The new GPU-based HPC cluster at ReCaS-Bari

G Vino[1], M Antonacci[1], A Italiano[1], D Elia[1] and G Donvito[1]

[1] INFN, Unit of Bari (Italy)

E-mail: gioacchino.vino@ba.infn.it

Abstract. The ReCaS-Bari datacenter enriches its service portfolio providing a new GPU-based HPC cluster for Bari University and INFN users. This new service is ideal for complex applications that require a massively parallel processing architecture. The cluster is equipped with cutting edge Nvidia GPUs, like V100 and A100. Artificial intelligence, complex model simulation (weather and earthquake forecasts, molecular dynamics and galaxy formation) and all high precision floating-point based applications are possible candidates to be executed on the new service. The cluster is made up of 13 nodes and has a total computing resource of 2500 cpus, 20 TB RAM, 6 TB local SSD disk per node and 47 high performance GPUs (27 Nvidia A100 and 20 Nvidia V100). Each node can access the ReCaS-Bari distributed storage based on GPFS amounting to 8 PB. Applications are executed only within Docker containers, conferring to the HPC/GPU cluster features like easy application configuration and execution, reliability, flexibility and security. Users can request web-based IDEs (JupyterLab and RStudio) and/or a job orchestrator for submitting complex workflows represented as DAG (Directed Acyclic Graphs). Apache Mesos, a distributed resource management system, is used to orchestrate the usage of a computer cluster equipped with GPUs in a scientific environment. Chronos, Marathon combined with Mesos, provide features like scalability, fault-tolerant, resource constraints and workflow orchestration. Users appreciated an impressive speed-up of their applications up to a factor of 10. The evolution of the service, where a performance evaluation of Kubernetes as a replacement for Apache Mesos, is in the pipeline.

Keywords: Apache Mesos, Jupyter, Docker, Workflow, Nvidia, machine learning

1. Introduction

In recent decades, computers have been increasingly used in scientific research, beginning as supporting tools and progressing to the creation of scientific applications capable of performing complex analysis, simulations and mathematical modelling. Today, numerous areas of scientific research use scientific applications such as astrophysics, meteorology, biology, chemistry, physics, geology and engineering. Scientific applications can be used to simulate and predict the behaviour of complex models, such as weather and earthquake forecasts, molecular dynamics and galaxy formation or focus on whole-genome sequencing. Some of the most advanced technologies used in scientific applications are artificial intelligence, machine learning and parallel processing on Graphics Processing Units (GPUs), which enable efficient and fast computations on large amounts of data. Year after year, the amount of data and the complexity of scientific applications grow, requiring the use of scalable and powerful parallel processing architecture. The use of computer clusters equipped with

GPUs represents an interesting option for taking advantage of the hardware acceleration offered by these devices in scientific applications.

University of Bari [1] and INFN [2] also showed such tendency and the increasing interest in a high-performance computing infrastructure has resulted in the creation of a new GPU-based HPC cluster at ReCaS-Bari [3]. The most popular research areas at the University of Bari and INFN requesting such a computing power are machine learning and deep learning, whole-genome sequencing and meteorological forecasting. To deal with the enormous required computing loads and large data to process, these scientific applications require a high availability of parallel hardware accelerators, such as those provided by GPU equipped clusters or HPC[4] clusters. Without such parallel processing architecture, applications that can run on personal computers or small servers would take an unacceptable amount of time to complete the workload. In this scenario, a framework capable of managing different workloads, taking advantage of the available parallel computational accelerators, is the best solution for supporting these heterogeneous scientific applications. Our goal is to build a parallel computing infrastructure that can support the majority of research teams in the execution of their scientific applications. In this paper a GPU-based HPC cluster is described although the HPC functionalities are in the pipeline.

The hardware aspects of the cluster are presented in Section 2, more details about the used frameworks and the architecture are provided in Section 3, whereas in Section 4 the user experience and a user case are presented and finally observations about the work and future improvements are discussed in Section 5.

2. Hardware

The ReCaS-Bari datacenter has been equipped with new high performance hardware to support the execution of scientific applications. In order to reach this goal, features like GPU equipped nodes, cluster manager, web-based interactive development environment (IDEs) and workflow manager were taken into account during the design. In the following table, hardware details about the cluster are provided.

Table 1. Hardware details of the GPU-based HPC cluster.

Nodes	13
CPUs	+2500
Memory	+20 TB
GPUs	47 (20 V100 and 27 A100)
GPFS size	8 PB
Local SSD Disk	6 TB
Network	10 Gbps
Os	CentOs 7

The hardware characteristics of the cluster have been chosen to satisfy all possible applications running on it. Some tools may require large memory to load all datasets, while others require high I/O from the distributed file systems, while still others require high performance GPUs with dedicated memory, high CPUs availability and large bandwidth to move big amounts of data. The GPUs used are Nvidia V100 32 GB and A100 40GB GPUs [5], which are well supported by most GPU-based applications. The provided distributed file system is a cluster file system from IBM General Parallel File System (GPFS) [6].

3. Software

Scientific applications rarely have a monolithic structure, preferring a modular approach. Container technology plays an important role because it allows for more secure, isolated, replicable and faster execution, resulting in “containerized” applications that are made up of one or more containers. In this scenario, the container orchestrator and the cluster management take a leading role.

3.1. Middleware

In 2020, when this cluster was designed, two open source container orchestrators were considered: Kubernetes[7] and Apache Mesos[8]. Kubernetes is an "open-source system for automating deployment, scaling, and management of containerized applications" released by Google in 2014 and Apache Mesos is an open source system able to manage a cluster of distributed nodes and execute distributed applications, namely frameworks. Both Mesos and Kubernetes can run fault-tolerant "containerized" GPU-based applications and support health checks, load balancing and scaling. From our point of view, Kubernetes and Apache Mesos both fulfil our requirements. We opted for Mesos over Kubernetes for its ease of management and because our team had prior expertise with it.

Apache Mesos manages computing resources belonging to each node (CPU, memory, storage, GPU) creating a single pool of resources available for distributed applications. Frameworks could be logically splitted in tasks, each of them executed on a given node of the cluster. Apache Mesos has a master/agent architecture and can be configured with high availability (HA) to eliminate single points of failure. Mesos works along with two frameworks that simplify batch jobs and long-running service orchestrators, respectively Chronos[9] and Marathon[10]. Chronos is a distributed and fault-tolerant scheduler that runs on top of Apache Mesos. It accepts scheduled and dependent jobs allowing the execution of complex workflows, represented as directed acyclic graphs (DAGs). Chronos includes a graphical user interface for adding, deleting, listing, modifying and running jobs. It provides a REST API so that the above operations can also be executed via command line. A JSON format file is used to describe the job. Marathon is a container orchestration framework which runs on Mesos, acting as a framework for the Mesos cluster. Marathon provides several benefits like service discovery, load balancing and metrics. Thanks to health checks, it ensures long-running services are always in execution. A user-friendly web-based UI and REST API are provided. Marathon is a powerful way to run other Mesos frameworks such as Chronos; in this case, if a failure is detected, Marathon will start another Chronos instance in a working node. Also Marathon uses a task description in JSON format, which can be submitted via UI or REST API.

A container is a standard unit of software that packages up code and all its dependencies so that the application can be moved from one computing environment to another and can run quickly and reliably. Docker container images [11] are lightweight, self-contained, secure, scalable and fast. Containers isolate software from its environment, ensuring that it works consistently despite differences such as those between development and staging. One of the advantages of Docker is the availability of official images from companies or communities (Nvidia, TensorFlow, etc.)

Unfortunately, there is no open-source solution capable of monitoring cluster node resource usage, framework resource usage, tasks state, user resource usage in real-time, in a single unified interface. As a result, a dedicated tool has been implemented. Sensors retrieve information from Mesos and Marathon metric endpoints, GPU dedicated information, running container processes and combine it with data retrieved from telegraf [12]. The gathered information is stored in a Neo4J [13] instance, a graph database. Finally, a consumer collects data from the database and inserts them in an InfluxDB [14] instance, a time-series database, in order to be visualised in Grafana [15] dashboards.

3.2. User services

JupyterLab [16] is a web-based IDE for notebooks, code and data. Notebooks are documents that contain computer code as well as rich text elements, such as equations, figures, links, etc. It is widely used in scientific applications due to its interactive and exploratory computing, multiple programming languages and visualisation support. Text editor, terminal, directory viewer, and a wide range of graphical functions are just a few of its features.

Although the Python programming language is widely used for a variety of applications such as data analysis, scientific computing and artificial intelligence, it is slow to execute. This issue could be mitigated by using optimised libraries. For this reason Dask [17] and Nvidia RAPIDS [18] Python libraries were selected. Dask scales Python libraries such as numpy, pandas, and scikit-learn, as well as generic Python code. Allows the same code to be run on a laptop or on a cluster. Dask can run tasks on Kubernetes, cloud or HPC. RAPIDS is an open-source software libraries and APIs suite designed for running data science pipelines entirely on GPUs. Taking advantage of their experience in GPUs,

machine learning, deep learning and high-performance computing, they are able to speed-up considerably the execution code. The ReCaS-Bari JupyterLab container image has been built starting from the Tensorflow Jupyter Docker image [19]. Additional libraries and packages are installed (Dask and Nvidia RAPIDS among them) and the same OS user used in the ReCaS-Bari is created in order to maintain compatibility with GPFS ownerships. Its execution on the cluster is obtained through Marathon and a JSON file describing the task, containing information about the user, docker image, constraints, Python module to install, GPFS paths to mount and Marathon-LB port.

Rstudio [20] web-based IDE has been provided for those users with more experience in the R programming language. RStudio provides features quite similar to those Jupyter has such as web-based access, workbench support and file system navigator. The container has been built starting from the Nvidia cuda base docker image [21] and on top of that RStudio suite and LDAP [22] authentication are installed and configured. To run it on the cluster, a Marathon JSON file describing the task is created and then submitted to Marathon.

Chronos provides the possibility to select the user who will execute jobs. Without rules, malicious behaviours may happen. To avoid this situation, the official Chronos repository [23] was forked and an environment variable was added to force the execution of containers with a given user. Then, a Docker container image was built from the patched code. In this scenario, each requested user has his/her own Chronos instance described with a dedicated Marathon JSON format file where the environment variable is defined. Moreover, in addition to the existing submission methods, a new submission method has been implemented. It consists of an in-house Python library that simplifies the submission of complex workflows. At the time of this writing, this library has not been published yet since important improvements are still being made during interactions with users.

3.3. Architecture

The installed Apache Mesos version is the latest, the 1.11.0. A single node master was used and a non-HA configuration was implemented because until now a pre-production architecture has been used. Marathon as well was configured to run in a single instance. All long-running and user-requested applications like JupyterLab, Rstudio and Chronos are deployed via Marathon. All these applications have been containerized in Docker containers and stored in the ReCaS Docker private registry, preferred to DockerHub due its rate limit. Harbor [24], configured with LDAP authentication, is the found solution for the deployment of the ReCaS Docker private registry.

In Figure 1 is shown the system architecture with a JupyterLab instance running on a node.

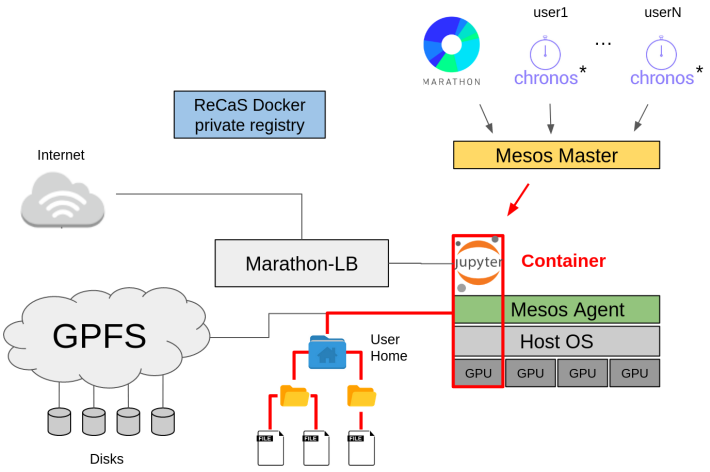

Figure 1. Figure shows system architecture with a JupyterLab instance running on a node.

When a user requires a JupyterLab instance, the Marathon JSON file describing the task is created, containing information about the user, docker image, constraints, Python modules to install, GPFS paths to mount, Marathon-LB port and then submitted in Marathon. If constraints are fulfilled, the

JupyterLab instance is launched and its host port is exported to the internet by Marathon Load Balancer. Finally the user can access its JupyterLab instance using the URL provided by the admin.

4. User experience and performance

After the implementation and initial test phases, a large number of research teams expressed their interest in the ReCaS-Bari GPU-cluster capabilities to speed up their applications. Artificial Intelligence, whole-genome sequencing and image processing are the most popular application categories running on the cluster. On average, the speed-up detected from users corresponds to a factor 4-5 (max 10-15). The overall knowledge of users on Docker container and parallel/distributed computing is low, leading us to invest in guides and courses. Half of these teams requested a web-based IDE, JupyterLab or RStudio. The latter half requested the capability to submit GPU-based jobs. Most of these research teams have not yet published their work so it is not possible to cite or share overall results, differently from computing related measurements. On average, users are pretty satisfied concerning the access to the web-based tools by which access data stored in GPFS, the provided support and the improved execution time of their applications. A single research work can be cited here to demonstrate the performance of the described new GPU-based HPC cluster at ReCaS-Bari. It is a preliminary physics research with title "Multi-charm baryons: Ξ_{cc}^{++} and Ω_{cc}^{+}" published in the ALICE 3 Letter of Intent [25]. In this work, a machine learning algorithm has been used to improve the detection of rare particles obtained at the Large Hadron Collider [26] for the ALICE experiment [27]. The tuned XGBoost algorithm [28] improved the traditional and manual approach up to a factor of 4-5 in the region of interest. The work consisted of an initial development phase, where a JupyterLab instance was used to write the code and for the model coarse-grain tuning. The second phase consisted of the conversion of the code in a pipeline, to be submitted via Chronos, for fine tuning the model. Thanks to the GPU-cluster performance and capabilities, the presented result has been obtained in a few weeks despite the enormous dataset size, amounting to 6 TB. Figure 2 shows how containers are deployed on the cluster and the resulting pipeline.

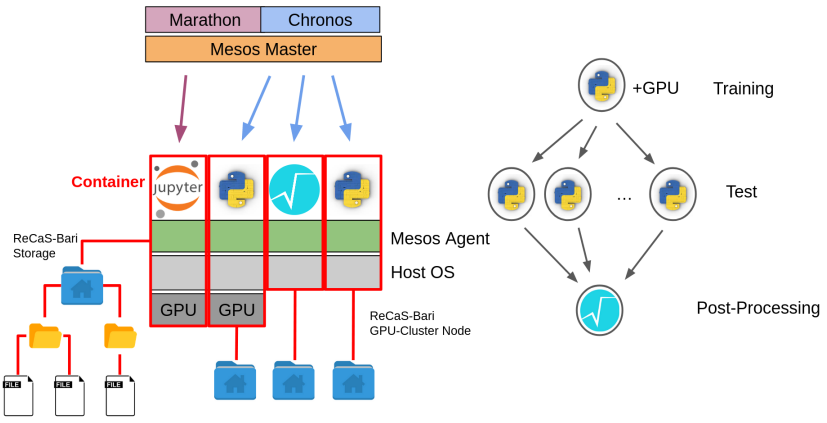

Figure 2. Figure shows how containers are deployed on the cluster and the resulting pipeline.

5. Conclusions

A new GPU-based HPC cluster at ReCaS-Bari and its impact on the scientific application running on it was presented. The cluster is composed of 13 nodes, 2500 CPUs, 20TB of RAM, 47 Nvidia GPUs (20 V100 and 27 A100), where each node has access to a 6TB fast local SSD disk and to the 8PB GPFS distributed file system. Apache Mesos, along with Marathon and Chronos, showed a great capability to manage cluster nodes and orchestrate containers, both for long running services and batch jobs, in a fault-tolerant, scalable, secure, isolating and efficient way. On top of that, two service types are successfully provided to the users: web-based IDEs (JupyterLab and RStudio) and the submission of GPU-based complex workflows. Users are pretty satisfied with their application performance and on average a speed-up with a factor 4-5 has been measured (max 10-15).

Thanks to the accumulated experience, improvements have been planned. Now, we don't have any strong reasons to still prefer Apache Mesos over Kubernetes. Its large community provides new versions, with new features and bug-fixing, quite often than Apache Mesos, where the last version is at least 2 years old. This will lead us to find alternatives for Marathon and Chronos. Kubernetes allows job submission but there is no support for complex workflows. Many open source solutions are available like Apache Airflow [29], KubeFlow [30] and MLFlow [31]. R kernel will be added to Jupyter so that an unique web-based IDE will be used. The integration between Jupyter, Dask and Kubernetes will be implemented to overcome the current limitation concerning the JupyterLab instance that reserves resources, in favour of resource release after an configurable inactive period. LDAP authentication for JupyterHub has been implemented and tested and its merging with IAM [32] will be investigated. Apache Spark will be added to the service portfolio enabling the execution of big data workloads. The cluster will be integrated in the national INFN-DataCloud PaaS [33], to be part of the federated computing nodes available for INFN-Cloud. Finally, aiming to have a HPC cluster, will investigate the use of Infiniband to speed-up the internode connections.

References

[1] University of Bari, https://www.uniba.it
[2] INFN, https://home.infn.it
[3] ReCaS-Bari, https://www.recas-bari.it
[4] HPC, https://en.wikipedia.org/wiki/High_performance_computing
[5] Nvidia GPUs, https://www.nvidia.com
[6] GPFS, https://www.ibm.com/docs/en/gpfs
[7] Kubernetes, https://kubernetes.io/
[8] Apache Mesos, https://mesos.apache.org/
[9] Chronos, https://mesos.github.io/chronos/
[10] Marathon, https://mesosphere.github.io/marathon/
[11] Docker, https://www.docker.com/
[12] Telegraf, https://www.influxdata.com/time-series-platform/telegraf/
[13] Neo4J, https://neo4j.com/
[14] InfluxDB, https://www.influxdata.com/products/influxdb-overview/
[15] Grafana, https://grafana.com/
[16] JupyterLab, https://jupyter.org/
[17] Dask, https://www.dask.org/
[18] Nvidia-rapids, https://rapids.ai/
[19] Tensorflow jupyter docker image https://www.tensorflow.org/install/docker
[20] RStudio, https://posit.co/
[21] Nvidia cuda base docker image, https://hub.docker.com/r/nvidia/cuda
[22] LDAP, https://en.wikipedia.org/wiki/Lightweight_Directory_Access_Protocol
[23] Chronos repository, https://github.com/mesos/chronos
[24] Harbor, https://goharbor.io/
[25] ALICE 3 Letter of intent, https://cds.cern.ch/record/2803563, 78-86
[26] LHC, https://home.cern/science/accelerators/large-hadron-collider
[27] ALICE experiment, https://alice.cern/
[28] XGBoost, https://xgboost.readthedocs.io/en/stable/
[29] Apache Airflow, https://airflow.apache.org/
[30] Kubeflow, https://www.kubeflow.org/
[31] Mlflow, https://mlflow.org/
[32] IAM, https://en.wikipedia.org/wiki/Identity_management
[33] INFN-DataCloud Paas, https://indigo-paas.cloud.ba.infn.it/home/login