Received: 30 March 2018 | Revised: 12 September 2019 | Accepted: 12 December 2019
DOI: 10.1002/cpe.5661

WILEY

SPECIAL ISSUE PAPER

Cloud architecture for plant phenotyping research

Olivier Debauche[1,2] | **Sidi Ahmed Mahmoudi**[1] | **Nicolas De Cock**[2] | **Saïd Mahmoudi**[1] | **Pierre Manneback**[1] | **Frédéric Lebeau**[2,3]

[1]Computer Science Unit, Faculty of Engineering, University of Mons, Mons, Belgium
[2]Biosystems Dynamics and Exchanges (BioDynE) Department of Biosystem Engineering (BioSE), Gembloux Agro-Bio Tech University of Liège, Gembloux, Belgium
[3]ITAP, University Montpellier, Irstea, Montpellier SupAgro, Montpellier, France

Correspondence
Olivier Debauche, Computer Science Unit, Faculty of Engineering, University of Mons, Place du Parc 20, B-7000 Mons, Belgium.
Email: olivier.debauche@umons.ac.be

Present Address
Olivier Debauche, Computer Science Unit, Faculty of Engineering, University of Mons, Place du Parc 20, B-7000 Mons, Belgium

Summary

Digital phenotyping is an emergent science mainly based on imagery techniques. The tremendous amount of data generated needs important cloud computing for their processing. The coupling of recent advance of distributed databases and cloud computing offers new possibilities of big data management and data sharing for the scientific research. In this paper, we present a solution combining a lambda architecture built around Apache Druid and a hosting platform leaning on Apache Mesos. Lambda architecture has already proved its performance and robustness. However, the capacity of ingesting and requesting of the database is essential and can constitute a bottleneck for the architecture, in particular, for in terms of availability and response time of data. We focused our experimentation on the response time of different databases to choose the most adapted for our phenotyping architecture. Apache Druid has shown its ability to respond to typical queries of phenotyping applications in times generally inferior to the second.

KEYWORDS

cloud architecture, digital phenotyping, lambda architecture, plant phenotyping, research application hosting platform

1 | INTRODUCTION

With the grow of global population, the need for crop production and raw fiber has also increased considerably. Indeed, the Food and Agricultural Organization of the United Nation (FAO) predicts that the global population will reach 8 billion people by 2025 and 9.6 billion people by 2050. This practically means that an increase of 70% in food production must be achieved by 2050 worldwide. The significant increase in global population and the rising demand for high-quality products create the need for the modernization and intensification of agricultural practices. Simultaneously, efficient use of water and other resources is required. Hence, selection of high efficiency cultivar must be achieved by improving plant breeding.

Plant phenotyping is defined as the application of methodologies and protocols to measure a specific trait; from the cellular level to the whole plant or canopy level.[1] Phenotyping is mainly based on imaging techniques (eg, 2D, 3D, or hyperspectral images). Plant productivity gets high from the interaction between its genotype and the environment. Therefore, phenotyping should be correlated with environmental conditions, which highlights the amount of required data. Plant phenotyping was identified as a priority for the European research area. Nowadays, research programs around phenotyping such as transPLANT* (Trans-national Infrastructure for Plant Genomic Science), EPPN2020† (European Plant Phenotyping Network) emerge and network research activities to increase interaction between phenotyping facilities and users are already in place such as the iPlant, a collaborative cyber-infrastructure, which can be extended using its API to meet researcher needs. The quantitative analysis of plant phenotypes, structure, and function of plants become the major bottleneck.

Moreover, the number of images and methods used to store and treat these data are continuously growing. Consequently, the high-throughput of data and the need of specific treatment in real or near real-time require several resources. The increasing amount of particular phenotyping in a given case study needs the development of specific applications. Cloud architectures offer means to store a wide range of huge and heterogeneous data. In addition, it hosts a large quantity of specific models and softwares to process these data.

This paper is an extended version of the paper: Debauche O, Mahmoudi S, Manneback P, Massinon M, Tadrist N, Lebeau F, Mahmoudi SA. Cloud architecture for digital phenotyping and automation. In: Cloud Computing Technologies and Applications (CloudTech), 2017 3rd International Conference of. IEEE 2018. ISBN 978-1-5386-1115-9

*http://transplantdb.eu

†http://plant-phenotyping-network.eu

This paper is organized as follow. In Section 2, we highlight key related works. Section 3 covers background, classified actual solutions, and their limitations. Section 4 describes our architecture and its multi-tenancy exploitation. Section 5 presents evaluation results. Section 6 outlines lessons learned. Section 7 concludes this paper.

2 | RELATED WORKS

Research infrastructures (RIs) are key instruments, which play an important role in the advancement of knowledge and technology. These infrastructures that can be classified in (semi-)controlled conditions, intensive field, lean field, modeling infrastructures require e-infrastructure to host databases and applications.

(Semi-)controlled infrastructure is located in greenhouses or growth chambers. This location allows investigation of the genetic variability of measured plants trait in response of given environmental conditions. This kind of infrastructure often uses non-invasive techniques (often imaging) of specific organs or of whole plants. Tisné et al[2] proposed an automated large-scale phenotyping platform offering high spatial homogeneity.

Smart/intensive field experimental sites for high throughput phenomics allow investigation of plants traits in canopies, in natural environments, and in controlled conditions of CO_2, water or limited nutrient availability. These infrastructures use extensive and high-resolution recording of the environmental conditions and detailed imaging carried by proximal or remote sensing systems. Virlet et al[3] proposed an automated robotic filed phenotyping platform for detailed crop monitoring.

Network of field experiments lean, efficient phenotyping close to practical breeding follow environmental gradients and prediction of performances in current and future climatic scenarios in order to establish the link between their performance and underlying traits at the stand level by testing of large number of genotypes in a wide range of conditions using simple phenotyping approaches. Andrade-Sanchez et al[4] developed and evaluated such a field-base high-throughput phenotyping platform.

Modeling platforms are used to test existing or virtual combinations of alleles in a variety of climatic scenarios and management practices. On one hand, models are developed to address plant structure function relations. On the other hand, it simulates the behavior of genotypes in different climatic conditions. New models are also produced by combination of elements from different models. Marshall-Colon et al[5] describe a generating virtual crops using an integrative and multi-scale modeling platform.

Many plant phenotyping databases and solutions exist but a major problem is the lack of commonly accepted standards, ontologies, or APIs, which need to be defined to allow cross-site phenotyping approaches and most efficient use of resources. Furthermore, novel challenges are foreseen in the area of whole genome integration and novel image extraction algorithms.

The variety and the throughput data generated by current platforms need specific distributed cloud infrastructure to collect, treat, and store all of these data. In literature, phenotyping infrastructures have proposed various solutions.

Wang et al[6] implement a cloud-based open source platform using iPlant (an opensource API) for big data management, sharing, and large-scale study of both genotype and phenotype data. Data are stored on a federated network of Integrated Rule Oriented System[7] (*iRODS*), which is a software middleware that organizes distributed data into sharable collection. The platform supports parallel computation on cloud-based high performance computing clusters. Furthermore, it allows the adding of new analysis tools and the construction of automated workflows from public and private apps by end-users. The platform executes and manages scientific analysis, based on a distributed service-oriented architecture (dSOA) and using JSON communication format. A web-based system allows researchers to execute analysis. This architecture provides big data management with possibilities of sharing and a large-scale computational support for treatments. It also gives the chance to the community to deploy its own models, contains popular tools, and provides automated workflows. However, data must be copied (to cluster) in order to prepare its processing with high performance computing (HPC) cluster.

Ananthakrishnan et al[8] developed Globus, which is software-as-a-service (SaaS) approach for research data management by means of an API in order to provide a flexible and powerful platform-as-a-service specifically for researcher community. Thus, authorized users can manage groups (creation, define properties, invite other users, visibility of the group, etc) by means of the Web interface or the Full REST API in order to share data. For instance, developers can use a wide range of security protocols. The platform provides a ''fire and forget'' data transfer mechanism and the GridFTP[9] protocol, which uses a set of heuristics of configuration based on the number and size of transferred files. Globus Genomics[10] is PaaS based on Galaxy and Globus (Transfer and Nexus) offering a scalable genome analysis pipeline creation and execution on the cloud. The main disadvantage of Globus Genomics is that I/O becomes a bottleneck when multiple concurrent applications start accessing the same file system thus deteriorating the performance.[11]

Caballer et al[12] presented a platform capable of deploying on specific demand and complex configurations crucial to conducting specific research. In addition, the platform is capable of integrating public and local cloud resources to meet the needs of researchers, and also to make computational resources available for other users. Virtual infrastructures are described in resources application description language (RADL) and deployed on virtual machines (VMs) hosted on machines to improve the use of equipment. A web application gives scientists access to available infrastructures and access to virtual infrastructures via SSH or remote desktop tools. This architecture takes advantage of the elasticity of cloud, and the virtualization (isolation and multi-tenancy). The major disadvantage of this architecture uses dynamic deployment from a fresh installation. In other terms, when all the data that have been produced and modified get lost, the infrastructure stops. In our architecture, the data are physically separated of computing and analysis platform.

Zaharia et al[13] used Apache Spark, which extends the programming model of MapReduce to resilient distributed datasets (RDDs): a data-sharing abstraction. Moreover, it proposes a unified API designed to develop easily applications. This approach helps to run multiple functions on the same set of data. Moreover, Apache Spark uses a lineage-based recovery, which is more efficient than the fault-tolerance using data replication, because writing data in RAM and the recovery of multiple RRD can be done on multiple nodes in parallel. Authors argue that RDDs address bottleneck in the cluster such as network and I/O, allow to emulate sharing data in memory, and can run MapReduce-like steps with 100ms latency on large cluster. Nevertheless, indexing is not yet implemented in Spark SQL.

Yan and Huang[14] proposed PVMAS, a PaaS supporting multiple languages to process large-scale image, built on the top of Apache CloudStack to provide IaaS, and a Hadoop-based; which is a high-performance cluster and users management oriented. This architecture is based on three major components: (1) a farm of virtual machines; (2) a HPC cluster that achieves image processing by means of Hadoop Yarn to schedule jobs, Hadoop MapReduce to parallelize jobs, HDFS to store and access to files, OpenCL[‡] and OpenCV[§] library; (3) a shared data storage and archive system. As a matter of fact, in our architecture, virtual machines are managed by Apache Mesos, and Apache Spark processes at high performance images while the storage and archiving of data are obtained by Apache Druid that was developed. We would like to mention that Apache Hadoop cannot handle low latency requirements.

Experimental data standardization must be (1) discoverable, available, easily readable, and in normalized format; (2) tagged with metadata using standard terms and ontologies in order to allow repeated experiments. Nowadays, information standards that are related to the description of phenotyping are mostly geared toward destructive phenotyping (MIAME,[15] MIAPE,[16] MSI,[17] MIxS[18]). Moreover, other specific extensions for plants exist such as MIAME-Plant,[19] CIMR,[20,21] and the MIxS[18] environmental package. Emphasis project has begun to work on minimal standards for plant phenotyping MIAPPE[22] and eventual link with ontologies to formalize them. Then, these ontologies make them readable from machines for phenotyping experiments such as plant and crop ontology. The use of API such as RESTful services allows to retrieve and collect data stored at the partners.

Two kinds of data storage can be distinguished: raw data (ie, images) and extracted data (ie, biomass data). In practice, only extracted data are used while raw data should also be preserved for further future exploitation. In all cases, algorithms and software used to extract data must also be registered to achieve for data reproducibility reasons. Nowadays, there is no centralized phenotyping repository. Moreover, the research in plant phenotyping needs the distributed storage of several petabytes of data, backed by professional data management and back up strategies. In addition, data must be traceable, sustainable, and discoverable by use of grid infrastructures.

Our contribution is a versatile architecture, which manages users finely, groups and access to data and that manages big data volumes within distributed cloud platform efficiently. In addition to exploiting HPC resources (CPU, GPU) for time-series and images treatment, users can use their own or public applications hosted and isolated in containers. The architecture allows also a fine-grained usage of frameworks installed on the platform.

In the past, our architecture has been progressively developed through different use cases such as cattle behavior,[23] Farm animals' behavior,[24] the health of bee hives,[25] connected pivot-center irrigation,[26] landslides monitoring,[27] and digital phenotyping.[28] In all these use cases, the lambda architectures were used to ingest streaming time series data from the Internet of Things. Cattle, farm animals' behavior, and the health of bee hives behavior consume relevant data from inertial measurement unit (IMU) transmitted with LoRaWan protocol or locally stored on the device, discharged offline and then ingested by batch processing of the lambda architecture. Pivot-center irrigation and digital phenotyping were integrated into the lambda architecture while a fast processing–based Apache Spark and Hadoop distributed file system (HDFS) are used in common with the lambda architecture.

In this paper, we propose a new approach that describes how logic synthesis works to match digital phenotyping needs and cloud possibilities. We present a solution that links a cloud infrastructure of processing and storage to an application hosting platform. A lambda cloud architecture based on Druid collects, stores, and treats, in near real-time an important amount of time series related data while, a stream processing using Apache Hadoop and Apache Spark is used to process rapidly and specifically 2D, 3D, and hyperspectral images. The data platform allows to host applications and access to stored data within the lambda architecture. The other advantage of this platform is to exchange, share, and access to different models and data between research teams while ensuring a complete data traceability, privacy, and security.

3 | BACKGROUND LAMBDA ARCHITECTURE

The literature shows that a lambda architecture[29] is formed by three layers: (1) the batch layer that ingests and stores immutable large datasets and provides batch views; (2) the speed layer, which processes stream data, produces views and deploys them on serving layer; (3) the serving layer receives client queries and produces serving up views merging batch and real-time views. This architecture is able to collect and store the wide range of data from phenotyping and environmental parameters such as temperature, relative humidity, cation-exchange capacity (CEC), NO_3, etc. We have chosen a Lambda Architecture because it is able to process data from various phenotyping systems either in real-time from automated high-throughput phenotyping systems or time-delayed batch processing of manually acquired images.

[‡]https://www.khronos.org/opencl/
[§]https://opencv.org/

The main applications of plant phenotyping show that requirements for information processing are very different and depend on the aim of the phenotyping. For instance, measuring phenomena, such as foliar reactions, require a rapid treatment of many images. In other cases, such as pattern of growth, there are many images over a long time that must be archived and prepared for further post-treatment. On one hand, phenotyping requires both rapid processing of large number of images and related data. On the other hand, it requires the massive storage of very large amounts of data and the processing capacities of all this massive information.

Large-scale data storage and multiple treatment application of these data require a cloud architecture platform. In digital phenotyping, wide range of images of different kind must be acquired and stored. These images should be completed with information from other sensors before being processed.

Data analysis in the field of smart agriculture is growing rapidly. However, in parallel with the increasing amount of data to be treated, processing systems fail to process information in short delays. Apache Hadoop ecosystem has proved its efficiency to overcome this problem in a wide range of use cases. This framework is a highly available open-source software framework dedicated to store and provide access to large amounts of data. Apache Hadoop is composed of a distributed file system (HDFS),[30] an application framework (MapReduce), and a resource manager (YARN). However, it does not offer any guaranteed performances on how that data can be accessed quickly. The performances decrease under heavy load. Furthermore, Apache Hadoop is unable to provide the sub-second data ingestion latencies. Finally, it is not optimized to store and make data immediately readable.[31,32]

A solution to reduce disk latency is to keep in memory data to reuse for multiples tasks. Apache Spark processes a large amount of data with low latency and includes fault tolerance by introducing a novel resilient distributed dataset abstraction. However, data sharing application must be written in external storage, such as Cassandra, Hive, Pig, Hbase, Chukwa, S3, and HDFS.[32]

Batch and stream processing frameworks such as Apache Storm, Apache Spark Streaming, Apache Apex, Apache Flink, Google Dataflow, Hazelcast Jet, Apache Nemo, and Apache Samza offer low-latency model to ingest and process stream at near real-time speed (Table 1).

Apache Samza is a distributed stream processing framework that treats stream coming from Apache Kafka, which is a distributed streaming platform. Apache Hadoop YARN is used to provide fault tolerance. However, these stream processing frameworks do not provide the same guarantees as batch processing frameworks in terms of correctness.[32] Moreover, the processing may suffer from duplicated events and other problems of data accuracy. The speed of data availability depends on how data are stored in the database. Open source Relational Data Management Systems and NoSQL key/value stores are unable to provide a low latency data storing. Furthermore, it is also not possible to provide query platform for interactive applications.[33] At the very beginning, raw data must be transformed or cleaned before their use.[32] Hence, the process of data loading and batch processing can take a long time (several hours).

Lambda architectures are designed to handle large amounts of data in conjunction with both batch and stream processing methods in combination with a serving layer.[32,34] The particularity of this cloud architecture is compatible with different cases. Lambda cloud architecture can treat all kinds of data, eg, images, video, temporal data, event data, or classic data. This paradigm allows processing at real-time data from stream and enables using data stored at high speed.

The aggregation of real-time and batch processed data is in the serving layer. Druid presented in the work of Yang et al[31] is a distributed column-oriented fault-tolerant database presenting real-time analytical data store. This platform performs requests at high level with less latencies. Druid is designed to solve problems around ingesting and exploring large quantities of times series data. The unit of storage in Druid is

TABLE 1 Comparison matrix of tools

Features	Apache Apex	Apache Flink	Gearpump	Apache Nemo	Apache Samza	Apache Spark	Google Dataflow	Hazelcast Jet	Apache Storm
License	Apache 2.0	Apache 2.0	Apache 2.0	Apache 2.0	Apache 2.0	Apache 2.0	Proprietary	Apache 2.0	Apache 2.0
Large scale	Yes	Yes	Yes	Yes	extremely wide	Yes	Yes	Yes	Yes
Low latency	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
High throughput	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Stream processing	Yes	Yes	Yes	Yes	High frequency	Yes	Yes	Yes	Yes
Batch processing	(Yes)	(Yes)	No	Yes	Yes	Yes	Yes	Yes	No
Unified stream	Yes	Yes	No	No	Yes	Yes	Yes	No	No
Fault-tolerance	Yes	exactly once	exactly once	Yes	incremental checkpoint	RDD Dstreams	Yes	exactly once	Yes
Dashboard	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Continuous job	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Hot re-deployement	No	No	Yes	No	Yes	No	No	No	No
Auto scaling	No	No	No	No	Yes	No	No	No	No
Upgrades	No	No	No	No	Yes	No	No	Yes	No
Rollback	No	No	No	No	Yes	No	No	No	No
YARN / Hadoop integration	Yes	Yes	No	Yes	Yes	Yes	No	No	No

called "segment." Each segment is composed of 5 to 10 million times-stamped events that covers one period of time. Segments are composed of a timestamp column and dimensions that have a variable size between 400 and 700 Mb. Segments can be compressed by LZ4 as default or LZF algorithm, and can also be stored in a column orientation database. The LZ4 algorithm achieves 400 Mb/s per core. Druid cluster is composed of 4 types of nodes and an indexing service.

Druid uses two external dependencies. The first one is MySQL, PostgreSQL, or SQL Server database in order to store metadata of segments. As default, Druid uses Apache Derby to store metadata but it is not usable for a production environment. The second external dependency is Zookeeper that monitors the four kinds of nodes present in the cluster. These four nodes coordinate, broke, store in real-time or archive data on a distributed storage system. Druid is able to import data from Apache Kafka, Stream data, or files data (TSV, CSV, JSON). Other data formats such as Apache Avro, Protobuf, Apache Orc, Apache Parquet, and Apache Thrift are all supported by the addition of extension developed by the community. Druid can use local storage or external service to deep store old segments such as S3, HDFS, Microsoft Azure, Google Cloud Storage, Apache Cassandra or on Rackspace Cloudfiles, and Firehose. Only HDFS and Amazon S3 are supported officially while other extensions are developed and maintained by the community.

The large amount of applications needs to treat data stored in the cloud on different frameworks. Today, to share a cluster, we have two main solutions. First, we can run one framework on one partition of the cluster. Second, the solution consists of allocating a set of virtual machines to each framework. However, these solutions cannot allow high use and efficient data sharing.

Apache Mesos is a fault-tolerant and highly available sharing layer that provides a framework common interface allowing a fine-grained sharing across diverse cluster computing frameworks. Fault tolerance is ensured by Apache Zookeper.[35] Apache Mesos offers a scalable and resilient core for enabling various frameworks. This is particularly important to share efficiently clusters. A master node manages slave daemons running on each node in the cluster.

Each framework that runs on the top of Apache Mesos uses a job scheduler that is registered to the master node and calls for resources while an executor process is on slave nodes to run tasks of the framework (Figure 1). Slave nodes report to the master nodes available resources (number of CPU and amount of memory) (1). Then, the master node invokes the allocation policy module and determines the amount of resources to be allocated to each framework, and the scheduler selects each nodes of the offered resources in order them to assign to the framework (2). At this step, the framework can reject the offered resources if they do not satisfy its requirements. If the framework accepts the offered resources (3), it sends to the master node a description of the tasks to launch on offered resources by nodes slave. A framework may specify a whitelist of nodes with which it can run and avoid node with which it always have offers reject. The master node sends the task to the slave node, which distributes resources to the framework executor.

The distribution of resources is performed by two modules. The first performs fair sharing between resources, and the second implements strict priorities. Frameworks executor on nodes slave is isolated by leveraging existing OS isolation. Resource offers are scalable and robust through three mechanisms: filers to the master node, the count of resources, and the re-offers of resources. Filters avoid communication by providing filters to master node for frameworks, which always reject certain resources. Apache Mesos counts resources offered to a framework in its allocation of the cluster. When a framework does not respond quickly enough to an offer, Apache Mesos can re-offer the resources to another framework. Fault tolerance uses Zookeeper to run multiple masters in a hot-standby configuration.[36] Apache Mesos provides also three containerization modes. It allows to use runtime environment, operating system control, and additional resources like disk usage limit.

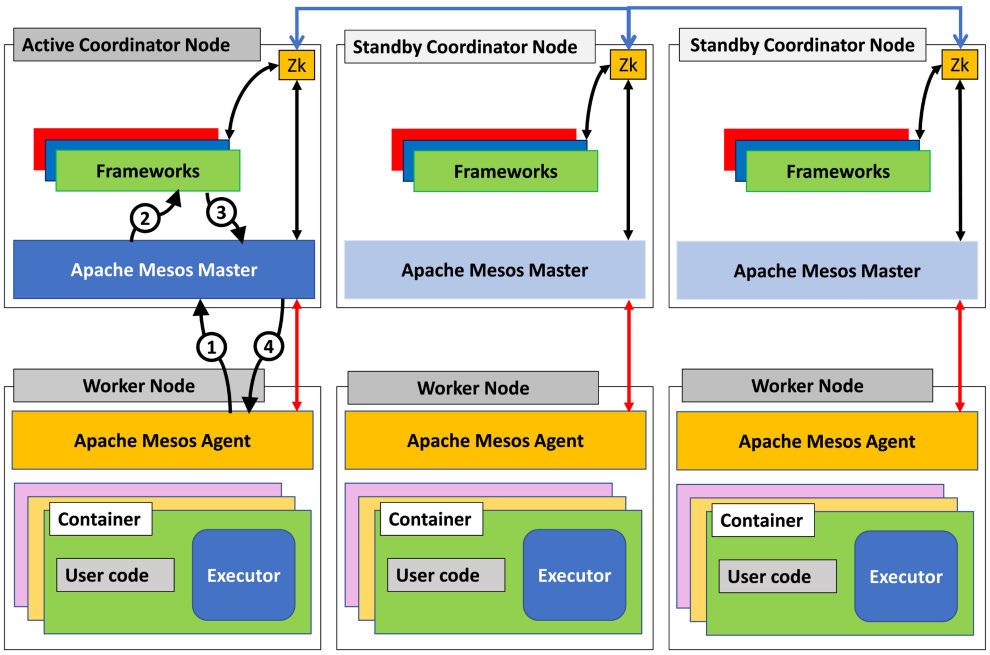

FIGURE 1 Apache Mesos architecture

Apache Mesos provides also Docker containerizing[37] in order to use tools coming with Docker package. The composing of both containerization technologies allow to test different types of resource isolation. Apache Mesos is a good solution to implement multiple frameworks and share fine-grained resource of the cluster. This solution hosts applications for multiple use cases of digital phenotyping.

4 | METHODOLOGY

The important amount of images and data generated and collected from different phenotyping sources requires specific data management and appropriate analytic tools. However, most analytical frameworks are designed for defined tasks and lack of flexibility for the researchers. The research community needs to implement new functionality, for example, the support of new imaging devices from different suppliers, what is usually not supported by closed software. This motivates our choice of software from the Apache ecosystem by the fact that this foundation is robust as softwares are developed and supported by an important community. Moreover, Apache 2.0 License allows marketing and closing the code, what eases research transfer.[38]

4.1 | Cloud architecture

Our framework aims to address the important diversity of needs faced by researchers in digital phenotyping. In addition to that, we exhibit the potential of the cloud to meet these needs. Thus, we contribute by proposing a cloud lambda architecture allowing to store, analyze, and host applications for plant phenotyping. This architecture provides strategic direction and guidance solution in order to process images from plant phenotyping and complementary sensor data.

We propose a cloud solution based on a lambda architecture to collect, store, and treat data from environmental and imaging sensors. The architecture is designed to host various applications and allows to use them with other experimental and commercial phenotyping platforms than those for which they were originally conceived. Various kinds of temporal and event data must be stored. Our cluster is built with Apache Kafka, Apache Samza, HDFS, Apache Druid, PostgreSQL, Apache Zookeeper, and Redis, illustrated in Figure 2.

Stream data are provided by a Firehose, which can be Apache Kafka, RabbitMQ, RocketMQ, or Rackspace. We have chosen Apache Kafka because it is supported. However, the third others are developed by the community. Then, data are ingested by the indexing service and "tranquility" client for Druid compatible with Apache Kafka, Apache Storm, Apache Samza, Apache Spark, Apache Spark Stream, and Apache

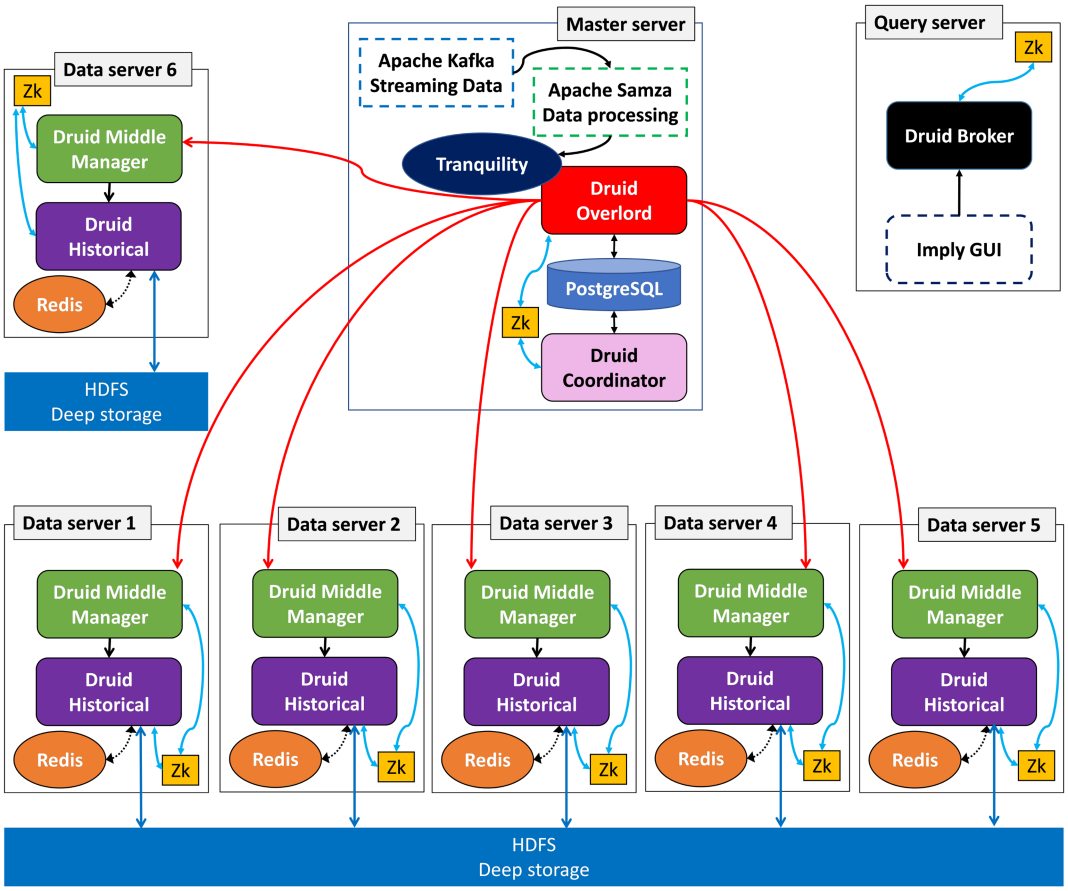

FIGURE 2 Our lambda architecture configuration

Flink. The "tranquility" client is directly integrated in the developed application. The indexing service consists of three main components: a hierarchical component called overlord that manages the distribution of tasks to middle managers that handle peons, which are components that can perform a single task at a time in a Java Virtual Machine (JVM). The repartition of Druid nodes on several servers is more complicated to implement, offers advanced configurability capabilities for scalability and high availability, and allows to finely modulate the architecture during advanced configurations. In this configuration, Druid cluster is able to consume 150,000 events by second.[34] Yang et al[31] have shown that performances can go up to 800,000 events/second/core for simple events. Druid is an open source data store for large datasets combining a column-oriented store layout, a distributed, shared-nothing architecture, and a fast indexing service allowing fast data ingestion and analytics events.[39] Druid is composed of five types of nodes: overlord, middle managers, historical, brokers, and coordinator nodes. Middle manager nodes provide functionality to ingest, query, index event streams for small time range. Indexes are maintained in-memory to be directly queryable. A background task merges indexes together and build immutable blocks from data ingested by middle manager nodes. Segments are uploaded to a HDFS[30] permanent backup storage. The HDFS is a distributed file system for storing distributed and replicated data in a cluster of server.[34] Historical nodes contain functionalities to load and serve the immutable blocks of data created by middle manager nodes. Broker nodes route incoming queries to historical or middle manager nodes and return a final consolidated result to the applicant. Historical nodes contain a caching system with a LRU[40] invalidation strategy and using Redis[41] to store key/value. Finally, coordinator nodes are in charge of data management and distribution on historical nodes: loading, dropping replication, and moving of data. A PostgreSQL database is connected to coordinator nodes that store operational parameters and configurations. This database contains also the list of all segments that can be served by historical nodes and rules to create, destroy, and replicate blocks of data in the cluster. The database can be updated by any service that creates persistent block of data. Batch data process event from static files in JSON, TSV, or CSV formats one at a time and produces segments directly uploaded in the deep storage. Batch data processing may take several hours by opposition of real-time where data are treated in sub second time.[31]

Client uses Druid API to send queries to Druid Broker, which consults Zookeeper data to address the request to Druid middle managers and Druid historical nodes, which produce real-time and batch views, respectively. Druid broker merges into serving up views before forwarding them the client, as shown in Figure 3.

4.2 | Sharing and hosting platform

We use a share and hosting platform to treat and explore data from the lambda architecture. The application hosting using Apache Mesos keeps the structure easily adaptable to various framework while proposing isolation and fine-grained resources of the cluster. This platform uses

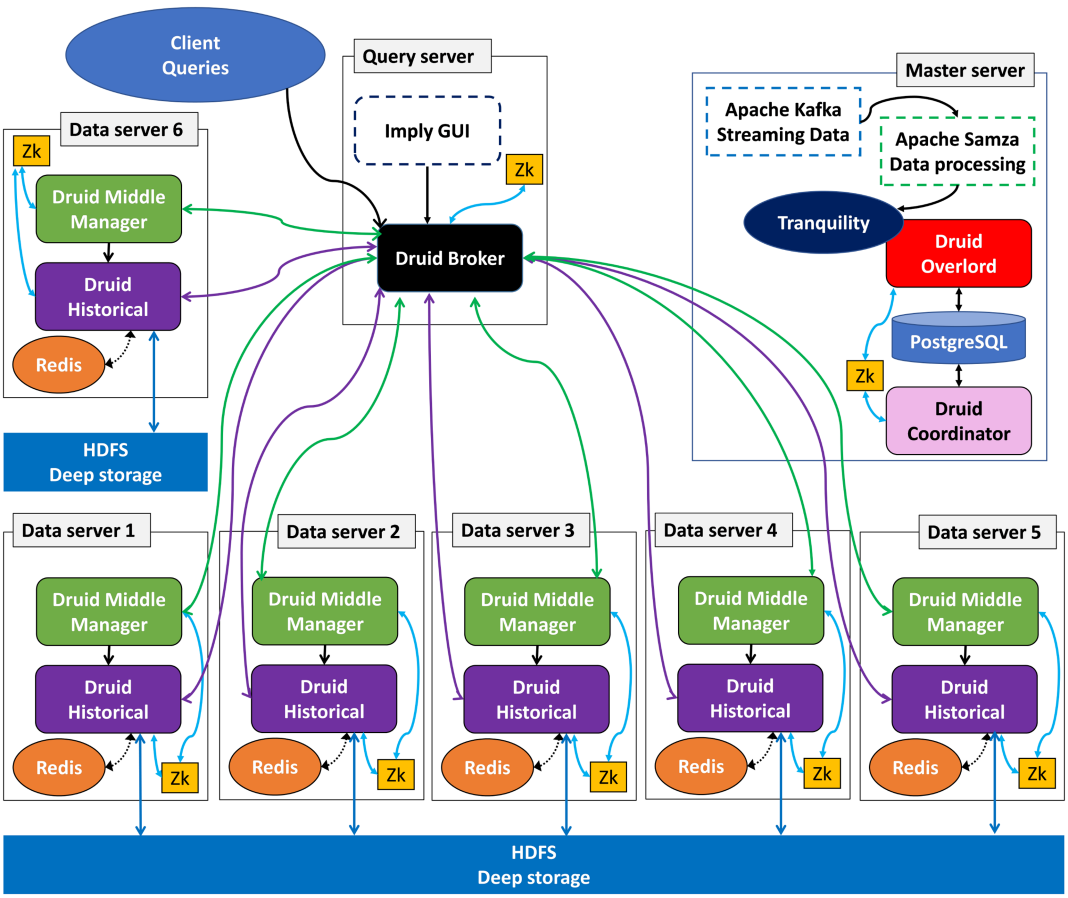

FIGURE 3 Queries on the lambda architecture

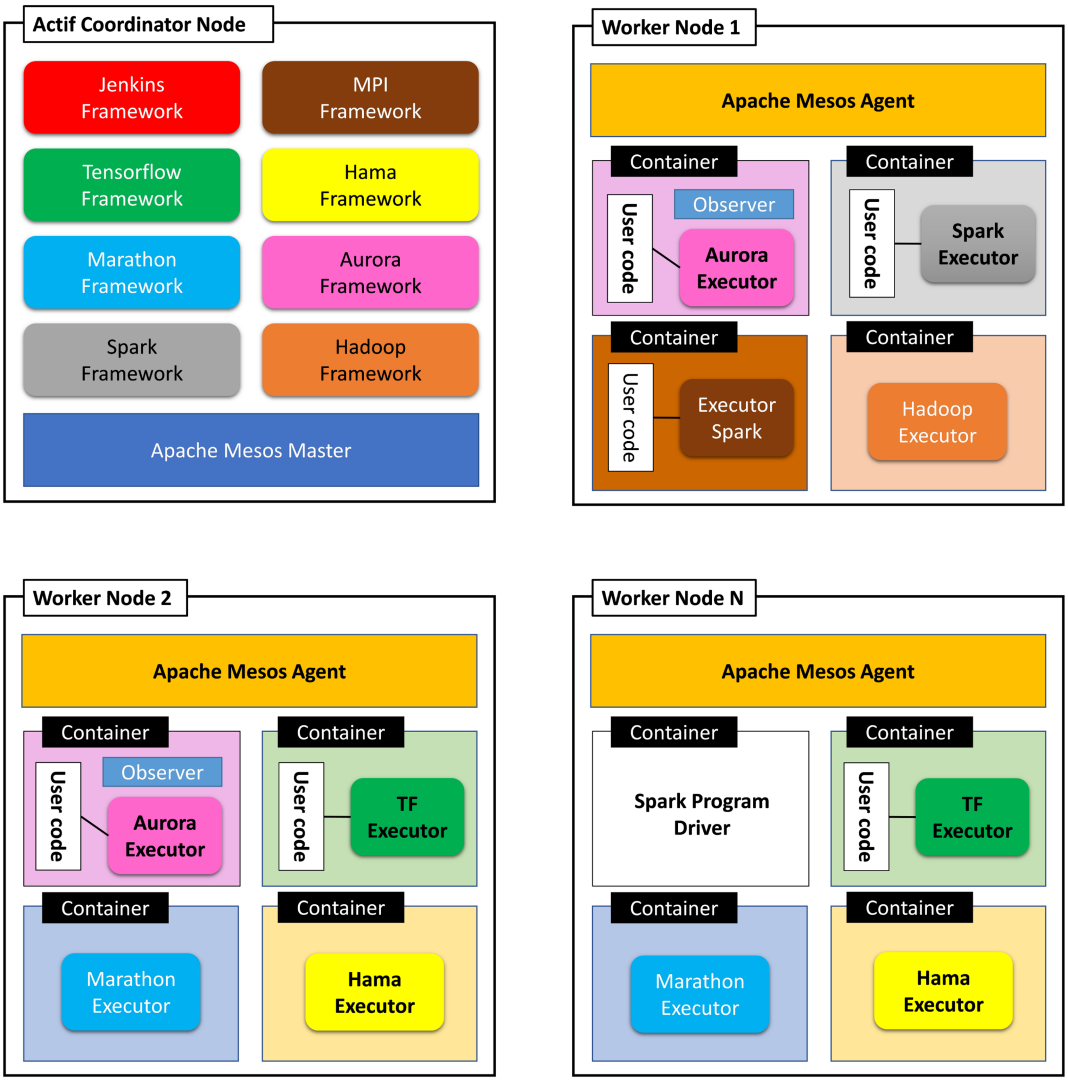

FIGURE 4 Application platform proposed

Apache Mesos isolation, containerization mechanisms, and Docker containers to isolate and host applications. We notice that Apache Mesos isolation is better than Docker. That is why we have mixed these both containerizing methods for compatibility reasons. The application sharing platform uses a quorum of 3 nodes Zookeeper: one master node and two master standby nodes. These two standby nodes ensure fault tolerance in the cluster. Apache Mesos offers several pluggable frameworks. Each one sends tasks to the master node, which transfers them to slave node available to execute the task. When the task is executed, the result is sent to node master, which forwards them to the framework. Docker slave node may host external application, which is not initially developed to work on frameworks plugged on Apache Mesos. Therefore, the application could be hosted with the container technology offered by Docker. Figure 4 illustrates only a part of the Apache Mesos active master (coordinator node) and few Apache worker nodes. The Zookeeper quorum and the two inactive master nodes in standby are not illustrated to simplify the figure; at a high level view including additional elements given in Figure 6.

Six framework plugins are installed on our application sharing platform. Jenkins framework allows continuous integration and dynamic launch of workers. This kind of framework depends upon the workload. Jenkins offers to researchers the possibility to develop algorithms and test them on the cluster. TensorFlow[42] enables to run distributed machine learning tasks with GPU. Tensor flow allows the researchers to experiment machine learning on set images acquired by 3D camera. Marathon is a private as a service (PaaS), which ensures to have an application "on" most of the time. It handles automatically hardware or software failures and guarantees the availability of paying services. The MPI[43] is a message-passing system to function on parallel computers. The MPI allows to accelerate application by starting parallel jobs. It has been plugged for compatibility reasons for some algorithms and models. Apache Hama[¶] is for distributed computing for massive scientific computations and big data analysis based on bulk synchronous parallel (BSP) computing techniques. It provides also vertex and neuron centric programming models. Apache Hama is mainly used in data analysis and model elaboration. Apache Aurora[44] is a service scheduler to run long-running services while taking advantages

[¶]Apache Foundation, "Apache Hama", https://hama.apache.org June2017

of scalability, isolation, and fault-tolerance of Apache Mesos. Apache Aurora is used to develop applications to treat raw data from the lambda architecture and execute cron jobs. Finally, Hadoop framework distributes MapReduce on the cluster, dedicated to cloud computing.

4.3 | Multi-user exploitation

To ensure good performance of our cloud-based applications, we have used a virtual machine (VM) that allows the access to our applications within a private and secured web site. Users can register, connect, and test the integrated applications. This responsive application was developed using the PHP and Bootstrap that allow having a multi-platform website running even on mobile devices (smartphone, tablet, etc). Moreover, the access to our cloud-based application will be secured within https protocol and a 2-factors authentication (password with a code generated and sent by sms).

Otherwise, we will use the Docker framework in order to provide a multi-user exploitation, different users can run the same application simultaneously (Figure 5). Notice that Docker container is an open source software platform that can be used to manage the process of applications development.[37] The main advantage of Docker is its capability to package applications in "containers," allowing them to be portable among any system running the Linux operating system (OS).

In our case, we will generate and configure a Docker image that includes the operating system (Ubuntu) and the different libraries that are required to turn the integrated applications. This image is called "docker-image" that can exploit high performance materials (multi-CPU or/and multi-GPU platforms) in order to provide fast and real-time treatments. In terms of scalability, the docker image allows to exploit multi-CPU and multi-GPU platforms in efficient thanks to the use of the runtime StarPU.[45] The latter provides a unified runtime system for heterogeneous multicore architectures, enabling to develop effective scheduling strategies.

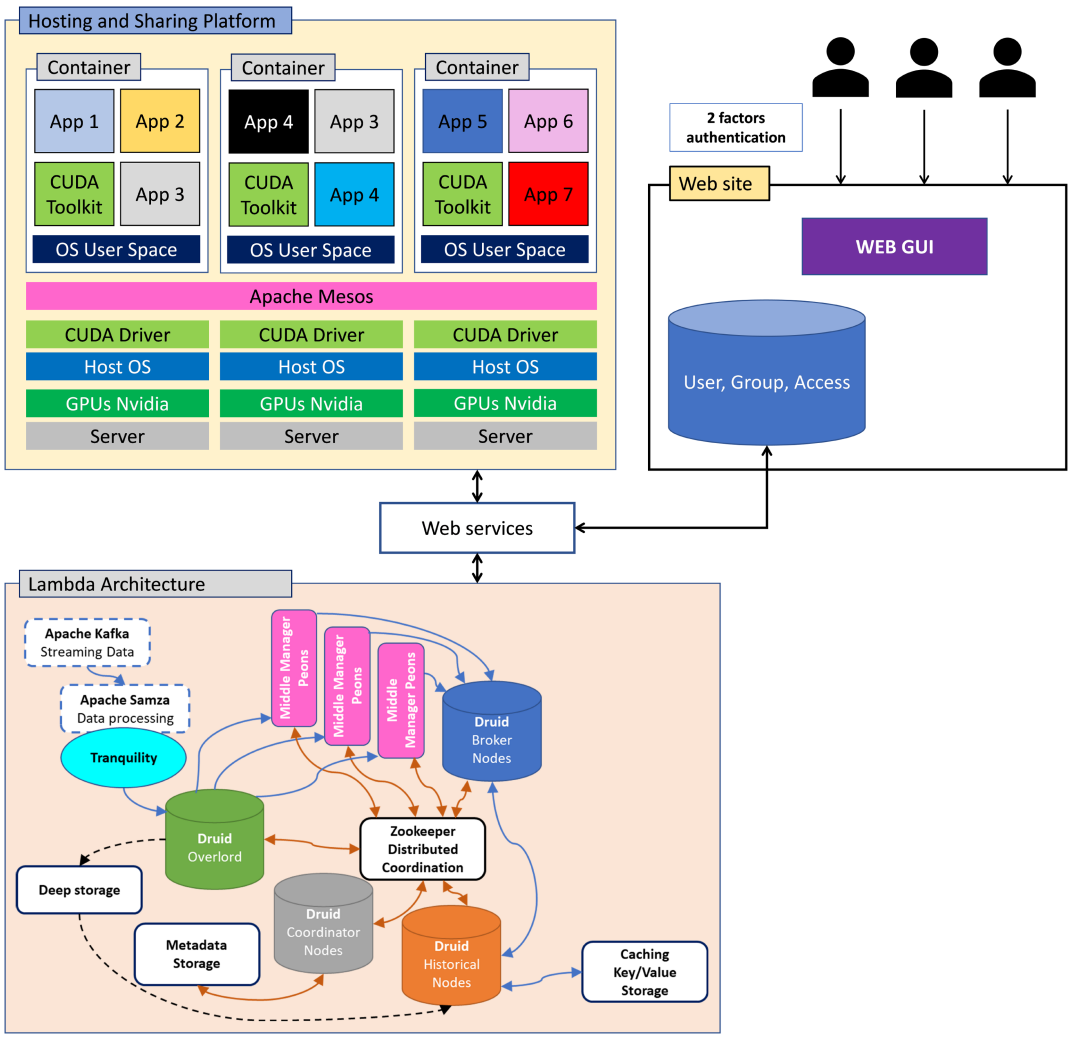

FIGURE 5 Multi-user execution

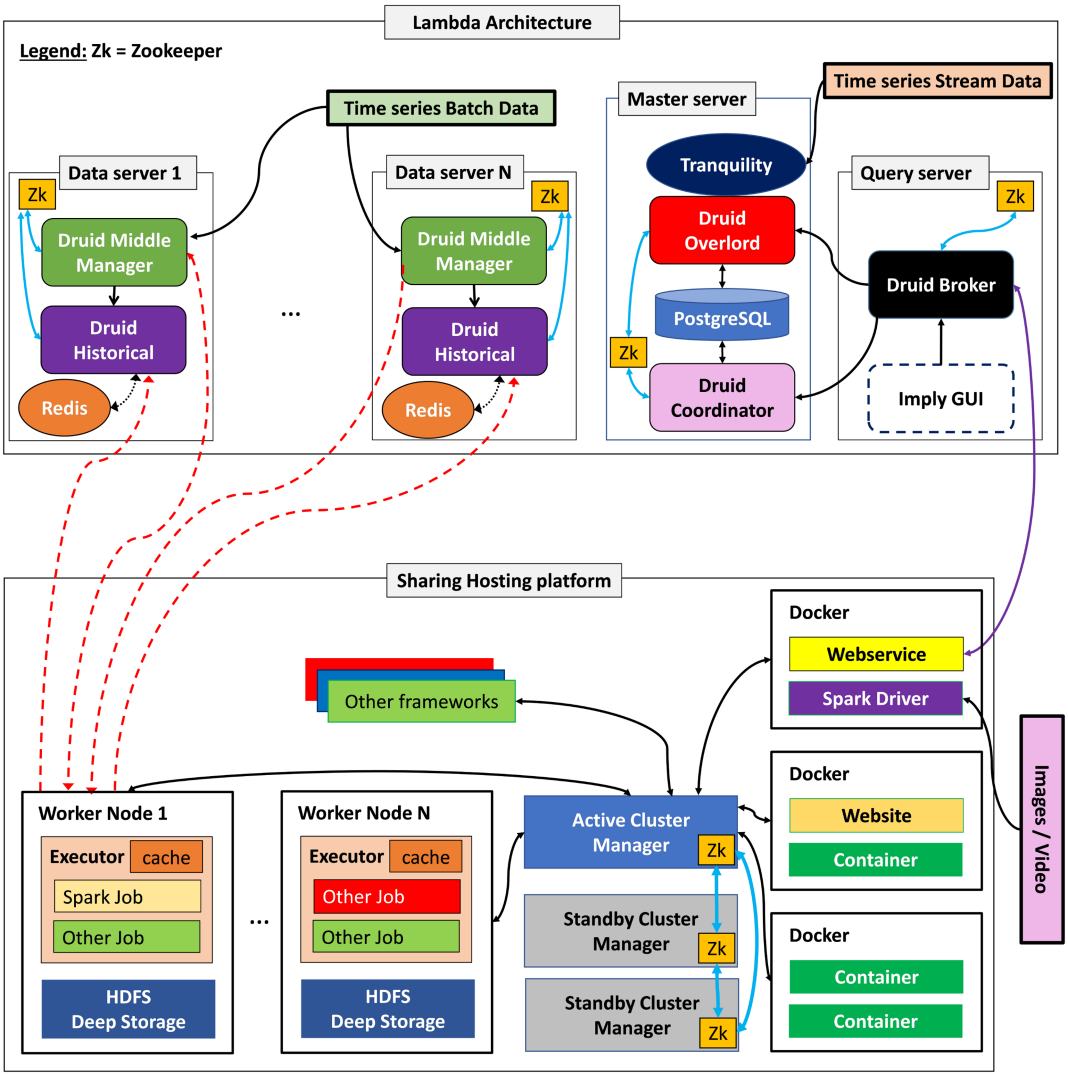

FIGURE 6 Communication between the lambda architecture and the hosting and sharing platform

This docker image allows the use of other libraries for image processing such as OpenCV,[#] CUDA,[‖] and OpenCL.[**]

- **OpenCV:** the OpenCV library presents an image and video processing framework that provides several algorithms for pre-processing, treatment, and analysis of images or videos (in real time).
- **CUDA:** CUDA (Compute Unified Device Architecture) presents a new programming approach, which exploits the unified design of the most current graphics processing units from NVIDIA. CUDA, GPUs consist of many processor cores that can address directly to GPU memories. As a result, CUDA allows to develop parallel implementations that exploit the high number of GPU's computing units. CUDA is so used in the field of image and video processing.[46-48]
- **OpenCL:** OpenCL (Open Computing Language) presents the free standard for parallel programming that can be executed on diverse processors such as CPUs, ATI, or NVIDIA GPUs. OpenCL is also used for several image and video processing applications.[49,50]

4.4 | Integration

A webservice using an API RESTful has been specifically developed to interface both architectures (Figure 6). Each containerized application on the hosting and sharing platform sends request to the Lambda architecture via the API. The webservice can be called from any application installed on the hosting and sharing platform using http requests. The web service controls on one hand the access authorization of each application to the lambda architecture and on the other hand allows the possibilities to query data by means of Druid API. Results are anonymized and all exchanges between Druid and the sharing and exchange platform are logged to ensure the traceability of the using of data. Moreover,

[#] OpenCV. www.opencv.org
[‖] NVIDIA CUDA. https://developer.nvidia.com/cuda-zone
[**] OpenCL. https://www.khronos.org/opencl/

```
-- count_star_interval
SELECT COUNT(*) FROM LINEITEM WHERE L_SHIPDATE BETWEEN '1992-01-03' AND '1998-11-30';

-- sum_price
SELECT SUM(L_EXTENDEDPRICE) FROM LINEITEM;

-- sum_all
SELECT SUM(L_EXTENDEDPRICE), SUM(L_DISCOUNT), SUM(L_TAX), SUM(L_QUANTITY) FROM LINEITEM;

-- sum_all_year
SELECT YEAR(L_SHIPDATE), SUM(L_EXTENDEDPRICE), SUM(L_DISCOUNT), SUM(L_TAX), SUM(L_QUANTITY)
  FROM LINEITEM GROUP BY YEAR(L_SHIPDATE);

-- sum_all_filter
SELECT SUM(L_EXTENDEDPRICE), SUM(L_DISCOUNT), SUM(L_TAX), SUM(L_QUANTITY)
  FROM LINEITEM WHERE L_SHIPMODE LIKE '%AIR%';

-- top_100_parts
SELECT L_PARTKEY, SUM(L_QUANTITY) FROM LINEITEM
  GROUP BY L_PARTKEY ORDER BY SUM(L_QUANTITY) DESC LIMIT 100;

-- top_100_parts_details
SELECT L_PARTKEY, SUM(L_QUANTITY), SUM(L_EXTENDEDPRICE), MIN(L_DISCOUNT), MAX(L_DISCOUNT)
  FROM LINEITEM GROUP BY L_PARTKEY ORDER BY SUM(L_QUANTITY) DESC LIMIT 100;

-- top_100_parts_filter
SELECT L_PARTKEY, SUM(L_QUANTITY), SUM(L_EXTENDEDPRICE), MIN(L_DISCOUNT), MAX(L_DISCOUNT)
  FROM LINEITEM WHERE L_SHIPDATE BETWEEN '1996-01-15' AND '1998-03-15'
  GROUP BY L_PARTKEY ORDER BY SUM(L_QUANTITY) DESC LIMIT 100;

-- top_100_commitdate
SELECT L_COMMITDATE, SUM(L_QUANTITY) FROM LINEITEM
GROUP BY L_COMMITDATE ORDER BY SUM(L_QUANTITY) DESC LIMIT 100;
```

Listing 1 Request used for experimentation (copied from the work of Léauté[52])

authorization is also verified on Druid before executing any query on the database. The Imply[††] GUI allows administrator to check the state of the stored databases. Apache Spark hosted on Apache Mesos processes images and videos. In this configuration, the Spark Master is replaced by the Apache Mesos Master and jobs created by drivers' program are then sent to Apache Mesos Master node, which transmits the job to an available node that executes them.

5 | EXPERIMENTATION

We have evaluated the response time of request on the infrastructure with the well-known TPC-H benchmark,[51] which assesses the performance of the architecture following two ways. (1) The power test measures the query execution power of the architecture when connected with a single user. Requests are run once at a time and the elapsed time is measured separately for each query. (2) The throughput test measures the capacity of the architecture to process concurrent queries in a multi-user environment. This benchmark has been selected to allow comparison with Léauté's benchmark[52] and is generally used to evaluate performances for a lambda architecture.[39] Our servers architecture shown in Figure 2 is deployed on VPS Contabo SSD XL[‡‡] (1 CPU E5-2630v4 - 10 cores 2.2Ghz, 1600 Gb SSD, RAM 60 Gb guaranteed, Lan 1000 Mbit/s port), which runs on Ubuntu 18.04.1 LTS with kernel 4.15. The 100 Gb database corresponds to a classic size obtained from one phenotyping experimentation. We compared response times to 3 types of classic queries (count, sum, and top) on MySQL 5.7.24 (MyISAM), MySQL 8.0.14 (MyISAM), MySQL 8.0.14 (InnoDB), Druid 0.13-incubating-iap8 with TPC-H databases of 1, 10, and 100 Gb installed on a single node. The three sizes of database

[††] https://imply.io/
[‡‡] https://contabo.com/?show=vps

represent: 6,001,215 rows, 60,003,588 rows, and 600,037,092 rows, respectively. The table *lineitem* used for experimentation is composed of daily events data span several years and a varied set of dimensions and metrics, including both very high cardinality and low cardinality dimensions such as *l_partkey* field with 20,272,236 unique values and *l_commitdate* field with 2466 distinct dates in the 100GB data set.

Each of the queries was done 100 times. The mean and standard deviation times obtained are expressed in seconds in Figure 7. Response times for *Count* and *Sum* requests are illustrated on the left in Figure 7, respectively, for databases of 1, 10, and 100 Gb. However, response times for *topN* requests are isolated on the right of Figure 7 for same databases size for very different scale time reason. MySQL 5.7.24 has been chosen to allow a comparison with previous benchmark done, ie, by Léauté in 2014.[52] In addition, MySQL 8.0.14 has been tested with MyISAM engine to be comparable with MySQL 5.7.24 and with InnoDB, the default engine in this version. The Druid version 0.13-incubating-iap8 was the last released version.

The analysis shows MySQL 5.7.24 is better than MySQL 8.0.14 on small database (1 Gb) on all requests type. MySQL 8.0.14 with InnoDB engine is more efficient on aggregation operations on database size from 1 to 100 Gb than MySQL 8.0.14 MyISAM. MySQL 8.0.14 with MyISAM engine is more efficient than the InnoDB engine on topN queries. Finally, Druid 0.13-incubating is incontestably more efficient than any version of MySQL with ratio up to 180 times faster. Moreover, response time to request evolves linearly with the size of the database. TopN queries with MySQL 8.0.14 take more than 250,000 seconds are not shown in Figure 7.

The scaling of Druid on the database TPC-H of 100Gb has been studied, respectively, on 1 node (10 cores), 3 nodes (30 cores), and 6 nodes (60 cores). A decrease almost proportional to the number of cores is observed on request count and aggregation requests (Figure 8). Another decrease between 2 and 3 is obtained for 3 nodes and between 4 and 5 for 6 nodes on top queries (Figure 9). This lower growth was also observed elsewhere by Léauté in 2014.[52] This most important observed decrease can be explained on one hand by the important amount of merging and on the other hand by the low cardinality of the data. Apache Druid uses column-oriented storage, it works best with queries using fewer columns. However, for queries with a larger number of columns, the benefits of row-oriented storage engines are becoming less important.

6 | RESULTS ANALYSIS AND DISCUSSION

Our database time response comparison used benchmark requests proposed by Léauté in 2014. Léauté benchmark was applied on old version of Druid 0.6.62 and MySQL 5.6.13. Our motivation to redo this benchmark is justified as follows. Druid has been actively developed since its integration into the Apache Foundation and has been significantly improved in the meantime. MySQL also has a steady pace of development since its acquisition by Oracle. Moreover, version 8 has, according to Oracle, been reworked in depth and would propose performances up to

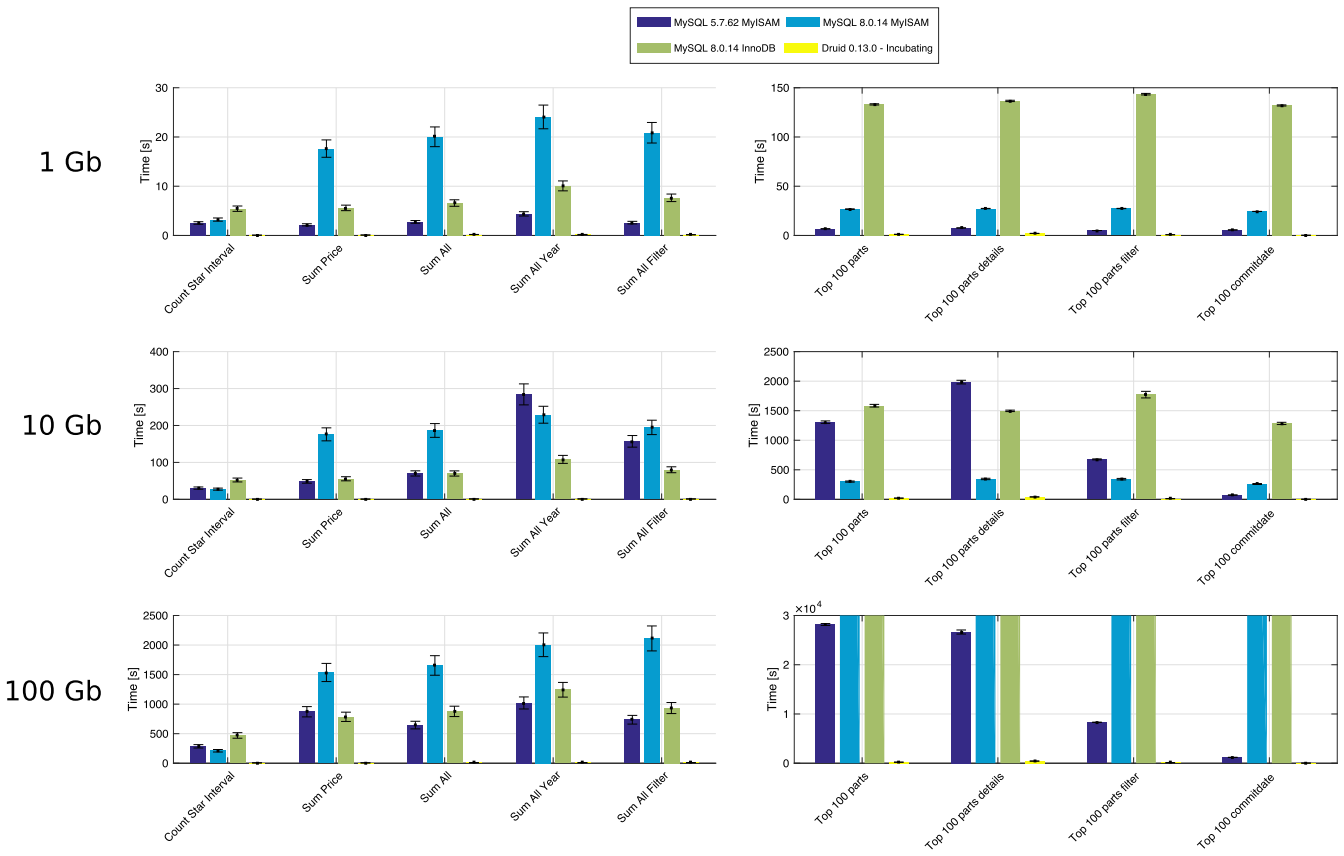

Figure 7 Query response time - DB TPCH- 1 to 100GB

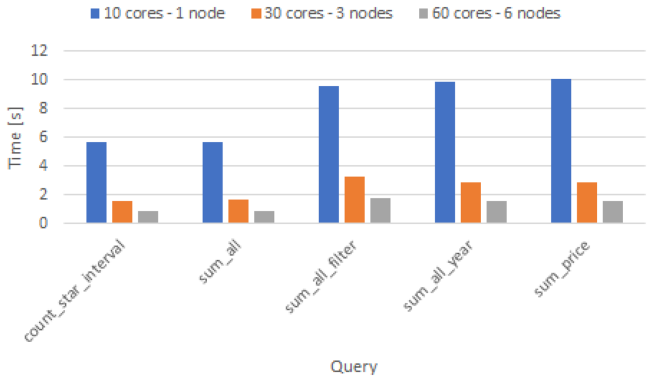

Figure 8 Druid scaling - DB TPC-H 100GB - part 1

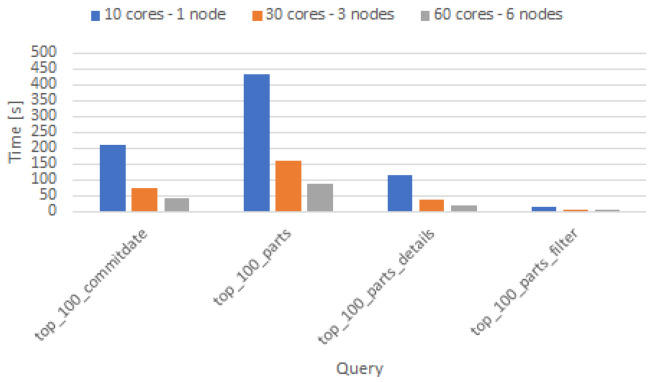

Figure 9 Druid scaling - DB TPC-H 100GB - part 2

TABLE 2 Synthesis of performance between databases

Database Size	MySQL 5.7 MyISAM	MySQL 8 MyISAM	MySQL 8 InnoDB	PostgreSQL 10.6
1 Gb	4-44s	13-212s	35-678s	2-11s
10 Gb	42-265s	8-297s	36-748s	1-17s
100 Gb	51-153s	37-8008s	83-25633s	3-54s

2.5 times better than branch 5.7 and switching from MyISAM to InnoDB as the default database engine. All these facts motivate us to compare Druid and MySQL current versions. We also compared it to PostgreSQL 10.6 because it is a direct competitor of MySQL and also offers very good performances. Versions used in our benchmark are MySQL 5.7.62 with MyISAM, MySQL 8.0.14 with MyISAM and InnoDB Engines, Druid 0.13-incubating, and PostgreSQL 10.6. MySQL 5.7.62 is used to allow the comparison with results obtains by Léauté, MySQL 8.0.14 with MyISAM is used to compare the performance variation between branches 5.7 and 8.0 of MySQL while InnoDB engine is also evaluated because it is the new default engine. Finally, our benchmark shows that Druid and PostgreSQL provide often better performances than MySQL on big databases. Analysis of results shows that Apache Druid response time evolves in proportion to the amount of data in the database while PostgreSQL has response time proportional to the database size only for the count request. PostgreSQL gives better response time than MySQL regardless of version or engine.

The analysis of Table 2 shows that Apache Druid performs almost all the queries anywhere between 4x and 44x faster than vanilla MySQL for 1 Gb database and better than ratio found by Léauté, which was comprised between 2 and 15x. Results obtained for 100 Gb database are comprised between 51 and 153x, which are slightly better than the results obtained by Léauté, which were 45 to 145x faster than MySQL. Comparatively, the results obtained by PostgreSQL are better than those obtained by MySQL regardless of the version and engine used. We impute the bad performances obtained by the version 8 of MySQL compared to branch 5.7, which can be explained by the youth of this branch and its lack of maturity.

7 | CONCLUSIONS

We have proposed a new and more efficient solution compared to traditional approaches, based on Apache Hadoop. Our solution consists of a new lambda architecture based on Druid and an application sharing platform using Apache Mesos.

Druid ingests incoming data and makes them queryable at sub second delay. Our architecture can provide large range of data mixing old data archived in deep storage (HDFS) and recent data, which are just treated but not yet archived. This aspect is particularly important for critical data, which need rapid processing and eventually reactions. The lambda architecture proposed can ingest a large panel of data such as time series,

images, video, etc. This lambda architecture is able to adapt to significant variation of quantity of data to treat at real time. Moreover, Druid's data ingestion latency is dependent directly of the complexity of the data set ingested.[31]

The wide range of applications of digital phenotyping shows that it is not possible to develop all applications on only one framework and need a set of frameworks on the same cluster. The proposed application of sharing data may, on one hand, facilitate the task for scientist to develop methods for specific application and to share them with the community. On the other hand, it allows to test these applications with other data.

The proposed platform is able to receive various kinds of applications to exploit heterogenous data. Our sharing platform uses Apache Mesos, which achieves a fine-grained allocation of resources in the cluster without drawback of multiple virtual machine (VM), siloed or over-provisioned cluster. Apache Mesos allocates resources of available slave nodes in function of the requested tasks.

Moreover, we have also validated and tested the scalability and the adaptability of our architecture on different use cases such as cattle behavior,[23] animal' behavior,[24] connected pivot-center irrigation,[26] digital phenotyping,[28] the health of bee hives,[25] and landslides monitoring.[27]

European legislation that will be translated into national law will require us to ensure the safety and traceability of the use of the data. Therefore, we plan to develop a mean to guarantee the security and confidentiality of the data exchanged between the lambda architecture and the application platform. It will be necessary to ensure that the data transmitted can only be used for the concurrent license duration and for the authorized applications.

ACKNOWLEDGMENTS

We would like to thank our colleagues from Biosystems Dynamics and Exchanges Axis, Biosystem Engineering Department, Gembloux Agro Bio-Tech (ULiège) without whom this work would not have been possible. We would especially like to thank Mr Rudy Schartz for his technical support and for setting up all the electronic systems necessary for carrying out this research. The authors would also like to thank Mr Adriano Guttadauria for his technical support and for setting up all the electronic and informatic systems necessary for carrying out this cloud architecture.

CONFLICT OF INTEREST

The authors declare no potential conflict of interests.

FINANCIAL DISCLOSURE

None reported.

AUTHOR CONTRIBUTIONS

In this paper, we propose a joint architecture able on one hand to collect, store, and treat data from IoT; and on the other hand, a sharing and hosting platform to host software developed by researchers. This platform is also able to use multi-GPU with docker technology. A webservice controls access of users and tiers applications, anonymizes data to ensure the privacy, and ensures the traceability of data use.

ORCID

Olivier Debauche https://orcid.org/0000-0003-4711-2694
Sidi Ahmed Mahmoudi https://orcid.org/0000-0002-1530-9524

REFERENCES

1. Singh A, Ganapathysubramanian B, Singh AK, Sarkar S. Machine learning for high-throughput stress phenotyping in plants. *Trends Plant Sci*. 2016;21(2):110-124.
2. Tisné S, Serrand Y, Bach L, et al. Phenoscope: an automated large-scale phenotyping platform offering high spatial homogeneity. *Plant J*. 2013;74(3):534-544.
3. Virlet N, Sabermanesh K, Sadeghi-Tehran P, Hawkesford MJ. Field scanalyzer: an automated robotic field phenotyping platform for detailed crop monitoring. *Funct Plant Biol*. 2017;44(1):143-153.
4. Andrade-Sanchez P, Gore MA, Heun JT, et al. Development and evaluation of a field-based high-throughput phenotyping platform. *Funct Plant Biol*. 2014;41(1):68-79.
5. Marshall-Colon A, Long SP, Allen DK, et al. Crops in silico: generating virtual crops using an integrative and multi-scale modeling platform. *Front Plant Sci*. 2017;8:786.
6. Wang L, Ware D, Lushbough C, Merchant N, Stein L. A genome-wide association study platform built on iPlant cyber-infrastructure. *Concurrency Computat Pract Exper*. 2015;27(2):420-432.
7. Xu H, Russell T, Coposky J, et al. iRODS primer 2: integrated rule-oriented data system. *Synth Lect Inf Concepts Retr Serv*. 2017;9(3):1-131.
8. Ananthakrishnan R, Chard K, Foster I, Tuecke S. Globus platform-as-a-service for collaborative science applications. *Concurrency Computat Pract Exper*. 2015;27(2):290-305.
9. Allcock W, Bresnahan J, Kettimuthu R, et al. The Globus striped GridFTP framework and server. In: Proceedings of the 2005 ACM/IEEE Conference on Supercomputing; 2005; Seattle, WA.

10. Liu B, Sotomayor B, Madduri R, Chard K, Foster I. Deploying bioinformatics workflows on clouds with galaxy and globus provision. In: Proceedings of the 2012 SC Companion: High Performance Computing, Networking Storage and Analysis; 2012; Salt Lake City, UT.
11. Bhuvaneshwar K, Sulakhe D, Gauba R, et al. A case study for cloud based high throughput analysis of NGS data using the globus genomics system. *Comput Struct Biotechnol J*. 2015;13:64-74.
12. Caballer M, Segrelles D, Moltó G, Blanquer I. A platform to deploy customized scientific virtual infrastructures on the cloud. *Concurrency Computat Pract Exper*. 2015;27(16):4318-4329.
13. Zaharia M, Xin RS, Wendell P, et al. Apache Spark: a unified engine for big data processing. *Commun ACM*. 2016;59(11):56-65.
14. Yan Y, Huang L. Large-scale image processing research cloud. *Cloud Computing*. 2014:88-93.
15. Brazma A, Hingamp P, Quackenbush J, et al. Minimum information about a microarray experiment (MIAME)-toward standards for microarray data. *Nature Genetics*. 2001;29:365-371.
16. Taylor CF, Panton NW, Lilley KS, et al. The minimum information about a proteomics experiment (MIAPE). *Nat Biotechnol*. 2007;25(8):887-893.
17. Fiehn O, Robertson D, Griffin J, et al. The metabolomics standards initiative (MSI). *Metabolomics*. 2007;3(3):175-178.
18. Yilmaz P, Kottmann R, Field D, et al. Minimum information about a marker gene sequence (MIMARKS) and minimum information about any (x) sequence (MIxS) specifications. *Nature Biotechnology*. 2011;29:415-420.
19. Zimmermann P, Schildknecht B, Craigon D, et al. MIAME/plant – adding value to plant microarrray experiments. *Plant Methods*. 2006;2(1). Article No. 1.
20. Fiehn O, Sumner LW, Rhee SY, et al. Minimum reporting standards for plant biology context information in metabolomic studies. *Metabolomics*. 2007;3(3):195-201.
21. Morrisson N, Bearden D, Bundy J, et al. Standard reporting requirements for biological samples in metabolomics experiments: environmental context. *Metabolomics*. 2007;3(3):203-210.
22. Krajewski P, Chen D, Ćwiek H, et al. Towards recommendations for metadata and data handling in plant phenotyping. *J Exp Bot*. 2015;66(18):5417-5427.
23. Debauche O, Mahmoudi S, Andriamandroso ALH, Manneback P, Bindelle J, Lebeau F. Web-based cattle behavior service for researchers based on the smartphone inertial central. *Procedia Comput Sci*. 2017;110:110-116.
24. Debauche O, Mahmoudi S, Andriamandroso ALH, Manneback P, Bindelle J, Lebeau F. Cloud services integration for farm animals' behavior studies based on smartphones as activity sensors. *J Ambient Intell Humaniz Comput*. 2018.
25. Debauche O, El Moulat M, Mahmoudi S, Boukraa S, Manneback P, Lebeau F. Web monitoring of bee health for researchers and beekeepers based on the internet of things. *Procedia Comput Sci*. 2018;130:991-998.
26. Debauche O, El Moulat M, Mahmoudi S, Manneback P, Lebeau F. Irrigation pivot-center connected at low cost for the reduction of crop water requirements. In: Proceedings of the International Conference on Advanced Communication Technologies and Networking (CommNet 2018); 2018; Marrakech, Morocco.
27. El Moulat M, Debauche O, Mahmoudi S, Aït Brahim L, Manneback P, Lebeau F. Monitoring system using Internet of Things for potential landslides. *Procedia Comput Sci*. 2018;134:26-34.
28. Debauche O, Mahmoudi S, Manneback P, et al. Cloud architecture for digital phenotyping and automation. In: Proceedings of the 3rd International Conference on Cloud Computing Technologies and Applications (CloudTech'17); 2018; Rabat, Morocco.
29. Marz N, Warren J. *Big Data: Principles and Best Practices of Scalable Real-Time Data Systems*. New York, NY: Manning Publications; 2015.
30. Shvachko K, Kuang H, Radia S, Chansler R. The Hadoop distributed file system. In: Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST); 2010; Incline Village, NV.
31. Yang F, Tschetter E, Léauté X, Ray N, Merlino G, Ganguli D. Druid: a real-time analytical data store. In: Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data; 2014; Snowbird, UT.
32. Yang F, Merlino G, Ray N, Léauté X, Gupta H, Tschetter E. The RADStack: open source lambda architecture for interactive analytics. In: Proceedings of the 50th Hawaii International Conference on System Sciences; 2017; Waikoloa Village, HI.
33. Tschetter E. Introducing druid: real-time analytics at a billion rows per second. 2011. Accessed June 13, 2017. http://druid.io/blog/2011/04/30/introducing-druid.html
34. Díaz M, Martín B, Rubio B. State-of-the-art, challenges, and open issues in the integration of internet of things and cloud computing. *J Netw Comput Appl*. 2016;67:99-117.
35. Bai G, Ge Y, Hussain P, Baenziger PS, Grae G. A multi-sensor system for high throughput field phenotyping in soybean and wheat breeding. *Comput Electron Agric*. 2016;128:181-192.
36. Hindman B, Konwinski A, Zaharia M, et al. Mesos: a platform for fine-grained resource sharing in the data center. In: Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation; 2011; Boston, MA.
37. Merkel D. Docker: lightweight linux containers for consistent development and deployment. *Linux Journal*. 2014;2014(239).
38. Klukas C, Chen D, Pape J-M. Integrated analysis platform: an open-source information system for high-throughput plant phenotyping. *Plant Physiology*. 2014;165(2):506-518.
39. Cuzzocrea A, Moussa R, Vercelli G. An innovative lambda-architecture-based data warehouse maintenance framework for effective and efficient near-real-time OLAP over big data. In: Chin FYL, Chen CLP, Khan L, Lee K, Zhang L-J, eds. *Big Data – BigData 2018: 7th International Congress, Held as Part of the Services Conference Federation, SCF 2018, Seattle, WA, USA, June 25-30, 2018, Proceedings*. Cham, Switzerland: Springer; 2018:149-165.
40. Lee D, Choi J, Kim J-H, et al. LRFU: a spectrum of policies that subsumes the least recently used and least frequently used policies. *IEEE Trans Comput*. 2001;50(12):1352-1361.
41. Zawodny J. Redis: lightweight key/value store that goes the extra mile. *Linux Magazine*. 2009;79.
42. Abadi M, Barham P, Chen J, et al. TensorFlow: a system for large-scale machine learning. In: Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI); 2016; Savannah, GA.
43. Gropp W, Lusk E, Doss N, Skjellum A. A high-performance, portable implementation of the MPI message passing interface standard. *Parallel Computing*. 1996;22(6):789-828.
44. Foundation Apache. Aurora is a Mesos framework for long-running services and cron jobs. 2017. Accessed June 7, 2017. http://aurora.apache.org

45. Lecron F, Mahmoudi SA, Benjelloun M, Mahmoudi S, Manneback P. Heterogeneous computing for vertebra detection and segmentation in X-ray images. *Int J Biomed Imaging*. 2011;2011.
46. Mahmoudi SA, Belarbi MA, Mahmoudi S, Belalem G. Towards a smart selection of resources in the cloud for low-energy multimedia processing. *Concurrency Computation*. 2018;30(12).
47. Mahmoudi SA, Manneback P. Multi-CPU/multi-GPU based framework for multimedia processing. In: *Computer Science and Its Applications: 5th IFIP TC 5 International Conference, CIIA 2015, Saida, Algeria, May 20-21, 2015, Proceedings*. Cham, Switzerland: Springer; 2015:54-65. *IFIP Advances in Information and Communication Technology*; vol. 456.
48. Mahmoudi SA, Manneback P. Multi-GPU based event detection and localization using high definition videos. In: Proceedings of the 2014 International Conference on Multimedia Computing and Systems (ICMCS); 2014; Marrakech, Morocco.
49. Mahmoudi SA, Ammar M, Joris GL, Abbou A. Real time GPU-based segmentation and tracking of the left ventricle on 2D echocardiography. In: *Bioinformatics and Biomedical Engineering: 4th International Conference, IWBBIO 2016, Granada, Spain, April 20-22, 2016, Proceedings*. Cham, Switzerland: Springer; 2016:602-614. *Lecture Notes in Computer Science*; vol. 9656.
50. da Cunha Possa P, Mahmoudi SA, Harb N, Valderrama C. A new self-adapting architecture for feature detection. In: Proceedings of the 22nd International Conference on Field Programmable Logic and Applications (FPL); 2012; Oslo, Norway.
51. Council Transaction Processing. TPC-H benchmark. 2013. Accessed February 1, 2019. http://tpc.org/tpc
52. Léauté X. Benchmarking Druid. 2014. Accessed January 8, 2019. http://druid.io/blog/2014/03/17/benchmarking-druid.html

How to cite this article: Debauche O, Mahmoudi SA, De Cock N, Mahmoudi S, Manneback P, Lebeau F. Cloud architecture for plant phenotyping research. *Concurrency Computat Pract Exper*. 2020;e5661. https://doi.org/10.1002/cpe.5661