Apache Spark usage and deployment models for scientific computing

Diogo Castro[1], *Prasanth* Kothuri[1], *Piotr* Mrowczynski[1], *Danilo* Piparo[1], and *Enric* Tejedor[1]

[1]CERN. 1 Esplanade des Particules, Meyrin, Switzerland

Abstract. This talk is about sharing our recent experiences in providing data analytics platform based on Apache Spark for High Energy Physics, CERN accelerator logging system and infrastructure monitoring. The Hadoop Service has started to expand its user base for researchers who want to perform analysis with big data technologies. Among many frameworks, Apache Spark is currently getting the most traction from various user communities and new ways to deploy Spark such as Apache Mesos or Spark on Kubernetes have started to evolve rapidly. Meanwhile, notebook web applications such as Jupyter offer the ability to perform interactive data analytics and visualizations without the need to install additional software. CERN already provides a web platform, called SWAN (Service for Web-based ANalysis), where users can write and run their analyses in the form of notebooks, seamlessly accessing the data and software they need. The first part of the presentation talks about several recent integrations and optimizations to the Apache Spark computing platform to enable HEP data processing and CERN accelerator logging system analytics. The optimizations and integrations, include, but not limited to, access of kerberized resources, xrootd connector enabling remote access to EOS storage and integration with SWAN for interactive data analysis, thus forming a truly Unified Analytics Platform. The second part of the talk touches upon the evolution of the Apache Spark data analytics platform, particularly sharing the recent work done to run Spark on Kubernetes on the virtualized and container-based infrastructure in Openstack. This deployment model allows for elastic scaling of data analytics workloads enabling efficient, on-demand utilization of resources in private or public clouds.

1 Introduction

Large Hadron Collider (LHC) is in an era of excellent performance delivering collisions at an ever increasing rate which increases the amount of information recorded by LHC experiments. As an example, a new record was set in August 2018 when 13.8 petabytes of data was recorded from all sources of LHC experiments. The burgeoning size of the datasets is leading the High Energy Physics (HEP) community to modernize the analysis infrastructure with the new approaches developed in the industry. One such distributed data analytics engine that is gaining wide adaption across CERN [1] accelerator sector, physics researchers and IT infrastructure is Apache Spark [2].

Apache Spark is a unified computing engine and a set of libraries for massive parallel data processing on computer clusters. Spark supports multiple widely used programming languages (Python, Java, Scala, and R), includes libraries for diverse tasks ranging from SQL to streaming and machine learning, and runs anywhere from a laptop to a cluster of thousands of servers. This makes it an easy system to start with and scale-up to big data processing of incredibly large scale.

At the same time, there is an evolving trend towards interactive analysis at CERN, where researchers are using hosted Jupyter notebooks to perform data analysis in the cloud using SWAN [3] service. SWAN - Service for Web based data ANalysis is a cloud-based and interactive data analysis platform at CERN, accessible via a web interface. SWAN is built on top of the Jupyter [4] notebook platform for interactive data analysis, and is integrated with the CERN ecosystem of technologies for what concerns storage and software.

To address ever demanding needs of researchers to perform data analysis at scale, several extensions, optimizations and integrations are made to Apache Spark to serve the needs of HEP community. In this paper we present the recent changes and innovations of data analysis infrastructure built around Apache Spark.

This paper is structured as follows. Section 2 introduces the integration of SWAN with spark clusters. Section 3 describes the cloud-native deployment of Spark on Kubernetes [5] resource manager. Section 4 details the Evaluation and analysis of Spark on Kubernetes deployment. Finally, Section 5 discusses conclusions and future work.

2 Integration of SWAN with Spark Clusters

The Hadoop [6] and Spark service provided by CERN IT is used by the IT Monitoring service which is critical for CC operations and WLCG, IT Security for intrusion detection, LHC experiments (CMS, ATLAS) for the analytics on computing data and more recently by CERN Beams department who are developing the next generation of the CERN accelerator logging platform. All these projects use Apache Spark both for the ETL to ingest and organize data on HDFS and analytics. The CERN accelerator logging platform (NXCALS [7]) has around 1000 users all around CERN who today perform control system monitoring, operation and analysis. The team has chosen SWAN as the ideal NXCALS entry point to submit computations on the accelerator's complex logging data to the Spark, so that the results can be visualized in the Jupyter interface and stored as notebooks. This meant there was a need to integrate SWAN, the hosted Jupyter notebooks service with distributed computing framework, Apache Spark. The following sub-subsections describe the various components of the integration.

2.1 Spark Connector

SWAN users once authenticated are forwarded to JupyterHub [8] server, which manages user sessions. JupyterHub spawns a Docker container for every user in order to encapsulate their session. User containers run in unprivileged mode and their usage of the host's processors and memory is restricted. Inside the container, a Jupyter server process is created to handle the requests of the users when they are working in the notebook interface. In addition, the container has access to CVMFS [9] for software which include Apache Spark Software and CERNBox [10] for data from their sessions.

From the notebook inside the container a Spark session can be established that connects to a master endpoint on the Spark cluster in order to run a job. The master obtains resources for the job and makes them available to the driver running inside Spark session. As shown in Figure 1, the communication between the Spark driver and the cluster is bi-directional: the driver also acts as a server and listens on three ports. These ports need to be open for the driver servers to receive the incoming connections. Since the driver is running inside a Docker [11] container, some extra configuration is needed to:

- Open three ports on the user container, so that the driver servers can listen on them.
- Open a range of ports on the host (i.e. SWAN server machine). Since a SWAN server can potentially run multiple user sessions at the same time, the range has to be wide enough to provide ports for all the corresponding user containers.
- At container creation time, link the container ports to a subset of the ports in the host, so that incoming connections to those host ports are redirected to the driver servers inside the container.

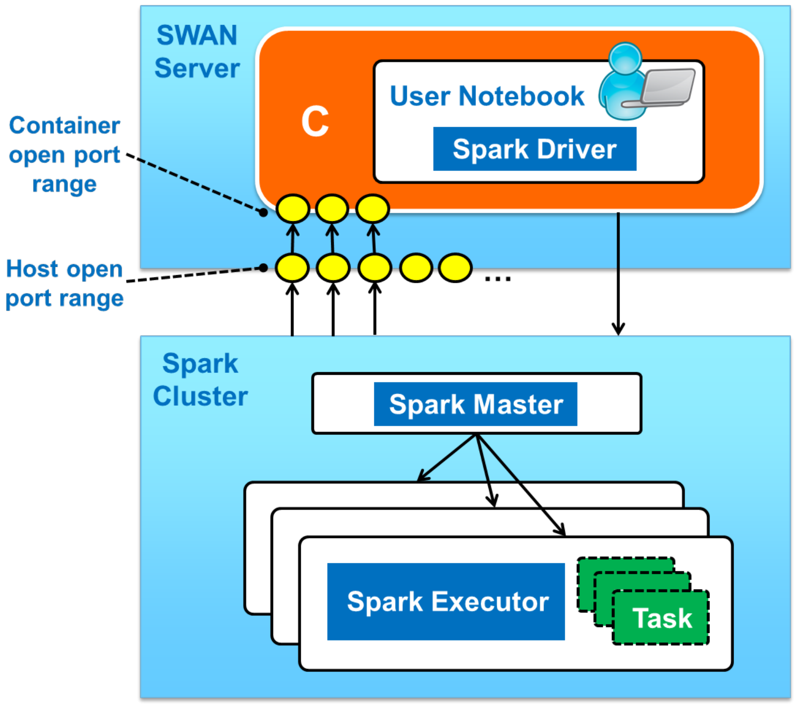
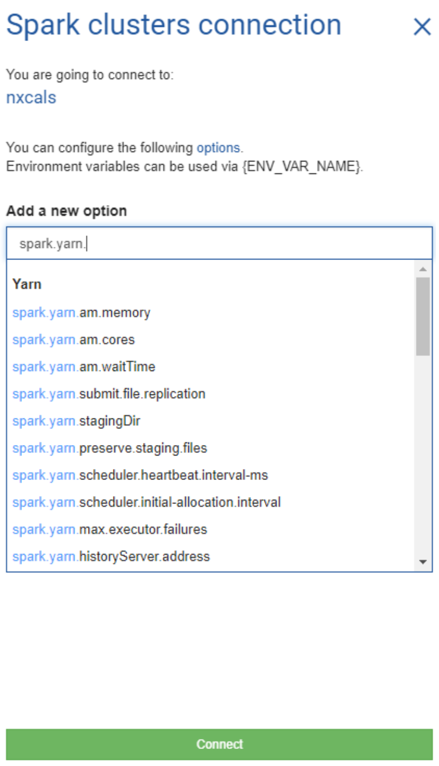


Figure 1: Interaction between SWAN server and Spark Cluster

The Spark Connector which is implemented as a Jupyter extension allows users to specify additional spark configuration as shown in the figure 2 while creation of the spark session. In addition for the most common use cases there is a possibility to create a configuration bundle as show in the figure 3 which the users can select while creating the spark session.


Spark clusters connection
×
You are going to connect to:
nxcals
You can configure the following options.
Environment variables can be used via {ENV_VAR_NAME}.
Add a new option
spark.yarn.
Yarn
spark.yarn.am.memory
spark.yarn.am.cores
spark.yarn.am.waitTime
spark.yarn.submit.file.replication
spark.yarn.stagingDir
spark.yarn.preserve.staging.files
spark.yarn.scheduler.heartbeat.interval-ms
spark.yarn.scheduler.initial-allocation.interval
spark.yarn.max.executor.failures
spark.yarn.historyServer.address
Connect

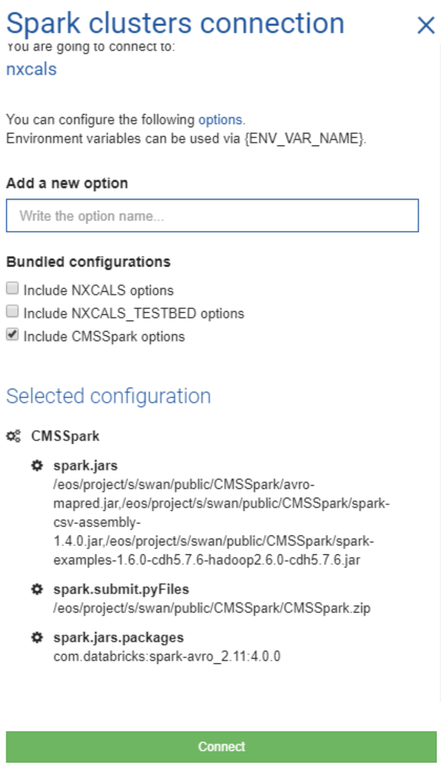
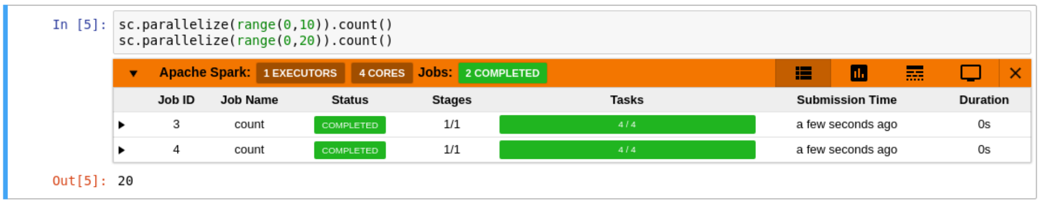

Spark clusters connection
×
You are going to connect to:
nxcals
You can configure the following options.
Environment variables can be used via {ENV_VAR_NAME}.
Add a new option
Write the option name...
Bundled configurations
☐ Include NXCALS options
☐ Include NXCALS_TESTBED options
☑ Include CMSSpark options
Selected configuration
CMSSpark
spark.jars
/eos/project/s/swan/public/CMSSpark/avro-mapred.jar,/eos/project/s/swan/public/CMSSpark/spark-csv-assembly-1.4.0.jar,/eos/project/s/swan/public/CMSSpark/spark-examples-1.6.0-cdh5.7.6-hadoop2.6.0-cdh5.7.6.jar
spark.submit.pyFiles
/eos/project/s/swan/public/CMSSpark/CMSSpark.zip
spark.jars.packages
com.databricks:spark-avro_2.11:4.0.0
Connect

Figure 2: Spark Configuration **Figure 3 : Configuration Bundle**

2.2 Spark Monitor

After creating the spark session, any computation triggered in the notebook is offloaded to the spark cluster. It is not trivial to monitor and debug the status of the computation that is offloaded to the cluster, it requires connecting to the external cluster and accessing Spark web UI server. Spark Monitor [12] extension developed in the HSF GSoC [13] enables the monitoring of jobs sent from a notebook application, from within the notebook itself.

The spark monitor extension integrates with the cell structure of the notebook and automatically detects jobs submitted from a notebook cell. It displays the jobs and stages of spark application spawned from a cell, with real time progress bars, status and resource utilization as depicted in figure 4. The extension provides an aggregated view of the number of active tasks and available executor cores in the cluster. An event timeline displays the overall workload split into jobs, stages and tasks across executors in the cluster as depicted in figure 5 which is useful to identify the stragglers and take appropriate action. An event timeline displays the overall workload split into jobs, stages and tasks across executors in the cluster.



```
In [5]: sc.parallelize(range(0,10)).count()
        sc.parallelize(range(0,20)).count()
```

	Job ID	Job Name	Status	Stages	Tasks	Submission Time	Duration
▸	3	count	COMPLETED	1/1	4 / 4	a few seconds ago	0s
▸	4	count	COMPLETED	1/1	4 / 4	a few seconds ago	0s

```
Out[5]: 20
```

Figure 4: Visualization of status and resource utilization

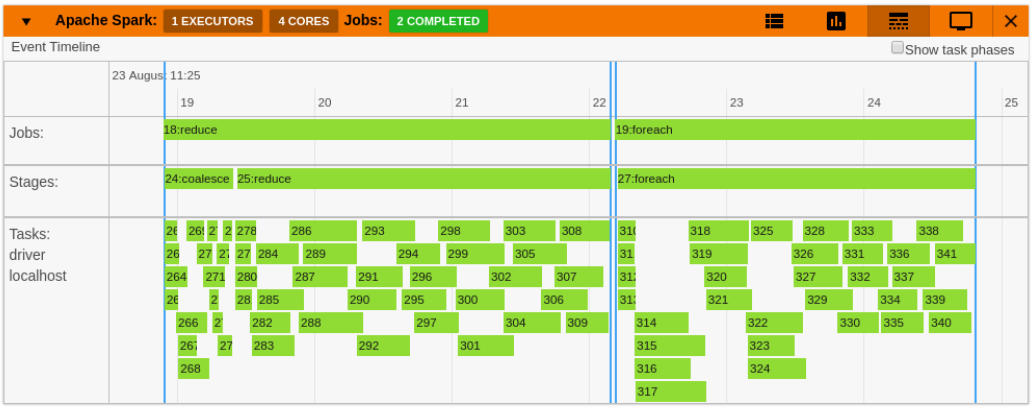
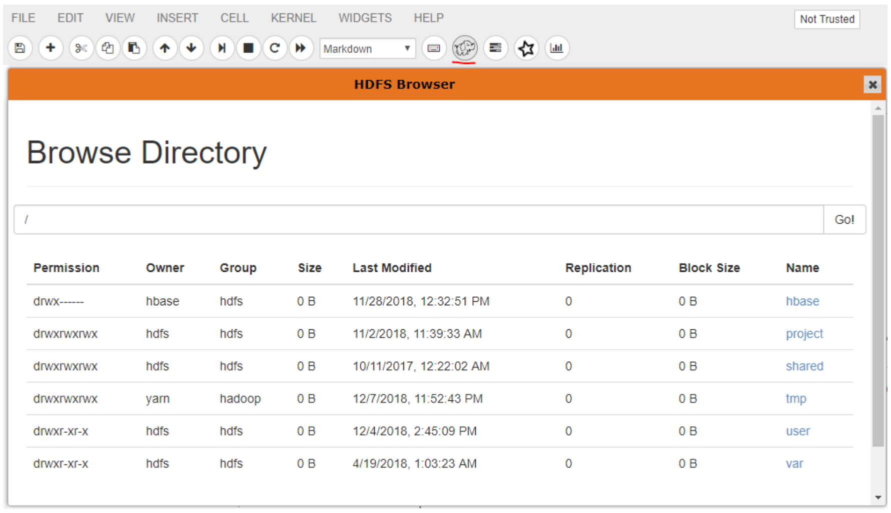


Figure 5: Visualisation of Spark Application decomposition

2.3 HDFS Browser

The input data for the analysis is on an HDFS filesystem. It is often not trivial for the physicist to know the deep directory structures to select the input data for analysis. To overcome this obstacle, HDFS Browser Jupyter plugin is created to browse the HDFS filesystem as depicted in figure 6. While creating the user container on SWAN, webhdfs token is obtained for the user and WEBHDFS_TOKEN environment variable is populated which is appended to the all the WEBHDFS REST API [14] calls.



Browse Directory

/

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
drwx------	hbase	hdfs	0 B	11/28/2018, 12:32:51 PM	0	0 B	hbase
drwxrwxrwx	hdfs	hdfs	0 B	11/2/2018, 11:39:33 AM	0	0 B	project
drwxrwxrwx	hdfs	hdfs	0 B	10/11/2017, 12:22:02 AM	0	0 B	shared
drwxrwxrwx	yarn	hadoop	0 B	12/7/2018, 11:52:43 PM	0	0 B	tmp
drwxr-xr-x	hdfs	hdfs	0 B	12/4/2018, 2:45:09 PM	0	0 B	user
drwxr-xr-x	hdfs	hdfs	0 B	4/19/2018, 1:03:23 AM	0	0 B	var

Figure 6 : Browsing the HDFS filesystem from Jupyter notebook

2.4 Authentication and Encryption

Spark supports a custom authentication protocol influenced by the ISO/IEC 9798 protocol [15] and AES encryption for the communication between the various endpoints of the driver and executor. This configuration for authentication and encryption is enabled for spark.driver.port and Block manager endpoints. For Spark on YARN deployment which is the case with SWAN, Spark automatically generates and distributes the shared secret using YARN RPC end points.

3 Apache Spark deployment models

In some cases, organizations are bound to usage of external mass storage systems for data analysis use-cases - Amazon S3, Google Cloud Storage in public clouds or on-premise mass storages like EOS at CERN. These reasons are usually already existing large datasets at rest in external storage systems, growing amount of data which cannot be handled cost-efficiently on traditional systems as HDFS or system complexity. One of such cases is current data-processing pipeline architecture at CERN, where Spark runs over Hadoop/YARN cluster with existing massive data collections on HDFS from stable production-grade use cases. The Hadoop/Spark clusters are sized for known production workloads to achieve highest possible utilization. Due to limited capacity on existing dedicated Hadoop cluster, and increasing need for large scale physics analysis of files stored at external storage service at up to 1 PB scale, there is a need for increased flexibility and faster capacity provisioning for Apache Spark clusters - specifically for workloads operating on datasets coming from external and elastic storage services. In this architecture of Spark/Hadoop, it is very difficult to scale the cluster dynamically according to the users' needs, due to the fact Spark operates on physical machines containing HDFS data. Additionally, YARN clusters are difficult to configure, maintain and have limited reproducibility and portability across infrastructures. Emergence of cloud-native and container-centric technologies as Kubernetes are expected to leverage cloud rapid resource provisioning and elasticity. Kubernetes popularity and demand in industry triggered contributions to upstream Apache Spark to allow cloud native data processing using Spark on Kubernetes.

3.1 Decoupling Compute and Storage for Big Data

The reseach by Ganesh Ananthanarayanan et. al. [16] defends the hypothesis that in cluster computing disk-locality is becoming irrelevant. This is due to the fact, that two fundamental assumptions - disk bandwidth higher than network bandwidth, and disk I/O being a considerable fraction of task duration - are no longer valid. They assume that fast networks and good datacenter networking, plus economic aspects of large-scale data storage in external service are in favour of architectures in which compute nodes are decoupled from nodes optimized for storage. In the paper published by Accenture Technology Labs on Cloud-based Hadoop Deployments [17], the Google Cloud Storage (GCS) was compared to Hadoop Distributed File System (HDFS) in terms of performance for processing the data in three scenarios: recommendation engine, sessionization and document clustering. They concluded that not only external storages as GCP offer better performance-cost ratio and better data availability, but also for fine-tuned workloads they achieved higher performance than reading from HDFS preserving data-locality. Databricks in the blog post about choosing S3 over HDFS, proves based on customer research that S3 offers higher SLA (availability and durability), performance-cost ratio, and is much more elastic data storage type than HDFS. In their benchmark of data platform in the cloud [18], they also prove that while in default setup, HDFS can reach much higher local node throughput thanks to data-locality, in the optimized setup reading from S3 can outperform in performance HDFS based on-premise platform (Cloudera Impala).

3.2 Provisioning of Spark on Kubernetes cluster

Spark on Kubernetes can be deployed using on-premise physical machines or as a cloud-managed service over cloud resources. At CERN, Spark on Kubernetes cluster is primarily deployed over OpenStack [19] private cloud using MAGNUM, creating clusters consisting of hundreds of nodes in few minutes. This allows to abstract from administrators the DevOps effort of provisioning, running and maintaining clusters, as this is a case of Spark/YARN and Spark/K8S with on-premise. Comparison of architectures regarding deployments is shown on Figure 7.

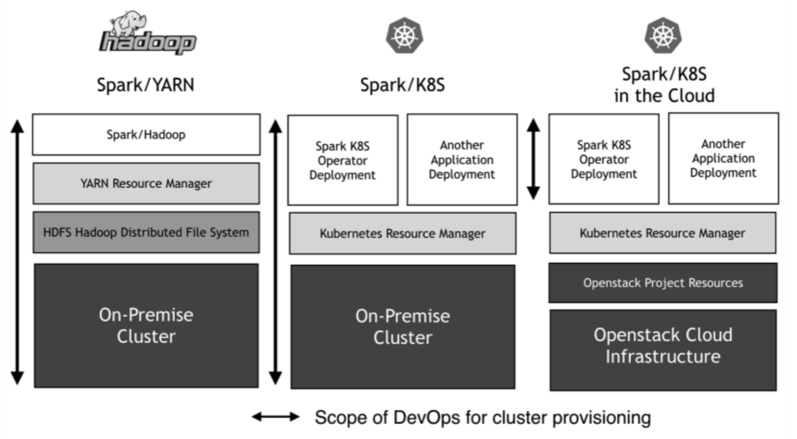

Spark/YARN
Spark/Hadoop
YARN Resource Manager
HDFS Hadoop Distributed File System
On-Premise Cluster
Spark/K8S
Spark K8S Operator Deployment
Another Application Deployment
Kubernetes Resource Manager
On-Premise Cluster
Spark/K8S in the Cloud
Spark K8S Operator Deployment
Another Application Deployment
Kubernetes Resource Manager
Openstack Project Resources
Openstack Cloud Infrastructure
Scope of DevOps for cluster provisioning

Figure 7: Top level comparison of Spark deployments

3.3 Spark Kubernetes Operator – Managing the lifecycle of Spark Applications

To control and manage Spark Applications running Kubernetes cluster, Spark Kubernetes Operator is used. In the basic concept, Spark Kubernetes Operator [20] has the following roles in the cluster:

1. Enables declarative application specification in YAML format and management of applications via dedicated API call (command line tool). Allows to customize driver and executors in a declarative way.
2. Deploys Spark Driver in the cluster (which later deploys on random/selected nodes executors) on behalf of the user via API call (command line tool).
3. Monitors driver and executors of submitted applications in a granular way.
4. Resiliency and supervision of Spark Applications. Automated application restart with a configurable restart policy, automated retries of failed submissions with optional linear back-off, automated application re-submission for updated SparkAppliation definition, and allows to run applications on schedule.

Spark Operator has been developed (mainly by Google, with effort from CERN, Microsoft and RedHat) as an open-source project.

In the cloud-native architecture of Spark on Kubernetes, there is a layered structure of software components, which are maintained using dedicated command line tools (Figure 13:

- OpenStack Magnum (OpenStack coe cluster): used to create, update, resize, delete and access the cluster.
- Kubectl (kubectl): basic tool for managing Kubernetes. Used to maintain, deploy and configuration the cluster and cluster applications.
- Helm (helm): Package manager for Kubernetes. Used to deploy and update Spark Operator in the Kubernetes cluster
- Sparkctl (sparkctl): used to interact with Spark Operator in order to submit and monitor applications defined in YAML files. Uses libraries from Kubectl to operate

The figure 8 depicts the client tools required to manage Spark Kubernetes clusters over OpenStack resources at CERN

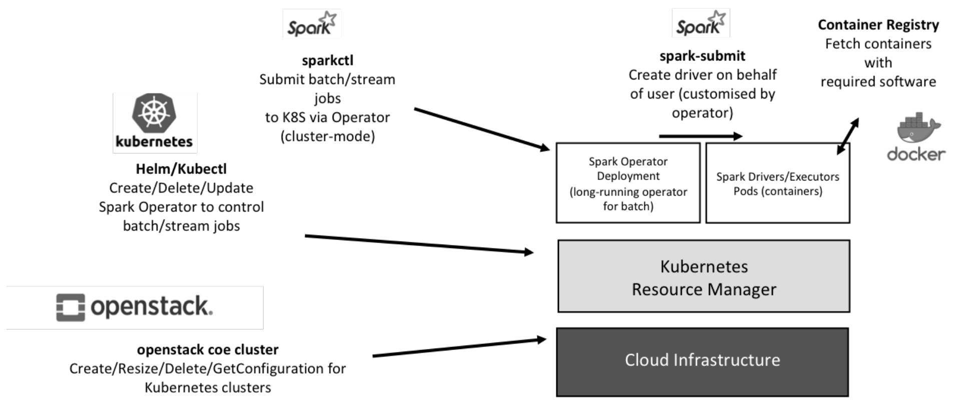

sparkctl
Submit batch/stream jobs to K8S via Operator (cluster-mode)
spark-submit
Create driver on behalf of user (customised by operator)
Container Registry
Fetch containers with required software
kubernetes
Helm/Kubectl
Create/Delete/Update Spark Operator to control batch/stream jobs
Spark Operator Deployment (long-running operator for batch)
Spark Drivers/Executors Pods (containers)
docker
Kubernetes Resource Manager
openstack.
openstack coe cluster
Create/Resize/Delete/GetConfiguration for Kubernetes clusters
Cloud Infrastructure

Figure 8: Client tools to manage Spark Kubernetes cluster

4 Evaluation of Spark on Kubernetes

The deployments of Spark on YARN and Kubernetes is validated with industry standard TPC-DS [21] benchmarking toolkit. TPC-DS provide set of repeatable, controlled and highly comparable tests which evaluate upward boundaries of systems in aspects as CPU, Memory and I/O utilization, and ability of systems to compute and examine large volumes of data. Benchmark consists of over 100 queries and has been performed in order to demonstrate differences between the architectures in terms of network throughput (I/O intensive queries), processing power and virtualization overhead (CPU intensive queries) and performance of Spark and shuffling (Shuffle intensive queries).

TPCDS Benchmark has been performed for Apache Spark with two resource managers and infrastructures (YARN/On-Premise and Kubernetes/Cloud) in order to ensure comparable and efficient execution for different types of workloads with exact Apache Spark version and exact number of resources (spark executors) processing the data stored in EOS, multiple iterations of the query have been performed and the average execution time is taken into consideration for comparison. The results of the selected queries is shown in figure 9

The following conclusions were drawn based on the results;

- I/O Intensive Query on Spark/Kubernetes and Spark/YARN could reach similar I/O Performance.
- Spark/YARN can reach higher performance on Executor CPU Time, which indicated average loss of 5% CPU efficiency due to virtualization.
- Spark/Kubernetes and Spark/YARN has similar execution time for Shuffle Intensive Query.

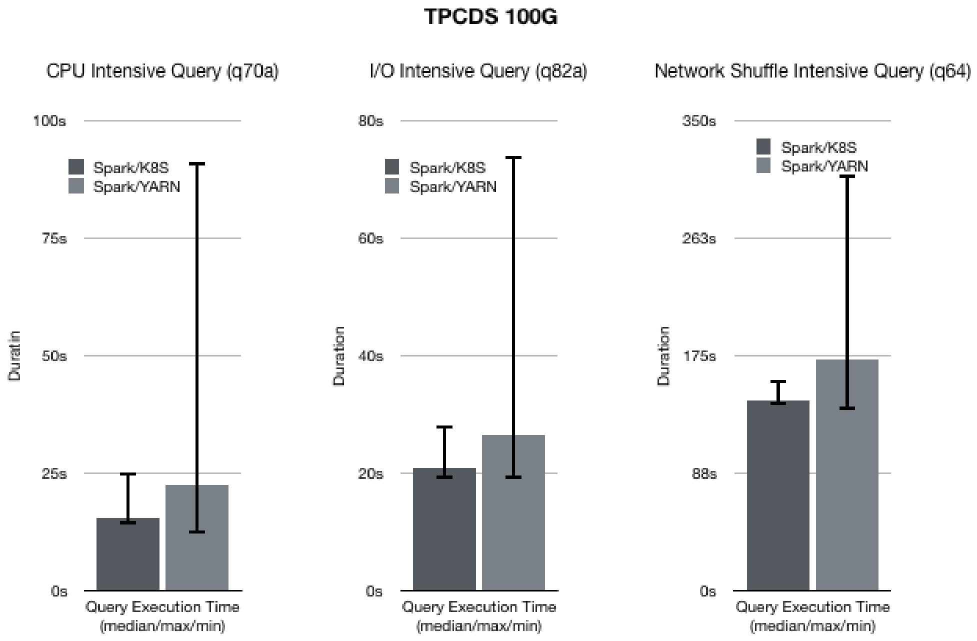


Figure 9: Elapsed time of selected queries executed against YARN & K8s

5 Conclusions and Future work

Integration of Spark clusters with SWAN allows interactive data exploration and analysis from notebook interface which allows the researchers from the accelerator sector to monitor and analyse the performance of various components of LHC machine. In addition integration of distributed processing framework like Apache Spark with hosted Jupyter notebook service like SWAN allows us to take first steps towards a unified analytics platform to handle all analytics workloads. The current deployment of Spark (on YARN) satisfies the needs of stable, predictable and production workloads from NXCals, WLCG & CC monitoring, IT security etc. The future demand from the users, especially the usage of spark for physics analysis from experiments is satisfied with Spark on Kubernetes on an OpenStack deployment model.

In Future, there are plans to develop functionality for the creation and usage of disposable Spark on Kubernetes clusters from the SWAN platform. This makes it a truly elastic unified data analysis platform with efficient usage of CERN computing resources.

References

[1] European Organisation for Nuclear Research, http://www.cern.ch

[2] Apache Spark, http://spark.apache.org/

[3] E. Tejedor, D. Piparo, J. Moscicki, L. Mascetti, P. Mato, M. Lamanna, SWAN: a Service for Web-Based Data Analysis in the Cloud, in Proceedings of the 22nd International Conference on Computing in High Energy and Nuclear Physics (CHEP 2016) (Journal of Physics: Conference Series, 2016)

[4] Jupyter: Open source, interactive data science and scientific computing, http://jupyter.org

[5] Kubernetes: Production-Grade Container Orchestration, https://kubernetes.io/

[6] Apache Hadoop, http://hadoop.apache.org/

[7] The Architecture of the Next CERN Accelerator Logging Service, https://databricks.com/blog/2017/12/14/the-architecture-of-the-next-cern-accelerator-logging-service.html

[8] JupyterHub, https://jupyterhub.readthedocs.io

[9] CernVM File System, https://cernvm.cern.ch

[10] L. Mascetti, H.G. Labrador, M. Lamanna, J. Moscicki, A. Peters, Journal of Physics: Conference Series 664, 062037 (2015)

[11] Docker: Container runtime, https://www.docker.com/

[12] Spark Monitor, https://krishnan-r.github.io/sparkmonitor/

[13] Google Summer of Code - HEP Software Foundation, https://hepsoftwarefoundation.org/activities/gsoc.html

[14] WEDHDFS REST API, https://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/WebHDFS.html

[15] Spark Security: Spark Auth Protocol and AES Encryption Support https://github.com/apache/spark/blob/master/common/network-common/src/main/java/org/apache/spark/network/crypto/README.md

[16] G. Ananthanarayanan, A. Ghodsi, S. Shenker, and I. Stoica, "Disk-locality in datacenter computing considered irrelevant," Proceedings of the 13th USENIX Conference on Hot Topics in Operating Systems, 2011.

[17] A. Wendt, "Cloud-based Hadoop deployments: benefits and considerations." Accenture Technology Labs, 2017. https://goo.gl/ZhNbVy.

[18] R. Xin, "Benchmarking Big Data SQL Platforms in the cloud." Databricks Blog, 2017. https://databricks.com/blog/2017/07/12/benchmarking-big-data-sql-platforms-in-the-cloud.html.

[19] OpenStack, https://www.openstack.org/

[20] Y. Li, P. Mrowczynski, et al., "Spark K8S Operator - Spark Application API," 2018. https://github.com/GoogleCloudPlatform/spark-on-k8s-operator/blob/master/docs/api.md

[21] TPC Benchmark DS (TPC-DS), http://www.tpc.org/tpcds