Advances in Engineering Software 159 (2021) 103029

Contents lists available at ScienceDirect

Advances in Engineering Software

journal homepage: www.elsevier.com/locate/advengsoft

Research paper

Cloud-agnostic architectures for machine learning based on Apache Spark

Enikő Nagy, Róbert Lovas, István Pintye, Ákos Hajnal *, Péter Kacsuk

Institute for Computer Science and Control (SZTAKI), Eötvös Loránd Research Network (ELKH), Budapest, Hungary

ARTICLE INFO

Keywords:
Reference architectures
Big data
Artificial intelligence
Machine learning
Cloud computing
Orchestration
Distributed computing
Stream processing
Spark

ABSTRACT

Reference architectures for Big Data, machine learning and stream processing include not only recommended practices and interconnected building blocks but considerations for scalability, availability, manageability, and security as well. However, the automated deployment of multi-VM platforms on various clouds leveraging on such reference architectures may raise several issues. The paper focuses particularly on the widespread Apache Spark Big Data platform as the baseline and the Occopus cloud-agnostic orchestrator tool. The set of new generation reference architectures are configurable by human-readable descriptors according to available resources and cloud-providers, and offers various components such as Jupyter Notebook, RStudio, HDFS, and Kafka. These pre-configured reference architectures can be automatically deployed even by the data scientist on-demand, using a multi-cloud approach for a wide range of cloud systems like Amazon AWS, Microsoft Azure, OpenStack, OpenNebula, CloudSigma, etc. Occopus enables the scaling of cluster-oriented components (such as Spark) of the instantiated reference architectures. The presented solution was successfully used in the Hungarian Comparative Agendas Project (CAP) by the Institute for Political Science to classify newspaper articles.

1. Introduction

Cloud-based Big Data and Machine Learning (ML) applications [1,2] are becoming increasingly popular in the industry, also in academic and education sectors. In many cases, clouds are used to support the computation and storage needs of such applications by building and managing multi-VM virtual infrastructures (clusters) using some IaaS (Infrastructure as a Service) cloud-based system, temporarily or for a longer period of time, respectively. Machine Learning-as-a-service (MLaaS) [3,4] introduces a self-service model for ML, by provisioning and managing infrastructures tailored for ML behind the scenes.

Recently, a popular choice to convey big data analytics or machine learning is to use Apache Spark [5], which is an open source, distributed, cluster computing system. For the best performance, it is also recommended to use Hadoop Distributed File System (HDFS) [6] along with Spark [5], with which Spark can perform data processing in a data locality-aware way, i.e. moves computation to the site of the data, therefore avoids data movement overhead. Manual deployment and configuration of such a cluster in a cloud environment is non-trivial, error-prone and tedious task, requiring considerable expertise in both cloud and Spark-Hadoop technologies. It might take several days with testing, which may even exceed the time required to perform the actual data processing. After discarding the infrastructure when the current computation is done, the same infrastructure might have to be re-built again for a subsequent computation later, or when deciding to choose another cloud provider, respectively.

This paper proposes a solution to manage (create, monitor, scale and discard) such infrastructures rapidly, easily, and efficiently in clouds. In a manner that the infrastructure is guaranteed to be consistent, properly configured, well integrated, controllable, scalable and fault-tolerant. For this purpose, the reference architecture approach has been chosen. An important advantage of the proposed solution is that the main parameters of the Apache Spark architecture (such as the size of the cluster, number of CPU cores and memory configurations per workers, etc.) can be customized, the computing capacity required for processing can be scaled and cloud-independent. To fulfill these goals we used a hybrid-cloud orchestration tool called Occopus [7], which was developed by SZTAKI. Occopus uses descriptors to define the required infrastructure, which contains the definition of the node types (e.g. Spark Master and Worker). In the simplest configuration, the node acting as Spark Master is also the HDFS NameNode, and Spark Worker nodes are also HDFS DataNodes to enable data locality-aware processing. The number of the worker nodes is configurable before deployment and scalable even at runtime. Occopus uses these descriptors (infrastructure descriptor, node

* Corresponding author.
E-mail addresses: eniko.nagy@sztaki.hu (E. Nagy), robert.lovas@sztaki.hu (R. Lovas), istvan.pintye@sztaki.hu (I. Pintye), akos.hajnal@sztaki.hu (Á. Hajnal), peter.kacsuk@sztaki.hu (P. Kacsuk).

https://doi.org/10.1016/j.advengsoft.2021.103029
Received 15 November 2019; Received in revised form 18 May 2021; Accepted 21 May 2021
Available online 5 June 2021

definition descriptor and optionally the cloud-init files for contextualization) to deploy the required infrastructure in the target cloud. Occopus supports several interfaces to various clouds (EC2, Nova, CloudBroker, CloudSigma, etc.), which allows of easily deploying the very same infrastructure in almost any commercial or private cloud providing such an interface (Amazon, Microsoft Azure, OpenStack, OpenNebula, etc.).

The paper also describes the validation and extension of this solution towards more complex use cases.

As a result of this work, the scientists from the Eõtvõs Lornd Research Network (ELKH) and the Artificial Intelligence National Laboratory [8] are now able to create (among others) a scalable Spark-HDFS cluster in the ELKH Cloud, in a user-friendly way, using only a few commands. Researchers can thus focus on their own work in their specific domain of knowledge, without having to know technical details about cloud computing, Spark deployment or networking at all.

The presented work significantly extends our achievements published in the latest PARENG conference proceedings [9]. In this paper we put our previous approach and solution into a much wider context, and started validating the results in cloud environment. Therefore, we added the reference architecture concept and more related works (see section 2). As the second step, we extended the reference architecture with more components, including HDFS (see section 5.1), RStudio (see section 5.2), Python (see section 5.3), and Kafka (see section 5.4). Finally, we performed the first set of measurements for benchmarking the scalability features of the Spark cluster in an instantiated reference architectures on the ELKH Cloud (see section 7).

2. Related work

Machine Learning-as-a-service includes various fundamental steps for machine learning (ML), such as data pre-processing, model training, and model evaluation, with further prediction. Feeding the model with data and using the prediction results can be bridged with on-premise IT infrastructures through APIs. Amazon Machine Learning services, Azure Machine Learning, Google Cloud AI, and IBM Watson are four leading commercial cloud-based MLaaS providers. For the various AI application scenarios, the major cloud providers offer reference architectures (such as Azure Reference Architectures [10]) including recommended practices, along with considerations for scalability, availability, manageability, and security. Similar state-of-the-art reference architectures are available from several HPC vendors if we want to broaden the landscape: Hewlett Packard Enterprise elaborated its reference architecture for AI [11] (with TensorFlow and Spark) that copes not only with open-source software components, but their own propriety cluster-based hardware platform as well. IBM Systems provides similar solutions [12].

Building ML platforms from open-source modules on IaaS clouds using software containers with generic cloud orchestrators (e.g. Terraform, MiCADO/Occopus [13]) is another feasible option that offers several benefits compared to the above described MLaaS approaches, e.g. less vendor lock-in, higher level of (security) control, etc. All these approaches leverage mostly on open source tools and frameworks [1] [2], such as TensorFlow or Apache Spark (see HDInsight from MS Azure or Dataproc from GPC). Concentrating on the manufacturing sector as one of the main targeted areas, a reference architecture has been recently proposed by Fraunhofer IOSB [14]. It was designed for scalable data analytics in smart manufacturing systems, and complies with the higher-level Reference Architecture Model for Industrie 4.0 (RAMI 4.0). The new reference architecture was implemented and validated in the Lab Big Data at the SmartFactoryOWL based on various open-source technologies (Spark, Kafka, and Grafana). As machine learning tools and methods are embedded as analytical components in many Big Data systems, the Big Data related reference architectures were also considered as a baseline in our project [15].

According to our findings, reasoning on big data (with ML), especially on the execution, facilitation and optimisation of such frameworks/applications on clouds can be significantly enhanced. Better support for the creation/assembly of ML applications from pre-engineered proprietary or open-source building blocks can drive to a new generation of reference architectures that can be deployed on an orchestrated platform, leveraging on the available software components from a repositories. Contrary to the currently followed approaches, our new generation of reference architectures will be provisioned and orchestrated on the targeted infrastructure elements in a highly automated way, i.e. our work is to fill the gap between the theoretical frameworks and their actual deployments in a more effective way.

3. Background

3.1. Apache Hadoop: HDFS

The Hadoop Distributed File System (HDFS) [6] is one of the original components of Hadoop. It provides a distributed fault-tolerant file system that can be installed on commodity low-cost hardware. It consists one or two NameNodes and (theoretically) unlimited number of DataNodes. DataNodes store data, while the NameNodes store the metadata. HDFS provides a hierarchical file system using a single namespace. It is a block based storage using 128MB blocks by default, and with a replication factor of 3. It is rack-aware, and blocks are distributed in a fashion that it can tolerate even a complete rack failure. One particularity of HDFS is that files cannot be appended directly, rather they need to be completely rewritten.

3.2. Apache Spark

Apache Spark is an open source, fast and general purpose cluster framework, designed to run high performance data analysis applications. Instead of the Apache Hadoop's original MapReduce programming paradigm [16], it performs internal (in-memory) data processing that results in a more flexible and faster execution. This type of approach may exceed up to ten times the speed of traditional MapReduce based data processing [5].

Apache Spark was written in Scala and it offers easy-to-use APIs available for Scala, Java, Python and R. It was designed specifically for handling large data sets. In addition to the Spark Core API, other libraries are part of the Spark ecosystem, providing additional opportunities for large-scale data analysis and machine learning. These include Spark SQL for structured data processing, MLlib for Machine Learning, GraphX for graph processing, and Spark Streaming for real-time data analysis on large amounts of data [17].

Spark applications execute as a set of processes on nodes (tasks on executors), coordinated by the driver (main program of the application). The driver interacts with the master (or cluster manager) and executors are started to process data. Spark supports different deployment modes. First, the standalone mode runs without a cluster manager (e.g., YARN [18] or Mesos), but actually it runs on a simple built-in cluster manager. In this mode the master and the workers can be started either by hand or by some automation method. Second, it supports running under cluster managers, currently Yarn, Mesos or Kubernetes. Here the driver interacts with the cluster manager that allocates executors on the nodes of the cluster. Historically, there was an option for running on Hadoop 1.x, referred as 'Spark in MapReduce'.

3.3. Occopus cloud orchestrator

Occopus [7,19] is a hybrid cloud orchestration tool developed by SZTAKI, which enables end users to build and manage virtual machines and complex infrastructures in the target cloud. Occopus was designed to be cloud-independent; currently supported cloud interfaces are: EC2, Nova, OCCI, CloudBroker [20], Docker [21], and CloudSigma [22]. It can handle interchangeable plugins, thus is able to simultaneously implement cloud-dependent interactions via these various cloud interfaces (multi-cloud support). The Occopus tool is also able to interact

with different configuration management tools (”multi-config”), applying multiple contextualization methods and health-check services. As a result, the device can be used in many cloud environments using any combination of the plugins. The task of the orchestrator motor is to set the target state of the infrastructure and to maintain it continuously. The essence of life cycle management is that, in order to reach the target state, Occopus calculates the delta between the desired and the current infrastructure state by monitoring the nodes, and then performs the steps required to reach the desired state (e.g. provisioning, terminating or reconfiguring nodes). Thus, the environment becomes fault-tolerant and will be able to recover lost or failed cluster nodes by restarting or building a new node, respectively.

Occopus also supports scaling that is multiple instances of any given node type can be launched or destroyed in the infrastructure within a range specified by the user. Scaling can even happen at runtime; it is however done manually available via the command line and the REST interface. MiCADO/Occopus [13] supports automatic scaling based on user-defined metrics, thresholds, and rules (by the so called policy keeper component).

Occopus was designed as a personalized service. It does not contain built-in authentication/authorization services. The infrastructure built however may include such components, which enables constructing multi-user environments.

3.3.1. *Occopus descriptors*

The virtual infrastructure which should be instantiated by Occopus consists of nodes. The node is an abstract component implemented by a virtual machine in the cloud or a container in the case of Docker. A node may contain as many services as necessary for its functionality. Occopus works on the basis of so-called descriptors, which describe the design of the virtual infrastructure to be built, the individual nodes, the resources to be used, the configuration management details, the contextualization of the nodes, and the way to monitor the services running on the nodes. For a virtual infrastructure specification, there are two types of descriptors to be provided: infrastructure descriptor (*infra descriptor*) and node descriptor (*node definition*). In addition, there are as many different contextualization files as many types of nodes are in the infrastructure. The format of the Occopus descriptors is YAML, which is easy to read or edit, supports structured information and can be easily processed by Python (Occopus was implemented in Python).

4. Spark cluster deployment and scaling by Occopus

Figure 1 shows the architecture of the implementation at a high abstraction level. In order to be able to create an Apache Spark cluster in the target cloud by Occopus the appropriate Apache Spark descriptor files are needed that have been made publicly available on the official website of Occopus [19]. End users need only to customize the node definition files to specify what resources to be used to build up the Spark cluster (cloud endpoint, VM size, image ID, firewall settings, etc.). Based on the personalized descriptors, the Occopus tool can build, maintain, scale, and delete the Spark infrastructure in any of the supported computing clouds. It uses the predefined authentication data and the cloud API for deployment. The cluster can include a Spark Master (see SM in Figure 1) and any number of Spark Worker nodes (see SW in Figure 1). The latter parameter depends on the scaling parameters can be set in advance, only limited by the available quota of the user on the target cloud.

As discussed in chapter 3.3.1, Occopus needs an infrastructure descriptor and a node definition file as an input to be able to build an Apache Spark cluster in the target cloud. The infrastructure descriptor defines the node types, in this case, the Spark Master and Spark Worker nodes (see figure 2). The dependency between the two node types was set, that is, the Worker nodes depend on the Master node. This ensures that Occopus will launch the Master node first, and only after that, having the required Spark daemon up and running, it starts creating

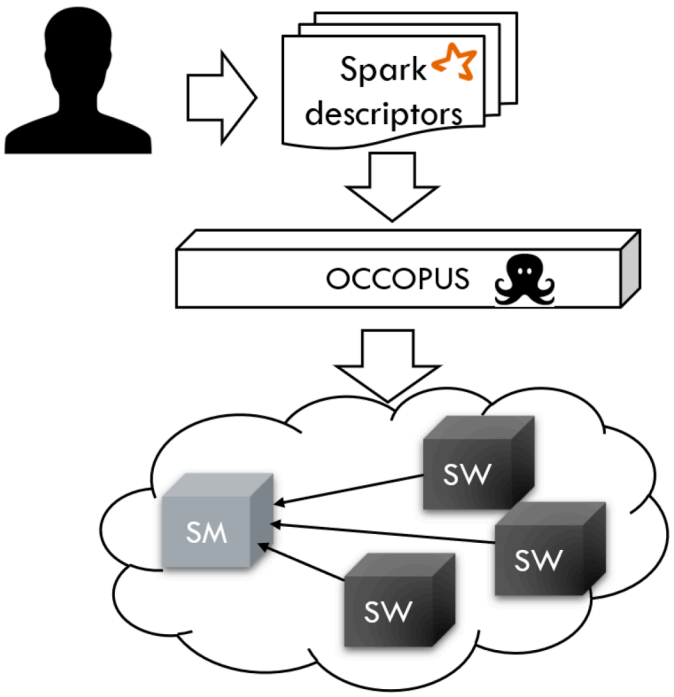

Fig. 1. Spark cluster deployment using Occopus

Worker nodes (in parallel), which will join the Spark cluster. Scaling is allowed for Spark Worker nodes only. In Occopus, for each node a lower and an upper limit can be given for scaling. It ensures the number of copies stays within these limits. At startup, node instances corresponding to the lower limit will be launched, which can be scaled at runtime. If no value is specified, by default, one node is built in the cluster, which cannot be scaled.

Figure 3 shows the structure of the node definition file. For security reasons, authentication and authorization identifiers have been removed from the example. The ‘resource‘ section describes the resource-related parameters such as the endpoint, the image identifier, the flavour name, etc. that are required to initialize a new virtual machine. These parameters depend on the used resource. In this example, the Nova plugin is used, as it is needed to access the ELKH Cloud. The contextualization section defines the method for VM contextualization, in this case it is cloud-init. The cloud-init configuration files were specified as well for the individual nodes. Finally, in the ‘health-check‘ section, different configured ports allow checking the availability of the Spark daemons.

Occopus performs customization of nodes in infrastructure based on the cloud-init files. Therefore, completely ”blank” images can also be used, which should contain only a Linux operating system. All deployments and configurations that are needed to have a properly functioning Spark cluster will be carried out through cloud-init using the Occopus orchestration tool. For each node type, there is a cloud-init file, in the case of a Spark cluster, a total of two, one for the Spark Master and one for the Spark Worker. These files are a bit more complicated than the other descriptors, but they do not need modification by the end-users to build the cluster (unless an advanced user wants to fine-tune the configuration of the Apache Spark cluster).

5. New generation reference architectures for ML on clouds

5.1. *Spark with HDFS*

For our blueprints we chose the standalone deployment mode for Spark (and a minimal HDFS deployment). There are several reasons for this decision. First, each deployed infrastructure instance is intended for

```
nodes:
    - &M
        name: spark-master
        type: spark_master_node
    - &W
        name: spark-worker
        type: spark_worker_node
        scaling:
            min: 2
            max: 10
dependencies:
    -   connection: [ *W, *M ]
```

Fig. 2. Infrastructure descriptor for reference architectures

```
'node_def: spark_master_node ':
    -
        resource:
            type: nova
            endpoint: https://sztaki.cloud.mta.hu...
            image_id: ...
            network_id: ...
            flavor_name: ...
            security_groups:...
        contextualisation:
            type: cloudinit
            context_template: !yaml_import
                url: file://cloud_init_spark_master.yaml
        health_check:
            ports:
                - 50070
'node_def: spark_worker_node ':
    -
        resource:
            type: nova
            endpoint: https://sztaki.cloud.mta.hu...
            project_id: a9c30db63ddf47a98045ef9c726c7436
            image_id: ...
            network_id: ...
            flavor_name: ...
            security_groups:...
        contextualisation:
            type: cloudinit
            context_template: !yaml_import
                url: file://cloud_init_spark_workere.yaml
        health_check:
            ports:
                - 50075
```

Fig. 3. Node definition file for reference architectures

a single scientific use case or application, thus there is no need for a resource scheduler. Second, we wanted to minimize the resource overhead by keeping the number of components to a minimum (omitting a resource manager; number of HDFS NameNodes, Zookeeper instances). In accordance with this and to exploit data locality, each Spark worker node is also a HDFS DataNode. We understand that to truly exploit data locality the nodes should have local storage attached rather than storage provisioned from a network storage. Most cloud providers support this.

The use of HDFS nodes in the cluster is optional, but enable several use cases, for example: (1) running multiple processing tasks on the same dataset: in this case the data should be transferred only once from the remote storage, and then be processed multiple times from local infrastructure, and the results returned; or (2) aggregate or pre-process incoming raw data: high-granularity incoming data is aggregated,

processed before sent to long-term (cold) storage.

5.2. RStudio with Spark

Statistical data processing is a typical Big Data application area, where the R programming language is widely adopted. RStudio [23] is an open-source integrated development environment (IDE) for the R programming language, which includes a console, a syntax highlighting editor with direct code execution possibility, and a set of tools to support drawing, debugging, and workspace management.

To help the work of statisticians willing to use Spark we have created an extended version of the Spark infrastructure placing the *sparklyr* library on Spark workers. Additionally, we have integrated the user-friendly RStudio user interface. As a result, researchers who use the statistical R packages can easily and quickly deploy a complete R-oriented Spark cluster with components: RStudio, R, sparklyr, Spark and HDFS.

The architecture of the solution is shown in figure 4. End users can thus have one Spark Master and multiple Spark Worker nodes in the target cloud, whose number, in case of the ELKH Cloud, depends only on their quota. Each Spark Worker also acts as a HDFS DataNode, from where data can be accessed directly. RStudio provides a web based editor for the end users to write their own applications, and the Spark cluster can be used for distributed background execution. Parallel execution and optimization of the application is no longer the programmer's responsibility, it is delegated to Apache Spark.

As a result of this work, and with the help of Occopus and RStudio with Spark reference architecture descriptors, a convenient, fast and efficient environment can automatically be created on the ELKH Cloud for statistical researchers or other researchers using statistical packages of R. RStudio is launched on the Spark Master node, and accessible using a public IP address. RStudio UI will be available from the browser on port 4040 for the end users. The RStudio UI and the Spark UIs are password protected, which can be customized by the user as well as the port number before deployment. An RStudio stack tutorial is also available on the Occopus [19] and the ELKH Cloud [24] websites.

5.3. Jupyter Notebook with Python and Spark

Similarly to the R-oriented Spark environment, an infrastructure descriptor was also created for a machine learning environment that can be built in the ELKH Cloud. In this scenario, the programming language is Python and the development environment is Jupyter Notebook [25].

Jupyter Notebook is an open source web application that allows of creating and sharing code, equations, and documents containing narrative texts. Jupyter Notebook is widespread in the research community. It provides easy-to-use code demonstration and visualization of results, also helps in collaborative research activities. Jupyter supports more than 40 programming languages, including Python, R, Julia and Scala. It can be integrated with an Apache Spark cluster.

The machine learning environment consists of the following components: Jupyter, Python, Spark and HDFS. The deployment of this machine learning environment is also automatically done by Occopus, and the number of Spark workers can be specified by the user.

The architecture of this solution is very similar to the one shown in figure 4 except that in the WEB UI layer there is Jupyter Notebook on top of the Python interpreter instead of RStudio Web Server on top of the R interpreter.

The Spark cluster executes the distributed applications created by the end user. Each Spark Worker node acts as a HDFS DataNode as well, and the web-based editor is Jupyter Notebook.

The Jupyter Notebook server can be accessed from the browser using a public IP address and port 8888. The Jupyter Notebook UI is password protected. Together with the port number it can also be customized

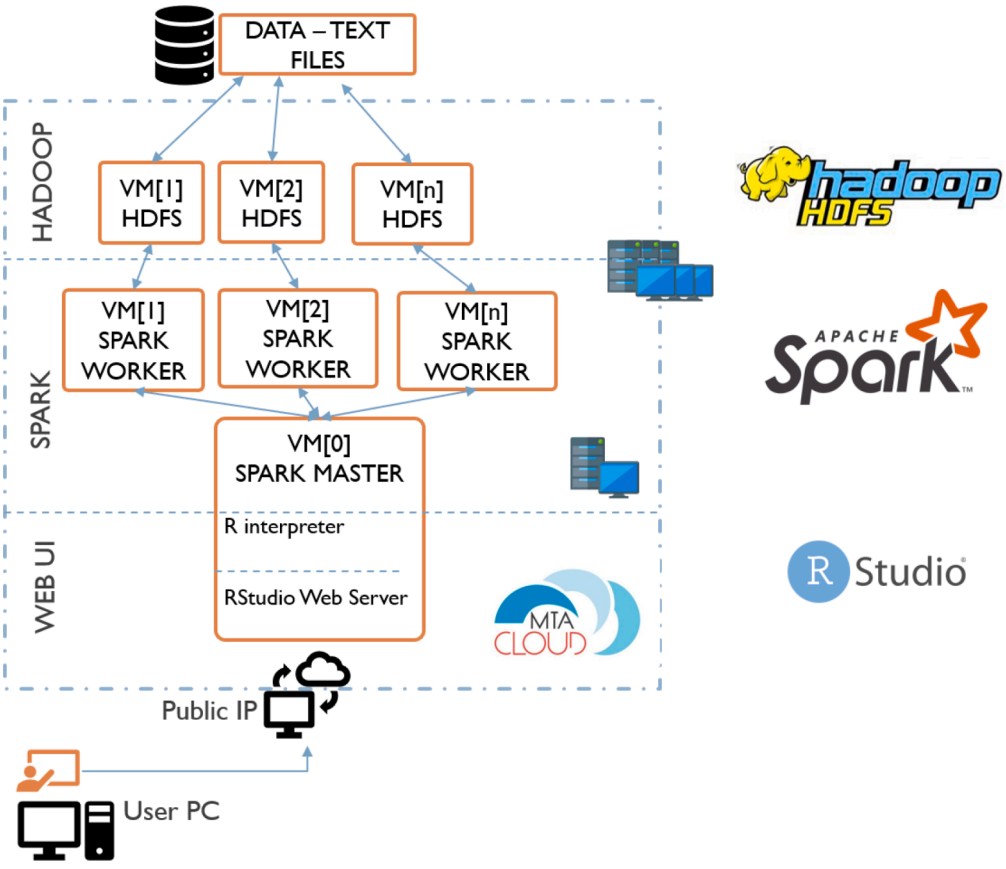

Fig. 4. RStudio Stack for reference architecture

before the deployment process. A complete Python stack tutorial is available on the Occopus [19] (Big Data and AI applications tutorials) and on the ELKH Cloud [24] website.

5.4. Stream processing

Spark is essentially a batch processing framework, however, it supports stream processing as well, in the form of micro-batch-based processing or continuous processing (native stream processing). In case of stream processing, data producers send data continuously, while consumers process these data. Fast producers might overwhelm slow consumers resulting in data loss, therefore an intermediary component should act as a buffer between producers and consumers to solve this problem. Apache Kafka [26] is one of the most wide-spread solutions for this purpose. A typical stream processing pipeline using Apache Kafka is shown in figure 5.

Apache Kafka offers a low-latency, high-throughput, and fault-tolerant solution for buffering streamed data. Kafka can be deployed in a distributed way to spread over several hosts. Logically, data buffers, called ”topics” are further decomposed into ”partitions”, which contain the actual ”messages” (also referred to as events, logs, facts). Topic partitions can be placed onto different hosts, moreover, a configurable replication mechanism can make copies of the partitions on multiple hosts to ensure fault-tolerance. Apache Kafka implies no restriction on the contents, format, or the length of the messages. The total amount of data to be buffered or the duration how long to keep them, respectively, can be configured via retention policies. Although Kafka also provides APIs to process the streamed data entirely within Kafka itself (called KStream and KTables), often, Apache Spark is used to further process data coming from producers. Kafka in these scenarios is only used to buffer and potentially preprocess the streamed data as a first step of the chain, e.g. filter, convert data, aggregate multiple streams, etc. for sake of efficiency.

The reason of using Spark for stream processing lies in its enhanced APIs, namely, DStreams and Structured Streaming. These allow data analysts to focus on data processing tasks and rely on familiar concepts DataSets/DataFrames. Spark can connect to Kafka streams (topics and partitions) ”natively” using built-in libraries, which convert raw data to Spark structures (such as RDDs [27]) automatically. These mechanisms eliminate the technical difficulties that may arise at connecting and de-serializing data (message contents). Another important issue, which might be faced by every stream processing application developer, is ”delivery guarantee” often referred by terms: at-least-once, at-most-once, exactly-once semantics. In some systems, at-least-once or at-most-once guarantees are sufficient, however in mission-critical systems, exactly-once delivery is required. Apache Spark provides exactly-once semantic out-of-the-box in Structured Streaming. Spark’s Structured Streaming API provides further powerful tools such as representing streamed data as data tables and allowing SQL-like queries over them, which could actually never happen for streamed data (there is no point in time when all the data is available in memory at once). The Spark engine still ensures such a look, and performs the hard work by automatically and incrementally maintaining the result set in the background.

Another reason to use Spark lies in the provided machine learning libraries specifically developed for distributed stream processing (e.g. Streaming Linear Regression, Streaming KMeans).

Occopus can also be used to deploy such an infrastructure containing Apache Spark and Apache Kafka nodes, properly configured (with ZooKeeper, cluster hosts, brokers, replication factor, etc.). Data analysts can immediately get a working and fully functional stream processing infrastructure in the cloud of their choice and focus only on data processing development. We omit details of the descriptors for space considerations, just highlight some design concepts.

In this reference architecture ZooKeeper [28] serves as a persistence layer for Kafka in order to maintain topics, partitions, brokers, read and write offsets. Spark Master and ZooKeeper are placed on one host, whereas Kafka brokers are spread over all other cluster nodes in align with Spark Workers. This makes possible to create topics with a number of partitions equals to the number of worker hosts. These partitions are stored in a distributed way by Kafka. Thus, the application can exploit maximum processing parallelism. Note that a single Kafka partition can be read by a single ”consumer” at a time (i.e., Spark job in this context). This design is similar to the case where HDFS DataNodes are aligned with Spark workers for best exploiting data locality-aware processing and maximum parallelism. Also note that in production environments ZooKeeper should also be replicated on multiple servers (3 or more) for fault-tolerance reasons. Occopus deployment also ensures that the necessary libraries (spark-sql-kafka jars) needed to access Kafka from Spark are installed on both Spark Master and Worker hosts.

6. Validation by the Hungarian Comparative Agendas Project

Our methodology was applied by the Institute for Political Science of the Hungarian Academy of Sciences in their CAP [29] and POLTEXT Incubator Projects [30], where the main objective was to classify front-page articles of the two leading Hungarian daily newspapers, Nèpszabadsg and Magyar Nemzet between 1990 and 2014. The coding of public policy major topics on various legal and media corpora served as an important input for testing a wide range of hypotheses and models in political science. Therefore, they investigated how to use large computing resources like ELKH Cloud for this purpose. The main issue was how to exploit the available large number of physical and virtual machines in order to accelerate the process of article classification.

They chose our Spark based reference architecture, because it offered several built in machine learning algorithms and is able to handle data

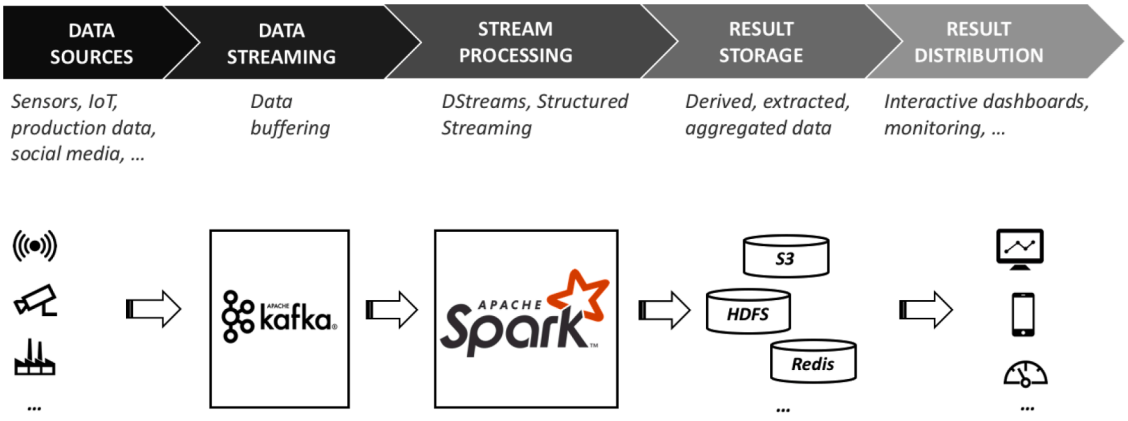

Fig. 5. A stream processing pipeline reference architecture

transformation in a distributed fashion. The combination of Jupyter notebook and Spark extended with R kernel and library imports provides a very productive environment for such text analysis. Spark can be used for several machine learning tasks: regression, classification and clustering, which are supported by built-in Spark ML libraries. In the text, document dataset, the performance indicators were evaluated using five different classifiers, namely, the multinomial logistic regression, the Naive Bayes, the Random Forest, the multi layer perceptron with 2 hidden layers and the convolution neural network, respectively.

The experiments were conducted on a cluster consisting of eleven nodes: one master node and ten worker nodes. Each node had 8 virtual CPU core and 16 GB of RAM. The methodology consisted of five consecutive stages (see figure 6) developed for learning and evaluating classification models in parallel using Spark.

The first stage is to create Resilient Distributed Dataset (RDD), which is the data structure used by Spark to divide the dataset into logical partitions. These partitions may then be processed in parallel on different nodes of the cluster. The second stage is to tokenize the documents to an array of words. Spark machine learning library (MLlib) has a lot of built in functions for text mining such as RegexTokenizer, StopWordRemover, CountVectorizer. CountVectorizer aims at converting a collection of text documents to vectors of token counts. It was used to extract the vocabulary, and generates an array of strings from the document. This method produced a sparse representation for the documents over the vocabulary instead of dense vector representation. In this application, these vectors were the inputs of the above mentioned machine learning algorithms. Before learning, we randomly shuffled and stratified our data/documents. The fourth state is testing, measuring, evaluating and ranking of the classification models. The fifth stage is to use the best classification model to classify the new incoming document. The configuration of Spark is adjusted to control the level of parallelism applied to the data.

This text analysis application was successfully handled by using the Spark-based reference architecture deployed on the ELKH Cloud, and the scientific findings have been already publicly released [31].

7. Experimental results with various ML libraries

For evaluation, we deployed our Python with Spark reference architecture (see section 5.3) using the Occopus orchestration tool (see section 3.3) on the ELKH Cloud. All Spark cluster nodes ran as virtual machines on the OpenStack compute nodes built from Supermicro SuperServer 1028GR-TR: 2 x Intel Xeon E5-2683 v4 CPU @ 2.10GHz (16c/32t), 16 x 32GB DDR4 2400MHz RAM, 2 x 1.0TB SATA 2.5" Seagate Enterprise Capacity (512e), Mellanox 10Gb Ethernet Adapter ConnectX-3 EN MCX312A (2x SFP+). The servers are interconnected via 10Gbit/s networking, and m1.large type VMs were used, with the following available resource for each: 16 GB RAM, 8 vCPU, 160GB disk, using Ubuntu 16.04 OS.

Benchmarks aimed at measuring the speedup of the different machine learning algorithms using up to 8 worker nodes in the Spark cluster. The results are shown in figures 7 and 8. On the X axis in figure 8, the number of worker nodes is indicated (up to 8 Spark Worker nodes with one Spark Master node), and on the Y axis, the speedup is indicated. We also measured the speedup of the following machine learning algorithms using Spark MLlib library: Logistic Regression, Naive Bayes, Multi Layer Perceptron, and Word2Vec. The best speedup was achieved with Naive Bayes and with 8 worker nodes (3,42x speedup). The results of first set of benchmarks seem promising, the fine-tuning of Occopus descriptors and further evaluation on other widely used cloud platforms (Amazon AWS, Microsoft Azure, etc.) is in progress.

8. Conclusions and future work

There is a growing need to process very large scientific and commercial data applying Big Data and Machine Learning techniques. Apache Spark cluster together with HDFS represents an important solution for Big Data and machine learning applications, enabling the data local parallel processing of large data sets. On the other hand, setting up a Spark cluster with HDFS from scratch on any cloud provider is not straightforward, requiring deep knowledge of both the cloud provider and Apache Spark architecture. High-level tools, such as RStudio and/or Jupyter notebooks improve usability and time to value. To make this (and similar) efforts more efficient we have elaborated a new generation

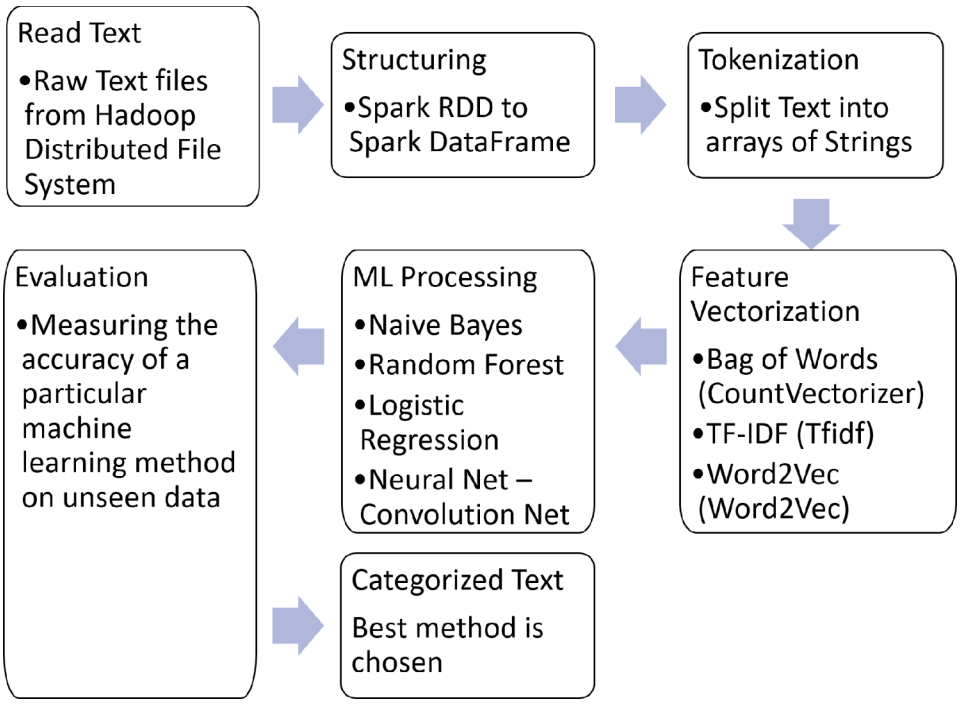

Fig. 6. Stages of learning and evaluating classification models in parallel using Spark

Nodes	Logistic Regression	Naive Bayes	Multi Layer Perceptron	Word2Vec
1	1,00	1,00	1,00	1,00
2	1,39	1,23	1,80	2,12
3	1,24	1,54	1,59	2,14
4	1,42	1,77	1,88	3,10
5	2,79	1,58	1,74	3,22
6	2,74	1,58	1,94	3,23
7	2,81	1,58	1,79	3,32
8	2,99	1,63	2,16	3,42

Fig. 7. Speedup results of orchestrated Spark cluster on ELKH Cloud with ML algorithms

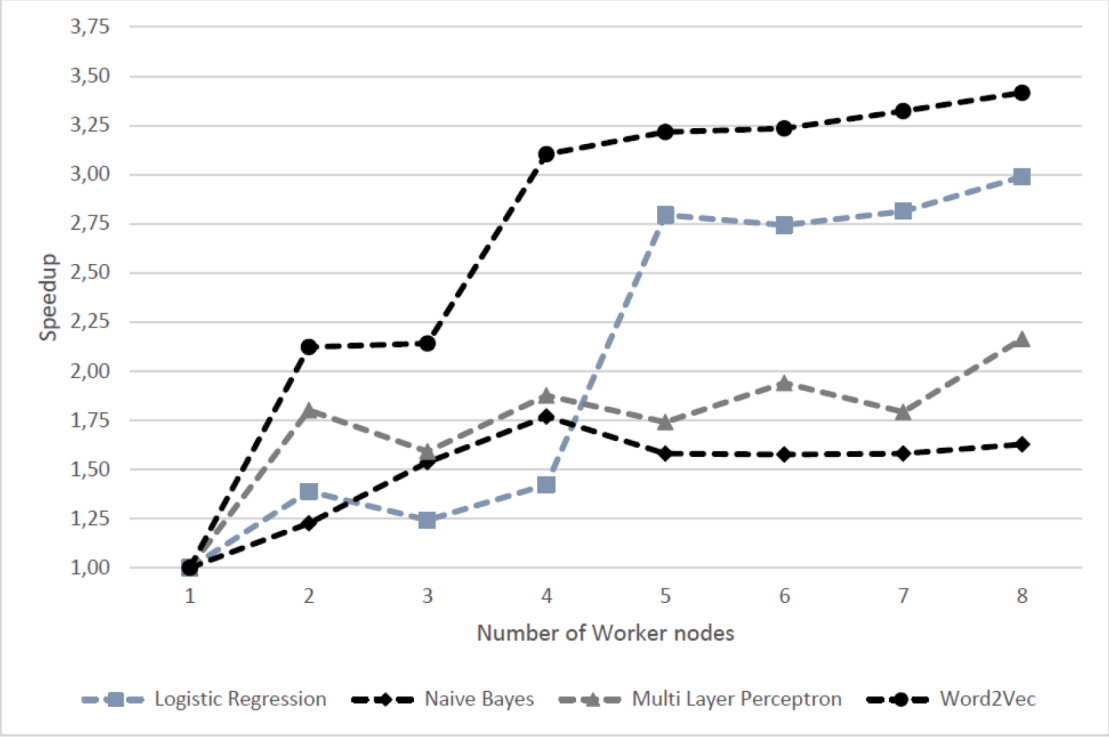

Fig. 8. Speedup diagram of scaled out Spark cluster on ELKH Cloud with ML algorithms

of reference architectures and made public the infrastructure descriptors of these. The publicly available Occopus multi-cloud orchestrator can automatically deploy and orchestrate these architectures (Spark including MLlib for machine learning) with user specifiable number of workers.

The first set of descriptors of the reference architectures are available in the Occopus repository (e.g. [24] or Big Data and AI applications tutorials [19]).

The descriptors can further be configured and extended according to the needs of other scientific communities and/or commercial companies. The results of this work are currently utilized in several EU funded H2020 research projects (COLA [13], CloudiFacturing [32], NEANIAS [33]) as well as in national initiatives (ELKH Cloud and Artificial Intelligence National Laboratory [8]).

As future work we would like to further generalize and extend the architectures: (i) provide alternatives for HDFS and Spark; (ii) extend the architectures to provide failure tolerance (e.g., use two NameNodes, three Zookeeper nodes, Yarn resource manager or Kubernetes, etc.). Additionally, (iii) we would like to focus more on the needs of ML researchers by providing different environments for the different use cases and tools (e.g., GPU support with PyTorch, MXNet, Tensorflow), and generally provide a cloud-agnostic MLaaS platform.

CRediT authorship contribution statement

Enikő Nagy: Conceptualization, Methodology, Software, Writing - original draft. **Róbert Lovas:** Conceptualization, Supervision, Writing - original draft, Writing - review & editing. **István Pintye:** Conceptualization, Software, Validation, Writing - original draft. **Ákos Hajnal:** Conceptualization, Methodology, Software, Writing - original draft, Writing - review & editing. **Péter Kacsuk:** Writing - original draft, Supervision.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgement

The research was supported by the Ministry of Innovation and

Technology NRDI Office within the framework of the Artificial Intelligence National Laboratory Program, and the "NEANIAS: Novel EOSC services for Emerging Atmosphere, Underwater and Space Challenges" project under Grant Agreement No. 863448 (H2020-INFRAEOSC-2019-1).

We gratefully acknowledge the financial support of the Hungarian Scientific Research Fund (OTKA K 132838). The presented work of R. Lovas was also supported by the János Bolyai Research Scholarship of the Hungarian Academy of Sciences. On behalf of the Project Occopus, we thank for the usage of ELKH Cloud (https://science-cloud.hu) that significantly helped us achieve the results published in this paper, and Attila Csaba Marosi for his valuable contribution.

References

[1] Nguyen G, Dlugolinsky S, Bobák M, Tran V, García ÁL, Heredia I, Malík P, Hluchý L. Machine learning and deep learning frameworks and libraries for large-scale data mining: a survey. Artif Intell Rev 2019;52(1):77–124.
[2] Pop D, Iuhasz G, Petcu D. Distributed platforms and cloud services: enabling machine learning for big data. Data Science and Big Data Computing. Springer; 2016. p. 139–59.
[3] Li LE, Chen E, Hermann J, Zhang P, Wang L. Scaling machine learning as a service. International Conference on Predictive Applications and APIs. PMLR; 2017. p. 14–29.
[4] Yao Y, Xiao Z, Wang B, Viswanath B, Zheng H, Zhao BY. Complexity vs. performance: empirical analysis of machine learning as a service. Proceedings of the 2017 Internet Measurement Conference. 2017. p. 384–97.
[5] Zaharia M, Chowdhury M, Franklin MJ, Shenker S, Stoica I, et al. Spark: Cluster computing with working sets. HotCloud 2010;10(10-10):95.
[6] Shvachko K, Kuang H, Radia S, Chansler R. The Hadoop Distributed File System. 2010 IEEE 26th symposium on mass storage systems and technologies (MSST). IEEE; 2010. p. 1–10.
[7] Kovács J, Kacsuk P. Occopus: a multi-cloud orchestrator to deploy and manage complex scientific infrastructures. J Grid Comput 2018;16(1):19–37.
[8] Artificial Intelligence National Laboratory. Artificial Intelligence National Laboratory website. https://mi.nemzetilabor.hu; 2021. Accessed: 2021-04-20.
[9] Nagy E, Hajnal Á, Pintye I, Kacsuk P. Automatic, cloud-independent, scalable Spark cluster deployment in cloud. CIVIL-COMP PROC 2019;112:1–9.
[10] Microsoft. Azure reference architectures website. https://docs.microsoft.com/en-us/azure/architecture/reference-architectures; 2021. Accessed: 2021-04-20.
[11] Hewlett Packard Enterprise. HPE reference architecture for AI on HPE elastic platform for analytics (EPA) with TensorFlow and Spark, white paper, HPE, 2018. https://assets.ext.hpe.com/is/content/hpedam/documents/a00060000-0999/a00060456/a00060456enw.pdf; 2021. Accessed: 2021-04-20.
[12] Lui K., Karmiol J.. AI infrastructure reference architecture. https://www.ibm.com/downloads/cas/W1JQBNJV; 2018.
[13] Kiss T, Kacsuk P, Kovács J, Rakoczi B, Hajnal Á, Farkas A, Gesmier G, Terstyanszky G. MiCADO-microservice-based cloud application-level dynamic orchestrator. Future Generat Comput Syst 2019;94:937–46.
[14] Al-Gumaei K, Müller A, Weskamp JN, Santo Longo C, Pethig F, Windmann S. Scalable analytics platform for machine learning in smart production systems. 2019 24th IEEE international conference on emerging technologies and factory automation (ETFA). IEEE; 2019. p. 1155–62.
[15] Pääkkönen P, Pakkala D. Reference architecture and classification of technologies, products and services for big data systems. Big Data Res 2015;2(4):166–86.
[16] Dean J, Ghemawat S. Mapreduce: simplified data processing on large clusters. OSDI'04: Sixth symposium on operating system design and implementation. San Francisco, CA. 2004. p. 137–50.
[17] Salloum S, Dautov R, Chen X, Peng PX, Huang JZ. Big data analytics on Apache Spark. Int J Data Sci Anal 2016;1(3):145–64.
[18] Vavilapalli VK, Murthy AC, Douglas C, Agarwal S, Konar M, Evans R, Graves T, Lowe J, Shah H, Seth S, et al. Apache Hadoop YARN: Yet another resource negotiator. Proceedings of the 4th annual symposium on cloud computing. 2013. p. 1–16.
[19] SZTAKI. Occopus website. http://occopus.lpds.sztaki.hu; 2021. Accessed: 2021-04-20.
[20] Taylor SJ, Kiss T, Anagnostou A, Terstyanszky G, Kacsuk P, Costes J, Fantini N. The cloudsme simulation platform and its applications: A generic multi-cloud platform for developing and executing commercial cloud-based simulations. Future Generat Comput Syst 2018;88:524–39. https://doi.org/10.1016/j.future.2018.06.006. https://www.sciencedirect.com/science/article/pii/S0167739X17329102
[21] Merkel D. Docker: lightweight linux containers for consistent development and deployment. Linux J 2014;2014(239):2.
[22] CloudSigma Holding AG. CloudSigma website. http://www.cloudsigma.com; 2021. Accessed: 2021-04-20.
[23] RStudio Team. RStudio: Integrated Development Environment for R. RStudio, PBC.; Boston, MA; 2020. http://www.rstudio.com/.
[24] ELKH Cloud. ELKH Cloud Services. https://science-cloud.hu/en/reference-architectures-and-services; 2021. Accessed: 2021-04-20.
[25] Kluyver T, Ragan-Kelley B, Pérez F, Granger B, Bussonnier M, Frederic J, Kelley K, Hamrick JB, Grout J, Corlay S, Ivanov P, Avila D, Abdalla S, Willing C, Team JD. Jupyter notebooks - a publishing format for reproducible computational workflows. ELPUB. 2016.
[26] Kreps J, Narkhede N, Rao J, et al. Kafka: A distributed messaging system for log processing. Proceedings of the 6th International Workshop on Networking Meets Databases (NetDB). vol. 11; 2011. p. 1–7.
[27] Zaharia M, Chowdhury M, Das T, Dave A, Ma J, McCauly M, Franklin MJ, Shenker S, Stoica I. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. 9th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 12). 2012. p. 15–28.
[28] Hunt P, Konar M, Junqueira FP, Reed B. Zookeeper: Wait-free coordination for internet-scale systems. 2010 USENIX Annual Technical Conference (USENIX ATC 10); 2010. USENIX Association; 2010. https://www.usenix.org/conference/usenix-atc-10/zookeeper-wait-free-coordination-internet-scale-systems.
[29] Sebök M, Kacsuk Z. The multiclass classification of newspaper articles with machine learning: The hybrid binary snowball approach. Politic Anal 2021;29(2):236–49. https://doi.org/10.1017/pan.2020.27.
[30] Centre for Social Sciences. poltextLAB website. 2021. Accessed: 2021-04-20 https://poltextlab.tk.hu/en.
[31] Pintye I, Kail E, Kacsuk P, Lovas R. Big data and machine learning framework for clouds and its usage for text classification. Concurren Comput.: Pract. Experienc. 2020.
[32] Kiss T. A Cloud/HPC platform and marketplace for manufacturing SMEs. 11th International Workshop on Science Gateways, IWSG 2019. 2019.
[33] Sciacca E, Vitello F, Becciani U, Bordiu C, Bufano F, Calanducci A, Costa A, Raciti M, Riggi S. Towards porting astrophysics visual analytics services in the european open science cloud. Science and Information Conference. Springer; 2020. p. 598–606.