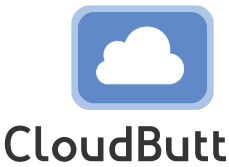


HORIZON 2020 FRAMEWORK PROGRAMME

CloudButton

(grant agreement No 825184)

Serverless Data Analytics Platform

D2.3 CloudButton Architecture Specs and Early Prototypes

Due date of deliverable: 30-06-2020
Actual submission date: 04-08-2020

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	46
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), Gerard París (URV), Rut Palmero (ATOS), Tonny Velin (ANSWARE), Elvis Jorge (Matrix), Theodore Alexandrov (EMBL), Paolo Ribeca (JHI).
Partner(s) Contributing	URV, ATOS, ANSWARE, MATRIX, EMBL, JHI.
Document ID	CloudButton_D2.3_Public.pdf
Abstract	Reflections on project vision and Serverless Analytics state of the art. Overview of the project architecture and APIs. Early prototypes of the different software components. Description of the project testbed. First description and evaluation of results obtained from validation in use cases using different experiments and workloads.
Keywords	use cases, experiments, architecture, state of the art.

History of changes

Version	Date	Author	Summary of changes
0.1	18-06-2020	Gerard París (URV)	Table of contents.
0.2	19-06-2020	Rut Palmero (ATOS)	Testbed description.
0.3	09-07-2020	Pedro García (URV), Gerard París (URV)	Project Vision, Towards Transparency, CloudButton Architecture
0.4	10-07-2020	Tonny Velin (ANSWARE)	Geospatial use case.
0.5	13-07-2020	Pedro García (URV)	Revision of overall view.
0.6	17-07-2020	Elvis Jorge (MATRIX)	Geospatial use case (experiments 1 and 2)
0.7	23-07-2020	Theodore Alexandrov (EMBL)	Metabolomics use case.
0.8	25-07-2020	Paolo Ribeca (JHI)	Genomics use case.
1.0	04-08-2020	Gerard París (URV)	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 D2.2 "CloudButton Architecture Specs and Early Prototypes" aims to present the specification of the CloudButton architecture that describes how the various components and building blocks fit together and suitably communicate and interact.

This deliverable also describes the testbed that will be used to validate project results, and presents the progress of the three CloudButton use cases: Genomics, Metabolomics and Geospatial.

The document is structured as follows. Section 2 provides a brief introduction to the deliverable. Section 3 presents the project vision and an analysis of Serverless Analytics State of the Art. In section 4 we present our reflections on several paths to achieve transparency in disaggregated systems. Section 5 describes the project Architecture and main building blocks whereas section 6 provides a description of the project testbed. Sections 7-9 are devoted to the description of the use cases. Finally, section 10 contains the conclusions of this document.

2 Introduction

With the emergence of serverless computing, the cloud has found a new paradigm that removes much of the complexity of its usage by abstracting away the provisioning of compute resources. This fairly new model was culminated in 2015 by Amazon in its Lambda service. This service offered cloud functions, marketed as FaaS (Function as a Service), and rapidly became the core of serverless computing. We say "core", because cloud platforms usually provide specialized serverless services to meet specific application requirements, packaged as BaaS (Backend as a Service). However, the words "serverless computing" and "FaaS" are often used interchangeably, and so occurs in this deliverable, which is focused on the FaaS model. The reason why FaaS drew widespread attention is because with FaaS platforms, a user-defined function and its dependencies are deployed to the cloud, where they are managed by the cloud provider and executed on demand. Simply put, users just write cloud functions in a high-level language and the serverless systems is who manages everything else: instance selection, auto-scaling, deployment, sub-second billing, fault tolerance, and so on. The programming simplicity of functions paves the way to soften the transition to the cloud ecosystem for end users.

This new cloud paradigm suits well for many applications, and researchers have already begun investigating the feasibility of serverless computing for data analytics. Unfortunately, today's serverless computing presents important limitations that make it really difficult to support all sorts of analytics workloads. In this deliverable we identify three fundamental trade-offs of today's serverless computing model and their relationship with data analytics. The consequence of these trade-offs is that analytics applications may well end up embracing hybrid systems composed of serverless and serverful components.

We also review the state of the art on Serverless Analytics, and identify that most of serverless data analytics systems are good examples of these hybrid systems composed of serverless and serverful components.

Another aspect that we analyze in this deliverable is how to achieve transparency when moving existing codebases to the serverless paradigm. Full transparency implies that we can compile, debug and run unmodified single-machine code over effectively unlimited compute, storage, and memory resources. We elaborate in this deliverable why resource disaggregation in serverless computing can be the definitive catalyst to enable almost full transparency in the Cloud.

This vision of full transparency has implications for the development of applications in the Cloud and also means that CloudButton should aim to minimize the creation of new Cloud APIs, and instead rely on the interception of existing programming libraries for accessing remote computing resources. In this deliverable, we present an overall view of the project architecture, showing the software pillars that share this CloudButton goal of expressing a wide range of existing data-intensive applications with minimal changes.

We also present the main lines of integration between project results, highlighting CNCF technologies and Knative as the unifying framework for all the software components created in the context of the CloudButton project.

Finally, this deliverable describes the testbed that will be used to validate project results, and presents the progress of the three CloudButton use cases: Genomics, Metabolomics and Geospatial.

3 Project Vision

Current practice shows that the FaaS model is well suited for many types of applications, provided that they require a small amount of storage and memory. Indeed, this model was originally designed to execute event-driven, stateless functions in response to user actions or changes in the storage tier (e.g., uploading a photo to Amazon S3), which encompasses many common tasks in cloud applications. What was unclear is whether or not this new computing model could also be useful to execute data analytics applications.

This question was answered partially in 2017 with the appearance of two relevant research articles [1] [2] that 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: [3] and [4]. 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 deliverable D2.1 [5], we identified the hybrid applications combining Serverless and Serverful services as **ServerMix**, and we showed how most related work can be classified under the ServerMix umbrella term.

We also identified three important tradeoffs of the Serverless model and how these tradeoffs could be relaxed to obtain more performance:

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 fundamental 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.

We claimed 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.

Therefore, in CloudButton project 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.

3.1 Serverless Analytics State of the Art

Despite the stringent constraints of the FaaS model, a number of works have managed to show how this model can be exploited to process and transform large amounts of data [2, 6, 7], encode videos [1], and run large-scale linear algebra computations [8], among other applications. Surprisingly, and contrary to intuition, most of these serverless data analytics systems are indeed good *ServerMix* examples, as they combine both serverless and serverful components.

In general, most of these systems rely on a external, serverful provisioner component [2, 6, 7, 1, 8]. This component is in charge of calling and orchestrating serverless functions using the APIs of the chosen cloud provider. Sometimes the provisioner is called “coordinator” (e.g., as in ExCamera [1]) or “scheduler” (e.g., as in Flint [7]), but its role is the same: orchestrating functions and providing some degree of fault tolerance. But the story does not end here. Many of these systems require additional serverful components to overcome the limitations of the FaaS model. For example, recent works such as [9] use disaggregated in-memory systems such as ElastiCache Redis to overcome the throughput and speed bottlenecks of slow disk-based storage services such as S3. Or even external communication or coordination services to enable the communication among functions through a disaggregated intermediary (e.g., ExCamera [1]).

To fully understand the different variants of *ServerMix* for data analytics, we will review each of the systems one by one in what follows. Table 1 details which components are serverful and serverless for each system.

PyWren [2] is a proof of concept that MapReduce tasks can be run as serverless functions. More precisely, PyWren consists of a serverful function scheduler (i.e., a client Python application) that permits to execute “map” computations as AWS Lambda functions through a simple API. The “map” code to be run in parallel is first serialized and then stored in Amazon S3. Next, PyWren invokes a common Lambda function that deserializes the “map” code and executes it on the relevant datum, both extracted from S3. Finally, the results are placed back into S3. The scheduler actively polls S3 to detect that all partial results have been uploaded to S3 before signaling the completion of the job.

IBM-PyWren [6] is a PyWren derived project which adapts and extends PyWren for IBM Cloud services. It includes a number of new features, such as broader MapReduce support, automatic data discovery and partitioning, integration with Jupiter notebooks, and simple function composition, among others. For function coordination, IBM-PyWren uses RabbitMQ to avoid the unnecessary polling to the object storage service (IBM COS), thereby improving job execution times compared with PyWren.

ExCamera [1] performs digital video encoding by leveraging the parallelism of thousands of Lambda functions. Again, ExCamera uses serverless components (AWS Lambda, Amazon S3) and serverful ones (coordinator and rendezvous servers). In this case, apart from a coordinator/scheduler component that starts and coordinates functions, ExCamera also needs of a rendezvous service, placed in an EC2 VM instance, to communicate functions amongst each other.

Stanford’s gg [10] is an orchestration framework for building and executing burst-parallel applications over Cloud Functions. gg presents an intermediate representation that abstracts the compute

Table 1: *ServerMix* applications

Systems	**Components**	
	Serverful	***Serverless***
PyWren [2]	Scheduler	AWS Lambda, Amazon S3
IBM PyWren [6]	Scheduler	IBM Cloud Functions, IBM COS, RabbitMQ
ExCamera [1]	Coordinator and rendezvous servers (Amazon EC2 VMs)	AWS Lambda, Amazon S3
gg [10]	Coordinator	AW Lambda, Amazon S3, Redis
Flint [7]	Scheduler (Spark context on client machine)	AW Lambda, Amazon S3, Amazon SQS
Numpywren [8]	Provisioner, scheduler (client process)	AWS Lambda, Amazon S3, Amazon SQS
Cirrus [4]	Scheduler, parameter servers (large EC2 VM instances with GPUs)	AWS Lambda, Amazon S3
Locus [9]	Scheduler, Redis service (AWS ElastiCache)	AWS Lambda, Amazon S3

and storage platform, and it provides dependency management and straggler mitigation. Again, gg relies on an external coordinator component, and an external Queue for submitting jobs (gg's thunks) to the execution engine (functions, containers).

Flint [7] implements a serverless version of the PySpark MapReduce framework. In particular, Flint replaces Spark executors by Lambda functions. It is similar to PyWren in two main aspects. On one hand, it uses an external serverful scheduler for function orchestration. On the other hand, it leverages S3 for input and output data storage. In addition, Flint uses the Amazon's SQS service to store intermediate data and perform the necessary data shuffling to implement many of the PySpark's transformations.

Numpywren [8] is a serverless system for executing large-scale dense linear algebra programs. Once again, we observe the *ServerMix* pattern in numpywren. As it is based in PyWren, it relies on a external scheduler and Amazon S3 for input and ouput data storage. However, it adds an extra serverful component in the system called provisioner. The role of the provisioner is to monitor the length of the task queue and increase the number of Lambda functions (executors) to match the dynamic parallelism during a job execution. The task queue is implemented using Amazon SQS.

Cirrus machine learning (ML) project [4] is another example of a hybrid system that combines serverful components (parameter servers, scheduler) with serverless ones (AWS Lambda, Amazon S3). As with linear algebra algorithms, ML researchers have traditionally used clusters of VM instances for the different tasks in ML workflows. Nonetheless, a fixed a cluster size can either lead to severe underutilization or slowdown, since each stage of a workflow can demand significantly different amounts of resources. Cirrus addresses this challenge by enabling every stage to scale to meet its resource demands by using Lambda functions. The main problem with Cirrus is that many ML algorithms require state to be shared between cloud functions, for it uses VM instances to share

and store intermediate state. This necessarily converts Cirrus into another example of a *ServerMix* system.

Finally, the most recent example of *ServerMix* is Locus [9]. Locus targets one of the main limitations of the serverless stateless model: data shuffling and sorting. Due to the impossibility of function-to-function communication, shuffling is ill-suited for serverless computing, leaving no other choice but to implement it through an intermediary cloud service, which could be cost prohibitive to deliver good performance. Indeed, the first attempt to provide an efficient shuffling operation was realized in PyWren [2] using 30 Redis ElastiCache servers, which proved to be a very expensive solution. The major contribution of Locus was the development of a hybrid solution that considers both cost and performance. To achieve an optimal cost-performance trade-off, Locus combined a small number of expensive fast Redis instances with the much cheaper S3 service, achieving comparable performance to running Apache Spark on a provisioned cluster.

We did not include SAND [11] in the list of *ServerMix* systems. Rather, it proposes a new FaaS runtime. In the article, the authors of SAND present it as an alternative high-performance serverless platform. To deliver high performance, SAND introduces two relaxations in the standard serverless model: one in *disaggregation*, via a hierarchical message bus that enables function-to-function communication, and another in *isolation*, through application-level sandboxing that enables packing multiple application-related functions together into the same container. Although SAND was shown to deliver superior performance than Apache OpenWhisk, the paper failed to evaluate how these relaxations can affect the scalability, elasticity and security of the standard FaaS model.

Recent works also outline the need for novel serverless services providing flexible disaggregated storage to serverless functions. This is the case of Pocket's ephemeral storage service [12], which provides auto-scaling and pay-per-use as a service to cloud functions. Similarly, [4] proposes as a future challenge the creation of high-performance, affordable, transparently provisioned storage. This work discusses two types of unaddressed storage needs: *Serverless Ephemeral Storage* and *Serverless Durable Storage*, both of which should deliver micro-second latencies, fault-tolerance, auto-scalability and transparent provisioning for multi-tenant workloads. The paper suggests that with a shared in-memory service, spare memory resources from one serverless application can be allocated to another. Finally, it also elaborates on why existing cloud services such as Redis or MemCached cannot fulfill the aforementioned storage needs. Actually, both in-memory services can be deemed as *serverful* due to their need for explicit provisioning and dedicated resources per tenant.

3.2 Serverless orchestration systems

An interesting alternative could be to use serverless orchestration services to coordinate and provision data analytics applications. The key question is whether these systems are actually well-prepared to orchestrate massively parallel computations. In this sense, our recent paper [13] compares three commercial serverless orchestration services along several axis: Amazon Step Functions, Azure Durable Functions, and IBM Composer, updating a previous analysis [14]. The conclusion of our study was that none of the platforms solve the orchestration of parallel computations with suitable performance.

We find that IBM Composer is the best solution available as of today. Its idea of using serverless functions for the orchestration logic is indeed a good decision to achieve fitting elasticity. However, its implementation of parallel executions fails to fulfill a reactive scheme by adding a synchronous external connection. Furthermore, the system falls short on allowing long-running workflows (some steps would be billed for idle time and waste resources); a direct consequence of the orchestration system being so tightly integrated in the FaaS infrastructure itself. We also believe that orchestration of serverless functions requires a reactive event-based design, avoiding double billing and blocking external invocations.

Unfortunately, as of today, the reality is that cloud applications are still inadvertently bound to the *ServerMix* model. In fact, several cloud providers are now transitioning to hybrid models combining serverless and serverful concepts. For instance, we have recently seen how some cloud providers like Microsoft offer *ServerMix* FaaS services such as the Azure Premium Plan for running functions

on dedicated machines while abstracting users from the provisioning phase. The Azure platform is even allowing users to pre-warm functions to reduce their cold starts.

3.3 Serverless container services

Hybrid cloud technologies are also accelerating the combination of serverless and serverful components. For instance, the recent deployment of Kubernetes (k8s) clusters in the big cloud vendors can help overcome the existing application portability issues in the cloud. There exists a plenty of hosted k8s services such as *Amazon Elastic Container Service* (EKS), *Google Kubernetes Engine* (GKE), and *Azure Kubernetes Service* (AKS), which confirm that this trend is gaining momentum. However, none of these services can be considered 100% "serverless". Rather, they should be viewed as a middle ground between cluster computing and serverless computing. That is, while these hosted services offload operational management of k8s, they still require custom configuration by developers. The major similarity to serverless computing is that k8s can provide short-lived computing environments like in the customary FaaS model.

But a very interesting recent trend is the emergence of the so-called serverless container services such as AWS Fargate, Azure Container Instances (ACI), and Google Cloud Run (GCR). These services reduce the complexity of managing and deploying k8s clusters in the cloud. While they offer serverless features such as flexible automated scaling and pay-per-use billing model, these services still require some manual configuration of the right parameters for the containers (e.g., compute, storage, and networking) as well as the scaling limits for a successful deployment.

These alternatives are interesting for long-running jobs such as 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 by easily adding or removing container instances. In this case, the user establishes a simple CPU or memory threshold and the service is responsible for monitoring, load balancing, and instance creation and removal. It must be noted that if the service or application is more complex (e.g., a stateful storage component), the utility of these approaches is rather small, or they require important user intervention.

For example, AWS Fargate offers two models: Fargate launch type and EC2 launch type. The former is more serverless and requires less configuration. The latter gives users more control but also more responsibility. An analogous issue occurs with Google: Cloud Run vs. Cloud Run on GKE. The former is automated and uses standard vCPUs, while the latter enables customers to select hardware requirements and manage their cluster.

An important open source project related to serverless containers is CNCF's Knative. In short, Knative is backed by big vendors such as Google, IBM and RedHat, among others, and it simplifies the creation of serverless containers over k8s clusters. Knative simplifies the complexity of k8s and Istio service mesh components, and it creates a promising substrate for both PaaS and FaaS applications. GCR is based on Knative. IBM Cloud is also offering seamless Knative integration in their k8s services. Yet, it is hard to see how, in future terms, hosted Knative and k8s cloud services will reshape the current FaaS landscape, since, in its present form, have important implications on the key traits of the FaaS model such as disaggregation and scheduling.

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 few years, thereby blurring the borders between both serverless and serverful models. Further, container services may become an interesting architecture for *ServerMix* deployments.

4 Towards transparency

An aspect to consider when moving existing codebases to the serverless paradigm is that of the transparency. Achieving full transparency would imply that we can compile, debug and run unmodified single-machine code over effectively unlimited compute, storage, and memory resources.

Transparency is an archetypal challenge in distributed systems that has not yet been adequately solved. Transparency implies the concealment from the user and the application programmer of the complexities of distributed systems. Coulouris et al. [15] define eight forms of transparency: access, location, concurrency, replication, failure, mobility, performance, and scalability.

But, despite all previous efforts, the problem is still open as seen in recent literature [2] and is, in fact, one of the main objectives of this project: simplifying the overall life cycle and programming model (in the application domain of big data analytics).

Waldo et al. [16] explain that the goal of merging the programming and computational models of local and remote computing is not new. They state that around every ten years "a furious bout of language and protocol design takes place and a new distributed computing paradigm is announced". They mention messages in the 70s, RPCs in the 80s, and objects in the 90s.

In every iteration, a new wave of software modernization is generated, and applications are ported to the newest and hot paradigm. Waldo et al. claim that all these iterations may be evolutionary stages to unify both local and distributed computing. But they are pessimistic, and they believe that this will not be possible because of latency, memory access, concurrency and partial failure.

That visionary paper even considers that, in the future, hardware improvements could make the difference in latency irrelevant, and that differences between local and remote memory could be masked. But they still claim that concurrency and partial failures preclude the unification of local and remote computing. Unlike an OS, they are telling us that a distributed system has no single point of resource allocation, synchronization, or failure.

But, what if novel cloud technologies could make the unification of local and remote paradigms possible? Are we close to the end of the cycles of software modernization? Can we just compile to the Serverless SuperComputer [17] imagined by Tim Wagner, inventor of AWS Lambda?

We argue in this deliverable that recent reductions in network latency [18, 19] are boosting resource disaggregation in the Cloud, which is the definitive catalyst to achieve transparency. Even if existing Cloud services are still in the millisecond range (100ms Lambda overhead, 10ms in Kafka, 5-20ms in S3), disaggregation has already fueled the creation of serverless computing services like Function as a Service, Cloud Object Storage, and messaging. If we can go down to µs RPCs [20, 21], novel opportunities for transparency will emerge [22, 19].

We believe that the full transparency for serverless computing will arrive when all resources (compute, storage, memory) can be offered in a disaggregated way with unlimited flexible scaling. This will also require a new generation of locality-aware scalable stateful services, smartly combining disaggregation and local resources. Also, we indentify and study five open research challenges required to achieve full transparency for most applications: (i) granular middleware and locality, (ii) memory disaggregation, (iii) virtualization, (iv) elastic programming models, and (v) optimized deployment.

4.1 DDC path to transparency

The DDC path is probably the most direct but also the most shocking for the distributed systems community. In line with recent industrial trends on Disaggregated Data centers (DDC) [23], it implies a distributed OS transparently leveraging disaggregated hardware resources like processing, memory or storage.

A canonical example is LegoOS: A disseminated, distributed OS for hardware resource disaggregation [24]. LegoOS exposes a distributed set of virtual nodes (vNode) to users. Each vNode is like a virtual machine managing its own disaggregated processing, memory and storage resources. LegoOS achieves transparency and backwards compatibility by supporting the Linux system call interface and Linux ABIs (Application Binary Interface), so that existing unmodified Linux applications can run on top of it. Even distributed applications that run on Linux can seamlessly run on a

LegoOS cluster by running on a set of vNodes. For example, LegoOS shows how two unmodified applications can be run in a distributed way: Phoenix (a single-node multi-threaded implementation of MapReduce) and TensorFlow.

Another relevant work is Arrakis: The Operating System is the Control Plane [25]. Arrakis comes from previous efforts aimed at optimizing the kernel code paths to improve data transfer and latency in the OS. In Arrakis, applications have direct access to virtualized I/O devices, which allows most I/O operations to bypass the kernel entirely without compromising process isolation. Arrakis virtualized control plane approach allows storage solutions to be integrated with applications, even allowing the development of higher level abstractions like persistent data structures. Even more, Arrakis control plane is a first step towards integration with a distributed data center network resource allocator.

If the OS can be extended with unbounded resources in a transparent way, distribution may no longer be needed for many applications – single-node parallel programming is sufficient. This is completely in line with the following assessment from the COST paper [26]: "You can have a second computer once you've shown you know how to use the first one". This paper presents a critique of the current research in distributed systems, and even suggests that "there are numerous examples of scalable algorithms and computational models; one only needs to look back to the parallel computing research of decades past".

COST stands in that paper for the "Configuration that Outperforms a Single Thread". They mainly compare optimized single-threaded versions of graph algorithms, with their equivalents in distributed frameworks like Spark, Naiad, GraphX, Giraph or GraphLab. For example, Naiad has a COST of 16 cores for executing PageRank on the twitterrv graph, which means that Naiad needs 16 cores to outperform a single-threaded version of the same algorithm in one machine.

An important reflection from this paper is that the overheads of distributed frameworks (coordination, serialization) can be extremely high just in order to justify scalability. But the COST paper is not proposing a solution to the scalability problem, since it is obvious that a single machine cannot scale enough for many algorithms.

But, what happens if we combine the COST idea with the DDC research? This is precisely what Gao et al.[23] validated in a simple experiment comparing a COST version with a COST-DDC one that relies on disaggregated memory (Infiniswap [27]). They demonstrate in this paper that the same code can overcome the memory limits thanks to disaggregation and still obtain good performance results.

DDC is openly challenging the so-called server-centric approach of development for the data center. DDC advocates claim that the monolithic server model where the server is the unit of deployment, operation, and failure is becoming obsolete. They advocate for a paradigm shift where many existing server-centric (cluster computing) approaches must yield to a more efficient way of managing resources in the Data Center.

The DDC paradigm is presenting server-centric cluster technologies as obsolete. But current mature Cloud technologies are built on top of server-centric models with commodity hardware and Ethernet networks. Hardware resource disaggregation is interesting, but it still relies on server-centric clusters for scaling. For example, LegoOS emulates disaggregated hardware components using a cluster of commodity servers. Existing OS approaches like LegoOS have still not dealt with serious distributed systems problems like scalability, consistency and security of the disaggregated resources in a multi-tenant Cloud setting.

4.2 Server-centric path to Transparency

Recent proposals are intercepting language libraries in order to access remote Cloud resources in a transparent way. For example, in the context of the CloudButton project, Crucial [28] implements a Serverless Scheduler for the Java Concurrency library. Crucial can run Java threads in Serverless functions transparently, and it also provides synchronization primitives and consistent mutable state data structures over a disaggregated in-memory computing layer. Crucial does not provide flexible memory scaling or storage transparency, and it is limited to Java applications using that library.

Another example of language level transparency is Fiber [29]. Fiber implements an alternative Python multiprocessing library that works over a scalable Kubernertes cluster. Fiber supports many Python multiprocessing abstractions like Process, Pool, Queue, Pipe and also remote memory in Manager objects. It demonstrates transparency executing unmodified Python applications from the OpenAI Baselines machine learning project. But Fiber does not support transparent disaggregated storage and memory, and it is limited to Python applications using that library.

The Fabric for Deep Learning (FfDL) [30] system moves existing Deep Learning frameworks like PyTorch or TensorFlow to the Cloud on top of cluster technologies like Kubernetes. [30] transparently provides dependability thanks to checkpointing, intercepting storage flows (file system) using optimized storage drivers to cloud object storage, and supporting locality with a gang scheduling algorithm that schedules all components of a job as a group.

FfDL gives us two interesting insights from their authors about the limitations of this system in scalability and transparency. On the one hand, the scaling was observed to be framework dependent so they could not achieve full scaling transparency. On the other hand, they explain that "the service was then increasingly used by data scientists who wanted as much control over their FfDL jobs as with their local machine". Users wanted to download datasets or Python packages from the public Internet, interactively debug models, and stream logs to local scripts in order to monitor the progress of jobs. Many of these requests were not possible due to security limitations and the architecture of the system, which frustrated some of their users.

Another example of transparency in a serverless context, and also a software output of CloudButton project, is Faasm [31]. Faasm exposes a specialised system interface which includes some POSIX syscalls, serverless-specific tasks, and frameworks such as OpenMP and MPI. Faasm transparently intercepts calls to this interface to automatically distribute unmodified applications, and execute existing HPC applications over serverless compute resources.

Faasm allows colocated functions to share pages of memory and synchronises these pages across hosts to provide distributed state. However, this is done through a custom API where the user must have knowledge of the underlying system, hence breaking full transparency. Furthermore, when functions are widely distributed, this approach exhibits performance similar to traditional distributed shared memory (DSM), which has proven to be poor without hardware support [32, 33].

4.3 Limits of disaggregation and transparency

Current data center networks already enable disk storage disaggregation [34], where reads from local disk are comparable (10ms) to reads from the network. In contrast, creating a thread in Linux takes about 10 µs, still far from the 15ms/100ms (warm/cold) achieved today in FaaS settings. With that, compute disaggregation is already feasible when job time renders these delays negligible.

Advances in datacenter networking and NVMs have reduced access to networked storage to 1 µs, however this is still an order of magnitude slower than local memory accesses which are in the nanosecond range [19] (100ns), and local cache accesses in the 4ns-30ns range. This means that local memory cannot be neglected, and should be smartly leveraged by memory disaggregation efforts [35]. Existing efforts in memory disaggregation [36, 27, 37, 12] strive to play in the µs range, which can be a limiting factor.

This is directly related to locality and affinity requirements for many stateful applications. The systems community is starting to acknowledge that stateful services need a different programming model and resource management than the stateless ones [22, 3]. Stateful services have very different requirements of coordination, consistency, scalability and fault tolerance, and they need to be addressed differently. Stateful services show the limits of disaggregation versus locality, since in some scenarios locality still matters.

For now, locality still plays a key role in stateful distributed applications. For example: (i) where huge data movements still are a penalty and memory-locality can be still useful to avoid data serialization costs; (ii) where specialized hardware like GPUs must be used [30]; in (iii) some iterative machine-learning algorithms [38]; in (iv) simulators, interactive agents or actors[39].

Finally, another important limitation is scaling transparency, which means that applications can

expand in scale without changes to the system structure or the application algorithms. If the local programming model was designed to use a fixed amount of resources, there is no magic way of transparently achieving scalability, not to mention elasticity.

Workloads that do not need elasticity, such as enterprise batch jobs or scientific simulations, can use disaggregated resources the same way as local as they do not need scalability. However, for more user driven and interactive services, such as internal enterprise web applications, simple porting of the executables (sometimes referred as "lift-and-shift") is rarely enough. The unchanged code is not able to take advantage of the elasticity of disaggregated resources and it is expensive to run code that is not used.

4.4 Challenges ahead

Let us review the major challenges to enable transparency for many applications:

- **Granular middleware and locality**: In line with granular computing [22, 19], we require microsecond latencies in existing middleware (compute, storage, memory, communication). In particular, there is a need to handle extremely short instantiation and execution times and more lightweight container technologies. We also require microsecond latencies in disaggregated storage and memory, messaging and collective communication.

  Granular applications are amenable to fine-grained elastic scaling, but this will not provide adequate performance without data locality. Locality and fine-grained resource management may also reduce the current cost of disaggregated resources. Locality is also needed to scale stateful services with different requirements of coordination, concurrency, consistency, distribution, scalability and fault tolerance. But existing FaaS services provide very limited locality/affinity mechanisms and limited networking, precluding inter-function communications. We foresee that next-generation container technologies may enable inter-container communication and provide affinity services for grouping related entities (e.g. gang scheduling [30]).

- **Memory disaggregation and Computational memory**: Disaggregated memory is still an open challenge and there is no available Cloud offering in this line. Many cluster technologies like Apache Spark, Dask, or Apache Ray rely on coupled and difficult-to-scale in-memory storage. Fast disaggregated memory and storage services [36, 12] can facilitate the elasticity of many cluster technologies [40].

  An important problem here is that disaggregated memory services cannot ignore the memory available in existing server-centric nodes in most Cloud providers. One option is to combine both local and remote memory resources efficiently [35]. Another potential solution here is the recent line on computational memory [41] and in-memory computing devices. Compute and memory locality (similar to mammalian brain where memory and processing are deeply interconnected) may considerably enhance computational efficiency.

- **Virtualization** Accessing disaggregated resources in a transparent manner requires a form of lightweight, flexible virtualization that does not currently exist. This virtualization must intercept computation and memory management to provide access to disaggregated resources, and must do so with native-like performance and no input from the programmer. Current serverless platforms use Linux containers and VMs for virtualization [42, 43], which have proven to be too heavyweight for fine-grained scaling, and inappropriate for stateful applications [31, 22, 3, 44]. Software-based virtualization is a more lightweight alternative that is seeing adoption in the serverless context [45, 31], and as a replacement for Docker [46], but is not yet mature enough to transparently support non-trivial existing applications.

  Virtualization necessarily defines an interface between users' code and the underlying system, but the nature of this interface in a transparent disaggregated context is unclear. Exposing a full Linux API makes porting legacy applications easy as shown in LegoOS [24], but requires heavy engineering and introduces historical idiosyncrasies such as `fork` [47]. WASI [48] aims

to provide lightweight virtualization with a subset of POSIX-like calls and a custom libc, but can only support a small subset of existing C/C++ applications. Platform-independent runtimes such as GraalVM [49] raise the virtualization layer into the language runtime itself. This affords flexibility in supporting a range of underlying hardware, but is restricted to a a subset of programming languages and applications.

- **Elastic programming models and developer experience:** In some cases, virtualization technologies cannot solve problems like scaling transparency if the code is programmed to use a fixed amount of resources. We then need elastic programming models for local machines that can be used without change when running over Cloud resources. Such elastic models should take care of providing the different transparency types (scaling, failure, replication, location, access) and other aspects of application behavior when it is moved between local and distributed environments. The local executable APIs may need to be expanded to include elastic programming abstractions for processes, memory, and storage.

  To fulfill the vision of disaggregation and transparency it will also be critical to provide tools for developers, enabling them to code both locally and remotely in the same manner with full transparency. Developers will need to be able to use tools to debug, monitor, profile, and if necessary access control planes to optimize their applications for cost and performance.

- **Optimized deployment**: Existing applications are a blackbox for the cloud, but the transition will imply a "compile to the Cloud" process. In this case, the Cloud will have access over applications' life cycle and it will be able to optimize their execution performance and cost. This means that they can perform static analysis to predict resource requirements, dependencies and potential for hardware acceleration. Future Cloud orchestration services will explicitly leverage data dependencies and execution requirements for improving workloads and resource management thanks to machine learning techniques [50, 51]. This compile process will also allow advanced debugging mechanisms for Cloud applications.

  Transparency efforts for different types of applications will require customizable control planes for applications. Such customization will be based on advanced observability and fast orchestration mechanisms relying on standard services and protocols. Monitoring and interception of the different resources (compute, storage, memory, network) should be available and even integrated into the data center, enabling coordinated actuators at different levels. This can enable the creation of millions of tiny control planes [52] adapted to the different applications and programming models.

We argue that full transparency will be possible soon in the Cloud thanks to low latency resource disaggregation. We foresee that next generation serverless technologies may overcome the limitations exposed by Waldo et al. [16] more than twenty five years ago. In the next years, we will be able to develop programs without taking care of address spaces, while a modern cloud environment will transparently and efficiently execute those on disaggregated resources.

The next frontier for transparency is to go beyond the boundaries of the data center, and seamlessly support heterogeneous resources in the Cloud Continuum (Hybrid/Edge/Federated/Cloud). Another important challenge is to devise elastic parallel programming models for a single machine that can transparently leverage heterogeneous resources in the Cloud Continuum.

5 CloudButton Architecture

5.1 Overall view

The main goal of CloudButton is to overly simplify Big Data applications thanks to Serverless Computing. Our recent vision paper entitled "Serverless End Game: Disaggregation enabling transparency"[53] is precisely explaining how the disaggregation of computing resources enabled by serverless computing technologies is going to enable almost full transparency.

This vision has long-lasting implications for the development of applications in the Cloud. In particular, it implies that we can move almost unmodified local applications to the Cloud. When our program uses threads and processes, CloudButton transparently runs this code in remote containers (FaaS, Knative), when our program accesses files and directories, CloudButton transparently interacts with Cloud remote Storage (Amazon S3, IBM Cloud Object Storage, Ceph, ...), and when our program interacts with memory CloudButton transparently accesses disaggregated memory (Infinispan, Redis)

This also means that CloudButton aims to minimize the creation of new Cloud APIs, and instead rely on the interception of existing programming libraries for accessing remote computing resources. The project is targeting three major software environments: Python, Java, and C++. Python is now a very popular language for data analysts with popular tools like Python Jupyter Notebooks and libraries like Numpy, Pandas, or scikitlearn. Java is also heavily used in MapReduce and machine learning environments like Apache Spark, Hadoop of Smile project among others. Finally, the project is interested in supporting HPC applications written in C/C++ and using popular environments like MPI and OpenMP.

As we see in Figure 1, CloudButton project presents three main pillars which correspond to the aforementioned Python, Java, and C++ environments. In the three cases, we see how the project is transparently decoupling the underlying disaggregated Cloud resources (Compute, Storage, In-Memory).

Aside from being built on top of disaggregated resources, the three software pillars of Figure 1 share the CloudButton goal of expressing a wide range of existing data-intensive applications with minimal changes.

In the case of Python, IBM and URV are leading efforts around IBM Pywren and the interception of native Python libraries like `multiprocessing` and `concurrent.futures` libraries. Porting an existing Python multi-threaded application that uses `multiprocessing` can be as simple as modifying some import statements. These efforts have demonstrated simplicity by moving very different Python applications and notebooks to the Cloud in Metabolomics and GeoSpatial use cases. There are also ongoing efforts to transparently support Python libraries like Scikitlearn on top of our serverless disaggregated software stack. As we explain in WP3 and D3.2 deliverable, IBM PyWren is a popular open source project with an active community and already used in production inside IBM.

In the case of Java, IMT and URV are leading efforts around the Crucial Java Serverless Toolkit. Crucial is also intercepting the standard Java Concurrency APIs, by providing a Serverless Executor that can run threads over Serverless Functions. Crucial also provides shared mutable state and synchronisation primitives in the Concurrency API (Barrier, AtomicLong, AtomicList) over RedHat Infinispan In-Memory middleware. Thanks to these APIs, we demonstrated different machine learning algorithms and applications like KMeans, Logistic Regression, and Random Forests in SMILE, a Java ML library [54]. As we explain in WP4 and D4.2 deliverable, Crucial is an open source project that leverages and benefits from Redhat Infinispan's Java open source community.

In the case of C/C++ and HPC, IMP is leading efforts around FaasM WebAssembly serverless Toolkit. WebAssemby is a popular Intermediate Language that allows the compilation/execution of multiple programming languages like C/C++, Javascript and many others. FaasM is intercepting POSIX APIs in WebAssembly to execute threads in serverless containers that can also benefit from locality and shared memory. IMP is demonstrating that is is possible to port existing C++ HPC applications using MPI and OpenMP. As we explain in WP5 and D5.2 deliverable, FaasM is an open source project that has raised the interest of the WebAssembly and serverless community.

In summary, all three pillars software outcomes are committed to the global objective of providing simple programming models for serverless cloud infrastructures targeted to existing data-intensive applications and implying as few changes as possible.

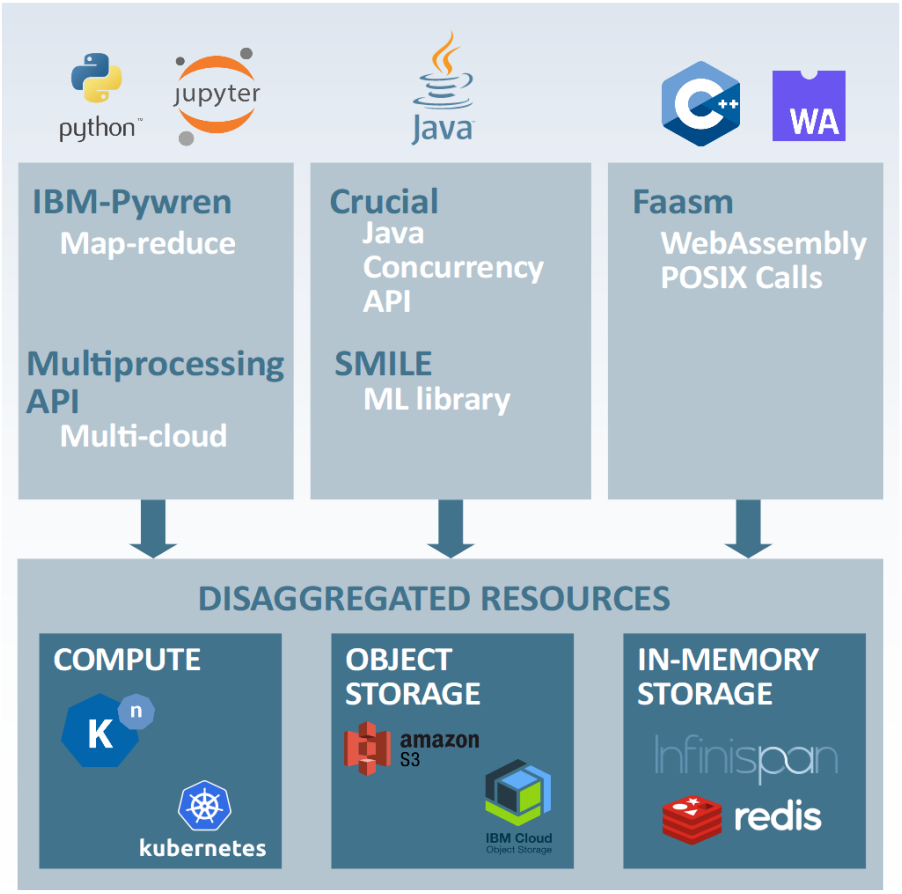


Figure 1: Big picture of the CloudButton architecture

5.1.1 Integration among the different components and contributions

As the project advances, we have seen an increase on the maturity and popularity of Knative and CNCF technologies as standard serverless building blocks for Kubernetes. For this reason, we believe that Knative can be the unifying framework for all the software components created in the context of the CloudButton project. Moreover, the adoption of Knative is a step towards serverless standardization: it simplifies the building of container-based, serverless applications that can deploy and run on any cloud: public, private, and hybrid.

On the one hand, CNCF K8s is becoming a Cloud standard, with many public Cloud providers supporting those technologies. But Knative as a serverless layer over K8s has a strong potential for **portability** and standardisation. Google is now offering a multi-tenant serverlss K8s and Knative in their Google Cloud Run service, but many others are following like IBM coligo, VMware/Dell, NetApp, or Alibaba among others.

The three main pillars of the project will run on top of Knative technologies in a transparent way. Some of the components of the CloudButton project have already started the convergence towards Knative. For instance, IBM-PyWren has support for running over Knative on any Kubernetes cluster [55], and Faasm is already integrated with Knative [56]. There are ongoing efforts in Crucial (URV, IMT) to deploy it over Knative clusters.

ATOS's efforts are focused on instrumentation and SLAs of Knative clusters using K8s technologies. This means that all their work is compatible with the different software stacks built on top of it.

Also the testbed provided by ATOS is build around Knative tachnologies.

RHT Infinispan has actively worked in the integration of Infinispan with K8s technologies, first providing a K8s operator, and then providing an auto-scaling service that can be integrated in a Knative cluster. Redhat also improved the interoperability of Infinispan with other programming languages. Again, all pillars can benefit from an Infinispan in-memory service deployed in the Knative cluster and offering auto-scaling support.

Finally, an essential service for interconnecting and integrating different services is Serverless Orchestration relying on Knative and CNCF CloudEvents standards. We refer here to WP3 D3.2 section on Serverless Orchestration, and in particular to the recent publication title "TriggerFlow: Trigger-based Orchestration of Serverless Workflows" [57]. In this work, we present an integrated event-based architecture that enables the orchestration of Cloud technologies.

IBM and URV are working together in the integraton of serverless orchestration technologies using Knative Eventing, KEDA, but also serverless orchestrators for K8s like Argo and KubeFlow. In the end of the project, we will be able to orchestrate Big Data pipelines with stages using different technologies like Python, Java or FaasM.

The convergence towards Knative will enable the orchestration of serverless workflows using different components of the CloudButton architecture. In order to allow these workflows orchestration, each component will provide an invocation endpoint and termination events. Platforms like ARGO and Triggerflow (developed in the context of this project) will be used to put together workflow steps implemented with different serverless technologies and even other steps that require serverful components.

5.2 Early prototypes

In this section we will describe the different early prototypes that make up the main 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 are collaborating 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. See D3.2 [58] for a detailed description of the serverless infrastructure.

- Serverless Orchestration (Airflow, Kubeflow, Argo): IBM and URV are working 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. For example, we have extended Apache Airflow with new FaaS Big Data Operators [59] and we also have adapted Argo for Big Data pipelines on serverless containers.

- Mutable State Middleware (Crucial, Infinispan): IMT, URV, and RHAT have created a novel disaggregated mutable middleware including consistent data structures and programming abstractions for stateful Big Data analytics over Infinispan. We offer proof of concept machine learning algorithms packaged as Functions that can be executed and orchestrated by CloudButton Core in a K8s cluster. RHAT is working to integrate Infinispan and this middleware in the K8s stack, and it will provide controllers for flexible auto-scaling of ephemeral and replicated Infinispan clusters. For an in-depth description of the mutable state middleware see D4.2 [60]. Source code repositories: `https://github.com/danielBCN/crucial-dso` / `https://github.com/infinispan/infinispan`

- WebAssembly FaaS Runtime (Faasm): Imperial have created a novel lightweight and polyglot FaaS runtime over WebAssembly technology. This middleware offers code-shipping models where lightweight functions can be colocated and access local shared memory in an efficient

way. Faasm is described in detail in D5.2 [61]. Source code repository: `https://github.com/lsds/faasm`

- CloudButton toolkit multiprocessing API: a multicloud framework that enables the transparent execution of unmodified, regular Python code against disaggregated cloud resources. It provides the same API as Python's standard `multiprocessing` and `concurrent.futures` libraries. Source code repository: `https://github.com/cloudbutton/cloudbutton`

5.2.1 Multi-cloud support

Current serverless computing platforms range from the traditional Functions-as-a-Service [62, 63] to the newest Container-as-a-Service [64, 65] services. Moreover, tens of open-source frameworks backed by Kubernetes are available in the community [66, 67]. With such an array of options, it is not unreasonable for a user to wonder: "*Which serverless computing service should I use?*", particularly under the shadow of the vendor lock-in problem. Vendor-specific APIs and SDKs make it difficult for users to move their applications from one under-performing cloud to another, simply because switching to a better provider is deemed too costly. In this sense, the continuous evolution of serverless APIs and features does nothing but to further aggravate this problem. In such a scenario, a multicloud approach can bring the needed flexibility to leverage the best of each cloud platform and overcome vendor lock-in, so developers can fully enjoy the benefits of serverless computing.

Therefore, the CloudButton toolkit adopts a multicloud-agnostic architecture, in an effort to ensure portability and overcome vendor lock-in problems. The CloudButton toolkit enables transparent access for users to virtually unbounded multicloud resources as nothing more than writing a program with a familiar language.

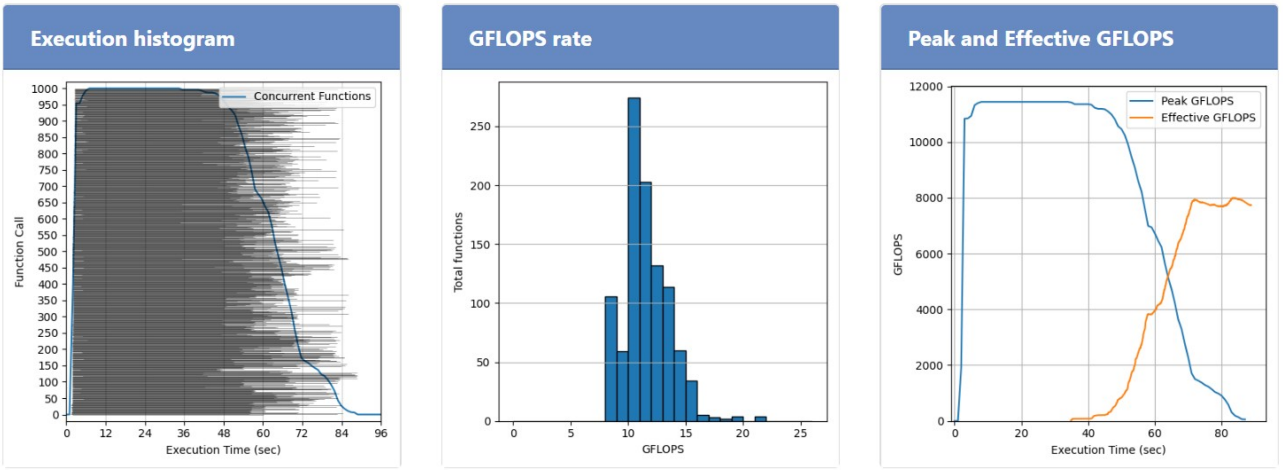


Figure 2: CloudButton Serverless Benchmark – IBM Cloud Functions results

An interesting example of the multi-cloud support is the CloudButton Serverless Benchmark (`https://cloudbutton.github.io/benchmarks/`), a helpful tool to compare serverless offerings (Compute, Storage) in different public cloud providers. The benchmark evaluates compute power and scalability of cloud providers by running multiple compute intensive tasks concurrently. It also evaluates the throughput of read and write operations to the object storage services of different cloud providers Currently, the benchmark supports 6 FaaS services (IBM Cloud Functions, AWS Lambda, Microsoft Azure Functions, Google Cloud Functions, Google Cloud Run, and Alibaba Aliyun Function Compute) and 5 Object Storage services (IBM Cloud Object Storage, AWS S3, Microsoft Azure Blob, Google Storage, Alibaba Aliyun Object Storage Service).

6 Testbed description

For the experimentation with components developed within CloudButton, the testbed has been set up with the following features:

6.1 Features of the contract with ITER

- Three bare-metal servers (see Table 2)

	node1714-2	**node1714-3**	**node1714-4**
CPU	16 Cores (32 threads)	16 Cores (32 threads)	16 Cores (32 threads)
Memory	32 GB RAM	64 GB RAM	64 GB RAM
Disks	3 x 6TB HDD	2 x 6TB HDD	2 x 6TB HDD
Network	2 x 1 GbE	2 x 1 GbE	2 x 1 GbE
OS	Ubuntu 18.04.4 LTS	Ubuntu 18.04.4 LTS	Ubuntu 18.04.4 LTS

Table 2: Testbed provided by ITER

- One private network connecting the three nodes.
- One public network C class.
- The offer includes outbound network traffic on a shared channel with no capacity limitations on the link available for commercial traffic. The link for commercial traffic has a maximum shared throughput of 2.000 Mbps.

In order to maximise the use of the **storage**, we have organised the disks on each of the nodes as follows:

- For nodes node1714-2 and node1714-3 we have set the disks on RAID1, to add up the total storage capacity (18Gb and 12Gb respectively):
- For node1714-4 we have one physical disk (6T) and one logical volume (5T) assigned to Ceph Object Store, as it needs dedicated volumes, while 1T logical volume is left for the operating system

6.2 Object Storage

6.2.1 Ceph

We have installed a Ceph Object Storage [68] cluster on the three nodes, making use for the storage of the disks (one logical of 5Tb and one physical of 6Tb capacity) in node1714-4 (Ceph Storage node). Ceph is ideal for storing unstructured data that can grow without bound. The cluster is made up by three Ceph MONitor nodes (node1714-2, node1714-3 and node1714-4), two Ceph ManaGeR nodes (node1714-2 active and node1714-3 standby), and two Ceph Object Storage Services (OSDs) on node1714-4. We have installed an Ubuntu VM to work as the Ceph RADOS Gateway [69] that provides the Ceph S3 API-compatibility, that is necessary for the CloudButton processes to access the object storage.

6.2.2 MinIO

In order to split the disk access of the different use-cases, and due to the fact that they have different needs, we have instaled Minio Object Storage [70]. This object storage is also S3 API compatible, and will allow us to make use of the storage of nodes node1714-2 and node1714-3 storage. Minio is a High-Performance Object and it can be configured as a native Kubernetes storage. We have used aws cli tool to access the object storage and verify S3 access.

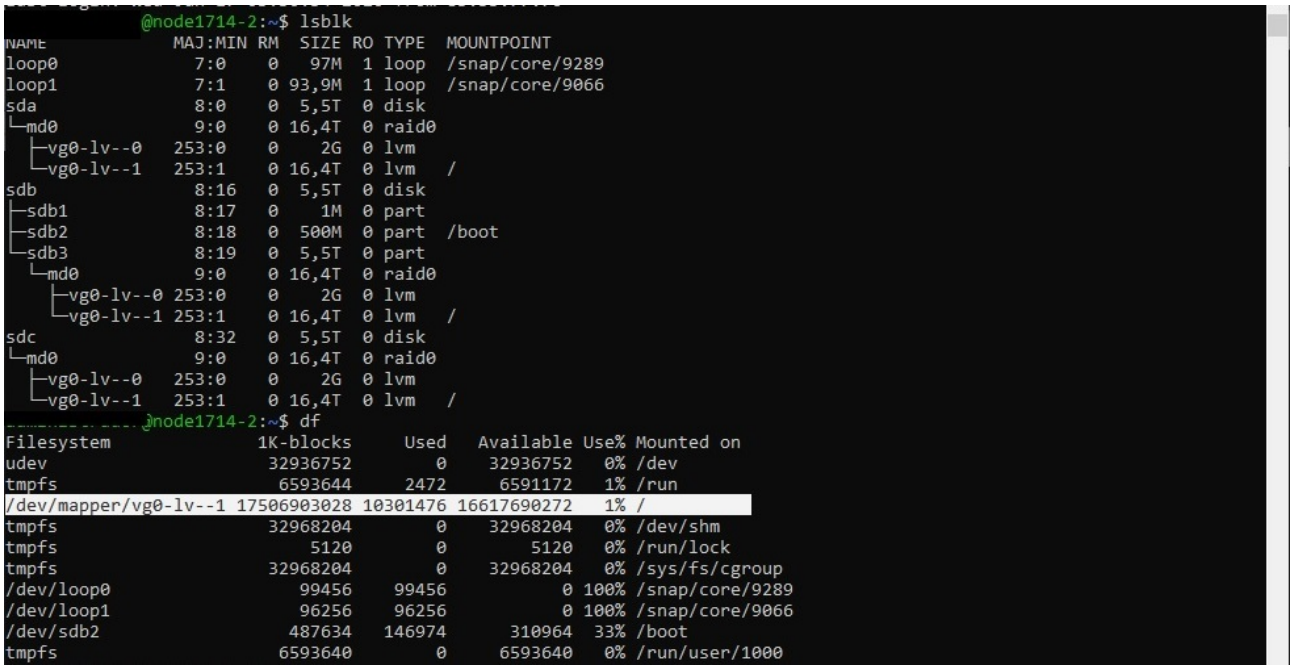


Figure 3: Storage structure of node1714-2.

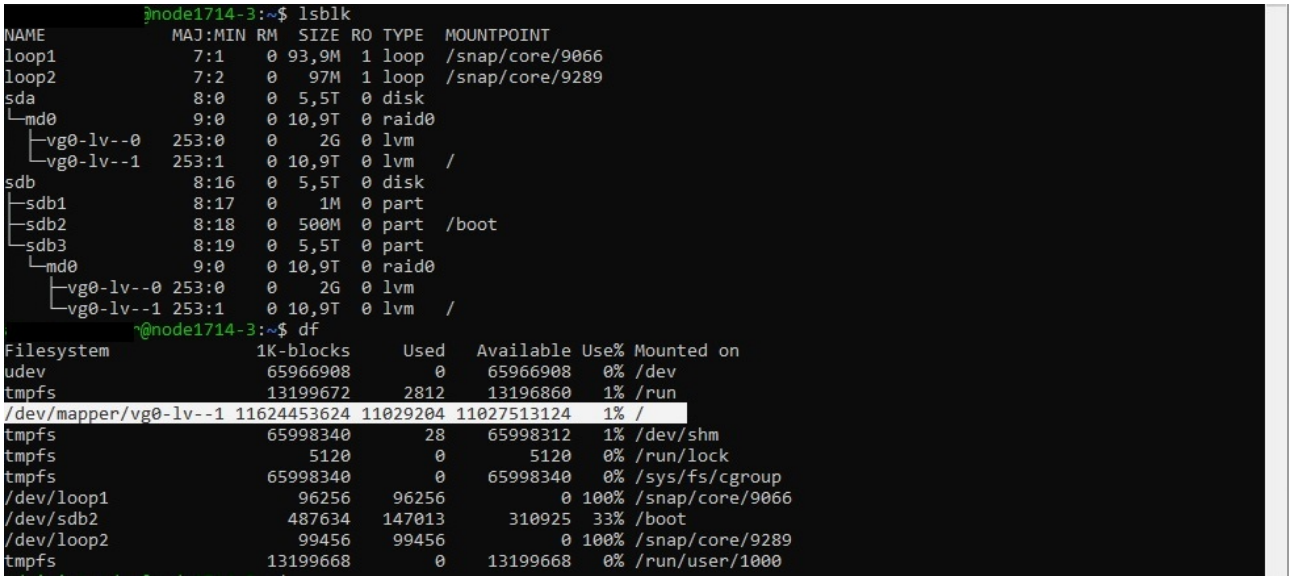


Figure 4: Storage structure of node1714-3.

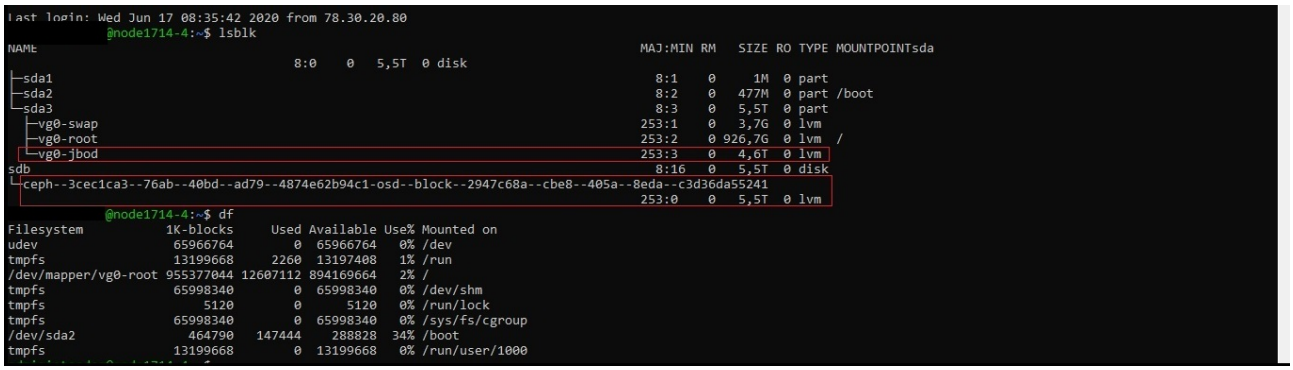


Figure 5: Storage structure of node1714-4.

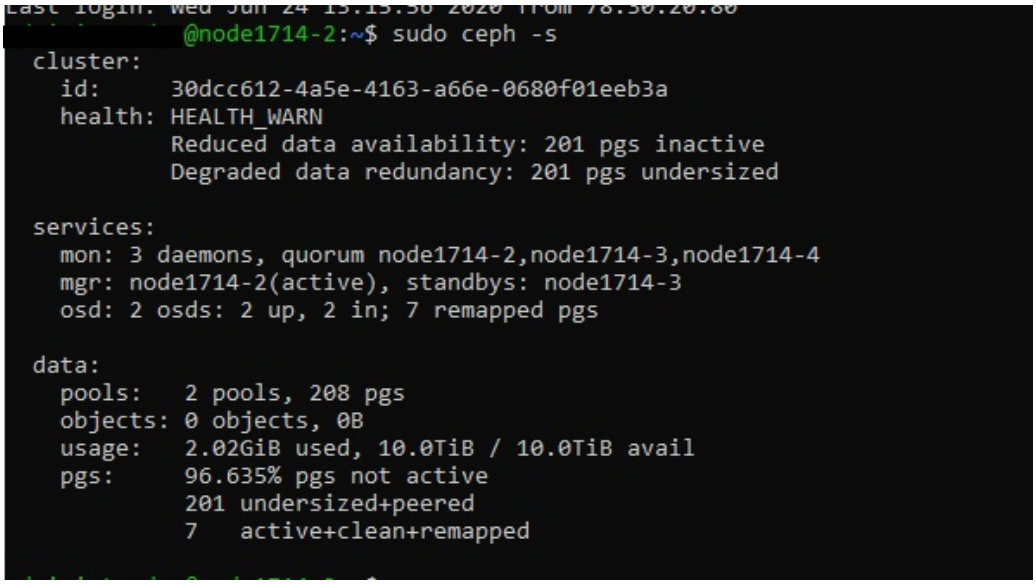


Figure 6: Ceph health.

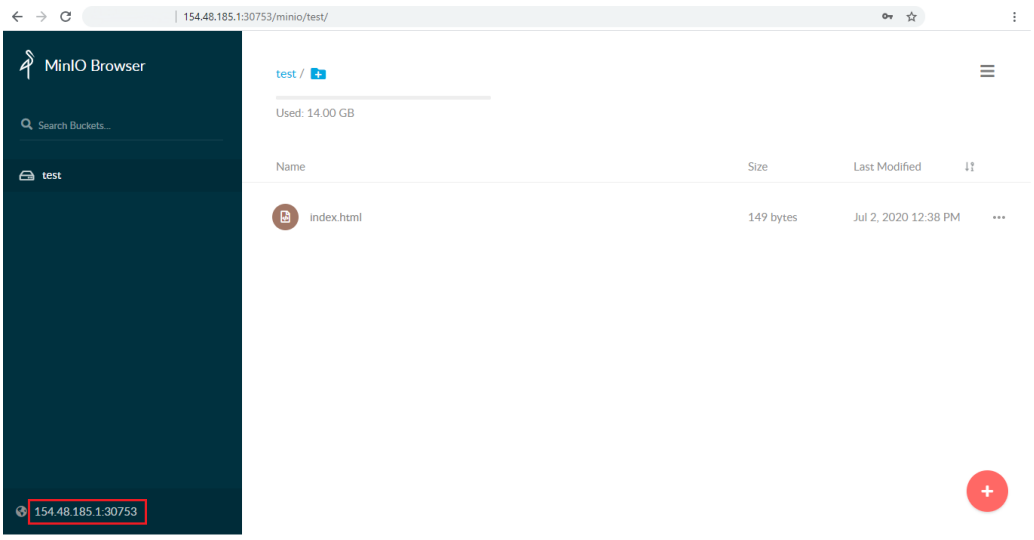


Figure 7: MinIO on node1714-2.

```
@node1714-2:~$ kubectl get nodes
NAME          STATUS   ROLES    AGE    VERSION
node1714-2    Ready    master   125d   v1.18.2
node1714-3    Ready    <none>   122d   v1.18.2
node1714-4    Ready    <none>   47d    v1.18.2
@node1714-2:~$
```

Figure 8: Kubernetes Cluster Nodes.

```
@node1714-2:~$ kubectl get pods --all-namespaces
NAMESPACE             NAME                                      READY   STATUS    RESTARTS   AGE
istio-system          istio-ingressgateway-6b94845f56-627bn     1/1     Running   3          43d
istio-system          istio-pilot-66dd769f9c-tfxgk              1/1     Running   3          43d
knative-eventing      broker-controller-67f49678fc-vpzlz        1/1     Running   2          43d
knative-eventing      eventing-controller-76fc696c98-vj24h      1/1     Running   3          43d
knative-eventing      eventing-webhook-6c76bcb89-4ktf2          1/1     Running   2          43d
knative-eventing      imc-controller-7b8fbcf6b-8s7n5            1/1     Running   3          43d
knative-eventing      imc-dispatcher-5fc6977498-8fpkj           1/1     Running   3          43d
knative-monitoring    grafana-686b5fcbf8-jzks2                  1/1     Running   2          43d
knative-monitoring    kube-state-metrics-b6bcff8f4-fsvgt        1/1     Running   2          43d
knative-monitoring    node-exporter-62sjn                       2/2     Running   4          43d
knative-monitoring    node-exporter-8fmrr                       2/2     Running   6          43d
knative-monitoring    node-exporter-xt59g                       2/2     Running   14         43d
knative-monitoring    prometheus-system-0                       1/1     Running   2          43d
knative-monitoring    prometheus-system-1                       1/1     Running   3          43d
knative-serving       activator-65fc4d666-q5j7k                 1/1     Running   2          43d
knative-serving       autoscaler-74b4bb97bd-4v7xv               1/1     Running   2          43d
knative-serving       controller-6b6978c965-82t48               1/1     Running   2          43d
knative-serving       istio-webhook-856d84fbf9-v7gdz            1/1     Running   3          43d
knative-serving       networking-istio-6845f7cf59-d8dn7         1/1     Running   2          43d
knative-serving       webhook-577576647-ct9qm                   1/1     Running   2          43d
kube-system           calico-kube-controllers-6b9d4c8765-rtmqq  1/1     Running   11         126d
kube-system           calico-node-cq65c                         1/1     Running   11         126d
kube-system           calico-node-pm4wq                         1/1     Running   5          49d
kube-system           calico-node-qcbc4                         1/1     Running   5          124d
kube-system           coredns-6955765f44-lh4kv                  1/1     Running   12         126d
kube-system           coredns-6955765f44-mj9g2                  1/1     Running   12         126d
kube-system           etcd-node1714-2                           1/1     Running   8          53d
kube-system           kube-apiserver-node1714-2                 1/1     Running   8          53d
kube-system           kube-controller-manager-node1714-2        1/1     Running   8          53d
kube-system           kube-proxy-gzs47                          1/1     Running   5          124d
kube-system           kube-proxy-jzfvj                          1/1     Running   11         126d
kube-system           kube-proxy-vmxh9                          1/1     Running   5          49d
kube-system           kube-scheduler-node1714-2                 1/1     Running   8          53d
```

Figure 9: Knative running pods.

6.3 Software installed on the nodes

After gathering internal requirements for the software to be available in the testbed, we decided to install a Kubernetes cluster with three nodes. node1714-2 will be the master of the Kubernetes cluster and the point of entrance to the testbed, from a ssh key-protected connection to a Public IP.

Knative v.0.14.0 serving and eventing [71] and the Observability Plugin [72] are also installed, in order to serve the CloudButton Core.

Knative Observability plugin is being used by the CloudButton-SLA component described on D3.2 [58] to obtain the metrics necessary for task T3.3 Instrumentation and QoS. The CloudButton-SLA is already installed on node1714-3, and the development and unit tests of the tool are made in there.

IBMPywren on Knative has also being installed [73]. Pywren functions are the basis of CloudButton Serverless Data Analytics Platform.

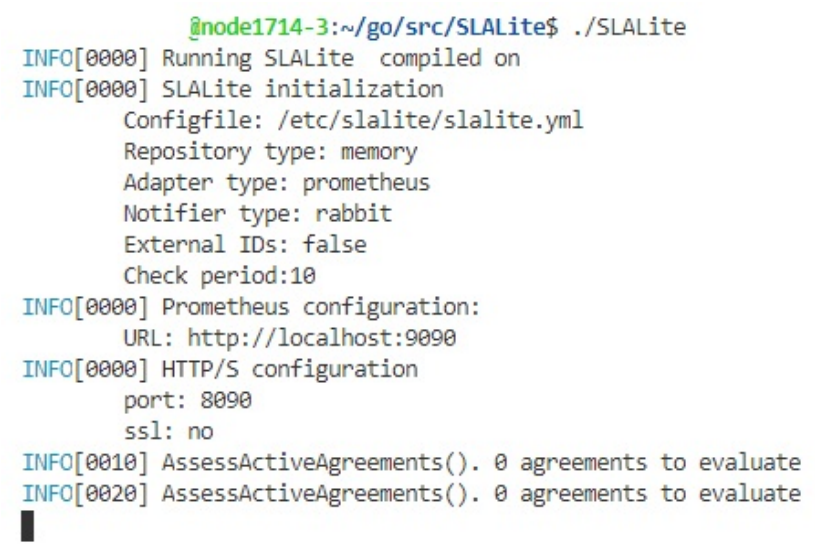


Figure 10: cloudbutton-SLA running on the testbed.

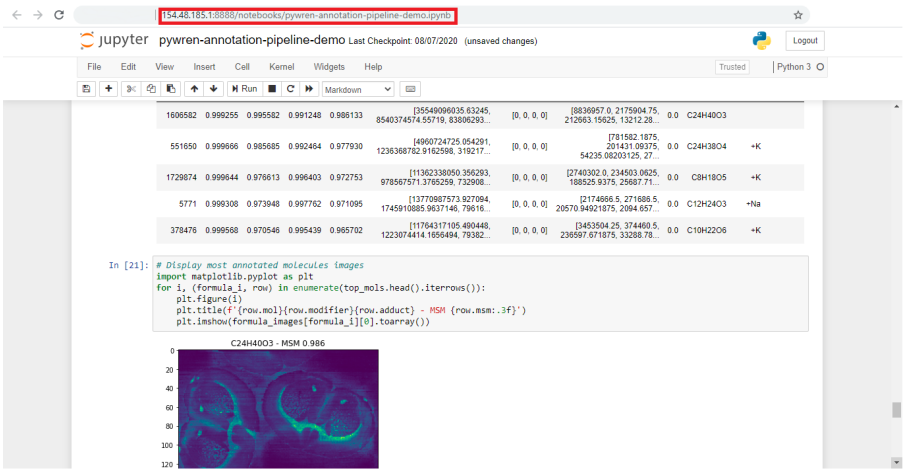


Figure 11: Output of annotated pipeline demo running on node1714-2 from IBM Cloud Functions.

6.4 Metabolomics Use case on the testbed

The metabolomics use case Pywren annotation pipeline[74] can run on the testbed. A JupyterLab[75] has been installed in node node1714-2 in order to run the Python notebooks that made up the experiments. From all the possible settings that Pywren implements, today we can use IBM Cloud Functions[76] and Knative as computer backends and IBM Cloud Storage[77] as storage backend. Thus, the pipeline makes use of Pywren to create a runtime that is executed in Knative Serving. The metabolomics use case experiments are being used to test the cloudbutton-SLA as described in Deliverable D3.2 [58]..

6.5 Future Work

- With the final implementation of Ceph for Pywren, metabolomics use case will run on ceph storage, totally independent from resources other that those provided by the testbed. Also, with the integration of Minio into Pywren, this object storage will also be tested.
- Use cases will be able to test the performance in a private Kubernetes cloud, and the cloudbutton-

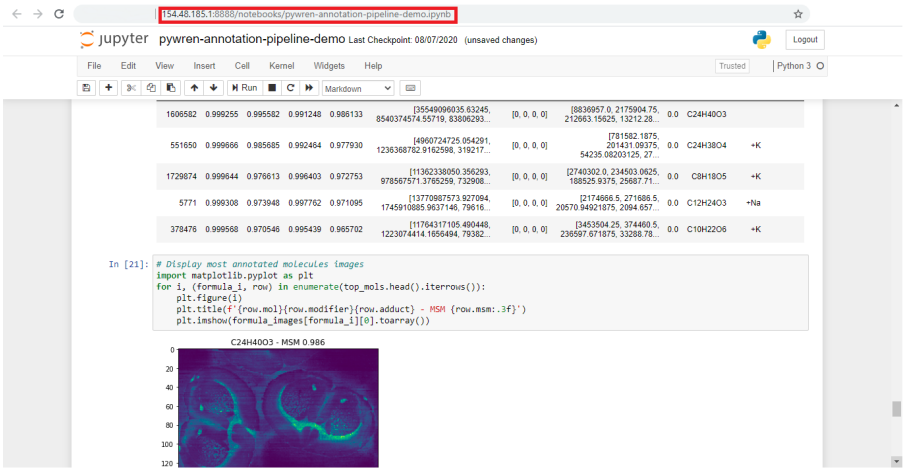


Figure 12: Output of experiment-1 running on node1714-2 from Pywren-Knative runtime.

SLA will be able to collect contextual metrics and perform the benchmarking to define an appropriated SLA for the performance of each of the applications, as described in D3.2[58]

- A backup scheduling for the data between the two Object Storages will be defined, to increase the high availability of the data.

7 Metabolomics use case

7.1 Experiments description

Metabolomics is an emerging field of biological and medical importance. Metabolomics complements genomics, transcriptomics, and proteomics in detecting and interpreting small molecules, the final readout of cellular programs. Metabolomics is already shedding light on cancer, diabetes, immunity, where reprogrammed metabolism is accepted to be a key hallmark, and carries an undisputed potential to unveil impact of microbiota and exposome on human health and disease . In the Horizon2020 project METASPACE (2015-2018), EMBL has developed a cloud platform for metabolite annotation in the cloud, that is currently being used by 100+ research labs worldwide and represents a rapidly growing collection of 8000+ spatial metabolomics datasets (Figure 13).

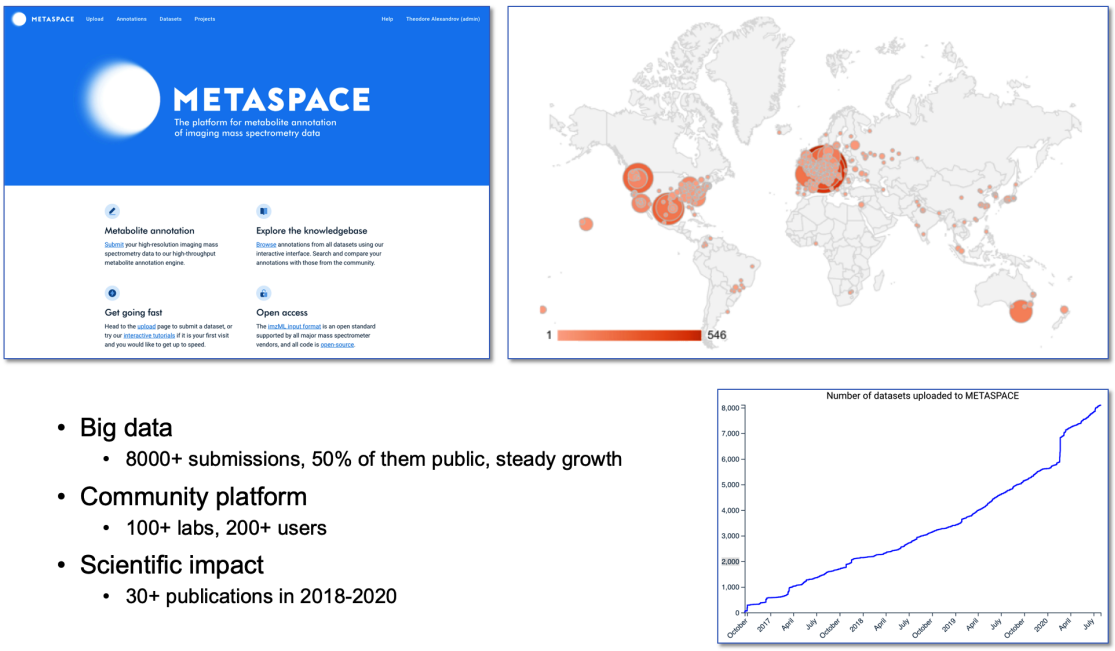


Figure 13: The METASPACE cloud platform (`http://metaspace2020.eu`), the geographical map of its worldwide usage, the growing numbers of submissions and the key points characterizing the big data aspects, its growth, community support, and scientific impact.

The METASPACE knowledgebase has two components: raw imaging mass spectrometry data and metabolite images produced from the raw data by our bioinformatics engine. Each raw dataset is a multi-gigabyte hyperspectral image of over 1 million of channels. Metabolite images are obtained from the raw data with the help of advanced custom bioinformatics and data-intensive computing currently performed on AWS using Apache Spark. The results represent a fraction of percent of the raw data and are accessed either through a web-app for interactive exploration or through a GraphQL API particularly by using Python Jupyter notebooks. However, currently most of the raw data is left unexplored, often called as "black matter" in spatial metabolomics. What is important for CloudButton is that the computational workflow of METASPACE includes custom algorithms and involves invokations of numerous repetitions of individual steps that makes it "embarrassingly parallelizable" (Figure 14).

7.2 Current status and next steps

In the current reporting period, we have achieved all the planned objectives.

Datasets: EMBL has prepared the benchmark datasets and shared them with the partners as well as publicly through the specially-setup repository: https://github.com/metaspace2020/pywren-annotation-pipeline#example-datasets. This included datasets from six tissue sections provided by

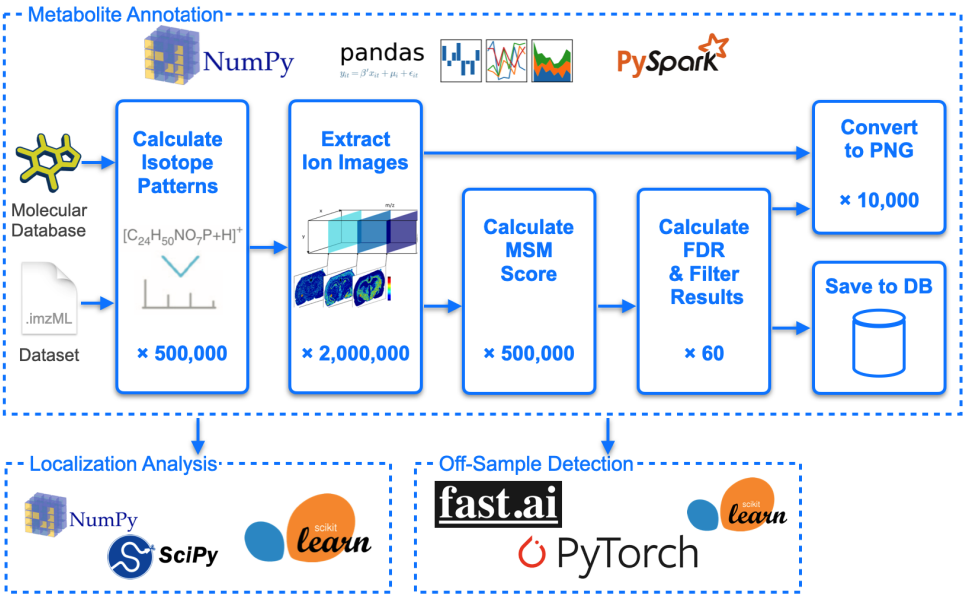


Figure 14: The workflow of the serverful METASPACE indicating the steps and the numbers of invokations of every step for processing of one dataset. Numerous repetitions of each step make this workflow a perfect target for implementation into a serverless version ("embarrassingly parallelizable").

EMBL and our collaborators, as well as 12 molecular databases which are used in METASPACE for molecular annotation. In every experiment, molecules from a database are searched for in a dataset. The datasets and databases were selected so that their combinations represent various scenarios: from small to typical to large to huge computing scenarios.

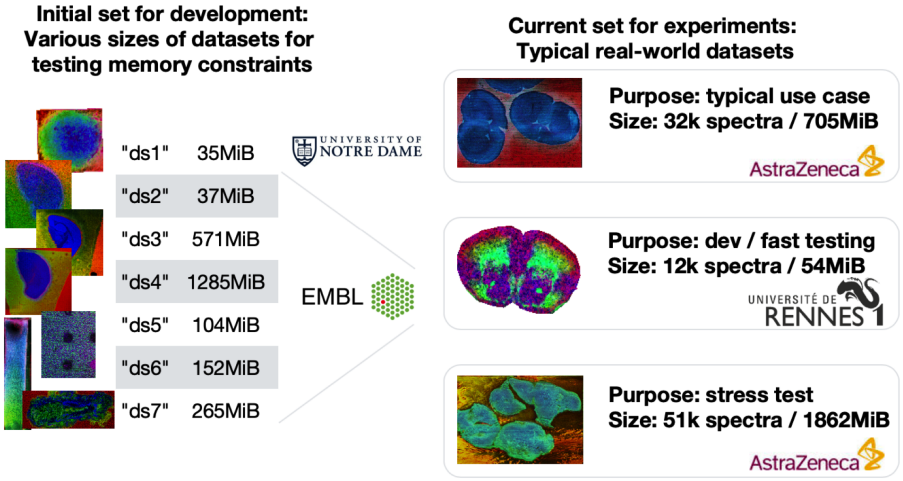
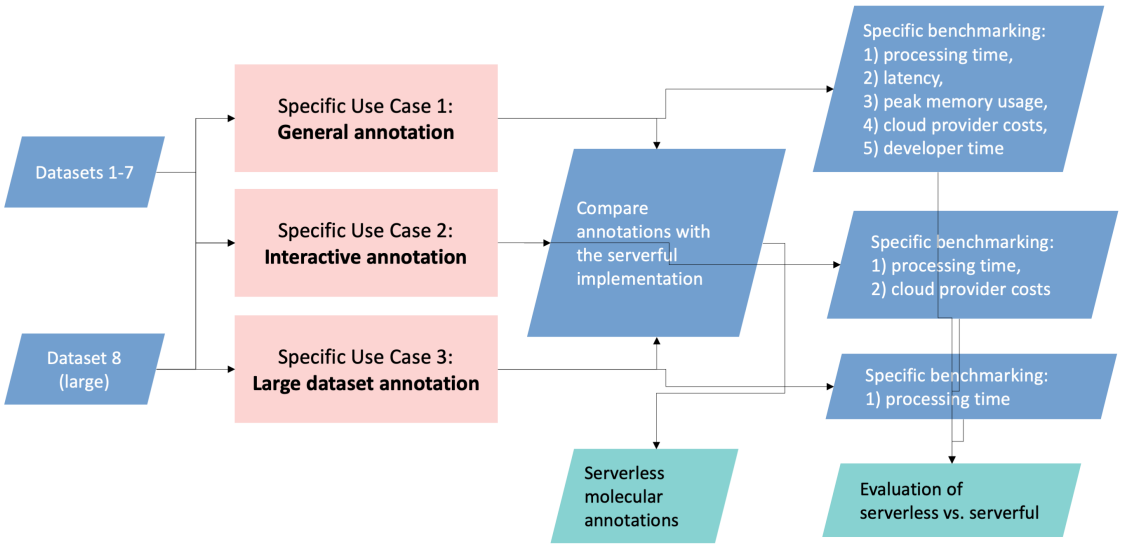


Figure 15: Overview of the datasets provided by EMBL and its collaborators for benchmarking the serverless implementation of METASPACE. With owners permission, we have made these datasets public through our GitHub `https://github.com/metaspace2020/pywren-annotation-pipeline`.

Experiments: EMBL has prepared three experiments representing the main computing scenarios in METASPACE. **Experiment 1, "Typical use case"**, is representative of a normal use-case on METASPACE, which makes it suitable for head-to-head comparisons. There is often limited time available on the higher-spec PC used for initial data capture as it is a shared resource, so usually the analysis will

be performed from scientists' or students' lower-spec laptops. **Experiment 2, "Interactive reprocessing"**, is representative of a new type of functionality that we currently don't support in METASPACE because it's uneconomical with the serverful approach. While looking for specific compounds, scientists tend to have relatively short lists of molecules of interest, and iteratively try different adducts or modifiers until they find the data they're interested in. **Experiment 3, "Stress test"**, aims to ensure that the limits of are similar to METASPACE, this is one of the larger datasets that has been processed. For every experiment, we have prepared the relevant datasets and databases as well as prepared the metrics to be used for benchmarking.

Metrics: EMBL has formulated the following metrics to be used for benchmarking: 1) total processing time, 2) latency for retrieving all images of target ions, 3) cloud provider costs, and finally 4) developer time. For every experiment, we decided which metrics are critical and which goals should be achieved (Figure 16).



Example notebooks

The main notebook is `pywren-annotation-pipeline-demo.ipynb`, which allows you to run through the whole pipeline, and see the results at each step.

There are also 3 notebooks prepared for benchmarking that can be run with Jupyter Notebook:

1. `experiment-1-typical.ipynb` - Demonstrates running through the whole Serverless metabolite annotation pipeline with a typical dataset,
   downloading the results and comparing them against the Serverful implementation of METASPACE.
2. `experiment-2-interactive.ipynb` - An example of running the pipeline against a smaller set of molecules, to demonstrate the potential of Serverless to provide low-latency access to computating resources.
3. `experiment-3-large.ipynb` - A stress test that runs the Serverless metabolite annotation pipeline with a large dataset and many molecular databases.

Figure 16: The specific use cases formulated for benchmarking of the serverless version of METASPACE, the decision diagram for evaluating the improvements compared to the state-of-the-art serverful version, as well as a screenshot from our GitHub repository where we have prepared Jupyter notebooks for every specific use case `https://github.com/metaspace2020/pywren-annotation-pipeline#example-notebooks`.

EMBL and IBM together have developed a prototype serverless implementation of the METASPACE. The implementation is available at our GitHub with 273 commits as of July 2021 (Figure 17): `https://github.com/metaspace2020/pywren-annotation-pipeline`. The implementation was de-

ployed on the IBM Cloud and the benchmarking is in progress. For the benchmarking, EMBL and IBM have prepared Jupyter notebooks which use the serverless implementation of METASPACE: `https://github.com/metaspace2020/pywren-annotation-pipeline#example-notebooks`. Each notebook executes the serverless METASPACE and performs the benchmarking according to the specified metrics. Although the benchmarking is still in progress, based on our experience in porting the Apache Spark implementation of METASPACE into serverless using PyWren-IBM-Cloud, we can already say that the serverless technology and in particular PyWren-IBM-Cloud provides a competitive alternative to Apache Spark in terms of the code readability and ease of the development. It allowed us in a short time to implement a serverless version of METASPACE computing engine and promises to dramatically increase its scalability, reduce the costs, and minimize the deployment efforts.

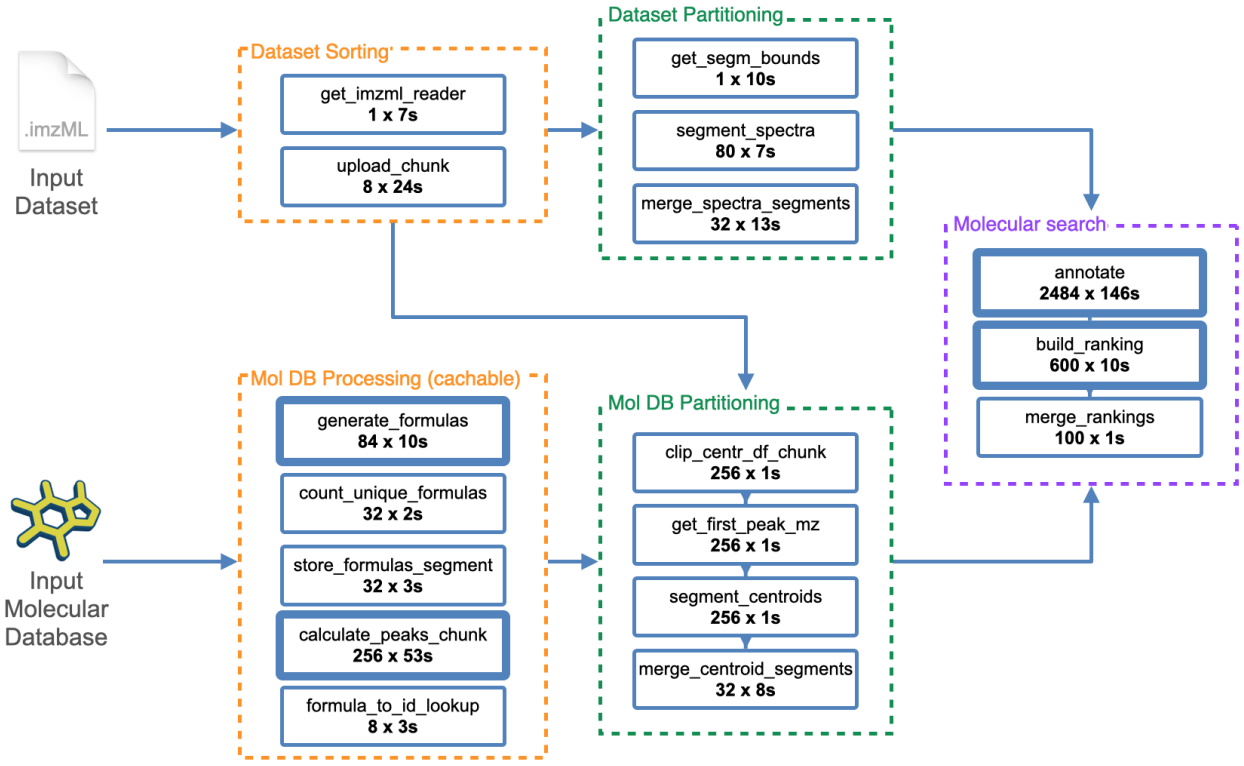


Figure 17: The architecture of serverless METASPACE showing preliminary estimates of times in seconds for processing an average dataset. The comprehensive benchmarking of serverless METASPACE is in progress and will be reported in the next reporting period.

8 Genomics use case

The importance of genomics to our society could hardly be overstated. An increasingly large number of applications have genomics at their focus — among them molecular cancer treatments, personalised medicine, and combating viral diseases. The recent worldwide reaction to Covid-19, and the promise to deliver an effective vaccine in a short time window, all hinge upon the availability of technology to sequence genomic material (nucleic acids) quickly and effectively (Figure 18).

Genomics is a data- and compute-hungry science, rooted as it is in sequence analysis. Genomes and their products (mainly DNA and RNA) are routinely sampled and decoded in order to establish how the cellular machinery works in different species. That has spurred the production of a staggering amount of sequence data, which results in several PB being added to global data stores every year (Figure 19).

In this situation, it is obvious very difficult for any single institution to be able to keep up. While high-performance computing (HPC) installations are becoming increasing commonplace in biology

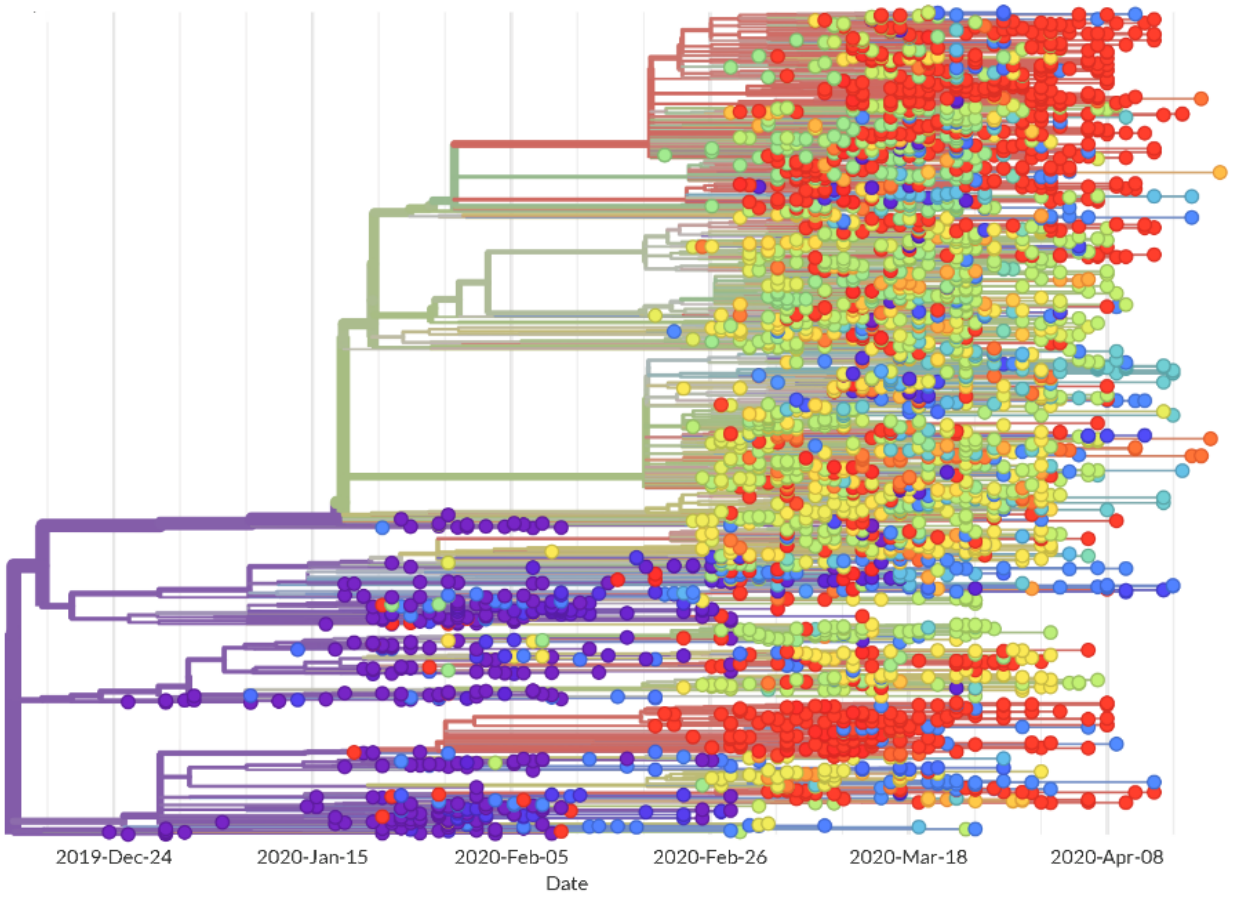


Figure 18: Phylogenetic tree of SARS-CoV-2, the virus causing Covid-19. Each point corresponds to a viral sequence obtained from high-throughput sequencing data (source: NextStrain).

departments, providing enough capacity to support demand peaks or (re-)analyse public datasets produced by large consortia is becoming increasingly challenging. The cloud would be an obvious solution to this problem, as it can supply enough elasticity to accommodate exceptional demand in HPC power without the need for scientific institutions to permanently acquire and maintain the corresponding physical infrastructure.

However, traditional cloud services are typically much more expensive than buying computing hardware, which discourages medium- and large-sized institutions from giving up their HPC clusters. With its offer of virtually unlimited computing power for a fraction of the cost of regular cloud services, serverless computing might provide a cheaper, more scalable and more attractive alternative adequate for genomics.

8.1 Experiments description

The genomics use case is being developed by BioSS, a scientifically independent sub-institute of The James Hutton Institute (and previously by The Pirbright Institute – the Pirbright PI moved to BioSS/JHI a few months after the beginning of the CloudButton project, resulting in a change of consortium partner). The use case revolves around using serverless cloud computing to facilitate large data analyses that are needed to contain, control and eliminate economically and medically important diseases within a One Health approach.

One-Health sees all the components of an ecosystem as intimately related – as Covid-19 and other pandemics clearly show, all stages of the viral life cycle and their direct and indirect effect on all potential host species must be taken into account as a whole. Hence, for a viral disease, possible interactions with vectors, different reservoirs, zoonotic spillages of infection from animal to human

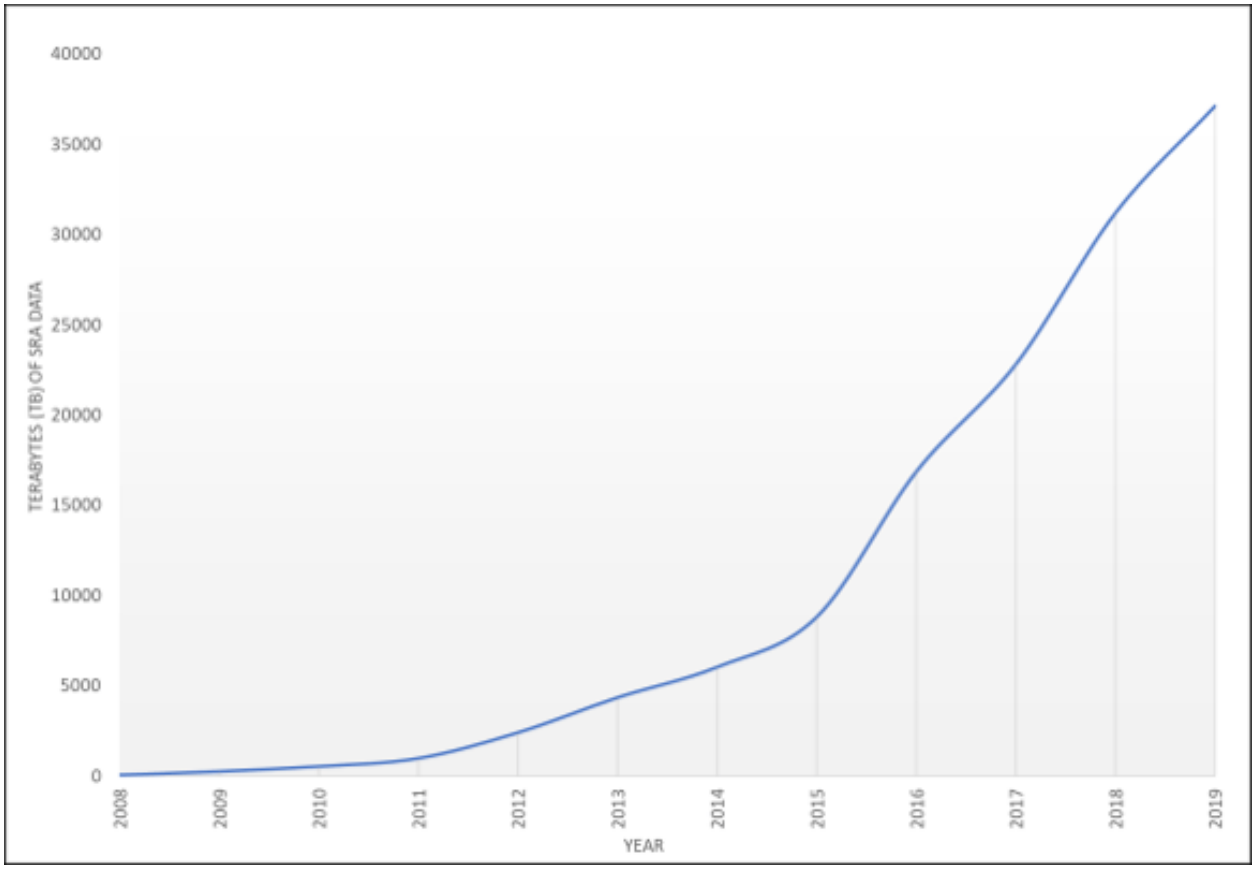


Figure 19: Time evolution of the amount of sequencing data stored by NCBI in the Sequence Read Archive (source: SRA).

hosts, and consequences on agriculture and human development, are all subjects of study. Whenever policy recommendations are needed by deciders in the government and other public bodies, the insights resulting from One-Health are likely to be much more effective than those produced by other approaches.

Our original approach focused on a use case whereby, due for instance to the emergence of a novel pathogen, a researcher had to quickly (re-)analyse in the cloud a very large and heterogeneous dataset touching upon different biological aspects of the pathogen's action. In order to simulate such a scenario, we proposed to re-process a number of publicly available datasets in the domains of:

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.

Virus-host interaction. That is obviously at the core of the response to a viral disease — understanding the mechanisms and details by which cells are differently susceptible to a given virus is essential to decipher and control the disease.

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.

After the inception of the project it was decided that the datasets at point (3) will be dropped, due to potential GDPR-related concerns formulated within the consortium.

Most of the datasets for points (1) and (2) are publicly accessible ones from the FAANG (Functional Annotation of ANimal Genomes) consortium [78, 79]. They provide genomic and functional information on livestock species (see [80]). Each file considered for those datasets is typically in the range of 10-30GB uncompressed, and during the first 6 months of CloudButton alone several thousand datasets have been added to the repository.

The experiments aim to demonstrate a number of facts:

- Prove that our existing genomics 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. In the scenario we are considering, such insights would arise from the comparison between our local datasets 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.

We will reprocess the datasets we have selected through our in-house bioinformatics pipeline (an evolution of the RNA-sequencing analysis pipeline based on the GEM mapper [81] used in [82]) re-implemented on the top of the CloudButton toolkit, and compare them to the results obtained by running the pipeline locally on the HPC cluster shared between BioSS/JHI and other UK research institutes.

The first basic step performed by most analysis workflows, and the first one we would like to migrate to the cloud, is called the *alignment* of sequence reads to a reference genome. By this we mean that, given a short chunk of DNA produced by a sequencer (a *read*), we want to find all the regions in the genome having a similar sequence, up to a pre-determined amount of differences between the read and the identified region. Equivalently, we want to locate all the regions in the genome that might have originated the sequencing read, as the accumulation of reads allows us to identify parts of the genome which are functionally active under different biological conditions (see Figure 20). Doing so is a computationally expensive operation, which usually takes most of the time used by analysis pipelines – the reference genome can be large and repetitive, one has to perform string searches with mismatches due to possible errors in the read or the reference, and billions of reads need to be processed, possibly more than once, for each experiment.

In fact, each RNA-sequencing read needs to go through a much more complex analysis workflow than just alignment, due to the biology underlying transcription of DNA into RNA. The basic processing steps can be described as follows:

1. First, the read is aligned continuously, from its beginning to its end, to the reference genome
2. Second, the read is continuously aligned to the transcripts present in the annotation, if such annotation has been provided
3. Third, the read is 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

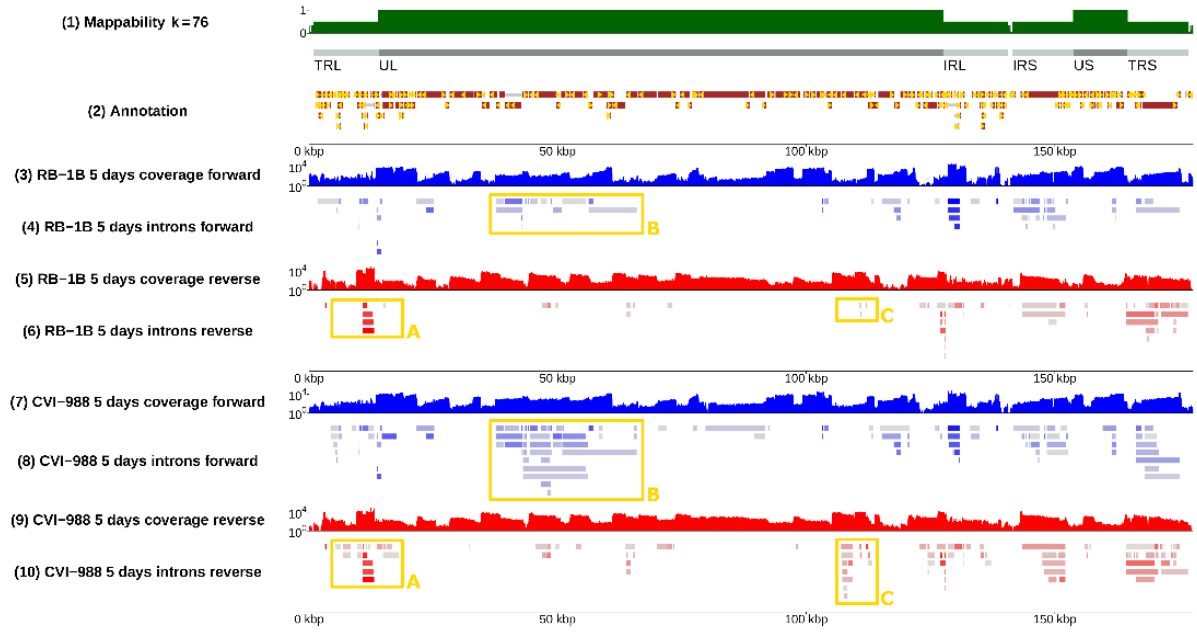


Figure 20: Transcriptional landscape of Marek's disease virus, a virus causing cancer in chicken and large economic losses to the poultry industry. The plot show the results of several RNA-sequencing experiments as a genome browser. Local accumulations of reads due to alignment produce a biologically relevant signal.

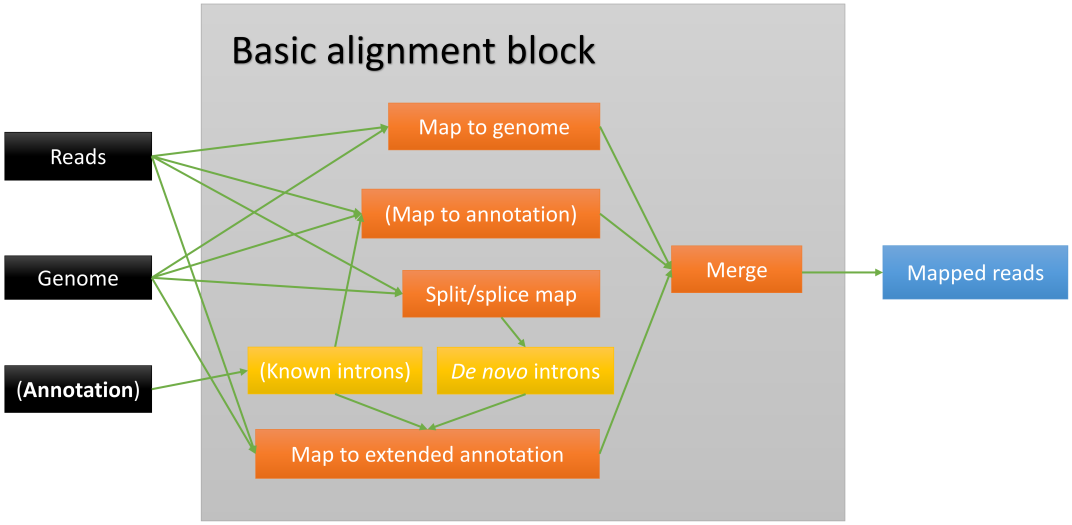


Figure 21: The computational structure of the basic sequence alignment step as performed by our workflow to analyse RNA-sequencing data.

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

4. 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

5. 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 lower the final score).

Each step of the alignment block requires a highly optimised piece of HPC code. The code base for the GEM mapper alone comprises several tens of thousands of lines of C code, and other steps are implemented as a large library written in the OCaml [83] functional programming language.

Unfortunately, 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. Unfortunately, such large amounts of memory are not generally available in a serverless framework. Another problem is the fact that at the moment there is no support for OCaml in WebAssembly.

8.2 Current status and next steps

The initial stages of the genomics use case have been executed according to the original program – in particular resulting in the successful formulation of part of Deliverable 2.1.

Unfortunately, further progress in the genomics use case has been hindered by the move of the Pirbright PI to another institution, BioSS, roughly six months after the start date of the CloudButton project. Due to that, hiring personnel at Pirbright was judged unwise and postponed. At the same time a partner change from Pirbright to BioSS was initiated, and took several more months to be completed. When the funding was finally transferred to BioSS in February 2020, Covid-19 hit, resulting in a recruitment ban which was lifted only very recently (end of June 2020). As a result, very small amounts of the original budget have been spent so far. Recruitment is now under way, and a suitable plan is being put in place in order to make up for the time lost so far.

Despite all these problems, in the meantime Pirbright/BioSS was able to team up with the Large-Scale Data and Systems Group at Imperial and instigate substantial initial progress in the porting to serverless of the key alignment block of Figure 21.

It is possible to devise several algorithmic ways of circumventing the problem of the large index to be loaded into memory prior to alignment. One of these is the idea 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 (see Figure 22). The latter stage is compute intensive and relatively complex from the standpoint of data transport, as it requires data produced on a number of machines located anywhere in the cloud to be collected and processed at the same place. 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.

Together with the porting of the GEM codebase to WebAssembly, which has also been successfully performed by Imperial, this design represents a first, fundamental ingredient towards migrating to serverless architectures arbitrarily complex workflows for the analysis of sequencing data. Eventually the analyst will be able to choose between local execution, when available resources are sufficient, and remote processing on the cloud, whenever the situation requires very fast processing of a very large amount of data. The tools will certainly be more and more useful in the future, as our biosecurity is challenged by novel and increasingly aggressive threats.

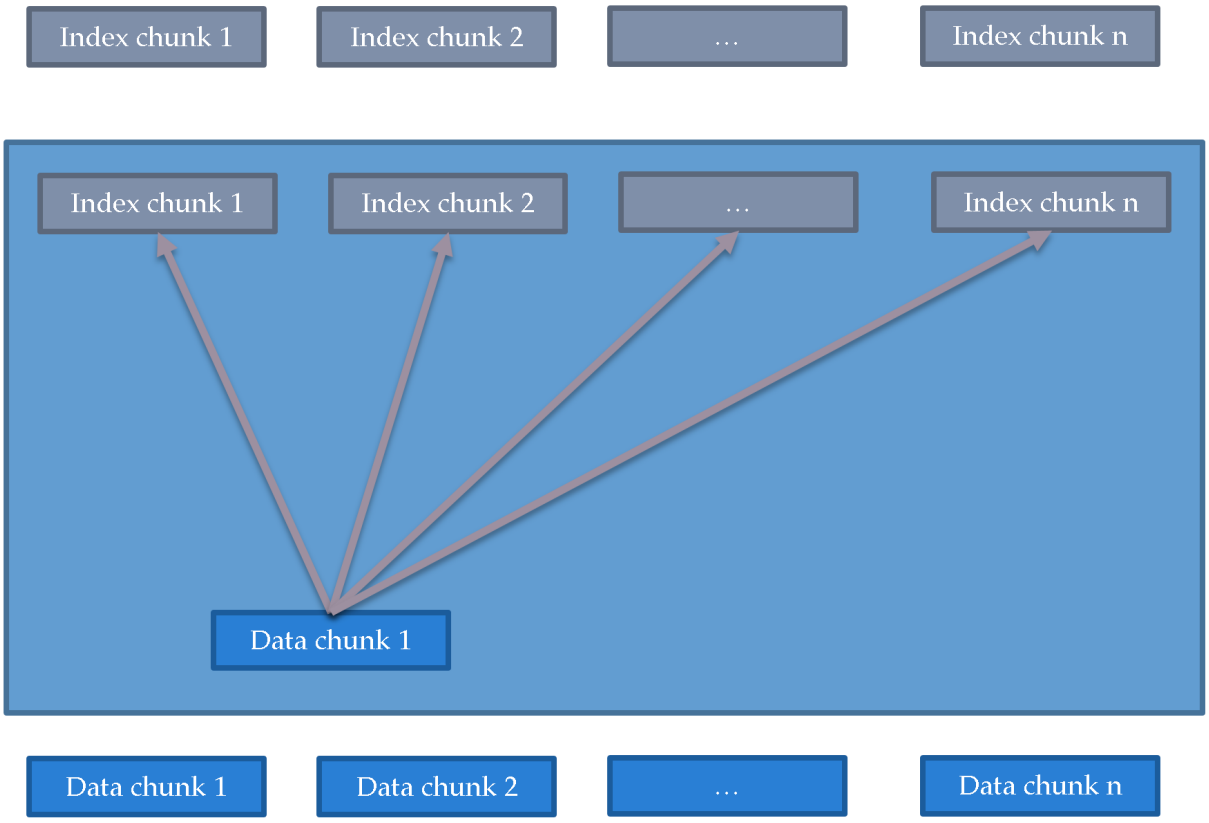


Figure 22: A scheme to parallelise sequence alignment and eliminate the problem of the stateful genomic index.

9 GeoSpatial use case

9.1 Experiment 1 and 2

The first two experiments of this use case, described in Deliverable D2.1, are the following:

- **Experiment 1: 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 have carried 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, and after that, the land-cover discrimination outcome will be refined by LiDAR-derived 3D metrics using this toolkit has been performed in local storage. Data acquisition as well as pretreatment and data processing with the results for each use case has been developed.
- **Experiment 2: 3D fuel mapping for forest risk assessment:**. We have used the same source of satellite and LiDAR big data described in the first experiment. We have generated 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) have been used as input data in spatial fire risk modeling, that also benefits from high resolution elevation models and other topographic input variables. The outcome 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). See Figure 23.

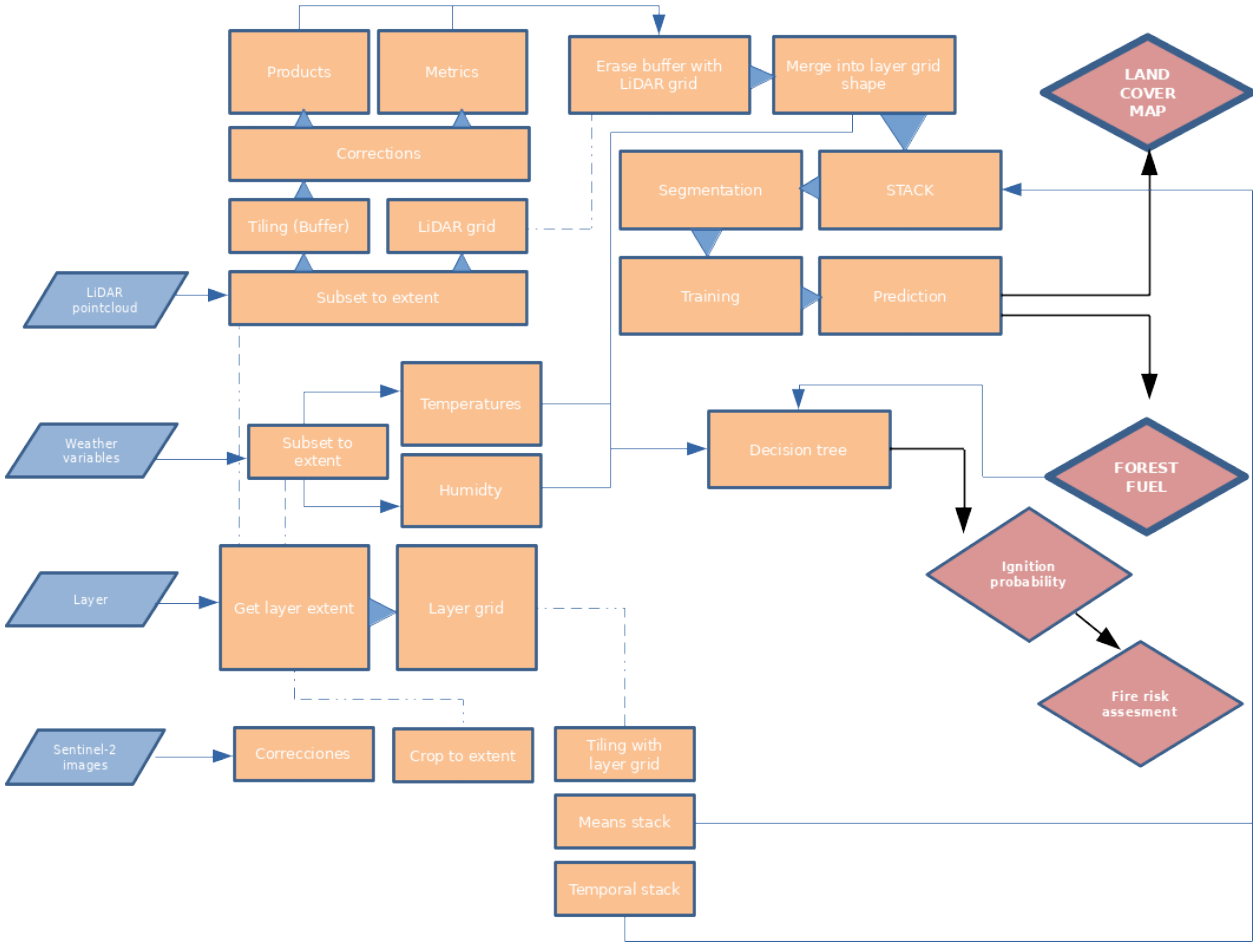


Figure 23: Geospatial use case. Pipelines of the experiments 1 and 2

9.1.1 Implementation

A series of scripts were developed to automate the downloading of sample data. There is a script for each data source to consult. The results of each of the scripts are stored in the corresponding directory directory "inputs/results". Some of the developed scripts are the following:

- **Script lidar.py**: This module contains useful methods to download images
- **Script read_dbLiDAR.py**: Creates a .laz file from a sql query to a postgresql database with the option, recommended, of indicating the maximum extent or the path to a vector layer from which to get the extent. If the created file it's too big, must be divided into tiles afterwards.
- **Script read_temperatures.py**: Returns a shapefile with temperature points from a database. The contains AEMET temperature data for the period 1971-2017.
- **Script upload_lidar.py**: This module contains a method for uploading point clouds.

The next step developed was the treatment of the data to generate samples and classes.

The script `fcc.py` creates the ratio of one class with respect to the all the points based on the calculation of the Covered Fit Fraction (FCC), presented by García et al [84]. It will depend on the raster entered in the function if the FCC, TCC or SCC is calculated.

Starting at this point we can proceed with the Machine Learning approach. Here we will carry out the training and prediction with data in GPKG files. A Docker container with Spark 2.4 pre-installed has been used and Geopandas 0.4.1 has been installed. Geopandas is used to read the

GPKG and Spark format for Machine Learning: train the model and generate the prediction. The docker container used is `jupyter/pyspark-notebook`. Firstly, we generate an ML model to determine land use, then we will use the ML model generated with the training script to try to predict land use.

Then we proceed with the validation. Scikit-learn 0.20.3 and Geopandas 0.4.1 have been used. Geopandas is used to read the GPKG format and Scikit-learn to verify the results using "Cross Validation" to validate the model.

Preprocessing section: Here we classify the overlap points as noise, this way it is easier to ignore them in the subsequent processes.

```
import pandas as pd
import numpy as np
from geopandas import read_file,GeoDataFrame
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier

datosGlobales=read_file("../prediccionV2/segmented_TopFixed.gpkg")
datosEntrenamiento=read_file("../prediccionV2/ForClassifySegments.gpkg")
df=datosEntrenamiento[["cat","tipo"]].merge(datosGlobales,on="cat",how="left").dropna()

columnas=['area', 'perimeter',  'fd', 'compact_circle', 'B1_sum', 'B1_mean', 'B1_median', 'B1_stdev', 'B1_min', 'B1_max', 'B1_variance', 'B2_sum',

y=np.array(df['tipo'])
X=np.array(df[columnas])

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,random_state=0)

clasificador=RandomForestClassifier(n_estimators=30)

from sklearn.model_selection import ShuffleSplit,cross_val_score

cv = ShuffleSplit(n_splits=10,test_size=0.2,random_state=0)

evaluacion=cross_val_score(clasificador,X,y,cv=cv)

print("EN %d PREDICCIONES\nACIERTO media:  %0.2f, desviación típica: %0.2f" % (cv.n_splits,evaluacion.mean(),evaluacion.std()*2))
```

Figure 24: Sample script: validation of the ML model to determine land use.

9.1.2 Current progress

The serverless model provides us with many benefits for these use cases, once the model has been trained we can perform the classification on new maps and specific regions simply by sending a trigger to our preconfigure system, this allows us to save a lot on costs since the system would not have to be running constantly as would be the case with the traditional virtualization system.

During the first months of the project, important progress has been made regarding the first two experiments. Difficulties with different libraries and the parallelization of processes have been resolved achieving the development of local use cases successfully.

Fundación Matrix had recruitment problems which were aggravated by the existing health crisis. This is the reason for not having implemented the experiments in CloudButton. However, at the moment, we are actively working on it to obtain the expected results as soon as possible.

IBM Cloud service has been recently configured. The next goal is to work in a serverless paradigm, the code migration to the new scenario, use IBM-PyWren to parallelize tasks and to do preliminary tests for running several functions along with scaling up use case experiments. The expected result will be a clear improvement of map quality in comparison with current CLC mapping. 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.

9.2 Experiment 3: Water Consumption

The use case for this experiment was specified in D2.1 Experiments and Initial Specifications [5], chapter 5.

The use case consists in comparing water use estimates obtained from two different databases over extended regions of irrigated crop fields and then map differences in the water use footprint of irrigated arable lands in representative large areas of Peninsular Spain.

On the one side, we use high-resolution NDVI index (Normalized Difference Vegetation Index) derived from satellite imagery, we identify actual irrigated crop areas and we estimate water consumption using multi-date imagery data along the growing season. The continuous update of open-access databases and the utilization of the CloudButton capabilities make possible the mapping these variations along a certain period of time with frequent updates. Originally the intention was to combine these results with LiDAR data (Laser Imaging Detection and Ranging), a remote-sensing technology, from PNOA to discard tree vegetation areas and refine the model. LiDAR data were finally not used because there were not needed for this experiment.

On the other side, we estimate and map water consumption indicators considering the officially declared and georeferenced irrigated arable land area which is available from SIGPAC, the Geographic Information System for the Agricultural Common Policy open access database and specific correction factors (irrigated land area where to calculate the crop water consumption volume).

The comparison of both results identifies non-coincident areas which help to monitor water use efficiency and funding resource allocation.

An inventory of the data sources needed for this experiment was performed:

- to select the region to work with it, we use administrative regional data, i.e. National Geographic Institute (CNIG-IGN).
- to extract multispectral bands and calculate indexes we use Sentinel-2 data from Copernicus constellation of satellites (European Space Agency).
- to estimate the kind of crop in the selected area we use SIGPAC, the spatial information contained in the Geographic Information System of the Common Agricultural Policy.
- to get the meteorological information data, we use different meteorological stations located in the study area:
  - Agricultural Information Service of Murcia (SIAM)
  - Spanish Meteorological Agency (AEMET)
- All the above data are used to estimate the water requirements of the crops.
- to evaluate the water estimations that the experiment provides as output, we use *Irrigation communities* information.

9.2.1 Script development and notebook creation

The following scripts were implemented for:

- Downloading data from satellite and other sources:
  - Download images Sentinel-2 web Copernicus
  - Download LiDAR web CNIG data (finally not used in this experiment)
  - Download different meteorological data sources
  - SIGPAC data request
- Correction and filtering of data:
  - Radiometric correction in Sentinel-2 images
  - Geometric correction in images Sentinel-2
  - Topological filtering of SIGPAC data

- For selection of areas:
  - Selection of work zones to homogenize the data
- For calculation of parameters including water consumption:
  - Calculation of agricultural SIGPAC areas
  - Calculation of irrigated SIGPAC areas
  - Calculation of NDVI (Normalized Difference Vegetation Index): NDVI (Normalized Difference Vegetation Index) is a simple graphical indicator that can be used to analyze remote sensing measurements, often from a space platform, assessing whether or not the target being observed contains live green vegetation.
  - Calculation of average NDVI (per area and per month)
  - Calculation of NDVI statistics at the SIGPAC plot level
  - Evapotranspiration: from NDVI and Kc coefficient evapotranspiration can be calculated and therefore water consumption.

We implemented other scripts in Phyton: preparation of meteorological maps, Areas of cultivation and changes in land use, Arboreal crops, and State of crops.

This project consist of eight notebooks, the three first notebooks explained are the ones meant to be used by the end user:

- **Data preparation**: This notebook will get the tiles captured by Sentinel-2 and will perform an atmospheric correction process to clean the result. The generated files will be then transformed into a cloud optimized GeoTiff format which will be uploaded to IBM Cloud Object Storage (COS).
- **NDVI Calculation**: processes previously uploaded files in Data preparation notebook to IBM Cloud Object Storage and coumputes NDVI (Normalized Difference Vegetation Index) in parallel using PyWren for IBM Cloud.
- **Average NDVI Calculation**: The current notebook processes previously calculated NDVI values in NDVI Calculation notebook in IBM Cloud Object Storage and coumputes an average NDVI by month or by SIGPAC zone.
- **Memory Profiler**: utility notebook used to measure memory consumed by a python script. Imposes serious memory limitations for containers being executed: each container can not exceed 2GB of RAM usage.
- **Multiprocess Data preparation**: a multiprocess version of Data preparation. This notebook will get the tiles captured by Sentinel-2 and will perform an atmospheric correction process to clean the result. The generated files will be then transformed into a cloud optimized GeoTiff format which will be uploaded to IBM Cloud Object Storage (COS).

There are two other notebooks:

- **Performance**: Comparison of NDVI Calculation: local vs cloud: This notebook aims to compare local calculation of NDVI index versus the cloud calculated version.
- **Water consumption**: computes an interpolation of temperatures in each pixel based on SIAM extracted data.

```
PyWren v1.3.0 init for IBM Cloud Functions - Namespace: Namespace-CloudButton - Region: eu_de
ExecutorID 2ce65f/4 | JobID M000 - Selected Runtime: cloudbuttonans/geospatial-runtime:3.7-v4 - 2048MB
ExecutorID 2ce65f/4 | JobID M000 - Uploading function and data - Total: 8.4KiB
ExecutorID 2ce65f/4 | JobID M000 - Starting function invocation: combine_calculations()  - Total: 10 activations
ExecutorID 2ce65f/4 - Getting results...
```

Visualization of results

```
plot_results(BUCKET, items[:6])
```

```
plot_results(BUCKET, evapotranspiration_results[:6])
```

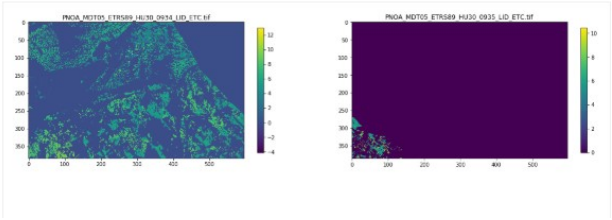


Figure 25: Visualization of results in the Water Consumption notebook.

9.2.2 Calculation of NDVI and average NDVI

Figure 26 represents the workflow to calculate the NDVI and the average NDVI.

There is a first phase of the process that requires the download and filtering of the base information. For this, the required dates of the products must be selected, as well as the portion of territory that is desired as a work area. Once this is done, the information is downloaded, but this process is slow and its parallelization is not possible. On the other hand, atmospheric and geometric correction is necessary and recommended to minimize distortions in the image that may be caused by clouds or alterations in satellite movement, sensor failure, etc. From this moment, when the starting information has been optimized, it can be uploaded to the cloud.

These cleaned data from the Sentinel-2 satellite serve, among many other things, to calculate the NDVI (Normalized Difference Vegetation Index). This process must be calculated every month of the year, in order to have a reference of the evolution of the crops in the study area.

On the other hand, the SIGPAC (Geographic Information System Common Agriculture Policy) plot is used, where the different crops that occur in the territory are officially reflected. This vector information also provides the parcel reference limits, so it constitutes very useful information to compare with the NDVI. This allows discrimination between arable land and tree crops, which means that, knowing the type of crop, the Kc value necessary to calculate their evapotranspiration can be established.

At this point, joining the NDVI (raster) and the SIGPAC (vector) plot, it is possible to calculate the mean of NDVI values for each of the plots.

SIGPAC data were not precise enough for the crop which is a requirement to calculate Kc coefficient but these SIGPAC data help to get to know two things: areas with crops and their geometry and extension.

9.2.3 Identification of parallel tasks

We performed calculations of sequential time, RAM and Space of all the scripts that were generated. We then performed a first identification of potential parallel tasks to be considered for the transition to serverless. We made a study and design for parallelization of NDVI calculation with PyWren for IBM Cloud.

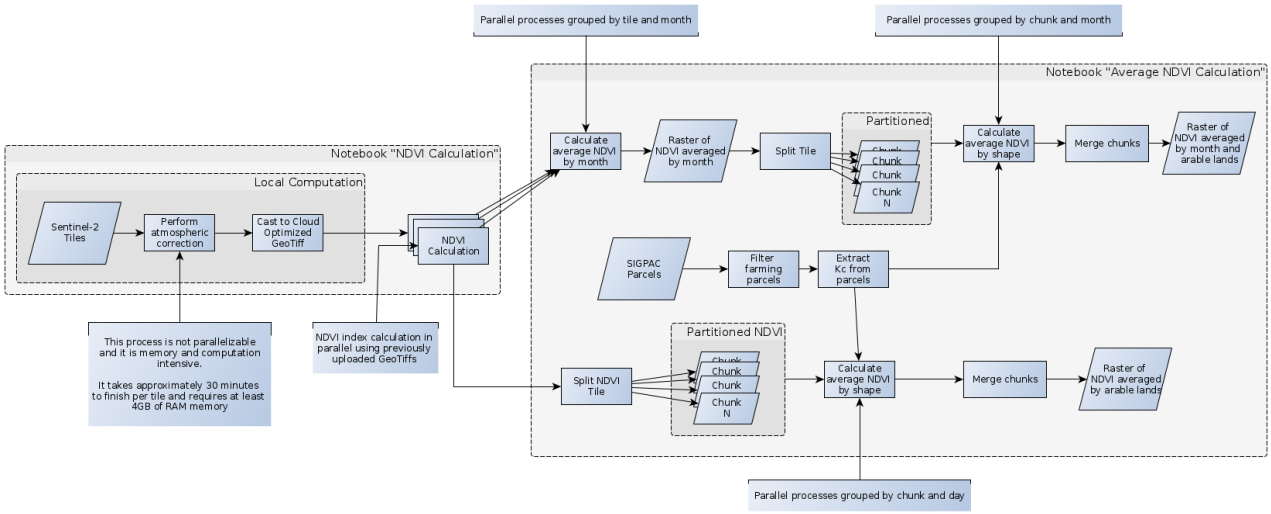


Figure 26: Workflow to calculate NDVI and Average NDVI

Image data transformation was optimized for cloud using Cloud Optimization GeoTiff file format. We went though the last step finalisation of the experiment to estimate the water consumption requirements of crops. And finally we performed the serverless implementation of the calculation of functional potential evapotranspiration.

All the scripts are available at the following source code repository: `https://gitlab.answare-tech.com/cloud-button/notebook/tree/master`

The works linked to the GIS (Geographic Information System) and Remote Sensing, by its very nature, cause large volumes of data to be handled, both in raster and vector formats. This implies, on the one hand, the need to work with increasingly precise data, or what is the same, with increasingly smaller pixels (raster). On the other hand, the ability to handle the information associated with vector data is also an increasingly prevalent need for companies and public administrations. This means that datasets are getting bigger and more complex. The ability to process and store all this information is another factor to consider, which limits, or makes the process very expensive.

The circumstances expressed justify that both private companies, research centers or public administrations, may require the technologies supported by CloudButton tools and serverless technology, since it greatly streamlines the workflow, avoiding bottlenecks when processing these bulky datasets, allowing projects to move forward and take on a greater workload.

In the presented use case, work has been done on a regional scale, but this technology would allow working on a national or continental scale with a unique agility, being of great help to all the technicians who are dedicated to these work areas and who, in many cases, see that their jobs or expectations are limited because they do not have sufficient technical capabilities available.

10 Conclusions

In this deliverable, we presented an overview of the project architecture, alongside with the project vision and Serverless Analytics and state of the art. We also described the current progress of the use cases and the validation testbed.

As aforementioned, the next steps will be focused on the integration of the different building blocks and software components. These building blocks are already starting to converge to use Knative as their common serverless platform. We believe that multitenant Knative technologies are going to revolutionize the serverless setting accross all cloud providers. An essential service for interconnecting and integrating different services is Serverless Orchestration relying on Knative and CNCF CloudEvents standards.

Another objective that will be further addressed during the second half of the project is the search of the full transparency to seamlessly port existing codebases to the serverless paradigm. The goal is to provide simple programming models for serverless cloud infrastructures targeted to existing data-intensive applications and implying as few changes as possible.

Likewise, we will devote efforts to ensure that CloudButton toolkit supports all major cloud providers, a key aspect to increase the adoption of the project software outcomes.

References

[1] 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.

[2] 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.

[3] 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," arXiv preprint arXiv:1812.03651, 2018.

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

[5] CloudButton Consortium, "Deliverable D2.1 - Experiments and Initial Specifications."

[6] 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.

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

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

[9] 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.

[10] S. Fouladi, F. Romero, D. Iter, Q. Li, S. Chatterjee, C. Kozyrakis, M. Zaharia, and K. Winstein, "From laptop to lambda: Outsourcing everyday jobs to thousands of transient functional containers," in 2019 USENIX Annual Technical Conference (ATC'19), pp. 475–488, 2019.

[11] 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.

[12] 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.

[13] D. B. Pons, P. G. López, Á. R. Ollobarren, A. Gómez-Gómez, G. París, and M. S. Artigas, "Faas orchestration of parallel workloads," in Proceedings of the 5th International Workshop on Serverless Computing, WOSC@Middleware 2019, Davis, CA, USA, December 09-13, 2019, pp. 25–30, ACM, 2019.

[14] 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.

[15] G. Coulouris, J. Dollimore, T. Kindberg, and G. Blair, Distributed Systems: Concepts and Design. USA: Addison-Wesley Publishing Company, 5th ed., 2011.

[16] J. Waldo, G. Wyant, A. Wollrath, and S. Kendall, "A note on diåstributed computing," in International Workshop on Mobile Object Systems, pp. 49–64, Springer, 1996.

[17] T. Wagner, "The Serverless SuperComputer," 2019.

[18] S. M. Rumble, D. Ongaro, R. Stutsman, M. Rosenblum, and J. K. Ousterhout, "It's time for low latency.," in HotOS, vol. 13, pp. 11–11, 2011.

[19] L. Barroso, M. Marty, D. Patterson, and P. Ranganathan, "Attack of the killer microseconds," Communications of the ACM, vol. 60, no. 4, pp. 48–54, 2017.

[20] A. Kalia, M. Kaminsky, and D. Andersen, "Datacenter rpcs can be general and fast," in 16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19), pp. 1–16, 2019.

[21] M. Kogias, G. Prekas, A. Ghosn, J. Fietz, and E. Bugnion, "R2p2: Making rpcs first-class datacenter citizens," in 2019 USENIX Annual Technical Conference (USENIX ATC 19), pp. 863–880, 2019.

[22] C. Lee and J. Ousterhout, "Granular computing," in Proceedings of the Workshop on Hot Topics in Operating Systems, HotOS '19, p. 149–154, 2019.

[23] P. X. Gao, A. Narayan, S. Karandikar, J. Carreira, S. Han, R. Agarwal, S. Ratnasamy, and S. Shenker, "Network requirements for resource disaggregation," in 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16), pp. 249–264, 2016.

[24] Y. Shan, Y. Huang, Y. Chen, and Y. Zhang, "Legoos: A disseminated, distributed OS for hardware resource disaggregation," in 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18), pp. 69–87, 2018.

[25] S. Peter, J. Li, I. Zhang, D. R. Ports, D. Woos, A. Krishnamurthy, T. Anderson, and T. Roscoe, "Arrakis: The operating system is the control plane," ACM Transactions on Computer Systems (TOCS), vol. 33, no. 4, pp. 1–30, 2015.

[26] F. McSherry, M. Isard, and D. G. Murray, "Scalability! but at what cost?," in 15th Workshop on Hot Topics in Operating Systems (HotOS 15), 2015.

[27] J. Gu, Y. Lee, Y. Zhang, M. Chowdhury, and K. G. Shin, "Efficient memory disaggregation with infiniswap," in 14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17), pp. 649–667, 2017.

[28] D. Barcelona-Pons, M. Sánchez-Artigas, G. París, P. Sutra, and P. García-López, "On the FaaS track: Building stateful distributed applications with serverless architectures," in Proceedings of the 20th International Middleware Conference, pp. 41–54, 2019.

[29] J. Zhi, R. Wang, J. Clune, and K. O. Stanley, "Fiber: A platform for efficient development and distributed training for reinforcement learning and population-based methods," arXiv preprint arXiv:2003.11164, 2020.

[30] K. Jayaram, V. Muthusamy, P. Dube, V. Ishakian, C. Wang, B. Herta, S. Boag, D. Arroyo, A. Tantawi, A. Verma, et al., "Ffdl: A flexible multi-tenant deep learning platform," in Proceedings of the 20th International Middleware Conference, pp. 82–95, 2019.

[31] S. Shillaker and P. Pietzutch, "Faasm: Lightweight isolation for efficient stateful serverless computing," in 2020 USENIX Annual Technical Conference (USENIX ATC 19), 2020.

[32] J. Nelson, B. Holt, B. Myers, P. Briggs, L. Ceze, S. Kahan, and M. Oskin, "Latency-Tolerant Software Distributed Shared Memory," in 2015 USENIX Annual Technical Conference, 2015.

[33] J. B. Carter, J. K. Bennett, and W. Zwaenepoel, "Implementation and Performance of Munin," SIGOPS Oper. Syst. Rev., 1991.

[34] G. Ananthanarayanan, A. Ghodsi, S. Shenker, and I. Stoica, "Disk-locality in datacenter computing considered irrelevant.," in HotOS, vol. 13, pp. 12–12, 2011.

[35] L. Liu, W. Cao, S. Sahin, Q. Zhang, J. Bae, and Y. Wu, "Memory disaggregation: Research problems and opportunities," in 2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS), pp. 1664–1673, IEEE, 2019.

[36] A. Dragojević, D. Narayanan, M. Castro, and O. Hodson, "Farm: Fast remote memory," in 11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14), pp. 401–414, 2014.

[37] J. Ousterhout, P. Agrawal, D. Erickson, C. Kozyrakis, J. Leverich, D. Mazières, S. Mitra, A. Narayanan, D. Ongaro, G. Parulkar, et al., "The case for ramcloud," Communications of the ACM, vol. 54, no. 7, pp. 121–130, 2011.

[38] Y. Huang, X. Yan, G. Jiang, T. Jin, J. Cheng, A. Xu, Z. Liu, and S. Tu, "Tangram: bridging immutable and mutable abstractions for distributed data analytics," in 2019 USENIX Annual Technical Conference (USENIX ATC 19), pp. 191–206, 2019.

[39] RISELab, "Apache Ray." https://github.com/ray-project/ray.

[40] P. Stuedi, A. Trivedi, J. Pfefferle, A. Klimovic, A. Schuepbach, and B. Metzler, "Unification of temporary storage in the nodekernel architecture," in 2019 USENIX Annual Technical Conference (USENIX ATC 19), pp. 767–782, 2019.

[41] A. Sebastian, M. Le Gallo, R. Khaddam-Aljameh, and E. Eleftheriou, "Memory devices and applications for in-memory computing," Nature Nanotechnology, pp. 1–16, 2020.

[42] L. Wang, M. Li, Y. Zhang, T. Ristenpart, and M. Swift, "Peeking Behind the Curtains of Serverless Platforms," in USENIX Annual Technical Conference, 2018.

[43] A. Agache, M. Brooker, A. Iordache, A. Liguori, R. Neugebauer, P. Piwonka, and D.-M. Popa, "Firecracker: Lightweight Virtualization for Serverless Applications," in 17th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 20), 2020.

[44] V. Sreekanti, C. W. X. C. Lin, J. M. Faleiro, J. E. Gonzalez, J. M. Hellerstein, and A. Tumanov, "Cloudburst: Stateful functions-as-a-service," arXiv preprint arXiv:2001.04592, 2020.

[45] S. Boucher, A. Kalia, D. G. Andersen, and M. Kaminsky, "Putting the "Micro" Back in Microservice," USENIX Annual Technical Conference (USENIX ATC), 2018.

[46] Microsoft Research, "Krustlet."

[47] A. Baumann, J. Appavoo, O. Krieger, and T. Roscoe, "A fork() in the road," in Proceedings of the Workshop on Hot Topics in Operating Systems, HotOS 2019, HotOS '19, ACM, 2019.

[48] Mozilla, "WASI: WebAssembly System Interface," 2020.

[49] T. Würthinger, C. Wimmer, A. Wöß, L. Stadler, G. Duboscq, C. Humer, G. Richards, D. Simon, and M. Wolczko, "One VM to rule them all," in ACM international symposium on New ideas, new paradigms, and reflections on programming & software, 2013.

[50] D. Buchaca, J. Marcual, J. L. Berral, and D. Carrera, "Sequence-to-sequence models for workload interference prediction on batch processing datacenters," Future Generation Computer Systems, 2020.

[51] J. Mace, P. Bodik, R. Fonseca, and M. Musuvathi, "Retro: Targeted resource management in multi-tenant distributed systems," in 12th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 15), pp. 589–603, 2015.

[52] M. Brooker, T. Chen, and F. Ping, "Millions of tiny databases," in 17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20), pp. 463–478, 2020.

[53] P. García-López, A. Slominski, S. Shillaker, M. Behrendt, and B. Metzler, "Serverless end game: Disaggregation enabling transparency," arXiv preprint arXiv:2006.01251, 2020.

[54] Github, "Smile (statistical machine intelligence and learning engine)." https://github.com/otrack/smile.

[55] Contributors of PyWren for IBM Cloud, "PyWren on Knative." `https://github.com/pywren/pywren-ibm-cloud/blob/master/docs/knative.md`, 2020.

[56] LSDS Group, "Faasm Kubernetes/ Knative integration." `https://github.com/lsds/faasm/blob/master/docs/kubernetes.md`, 2020.

[57] P. G. López, A. Arjona, J. Sampé, A. Slominski, and L. Villard, "Triggerflow: Trigger-based orchestration of serverless workflows," in Proceedings of the 14th Annual ACM Conference on Distributed Event Based Systems, DEBS '20, 2020.

[58] CloudButton Consortium, "Deliverable D3.2 - Serverless Compute Engine Design and Prototype."

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

[60] CloudButton Consortium, "Deliverable D4.2 - Specification and partial support for degradable objects."

[61] CloudButton Consortium, "Deliverable D5.2 - CloudButton Prototype of Abstractions, Fault-tolerance and Porting Tools."

[62] Amazon, "AWS Lamda." `https://docs.aws.amazon.com/lambda/`.

[63] IBM, "Cloud Functions." `https://cloud.ibm.com/docs/openwhisk`.

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

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

[66] "Knative Platform." `https://cloud.google.com/knative/`, 2019.

[67] KEDA, "Kubernetes-based event-driven autoscaling." `https://keda.sh/`.

[68] "Ceph Object Storage:." https://ceph.io/ceph-storage/object-storage/, 2020.

[69] "Ceph RADOS Gateway:." https://docs.ceph.com/docs/master/radosgw/, 2020.

[70] "Minio Object Storage:." https://min.io/, 2020.

[71] "Knative v0.14.0 :." https://knative.dev/v0.14-docs/install/any-kubernetes-cluster/, 2020.

[72] "Knative Observability Plugin :." https://knative.dev/v0.14-docs/install/any-kubernetes-cluster/#installing-the-observability-plugin, 2020.

[73] "Pywren in Knative :." https://github.com/pywren/pywren-ibm-cloud/blob/master/docs/knative.md#test-if-everything-is-working-properly, 2020.

[74] "Metabolomics Usecase :." https://github.com/metaspace2020/pywren-annotation-pipeline, 2020.

[75] "Jupiter :." https://jupyter.org/, 2020.

[76] "IBM Cloud Functions :." https://cloud.ibm.com/functions/, 2020.

[77] "IBM Cloud Object Storage :." https://www.ibm.com/cloud/object-storage, 2020.

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

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

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

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

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

[83] "OCaml website." `https://ocaml.org`, 2020.

[84] M. García, F. M. Danson, D. Riaño, E. Chuvieco, F. A. Ramirez, and V. Bandugula, "Terrestrial laser scanning to estimate plot-level forest canopy fuel properties," International Journal of Applied Earth Observation and Geoinformation, vol. 13, no. 4, pp. 636 – 645, 2011.