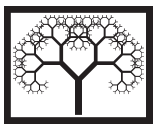

Proceedings of the Sixth International Conference on
Parallel, Distributed, GPU and Cloud Computing for Engineering,
P. Iványi and B.H.V Topping (Editors)
Civil-Comp Press, Stirlingshire, Scotland

Paper 26

Automatic, cloud-independent, scalable Spark cluster deployment in cloud

E. NAGY, Á. HAJNAL, I. PINTYE, P. KACSUK
Institute for Computer Science and Control, Hungarian Academy of Sciences (MTA SZTAKI), Budapest, Hungary

Abstract

This paper shows the results of the second step of a project targeting the creation of all the major Big Data and Machine Learning environments by the use of Occopus, a cloud orchestrator tool. In this second step we have created the Occopus descriptors of a Spark environment that is combined with Hadoop HDFS, scalable and deployable in major cloud systems like Amazon, OpenStack, OpenNebula, CloudSigma, etc. The deployment of such a Spark environment is automatic and hence facilitates the usage of Spark technology in clouds. The Spark environment was successfully used by the Institute for Political Science of the Hungarian Academy of Sciences to the classification of newspaper articles.

Keywords: artificial intelligence, big data, cloud computing, distributed computing, orchestration, Spark

1 Introduction

Cloud-based Big Data and Machine Learning applications 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 cloud-based system (IaaS), temporarily or for a longer period of time, respectively.

Recently, a popular choice to convey big data analytics or machine learning is to use Apache Spark, which is an open source distributed, cluster computing system. For best performance, it is also recommended to use Hadoop File System (HDFS) along with Spark, with which Spark can perform data processing in data locality-aware way, i.e. moves computation to the site of the data, so avoiding data movement overhead. Manual deployment and configuration of such a cluster in a cloud is non-trivial, error-prone and tedious task, requiring considerable expertise in both cloud and Spark-Hadoop technologies. It might take days, 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 and discard) such infrastructures rapidly, easily, and efficiently in clouds, which are guaranteed to be consistent, properly configured, well integrated, controllable, scalable and fault-tolerant. 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, which was developed by MTA SZTAKI. Occopus uses so called descriptors to define the required infrastructure, which contains the definition of the node types (Spark Master and Worker). The Spark Master will also be the HDFS Name Node, and the Worker nodes will be Data Nodes for the HDFS system at the same time 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 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, OpenStack Nova, Docker, CloudBroker, CloudSigma, which allows of easily deploying the very same infrastructure in almost any commercial or private cloud providing such an interface (Amazon, OpenStack, OpenNebula, etc.).

As a result of this work, the MTA Cloud research community is now able to create a scalable Spark-HDFS cluster in cloud of Hungarian Academy of Sciences (MTA 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.

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 Map Reduce programming paradigm [1], it performs internal computational data processing that results in a more flexible and faster run. The module uses a parallel data processing framework that stores data in memory and, if necessary, on disk. This type of approach exceed up to ten times the speed of Hadoop Map Reduce data processing [2].

The Apache Spark project was launched in 2009 and was originally developed by the UC Berkeley University's AMPLab lab [3] as an alternative to the Hadoop's Map Reduce framework. The Spark code base was later donated to the Apache Software Foundation in 2010 and became open source [4]. Originally created by its creators, Databricks [5] is currently working on expanding the open source project, simplifying the running of Big Data and machine learning applications, developing a web platform for Spark that automates cluster management and provides IPython-style notebooks, organizing large-scale open online Spark courses as well as being the leader of Spark Summit, which is the largest Spark conference.

Apache Spark was written in Scala, and one of its favourite feature is its highly-developed easy-to-use APIs, such as Scala, Java, Python and R, designed specifically for handling large data sets. In addition to the Spark Core API, other libraries are part of the Spark ecosystem,

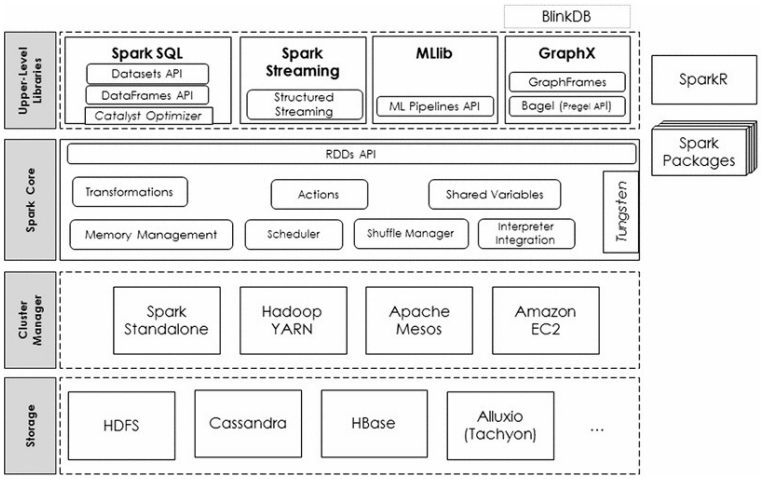

Figure 1: High-level architecture of the Apache Spark stack

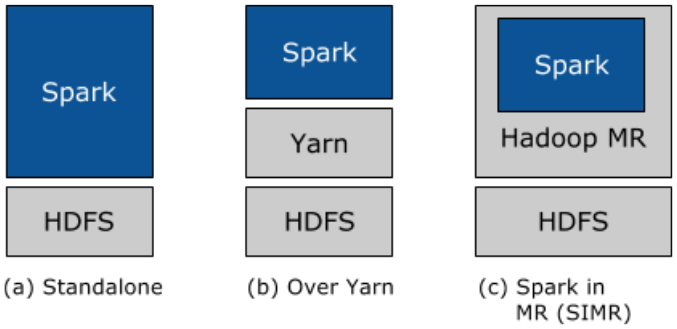

Figure 2: Integrating Spark with Hadoop

providing additional opportunities for large 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. Figure 1 shows a high-level architecture of Apache Spark stack. [6]

It is important to emphasize that Apache Spark is not a substitute for Apache Hadoop, but a kind of extension of it. Spark has been designed to be able to read and write data from Hadoop's own distributed file system (HDFS), and other storage systems such as HBase or Amazon S3. Thus, Hadoop users can enrich their processing capabilities by combining Spark with Hadoop MapReduce, HBase, and other Big Data frameworks. Spark can be used on the same cluster that embodies the Hadoop cluster, along with the MapReduce framework, either alone or as a processing framework. Spark applications can also be run on the Hadoop cluster manager, YARN. MapReduce and Spark can be used in conjunction with MapReduce for batch processing and Spark for real-time data processing. There are three basic ways to install Spark in the Hadoop cluster: standalone, YARN, and SIMR (Spark in MapReduce) (see FigureFigure 2). [7]

3 Occopus

Occopus [12] is a hybrid cloud orchestration tool developed by MTA 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, the currently supported cloud interfaces are: EC2, Nova, OCCI, CloudBroker [9], Docker [10] and CloudSigma [11]. It can handle interchangeable plugins, thus being able to simultaneously implement cloud-dependent interactions via these various cloud interfaces (multi-cloud support). The Occopus tool is 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 difference between the desired and the current infrastructure state (delta) by monitoring the nodes, and then performs the steps required to reach the desired state (e.g. starting / breaking / restarting a new node). Thus, the device becomes fault-tolerant and will be able to recover lost or failed cluster nodes by restarting or building a new node. Occopus supports manual scaling, and could be used via command line interface (REST as a service or library). [8]

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 and 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, an infrastructure descriptor (infra descriptor), a node descriptor file (node definition), and as many different contextualization files as many types of nodes are in the infrastructure. The language format of the Occopus descriptors is YAML, which is easy to read and edit, supports structured information, and can be easily processed by Python (Occopus was implemented in Python 2.7 program languages).

4 Spark deployment and scaling by Occopus

Figure 3 shows the architecture of the implementation at a higher abstraction level. In order for Occopus to be able to create an Apache Spark cluster in the target cloud, the appropriate Apache Spark descriptor files are needed, that have been made publicly available on the official website of Occopus [12]. End users must customize the node definition files to specify what resources to be used while building 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 computing clouds supported by the Occopus tool. It uses the predefined authentication data and the cloud API for deployment. The cluster can include a Spark Master and a number of Spark Worker nodes. The latter

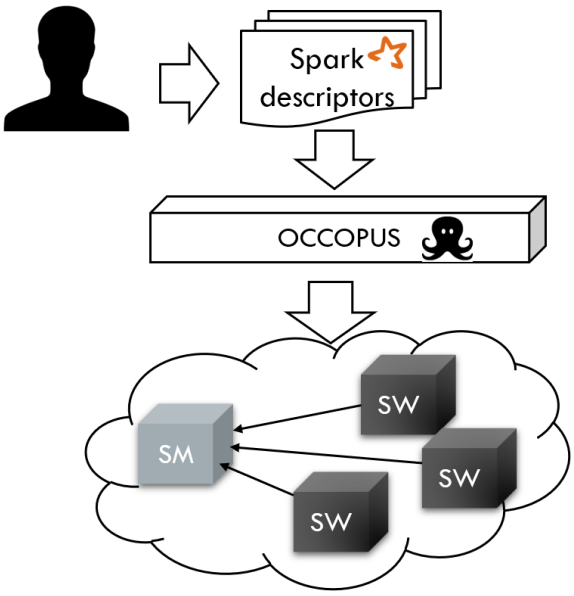

Figure 3: Spark cluster deployment using Occopus

depending on the preset scaling parameters and the size of the available quota on the target cloud.

As discussed in chapter 3.1, the Occopus tool needs an infrastructure descriptor and a node definition file as an input (including cloud-init configurations for each node type) 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 4). The dependency between the two node types was set, therefore the Worker nodes depend on the Master node. This ensures that Occopus, to launch the Master node firstly, and only after it has completed the required Spark daemons, it starts to create Worker nodes in parallel, which will join to the Spark cluster. Scaling is only allowed for Spark Worker nodes. In terms of scaling, an Occopus infrastructure, for each node, can be given a lower and an upper limit. The task of Occopus is to keep the number of copies between these limits. At startup, the minimum node number will be implemented. The scaling mechanism is available even at running time. If no value is specified, by default 1 node is built in the cluster, which cannot be scaled.

Figure 5 shows the structure of the node definition file. For security reasons, cloud-dependent 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 resource used, in this example nova plugin was used, which is needed to use MTA Cloud. In the contextualization section, it was determined that the contextualization will be done with cloud-init, as well as the cloud-init configuration files for the individual nodes. Finally, in the ”health-check” section ports were configured to check the availability of Spark deamons.

Occopus performs customization of nodes in infrastructure based on cloud-init files. Therefore completely blank images can be used, which should contain only a Linux operating system. All deployments and configurations that are needed for a properly functioning Spark

```
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 ]
```

Figure 4: Infrastructure descriptor

```
'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
```

Figure 5: Node definition file

cluster will be implemented through cloud-init, using the Occopus orchestration tool. For each node type, there is a cloud-init file, in a case of a Spark cluster, a total of two, one for the Spark Master and one for the Spark Worker. These files are the longest and most complex among the descriptors, but they do not need to be modified by the end users unless an advanced user wants to fine-tune the configuration of the Apache Spark cluster to be built before building.

5 Applications

Spark has been used for machine learning on big data. The large volumes of data need appropriate analytics tools to extract value hiding in the data. Spark is a very popular big data analitical tool mainly because it has several built in machine learning algorithms and is able to handle data transformation in a distributed fashion, as well. The combination of Jupyter notebook and Spark extended with R kernel and library import provides a very productive environment for text analysis. The traditional desktop analytics tools are not meant to handle Big Data. Hadoop, though being a popular framework for data intensive computations, does not perform well on iterative processes (like text classification) due to the cost appearing for data reloading from disk for each iteration. In order to make the usage of this analitical and machine learning tool more convenient, a Jupyter Notebook and an RStudio Web Server was installed on the Spark Master Node. Spark can be used for several machine learning tasks. Regression task, classification and clustering are built in spark ml library. This task can be used in a very few lines of code. In the text, document dataset, the performance indicators are 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. The obtained results are compared and will be discussed later. The experiments are conducted on a cluster consisting of eleven nodes: one master node and ten worker nodes. Each node has 8 virtual CPU core and 16GB of RAM. A cloud-based methodology that consists of five consecutive stages is developed for learning and evaluating classification models in parallel using Spark. The first stage is to create Resilient Distributed Dataset, which is data structure of Spark by dividing the dataset into logical partitions. These partitions may be computed 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 can be used to extract the vocabulary, and generates an array of strings from the document. This method produces a sparse representation for the documents over the vocabulary instead of dense vector representation. In our application this vectors are the inputs of the above mentioned machine learning algorithms. Before the learning we randomly shuffle and stratify our data/documents. The fourth state is testing, measuring, evaluating and ranking of the classification models.

The fifth stage is using 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. We used 80 vcpu core and the data was partitioned into 160 parts in order to use maximum capability of our spark cluster. Our methodology was applied by the Institute for Political Science of the Hungarian Academy of Sciences in their CAP [14] and POLTEXT Incubator Projects [15] to the classification of the front-page articles of the two leading Hungarian daily

newspapers, Népszabadság (NS) and Magyar Nemzet (MN) from between 1990 and 2014. The coding of public policy major topics on various legal and media corpora serves 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 MTA Cloud. The main issue here was how to exploit the available large number of physical and virtual machines in order to accelerate the process of article classification. Their application was successfully handled by using the Spark ecosystem we have developed and deployed on MTA Cloud.

6 Conclusions

There is a growing need to process very large scientific and commercial data applying Big Data and Machine Learning techniques. However, these applications require computing and storing capacity beyond the average application needs. Clouds with their elasticity are good candidates to solve these problems. However, creating Big Data and Machine Learning environments in IaaS clouds is beyond the capabilities of most scientists and industrial users. Therefore, they need higher level tools by which the creation of such systems is manageable. Our goal was to provide such a high level cloud orchestrator tool Occopus and by means of Occopus to enable the rapid and easy creation of Big Data and Machine Learning environments. The first such environment was a Hadoop/Mapreduce environment [16] and recently in the second phase of our project we have extended this experiment with the creation of a Spark environment that was successfully used by the Institute for Political Science of the Hungarian Academy of Sciences to the classification of newspaper articles. The current Spark environment supports the Spark MLlib for Machine Learning. In the third phase we are currently instigating how to create and deploy Spark Streaming environment for real-time data analysis on large amounts of data using the power of Occopus. In a long term our plan is to create all the major Big Data and Machine Learning environments (including the support of deep learning environments, too) and make accessible their Occopus descriptors in a repository. This repository will be available for the users of MTA Cloud and if other scientific communities and/or commercial companies are interested in using them we will open up this repository for further user communities. Currently results of this work are available for the academic and commercial project partners in two large EU projects: ”COLA - Cloud Orchestration at the Level of Application” and ”CloudiFacturing”.

Acknowledgement

This work was partially funded by the European ”COLA - Cloud Orchestration at the Level of Application” project, Grant Agreement No. 731574 (H2020-ICT-2016-1), by the National Research, and by the European CloudiFacturing project, Grant Agreement No. 768892 (H2020-FoF-2017). On behalf of the Project Occopus, we thank for the usage of MTA Cloud (https://cloud.mta.hu) that significantly helped us achieve the results published in this paper.

References

[1] Apache Hadoop Map Reduce Tutorial `https://hadoop.apache.org/docs/r1.2.1/mapred_tutorial.html`

[2] Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, Ion Stoica: Spark: Cluster Computing with Working Sets, Proceedings of the 2nd USENIX conference on Hot topics in cloud computing, pp. 10-10 (2010)
[3] AMPLab UC Berkleyhttps://amplab.cs.berkeley.edu/
[4] Apache Spark website https://spark.apache.org/
[5] Databricks website https://databricks.com/
[6] Salman Salloum, Ruslan Dautov, Xiaojun Chen, Patrick Xiaogang Peng, Joshua Zhexue Huang: Big data analytics on Apache Spark, International Journal of Data Science and Analytics, pp. 145-164, Volume 1, Issue 3-4
[7] DataBricks - Apache Spark and Hadoop: Working Together https://databricks.com/blog/2014/01/21/spark-and-hadoop.html
[8] Jozsef Kovacs, Peter Kacsuk, "Occopus: a Multi-Cloud Orchestrator to Deploy and Manage Complex Scientific Infrastructures", Journal of Grid Computing, Volume 16, Issue 1, pp 19–37, 2018
[9] CloudBroker website http://cloudbroker.com
[10] Merkel, D.: Docker: lightweight Linux containers for consistent development and deployment. Linux J. 2014, 239 (2014)
[11] CloudSigma website http://www.cloudsigma.com
[12] Occopus website http://occopus.lpds.sztaki.hu/
[13] COLA project website https://project-cola.eu/
[14] CAP Project website https://cap.tk.mta.hu/
[15] POLTEXT Incubator Project website https://qta.tk.mta.hu/
[16] R. Lovas, E. Nagy, J. Kovacs: Cloud agnostic Big Data platform focusing on scalability and cost-efficiency, Advances in Engineering Software, 2018