Available online at www.prace-ri.eu

Partnership for Advanced Computing in Europe

The PRACE Data Analytics Service

Service description and results

Agnès Ansari [a], Alberto Garcia Fernandez [a], Bertrand Rigaud [b], Marco Rorro [c], Andreas Vroutsis [d]

[a] CNRS/IDRIS - agnes.ansari@idris.fr, alberto.garcia-fernandez@idris.fr
[b]CNRS/CC-IN2P3- bertrand.rigaud@cc.in2p3.fr, [c]CINECA – m.rorro@cineca.it,
[d] EPCC - A.Vroutsis@epcc.ed.ac.uk

Abstract

This paper describes the work completed in the scope of PRACE 5IP/WP6 - Service 6 Data Analytics in Task 6.2: Design and Development of new Service prototypes.
Among the technical domains covered by the "Data analytics" term, we decided to focus on the current trends that show a growing interest in the community of data scientists: machine learning and deep learning techniques, which make use of automated algorithms, as they can offer faster dataset analysis than more conventional methods. Thus, we evaluated how these techniques can benefit from HPC environments with powerful CPUs and GPUs to manage the models' complexity and accelerate the training, as well as to handle the increased amount of training data.
We describe the PRACE Data Analytics service that relies on a set of coherent components: frameworks, libraries, tools and additional features to support, facilitate and promote the data analytics activities in PRACE and help users in running their machine and deep learning tasks over the PRACE systems.
Then, we present the results we obtained for a set of deep learning benchmarks and real use cases we ran on the different PRACE architectures while using these components, that confirm their efficiency.

1 Introduction

Data Analytics refers to techniques able to help to forecast future behaviours or events, and to recommend actions or to guide decision making. It focuses on why some events happened - and what will happen next. Data Analytics includes various domains, tools and techniques, among which, the machine learning field. Machine learning concentrates on prediction from known and learned properties, based on a learning/training set. Deep learning is part of the machine learning field, and is concerned with techniques inspired by the brain, called neural networks.

The working group's activity mainly focused on deep learning, given the growing interest in the community of data scientists in this field.

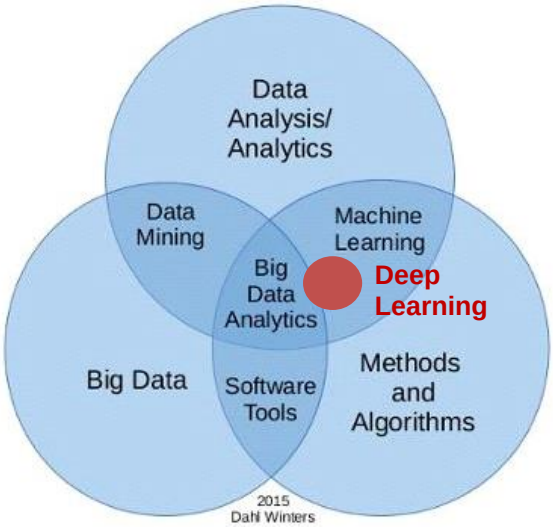

Figure 1 - The data science fields and their relationships

The purpose of the T6.2 – Service 6 - Data Analytics was to investigate the deployment of new Data Analytics technologies on HPC systems, and develop prototype solutions by means of pilot scientific use cases to access the functionalities, and to evaluate how the new prototypes could be adopted, as production services, in a next phase.
To achieve these objectives, we focused on the following actions, in order to develop the prototypes, and think about a global data analytics service, for the purpose of a future deployment and production service:

- Identify a set of Deep Learning and Machine Learning services, frameworks, tools and libraries to be made available on the PRACE Data Analytics infrastructure, composed of systems running different architectures.
- Offer any additional tools or services that facilitate the use of the Deep Learning and Machine Learning services on the PRACE infrastructure and allow users to quickly gain expertise in these fields.
- Provide user documentation and best practice guides that describe the most efficient way to use these prototypes/technologies on the various architectures.
- Offer from basic to advanced Deep Learning and Machine Learning user trainings to satisfy the user expectations and allow them to complement their expertise.
- Guarantee the availability of these services, frameworks, tools and libraries, by setting up periodic health checks that will reflect the quality of the PRACE services.

The document is organized as follows:
Chapter 2 describes the machine learning workflow. It helped us to characterize the user working environment that will be described thereafter. In chapter 3, we present the frameworks and libraries we have identified as being currently the most relevant to our concern. Chapter 4 describes the user working environment and the related tools that might be considered. Chapter 5 summarizes the HPC resources available over the PRACE infrastructure. Chapter 6 discusses the integration of the Data Analytics infrastructure in the HPC environment.
Chapter 7 describes the additional services we have identified following the experiences we got from running the prototypes. They provide additional features that offer users a full data analytics environment that can greatly enhance the user productivity and be profitable. A dataset download service that can make standard datasets easily

available to users and the Data Analytics GitLab project available within the PRACE GitLab are part of these additionald features.
Chapter 8 presents the results we obtained while running a set of deep learning benchmarks and real use cases and shows the efficiency over the various architectures.

2 The Machine Learning Workflow

In order to characterize and develop the most appropriate data analytics services, we relied on the machine learning workflow to better identify user needs, depicted in Figure 2, below. For deep learning cases, models refer to neural networks. This workflow consists of managing data during an initial data acquisition phase, followed by a data preprocessing phase and then to train, evaluate and deploy the models during a inferencing phase. Then, it is possible to make predictions.
In an HPC environment, these workflows are executed through jobs that are submitted to a batch scheduler.
The entire process can be monitored and visualization tools can be used to display data or results.

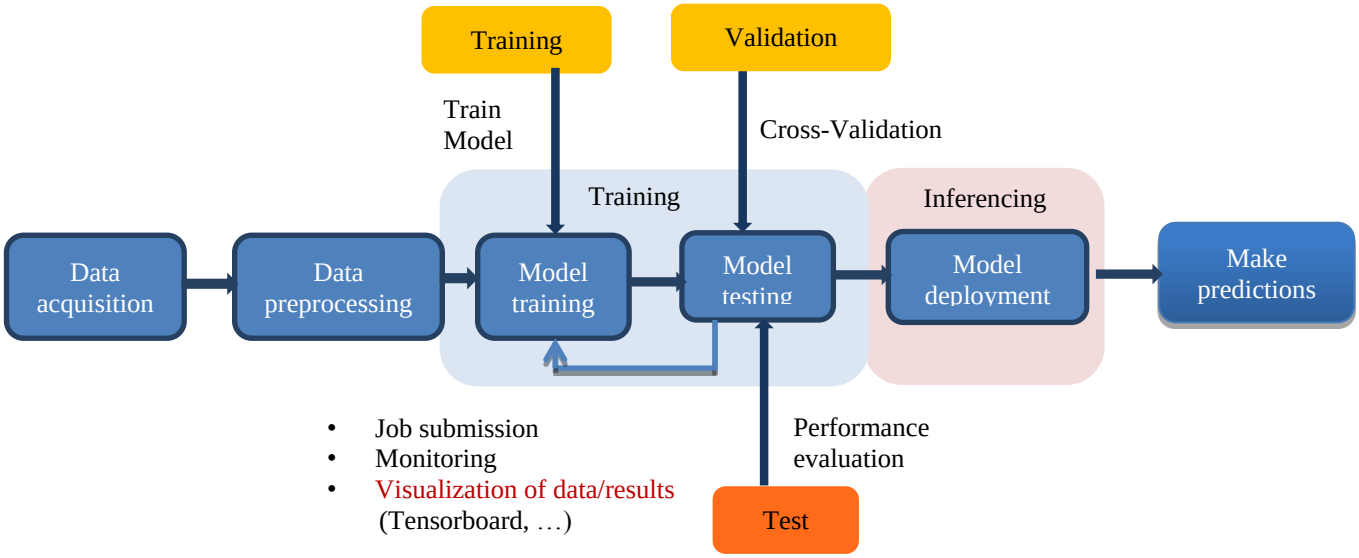

Figure 2 - The Machine Learning workflow

3 Frameworks and libraries

The deep learning experiments have been conducted with two of the most popular frameworks, currently **TensorFlow** and **Caffe**. Apart these frameworks, some additional libraries and components have been considered such as **Keras**, **Horovod** and **Anaconda**. All these components are described below. Note that Keras is now fully integrated to TensorFlow 2.0. It also includes a new distributed training feature.

Caffe (http://caffe.berkeleyvision.org, n.d.) stands for "*Convolutional Architecture for Fast Feature Embedding",* is a C++ library released under BSD-2 license, developed by Berkeley AI Research (BAIR) and by community contributors.

This library was widely adopted both in academic research and in large-scale industrial applications (Facebook, NVIDIA, Intel, Sony, Yahoo! Japan, Samsung, Adobe, A9, Siemens, Pinterest and Embedded Vision Alliance, among others). Data Layers can read input from raw images, databases, HDF5 files or directly from memory. Vision Layers implement tasks commonly encountered in imaging (convolution and deconvolution, pooling, spatial pyramid pooling and crop), while Recurrent Layers introduce the concept of memory (Recurrent Neural Network - RNN, Long Short Term Memory - LSTM). A number of activation layers are implemented, and common loss functions are present.
The simplest approach involves the definition of a neural network through the compilation of a plain text file containing a protocol buffer schema; solver parameters are defined in a secondary text file. Both files are then fed to the main Caffe executable for a training procedure. The learned model is then serialised as a binary protocol

buffer and can be exported, along with the definition of the network, to one of the supported architectures, even mobile phones, for inference. Due to its open source (BSD) licence, this library counts a relevant number of forks; the most notable ones are:

- IBM-Caffe from the PowerAI development team, contains performance enhancements, aiming at leveraging NVIDIA NVlink bandwidth on Power8 systems.
- Intel Caffe is dedicated to improving BVLC original code when running on CPU (in particular Haswell, Broadwell and Xeon Phi).

It is worth noting that Caffe, from version 1.0, is in maintenance mode: development will continue with upcoming version 1.1, but it will be placed side by side with a new development line: Caffe2.

TensorFlow (https://www.tensorflow.org, n.d.) is an open source software library for machine learning. It has been originally developed by the Google Brain Team and can be used for a large variety of applications. The Deep Learning (DL) models are expressed as a data flow graph, where each vertex (node) in the graph is a computation (mathematical operations), while the edges represent the tensors, i.e. multidimensional data arrays that flow between the nodes. There has been a very fast community growth around this tool and it is now the major tool for Deep Learning experiments and currently overpasses Caffe/Caffe2. TensorFlow can be deployed to one or more CPU and GPUs and also offers a distributed (multi-node) capability. TensorFlow provides various APIs for python, java, C and Go.
It has been deployed in 2 different ways in the Data Analytics working group i.e., whether we compiled the source code or we used the PowerAI platform from IBM. PowerAI offers a compiled and optimized version by IBM and runs inside containers at IDRIS. We avoided the use of downloaded versions which have shown lower performance.
TensorFlow is still under development, with every new release providing both a set of new functionalities and significant better performance when compared to the previous release. For performance reasons we avoid to use the TensorFlow package available with Anaconda.

The Tensorboard data visualization toolkit is part of TensorFlow. It includes tools for tuning a neural network and for monitoring its performance.

Keras (https://keras.io, n.d.) is a high-level neural network API. It has seen a broad adoption in the research community and within industry, and it is often used on top of TensorFlow. It supports different types of network topologies and allows for easy and fast prototyping. Keras offers a set of pretrained models that can be reused such as VGG16-19, ResNet50 and Inception V3. In this case, the network weights and parameters come from a pretraining on the ImageNet dataset, then they are considered during the model creation phase. As an additional layer on top of to Tensorflow, we experienced some overhead by using Keras compared to Tensorflow only.

It supports different hardware platforms beyond CPUs, such as NVidia GPUs and Google TPUs.
Keras has a built-in support for multi-GPU data parallelism and a distributed training support through Horovod (see below) and the TensorFlow estimators.

Horovod (https://eng.uber.com/horovod, n.d.) is an open source distributed deep learning framework for TensorFlow developed by Uber. Horovod supports the data parallelism (instead of model parallelism) approach to distributed training, which involves splitting up the data and training on multiple nodes in parallel. Figure 3 below describes the data parallelism approach.

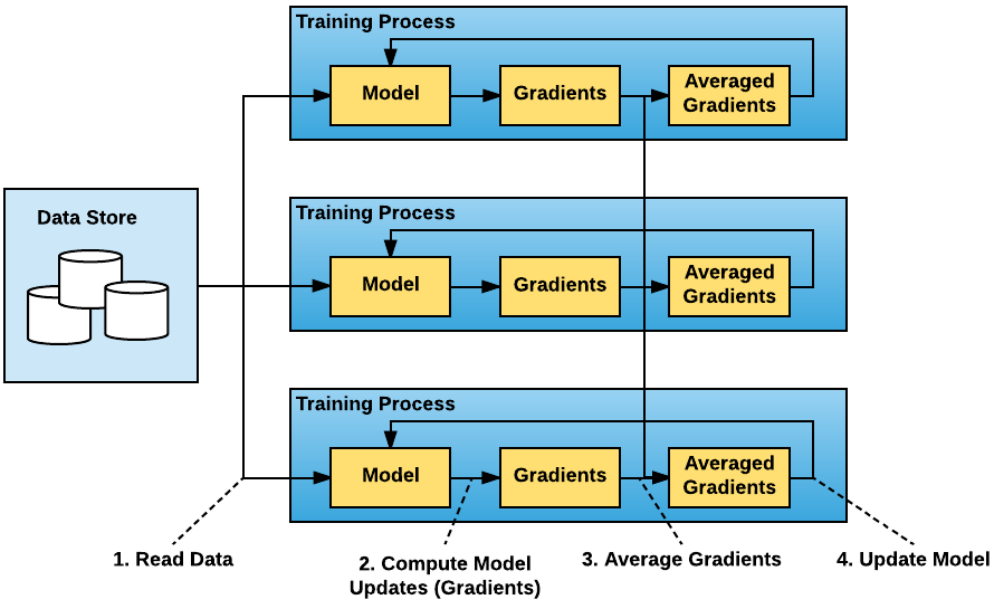

Figure 3 - The data parallelism paradigm.

The data-parallelism distributed training paradigm is as follow:

1. *Run multiple copies of the training script and each copy:*
   a) reads a chunk of the data
   b) runs it through the model
   c) computes model updates (gradients)
2. *Average gradients among those multiple copies*
3. *Update the model*
4. *Repeat (from Step 1a)*

Unlike the standard distributed TensorFlow package, which runs with workers that train data and compute gradients and parameter servers to average their gradients, Horovod is based on a different algorithm for averaging gradients and communicating those gradients to all nodes (Steps 2 and 3 above), called ring-allreduce.

In the ring-allreduce algorithm, each node communicates with two of its peers, in order to send and receive chunks of the data buffer. Concretely, users utilize MPI (such as Open MPI) to launch TensorFlow. MPI then transparently sets up the distributed infrastructure necessary for workers to communicate with each other.
Horovod is written in python and is based on NCCL 2, the NVIDIA's library for collective communication that introduced the ability to run ring-allreduce across multiple machines.

Anaconda (https://www.anaconda.com/distribution, n.d.) is a distribution that groups the most popular python data science libraries. It allows for efficient data manipulation, scientific and statistical computation and visualization. It can also be used to create machine learning and deep learning models. We used it to manage packages and dependencies.

Note that even though we did not use them during our experiments, it is worth mentioning the **Jupyter Notebooks** [6], often used by beginner software developers in machine learning and deep learning. Several tutorials and codes are available on Internet based on notebooks rather than scripts, in order to facilitate the user learning curve and makes it more educational.

4 Working Environment

In order to conduct a neural network training, data scientists might perform the following tasks, which will guarantee the effectiveness of the training:

- Monitor the training
- Check the training process and visualization
- Debug
- Checkpoint the training process

These tasks are detailed in the following sections. They require some additional tools that should be installed and will constitute the user working environment in addition to the frameworks and libraries previously mentioned.

4.1 Monitoring the training

Monitoring the execution of a training process is fundamental, as the execution time can take up to several hours. Therefore, it is very important to check the progression of the learning process, as well as the optimal usage of the hardware resources.

Our experiments were conducted by using the **nvidia-smi** and the standard Linux **htop** tools for checking respectively the GPUs and CPUs resource utilization as well as the standard Linux **nmon** which allows to monitor both GPUs and CPUs and other system resources and provides a real-time monitoring or a recording mode for later use.

Note: that during training, most calculations are performed on GPUs, so the CPUs load usually appears low.

4.2 Checking the training progress and visualization

The progression of the learning process requires to check the evolution of the loss and of the accuracy of the model in order to detect if it is learning properly or if there is a risk of any wrong behaviour such as overfitting. We used the **Tensorboard** visualization tool, which comes with Tensorflow, to display the progress of the TensorFlow applications. During training execution, TensorFlow is able to generate event files, which gather summary data. These files can be used as inputs to the Tensorboard web application, to inspect runs and graphs. A common practice is to use a dashboard that visualizes statistics over time, such as model loss or learning rate. Tensorboard can also be used to visually compare multiple executions of a model with different hyperparameters, and to check the behaviours of these different runs. Figure 4 below shows a comparison of different runs with Tensorboard.

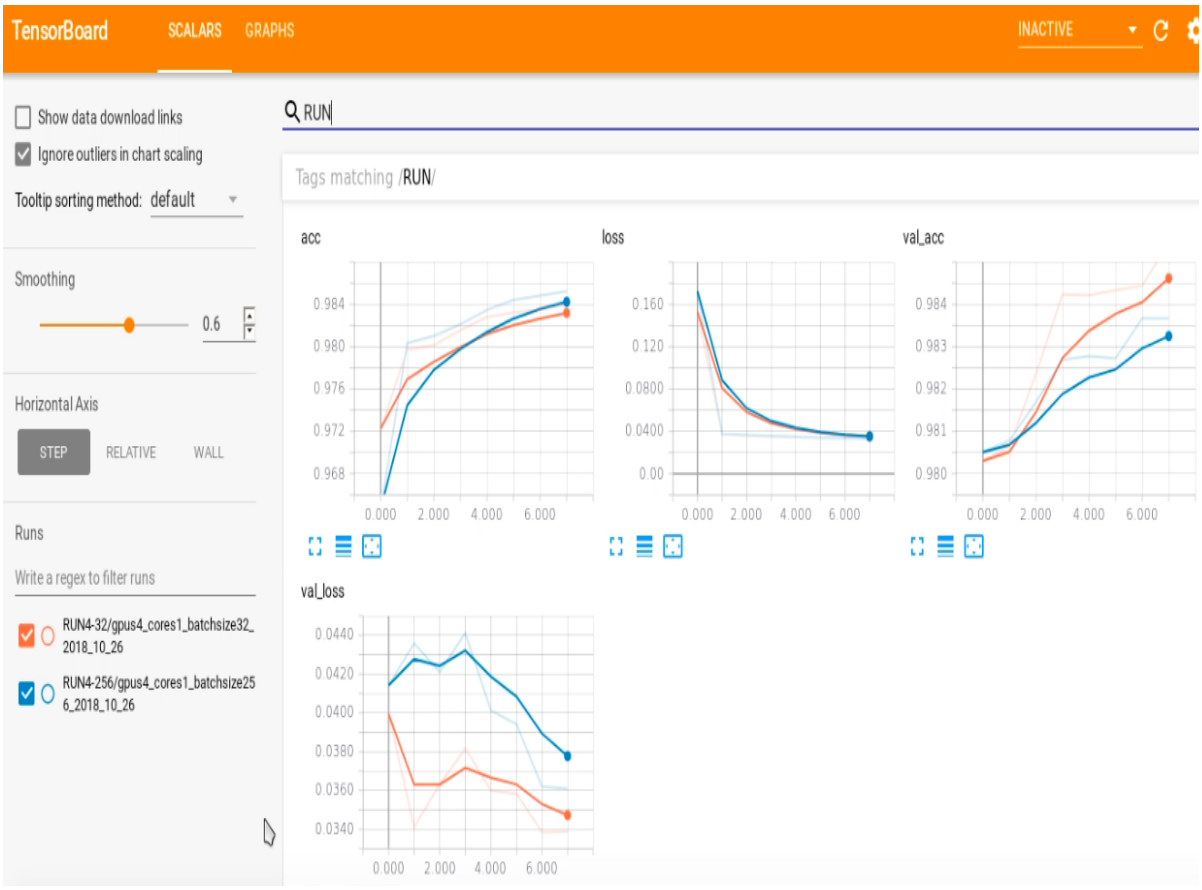

Figure 4 - Comparison of different runs with Tensorboard

4.3 Debugging

Debugging parallel applications remains a hard task, when users are required to collect and cross-reference profiles from several compute nodes. Horovod offers a profiling tool, called *Timelines.* It provides a way to get operation timelines across nodes. Thus, users can view exactly what each node was doing at each time step throughout a training job. This helps identify bugs and performance issues. This tool is compatible with the Chrome trace event profiling viewer.
Users can enable timelines by setting a single environment variable and can view the profiling results in the browser through *chrome://tracing*.

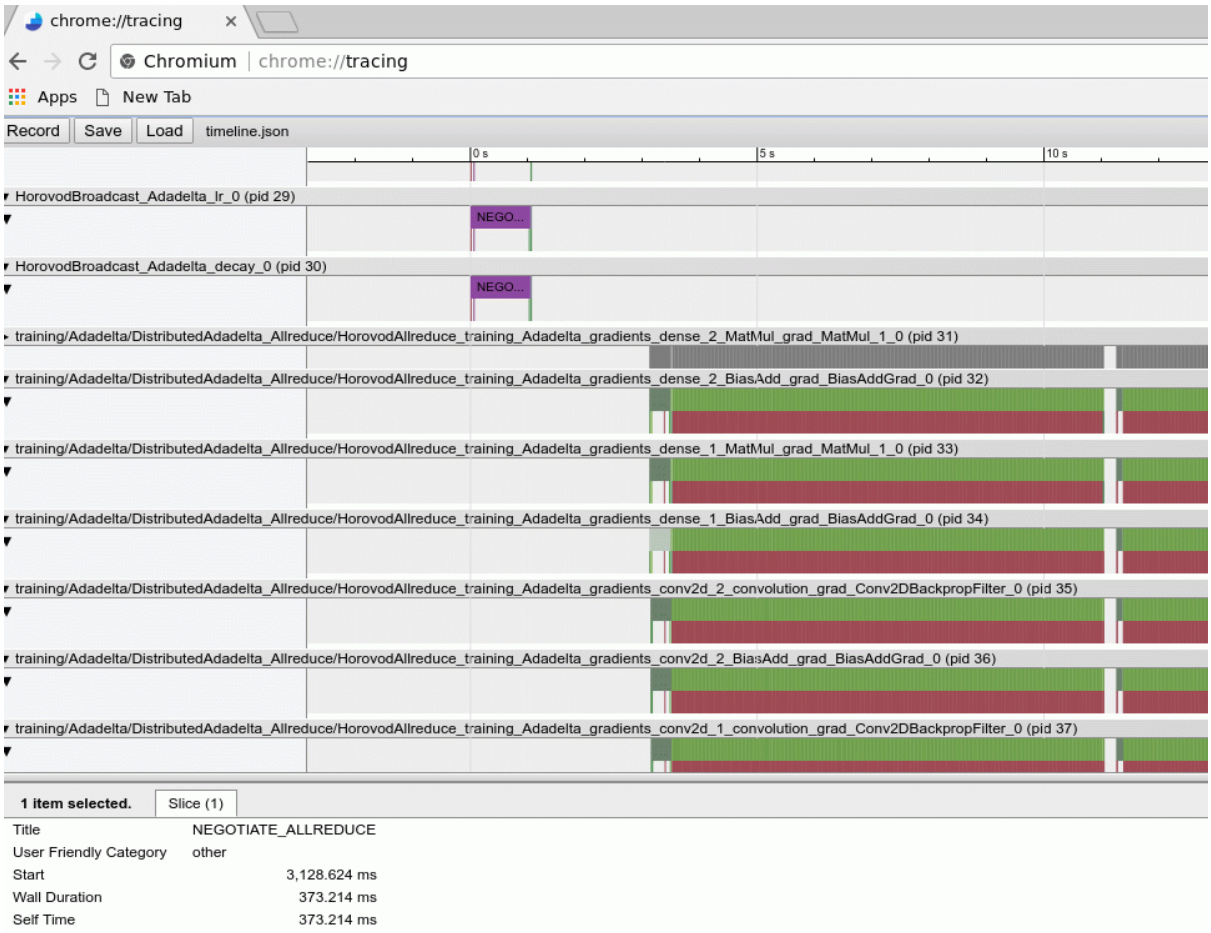

Figure 5 - The Horovod Timeline profiling tool. The processes running on the nodes are depicted as a time function.

4.4 Checkpointing

Training deep learning models can take hours. If the experiment is aborted due to an unexpected error, such as a lack of memory, for example, then a lot of work is lost. Therefore, it is important to perform checkpointing of experiments. Checkpointing can also be used when one might want to just resume a particular state of the training for a new experiment, or to try different options, restarting from a given state. But the major reason why checkpointing a training model is needed, is that it is how to keep the trained models, for later use, during the prediction (inference) phase, the ultimate goal of Machine Learning.

A checkpoint contains the information needed to save the current experiment state, so that one can resume training from this point. This information includes:

- The model architecture, allowing its re-creation.
- Model weights.
- Training configuration (loss, optimizer, epochs).
- The optimizer's state. This allows training to resume exactly where it has been left.

Given that an epoch is when an entire dataset is passed forward and backward through the network, the interval, i.e. the number of epochs between checkpoints, can be specified. The checkpointing strategy ranges from a checkpoint only at the end of the training, to one at the end of every epoch, to multiple checkpoints every *n epochs* and keep track of the *best* one with respect to some validation metric for longer training.

Both TensorFlow and Caffe have checkpointing features.

Note: Checkpointing can also be set up at the batch scheduler level, although it does not provide the same features.

5 HPC Resources

The following table describes the HPC resources with their main characteristics available within the PRACE infrastructure to run Data Analytics:

Site	Type/ Cluster Name	Architecture/CPU	Accelerators	Librairies/ Applications/ Services	LRMS	Environnement	OS/Storage
CNRS/ IDRIS	Prototype cluster Ouessant	• Power 8 • 2 Power 8/node • Processors with NVLink bus	1.1.1. 4 GPUs/node (Nvidia Tesla P100 SXM2) 16GB	• PowerAI framework (TensorFlow, Caffe, Keras) • Compiled TensorFlow with Keras, • Distributed environments: Horovod, DLL, gRPC	Spectrum LSF	• Module • Secured docker installation to run PowerAI jobs • Anaconda or Miniconda	• RHEL 7.3 • IBM Spectrum Scale (GPFS) : 128 TB
CINECA	Prototype cluster Davide	• Power 8 • 2 Power 8/node • Processors with NVLink bus	1.1.2. 4 GPUs/node (Nvidia Tesla P100 SXM2) 16GB	• Compiled TensorFlow with Keras, Distributed environments: Horovod,, gRPC	SLURM	• Module	• CentOS 7.5 • NFS 697TB
CNRS/ CC-IN2P3	Production K80 cluster	• Intel • 2 Xeon E5-2640v3/node	4 GPUs/node (Nvidia K80) 12GB	• Compiled TensorFlow with Keras, Distributed environments: Horovod,, gRPC	Univa Grid Engine	• Module • Singularity • Miniconda	• CentOS • GPFS
EPPC	Urika-GX	• Intel • 2 Xeon E5-2695/node • 256 GB	-	Intel-TensorFlow BVLC-Caffe Intel-Caffe	MESOS	• Anaconda • Virtual conda environments	• CentOS 7 • 2 or 4 TB HDD per node • 4 to 8 TB SSD per compute node • Lustre
EPCC	Cirrus	• Intel • 2 Xeon E5-2695/node • 256 GB	-	Intel-TensorFlow BVLC-Caffe Intel-Caffe	PBS	• Anaconda Virtual conda environments	• Lustre 406 TB
CINECA	Marconi A2	• Intel • Xeon Phi 7250	-	Compiled TensorFlow with Keras	SLURM	• Module	• CentOS 7.2 • GPFS 10 PB
CINECA	Galileo	• Intel • 2 Xeon-E5-2630/node	2 GPUs/node (Nvidia K80)	• Compiled TensorFlow with Keras, Distributed environments: Horovod,, gRPC	SLURM	• Module	• CentOS 7.0 GPFS 5 PB

As shown in Fig 6 below, systems with Power8 CPUs offer a full NVLink connectivity between CPUs and GPUs. On systems with x86 CPUs (ex Intel Xeon), the connectivity of the CPU to the GPU happens through the PCIe bus, although the GPUs connect to each other through NVLink.

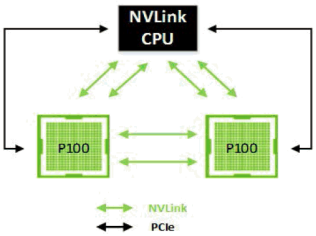

Figure 6 - The connectivity of the Nvidia Tesla P100 GPU on Power 8 machine.

6 Integration in the HPC environment

Today, the deep learning frameworks have not yet reached a sufficient level of maturity, so they continue to evolve quickly, with new versions being frequently released. These releases tend to introduce loss of compatibility between the various components and previous versions, which complicates installations and mandate frequent code changes. Besides, not all frameworks and tools are always available on all the different Linux flavours (eg: PowerAI from IBM ran on Ubuntu only, and was not available on RHEL). It is, thus, required to offer users tools that provide them flexibility during the installation phase, in order to allow them to easily package their own environment. To fulfil these requirements, several solutions have been deployed by partners in the working group such as:

a) Container-based solutions, with Singularity, or Docker.

b) No-container solutions:

- Conda virtual environments in anaconda/miniconda installations.
- Anaconda/miniconda installations.

The container solution a) is studied in the T6.2.5 “Lightweight virtualization” group so, we will not discuss it in this paper. Conda virtual environments provide an alternative to containers, and can simplify the security concerns which comes from the Docker solutions, in the sense that no privileges are required to use them and the support of this technology is not Linux-flavour dependent. Conda virtual environments are usually deployed by end-users to manage their own installations. They can also be deployed on a shared base, so that users can reference these global shared installations. However, this solution appeared to be tedious to manage and error-prone, with some bugs identified. The anaconda/miniconda installations remain the simplest, but don’t provide a lot of flexibility to users.

Special attention must be given to the number of inodes, which tends to grow very quickly with the various conda installations that can be cloned for different purposes. So, building the conda environments from miniconda2/3 rather than anaconda2/3 can be more appropriate for some systems.

Given these environments, the batch jobs often look like the following templates:

```
# Simple Deep Learning job skeleton without container

# Batch scheduler ressources specification
…
# Load the user environment
ml tensorflow/py3/1.10

# Run the DL application
python ./myNN.py
```

```
# Simple Deep Learning job skeleton with container

# Batch scheduler ressources specification
…
# Load the user environment
module load singularity

# Run the DL application using Singularity (or Docker)
singularity exec tflow.img myTraining.sh
```

7 Additional Features

We have developed two additional features to help users to perform their Data Analytics tasks:

- The PRACE GitLab Data Analytics project.
- The Dataset download service.

These additionald features are described in the following sections.

7.1 The PRACE GitLab Data Analytics project

Given the large amount of work completed within the group, one of our concerns was to make this work available to users, so they can easily gain expertise, and, in turn, share their own experiences and results. For that we defined the Data Analytics GitLab project within the PRACE GitLab, in order to build some kind of a PRACE AI knowledge database. We started to build this database by gathering the material we used in the Data Analytics working group for the evaluation of the DL/ML tools over the different HPC architectures, including standard benchmarks and small use-cases. This material can be reused by researchers that often start their first AI evaluation steps in a similar way, namely by running these types of benchmarks or use-cases.
It is available at https://repository.prace-ri.eu/git/Data-Analytics.It contains three projects:

- Benchmarks
- Use-cases
- Datasets

The Datasets project, see Figure 7, gathers information on datasets and scripts that can be used to download them. These datasets can be used by any Data Analytics user projects. The following template shows how a dataset is described using metadata in the Data Analytics GitLab:

Dataset Name: self-explanatory.
Description: a short description of the dataset and of its content.
Size: indicates the dataset size (training, tests and validation sets).
File type: can be compressed archive, archive file, and so on.
License: specifies the license type and usage restriction.
Tag: defines tags that can be used to search for.
Url: specifies the site url.
Dataset download url: specifies the dataset download url and its checksum.
PRACE dataset location: indicates where to find the dataset in the PRACE infrastructure (in the case of large file, the datasets are not stored in the GitLab DA projects group).
PRACE download script: indicates where a download script can be found in the DA projects group.

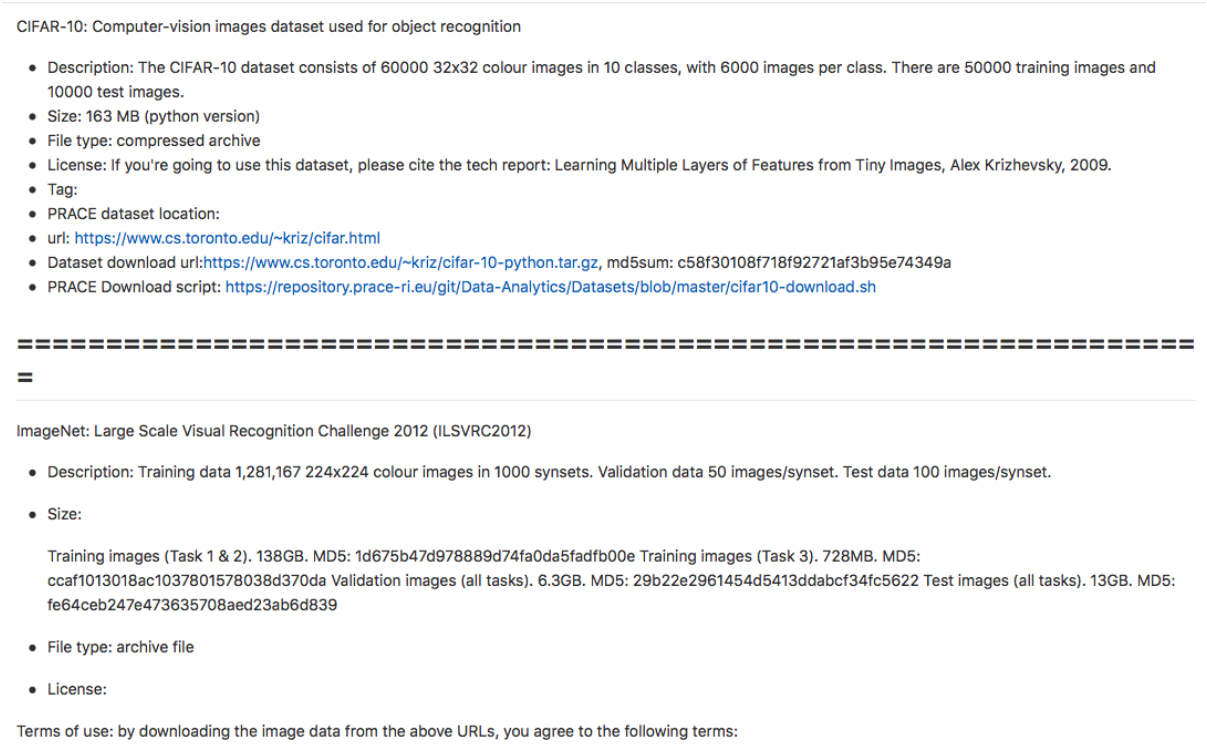

Figure 7 - The Dataset project

The benchmarks project contains the available benchmarks and all the files needed to run them. The following template shows how a benchmark is described using metadata in the Data Analytics GitLab:

Benchmark Name: name of the benchmark.
Authors: provided by: name and email address.
License: specifies the license type and usage restriction.
Description: gives a short benchmark description.
Type: scripts or notebooks**.**
Language: can be python 2/3, java, R, Julia, …
Environment: example: python 3.6+, TensorFlow, Keras, Horovod.
Tag: defines tags that can be used to search for.
Output: describes the benchmark output: example: data files, visualization, log files, models, …
PRACE url: indicates where the source code can be found in the Data Analytics projects group.

The use-cases project contains the available use-cases and ancillary tools needed to run these use cases. The use-case template is similar to the benchmark template, as described above.

7.2 The Dataset Download Service

During the development of our prototypal services, we identified the need to offer users easy access to datasets that are commonly used in the AI domain or to any other specific datasets that can be defined as reference datasets for the PRACE AI infrastructure. Indeed, whereas small data files can be downloaded quickly from the internet, dataset downloads of large files can become tedious, lasting for several hours for a complete download. A possible solution is to share these datasets in a dedicated storage space in the PRACE infrastructure, in order to benefit from the fast, reliable and secure PRACE VPN network and enhance the user productivity.

Currently, the first release of this service requires to install an iRods client (root access needed, rpm available for Linux distributions). It uses the iRods protocol which allows to create parallel threads so, it can speed up the transfer of large files but it is less popular than http/https. In this case, the datasets are read using the anonymous irods account. A future release will allow users to download the files by clicking on a url defined in a html page or use commands such as the standard curl or wget using the http/https protocols. The full dataset download service specifications are described in (Agnès Ansari C. , Dataset download service Specifications).

Today, the dataset download service is provided by CINECA (Marco Rorro, The dataset download service design document for use by the PRACE Data Analytics service) with one TB of storage disk. It is implemented through iRODS (https://irods.org, n.d.) (Integrated Rule-Oriented Data System) and currently used by the B2SAFE EUDAT (https://www.eudat.eu/b2safe, n.d.) service for the federation of data nodes. This service provides both dataset download and upload capabilities to the PRACE users using an anonymous access.

8 Results

Our approach was the following: We identified a set of benchmarks to run over the PRACE AI infrastructure, ranging from small datasets and simple models, to larger datasets and more complex models, in order to evaluate the performance impacts of the various architectures, while increasing test overall complexity. We extended this first benchmark-based by identifying real use cases which could help us to better understand the needs of real users, in terms of deep learning and machine learning.

Standard benchmarks are based on popular neural networks models, and use standard datasets. Real use cases have their own datasets, can use their own models, some popular model, or a customized model deriving from a popular model.
We first describe the datasets and the neural networks models we used for benchmarks and use cases. Then, we present the results for each experiment.

8.1 Datasets

In order to compare our results with the ones found in the scientific literature, we decided to use some open datasets, rather than proprietary ones. These standard datasets are used for applications ranging from image processing to speech recognition The main datasets we used for our experiments are CIFAR-10 and ImageNet, related to image processing applications. Table 1, below, describes such datasets:

	CIFAR-10	**ImageNet: ILSVRC 2012**
Elements	32x32 RGB pictures	RGB annotated pictures 400x350 on average
Total number of elements	60 000	1 250 000
Number of Training elements	50 000	1 200 000
Number of Validation elements	10 000	50 000
Categories	10	1000
Dataset size	163 MiB	138 GB (Training) + 6.3 GB (Validation) + 13 GB (Test set)
Availability/Location	https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz (also keras inner function tf.keras.datasets.cifar10.load_data())	http://www.image-net.org/challenges/LSVRC/2012/

Licence/ Restriction of use	If you're going to use this dataset, you need to cite the tech report: Learning Multiple Layers of Features from Tiny Images, Alex Krizhevsky, 2009.	Terms of use: by downloading the image data, you agree to the following terms: You will use the data only for non-commercial research and educational purposes. You will NOT distribute the above URL(s).p Stanford University and Princeton University make no representations or warranties regarding the data, including but not limited to warranties of non-infringement or fitness for a particular purpose. You accept full responsibility for your use of the data and shall defend and indemnify Stanford University and Princeton University, including their employees, officers and agents, against any and all claims arising from your use of the data, including but not limited to your use of any copies of copyrighted images that you may create from the data.

Table 1 - Datasets used during the benchmarks experiments.

8.2 Models

For our experiments we used a set of popular Neural Network models:

- **AlexNet** (Krizhevsky): Proposed by Alex Krizhevsky, Ilya Sutskever and Geoff Hinton, it is considered the first network that popularized Convolutional Layers in Computer Vision. AlexNet significantly outperformed the second runner-up (top 5 error of 16% compared to runner-up with 26% error) on ImageNet ILSVRC challenge in 2012. The network shares the same structure with LeNet (Yann LeCun), but was deeper, bigger, and features Convolutional Layers stacked on top of each other.

- **GoogLeNet** (Szegedy): Also known as Inception. It dramatically reduces the number of parameters in the network (7M, compared to AlexNet with 61M), were adopted by Szegedy et al. from Google in GoogLeNet, winner of ILSVRC 2014.

- **OverFeat** (Sermanet): Localization task winner of ILSVRC 2013, this integrated framework obtained remarkable results, also in classification and detection tasks. The novelty of the approach resides on the application of a CNN on multiple locations of the image and the ability to predict position and size of the bounding box surrounding objects.

- **UNet** (Olaf Ronneberger) is a network that belongs to the encoder/decoder architecture. The U-Net architecture is built upon the Fully Convolutional Network (Long) and modified in a way that it yields better segmentation in biomedical imaging. The U-Net owes its name to its symmetric shape, which is different from other FCN variants. It is the winner of the ISBI cell tracking challenge 2015. It is able to work with few training images while producing precise segmentations.

- **VGG11** (Simonyan): This is one of the neural network proposed by Visual Geometry Group (VGG) of Oxford University in order to study the effects of varying the convolutional network depth on image recognition tasks accuracy. This group ranked number one on localisation and second on classification tasks ILSVRC 2014.

- **VGG19** (Karen Simonyan) is a convolutional neural network that is trained on the ImageNet dataset. It has been released in 2014 by the Visual Geometry Group at the University of Oxford, this family of architectures achieved second place for the 2014 ImageNet Classification competition. Despite his

simple structure, it achieves competitive classification accuracy compared to more complicated nets such as GoogLeNet.

- **ResNet50** (Kaiming He) is a 50 layer Residual Network released in 2015 by Microsoft. The ResNet architecture obtained very successful results in the ImageNet and MS-COCO competition. The architecture of these models is based on residual connections that allows training of much deeper models with tens or even hundreds of layers.

8.3 Benchmarks

Our experiments rely on running 3 major benchmarks:

- The basic benchmark.
- The CIFAR-10 benchmark.
- The ImageNet benchmark.

These benchmarks rely on the standard datasets described in section 8.1. Each benchmark is described in detail in the following sections.

8.3.1 The basic benchmark [19] [20]

The benchmarks are based on the Convnet-benchmarks (https://github.com/soumith/convnet-benchmarks, n.d.) project. In it, proposed neural network implementations adopt a simplified approach: only computationally complex layers are retained (convolutional, fully connected), while others, such as activation functions and dropouts, are discarded. Collected measurements aim to assess the possibility of both inference and learning-based services, by tracking the throughput, in terms of images per second, of both forward and forward-backward steps.
The Convnet-benchmarks do not involve pre-processing or dataset loading: data representing an image batch is created in memory, with the relative distribution probabilities representing the "experience" used to train the network in a fully supervised approach to a problem of 1000 class image classification.

Site	Cluster Name	Frameworks
CINECA	Davide	TensorFlow 1.5 IBM Caffe 1.0.0 BVLC Caffe 1.0.0
CINECA	Galileo	TensorFlow 1.3.1 BVLC Caffe 1.0.0
CINECA	Marconi A2	TensorFlow 1.5 INTEL Caffe 1.0.0
CNRS/CC_IN2P3	K80 cluster	TensorFlow 1.3.0
CNRS/IDRIS	Ouessant	PowerAI/TensorFlow 1.2.1 PowerAI/IBM Caffe 1.0.0
EPCC	Urika-GX	TensorFlow 1.3.1 INTEL Caffe 1.0.0 BVLC Caffe 1.0
EPCC	Cirrus	TensorFlow 1.3.1 INTEL Caffe 1.0.0 BVLC Caffe 1.0

Table 2 - Basic benchmarks HPC resources

The benchmarks were run considering the following parameters:

- Code or framework used (TensorFlow or Caffe).
- Model trained (AlexNet, GoogLeNet, Overfeat and VGG11).
- Mode (Backward/Forward and Forward)

with a fixed number of iterations (100), on one device, and for a varying set of batch size as shown in Table 3. below.

Model	Batch size
AlexNet	256/512/1024/2048
GoogLeNet	64/128/256/512
Overfeat	128/256/512/1024
VGG11	64/128/256/512

Table 3 - Basic benchmark specificities

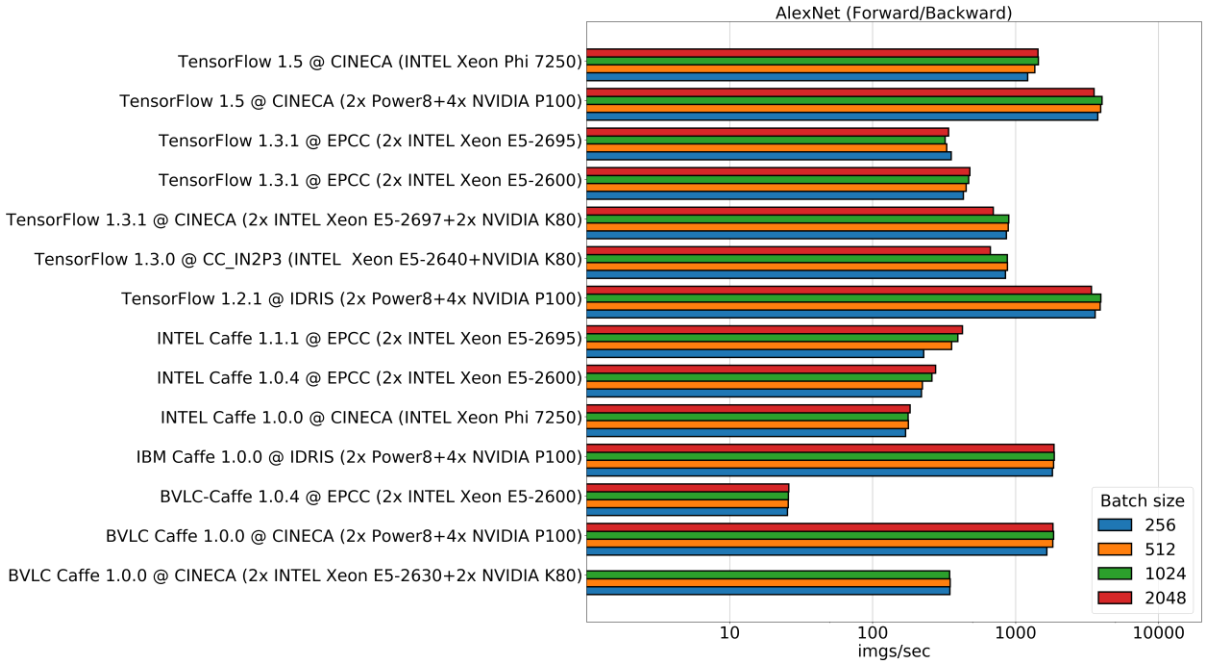

Figure 7 to Figure 14, show the performance measured in number of images per second (x axis) for different frameworks and architectures (y axis). Each figure represents the performance of a different network model for a different mode (forward/backward or forward). Due to the high variability of the results, we used a log scale for the x axis. Let us note that even if the single node possess more than one GPU, the experiments ran using only one GPU.

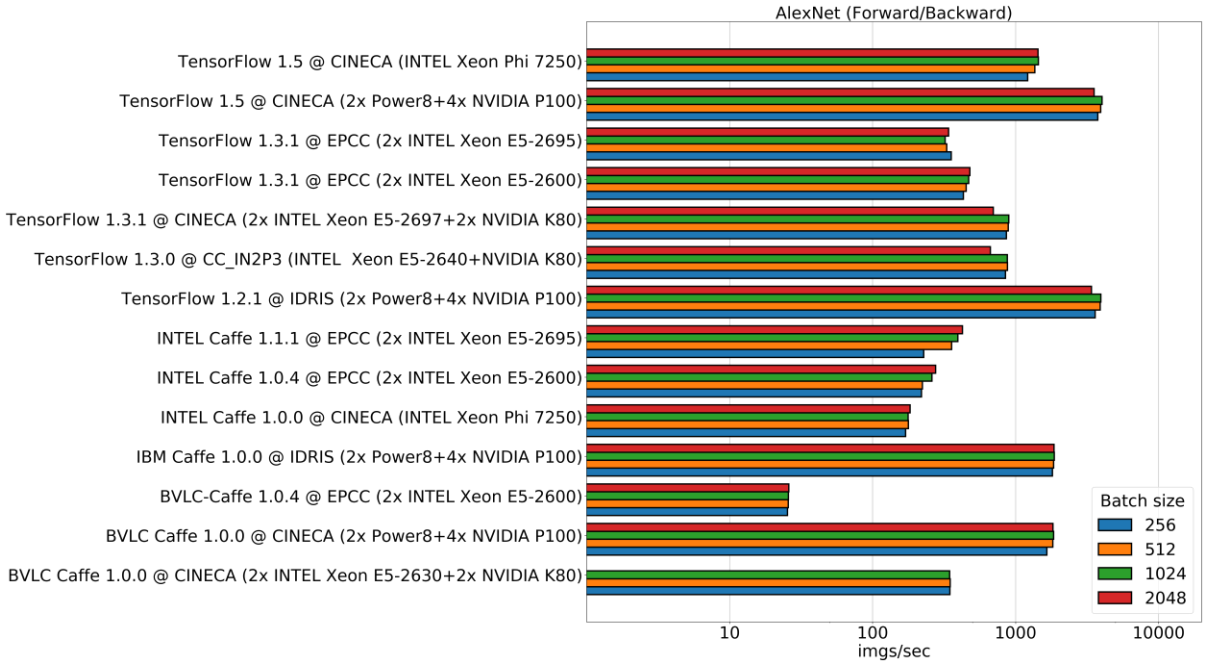

Figure 7: AlexNet Forward/Backward

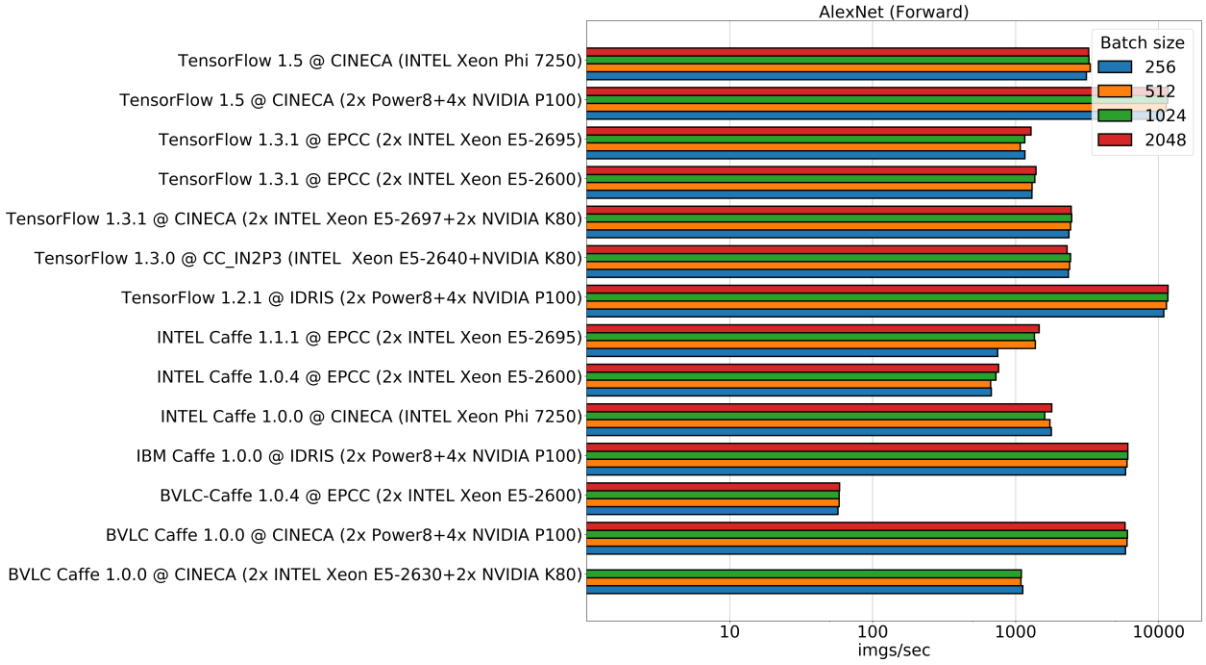

Figure 8: AlexNet Forward

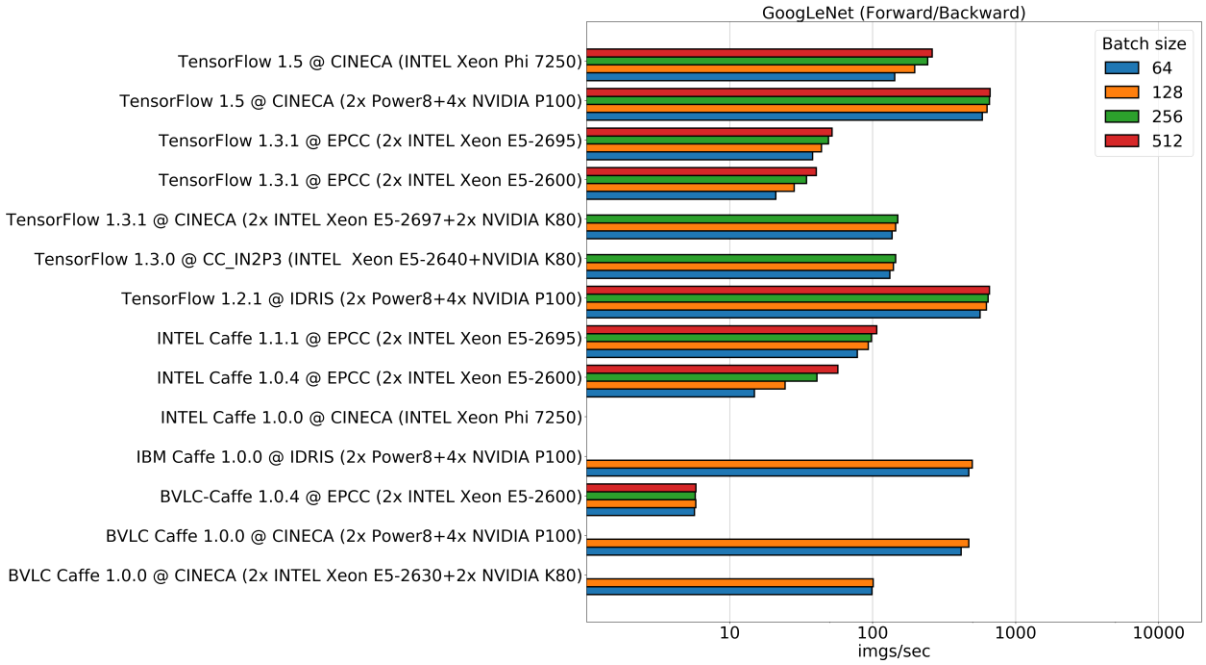

Figure 9: GoogLeNet Forward/Backward

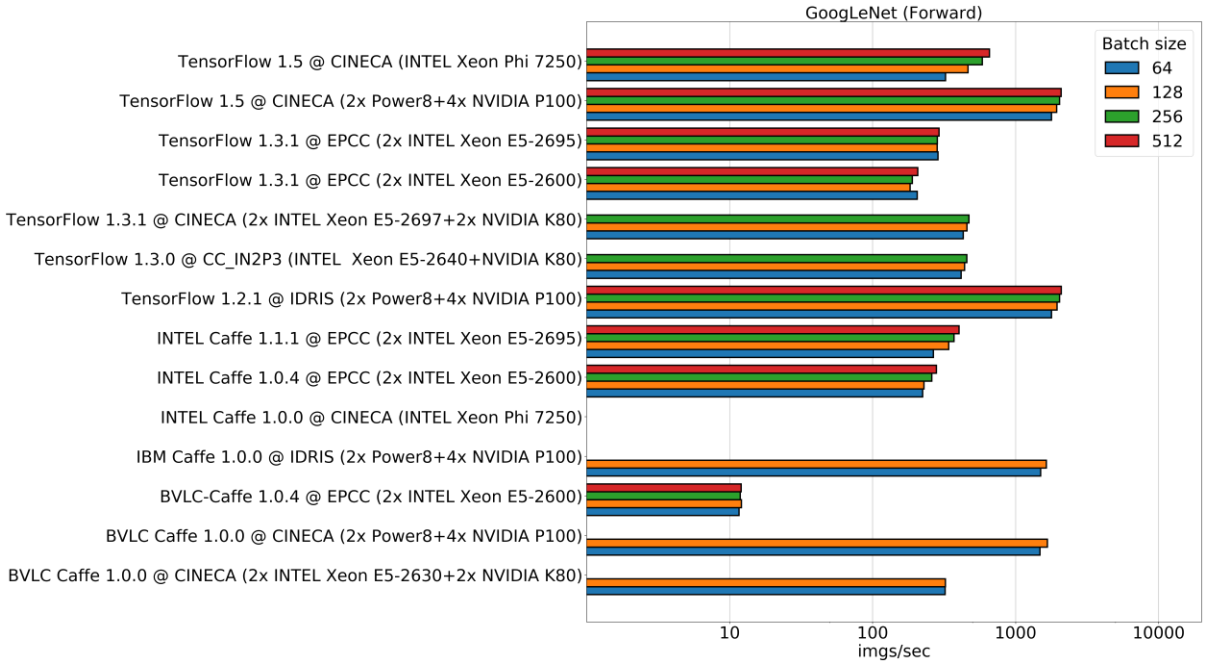

Figure 10: GoogLeNet Forward

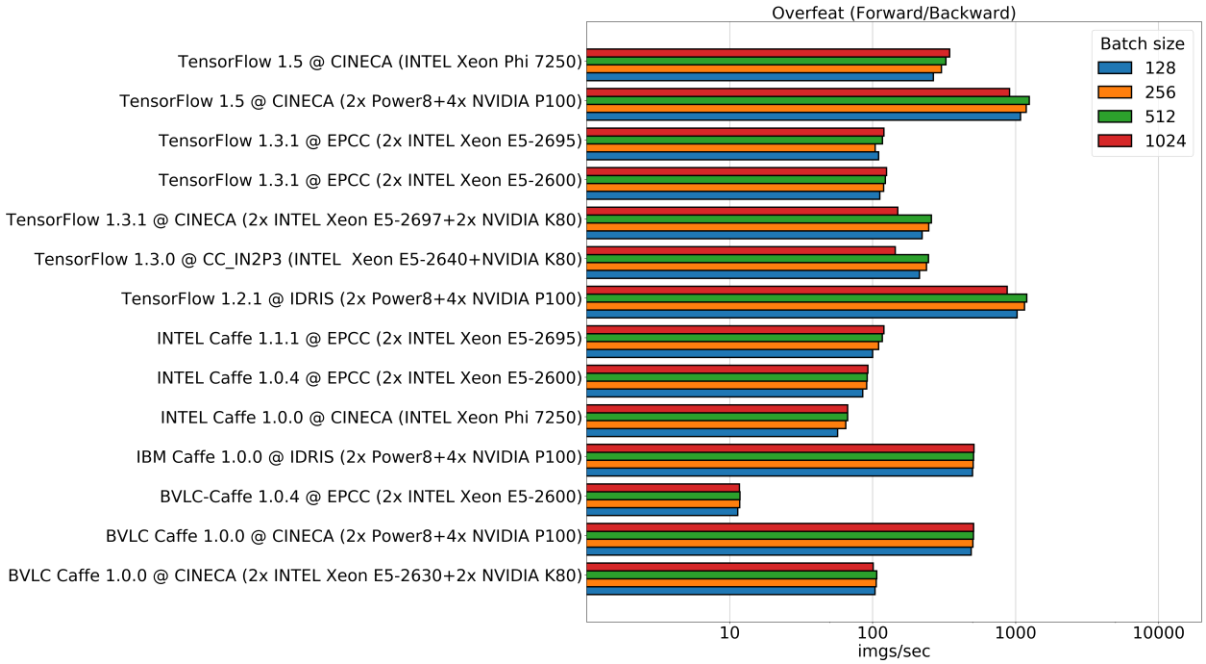

Figure 11: Overfeat Forward/Backward

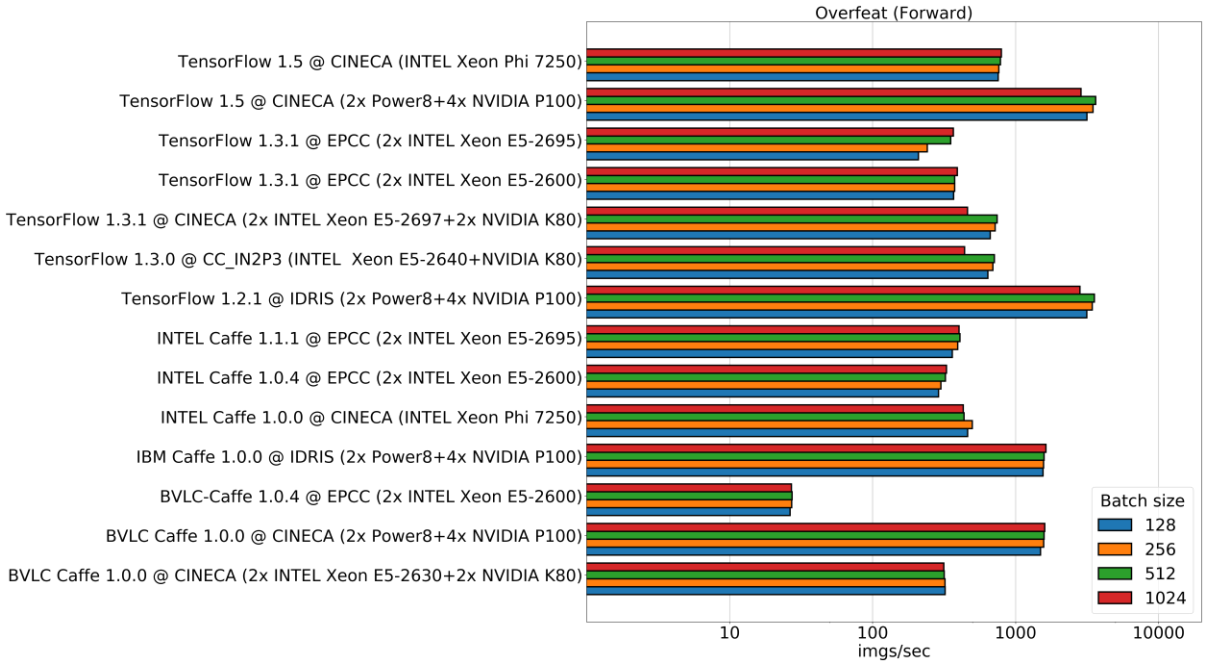

Figure 12: Overfeat Forward

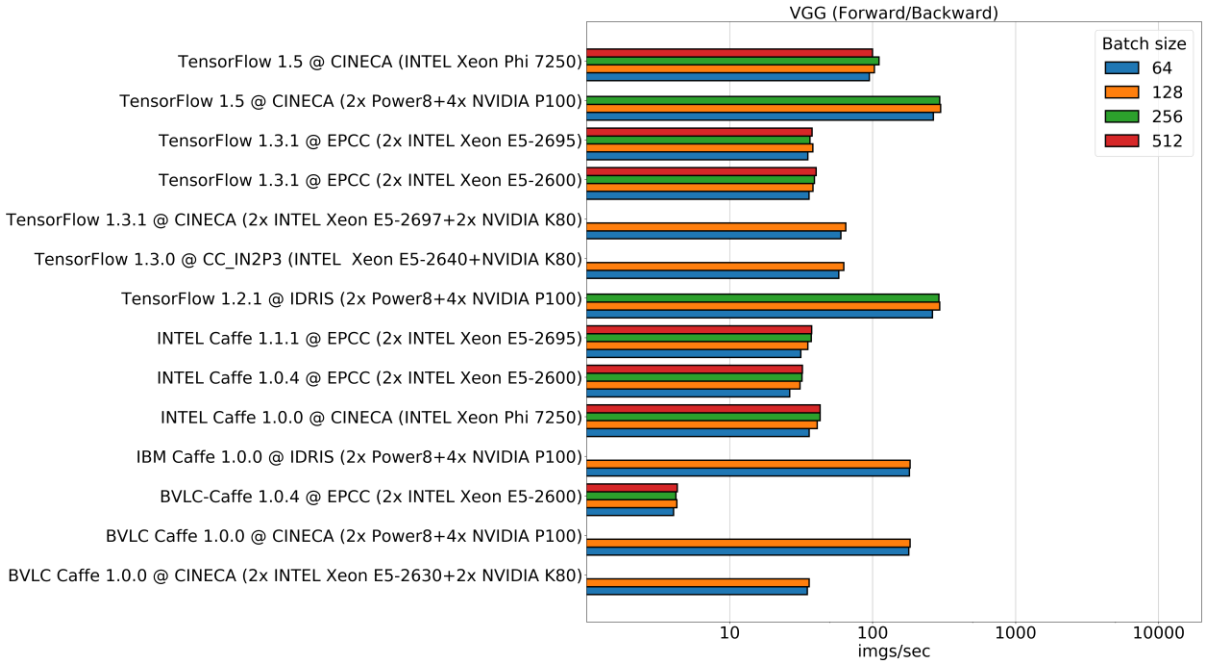

Figure 13: VGG Forward/Backward

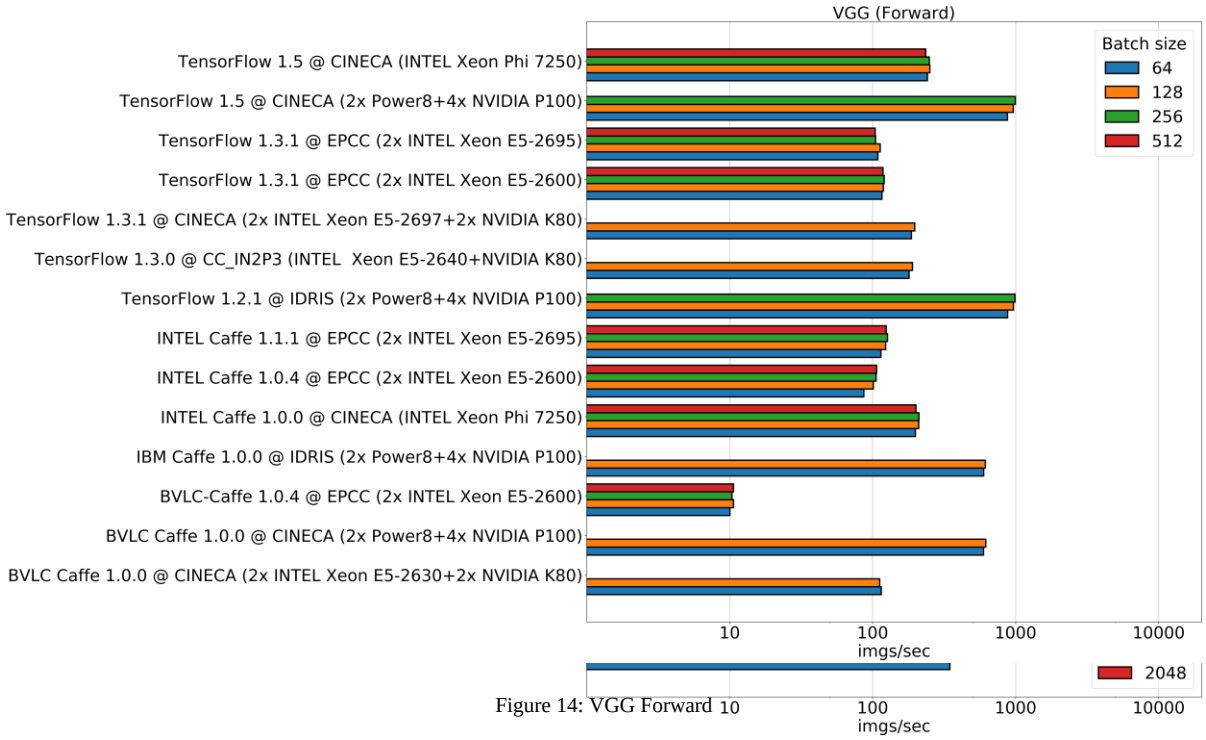

Figure 14: VGG Forward

Figure 7 to Figure 14 show that the performance levels depend mostly on the hardware configuration: the GPU NVIDIA P100 gets the best result running the models with TensorFlow, followed by the performance obtained with Caffe either the BVLC or IBM release. TensorFlow running with the INTEL Xeon Phi 7250 follows, doing better than the GPU NVIDIA K80.

8.3.2 The CIFAR10 benchmark [22]

The second benchmark we ran is a benchmark based on the CIFAR-10 dataset, and comes from the Keras CIFAR-10 example (https://github.com/keras-team/keras/blob/master/examples/cifar10_cnn.py, n.d.). Unlike the basic benchmark described in the previous section, this training is composed of the following steps:

- Dataset load in memory.
- Run the training for a given number of epochs.
- Run a validation phase to check the loss and accuracy between each epoch, in order to follow the training progress.
- Save the weights.
- Save the Tensorboard logs

Site	Cluster Name	Frameworks	Libraries
CINECA	Davide	TensorFlow 1.8	Cuda 9.2, Cudnn 7.1, Keras
CNRS/CC_IN2P3	K80 cluster	TensorFlow 1.8	Cuda 9.2, Cudnn 7.0.5, Keras 2.2.0
CNRS/IDRIS	Ouessant	PowerAI/TensorFlow 1.8 TensorFlow 1.8	Cuda 8.0, Cudnn 6, Keras 2.1.2 Cuda 9.2, Cudnn 7, Keras 2.2.0

Table 4 – CIFAR10 HPC resources

Dataset	CIFAR-10
Model	Simple CNN, 12 layers
Batch size	64, 128, 256, 512, 1024, 2048

Table 4 - CIFAR-10 benchmark specificities

We present the CIFAR-10 benchmark results thereafter.

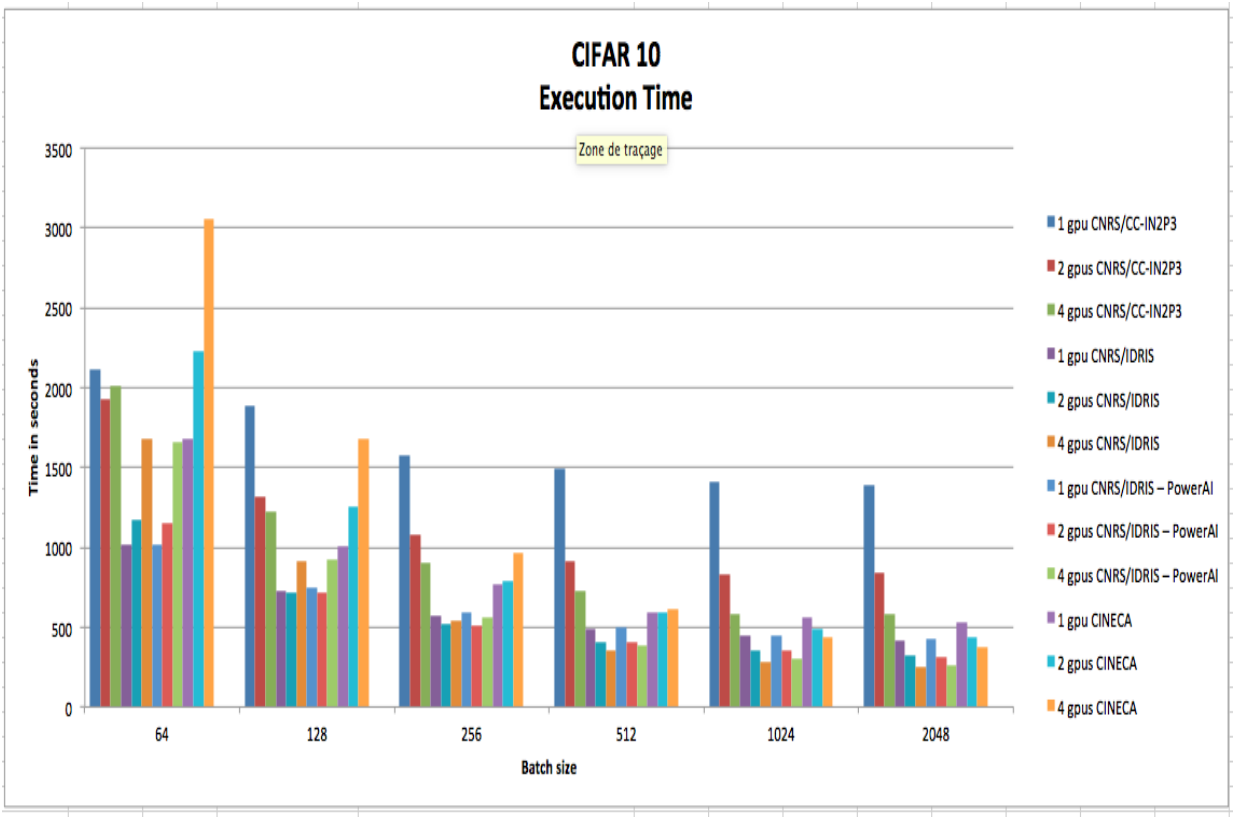

Figure 15 - CIFAR-10 Execution time. Performance of TensorFlow (X-axis: batch size, Y-axis: execution time (seconds) for the different architectures.

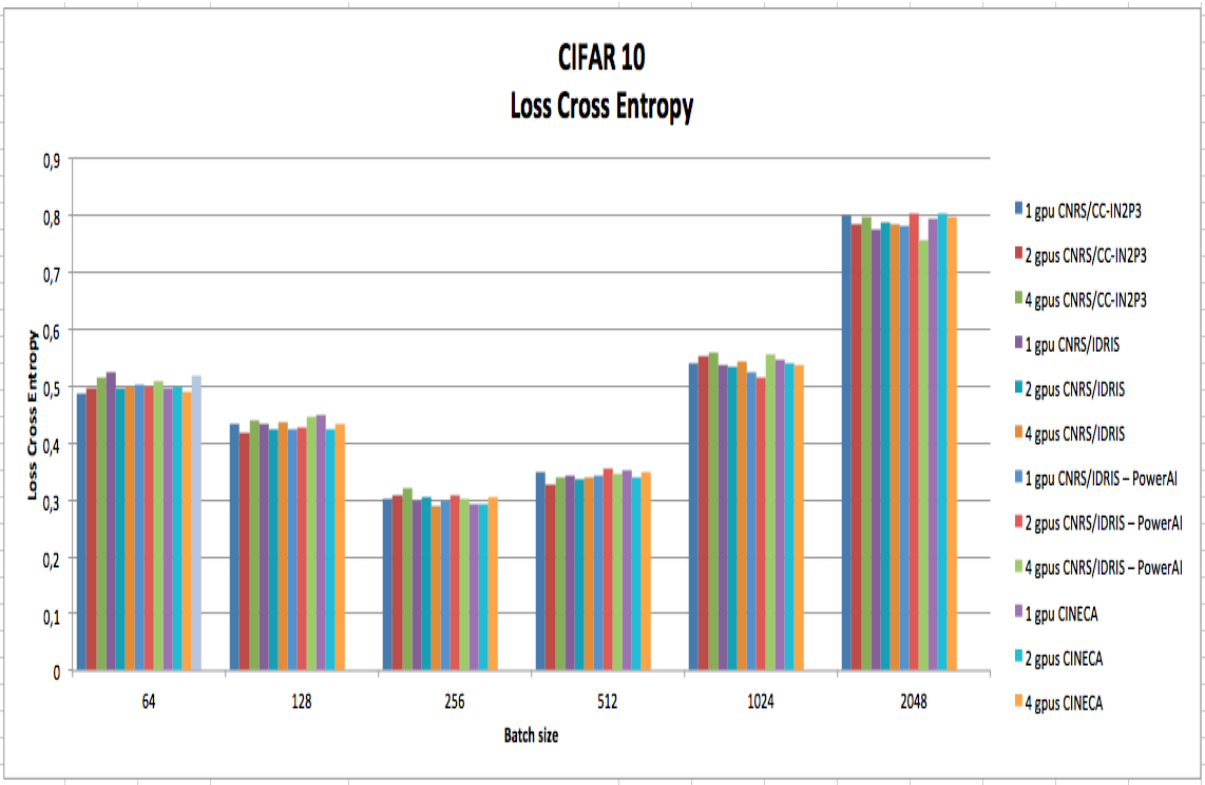

Figure 16 - CIFAR-10 Loss cross entropy. Performance of TensorFlow (X-axis: batch size, Y-axis: loss cross entropy for the different architectures.

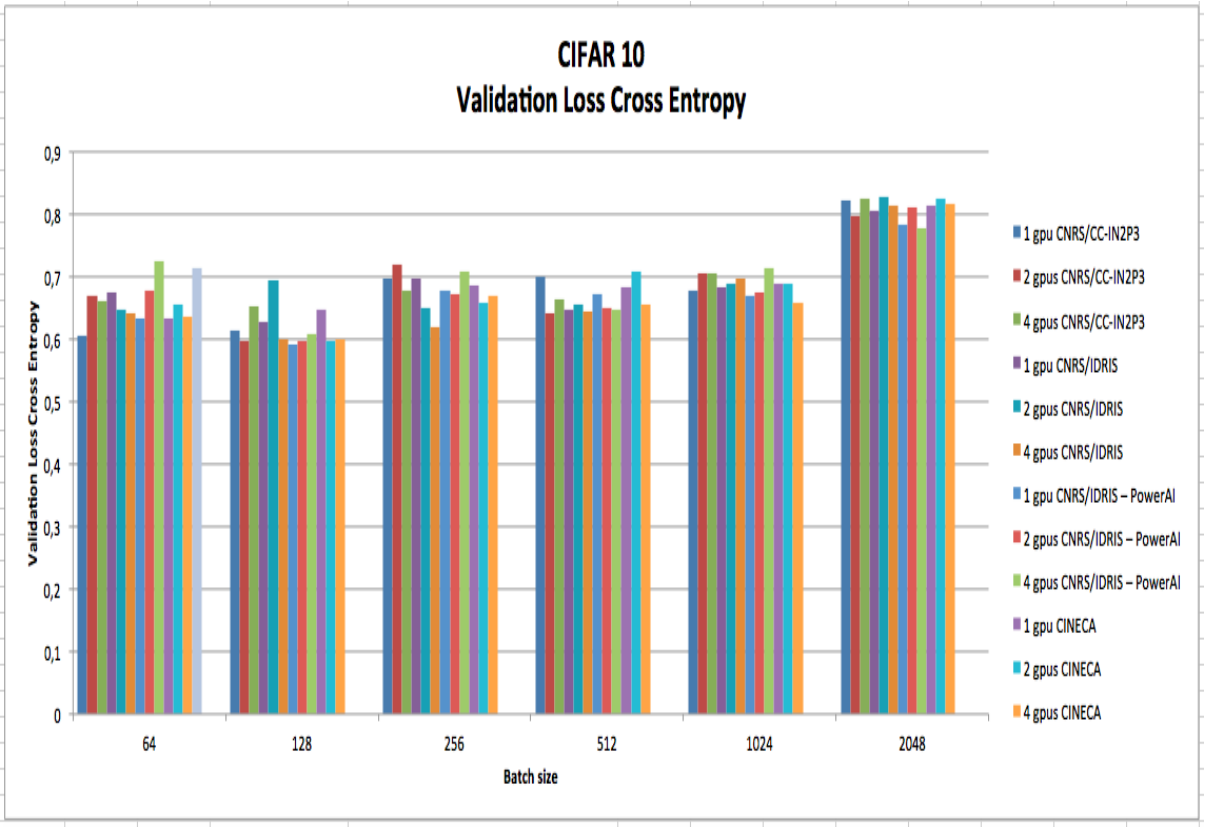

Figure 17 - CIFAR-10 Validation loss cross entropy. Performance of TensorFlow (X-axis: batch size, Y-axis: validation loss cross entropy for the different architectures.

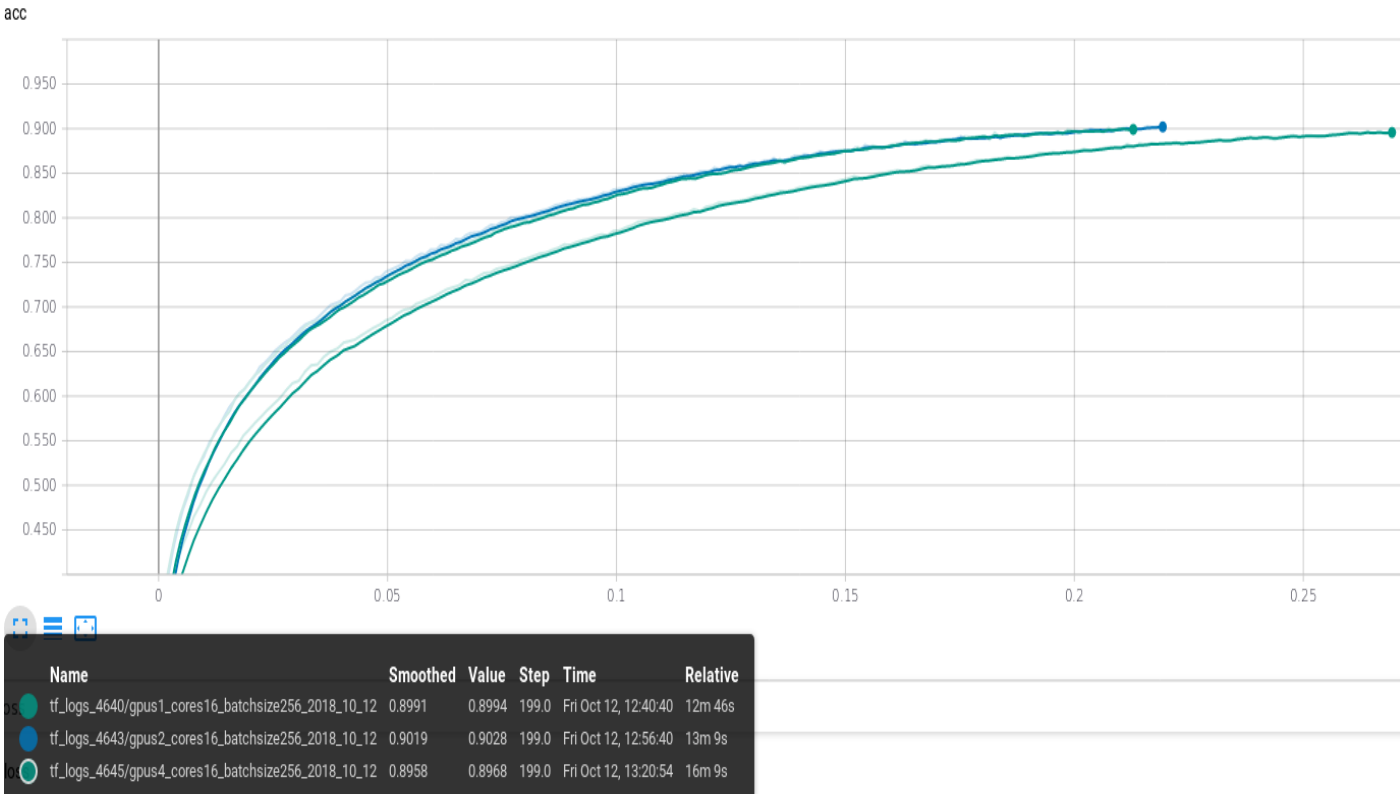

Figure 18 – Batch size 256 for 1, 2 and 4 gpus [CINECA] - relative time (x axis) / accuracy (y axis)

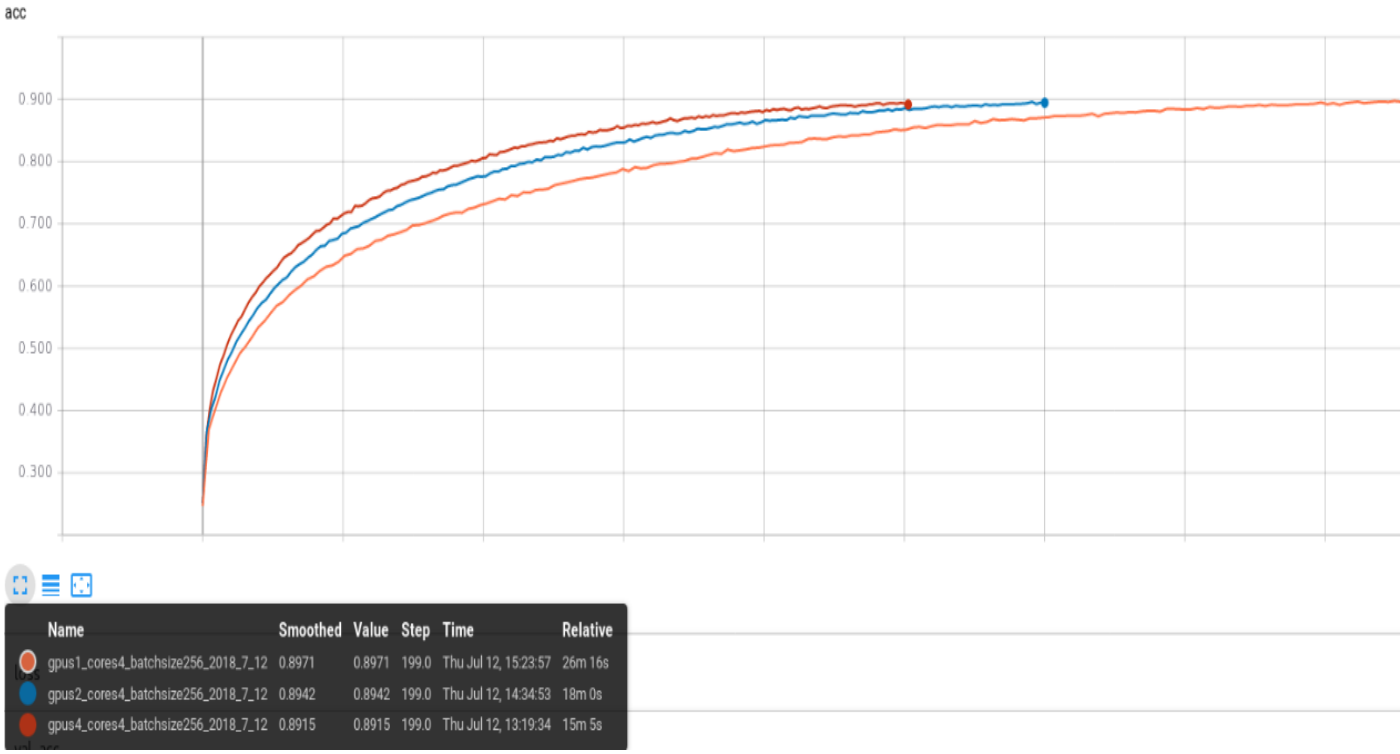

Figure 19 – Batch size 256 for 1,2 and 4 gpus [CNRS/CC-IN2P3] - relative time (x axis) / accuracy (y axis)

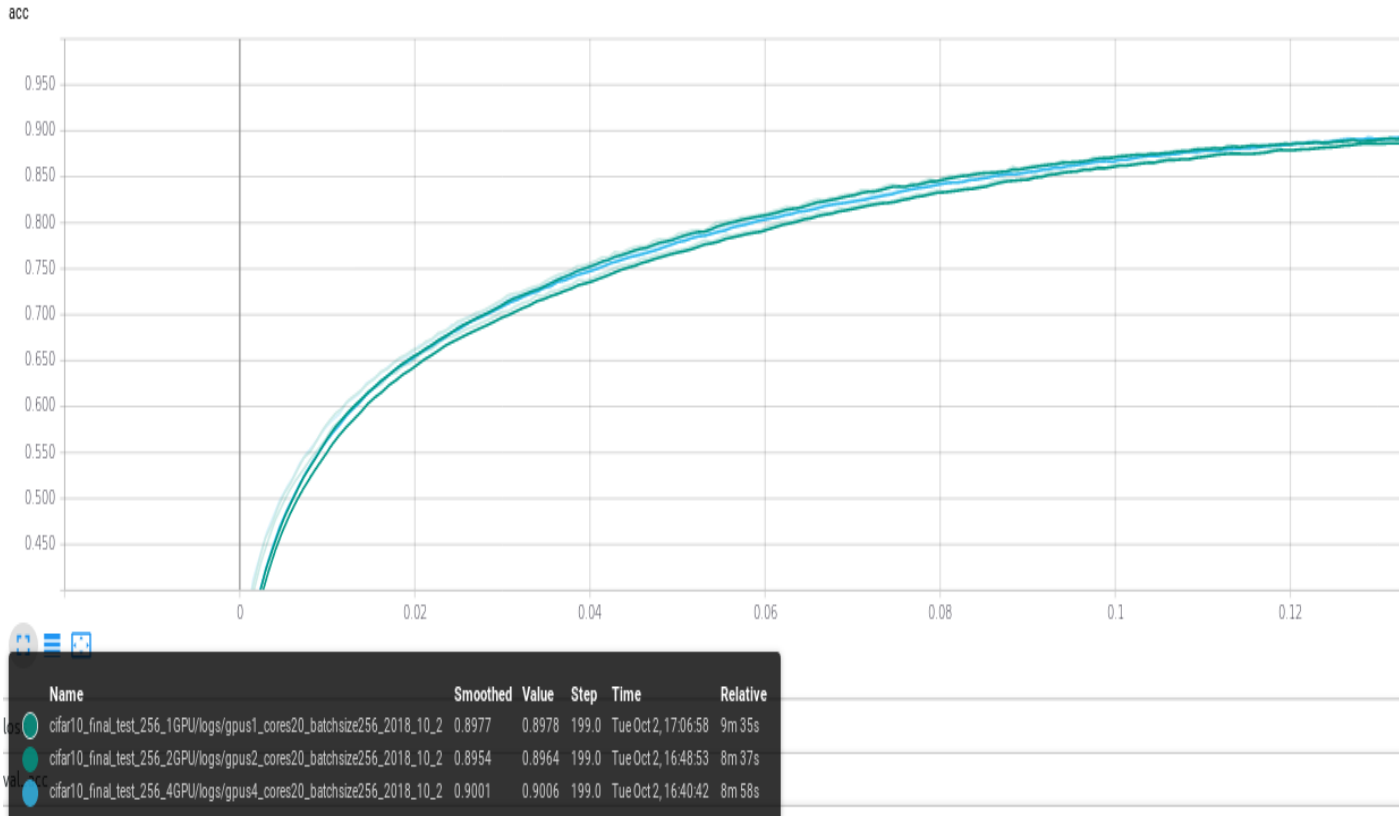

Figure 20 - Batch size 256 for 1,2 and 4 gpus [CNRS/IDRIS] - relative time (x axis) / accuracy (y axis)

As for the basic benchmarks described previously, this benchmark shows again that the performances depend mostly on the hardware, i.e the GPU NVIDIA P100 (IDRIS and CINECA) outperform the GPU NVIDIA K80 (CC-IN2P3).

The results show also the huge impact of the batch sizes with small batch sizes producing poor execution time performance, when compared to large batch sizes. One can notice that increasing the number of P100 GPUs is not effective and can be even disadvantageous in the case of small batch sizes, due to the penalty introduced by the high number of memory transfers and the device to device communication.

Indeed, there's a tradeoff in the granularity of the learning process, as bigger chunks digested by the model lead to a faster training, but smaller batch sizes provide smaller steps in the process of finding the global minimum, and therefore a potential higher accuracy, although smaller steps can lead the search problem to get stuck in a local minimum as well. The most appropriate batch size is always problem dependent. Nevertheless, in general a model trained with smaller batch sizes per device tends to provide better results, with higher validation accuracies and a lower loss. For the CIFAR10 benchmark, the lowest validation cross entropy is obtained with a batch size of 128, outperforming all the bigger batch size configurations.

Unlike the P100, the K80 shows a significant performance gain between one and two GPUs whereas the gain is less clear with four GPUs that shows a slight performance, but far from linear with the number of GPUs. One can think that it can be due to the K80 architecture composed of two cards each of them having 2 GPUs which can introduce a communication bottleneck but this hypothesis is not confirmed by the ImageNet benchmark.

The CIFAR10 benchmark shows that there were no observable differences in performance when running the benchmarks from within a container or not i.e using the PowerAI framework or a compiled Tensorflow release at IDRIS.

Modifying the batch size impacts the accuracy. The best training score is reached using a batch size of 256, independent of the GPU type.

8.3.3 The ImageNet benchmark [24]

The third benchmark we ran is a benchmark based on the ImageNet dataset. Here are the ImageNet benchmark specifications:

Site	Cluster Name	Frameworks	Libraries
CINECA	Davide	1- TensorFlow 1.8, Horovod 0.14.1 2- TensorFlow 1.10, Horovod 0.13.11	CUDA 9.0, CuDNN 7.0
CNRS/CC_IN2P3	K80 cluster	TensorFlow 1.8/Singularity	CUDA 9.0, CuDNN 7.0
CNRS/IDRIS	Ouessant	1- TensorFlow 1.8, Horovod 0.13.3, Caffe 1.0 2- PowerAI/TensorFlow 1.8, Power/AI Caffe 1.0	CUDA 9.2, CuDNN 7.1

Table 5 - ImageNet HPC resources

Dataset	ImageNet
Model	VGG-19, ResNet50
Batch size	32,64,128

Table 6 - ImageNet benchmark specificities

The choice of the batch sizes to test is as well much related to particular characteristics of the problem. The pictures of the ILSVRC2012 are bigger than those belonging to the CIFAR10. This does not only have implications only on the size of the input dataset, but also on the size of the model itself, as more and bigger convolution filters are needed to extract the features. The model has to fit in the memory of the device (the GPU in this case), and as usual the batch size of choice has an impact on the learning progress. Here we used 32, 64 and 128 as batch sizes, as a good trade-off between learning progress (the smaller the better), execution time (the bigger the faster) and GPU memory usage (the bigger the better but without exceeding the GPU memory limits). The following plots show the TensorFlow results. Figure 21 shows the top validation accuracies obtained after 90 epochs of training (Y-axis) for different numbers of GPUs and batch sizes (X-axis). We didn't plot the Caffe results, as they are very poor when compared to the TensorFlow ones. Indeed, the Caffe performance appear to be between 4 and 5 times worse for 1 and 2 GPUs, between 10 and 12 times worse for 4 GPUs and the BVLC Caffe doesn't include the distributed training feature.

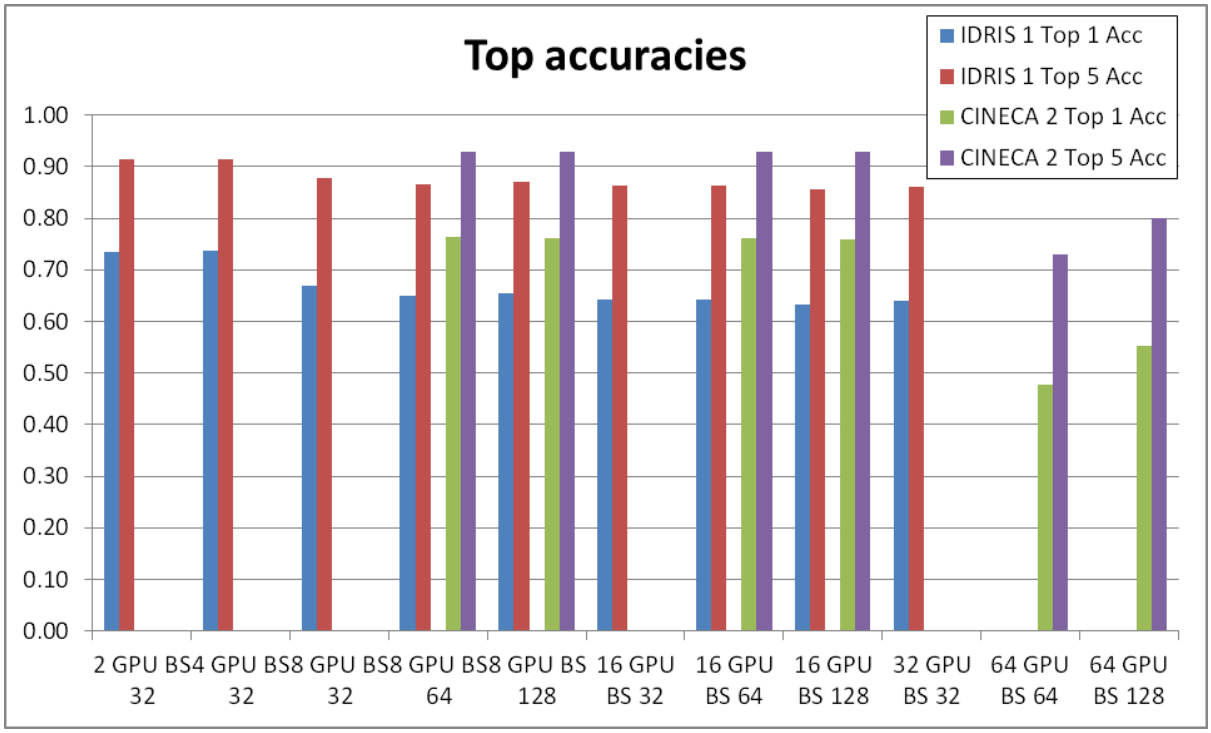

Figure 21 - Top validation accuracies (90 epochs) using basic hyper parameters (no tuning performed)

We succeeded in training this model with very good results and very good performance. The experiment shows that the increase in the number of GPUs and in batch size has an impact on the model accuracy. The more devices employed, and the bigger the batch size, the lower the validation accuracy. This might indicate that the degree of parallelism and the batch processing handicap to some extent the learning process.

Figure 22 shows the total time required for training 90 epochs (Y-axis) again for different numbers of GPUs and batch sizes tested (X-axis). Figures 23 and 24 show the model bandwidth (i.e the aggregated number of processed images per second (Y-axis) for different numbers of GPUs and batch sizes tested (X-axis). Figure 25 shows single-node experiments. Figure 26 includes multi-node experiments.

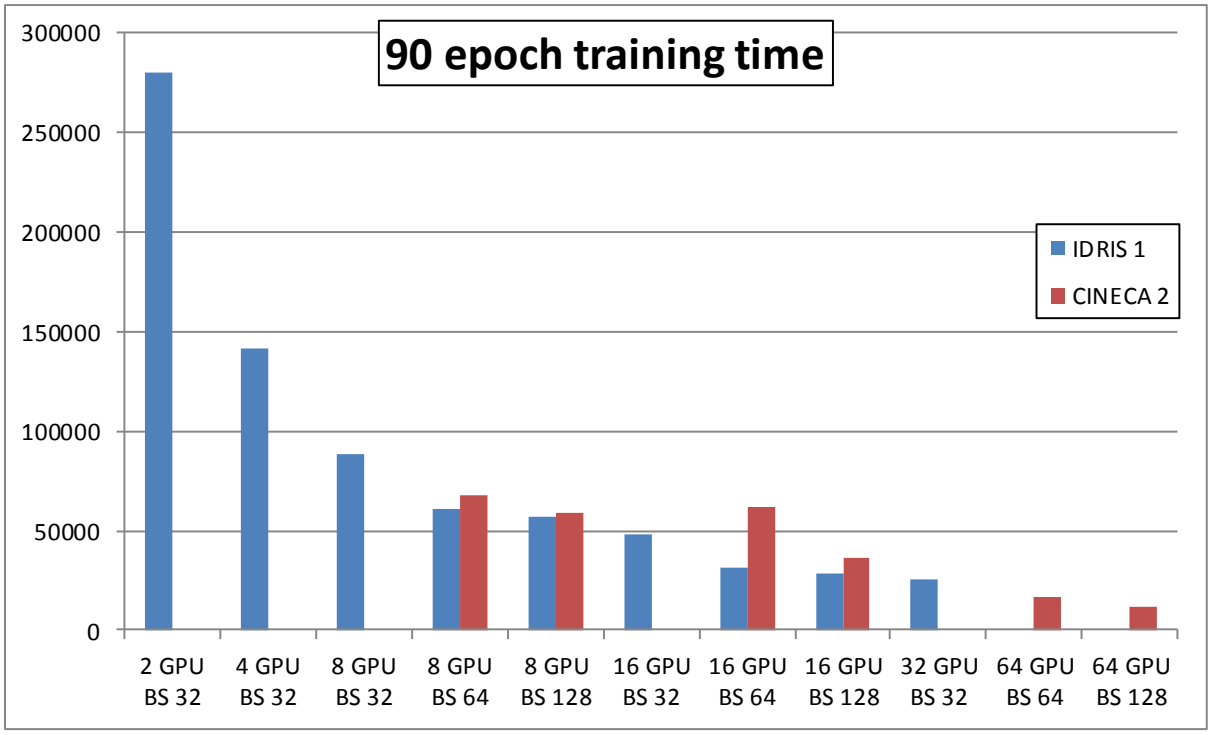

Figure 22 - Execution time (90 epochs)

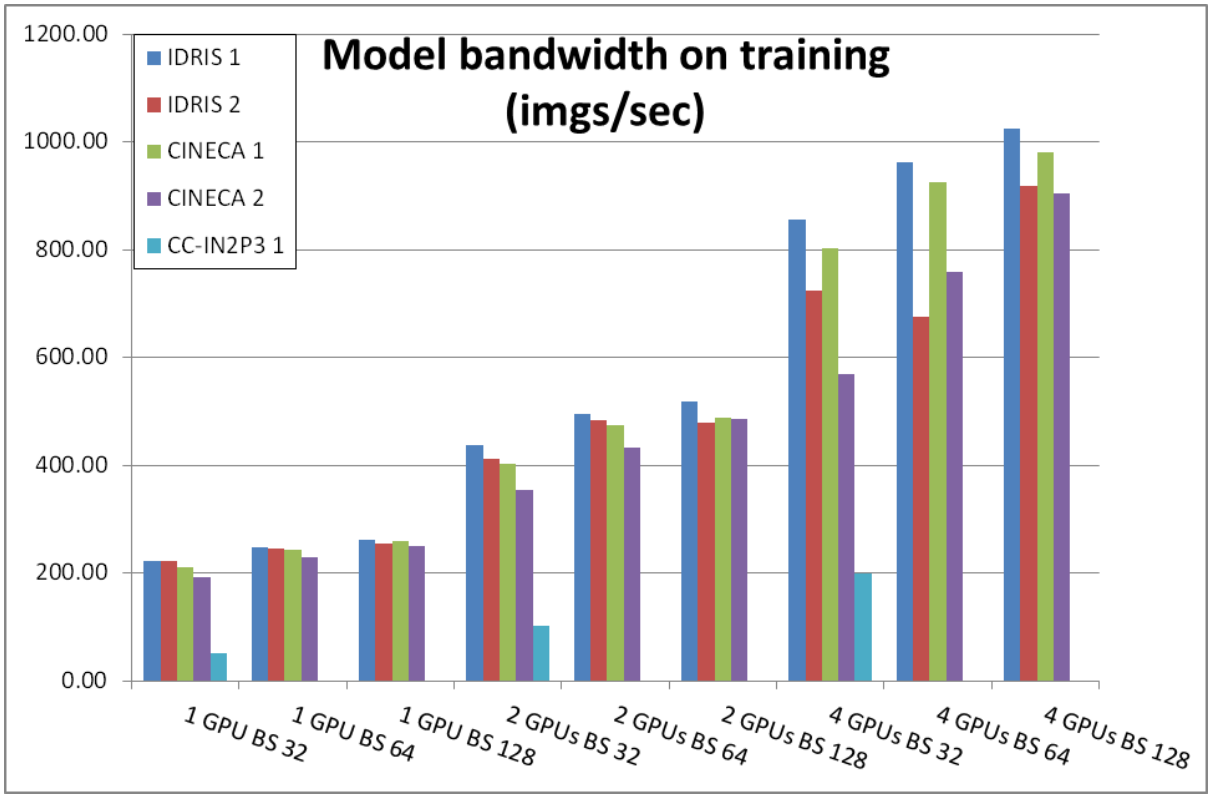

Figure 23 – Intra-node: TensorFlow bandwidth in number of images per second (y axis) for different batch sizes and GPUs (x axis).

One more time, these experiments show that the most significant factor is the hardware, i.e the GPU NVIDIA P100 (IDRIS and CINECA) outperform the GPU NVIDIA K80 (CC-IN2P3).

Bandwidth increases with the number of GPUs and batch sizes, which lets us perform faster training. However, as larger batch sizes may handicap the learning process, lowering the accuracy and incrementing the validation error, a tradeoff must be achieved between accuracy and performance in the training process.

The NVIDIA K80 runs Out Of Memory, as the model became too large for batch sizes of 64 and 128.

Figure 23 shows as well the impact produced by the use of containers , in this case a PowerAI docker image, versus the traditional way of deploying software i.e astandalone release of Tensorflow compiled at IDRIS. The series IDRIS 1 represents the latter approach, whereas the series IDRIS 2 represents the former based on containers. As it can be seen, containers introduce a slight overhead in performance which tends to increase with the number of GPUs and larger batch sizes (around 5% overhead on the total execution time).

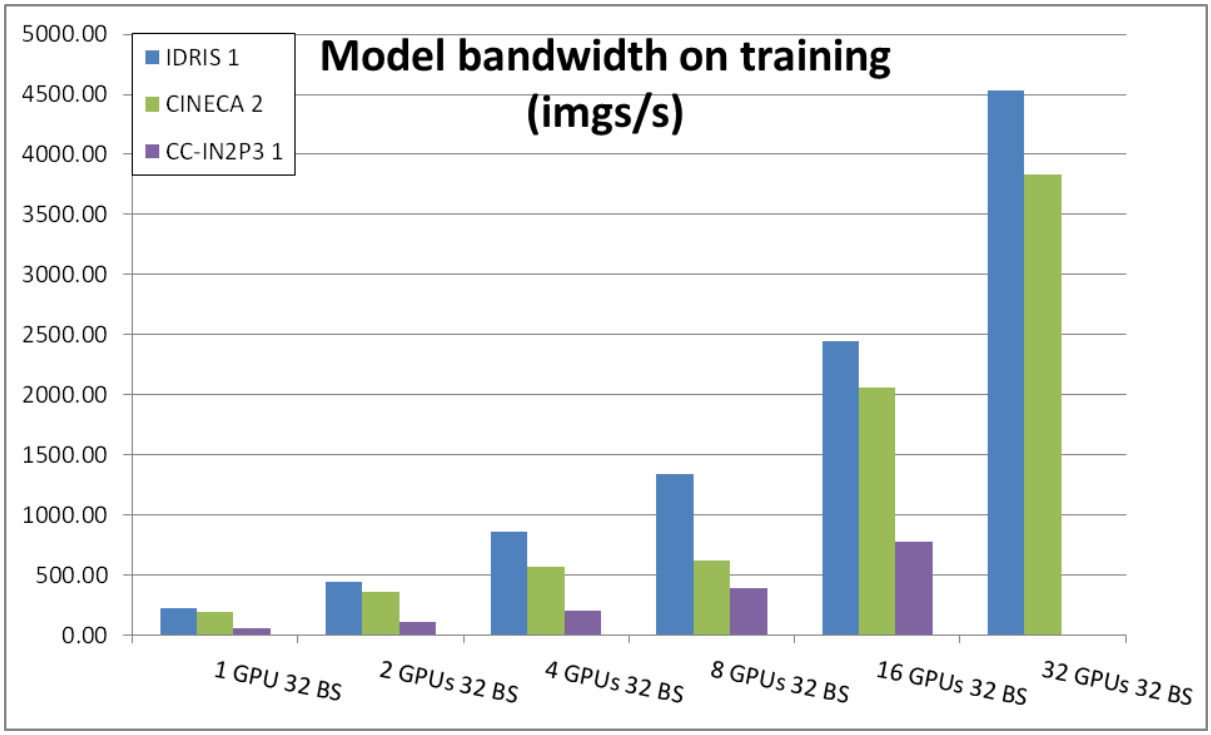

Figure 24 - Multi-node: TensorFlow bandwidth in number of imager second (y axis) for different batch sizes and GPUs (x axis)

For the NVIDIA P100, the plot shows a significant performance gain while increasing the number of GPUs from 1 to 32, but far from linear with the number of GPUs. The performance of the NVIDIA K80 is well below with a very limited gain from 1 to 16 GPUs.

We also studied the efficiency aspect of this benchmark. The results are reported below. Figure 27 and 28 describe the efficiency of TensorFlow multi-nodes experiments. The efficiency (η) is a metric of the improved utilization of the resources from a sequential to a parallel system defined as:

$$\eta = \frac{S}{A}$$

Where S is the speedup of the experiment with regard to the secuential experiment in this case, experiment for 1 GPU, and A is the execution capacity, i.e. the number of parallel elements in the architecture, in this case number of GPUs in the parallel experiments.

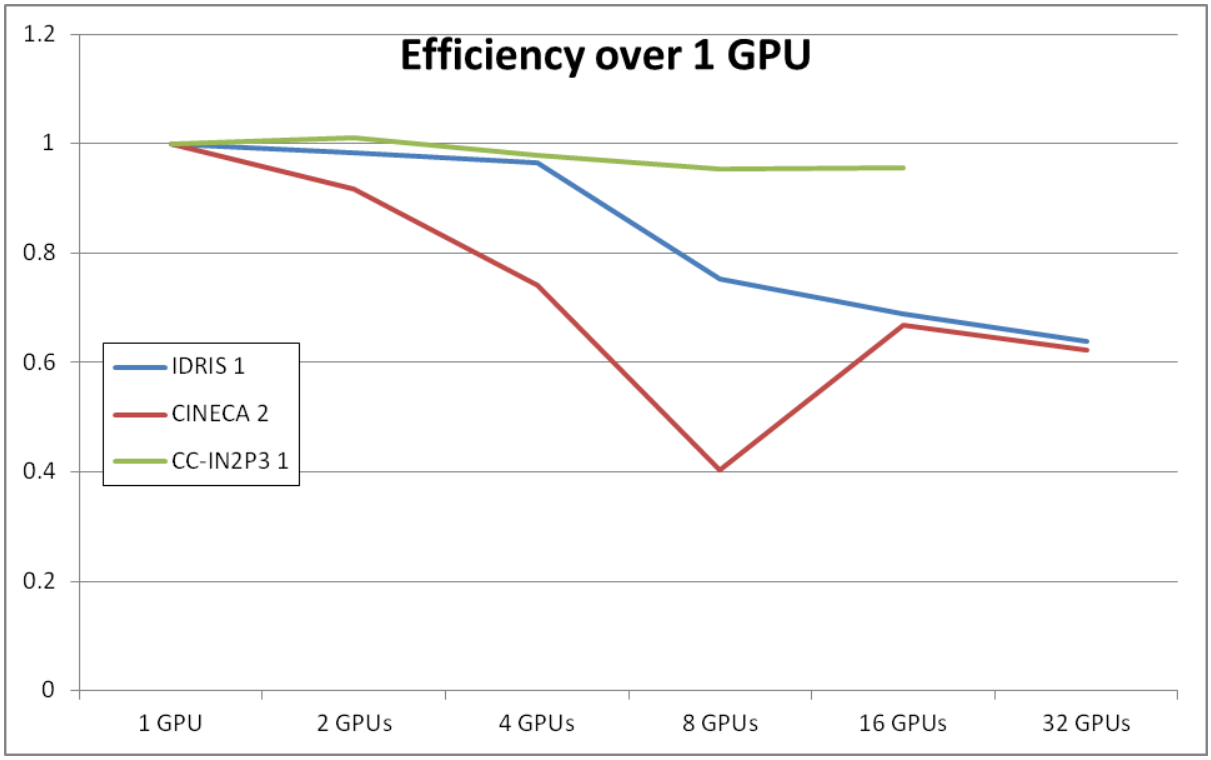

Figure 25 - Parallel efficiency over 1 GPU up to 32 GPUs (batch size 32)

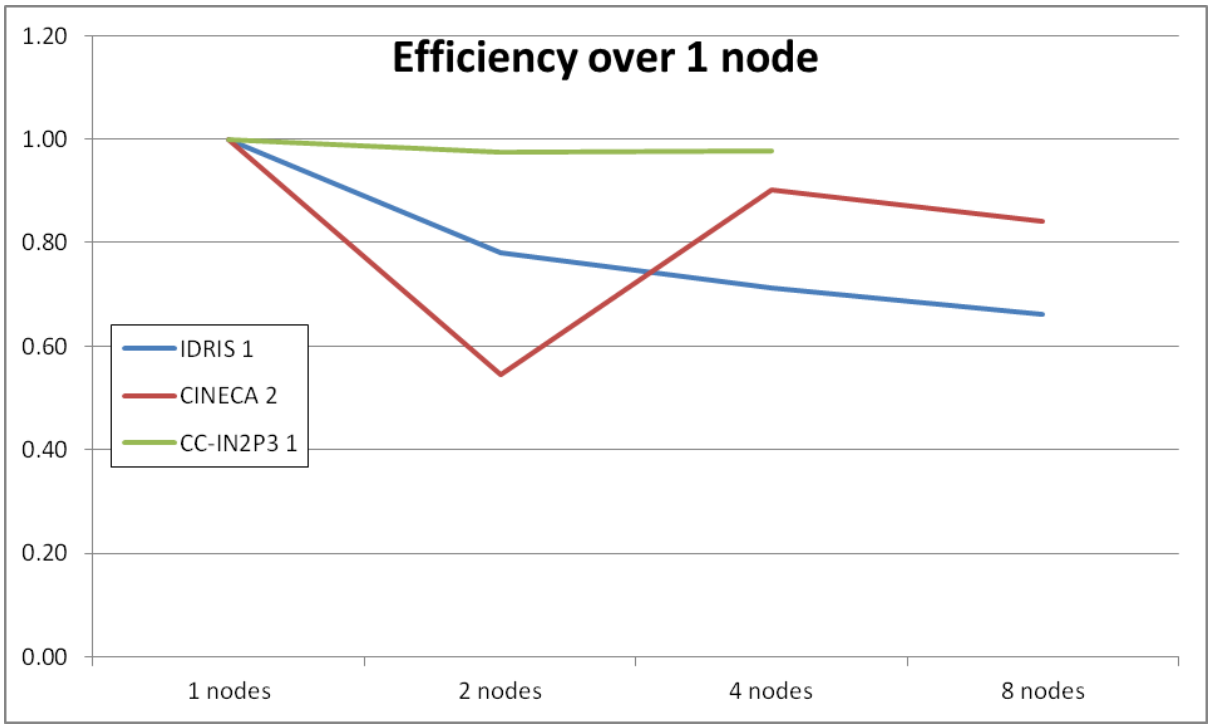

Figure 26 - Parallel efficiency over 1 node up to 8 nodes (batch size 32). 4 GPUs per node for IDRIS and CINECA, and 4 GPUs per node for CC-IN2P3

The plots show the very good efficiency of the NVIDIA K80 at CC-IN2P3 when increasing the number of GPUs and nodes. The results are worse with the NVIDIA P100 at IDRIS and at CINECA. Whereas IDRIS shows a good efficiency in an intra node environment, the efficiency at CINECA shows an important loss that can be related to memory transfers. In a multi-nodes environment, both IDRIS and CINECA infrastructures show an important loss. Further experimentation is needed in order to find the origin of this performance loss: if there's a particular

problem with IDRIS and CINECA infrastructures, if it's a suboptimal configuration, or if it's a problem related to the algorithm or framework used.

8.4 Real Use Cases

In the previous chapters, we described the benchmarks we used. From the simpler ones to larger ones, which helps us to gain expertise on the various architectures. However, the deployment of a Data Analytics service requires to get closer to real use cases in order to better understand users' needs and requirements, and to deploy the most appropriate tools to meet their expectations. In order to reach this goal, we launched a "Call for prototypes for Deep Learning, Machine Learning and Data Analytics" (Agnès Ansari C. , Call For Prototypes - https://bscw.zam.kfa-juelich.de/bscw/bscw.cgi/d2599979/CallForPrototypes.doc) in April-May 2018 within the PRACE community, in order to obtain the necessary information to implement these use cases on the PRACE architectures.

We got two answers in response to this call from the scientific communities that are described in the following sections. We gather this information in a "*Use cases master"* document (Agnès Ansari C.) which gives an overview of each use case with its major characteristics. Table 8 below depicts the use cases master document.

Both use cases were related to deep learning. The first use case comes from CNRS/LAL (High Energy Physics) and CC-IN2P3 and is related to Astrophysics and is about Galaxy deblending. The second one comes from the CERN (High Energy Physics), it is related to data certification for the RPS subsystem of the CMS experiment. These use cases are described in the following chapters. Afterwards we present the results we obtained on the PRACE architectures.

Use case	Users	Reference	Characteristics
Machine learning for data certification for RPC subsystem of the CMS experiment at CERN **Execution plan:** The plan is to train the network using the PRACE infrastructure. Then, the network will be used at CERN to classify the data.	Double affiliations: CERN (CERN users) and Sofia University or Institute of Information and Communications Technologies • CERN – CMS experiment • NCSA (National Center for Supercomputer Applications) Bulgaria	https://bscw.zam.kfa-juelich.de/bscw/bscw.cgi/d2640648/PRACE_ML_v3.pdf https://bscw.zam.kfa-juelich.de/bscw/bscw.cgi/d2681218/Muon_ML_PRACE.docx	**Dataset:** Total Size: 5000 to 10000 runs File type: ROOT Prace git tag: no Training dataset size: unknown Validation dataset size: unknown Test dataset size: unknown Sharing and licenses:no Small dataset availability: no **Neural Network:** Type: Autoencoder Number of layers: 7 to 9 (can change) Number of inputs: 7 to 60/histogram – The histograms (2D) are supplied bin by bin as an input. NN previously runs at another place: no information licences: no **Source code status:** Software in a developing phase Source code availability: no **Language:** Python release 2 and 3 **Frameworks/libraries:** Tensorflow, Keras **Storage requirements:** From (5000 files * 250MB) 1.25 TB to (10000 files*250MB) 2.5 TB **Input data: (from CERN to PRACE infrastructure)** The DQM output which is one file/run DQM file size: 250 MB which includes a set of 2D histograms **Ouput data: (from PRACE infrastructure to CERN)** Negligible data size: the network parameters **Possible data transfer tool:** xrootd and grid credentials, xrdcp, sftp or gsftp **Number of user accounts needed:** between 3 and 5 **Community related softwares:** ROOT, PyROOT (tools used in the HEP community to read the files). Installed by users.

			Benefit from using the PRACE infrastructure: GPUs **Project start date:** Sept 2018 **Project duration:** 7 months

Table 7 - The Use cases master document. Example of the CERN RPC/CMS use case

8.4.1 The Astrophysics V1 use case [27]

This benchmark comes from a deep learning challenge. It uses 20 000 images (128x128 pixels, 32 bits) made of two overlapping galaxies (1st image created), referred to as blended. These galaxies have been extracted from real images from the Hubble Space Telescope, and combined manually to create the blends. The goal of the challenge is to train a model that can automatically detect the contiguous region (mask) where the light of the two galaxies overlap (2nd image created). The model predicts a probability for each pixel to belong to such region, a threshold is then applied to the probabilistic image (3rd image created) to obtain an actual prediction (4th image created).

- Dataset load in memory
- Run the training for a given number of epochs
- Run a validation phase to check the validation loss and accuracy between each epoch in order to follow the training progress
- Save the weights
- Save the Tensorboard logs
- Save the result images

Site	Cluster Name	Frameworks	Libraries
CINECA	Davide	TensorFlow 1.10	Cuda 9.2, Cudnn 7.1, Keras
CNRS/CC_IN2P3	K80 cluster	TensorFlow 1.8	Cuda 9.2, Cudnn 7.0.5, Keras 2.2.0
CNRS/IDRIS	Ouessant	TensorFlow 1.10 PowerAI/TensorFlow 1.8	Cuda 9.2, Cudnn 7, Keras 2.2.0

Table 8 - Astrophysics V1 HPC resources

Dataset	The dataset (2 Gb) is divided into 12 000 images for the training, 4 000 for the validation (during training) and 4 000 images for the final evaluation of the model.
Model	UNet (see Section 8.2)
Batch size	64, 128, 256, 521, 1024, 2048

Table 9 - The Astrophysics V1 use case specificities

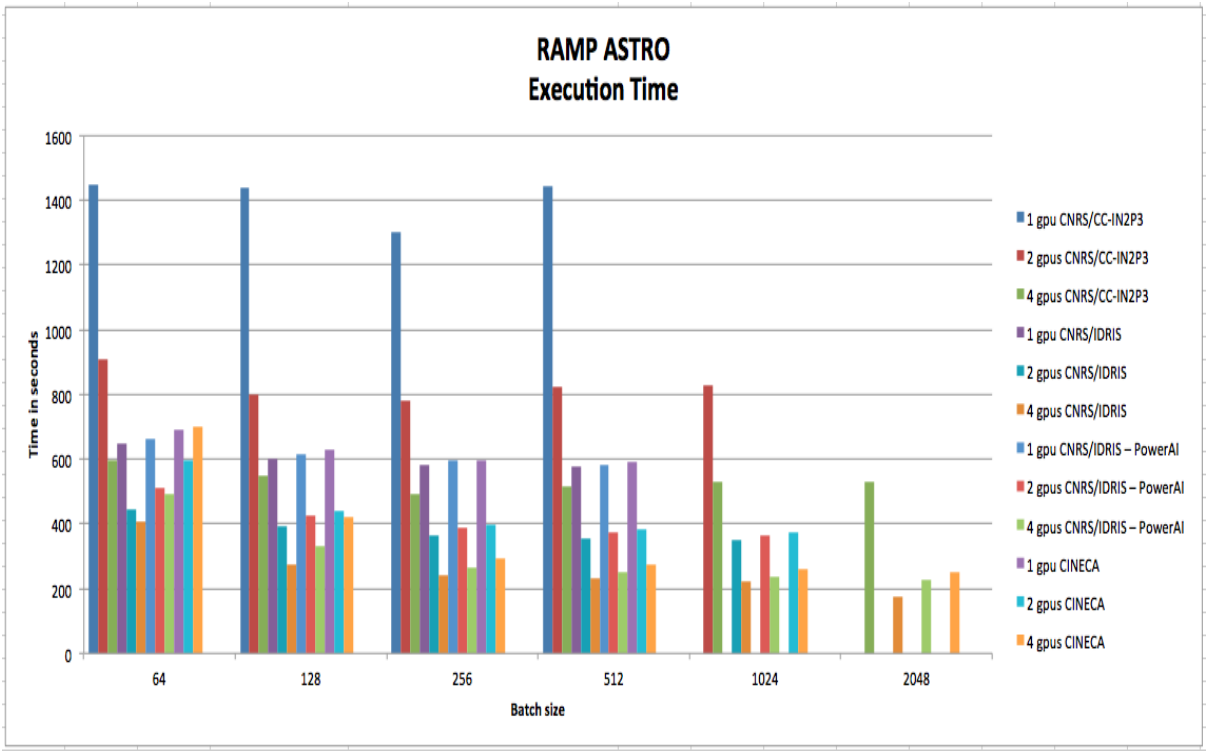

Figure 27 – Astrophysics use case. Execution time. Performance of TensorFlow (X-axis: batch size, Y-axis: execution time (seconds) for the different architectures.

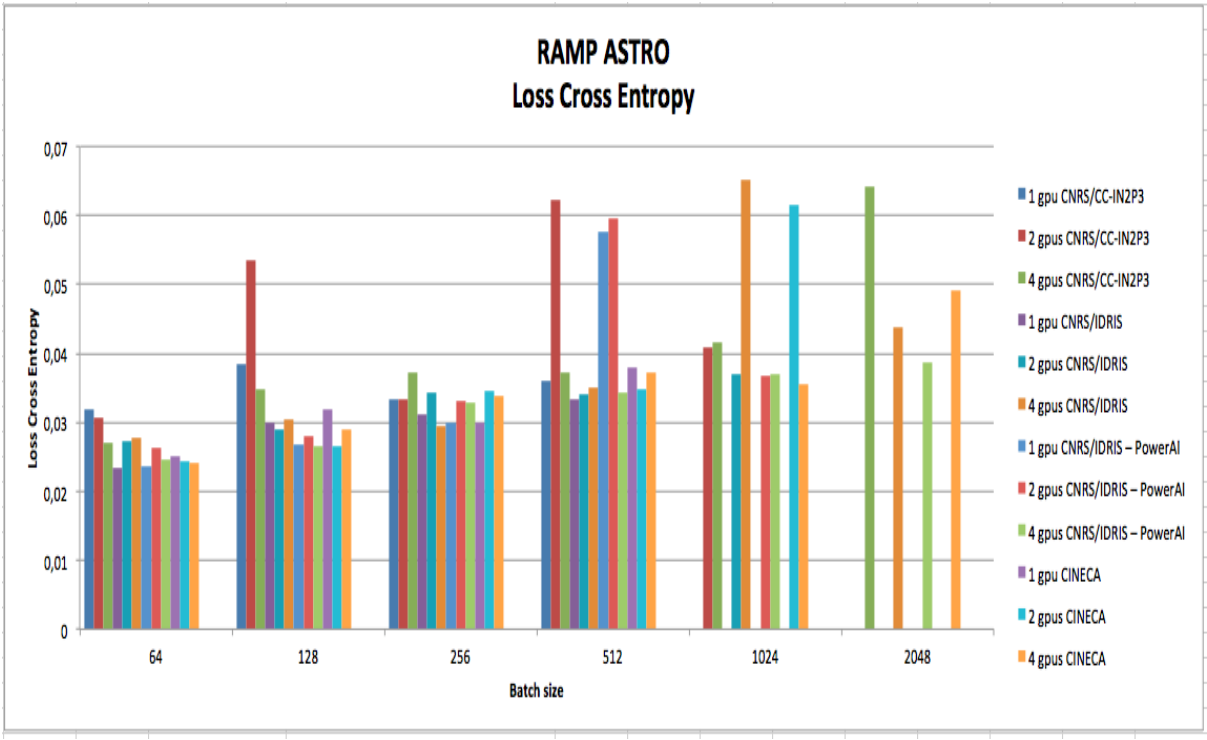

Figure 28 – Astrophysics use case. Loss cross entropy. Performance of TensorFlow (X-axis: batch size, Y-axis: loss cross entropy for the different architectures.

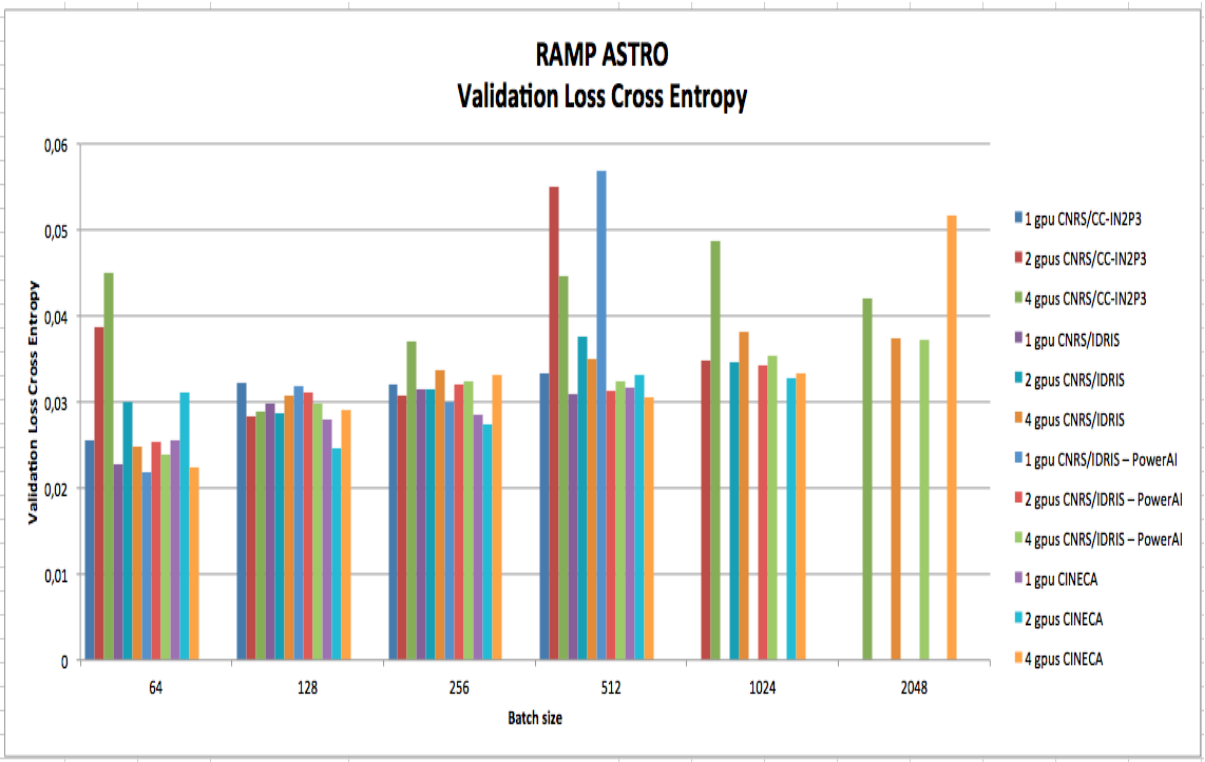

Figure 29 – Astrophysics use case. Validation loss cross entropy. Performance of TensorFlow (X-axis: batch size, Y-axis: loss cross entropy for the different architectures

Again, this use case shows that the performance depends on the hardware, i.e. the P100 (IDRIS and CINECA) outperform the K80 (CC-IN2P3). The difference in using small or bigger batch sizes is reduced compared to the CIFAR-10. Whatever the GPU type, best performance is achieved with 4 GPUs.

For this use case, one can see a slight difference in terms of execution time between the prototype services running with or without containers. With larger batch sizes, i.e. >=1024, the runs abort with an Out Of Memory error message.

8.4.2 The CERN RPC/CMS use case [28]

The goal of the CERN use case is to use a machine learning tool, for data certification, for the RPC system ofthe CMS experiment. CMS Data Quality Monitoring (DQM) and Data Certification (DC) ensures that only consistent data is used for physical analysis, and helps hardware experts to spots anomalies in detector operation. Currently, such certification is done by human experts, and is extremely expensive, in terms of human resources and required expertise. The purpose of the project is to aid DQM and DC, by means of a Neural Network (NN) tool. The plan is to train the network using the PRACE infrastructure. Then, the network will be used at CERN to classify the data. The project follows a bottom-up approach, i.e., to implement a Machine Learning tool to help and facilitate the data certification for one of the CMS subsystems, and then to try to extend the tools for the whole experiment. As a first stage, the goal is to implement Machine Learning for RPC subsystem of the Muon system. Here,, the next step comes naturally, and it is to extend the tool for the Muon system, and, ultimately, for the whole CMS experiment. Additional information regarding this use case is in [29].

The study of this use case has not been finished and is still ongoing at IDRIS. The IDRIS Data Analytics expert members do not run this use case, but provide support by offering users an appropriate environment. Once the project is over, we are expecting feedback from users, describing how they benefit of the PRACE infrastructure.

9 Conclusion

We have described the PRACE Data Analytics service, based on prototype services, whose effectiveness has been studied by running benchmarks and, use cases close to end-users' needs. The benchmarks and use cases have

shown that the performances depend mostly on the hardware. We have seen that the NVIDIA P100 GPU outperforms the NVIDIA K80 GPU as expected.

These prototype services include the most popular deep learning frameworks: TensorFlow and Caffe, with additional libraries: Keras, Horovod and the Anaconda distribution. TensorFlow remains the major player with a fast community growth, providing performance numbers far ahead from those of Caffe. Keras is now fully integrated to TensorFlow 2.0. TensorFlow 2.0 provides also a new distributed training feature. In order to guarantee the effectiveness of the training, monitoring, checking the training process, debugging and checkpointing tasks have to be considered. These tasks can be accomplished by the data scientists using the following tools: nvidia-smi, htop, nmon, Horovod Timeline. Checkpointing is a TensorFlow and a Caffe feature. Two additional services have been identified that can greatly enhance the user productivity: a dataset download service that can make standard datasets easily available to users and the Data Analytics GitLab project available within the PRACE GitLab which can help users to gain expertise and share their experience.

Although we are now closed to a position to offer a PRACE Data Analytics service, documentation, user trainings, frameworks and tools monitoring are required to get to a regular service. We have also noticed during our work that the deep learning field is still facing a fast pace technological evolution so a continuously update will have to be performed to provide the most relevant components at any time.

References

[1] Agnès Ansari, C. (n.d.). Call For Prototypes - https://bscw.zam.kfa-juelich.de/bscw/bscw.cgi/d2599979/CallForPrototypes.doc.

[2] Agnès Ansari, C. (n.d.). Dataset download service Specifications. https://bscw.zam.kfa-juelich.de/bscw/bscw.cgi/d2684726/DataSet-Download-ServiceV1.0.pdf.

[3] Agnès Ansari, C. (n.d.). The Use Cases Master document - https://bscw.zam.kfa-juelich.de/bscw/bscw.cgi/d2640846/UseCasesMasterDocument.docx.

[4] Alberto Garcia Fernandez, C. (n.d.). ILSVRC2012 Use Case Documentation and Results.

[5] Andreas Vroutsis, E. (n.d.). *Basic Benchmarks Documentation and Results.*

[6] Bertrand Rigaud, C.-I. (n.d.). *https://gitlab.in2p3.fr/brigaud/prace-keras-cifar10-benchmark.*

[7] Bertrand Rigaud, C.-I. (n.d.). *https://gitlab.in2p3.fr/brigaud/prace-ramp-astro-benchmark/tree/v1.0.*

[8] *http://caffe.berkeleyvision.org*. (n.d.).

[9] *https://eng.uber.com/horovod*. (n.d.).

[10] *https://github.com/keras-team/keras/blob/master/examples/cifar10_cnn.py*. (n.d.).

[11] *https://github.com/soumith/convnet-benchmarks*. (n.d.).

[12] *https://irods.org*. (n.d.).

[13] *https://jupyter.org*. (n.d.).

[14] *https://keras.io*. (n.d.).

[15] *https://www.anaconda.com/distribution*. (n.d.).

[16] *https://www.eudat.eu/b2safe*. (n.d.).

[17] *https://www.tensorflow.org*. (n.d.).

[18] Kaiming He, X. Z. (n.d.). *Deep Residual Learning for Image Recognition.*

[19] Karen Simonyan, A. Z. (n.d.). *Very Deep Convolutional networks for large scale image recognition.ICLR 2015.*

[20] Krizhevsky, S. a. (n.d.). *ImageNet Classification with Deep Convolutional Neural Networks - Proceedings of the 25th International Conference on Neural Information Processing Systems, 2012.*

[21] L. Litov, B. P.-C.-N. (n.d.). *https://bscw.zam.kfa-juelich.de/bscw/bscw.cgi/d2640648/PRACE_ML_v3.pdf.*

[22] L. Litov, B. P.-C.-N. (n.d.). *https://bscw.zam.kfa-juelich.de/bscw/bscw.cgi/d2681218/Muon_ML_PRACE.docx*.

[23] Long, J. S. (kein Datum). Fully convolutional networks for semantic segmentation.

[24] Marco Rorro, C. (n.d.). *Basic Benchmarks Documentation and Results.*

[25] Marco Rorro, C. (n.d.). The dataset download service design document for use by the PRACE Data Analytics service.

[26] Olaf Ronneberger, P. F. (n.d.). *U-Net: Convolutional Networks for Biomedical Image Segmentation.*

[27] Sermanet, E. Z. (n.d.). *Overfeat: Integrated recognition, localization and detection using convolutional networks. Computing Research Repository, 2013.*

[28] Simonyan, Z. (n.d.). *Very Deep Convolutional Networks for Large-Scale Image Recognition. Computing Research Repository, 2014.*

[29] Szegedy, L. J. (n.d.). *Going Deeper with Convolutions. Computing research repository.*

Acknowledgements
This work was financially supported by the PRACE project funded in part by the EU's Horizon 2020 Research and Innovation programme (2014-2020) under grant agreement 730913.