Towards a Cloud Native Big Data Platform using MiCADO

MOSA, A., Kiss, T., Pierantoni, G., Deslauriers, J., Kagialis, D. and Terstyanszky, G.

This is a copy of the author's accepted version of a paper subsequently published in the proceedings of ISPDC 2020, on-line event (originally Warsaw, Poland) 05 - 08 Jul 2020.

The final published version is available online at:

https://doi.org/10.1109/ISPDC51135.2020.00025

Towards a Cloud Native Big Data Platform using MiCADO

Abdelkhalik Mosa, Tamas Kiss, Gabriele Pierantoni, James DesLauriers, Dimitrios Kagialis, Gabor Terstyanszky
Centre for Parallel Computing, University of Westminster, London, UK
{a.mosa, t.kiss, g.pierantoni, j.deslauriers, d.kagialis, g.z.terstyanszky}@westminster.ac.uk

***Abstract*—In the big data era, creating self-managing scalable platforms for running big data applications is a fundamental task. Such self-managing and self-healing platforms involve a proper reaction to hardware (*e.g.*, cluster nodes) and software (*e.g.*, big data tools) failures, besides a dynamic resizing of the allocated resources based on overload and underload situations and scaling policies. The distributed and stateful nature of big data platforms (*e.g.*, Hadoop-based cluster) makes the management of these platforms a challenging task. This paper aims to design and implement a scalable cloud native Hadoop-based big data platform using MiCADO, an open-source, and a highly customisable multi-cloud orchestration and auto-scaling framework for Docker containers, orchestrated by Kubernetes. The proposed MiCADO-based big data platform automates the deployment and enables an automatic horizontal scaling (in and out) of the underlying cloud infrastructure. The empirical evaluation of the MiCADO-based big data platform demonstrates how easy, efficient, and fast it is to deploy and undeploy Hadoop clusters of different sizes. Additionally, it shows how the platform can automatically be scaled based on user-defined policies (such as CPU-based scaling).**

***Index Terms*—Hadoop big data platform, MiCADO, Docker Containers, Container-orchestration, cloud native, scaling big data infrastructure**

I. Introduction

"Big Data consists of extensive datasets — primarily in the characteristics of volume, variety, velocity, and/or variability — that require a scalable architecture for efficient storage, manipulation, and analysis" [1]. Big data is identified by huge *volumes* that can span several nodes; greater *variety* where the data can be structured, unstructured or semi-structured from multiple sources and/or domains; the *high-velocity* of data generation which requires higher speed for data processing; and *variability*, which reflects the change in velocity or structure. Therefore, there is a striking demand for scalable big data platforms that can handle the volume, variety, velocity, and variability challenges to achieve cost-effective performance. Such big data platforms need to offer appropriate solutions for the distributed (multi-node) storage and processing of these big datasets.

Apache Hadoop[1] has been known to be the de-facto standard big data platform. Apache Hadoop is an open-source distributed processing framework for storing and processing big data on large clusters of commodity hardware. To effectively store tremendous amounts of data (volume challenge), Hadoop stores data in a distributed way over multiple nodes, which provides fault-tolerance and enables the horizontal scaling of these nodes. To support structured, unstructured, and semi-structured data (variety challenge), Hadoop does not impose any schema validation, and hence it can support all types of data. To process data faster (velocity challenge), Hadoop moves the processing unit to data instead of moving data to the processing unit, so the processing logic is sent to the nodes where data is stored. The core Apache Hadoop project consists of Hadoop Distributed File System (HDFS), Yet Another Resource Negotiator (YARN), MapReduce, and Ozone. HDFS is a distributed file system that provides high-throughput access to application data stored on multiple nodes. YARN is a framework for job scheduling (scheduling tasks to be executed on different cluster nodes) and cluster resource management (*e.g.*, allocating system resources to the various applications running in a Hadoop cluster). Hadoop MapReduce is a YARN-based system for parallel processing of large data sets. Finally, Ozone is a new scalable, redundant, and distributed object store for Hadoop.

Hadoop has been introduced before the advent of cloud computing. Therefore, Hadoop was not originally built for the cloud, and it aimed to create big data clusters using commodity hardware. However, tremendous changes have happened since the launch of Hadoop. As an illustration, compute, storage and bandwidth resources become much cheaper. Therefore, with the recent advancement in hardware and software, along with the adoption of cloud computing, it is essential to move forward by building cloud native Hadoop-based big data platforms that can be easily deployed and managed both in public and private (on-premises) clouds.

"Cloud native technologies empower organisations to build and run scalable applications in modern, dynamic environments such as public, private, and hybrid clouds" [2]. In cloud native computing, the applications are deployed as *microservices* using open-source software tools. Every single microservice is *containerised*, and eventually, the containers are dynamically *orchestrated* to optimise resource utilisation and efficiently manage the entire application. A microservice is a small software component that can be independently developed, tested, deployed, and scaled [3]. The development and deployment of each microservice do not impact other parts of the application. With microservices, an application becomes a collection of loosely coupled services. A container is a software that packages up the application's code and its dependencies so that it can run quickly and reliably

[1] http://hadoop.apache.org/

in any computing environment. Containerisation enables fast development and efficient deployment of services and has been widely adopted by public providers. For example, Google runs all services in containers, including YouTube, Gmail, and Search. Container orchestration tools, such as Kubernetes[2] or Docker Swarm[3], automate the deployment, scaling, and management of containerised applications. Moreover, It is found that containerising genomic pipelines enable easy distribution and execution in a portable manner with a little impact on the performance [4].

Because Hadoop-based big data clusters consist of several disjoint big data tools and services, it becomes easier to containerise every single service. Therefore, Hadoop might be a good fit for cloud native computing and microservices architectures. In this paper, we are adopting MiCADO to orchestrate the big data services and manage the underlying infrastructure. MiCADO is not only utilising the container orchestration capabilities of Kubernetes or Docker Swarm, but it also provides flexible and programmable autoscaling based on user-defined policies. Additionally, MiCADO automates deployment and scales resources at both container and virtual machine (VM) levels.

Organisations are migrating 50% of their public cloud applications and/or data to either on-premises or private cloud environments in the next two years [5]. The main reason for this tremendous movement from public to private clouds is security concerns [5]; however, other top reasons include cost, performance, and control. Furthermore, automating the management of the infrastructure with the help of infrastructure as code (IaC) makes on-premises solutions more cost-effective than public Cloud for 85%-90% of workloads [6]. Therefore, providing organisations with a cloud independent (portable) automated deployment and autoscaling framework, such as the solution described in this paper, that supports the migration of various big data tools effortlessly and conveniently between multiple clouds has paramount importance.

The remainder of this paper is organised as follows: Section II describes some of the literature on big data platforms. Section III provides a quick overview of MiCADO. Section IV details the main tasks for implementing a cloud native big data platform using MiCADO. Section V describes the empirical evaluation of the MiCADO-based big data platform. Finally, Section VI concludes the work done and describes potential research directions that may require further investigation.

II. Related Work

This article summarises the literature of big data platforms in two main directions: (1) *academic literature on big data platforms*, (2) *managed cloud-based big data services*.

On the one hand, the recent academic literature on Hadoop-based big data solutions has revealed some emphasis on the cloud-based deployment as well as the performance models. For example, Loughran *et al*. [7] introduced a framework for the dynamic deployment of Hadoop MapReduce service in public and private cloud environments. They used SmartFrog[4] to manage the deployment, provisioning, and configuration of the Hadoop MapReduce. However, their proposed solution neither offers fault-tolerance nor generic policy-based autoscaling capabilities, which are two essential features in the modern big data platforms. Gugnani *et al*. [8] introduced an infrastructure-aware workflow system that integrates big data processing capabilities with workflow systems in science gateways. In their proposed solution, they automate the deployment and undeployment of a Hadoop cluster as part of the workflow. However, their solution does not offer automated scaling capability of the underlying Hadoop cluster. Astrova *et al*. [9] propose a scalable Hadoop-based infrastructure for running big data analytics. The proposed infrastructure is autoscalable based on computing and storage requirements. As a step towards building a container-based big data platform, Naik [10] proposed a container-based platform that is expected to be portable and hence runs in any cloud environment. The proposed solution focuses only on the deployment of Hadoop-based clusters using a containerised version of Apache Ambari. The proposed Hadoop-based big data cluster using Ambari is simulated in a single machine using Oracle VirtualBox. Their platform made use of pre-existing Ambari blueprints to create the Hadoop cluster. This work can be considered as an investigation of using containerised Ambari; however, it did not consider the automated orchestration or scaling of the Hadoop-based big data platform. Lovas *et al*. [11] made use of a cloud orchestrator, called Occopus, to fully automate the deployment and scalability of Hadoop clusters. Their Occopus-based deployment is portable and scalable; however, it does not incorporate the fine-grained management and scalability options found in cloud native deployments. Finally, for estimating realistic performance models, Gandhi *et al*. [12] developed a model-driven autoscaling solution for Hadoop clusters. They introduced performance models that relate job execution times to resource allocation and workload. These models are used to estimate the required resources to complete the submitted jobs based on the predefined service level agreement (SLA) requirements.

On the other hand, several public cloud providers offer managed big data solutions in the form of Platform as a Service (PaaS). Examples of these PaaS managed big data solutions include Amazon Elastic MapReduce (EMR)[5], Google Cloud Dataproc[6], and Azure HDInsight[7]. Amazon EMR is a cloud-based big data platform that offers big data tools such as Spark, Hive, HBase, and Flink. Google Cloud Dataproc offers data analytics processing suing MapReduce, Apache Hive, Pig, Flink and Spark. Moreover, Azure HDInsight offers popular open-source frameworks for enterprise-grade data analytics, including Apache Hadoop, Spark, and Kafka. However, these managed data big data platforms are provider-specific; there-

[2]https://kubernetes.io/

[3]https://docs.docker.com/engine/swarm/

[4]http://www.smartfrog.org/

[5]https://aws.amazon.com/emr/

[6]https://cloud.google.com/dataproc

[7]https://azure.microsoft.com/en-gb/services/hdinsight/

fore, organisations might face the problem of being locked-in to a specific cloud provider. Recall from Section I, several organisations are moving to private clouds for various reasons; therefore, building portable cloud native big data solutions that can efficiently work for both private and public clouds becomes a necessity.

In addition to the managed big data services, Hortonworks Data Platform (HDP)[8] is an example of cloud-agnostic big data platforms for private and public clouds. HDP is an open-architecture platform (composed of many Apache Software Foundation projects) that manages data in motion and at rest. However, deploying a Hadoop cluster using HDP is found to be time-consuming and challenging, especially for software developers who are only interested in using the platform without being involved in the mundane task of deployment and scaling (which is manual in HDP). Even though the deployment and scaling issues in HDP can be solved by using Cloudbreak, however; it is only limited to HDP blueprints and loses the benefits of cloud native solutions. This section emphasises the need for cloud native big data solutions, as the one demonstrated in this article, that are portable and can fully automate the deployment and scalability of big data clusters.

III. MiCADO Overview

MiCADO is an application-level multi-cloud orchestration and auto-scaling framework [13]. Fig. 1 shows the high-level architecture of MiCADO. MiCADO consists of two main components, namely Master Node and Worker Node. Master Node manages the MiCADO cluster by automatically deploying the Worker Nodes and schedule the microservices. Then, it monitors the deployed nodes and microservices and applies scaling policies. Worker Nodes are the ones running the actual containerised applications. Worker Nodes are continuously allocated or released based on the demand of the running microservices. MiCADO master runs and manages microservices on MiCADO workers. In MiCADO, applications are defined using a TOSCA-based Application Description Template (ADT) [14] which details the applications' topology and the required scaling and security policies.

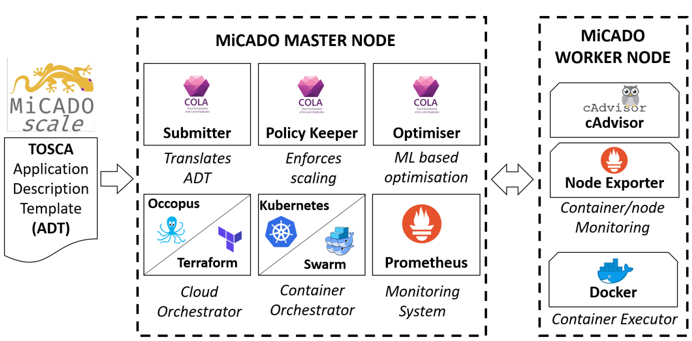

Fig. 1. MiCADO Architecture

The MiCADO Master Node follows a modular design consisting of six components. MiCADO Submitter is the primary service endpoint for creating an infrastructure to run an application and managing this infrastructure and the application itself. The MiCADO Submitter interprets the incoming ADT, and related parts are forwarded to other key MiCADO components. Creating new MiCADO Worker Nodes and deploying application containers on these Worker Nodes are the responsibility of the Cloud Orchestrator and Container Orchestrator components, respectively. The Cloud Orchestrator is responsible for communication with the Cloud API to allocate and release resources and create and shut down MiCADO Worker Nodes when necessary. The Container Orchestrator allocates new microservices (realised by containers) on the Worker Nodes, keeps track of their execution and destroys them if necessary. The Monitoring System collects metrics on Worker Node resources and on resource usage of the container services and makes this information available for the Policy Keeper component. It also provides alerting functionality in relation to the measured attributes to detect values that require reaction and sends these alerts to the Policy Keeper. Based on the metrics and alerts provided by the Monitoring System, the Policy Keeper applies the implemented scaling policies to make scaling decisions and call the components (Cloud and Container Orchestrators) responsible for allocating/releasing cloud resources and scheduling container services among the Worker Nodes. Moreover, this component makes sure that the Cloud and the Container Orchestrators are instructed in a synchronised way during the operation of the entire system. Lastly, the Optimiser is a background microservice performing long-running calculations on-demand for finding optimised setup of both cloud resources and container infrastructures.

MiCADO Worker Nodes contain the container/Node monitor that is responsible for measuring the load of the resources and the resource usage of the container services. The measured attributes are then received by the Monitoring System running on the Master Node. The Container Executor starts, executes and destroys containers upon request from the Container Orchestrator. Container components are realising the user services defined in the (container) infrastructure description submitted through the MiCADO Submitter on the Master Node.

The current version of MiCADO, 0.9, uses both Occopus [15] and Terraform [16] to provision and manage the infrastructure (such as virtual machines) using code, which enables execution on multi-cloud environments (OpenStack or OpenNebula, Amazon, Azure, Google, and CloudSigma), and also via the CloudBroker Platform [17]. For Container Orchestration, MiCADO uses Kubernetes [18]. The monitoring component is based on Prometheus [19], a lightweight, low resource consuming, but powerful monitoring tool. The MiCADO Submitter [20], Policy Keeper [21], and Optimiser components were custom implemented during the COLA Project[9].

[8]https://www.cloudera.com/products/hdp.html

[9]https://project-cola.eu/

IV. Cloud Native Big Data Platform using MiCADO

In this article, we are introducing a self-managing cloud native big data platform using MiCADO. The proposed platform is portable so that it can be deployed both in public and private clouds. Fig. 2 exhibits the architecture of the proposed MiCADO-based big data platform. As shown in Fig. 2, the MiCADO Master automates the deployment and the scaling of the big data cluster. The big data cluster follows a master/slave architecture with a pool of nodes acting as master nodes and another pool of slave nodes. The master node runs HDFS *NameNode*, besides YARN *ResourceManager* and *Timeline Server*. The pool of master nodes can include another master node to ensure high-availability, as well as a secondary NameNode. Each slave node runs a HDFS *DataNode* and a YARN *NodeManager*. Other big data services (*e.g.*, Spark, ZooKeeper, Kafka) can run either on the master or slave pool based on the required architecture of the big data platform. The main tasks for building the proposed MiCADO-based big data platform are summarised into four steps as follows: (1) containerising big data tools if they are not already containerised, (2) a choice between coupling and decoupling of compute and storage, (3) writing application description templates (ADTs) for the required big data scenario, and (4) submitting the ADT for deployment and orchestration using MiCADO. The details of these four steps are demonstrated in the following subsections.

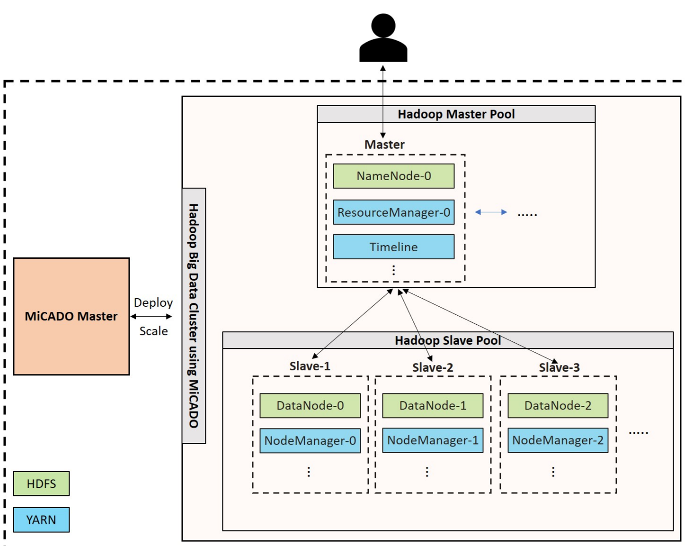

Fig. 2. Hadoop-based big data Platform using MiCADO

A. Containerising Big Data Tools

The Hadoop ecosystem consists of a set of loosely coupled tools or services such as HDFS, YARN, ZooKeeper, Kafka, Spark, and Hive. The first step in building a cloud native big data platform is to containerise this set of disjoint tools. The containerisation ensures consistent runtime environment, elasticity, and software isolation through application sandboxing. Currently, there are some excellent non-official trials to containerise Hadoop big data tools [22]. Additionally, there are also some officially containerised big data tools, such as Zookeeper[10]. However, at the time of writing this article, there are no official containers for the majority of the big data tools. A crucial task during the containerisation process is to create the proper configuration files from the relevant environment variables. As an illustration, running a containerised version of HDFs might involve running one container to act as a NameNode and other containers as DataNodes. In the case of YARN, one container can act as a ResourceManager (RM) and another as a NodeManager (NM). It does worth mentioning that the only difference between NN, DN, RM, and NM when running in MiCADO is just the environment variables (as they are all using the same container of the core Hadoop).

B. Coupling versus Decoupling Compute and Storage

By default, Hadoop couples compute and storage by bringing compute to the data to minimise data movement and overcome slow network access speed. The coupling of compute and storage is found to be a practical and effective solution with commodity hardware before adopting the cloud. However, there are problems associated with this coupling, which can be summarised in *cost* and *scalability*. Compute resources are usually more expensive than storage, so coupling will incur an extra cost when the compute resources are not efficiently utilised. Furthermore, both compute, and storage do not often need to scale the same way, so the coupling of compute and storage can lead to resource wastage when scaling both together. On the other hand, decoupling compute and storage contradicts the original Hadoop design that aimed to bring compute to data. Decoupling enables scaling compute and storage independently for better utilisation and cost savings. Decoupling is the standard in most managed cloud-based Hadoop clusters, such as HDInsight and Amazon EMR. However, decoupling may suffer from performance challenges. The proposed platform couples compute and storage; however, future work will consider decoupling compute and storage.

C. Creating Application Description Templates (ADTs)

Running containerised applications in MiCADO requires describing them in a TOSCA-based Application Description Template (ADT). The ADT is written in YAML format and describes the details of the application, such as the containers, volumes, VMs and scaling policies. Fig. 3 demonstrates an example of defining the NameNode component of the big data platform, which can be found under the $node\_templates$ section of the ADT. As an illustration, the NameNode is defined under *hdfs-namenode* and been attached to a node named *hdp-master*, while exposing the port 30010 for public access. The definition of the *hdp-master* node is shown in Fig. 4. The required number of VM instances (based on the min_workers, which is an input parameter to the ADT) is

[10] https://hub.docker.com/_/zookeeper

created in the desired public, or private cloud using either Occopus or Terraform, as shown under the *hdp-master* section of the ADT. In Fig. 4, the VM is created in AWS EC2 using Occopus. However, it can be created in any other supported public or private cloud by using the proper $type$ under *hdp-master* (*e.g.*, using *tosca.nodes.MiCADO.Nova.Compute* for OpenStack, or *tosca.nodes.MiCADO.GCE.Compute* for Google Compute Engine). Additionally, another Cloud Orchestrator (such as Terraform) can be used by merely replacing $Occopus$ under $interfaces$ with $Terraform$.

One challenge is maintaining the state of the big data tools in the proposed big data platform, which is handled using the *StatefulSet* object from Kubernetes. Each pod, the smallest deployable unit of computing in Kubernetes, in a StatefulSet have a unique ordinal index and a stable network identity. The deployment of pods happens in order; for example, DataNode-0, DataNode-1, DataNode-2. If one of the containers fails, such as DataNode-1, then Kubernetes will create a new one with the same name. Moreover, the undeployment of pods happens in order by undeploying the last deployed one first.

```
topology_template:
  node_templates:

    # Describe HDFS-NameNode
    hdfs-namenode:
      type: tosca.nodes.MiCADO.Container.Application.Docker.StatefulSet
      properties:
        image: flokkr/hadoop
        args: ["hdfs", "namenode"]
        envFrom:
        - configMapRef:
            name: hdfs-config
        labels:
          component: namenode
        ports:
        - port: 9870
          nodePort: 30010
          metadata:
            name: hdfs-namenode-public
        - port: 9870
          clusterIP: None
          metadata:
            name: hdfs-namenode
      requirements:
      - host: hdp-master
      - volume: emptydir-volume
      - container: hdfs-namenode-init
      interfaces:
        Kubernetes:
          create:
            inputs:
              spec:
                serviceName: hdfs-namenode
```

Fig. 3. A snippet of the ADT defining HDFS NameNode

Creating the ADT involves defining the scaling policies for VMs and big data tools. MiCADO can automatically scale VMs and the hosted pods (such as DataNodes and NodeManagers) horizontally based on the predefined minimum and the maximum number of nodes/pods, as shown in Fig. 5. Creating an autoscaling policy in MiCADO involves defining an *alert* and a *scaling rule*. The alert can be defined based on resources (*e.g.*, CPU, memory, storage), other application-specific metrics (*e.g.*, capacity used for HDFS, or AvailableVCores for YARN), or deadline-based. When the alert is fired, the scaling rule is applied by adding or removing pods and/or nodes.

Figure 6 demonstrates the definition of the *alerts* and the

```
# Describe hdp-master node
hdp-master:
  type: tosca.nodes.MiCADO.EC2.Compute
  properties:
    region_name: e.g. us-east-2
    image_id: e.g. ami-0d03add87774b12c5
    instance_type: e.g. t2.medium
    security_group_ids:
      - e.g. sg-93d46bf7
    key_name: -OPTIONAL e.g. ssh_key
  interfaces:
    Occopus:
      create:
        inputs:
          interface_cloud: e.g. ec2
          endpoint_cloud: e.g. https://ec2.us-east-2.amazonaws.com
  capabilities:
    scalable:
      properties:
        min_instances: {get_input: min_workers}
        max_instances: {get_input: max_workers}
```

Fig. 4. Definition of the *hdp-master* node in the ADT

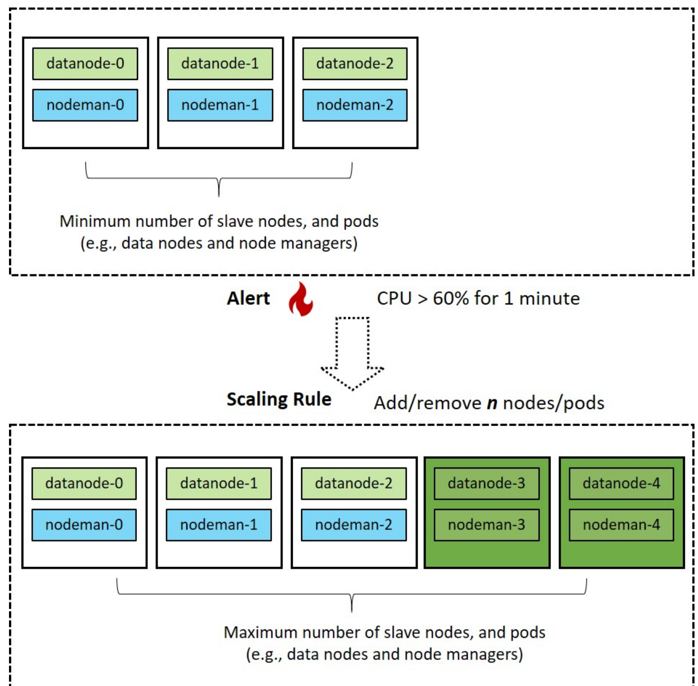

Fig. 5. Scaling of nodes and services in MiCADO

scaling rule inside the MiCADO ADT. In Figure 6, two alerts have been defined; for underload and overload detection of the *nodemanager* container based on the average CPU utilisation. The scaling rule is a simplistic Python-based script that increases or decreases the number of NameNode containers in case of service overload or underload, respectively. Further details about the ADT sections can be found here [23].

D. Deployment and Orchestration using MiCADO

MiCADO is used for automating the deployment and offering fine-grained scalability for the cloud-based big data platforms based on dynamically programmable scalability policies. Once the three preceding steps, in Sections (IV-A, IV-B, IV-C), have been completed, then the user can automatically

```yaml
policy_types:
  tosca.policies.Scaling.MiCADO.Container.CPU.nodemanager:
  derived_from: tosca.policies.Scaling.MiCADO
  properties:
    alerts:
      type: list
      description: pre-define alerts for container CPU
      default:
      - alert: service_overloaded
        expr: 'avg(rate(container_cpu_usage_seconds_total{
               container_label_io_kubernetes_container_name
               ="{{SERVICE_FULL_NAME}}"}[60s]))*100 > {{SERVICE_TH_MAX}}'
        for: 30s
      - alert: service_underloaded
        expr: 'avg(rate(container_cpu_usage_seconds_total{
               container_label_io_kubernetes_container_name
               ="{{SERVICE_FULL_NAME}}"}[60s]))*100 < {{SERVICE_TH_MIN}}'
        for: 30s
      required: true
    scaling_rule:
      type: string
      description: pre-define scaling rule for container CPU
      default: |
        if len(m_nodes) == m_node_count:
          if service_overloaded or m_node_count > m_container_count:
            m_container_count+=1
          if service_underloaded:
            m_container_count-=1
      required: true
```

Fig. 6. CPU-based scaling in MiCADO ADT

deploy the cluster without any deep understanding of the tools or the infrastructure, and MiCADO will also manage them at run-time. It should be pointed out that the three previous steps (dockerising, the choice between coupling/decoupling compute and storage, and writing the ADT) need to be done once, and the resulting ADT can be used in an unlimited number of deployments by several types of users. The final step is submitting the ADT to MiCADO, which initiates the fully automated deployment of the big data platform. Using the cloud orchestrator (Occopus or Terraform), it creates the specified number and type of VMs on the selected cloud, based on the node definition in the ADT. Once the nodes have been created and initialised, MiCADO, with the help of the container orchestrator (Kubernetes), deploys the big data tools and exposes these tools over the predefined ports. While the big data platform is running, MiCADO monitors the state of the running tools, applies automatic scaling based on user-defined scaling policies, and maintains the number of replicas required for each of the big data tools.

The proposed MiCADO-based big data platform supports the unique capabilities of the cloud (*e.g.*, rapid elasticity and on-demand self-service). Additionally, the MiCDO-based big data platform can be considered as cloud agnostic as it can be deployed both in public and private clouds without being locked only to a specific cloud provider. Most importantly, it is a step towards a cloud native big data platforms through the simplified build, deployment, and management of big data clusters by different types of users, while hiding the infrastructure complexities. As an illustration, anyone can easily and efficiently deploy a multi-node big data cluster using these four steps: (1) deploy MiCADO using Ansible[11], (2) download the ADT from the Github repository[12], (3) customise the ADT (*e.g.*, using another private or public cloud, defining different scaling policies, or changing the cluster size and the running big data tools), (4) submit the ADT to the MiCADO master. Finally, the proposed MiCADO-based big data platform enables defining fine-grained policies that enable the automatic horizontal scaling of big data tools and the underlying infrastructure (VMs).

[11]https://micado-scale.readthedocs.io/en/latest/deployment.html

[12]https://github.com/UoW-CPC/MiCADO-based-big-data-platform

V. Experimental Evaluation

This section describes the empirical evaluation of the MiCADO-based big data cluster. The experiments have been conducted on Amazon EC2 using an $t2.medium$ instance (2 vCPUs, 2.3 GHz, and 4.0 GiB Memory) for running the MiCADO master. All the Hadoop master and slave nodes in first experiment (Section V-A) are running on $t2.small$ instances (1 vCPUs, 2.5 GHz, and 2.0 GiB Memory). For the second experiment (Section V-B), EC2 instances of type $t2.xlarge$ (4 vCPUs, 2.3 GHz, 16 GiB Memory, and 2TB storage) have been used for the Hadoop master and slave nodes. The deployed MiCADO-based big data clusters consist of one Hadoop master node and n Hadoop slave nodes, where n ranges from three to twenty-seven nodes based on the required cluster size. The following subsections describe and analyse the conducted experiments in detail.

A. Hadoop Cluster Deployment Time

The purpose of this experiment is to estimate the average time required for deploying and undeploying various-sized Hadoop clusters (ranging from 4 to 28 nodes) using MiCADO. Fig 7 shows the time (in seconds) required to fully deploy and undeploy different sizes of Hadoop clusters on Amazon EC2. Both the deployment and undeployment times represent average times after deploying and undeploying each cluster size for seven different times. As can be seen from Fig 7, the average time for deploying a 28 nodes Hadoop cluster is around four minutes and 30 seconds. This experiment shows how quickly it is to deploy and undeploy various MiCADO-based Hadoop clusters without requiring any specially prepared images. It is worth mentioning that using a bigger instance type for the MiCADO master (*e.g.*, $t2.xlarge$ instead of $t2.medium$) can result in faster deployment and undeployment of the previously conducted experiments, particularly for larger cluster sizes.

B. Terasort Banchmark

The TeraSort benchmark sorts data to evaluate the performance of the MapReduce framework in the proposed MiCADO-based big data platform. TeraSort tests both the HDFS and MapReduce components of the Hadoop cluster. The TeraSort benchmark consists of three MapReduce programs namely, *TeraGen*, *TeraSort*, and *TeraValidate*. TeraGen is a MapReduce program that generates large data sets, based on the specified size, which will be the input for TeraSort. Then, TeraSort sorts the input data files that have been previously generated by TeraGen. Finally, the results of TeraSort can be validated using TeraValidate, which ensures that the keys are accurately sorted within each file.

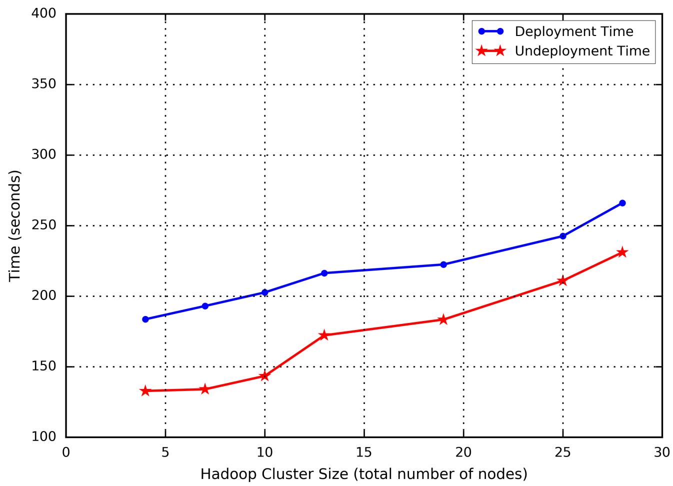

Fig. 7. Deployment time for various Hadoop cluster sizes

Fig. 8. Scalable and Non-scalable TeraGen and TeraSort

Fig. 8 shows the total execution time for running both TeraGen and TeraSort on both scalable and non-scalable MiCADO-based big data platform. TeraGen generates the input data (300GB, and 1TB in this experiment), which are then sorted by TeraSort. TeraGen runs only map tasks to generate the data set, and it does not perform any reduce tasks in the data generation process. This experiment starts with one master node and three slave nodes, which is the size of the Non-scalable cluster. The scalable version of the cluster uses a CPU-based scaling policy to add a new slave node when the CPU utilization of any of the slave nodes exceeds 60% for one minute. The maximum number of slave nodes is five, and hence the scaling policy adds a maximum of two (one at a time) extra slave nodes to the original cluster when the CPU alert is met. As shown in Fig. 8, the automatic scaling of the cluster, scalable, by adding two additional slave nodes results in approximately 40% time savings when sorting 1TB of data as opposed to the non-scalable deployment.

VI. Conclusion and Future Directions

This article started by articulating the need for the automated deployment and scalability of cloud native Hadoop-based big data clusters without being locked-in to a specific cloud provider. Moreover, it described the necessity for deploying these clusters in private clouds for several reasons such as security, cost and control. Accordingly, the article detailed the process of creating a fully-automated big data cluster using MiCADO, which is an open-source multi-cloud orchestration and auto-scaling framework for dockerised applications, orchestrated by Kubernetes. As a generic cloud orchestration and autoscaling framework, MiCADO can automate the deployment and scalability of various kinds of stateful and stateless applications in the cloud (e.g., the demonstrated Hadoop-based big data cluster). The main steps of building a cloud native big data cluster using MiCADO involves: (1) containerising the required big data tools is they are not already containerised, (2) choosing whether to couple or decouple compute and storage, (3) writing the TOCSA-based application description template, and (4) deploying and orchestrating using MiCADO. The proposed MiCADO-based deployment makes it easy for any user (such as application developer or system administrator) to automatically deploy and scale Hadoop-based big data clusters without a thorough understanding of infrastructure complexities.

Some possible future directions that require further investigation include: (1) A detailed analysis of the effect of containerisation on the performance of the big data clusters (as in the MiCADO-based deployment) as opposed to the non-containerised ones, such as HDP. (2) Decoupling compute and storage, besides adopting Apache Ozone in the MiCADO-based big data platform as it offers a scalable object store for Hadoop and functions effectively in containerised environments. (3) Authoring Hadoop-specific scaling policies that can estimate the required number of nodes to complete the execution of the required big data application based on a predefined amount of time. (4) Incorporating and testing other big data tools such as Spark, Hive as well as other user-friendly client interfaces for accessing the cluster.

Acknowledgment

This work was funded by the ASCLEPIOS – Advanced Secure Cloud Encrypted Platform for Internationally Orchestrated Solutions in Healthcare – project, No. 826093, European Commission (EU H2020). This paper is relevant and has been supported by the RABBDA (Reduce Access Barriers to Big Data Analytics) project: a Research effort between the University of Westminster, London and the Westminster International University, Tashkent.

References

[1] NIST Big Data Public Working Group (NBD-PWG), “Nist big data interoperability framework: Volume 1, definitions (version 3),” https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.1500-1r2.pdf, Tech. Rep., 2019.

[2] *Cloud Native Computing Foundation (CNCF) Charter*, (accessed March 30, 2020). [Online]. Available: https://github.com/cncf/foundation/blob/master/charter.md

[3] J. Thönes, "Microservices," *IEEE software*, vol. 32, no. 1, pp. 116–116, 2015.

[4] P. Di Tommaso, E. Palumbo, M. Chatzou, P. Prieto, M. L. Heuer, and C. Notredame, "The impact of docker containers on the performance of genomic pipelines," *PeerJ*, vol. 3, p. e1273, 2015.

[5] "Businesses moving from public cloud due to security," https://www.crn.com/businesses-moving-from-public-cloud-due-to-security-says-idc-survey, August 13, 2018.

[6] "Michael dell: On-premises solutions are more 'cost effective' than public cloud for 85 to 90 percent of workloads," https://www.crn.com/news/data-center/300100761/michael-dell-on-premises-solutions-are-more-cost-effective-than-public-cloud-for-85-to-90-percent-of-workloads.htm, March 16, 2018.

[7] S. Loughran, J. M. Alcaraz Calero, A. Farrell, J. Kirschnick, and J. Guijarro, "Dynamic cloud deployment of a mapreduce architecture," *IEEE Internet Computing*, vol. 16, no. 6, pp. 40–50, 2012.

[8] S. Gugnani, C. Blanco, T. Kiss, and G. Terstyanszky, "Extending science gateway frameworks to support big data applications in the cloud," *Journal of Grid Computing*, vol. 14, no. 4, pp. 589–601, 2016.

[9] I. Astrova, A. Koschel, F. Heine, and A. Kalja, "Scalable hadoop-based infrastructure for big data analytics," in *International Baltic Conference on Databases and Information Systems*. Springer, 2018, pp. 233–242.

[10] N. Naik, "Docker container-based big data processing system in multiple clouds for everyone," in *2017 IEEE International Systems Engineering Symposium (ISSE)*. IEEE, 2017, pp. 1–7.

[11] R. Lovas, E. Nagy, and J. Kovács, "Cloud agnostic big data platform focusing on scalability and cost-efficiency," *Advances in Engineering Software*, vol. 125, pp. 167–177, 2018.

[12] A. Gandhi, S. Thota, P. Dube, A. Kochut, and L. Zhang, "Autoscaling for hadoop clusters," in *2016 IEEE International Conference on Cloud Engineering (IC2E)*. IEEE, 2016, pp. 109–118.

[13] T. Kiss, P. Kacsuk, J. Kovács, B. Rakoczi, A. Hajnal, A. Farkas, G. Gesmier, and G. Terstyanszky, "Micado—microservice-based cloud application-level dynamic orchestrator," *Future Generation Computer Systems*, vol. 94, pp. 937–946, 2019.

[14] J. DesLauriers, T. Kiss, G. Pierantoni, G. Gesmier, and G. Terstyanszky, "Enabling modular design of an application-level auto-scaling and orchestration framework using tosca-based application description templates," in *11th International Workshop on Science Gateways, IWSG 2019*, 2019.

[15] J. Kovács and P. Kacsuk, "Occopus: a multi-cloud orchestrator to deploy and manage complex scientific infrastructures," *Journal of Grid Computing*, vol. 16, no. 1, pp. 19–37, 2018.

[16] *Terraform by HashiCorp*, (accessed March 30, 2020). [Online]. Available: https://www.terraform.io/

[17] S. J. Taylor, T. Kiss, A. Anagnostou, G. Terstyanszky, P. Kacsuk, J. Costes, and N. Fantini, "The cloudsme simulation platform and its applications: A generic multi-cloud platform for developing and executing commercial cloud-based simulations," *Future Generation Computer Systems*, vol. 88, pp. 524–539, 2018.

[18] *Production-Grade Container Orchestration*, (accessed March 30, 2020). [Online]. Available: https://kubernetes.io/

[19] *Prometheus*, (accessed March 4, 2020). [Online]. Available: https://prometheus.io/

[20] G. Pierantoni, T. Kiss, G. Terstyánszky, J. M. M. Rapun, G. Gesmeir, and J. Deslaurier, "Flexible deployment of social media analysis tools." in *IWSG*, 2018.

[21] J. Kovács, "Supporting programmable autoscaling rules for containers and virtual machines on clouds," *Journal of Grid Computing*, vol. 17, no. 4, pp. 813–829, 2019.

[22] *Apache Hadoop Big data projects on Kubernetes*, (accessed March 30, 2020). [Online]. Available: https://github.com/flokkr

[23] G. Pierantoni, T. Kiss, G. Terstyanszky, J. Deslauriers, G. Gesmier, and H. Dang, "Describing and processing topology and quality of service parameters of applications in the cloud," *Journal of Grid Computing*, 2020.