project [24] is again an hybrid combining some serverful components (parameter servers, scheduler) and serverless ones (Lambda, S3).

Finally, the most recent and related ServerMix related work is Locus [36]. They target one of the clear limitations of the serverless stateles model, shuffling in Map Reduce, or sorting data is not ideally suited for stateless functions. In [22] they already executed some shuffles in PyWren using 30 Redis ElastiCache servers.This proved to be an expensive solution.

In Locus, they try to provide a hybrid solution (ServerMix) that takes into account both cost and performance. In this line they minimize the use of the expensive Redis in-memory system and complements it with the cheaper S3 service.

We did not include SAND [37] in the list of ServerMix systems since it is mainly proposing a new runtime for Serverless Faas. They present an alternative High Performance Serverless Platform and they compare it with OpenWhisk. To improve performance they basically relax disaggregation introducing locality in their runtime (local message bus) and relax isolation through application level sandboxing. They do not analyze or evaluate how these changes could affect the scalability, elasticity or security of their solution.

Recent works also outline the need for novel serverless services providing flexible disaggregated storage to serverless functions. This is the case of Pocket [38] ephemeral storage service, which provides auto-scaling and pay-per-use as a service to serverless functions.

In the same line, [24] propose as a future challenge the creation of High-performance, affordable, transparently provisioned storage. They propose two services: Serverless Ephemeral Storage and Serverless Durable Storage that should provide micro-second latencies, multi-tenant workloads, fault-tolerance, auto-scalable and providing transparent provisioning. They suggest that with a shared in-memory service, any memory not used by one serverless application can be allocated to another in a multi-tenant scenario. They also explain that existing services like Redis or MemCached do not fulfill the aforementioned requirements. In this line, they can both be considered Serverful due to their explicit provisioning, and dedicated resources to one tenant.

Another interesting alternative could be to use FaaS Serverless Orchestration services to coordinate/provision the Data Analytics applications. In [39] we recently compared three different Serverless Orchestration mechanisms: Amazon Step Functions [40], Azure Durable Functions [41], and IBM Composer [42]. But our conclusion was that they are still young projects that are not designed to orchestrate massively parallel function computations (like MapReduce dataflows). IBM Composer recently improved their support for parallel function execution and in a future it could become a serious alternative.

As we can see, if Data Analytics applications could leverage serverless orchestration and serverless in-memory services, this table could change and many projects could avoid the use of serverful

entities. This would clearly create more native applications with consistent fault-tolerance models entirely provided in the cloud.

The reality is that in fact several providers are also transitioning to hybrid models combining serverless and serverful concepts. We already see how some providers like Azure already provide ServerMix FaaS services like Azure Premium Plan for Functions that run on dedicated machines while abstracting the provisioning phase. And Azure is even allowing users to pre-warm functions to reduce their cold starts.

Hybrid Cloud technologies are also accelerating the combination of serverless and serverful components. In this line, the deployment of Kubernetes Clusters in different Clouds can even overcome portability issues among providers. For example, Amazon is offering Amazon EKS Elastic Container Service for Kubernetes [43], Google is offering Google GKE (Google Kubernets Engine) [44], and Azure is offering Azure AKS (Azure Kubernetes Service) [45]. But the three of them (EKS, GKE, AKS) cannot be considered serverless since tthey require some management, scaling, and provisioning of the cluster.

A very interesting recent trend is the emergence of the so-called serverless container services like AWS Fargate [46], Azure Container Instances [47], and Google Cloud Run [48]. These services reduce the complexity of managing and deploying Kubernetes clusters in the Cloud. They offer serverless features like flexible automated scaling and pay-per-use, while they still require some provisioning and configuration of the required resources and scaling limits.

Serverless container services are interesting for long-running jobs like batch data analytics, while they offer more control over the applications thanks to the use of containers instead of functions. In any case, they can be very suitable for stateless scalable applications where the services can scale-out easily adding or removing container instances. In this case, the user establishes a simple CPU or memory threshold and the service is responsible of monitoring, load balancing, and instance creation and removal. But if the service or application is more complex (like a stateful storage component) their potential use is very limited or requires more user intervention.

For example, AWS Fargate [46] offers two models: Fargate launch type and EC2 launch type. The former is more serverless and requires less configuration, while the latter gives you more control but also more responsibility and different pricing models. An analogous thing occurs with Google: Cloud Run vs Cloud Run on GKE model, the former is automated and uses standard vCPUs, while the latter enables you to select hardware requirements and configure and manage your cluster.

An important open source project related to serverless containers services is CNCF's KNative [49]. KNative is backed by Google, IBM and RedHat among others, and it simplifies the creation of serverless containers over Kubernetes Clusters. Knative simplifies the complexity of Kubernetes and Istio service mesh components, and it creates a promising substrate for both PaaS and FaaS applications. Google Cloud Run is based on Knative and IBM Cloud is also offering seamless Knative integration in their Kubernetes services.

As a final conclusion, we foresee that the simplicity of the serverless model will gain traction among users, so many new offerings may emerge in the next years blurring the borders between both Serverless and Serverful models. As we will explain later, serverless container services may become an interesting architecture for servermix deployments.

7 General objectives

This work has been produced in the context of the European Research project "CloudButton: Serverless Data Analytics" [50]. CloudButton is a three-year european research project (2019-2022) including key industrial partners like IBM, RedHat, and Atos, academic partners like Imperial College London, Institute Mines Telecom, and Universitat Rovira i Virgili, and use cases like EMBL, Pirbright Institute, Answare and Fundacion Matrix. To demonstrate the impact of the project, we target two settings with large data volumes: bioinformatics (genomics, metabolomics) and geospatial data (LiDAR, satellital).

This project is inspired by the "Occupy the Cloud paper" [22] in which they refer to the following sentence from a professor of computer graphics at UC Berkeley : “Why is there no cloud button?” He outlined how his students simply wish they could easily “push a button” and have their code – existing, optimized, single-machine code – running on the cloud.”

Our main goal is to create CloudButton: a Serverless Data Analytics Platform. CloudButton will “democratize big data” by overly simplifying the overall life cycle and programming model. To this end, CloudButton will convert and deploy existing code to the Cloud thanks to serverless technologies.

While serverless computing is considered to be Cloud Computing’s next step, the challenges in moving toward serverless still remain one of the major obstacles for a wider adoption. Adapting existing applications to serverless is not a trivial process, but usually requires application redesign, new code development and learning new APIs. The same is true for developing new applications over serverless, as this also requires learning new deployment and development skills.

In CloudButton we would like to take this further and explore more options to ease serverless integration into existing code and frameworks. We plan to address the challenge on how to transition existing code and frameworks to serverless settings without the painful process of starting from scratch and or learning new skills. We explore the challenges involved to integrate existing applications and frameworks with serverless. Instead of redesigning existing applications, our goal is to explore how to scale certain code parts without rewriting and modifying existing applications. Understanding the impact of such approach will facilitate user adoption and allow applications to automatically perform flexible scaling on demand. Our final goal is to improve and simplify the user experience as much as possible.

To achieve these ambitious objectives, CloudButton defines the following goals:

- Create a High Performance Serverless Compute Engine for Big Data: This is the foundational technology for the CloudButton platform that must overcome the current limitations of existing serverless platforms. In particular, it includes extensions to i) support stateful and highly performant execution of serverless tasks, ii) optimized elasticity and operations management of functions thanks to new locality aware scheduling algorithms, iii) efficient QoS management of containers that host serverless functions, and iv) a Serverless Execution Framework supporting typical dataflow models.

- Support for Mutable Shared Data in Serverless Computing: To simplify the transitioning from sequential to (massively-)parallel code, we will design of a new middleware that allows to quickly spawn and share mutable data structures in a serverless computing platform. Our Mutable Shared Data middleware will i) offer an easy-to-use programming framework to add state to serverless computing, ii) provide dynamic data replication and tunable consistency to match the performance requirements of serverless data analytics, and iii) integrate this framework to an in-memory data grids for performance.

- Design novel Serverless Cloud Programming Abstractions: The overall objective is to provide a new programming model for serverless cloud infrastructures that can express a wide range of existing data-intensive applications with minimal changes. The programming model should at the same time, i) preserve the benefits of a serverless execution model in terms of resource efficiency, performance, scalability and fault tolerance, ii) explicit support for stateful functions

in applications, while offering guarantees with respect to the consistency and durability of the state.

7.1 High Performance Serverless Run-time

In many real-life cloud scenarios, enterprise workloads cannot be easily moved to a centralized public cloud due to the cost, regulation, latency and bandwidth or a combination of these factors. This forces enterprises to adopt hybrid cloud.

However, current serverless frameworks are centralized. Out of the box, they do not know how to leverage computational capacity available in multiple locations. Another gap in the current serverless computing engines implementations is their obliviousness to serverless functions QoS. In fact, serverless functions are treated uniformly even though performance and other non-functional requirements might differ dramatically from one workload to another.

Big Data analytics pipelines (also referred to as workflows) need to be efficiently orchestrated. There exist many serverless workflows orchestration tools [51, 52, 53, 54, 55], ephemeral serverless composition frameworks [42], and stateful composition engines [40, 41]. To the best of our knowledge, the workflow orchestration tools treat the FaaS run time as a black box oblivious to the workflow structure. This approach, while gaining in portability, has drawbacks related to performance, because an important information related to scheduling of the serverless functions that can be inferred from the workflow structure is not shared with the FaaS scheduler.

A major issue with FaaS, which is exacerbated in a multi-stage workflow, is the data shipment architecture of FaaS. Usually, the data is located in a separate service, such as Cloud Object Store (COS) and is shipped for computation to the FaaS cluster. Likewise, the output of the previous FaaS function(s) that might serve as input to the subsequent function(s) in the flow is re-shipped anew and, in general, FaaS functions are not scheduled with data locality in mind, even though data locality can be inferred from the workflow structure.

Also, to the best of our knowledge, none of the existing workflow orchestration tools is serverless in itself. In other words, the orchestrator is usually a statefull always on service. This is not necessarily the most cost-efficient approach for the long running big data analytics pipelines that might have periods of very high peakedness requiring massive parallelism interleaved with long periods of inactivity.

Last, but not the least, in a servermix model, which realistically assumes both serverless and non-FaaS components, the cost effectiveness of the whole pipeline depends on the time utilization of a component. Smart provisioning that helps selecting FaaS vs non-FaaS is required to improve cost-efficiency.

In CloudButton we will address the above challenges as follows.

- **Federated FaaS Model**: CloudButton will exploit K8s federation architecture to provide a structured multi-clustered FaaS run time to facilitate analytics pipelines spanning multiple K8s clusters. The FaaS frameworks that we plan to extend to fit the federated architecture are CNCF Knative and Apache OpenWhisk with public cloud FaaS offerings pluggable to the system, albeit with much less control over scheduling, as explained above.

- **SLA, QoS and Scheduling**: a programmer will be enabled to specify desired QoS levels for the functions that will be enforced by a specialized scheduler (implemented via the K8s custom scheduler framework). This scheduler will also take the structure of a workflow into account and use this information to improve performance by e.g., pre-warming containers, pre-fetching data, caching data from previous stages, and burst to remote clusters, when local capacity is exhausted. SLA corresponding to the specific QoS will be monitored and enforced.

- **Servermix Workflow Orchestration**: CloudButton will develop serverless orchestration framework for servermix big data analytics pipelines by extending mature native K8s tools, e.g., Argo [52]. Tasks in the servermix workflow might include massively parallel serverless computations, such as, e.g., PyWren. The orchestrator will take care of PyWren invocation restarts,

traffic shaping (e.g., how many per time unit), completion handling, etc. moving the burden of orchestration from the PyWren client to the platform and leaving PyWren with the application related tasks, such as smart data partitioning.

- **Operational Efficiency**: an operations cost-efficiency advisor will observe the time utilization of a servermix components and form recommendations on the more appropriate paradigm for every component. For example, a component, which is in constant use, might be more cost-efficiently provided and operated as a serverful one rather than FaaS, while a component utilized below some break-even point depending on the cost of the private infrastructure and/or public cloud services can be more efficiently operated using the serverless approach.

7.2 Mutable Shared Data for Serverless Computing

In the context of big data, workloads abide to what we would call storm computing, where thousands of serverless functions happen in a brief period of time. From a storage perspective, this requires the ability to scale abruptly the system in order to be on par with demand. To achieve this, it is necessary to shrink the startup time (e.g., with unikernels [56]) and consider new directions for data distribution (Pocket [38] for instance uses a central directory and provisions storage nodes in advance).

Current serverless computing platforms outsource state management to a dedicated storage tier (e.g., AWS S3). This tier is agnostic of how data is mutated by functions, requiring data (de-)serialization in serverless functions. Such an appraoch is cumbersome for complex data types, decreases code modularity and re-usability, and increases the cost of manipulating large objects. In contrast, we advocate that the storage tier supports in-place modifications (similarly to what DBMS systems offer with stored procedures [57]).

Serverless computing infrastructures have additional key requirements on the storage tier to permit efficient big data manipulations. This includes *(i)* fast access (sub-millisecond) to ephemeral mutable data in order to support iterative and stateful computations (e.g., ML algorithms); *(ii)* fine-grained operations to coordinate concurrent function invocations (similarly to coordination kernels [58]); and *(iii)* dependability to transparently support failures in both storage and compute tiers.

In CloudButton, we envision to tackle the above challenges by designing a new storage layer for stateful serverless computation. Our end goal is to simplify at most transitioning from sequential to massively-parallel code. This requires to advance the state of the art on several key questions in data storage and distributed algorithms. Below, we list the features that we aim to achieve in the storage system.

- **Language support for mutable shared data.** The programmer can declare mutable shared data types in a piece of serverless code. This declaration is integrated transparently to the programming language (e.g., with the help of annotations). The storage tier knows the data types, allowing in-place mutations. Furthermore, these data types are composable and sharded transparently for performance.

- **Tunable data consistency.** Shared objects are distributed and replicated across the storage tier. Strong consistency maintains application's sequential invariants but performance generally suggests to use weaker consistency models [59, 60]. To reconcile ease of programmation and performance, the programmer can degrade data consistency. This degradation is controlled at the level of individual object and integrated to the langague support.

- **Just-right synchronization.** Each object is implemented using state machine replication atop a consensus layer [61, 62]. This layer is adptable and self-adjusts to the consistency of each shared data item. Doing this, data replicas synchronize only when necessary, transforming consistency degradation into performance.

- **In-memory data storage.** Shared data is stored in-memory and overflows to external storage (e.g., filesystem, database, etc) when it is tagged as persistent (for instance, the centroids at the end of a k-means clustering). To cope with the short-lived highly-demanding nature of the

workload, *(i)* data distribution is computed before computation occurs; *(ii)* persistent and in-memory data nodes collaborate; and *(iii)* storage adapts replication and locality on-the-fly with the help of an external orchestrator.

7.3 CloudButton toolkit

Containers are the foundation of serverless runtimes, but the abstractions and isolation they offer can be restrictive for many applications. A hard barrier between the memory of colocated functions means all data sharing must be done via external storage, precluding data-intensive workloads and introducing an awkward programming model. Instantiating a completely isolated runtime environment for each function is not only inefficient, but at odds with how most language runtimes were designed.

This isolation boundary and runtime environment have motivated much prior work. A common theme is optimising and modifying containers to better suit the task, exemplified by SOCK which makes low level changes to improve start-up times and efficiency [32]. Others have sacrificed isolation to achieve better performance, for example by colocating a tenants' functions in the same container [37]. A few frameworks for building serverless applications have emerged [22, 34, 21], but these have clearly required a lot of engineering and porting existing applications is labour-intensive.

Software fault isolation (SFI) has been proposed as an alternative isolation approach, offering memory-safety at low cost [63]. Introducing an intermediate representation (IR) to unify the range of languages used in serverless has also been advocated [23]. WebAssembly is perfectly suited on both counts. It is an IR built on the principles of SFI, designed for executing multi-tenant code [64]. This is evidenced by its use in proprietary serverless technologies such as CloudFlare Workers [31] and Fastly's Terrarium [65].

With the CloudButton Toolkit we will build on these ideas and re-examine the serverless programming and execution environment. We will investigate new approaches to isolation and abstraction, focusing on the following areas:

- **Lightweight serverless isolation.** By combining SFI, WebAssembly and existing OS tooling we will build a new isolation mechanism, delivering strong security guarantees at a fraction of the cost of containers. This will be the foundation on which we construct the rest of the toolkit.
- **Efficient localised state.** This new isolation approach allows sharing regions of memory between colocated functions, enabling low-latency parallel processing as well as new opportunities for inter-function communication. We will build on this to tackle data-intensive workloads and investigate how our scheduling can benefit from colocation.
- **Stateful programming abstractions.** To make CloudButton programming seamless, we will create a new set of abstractions, allowing users to combine stateful middleware with efficient localised state to easily build high-performance parallel applications.
- **Serialisable execution state.** WebAssembly's simple memory model makes it easy to serialise and resume a function's execution state. We will create a checkpoint, migrate and restore mechanism to make horizontal scaling across hosts transparent to the user.
- **Polyglot libraries and tooling.** By using a shared IR we can reuse abstractions across multiple languages. In this manner we will build a suite of generic tools to ease porting existing applications in multiple languages, including the CloudButton genomics and geospatial use-cases.

8 CloudButton Architecture

In theory, serverless data analytics applications could be designed using pure serverless services in the Cloud. For example, embarrassingly parallel map-reduce jobs could be developed only relying on serverless orchestration technologies, serverless functions, and serverless Object Storage.

But the reality is that serverless orchestration services [39] are in their infancy and they do not adequately manage parallel workloads. Furthermore, there is a clear lack in cloud providers for serverless fine-grained mutable state services, and for high performance serverless coordination and communication services to support Stateful Big Data Analytics tasks like distributed machine learning.

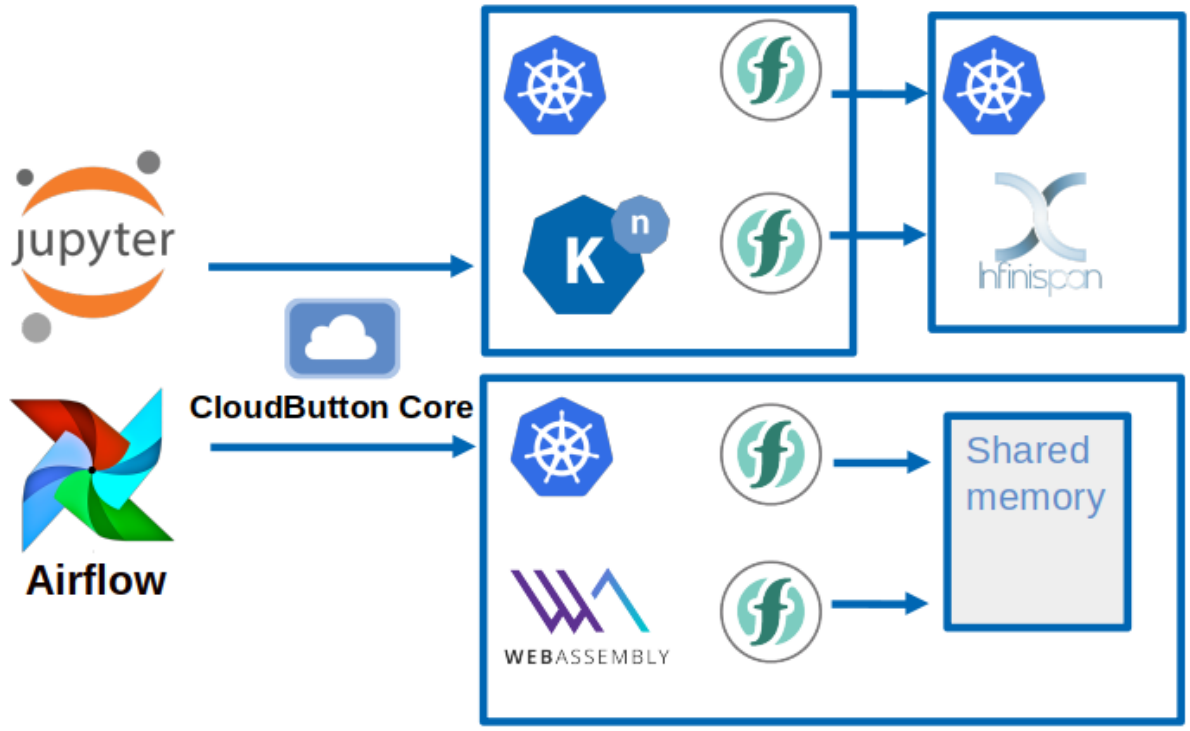

Figure 13: Software Architecture

For these reasons, the CloudButton project is going to leverage serverless container technologies using Kubernetes and the CNCF Knative software stacks. The major advantage of these technologies is that they will permit a straightforward portability of our results to a variety of Cloud providers.

The overall software architecture of the project is shown in Figure 13. Data Analytics applications may use imperative programming libraries (for example Jupyter notebooks accesing mapreduce tasks) or declarative Big Data pipelines modelled as Direct Acyclic Graphs (for example using Airflow or Kubeflow DAGs). The CloudButton Core will facilitate the deployment of data analytics code to Serverless functions and serverless containers. CloudButton Core will interact with container orchestration and resource scheduling systems such as Argo and Kubernetes schedulers to offer automated smart provisioning to applications.

Furthermore, the flexibility of container technology will enable us to easily integrate advanced components like Infinispan Mutable State Middleware, or WebAssembly specialized FaaS runtimes. Like we see in the Figure, the CloudButton Core abstracts away the access to heterogeneous runtimes executed in K8s clusters. Our smart provisioning policies will also be capable of selecting advanced hardware resources like GPUs or large memory and computing containers to boost some data analytics experiments.

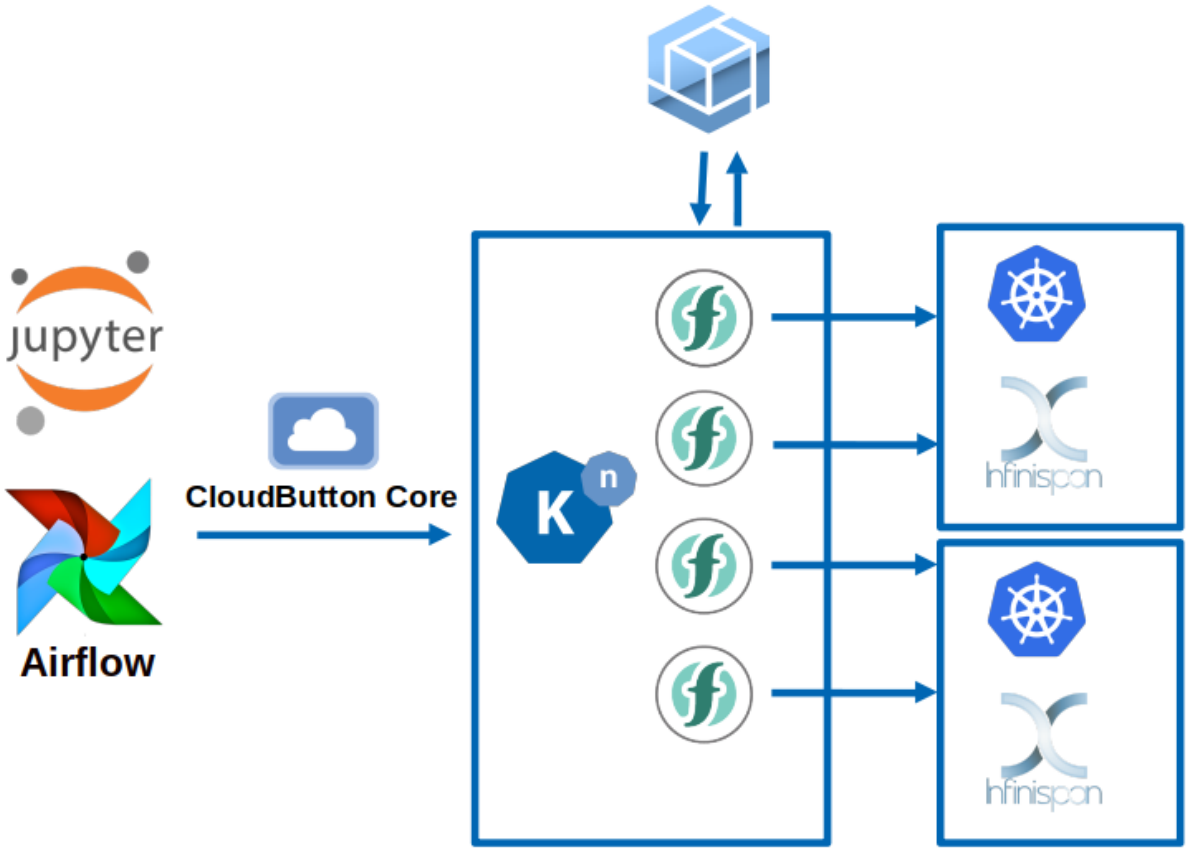

Figure 14: Servermix over K8s integrating heterogeneous runtimes

A key contribution of the CloudButton project will be the smart provisioning of Big Data Applications in serverless containers. Smart provisioning will aim to overcome the "No server Management" limitation of container technologies, by transparently managing and provisioning the required resources for the Big Data Experiments.

An important outcome is to design novel smart provisioning algorithms that optimize cost/performance for data analytics applications. Chosing the right serverful instances or hardware accelerators in a transparent or semi-transparent way to users will deserve a lot of research depending on the problems, datasets, and programming models. As stated in [24] , one approach is that developers explicitly state their requirements, data dependencies, or even data flows. For example, a KMeans clustering execution over a 1 TB dataset can automatically provision the required function and shared memory resources (Infinispan servers) for the experiment. Another more sophisticated approach is to automatically infer requirements analyzing source code or data dependencies.

Another interesting outcome is smart scheduling. The orchestration service keeps very valuable information about the workflows and function invocations in the state machine of a workflow. This can be smartly used to prewarm functions, prefetch data, or reserve resources in an efficient way. They can also provide consistent fault tolerance support that is not normally present in container technologies. But as we said, more meta-information is needed in this kind of services, not only the state machine, but also data dependencies (input, output), data formats, libraries, and even SLOs or performance/cost tradeoffs. For example, a smart scheduling decision could also entail changing the underlying FaaS runtime for a lightweight one offering function co-location. In CloudButton Core, our orchestrator could then decide to execute an application over a WebAssembly container in a transparent way for the end-user.

8.1 Initial Specifications and Software components

Let us describe the different software components of the CloudButton Architecture. The unifying framework for all components is the serverless cluster using CNCF K8s technologies.

- Serverless Infrastructure (OpenWhisk, KNative, Prometheus): IBM and ATOS will collaborate on smart scheduling and provisioning creating novel Kubernetes Schedulers adapted to Serverless Data Analytics. They will optimize both cost and performance for Big Data experiments deployed in the Cloud.
- Serverless Orchestration (Airflow, Kubeflow, Argo): IBM and URV will work together in creating new tools enabling the orchestration of Big Data pipelines over serverless functions and containers. The tools will provide declarative DAGs (Directed Acyclic Graphs) for the definition of the pipelines. Such DAGs will be leveraged by the underlying Serverless Infrastructure to optimize resource usage. We will extend Apache Airflow with new FaaS Big Data Operators and also leverage and adapt Argo for Big Data pipelines on serverless containers.
- Mutable State Middleware (Crucial, Infinispan): IMT, URV, and RHAT will create a novel disaggregated mutable middleware including consistent data structures and programming abstractions for stateful Big Data analytics over Infinispan. We will offer proof of concept machine learning algorithm packaged as Functions that can be executed and orchestrated by CloudButton Core in a K8s cluster. RHAT will integrate Infinispan and this middleware in the K8s stack, and it will provide controllers for flexible auto-scaling of ephemeral and replicated Infinispan cluaters.
- WebAssembly FaaS Runtime and programming abstractions: Imperial will create a novel lightweight and polyglot FaaS runtime over WebAssembly technology. This middleware will offer code-shipping models where lighweight functions can be colocated and access local shared memory in an efficient way. This FaaS runtime must be integrated in the K8s stack.
- CloudButton toolkit: The CloudButton toolkit will be created between Imperial, IBM, IMT, and URV and it will become the front-end of the project. Data analysts and practitioners will use the toolkit to realize the "push the button" objective of the project. The toolkit will include a number of technologies targeted to end-users. It will leverage existing tools like IBM PyWren, Jupyter notebooks, Airflow, KubeFlow, and many others to demonstrate both semi-transparent and fully transparent transition to serverless data analytics.

8.1.1 Infinispan

Infinispan is a distributed in-memory key/value data store with optional schema, available under the Apache License 2.0. It is available as an embedded Java library or as a language-independent service accessed remotely over a variety of protocols (Hot Rod, REST, Memcached). Its main uses are as a cache or a data grid, employing advanced functionality such as transactions, events, querying, persistence, distributed processing, off-heap and geographical failover. Infinispan can transparently be scaled up and down by adding and removing nodes dynamically, while still being operational. Infinispan automatically rebalances data to maintain similar memory usage across all nodes in the cluster. The dynamic discovery of new nodes works on Kubernetes, AWS, Azure, Google Cloud and OpenShift in bare-metal, containerized and virtualized environments. Monitoring and management of the cluster can be achieved through JMX, a CLI, a web-based console as well as, in more limited form, through the access protocols. A Kubernetes-based operator is also available which handles easy configuration, deployment, autoscaling and maintenance with as little user-intervention as possible. Infinispan integrates with JPA, JCache, Spring, Spark and many more. Clients for the Hot Rod protocol are available for Java, C++, C# and Node.js.

The following is a list of resources which describe the various components that make up Infinispan:

- The main Infinispan website which includes download links, documentation, examples and tutorials: `https://infinispan.org`
- Docker images for Infinispan Server: `https://hub.docker.com/r/jboss/infinispan-server`
- The Infinispan Kubernetes Operator: `https://operatorhub.io/operator/alpha/infinispan-operator.v0.2.1`
- Source repositories
  - Infinispan `https://github.com/infinispan/infinispan/`
  - Infinispan Docker image `https://github.com/jboss-dockerfiles/infinispan`
  - Infinispan Kubernetes Operator `https://github.com/infinispan/infinispan-operator`

8.1.2 Serverless infrastructure: Monitoring and SLA management

Monitoring is an essential building block for resource management and performance modelling for any system. Monitoring allows collecting metrics at multiple levels: computing infrastructure, storage systems and application levels developing system's observability which permits to understand its behavior. Collected information by means of monitoring becomes actionable data permitting to act in diverse undesirable system deeds: among others allowing to guarantee certain performance levels, to enable infrastructure elasticity and to facilitate fault tolerance. CloudButton's Serverless infrastructure will rely on widely used tools in order to provide this indispensable functionality. At this stage evaluated tools comprise Prometheus [66] and Grafana [67]. Prometheus is a Cloud Native Computing Foundation project fully integrated with Kubernetes with wide base of existing metrics and allowing extensibility by defining new metrics and data exporters. Prometheus is often integrated with Grafana to visualize gathered metrics.

SLA Management Service Level Agreements (SLAs) has been a mechanism used for managing QoS in multiple cloud research distributed environments [68]. At the same time, public Cloud providers are often criticized because solely considering availability as measurement of the performance being offered to their customers. Mechanisms to enable to expand performance metrics in Cloud providers beyond availability, as well as, to let application of penalties for the provider in the case of not fulfilling desired performance qualities, are becoming crucial in serverless architectures [23]. In this context CloudButton will rely on an existing SLA management framework REF SLA which will be extended with novel functionalities to be employed in this context. The exiting component provides a mechanism to support SLA Management lifecycle considering mechanism and tools for Service definition based on diverse Service Level Objectives (SLOs) in addition to assessment, enforcement and accounting of SLAs. These will work in close cooperation with Monitoring tools, in order to allow SLA enforcement to act in case of not satisfying agreed performance metrics. Specifically for CloudButton novel functionalities are evaluated in order to support diverse service levels at Service definition phase. These can include embracing diverse isolation degrees requirements as well as requirements for execution in heterogeneous hardware. Moreover, capabilities in relation to predictability on SLA performance will be explored as part of SLA Enactment phases.

Source repositories (private to Atos at this stage):

- SLALite: `https://gitlab.atosresearch.eu/ari/SLALite`
- SLALite Docker image: `https://gitlab.atosresearch.eu/ari/SLALite/tree/master/docker`

9 Conclusions

In this deliverable, we presented the experiments that will be used to validate the CloudButton platform, alongside with the project vision and an initial specification of the architecture.

In our view, the achievements of this deliverable represent a remarkable step towards the creation of a Serverless Data Analytics Platform. Furthermore, the progress of the project and the tight collaboration among partners draws promising milestones in the horizon. In particular, we outline the following innovative aspects that are related to the future architectural aspects of CloudButton:

- Smart provisioning of resources for data analytics applications to optimize both cost and performance.
- Smart scheduling of data analytics workflows to improve the efficiency of the serverless platform.
- A serverless machine learning library that leverages novel data structures and programming abstractions provided over a shared mutable state middleware.
- A common front-end targeted at end-users (data analysts and practitioners) that fulfills the promise of transparent execution of data analytics workloads in the cloud.

References

[1] "FAANG website." http://data.faang.org/home, 2019.

[2] F. c. L. Andersson, ..., "Coordinated international action to accelerate genome-to-phenome with FAANG, the Functional Annotation of ANimal Genomes project," Genome Biology, 2015.

[3] "FAANG dataset website." https://www.animalgenome.org/community/FAANG/index, 2019.

[4] E. D. T. Lappalainen, ..., "Transcriptome and genome sequencing uncovers functional variation in humans," Nature, 2013.

[5] "ENA website." https://www.ebi.ac.uk/ena/browse/download, 2019.

[6] "ICGC website." https://icgc.org, 2019.

[7] "ICGC data access policy." https://docs.icgc.org/download/data-access, 2019.

[8] "SAM specification." https://samtools.github.io/hts-specs/SAMv1.pdf, 2019.

[9] R. G. P. R. S. Marco-Sola, M. Sammeth, "The gem mapper: fast, accurate and versatile alignment by filtration," Nature Methods, 2012.

[10] E. Maderal, N. Valcarcel, J. Delgado, C. Sevilla, and J. Ojeda, "Automatic river network extraction from lidar data.," International Archives of the Photogrammetry, Remote Sensing & Spatial Information Sciences, vol. 41, 2016.

[11] P. C. Pandey, N. Koutsias, G. P. Petropoulos, P. K. Srivastava, and E. B. Dor, "Land use/land cover in view of earth observation: data sources, input dimensions and classifiers - a review of the state of the art," Geocarto International, vol. ePub ahead of print, 2019.

[12] IGN, "Centro Nacional de Información Geográfica." http://centrodedescargas.cnig.es/CentroDescargas/index.jsp, 2019.

[13] "Copernicus." https://scihub.copernicus.eu/.

[14] "Siam." http://siam.imida.es/apex/f?p=101:1:8786098632739315.

[15] "Aemet." http://www.aemet.es/es/portada.

[16] "Ministry of ecological transition." https://www.miteco.gob.es/es/cartografia-y-sig/ide/descargas/biodiversidad/enp.aspx.

[17] "Python." https://www.python.org/.

[18] "Qgis." https://www.qgis.org.

[19] "Pyqgis." https://qgis.org/pyqgis/3.0/.

[20] T. K. Ho, "Random decision forests," in Proceedings of 3rd international conference on document analysis and recognition, vol. 1, pp. 278–282, IEEE, 1995.

[21] S. Fouladi, R. S. Wahby, B. Shacklett, K. V. Balasubramaniam, W. Zeng, R. Bhalerao, A. Sivaraman, G. Porter, and K. Winstein, "Encoding, fast and slow: Low-latency video processing using thousands of tiny threads," in 14th USENIX Symposium on Networked Systems Design and Implementation (NSDI'17), 2017.

[22] E. Jonas, Q. Pu, S. Venkataraman, I. Stoica, and B. Recht, "Occupy the cloud: Distributed computing for the 99%," in Proceedings of the 2017 Symposium on Cloud Computing, SoCC'17, 2017.

[23] J. M. Hellerstein, J. Faleiro, J. E. Gonzalez, J. Schleier-Smith, V. Sreekanti, A. Tumanov, and C. Wu, "Serverless computing: One step forward, two steps back," Conference on Innovative Data Systems Research (CIDR'19), 2019.

[24] E. J. et al, "Cloud programming simplified: A berkeley view on serverless computing," https://arxiv.org/abs/1902.03383, 2019.

[25] Denis Kennely, "Three Reasons most Companies are only 20 Percent to Cloud Transformation." https://www.ibm.com/blogs/cloud-computing/2019/03/05/20-percent-cloud-transformation/.

[26] "Amazon AWS Serverless Definition." https://aws.amazon.com/serverless/, 2019.

[27] J. Gu, Y. Lee, Y. Zhang, M. Chowdhury, and K. G. Shin, "Efficient memory disaggregation with infiniswap.," in NSDI, pp. 649–667, 2017.

[28] Z. A.-A. et al, "Making serverless computing more serverless," in 2018 IEEE 11th International Conference on Cloud Computing (CLOUD), 2018.

[29] J. Sampé, M. Sánchez-Artigas, P. García-López, and G. París, "Data-driven serverless functions for object storage," in Proceedings of the 18th ACM/IFIP/USENIX Middleware Conference, pp. 121–133, ACM, 2017.

[30] "Open-sourcing gVisor, a sandboxed container runtime." https://cloud.google.com/blog/products/gcp/open-sourcing-gvisor-a-sandboxed-container-runtime, 2018.

[31] "WebAssembly on CloudFlare Workers." https://blog.cloudflare.com/webassembly-on-cloudflare-workers/, 2018.

[32] E. Oakes, L. Yang, D. Zhou, K. Houck, T. Harter, A. Arpaci-Dusseau, and R. Arpaci-Dusseau, "SOCK: Rapid task provisioning with serverless-optimized containers," in 2018 USENIX Annual Technical Conference (USENIX ATC 18), pp. 57–70, 2018.

[33] J. Sampe, M. Sanchez-Artigas, P. Garcia Lopez, and G. Paris, "Data-driven serverless functions for object storage," in Proceedings of the 18th ACM/IFIP/USENIX Middleware Conference, Middleware '17, (New York, NY, USA), pp. 121–133, ACM, 2017.

[34] Y. Kim and J. Lin, "Serverless data analytics with Flint," CoRR, vol. abs/1803.06354, 2018.

[35] V. Shankar, K. Krauth, Q. Pu, E. Jonas, S. Venkataraman, I. Stoica, B. Recht, and J. Ragan-Kelley, "numpywren: serverless linear algebra," 2018.

[36] Q. Pu, S. Venkataraman, and I. Stoica, "Shuffling, fast and slow: Scalable analytics on serverless infrastructure," in 16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19), (Boston, MA), pp. 193–206, USENIX Association, 2019.

[37] I. E. Akkus, R. Chen, I. Rimac, M. Stein, K. Satzke, A. Beck, P. Aditya, and V. Hilt, "SAND: Towards high-performance serverless computing," in 2018 USENIX Annual Technical Conference (USENIX ATC 18), pp. 923–935, 2018.

[38] A. Klimovic, Y. Wang, P. Stuedi, A. Trivedi, J. Pfefferle, and C. Kozyrakis, "Pocket: Elastic ephemeral storage for serverless analytics," in 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18), (Carlsbad, CA), pp. 427–444, USENIX Association, 2018.

[39] P. García López, M. Sánchez-Artigas, G. París, D. Barcelona Pons, Á. Ruiz Ollobarren, and D. Arroyo Pinto, "Comparison of faas orchestration systems," in 2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion), pp. 148–153, IEEE, 2018.

[40] Amazon, "AWS Step Functions." https://aws.amazon.com/step-functions/, 2016.

[41] Microsoft, "Azure Durable Functions." https://docs.microsoft.com/en-us/azure/azure-functions/durable-functions-overview, 2018.

[42] "Apache OpenWhisk Composer." https://github.com/apache/incubator-openwhisk-composer, 2018.

[43] Amazon, "Amazon Elastic Kubernetes Service." https://aws.amazon.com/eks/, 2018.

[44] Google, "Kubernetes Engine." https://cloud.google.com/kubernetes-engine/, 2014.

[45] Microsoft, "Azure Kubernetes Service (AKS)." https://azure.microsoft.com/en-us/services/kubernetes-service/, 2017.

[46] Amazon, "AWS Fargate." https://aws.amazon.com/fargate/, 2017.

[47] Microsoft, "Container instances." https://azure.microsoft.com/en-us/services/container-instances/, 2017.

[48] Google, "Google Cloud Run." https://cloud.google.com/run/, 2019.

[49] "KNative Platform." https://cloud.google.com/knative/, 2019.

[50] "H2020 CloudButton, Serverless Data Analytics." http://cloudbutton.eu, 2019.

[51] "Fission Flows." https://fission.io/workflows/, 2018.

[52] "Argo Workflows." https://argoproj.github.io/, 2018.

[53] "Apache Airflow." https://github.com/apache/airflow, 2018.

[54] "Brigade: Event-based Scripting for Kubernetes." https://github.com/brigadecore/brigade, 2018.

[55] "Kubeflow." https://www.kubeflow.org/, 2018.

[56] F. Manco, C. Lupu, F. Schmidt, J. Mendes, S. Kuenzer, S. Sati, K. Yasukata, C. Raiciu, and F. Huici, "My vm is lighter (and safer) than your container," in Proceedings of the 26th Symposium on Operating Systems Principles, SOSP '17, (New York, NY, USA), pp. 218–233, ACM, 2017.

[57] M. Stonebraker and G. Kemnitz, "The POSTGRES Next Generation Database Management System," Commun. ACM, vol. 34, pp. 78–92, Oct. 1991.

[58] P. Hunt, M. Konar, F. P. Junqueira, and B. Reed, "ZooKeeper: Wait-free Coordination for Internet-scale Systems," in USENIX Annual Technical Conference, USENIX ATC, USENIX Association, 2010.

[59] H. Wada, A. Fekete, L. Zhao, K. Lee, and A. Liu, "Data consistency properties and the trade-offs in commercial cloud storage: the consumers' perspective," in CIDR 2011, Fifth Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, January 9-12, 2011, Online Proceedings, pp. 134–143, 2011.

[60] H. Attiya and J. L. Welch, "Sequential consistency versus linearizability (extended abstract)," in Proceedings of the Third Annual ACM Symposium on Parallel Algorithms and Architectures, SPAA '91, (New York, NY, USA), pp. 304–315, ACM, 1991.

[61] L. Lamport, "The part-time parliament," ACM Trans. Computer Systems, vol. 16, pp. 133–169, May 1998.

[62] D. Ongaro and J. K. Ousterhout, "In search of an understandable consensus algorithm," in Proceedings of the 2014 USENIX Conference on USENIX Annual Technical Conference, USENIX ATC'14, (Philadelphia, PA, USA), pp. 305–319, June 2014.

[63] S. Boucher, A. Kalia, D. G. Andersen, and M. Kaminsky, "Putting the Micro Back in Microservice," 2018 USENIX Annual Technical Conference (USENIX ATC '18)., pp. 645–650, 2018.

[64] A. Haas, A. Rossberg, D. L. Schuff, B. L. Titzer, M. Holman, D. Gohman, L. Wagner, A. Zakai, and J. Bastien, "Bringing the web up to speed with WebAssembly," Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation - PLDI 2017, pp. 185–200, 2017.

[65] Fastly, "Fastly Labs - Terrarium." `https://www.fastlylabs.com/`.

[66] "Prometheus - Monitoring system & time series database." `https://prometheus.io/`.

[67] "Grafana." `https://github.com/grafana/grafana`.

[68] E. D. Kyriazis, "Cloud computing service level agreements— Exploitation of research results," tech. rep., European Commission Directorate General Communications Networks, Content and Technology Unit E2—Software and Services, Cloud, 2013.