White paper PRIMEFLEX® for Hadoop – Integrate Deep Learning on GPUs

In order to extend the potential value of Big Data in PRIMEFLEX for Hadoop, Deep Learning frameworks can yield new insights. Computations run in parallel on multiple GPUs make results faster available. What should you look out for when integrating a Deep Learning framework? What are the requirements with regard to the framework? This whitepaper illustrates the most significant aspects.

Content

Introduction

This document explains available Deep Learning frameworks for applicability of parallel GPU support on PRIMEFLEX for Hadoop. It contains a short framework comparison and provides instructions for selected frameworks.

What is the difference between CPUs and GPUs? A CPU is a general purpose processing unit with a few powerful cores where each one sequentially processes a different task. The first generation of GPUs were special purpose processing units for graphic related tasks. GPUs have evolved, and now they are general purpose graphical processing units (GPGPU) which contain thousands of small cores which are able to work together for processing parallel workloads efficiently, not only in graphics but also in other areas requiring massive parallel computations. Thus, GPUs are purpose-built for efficiently handling parallel workloads.

When to use GPUs? GPUs are best suited for compute-intensive massive parallel tasks. The training process for neural networks in Deep Learning frameworks is such an example. In big data analytics environments, GPU-enabled software allows making real-time business decisions. In engineering and structural mechanics applications profit from using the power of GPUs.

What does "parallel GPU support" mean? Deep Learning frameworks have evolved to more and more harness the thousands of cores in general purpose GPUs in order to speed up their compute intensive work while training a neural network. As of today, many frameworks support at least a single GPU. This feature requires explicitly offloading the execution of compute-intensive programming steps together with the data for that step to a GPU. Frameworks either automatically achieve this, which is more user friendly but less flexible, or expect the user or developer to specify the binding. The next stages are parallel use of multiple GPUs on the same server and distributed use on multiple servers. The advantage of having more GPUs at hand for the calculations comes with the drawback that more data transfers between CPUs and GPUs – on the same server or over the network – are necessary for consolidating intermediate results.

This paper focuses on NVIDIA GPU cards which are optionally available for PRIMEFLEX for Hadoop as AI-node component. NVIDIA provides the CUDA Toolkit which includes a large set of components for developing and running GPU-accelerated applications. The toolkit comprises a set of C/C++ GPU-accelerated libraries for Linear Algebra and Mathematics, Deep Learning, Parallel Algorithms, etc. Deep Learning frameworks use these libraries for accessing the GPUs.

Criteria for selecting framework

The search for suitable frameworks concentrated on the following filter criteria:

- Collaboration with PRIMEFLEX for Hadoop
    - Execution on Data Nodes
        - Preferably common resource management for components of PRIMEFLEX for Hadoop and Deep Learning framework
    - Execution on Edge Nodes
        - Combined workflows with PRIMEFLEX for Hadoop require data exchange, e.g. via HDFS
- Programming interface
    - Widely spread in AI: traditional data scientists are accustomed to Python or R
    - Already provided by the PRIMEFLEX for Hadoop software stack, Python preferred, but also Scala or Java API
- Simple installation (no or easy building of framework, required dependencies easy to fulfill)
- Amount and quality of examples included with the framework
- Easy and fast testing during developmentSimple switch from CPU-bound application to GPU-bound
    - Either framework automatically decides which code shall be offloaded to the GPU
    - Or application specifies the amount of GPUs needed
    - Or few code changes to bind code or data to GPU(s)
- Support by a large and active community
- Maturity and with a promising future
- Visualization of static information, e.g. network layer, and dynamic information, e.g. training progress
- License
    - Restrict to Open Source Software for Deep Learning in first study

The following criteria may also be applied during the search but have not been taken into consideration in this version of the paper:

- Programming interface
    - High-level tools: probably more comfortable for non-technical users
- Supported deep learning functionality
- Supported model file formats for exchange with other frameworks
- Good performance
- Documentation for administration and development
- License / Price
    - Scan commercial products

PRIMEFLEX for Hadoop software stack and GPUs

A good starting point for the framework search are the components of PRIMEFLEX for Hadoop which is currently based on RHEL7 and supports Cloudera, Hortonworks and MapR Hadoop distributions. All three distributions contain YARN as cluster-wide resource management and MapReduce and Spark as execution frameworks. This chapter presents what YARN, Spark, MapReduce and the distribution vendors say about GPU support and which Deep Learning frameworks they think are worth mentioning on their web pages.

Operating System and Python

Many Deep Learning frameworks are based on python and, apart from the basic python components, may require additional python modules or operating system packages. The easiest way to install python is as an operating system package. But this way may imply some restrictions concerning the exact python version and thus the available python modules, especially the availability of ready to use GPU enabled modules for Deep Learning frameworks. PRIMEFLEX for Hadoop comes with RHEL7 and python version 2 installed.

Hadoop resource management

YARN is the central resource management instance for applications in Hadoop clusters handling allocation of CPUs and memory. Starting with Apache Hadoop version 3.0, YARN supports an extensible resource model that can be enhanced for also managing GPUs. Hadoop distributions are still in the process of adapting to this major release.

Hadoop versions less than 3.0 offer several workarounds overcoming scheduling problems for GPU-utilizing jobs, as pointed out by the DeepLearning4j web page [1]:

- **Node labels** are a useful feature already available in YARN (versions 2.6 or greater) which provide a way for grouping nodes with similar characteristics, and applications can specify on which group of nodes to run. This feature can be used to distinguish between nodes with and without GPUs. The Cloudera Hadoop distribution does not support nodel labels, considering them not yet ready, see [2]. Node labels do not prevent YARN from trying to run multiple GPU-utilizing tasks on the same node if their CPU and memory requirements can be fulfilled on that node.
- **Allocating sufficient memory and cores to the GPU-utilizing tasks** ensures that YARN will not schedule other tasks on the same nodes. In combination with the node label, this solution will reserve the GPU node or nodes for exclusive use by a single GPU-utilizing task. This approach avoids concurrent access to GPUs, but wastes resources.
- **Using the Docker Container Executor** (DCE) for YARN jobs allows GPU resource management via docker. If the GPU is declared as being used in the docker container, then this ensures that the GPU is not allocated to multiple tasks. The docker container nvidia-docker implicitly handles this declaration. By default, YARN uses the Linux Container Executor(LCE) for executing the tasks of a job directly on top of the native operating system. With Hadoop versions 2.x, there can only be one type of container executor active in the cluster, either all YARN jobs run via LCE or via DCE. Cloudera *recommends waiting for Hadoop 3.0 before deploying Docker containers, citing security issues and other caveats* in the article [3]. In Hadoop version 3.0, LCE is able to run tasks on the native operating system and tasks in docker containers in parallel.

Hadoop execution frameworks

Spark provides basic machine learning functionality in its libraries MLlib and ML, thus offloading of computations to GPUs is a subject that has already arisen, but change requests are still open or not fixed, see [4]. Some established Deep Learning frameworks have inspired projects for running them in a distributed way on Spark. Examples are CaffeOnSpark and TensorFlowOnSpark or TensorFrames.

MapReduce will not be used as a starting point in the search for suitable Deep Learning frameworks as it does not include any machine learning functionality and has been superseded by Spark as a leading Hadoop execution framework.

Hadoop distributions

Cloudera

Cloudera's website and community posts referencing GPUs mainly demonstrate how to use Deep Learning frameworks on the Cloudera Data Science Workbench (CDSW):

- Caffe/CaffeOnSpark, TensorFlow/TensorFlowOnSpark, Deeplearning4j on the CDSW [5]
- Deeplearning4j with a Scala example [6]
- TensorFlow, Keras and Theano [7]

Cloudera plans to adapt their platform to Apache Hadoop version 3 features in 2018.

The current version of PRIMEFLEX for Hadoop supports optional nodes (AI Node) with GPUs which run the CDSW.
The CDSW supports parallel GPU utilization on a single server as described in [8]:
By enabling GPU support, data scientists can share GPU resources available on CDSW nodes. Users can request a specific number of GPU instances, up to the total number available on a node, which are then allocated to the running session or job for the duration of the run.
There is currently no documentation for GPU-utilizing jobs distributed over several nodes which have been installed with the CDSW.

Hortonworks

Hortonworks website and Hortonworks slides on slideshare point to:

- A blog showing distributed TensorFlow on YARN using nvidia-docker [9]
- A community entry for setting up Deeplearning4j with Apache Spark and an HDP cluster on AWS [10]

The Hortonworks platform may be the one which is closely linked with open source Hadoop, but there is apparently no public roadmap or planned release date for an HDP version based on Apache Hadoop version 3 and thus including the latest YARN resource model which can provide GPU support.

MapR

MapR's website references GPUs in:

- A blog entry demonstrating how to use Caffe and CaffeOnSpark in YARN cluster mode, but without test of GPU integration [11]
- An example with distributed TensorFlow on Kubernetes requiring manual steps to start worker pods [12]

There is currently no publicly available information about when MapR will adapt to Apache Hadoop version 3 features.

Integration considerations

Integrating a GPU-utilizing framework with PRIMEFLEX for Hadoop can be achieved in several ways:

- Setting up the framework on a node with read and write access to the HDFS for data exchange with PRIMEFLEX for Hadoop
- Running the framework jobs on the Data Nodes of the Hadoop cluster

The first integration solution can be achieved for the Cloudera Hadoop distribution by installing a node with GPUs, the CDSW and the HDFS gateway service. This solution is suitable for those frameworks that have APIs in programming languages supported by the CDSW, i.e. Python, R and Scala. For other programming languages or other Hadoop distributions, a framework can be set up on any computer with access to the HDFS, even a Windows PC or notebook.

The second integration solution can be achieved by running the framework jobs in parallel to the Hadoop YARN jobs on the Data Nodes, but this solution will require static division of resources such as CPU or memory between the framework and YARN jobs. This solution will not make good use of the resources. If the framework jobs shall be run on the Hadoop cluster, then the best solution is to run them via YARN.

PRIMEFLEX for Hadoop does not yet support GPUs on Data Nodes as the supported distributions do not yet provide suitable handling for GPUs in YARN. Hadoop distributions will adapt to newer versions of YARN with GPU support in the near future. Thus, looking ahead, this document assesses deep learning frameworks supporting distributed GPU computing on YARN.

Framework comparisons in the web

There are several web sites with comparisons of Deep Learning frameworks.

Wikipedia provides a comparison table for Deep Learning software including information about NVIDIA CUDA support for GPUs and parallel execution (multi node) at [15].

Skymind compare their software DeepLearning4j with other frameworks at [16].

The article [13] from March 2016 compares Caffe, TensorFlow, Theano, Torch and Neon with focus on single server setups.

There is another article [14] from February 2017 comparing the performance of Caffe, CNTK, TensorFlow, Theano, Torch, MXNet and Paddle. The authors draw the conclusion that *all tested tools can make good use of GPUs to achieve significant speedup over their CPU counterparts. However, However, there is no single software tool that can consistently outperform others*.

There is a post from October 2017 ranking 24 popular Deep Learning frameworks for data science based on GitHub and Stack Overflow activity, as well as Google search results [21].TensorFlow, Keras, Caffe, Theano, Pytorch, Sonnet, MXnet, Torch and CNTK are the top 10.

Framework comparison

This section provides a comparison of Deep Learning frameworks suitable for collaboration with PRIMEFLEX for Hadoop. The comparison is based on information readily available in the web and own experiences with installing and running Deep Learning frameworks. Conclusions about multiple GPU support have been drawn from web information and framework documentation as only a single GPU has been available for tests.

Dropped candidates

When it comes to Deep Learning frameworks and GPUs, a lot of names turn up. The following list contains frameworks which are not included in the detailed comparison as they do not match one or more of the set criteria:

Framework	Comment
BigDL by intel	is intended for running on CPUs via the Intel® Math Kernel Library (MKL).
Caffe	only supports multiple GPUs via its C/C++ API, does not support RBM/DBMs and includes no HDFS access.
dist-keras	is a distributed deep learning framework built op top of Apache Spark and Keras; there have only been 4 commits since July 2017, i.e. in the last 8 months.
Driverless AI by H2O	is a commercially licensed product, was started in September 2017 and supports multiple GPUs on single server and HDFS. Distributed GPU computing on GPU and Spark/YARN support planned for Q4 2018.
Elephas	brings deep learning with Keras to Apache Spark splitting training into portions for Spark workers, but does not support GPUs. The latest release was in 2016.
GOAi (GPU Open Analytics Initiative)	seeks to create an open spec and set of tools for data exchange between libraries and applications in a pipeline without needing to move data off the GPU.
GPUenabler by IBM	is a Spark package offloading calculations to NVIDIA GPUs. The developer has to use GPU enabled map/reduce methods
Jcuda	is a thin Java layer over CUDA.
MXnet on Spark	currently has a Scala API, but no Python API, is still experimental and there is no prebuilt package.
Neon by Nervana Systems	acquired by intel in August 2016, currently supports certain NVIDIA GPUs, but as intel intends to offer an own processor for deep learning workloads (see [17]), future support for NVIDIA GPUs in neon is questionable.
Numba	is a just-in-time compiler for Python array and numerical functions with native code generation for the CPU (default) and GPU. Numba can be used in a distributed system via Dask. The Numba community considers distributed GPU computing a bleeding edge capability.
Pyculib	is a package that provides access to several numerical libraries that are optimized for performance on NVIDIA GPUs.
Scikit-learn	a python package for machine learning, will not have GPU support in the near future, see [18].
SINGA with Apache 2.0 license	can run on multiple GPUs on a single server, but no information was available about multi server support.
Sonnet with Apache 2.0 license, by DeepMind	is not a framework of its own but a library on top of TensorFlow for building complex neural networks.
SparkNet	Is a distributed neural networks on top of Apache Spark and Caffe, does not provide a Python API, last committed changes were in 2016.
TensorFrames by Databricks	is a highly experimental TensorFlow binding for Scala and Apache Spark and is provided as a technical preview only.
Theano	can run on multiple GPUs on a single server, but no information was available about multi server support; GPU API is also new.
Torch	comes with Lua as programming language which is not as commonly used as Python, R, Scala or Java.

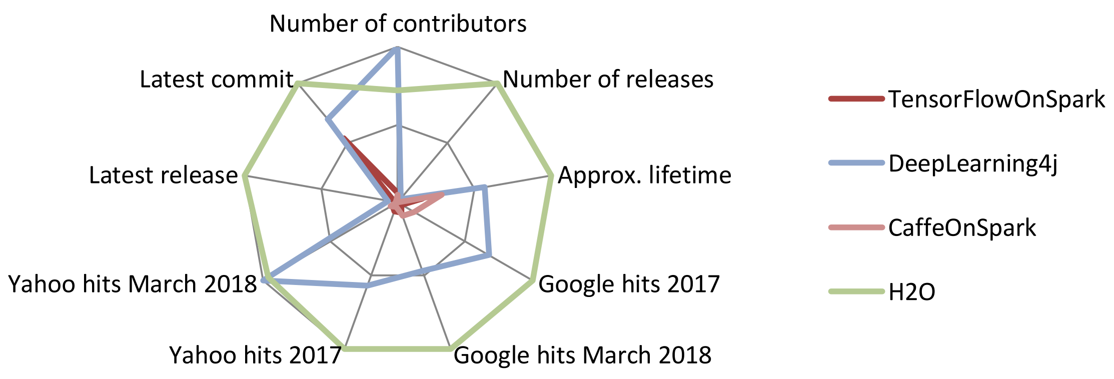
Finalists

The following tables list frameworks capable of running in parallel on multiple GPUs and distributed over multiple servers.

The first table shows the set of frameworks that can be run via Spark. Thus, YARN can control resources of framework specific applications.

Framework	Multi-GPU on single server	Multi-GPU on multiple servers	HDFS access for data and model	Runs with Spark, thus managed by YARN	Installation / prebuilt package	Maturity / Releases at Github	Comments / Experiences	Visualization
CaffeOnSpark, Apache 2.0 license	☺	☺	☺	☺	😐 build from C++ (Caffe) and Java sources	0 releases since 2016	☹ No development activities for more than a year (in February 2017) ☺ Following build instructions succeeded. Test suite on local server succeeded. ☹ Test suite with Spark 2 failed with missing jar archive.	☹ no built-in
DeepLearning4j, Apache 2.0 license Skymind Intelligence Layer (SKIL), Community and enterprise edition	☺ with application rebuild	☺ with application rebuild	☺ via Spark	☺ with application rebuild	😐 build with application from Java sources; SKIL as OS package or docker image	47 releases since 2014	☹ Python as programming language not supported. Java and Scala are supported. ☺ certified on Cloudera's CDH and Hortonworks's HDP distributions of the Hadoop ecosystem.	☺ DeepLearning4j UI
H2O, Deep Water and Sparkling Water	☺ via Deep Water	☺ via Deep Water	☺	☺ via Sparkling Water	😐 docker image or build required for Deep Water	1550 H2O releases since 2011, 2 releases since 2017 for Deep Water	☹ GPU support component Deep Water is no longer under active development 😐 Deep Water needs a backend. Currently supported are TensorFlow, MXnet, Caffe ☺ enterprise support	☺ H2O Flow
TensorFlowOnSpark, Apache 2.0 license	☺	☺	☺	☺	☺	0 releases since 2017	☹ Following installation and example instructions failed. ☺ Alternative installation solution and different example succeeded.	☺ TensorBoard

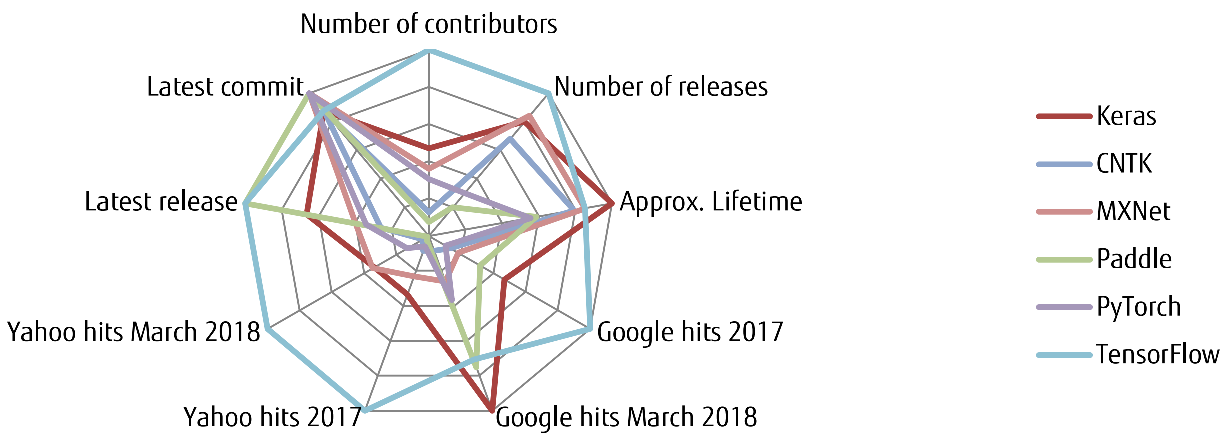
The following diagram shows some key values for the Deep Learning frameworks, where the outer ring of the net represents the best value in this set of frameworks. This diagram reflects a certain point in time. The values may change as Deep Learning frameworks evolve.



The second table shows the set of frameworks able to run in a distributed way on multiple servers, but not controlled via YARN.

Framework	Multi-GPU on single server	Multi-GPU on multiple servers	HDFS access for data and model	Runs with Spark, thus managed by YARN	Installation / prebuilt package	Maturity / Releases at Github	Comments / Experiences	Visualization
Caffe2, Apache 2.0 license	☺	😐 run script on each node	☹	☹	😐 build from C++ sources or prebuilt	4 releases since 2017	☹ Following installation instructions failed in CDSW.	☹ no built-in, use python package matplotlib
CNTK, MIT license	☺	☺	☹ no information	☹	😐 build from C++ sources or prebuilt	34 releases since 2016	☹ Following installation instructions for examples failed. Keras over CNTK failed on lock file access.	☺ TensorBoard
Keras, MIT license	☺ depends on backend, ok with TensorFlow	😐 depends on backend	😐 maybe via TensorFlow backend	☹	☺	40 releases since 2015	😐 Keras needs a backend. Currently supported backends are TensorFlow, CNTK and Theano; MXnet backend requires special Keras build	☺ TensorBoard if TensorFlow backend
MXnet, Apache 2.0 license	☺ with MXNet variant	☺ with MXNet variant	😐 with MXNet build setting HDFS flag	☹	😐 build from C++ sources for HDFS support	42 releases since 2015, apache incubator	😐 MXnet did not use all available CPU cores when GPU was disabled. ☹ Keras over MXnet ResNet example used 100 % GPU but stalled.	☹ no built-in, use python package graphviz
Paddle, Apache 2.0 license	☺	☺	☹	☹	☺	10 releases since 2016	Following installation and example instructions succeeded in CDSW session without GPU. ☹ Running example in GPU enabled CDSW session crashed session.	☺ PaddleBoard
Pytorch, [Own open source license]	☺	☺	☹	☹	☺	15 releases since 2016, beta	Following installation and example instructions succeeded in CDSW.	☹ no built-in, use python package graphviz
TensorFlow, Apache 2.0 license	☺	☺	☺	☹	☺	50 releases since 2015	☹ some articles show TensorFlow as slow ☺ NVIDIA's TensorRT integration in latest release 1.7 will speed up TensorFlow	☺ TensorBoard

The following diagram shows some some key values for the Deep Learning frameworks, where the outer ring of the net represents the best value in this set of frameworks. This diagram reflects a certain point in time. The values may change as Deep Learning frameworks evolve.



Conclusions

The amount of Deep Learning frameworks in the preceding finalists tables together with the list of sorted out candidates show that there are many development actitivities in this field. New frameworks pop up and frameworks on which hope was pinned are no longer actively developed.

There is a number of Deep Learning frameworks which are able to distribute their workload to multiple GPUs on a single server or multiple servers without help from YARN. Several frameworks are worth of further investigating their potential. If you need immediate data exchange between Deep Learning applications and analytics on PRIMEFLEX for Hadoop, then the Deep Learning framework needs HDFS access. This criteria further narrows down the field. TensorFlow and Keras over TensorFlow are the most promising concerning ease of installation, runnable examples, higher level extensions and active community.

It will be a logical step for PRIMEFLEX for Hadoop to support distributed GPU computing on YARN in the near future. Suitable deep learning frameworks have been assessed now. 4 present-day candidates have been studied:

- CaffeOnSpark looks promising for data scientists who are familiar with Caffe, but there have been no development activities on CaffeOnSpark for more than a year.
- DeepLearning4J is a candidate for professional Java developers familiar with building and installing complex Java applications.
- H2O, Deep Water and Sparkling Water complement each other, but Deep Water, the GPU supporting component for H2O is no longer under active development.
- TensorFlowOnSpark is currently a promising candidate for Python developers, even if it is relatively young and there is no release yet. Its advantages are that
  - existing TensorFlow programs can be migrated by changing less than 10 lines of code
  - there is an active development community
  - it can run on distributed GPUs with a GPU enabled TensorFlow

The above findings are only a snapshot of currently known and available Deep Learning frameworks with GPU support. This market is fast evolving, so that you have to keep an eye on it for new developments.

The following list gives an outlook for future research subjects in the next version of this document:

- Study new frameworks
- Study commercial products, e.g. SKIL by Skymind or Driverless AI by H2O
- Compare performance of framewor
  Build a comparable basis for a performance analysis of frameworks in order to verify performance results given in several articles, e.g. run Keras examples with Keras over different backends such as TensorFlow, CNTK and MXnet
- Study new technology
  Verify performance boost of NVIDIA's TensorRT with TensorFlow
  The integration of NVIDIA's TensorRT into TensorFlow, starting with version 1.7, shows a performance boost for inference as described by NVIDIA at [22]. The article shows a diagram with *ResNet-50 performing 8x faster under 7 ms latency with the TensorFlow-TensorRT integration using NVIDIA Volta Tensor Cores versus running TensorFlow only*.

Performance Boost with GPUs

Performance measurements were first run in sessions of the CDSW, i.e. in docker containers. This runtime environment allows easy setup and tears down of different configurations. Deep Learning frameworks based on Python have been chosen.
Docker containers provide a thin-layered virtual execution environment. This architecture leads to overhead visible as longer execution times. Article [19] presents the results from an analysis of the overhead due to running deep learning frameworks in docker containers and the authors come to the conclusion that the *docker engine manages to minimize the overhead pretty well*. In order to check this conclusion, some of the CDSW projects have been transferred to the native operating system, and examples included with the frameworks have been run there.

Deep Learning frameworks already include examples for different kinds of use cases. Measurements were based on one or more examples from each category. The focus was on training networks as this action is the most time consuming part. The elapsed time for training a network with a given number of cycles has been taken as measurement value. Examples have only been modified in order to control the runtime or saturate the CPU/GPU, but no tuning has taken place.

Hardware

- CPUs: 2 x Intel(R) Xeon(R) Silver 4110 CPU @ 2.10GHz, 16 cores each
- GPUs: 1 x Tesla P100-PCIE, 16GB memory, 3584 CUDA cores
- RAM: 128 GB

Software

- RHEL 7.3 / Ubuntu 16.04 in CDSW docker image
- CDSW version 1.2.1-1
- Python 2.5 / 2.7.11
- CUDA 8.0.61
- CUDNN 6.0.21

Frameworks

- Keras: Python module keras with version 2.1.2
- TensorFlow: Python module tensorflow-gpu with version 1.4.1
  The binary was not compiled to use: SSE4.1 SSE4.2 AVX AVX2 AVX512F FMA.
  Tests without GPU are also run with tensorflow-gpu which automatically falls back to CPU

Datasets

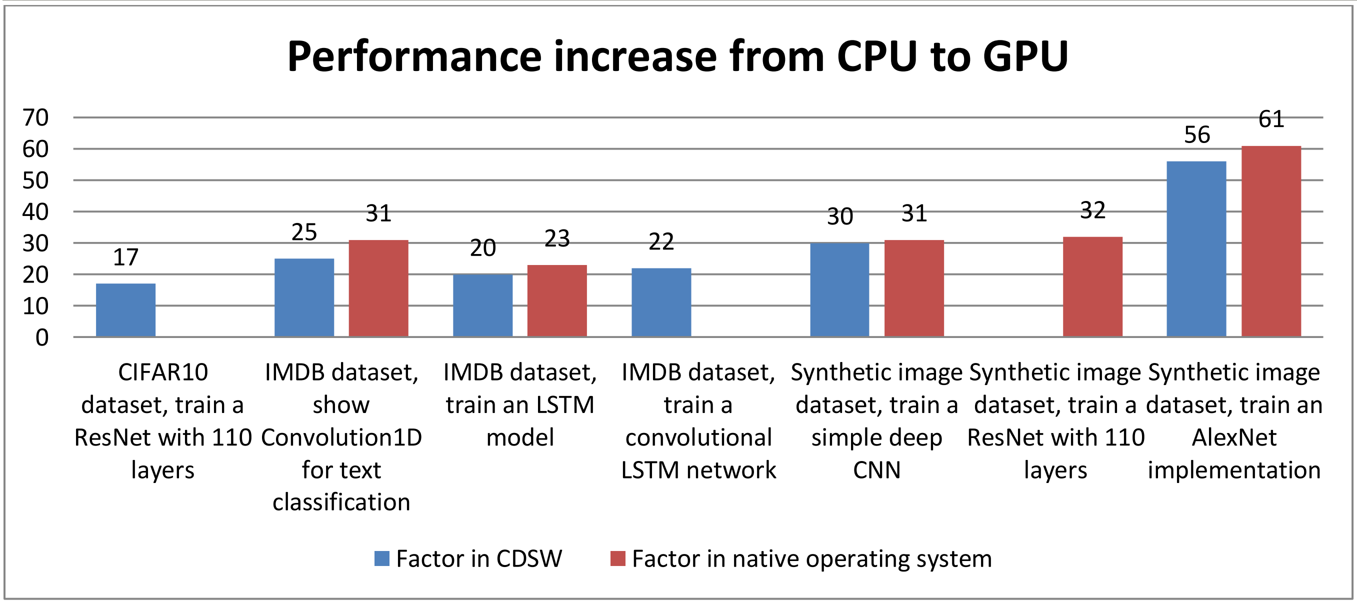
- CIFAR10 / CIFAR100: Dataset of 50,000 32x32 color training images, labeled over 10 or 100 categories
- IMBD: Dataset of 25,000 movies reviews from IMDB, labeled by sentiment (positive/negative)
- Synthetic image dataset of configurable dimension, e.g. RGB images with 128 pixel for width and height for generating enough load which could not be achieved with CIFAR10/CIFAR100 datasets

Neural Networks

- Convolutional neural network (CNN)
- Residual network (ResNet)
- Long Short Term Memory network (LSTM)
- AlexNet is a convolutional neural network which competed in the ImageNet Large Scale Visual Recognition Challenge in 2012

All tests have been run with Keras as frontend and TensorFlow as backend either in a session of the CDSW or on the native operating system of the CDSW server. This framework combination was easy to set up and provides a set of examples.

The following diagram shows the performance increase from CPU to GPU that was achieved with the different kinds of neural networks and training input datasets.



The test results in the diagram above confirm that there is an overhead for execution in a docker image of the CDSW compared to execution directly on the native operating system. You have to weigh the better performance gain in the native operating system against the flexibility of the CDSW. In all test cases, running the training on a GPU significantly decreased the training runtime. The factor varied from 17x to 61x depending on the type of network and input dataset. The more complex the neural network or the larger the dataset items, the greater to benefit.

Using TensorFlow, Keras and TensorFlowOnSpark

This section shows how to install TensorFlow, Keras and TensorFlowOnSpark and how to run an example included in each framework. Instructions are either available for an installation in the CDSW (CDSW) or in the native operating system of the server hosting the CDSW or a Hadoop data node. All example commands for CDSW have to be run in a terminal window of a session. The CDSW examples run the python scripts from the Run button, because the script output is then logged with the session. If you use a proxy for accessing the internet, make sure that the HTTP/HTTPS proxy environment variables http_proxy and https_proxy are set. In CDSW, either set them globally or in the project or within the terminal window of a session.

All descriptions use a Python 2 version with included pip. The CDSW has pip by default, but the native operating system may not. Thus, for the native operating system, check for pip and install if necessary:

1. Check for pip:
   ```
   type pip
   ```
2. If pip cannot be found, search for RPM package containing pip:
   ```
   yum search python2-pip
   ```
3. As root, install the above pip package:
   ```
   yum install python2-pip
   ```

By default, pip requires superuser rights as it installs the Python modules globally. You can install modules to your local user account with the option --user. The CDSW automatically adds this option.

Use Prebuilt TensorFlow

There is a Python module for TensorFlow without GPU support (tensorflow) and another one with GPU support (tensorflow-gpu).

Install a TensorFlow without GPU support by following these steps:

1. At shell prompt, install TensorFlow with pip:
   ```
   pip install --user tensorflow
   ```

Install a GPU enabled TensorFlow following these steps:

1. Install NVIDIA libraries in native operating system.
2. If a session of the CDSW is the target, then create a docker image with GPU support as described in the Cloudera online documentation, see [31].
3. At shell prompt, install GPU enabled TensorFlow with pip:
   ```
   pip install --user tensorflow-gpu
   ```

Test accessability of GPUs following these steps:

1. Create a tiny example script to check GPU availability in TensorFlow:

```
import tensorflow as tf
a = tf.constant([1.0, 2.0, 3.0, 4.0, 5.0, 6.0], shape=[2, 3], name='a')
b = tf.constant([1.0, 2.0, 3.0, 4.0, 5.0, 6.0], shape=[3, 2], name='b')
c = tf.matmul(a, b)

# Creates a session with log_device_placement set to True.
sess = tf.Session(config=tf.ConfigProto(log_device_placement=True))

# Runs the operation.
print(sess.run(c))

# Prints a list of GPUs available
from tensorflow.python.client import device_lib
def get_available_gpus():
    local_device_protos = device_lib.list_local_devices()
    return [x.name for x in local_device_protos if x.device_type == 'GPU']

print get_available_gpus()
```

2. Run script. Example output:

```
[…]
print get_available_gpus()
[u'/device:GPU:0']
2017-12-11 13:57:39.831698: I tensorflow/core/common_runtime/direct_session.cc:299] Device mapping:
/job:localhost/replica:0/task:0/device:GPU:0 -> device: 0, name: Tesla P100-PCIE-16GB, pci bus id: 0000:18:00.0, compute
capability: 6.0

2017-12-11 13:57:39.845429: I tensorflow/core/common_runtime/placer.cc:874] MatMul:
(MatMul)/job:localhost/replica:0/task:0/device:GPU:0
2017-12-11 13:57:39.845467: I tensorflow/core/common_runtime/placer.cc:874] b:
(Const)/job:localhost/replica:0/task:0/device:GPU:0
2017-12-11 13:57:39.845478: I tensorflow/core/common_runtime/placer.cc:874] a:
(Const)/job:localhost/replica:0/task:0/device:GPU:0
2017-12-11 13:57:39.875251: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1120] Creating TensorFlow device
(/device:GPU:0) -> (device: 0, name: Tesla P100-PCIE-16GB, pci bus id: 0000:18:00.0, compute capability: 6.0)
```

Thus, in this example, there is a single GPU of type Tesla P100.

Get all examples included with TensorFlow following these steps:

1. If a session of the CDSW is the target, then open terminal window in the session.
2. Clone TensorFlow Github repository to get the examples:

```
mkdir repos
cd repos
git clone https://github.com/tensorflow/tensorflow.git tensorflow
```

3. Check installed TensorFlow version:

```
pip list | grep tensorflow
```

4. Set clone to suitable TensorFlow version matching installed TensorFlow:

```
cd tensorflow
git checkout r1.6
```

5. If a session of the CDSW is the target, then, in the project file system explorer, navigate to the TensorFlow examples folder repos/tensorflow/tensorflow/examples
6. If a Hadoop node is the target, then navigate to the TensorFlow examples directory

```
cd $HOME/repos/tensorflow/tensorflow/examples
```

The next section shows how to run a TensorFlow example. It uses a simple tutorial example which classifies the images from the MNIST dataset and provides summary information for display in TensorBoard.

Run a TensorFlow example following these steps:

7. If a session of the Cloudera Data Science Workbench is the target
   a. In the project file system explorer, open an example from one of the examples subfolders by clicking on it, e.g. mnist_with_summaries.py in folder repos/tensorflow/tensorflow/examples/tutorials/mnist.
   b. Run example via the *Run* menu.
8. If a Hadoop node is the target
   a. Choose a script from one of the TensorFlow examples subdirectories, e.g. the python script mnist_with_summaries.py in subdirectory tutorials/mnist
   b. Run the choosen example with the python command:

```
python $HOME/repos/tensorflow/tensorflow/examples/tutorials/mnist/mnist_with_summaries.py
```

9. The output may indicate that TensorFlow is not optimized which is expected as the default TensorFlow package is not optimized.

TensorFlow provides an API for logging summary data to event files while training a network or during evaluation. TensorBoard is a graphical user interface for live or retrospective visualization of the logged data.

This example shows now to start the TensorBoard on a Hadoop node:

1. Run a TensorFlow example, e.g. mnist_with_summaries.py as this example generates data for the TensorBoard
2. While the example is running, start TensorBoard in background on a given port and with a given log directory:

```
nohup $HOME/.local/bin/tensorboard --port <YOUR_PORT> --log <YOUR_LOG_DIRECTOR> &
```

3. Point browser to the URL of your host with the given port.
4. Now you can view information about the running TensorFlow job for mnist_with_summaries.py.

This example shows how the TensorBoard can be started in a session of the CDSW:

5. Run a TensorFlow example, e.g. mnist_with_summaries.py as this example generates data for the TensorBoard
6. While the example is running, open terminal window in the same CDSW session.
7. Start TensorBoard on port $CDSW_PUBLIC_PORT and with log directory logs in the HOME directory of the session user:

```
nohup $HOME/.local/bin/tensorboard --port $CDSW_PUBLIC_PORT --log ./logs &
```

8. Get the TensorBoard URL:

```
echo http://$CDSW_ENGINE_ID.$CDSW_DOMAIN
```

9. Open echoed URL in browser.
10. Now you can view information about the running TensorFlow job for mnist_with_summaries.py.

Build TensorFlow

Prebuilt Python modules for TensorFlow are quite handy for first experiences with this framework, but a time may come when the configured functionality of the prebuilt modules is not enough.The prebuilt TensorFlow module does not include some optional features, e.g. supporting optimization functionality provided by the installed CPUs or KAFKA support. If you want to use a feature, then you have to build TensorFlow. This section shows how to build and install TensorFlow. The description follows the original TensorFlow instructions at https://www.tensorflow.org/install/install_sources. The following instructions cover installation in the CDSW and directly on a Hadoop node.

Prerequisites:

- Make sure that the environment variables http_proxy and https_proxy are set.
- Bazel build tool.
- CDSW session with at least 64 GB memory

The CDWS docker images have Ubuntu as operating system and there is a bazel repository for CentOS, but within a docker session you cannot install software. There are two ways to get bazel: Updating the CDSW docker image or building bazel in a CDSW session.
If you choose to update the docker image, then please follow the description on the Cloudera webpage http://blog.cloudera.com/blog/2017/09/customizing-docker-images-in-cloudera-data-science-workbench/ . This solution requires root access to the native operating system. If you choose to build bazel in a CDSW session, then follow the instructions below which are based on https://docs.bazel.build/versions/master/install-compile-source.html.
This section shows how to build and install TensorFlow in the CDSW or directly on a PRIMEFLEX for Hadoop node, e.g. the node hosting the CDSW. The description follows the original TensorFlow instructions at https://www.tensorflow.org/install/install_sources.

The following instructions show how to install bazel on a PRIMEFLEX for Hadoop node if you have access to the root account:

1. Login as root
2. Download bazel repository info from https://copr.fedorainfracloud.org/coprs/vbatts/bazel/ Epel for CentOS 7.
3. Copy file to /etc/yum.repos.d/
4. install bazel with yum; this may also ask for installing other packages:

```
yum install bazel
```

5. Install required Python packages:

```
yum -y install python-pip python-wheel # install if not yet present
# if there is a message that pip should be updated, do it
pip install --upgrade pip
# as there is no matching RPM package python-numpy found, use pip to install numpy
pip install numpy
# as there is no RPM package python-dev, check for RPM package python-devel
yum info python-devel
# if it is not installed, install it with yum
yum install python-devel
```

The following instructions show how to build bazel a session of the CDSW or in case you do not have access to the root account on a Hadoop node:

1. If a session of the CDSW is the target, then open terminal window in CDSW session with enough RAM
2. Download a distribution version of bazel source zip from https://github.com/bazelbuild/bazel/releases:

```
curl -O https://github.com/bazelbuild/bazel/releases/download/0.11.1/bazel-0.11.1-dist.zip
```

3. Unzip bazel and compile it:

```
mkdir $HOME/bazel-source $HOME/tmp
cd $HOME/bazel-source
unzip ../bazel-*dist.zip
# this step may help if the session terminates during compilation
TMPDIR=$HOME/tmp; export TMPDIR
bash ./compile.sh
# copy bazel binary file from subdirectory output to $HOME/bin
mkdir -p $HOME/bin; cp output/bazel $HOME/bin
# add $HOME/bin to PATH so that bazel binary can be easily called
PATH=$PATH:$HOME/bin; export PATH
```

Build TensorFlow as follows:

1. If a session of the CDSW is the target, then open terminal window in CDSW session with enough RAM.
2. Install required packages numpy, dev and wheel either as corresponding system packages or via pip:

```
pip install --user numpy dev wheel
```

3. Setup proxy for git if not yet set:

```
git config --global http.proxy http://proxyuser:proxypwd@proxy.server.com:8080
```

4. Clone git repository:

```
cd; mkdir repos; cd repos
git clone https://github.com/tensorflow/tensorflow tensorflow
```

5. Check prebuilt TensorFlow version that would be installed

```
pip search tensorflow
```

6. Checkout a matching TensorFlow version

```
cd tensorflow; git checkout r1.6
```

7. Configure TensorFlow with defaults as option --march=native includes CPU optimization features supported by the native machine:

```
./configure
Extracting Bazel installation...
You have bazel 0.7.0- (@non-git) installed.
Please specify the location of python. [Default is /usr/bin/python]:


Found possible Python library paths:
  /usr/lib/python2.7/site-packages
  /usr/lib64/python2.7/site-packages
Please input the desired Python library path to use.  Default is [/usr/lib/python2.7/site-packages]

Do you wish to build TensorFlow with jemalloc as malloc support? [Y/n]:
jemalloc as malloc support will be enabled for TensorFlow.

Do you wish to build TensorFlow with Google Cloud Platform support? [Y/n]: n
No Google Cloud Platform support will be enabled for TensorFlow.

Do you wish to build TensorFlow with Hadoop File System support? [Y/n]:
Hadoop File System support will be enabled for TensorFlow.

Do you wish to build TensorFlow with Amazon S3 File System support? [Y/n]: n
No Amazon S3 File System support will be enabled for TensorFlow.

Do you wish to build TensorFlow with Apache Kafka Platform support? [y/N]: y
Apache Kafka Platform support will be enabled for TensorFlow.

Do you wish to build TensorFlow with XLA JIT support? [y/N]:
No XLA JIT support will be enabled for TensorFlow.

Do you wish to build TensorFlow with GDR support? [y/N]:
No GDR support will be enabled for TensorFlow.

Do you wish to build TensorFlow with VERBS support? [y/N]:
No VERBS support will be enabled for TensorFlow.

Do you wish to build TensorFlow with OpenCL SYCL support? [y/N]:
No OpenCL support will be enabled for TensorFlow.

Do you wish to build TensorFlow with CUDA support? [y/N]:y
CUDA support will be enabled for TensorFlow.

Please specify the CUDA SDK version you want to use, e.g. 7.0. [Leave empty to default to CUDA 9.0]: 8.0


Please specify the location where CUDA 8.0 toolkit is installed. Refer to README.md for more details. [Default is
/usr/local/cuda]:


Please specify the cuDNN version you want to use. [Leave empty to default to cuDNN 7.0]: 6.0
Please specify the location where cuDNN 6 library is installed. Refer to README.md for more details. [Default is
/usr/local/cuda]:
Do you wish to build TensorFlow with TensorRT support? [y/N]:
No TensorRT support will be enabled for TensorFlow.

Please specify a list of comma-separated Cuda compute capabilities you want to build with.
You can find the compute capability of your device at: https://developer.nvidia.com/cuda-gpus.
Please note that each additional compute capability significantly increases your build time and binary size. [Default is:
6.0,6.0]

Do you want to use clang as CUDA compiler? [y/N]:
nvcc will be used as CUDA compiler.

Please specify which gcc should be used by nvcc as the host compiler. [Default is /usr/bin/gcc]:

Do you wish to build TensorFlow with MPI support? [y/N]:
No MPI support will be enabled for TensorFlow.

Please specify optimization flags to use during compilation when bazel option "--config=opt" is specified [Default is
-march=native]:

Add "--config=mkl" to your bazel command to build with MKL support.
Please note that MKL on MacOS or windows is still not supported.
If you would like to use a local MKL instead of downloading, please set the environment variable "TF_MKL_ROOT" every time
before build.
Configuration finished
```

8. Build TensorFlow with bazel:

```
bazel build --config=opt //tensorflow/tools/pip_package:build_pip_package
# the option --incompatible_load_argument_is_label=false might be needed for successful compilation
```

9. If a session of the CDSW is target, upgrade dask package if asked for:

```
pip install --upgrade --proxy <proxy-host>:<proxy-port> dask
```

10. Create TensorFlow Python package in wheel format

```
bazel-bin/tensorflow/tools/pip_package/build_pip_package $HOME/tmp/tensorflow_pkg
```

11. TensorFlow Python package can now be found in $HOME/tmp/tensorflow_pkg. Install wheel:

```
pip install --user $HOME/tmp/tensorflow_pkg/tensorflow-1.4.0-cp27-none-linux_x86_64.whl
```

12. If a session of the CDSW is the target, stop session and start a new one before using the newly installed TensorFlow

Do not build TensorFlow on a Hadoop node and then install the resulting Python package to the CDSW as the operating systems differ.

Keras with TensorFlow backend

This section shows how to install Keras with TensorFlow backend and run an example.

Install Keras following these steps:

1. Install python package for TensorFlow as described before, i.e. either Python module tensorflow or tensorflow-gpu depending on the presence of a GPU on the server.
2. Install Keras with pip:

```
pip install --user Keras
```

3. It may be necessary to install the package h5py

```
pip install --user h5py
```

4. It may also be necessary to upgrade the package dask

```
pip install --user --upgrade dask
```

Run a Keras example following these steps:

1. If a session of the CDSW is the target, then open a terminal window in a session
2. Clone Keras Github repository to get the examples:

```
mkdir repos
cd repos
git clone https://github.com/fchollet/keras.git keras
```

3. Check installed Keras version:

```
pip list | grep Keras
```

4. Set clone to suitable Keras version matching installed Keras:

```
cd keras
git checkout 2.1.2
```

5. If a CDSW session is the target
   a. Open an example, e.g. cifar10_cnn.py in directory $HOME/repos/keras/examples
   b. Run example.

6. If a Hadoop node is the target
   a. Run python command with example script as parameter, e.g. $HOME/repos/keras/examples/cifar10_cnn.py:

```
python $HOME/repos/keras/examples/cifar10_cnn.py
```

7. The output may indicate that TensorFlow is not optimized which is expected as the default TensorFlow package is not optimized.

If you use the CDSW on a server with GPU, you can install a GPU enabled TensorFlow, but run a session without GPU. All python scripts using TensorFlow will then issue a warning that there is no GPU, but will continue their work on the CPUs.

In order to programmatically disable GPU utilization, insert in python source before importing Keras/TensorFlow the following code:

```
import os
os.environ["CUDA_DEVICE_ORDER"] = "PCI_BUS_ID"   # see issue #152
os.environ["CUDA_VISIBLE_DEVICES"] = ""
```

If Keras uses TensorFlow as a backend, then a Keras Python script can be modified so that the TensorFlow backend writes events which can be displayed by TensorBoard. This example shows how to modify the Keras example cifar10_cnn.py so that it issues events suitable for display by TensorBoard.

Open the file cifar10_cnn.py and make the following changes:

1. Import TensorBoard by inserting a line

```
from keras.callbacks import TensorBoard
```

2. Create a TensorBoard object writing events to subdirectory log

```
tensorboard = TensorBoard(log_dir='./logs', histogram_freq=1,
                          write_graph=True, write_images=False)
```

3. Insert the TensorBoard object as callback

```
model.fit_generator(datagen.flow(x_train, y_train,
                                 batch_size=batch_size),
                                 steps_per_epoch=x_train.shape[0] // batch_size,
                                 epochs=epochs,
                                 callbacks=[tensorboard],
                                 validation_data=(x_test, y_test),
                                 workers=4)
```

A run with TensorBoard feature enabled may yield a lot of log data; check the amount of log data during training.

TensorFlowOnSpark

The webpage https://github.com/yahoo/TensorFlowOnSpark/wiki/GetStarted_YARN gives instructions for running TensorFlowOnSpark with YARN. The described way looks promising as it does not require root access on any server, but following these instructions did not succeed.

Thus, we promote a solution that requires root access to all data nodes of the cluster, but which is successful. The following example shows how to install TensorFlowOnSpark on data nodes which are not (yet) provided with GPUs. Switching to GPUs will require installing the Python module tensorflow-gpu instead of tensorflow.

Install pip on all data nodes if not yet present:

1. Use root accounts for the following steps
2. On each data node of the Hadoop cluster check whether the command pip is installed:

```
type pip
```

3. If pip is not present in your version of Python, get python2-pip RPM package from CentOS 7 Epel, e.g. http://dl.fedoraproject.org/pub/epel/7/x86_64/Packages/p/python2-pip-8.1.2-5.el7.noarch.rpm

```
curl -O http://dl.fedoraproject.org/pub/epel/7/x86_64/Packages/p/python2-pip-8.1.2-5.el7.noarch.rpm
```

4. Copy rpm file to all data nodes of the Hadoop cluster
5. On each data node install python2-pip with yum

```
yum install python2-pip-8.1.2-5.el7.noarch.rpm
```

Install TensorFlow and TensorFlowOnSpark as follows:

1. Use root accounts for the following steps
2. On each data node install TensorFlow. Set proxy if necessary. This step may install a lot of required packages.

```
pip install tensorflow
```

3. Install the module tensorflowonspark in the same way:

```
pip install tensorflowonspark
```

The next part shows how to run the an example using the TensorFlow TF-slim library and the cifar10 dataset:

1. Use any account which has a user specific directory in HDFS /user

2. Clone the TensorFlowOnSpark github repository in order to get the example scripts:

```
mkdir -p $HOME/repos
cd $HOME/repos
git clone https://github.com/yahoo/TensorFlowOnSpark.git
```

3. Get dataset with script examples/slim/download_and_convert.py from TensorFlowOnSpark cloned repository:

```
mkdir $HOME/cifar10-tsfos; python $HOME/repos/TensorFlowOnSpark/examples/slim/download_and_convert.py -dataset_name
cifar10 -dataset_dir $HOME/cifar10-tsfos
```

4. Adapt some code locations in examples/slim/train_image_classifier.py to make it run without GPU (no GPUs results in a devision by zero):

```
232   FLAGS = tf.app.flags.FLAGS
233   # FLAGS.job_name = ctx.job_name # there is no such flag, thus remove it or create it
234   FLAGS.task = ctx.task_index
235   # FLAGS.num_clones = FLAGS.num_gpus # do not overwrite number of clones with 0 GPUs
```

5. Call TensorFlowOnSpark/examples/slim/train_image_classifier.py in a shell script similar to the one on the webpage:

```
export TFoS_HOME=$HOME/repos/TensorFlowOnSpark
export DATASET_DIR=/user/${USER}/cifar10-tsfos
export TRAIN_DIR=/user/${USER}/slim_train

## # download and convert cifar10 dataset to TFRecords
## python ${TFoS_HOME}/examples/slim/download_and_convert_data.py --dataset_name cifar10 --dataset_dir
$HOME/cifar10-tsfos
##
hadoop fs -mkdir ${DATASET_DIR}
hadoop fs -copyFromLocal $HOME/cifar10-tsfos/cifar10_*.tfrecord /user/${USER}/cifar10-tsfos

# zip TensorFlowOnSpark slim example and copy to HDFS
# as a pure Python module can be used from HDFS
pushd ${TFoS_HOME}/examples/slim; zip -r ~/slim.zip .; popd
hadoop fs -put slim.zip

export PYTHON_ROOT=/usr
export LD_LIBRARY_PATH=${PATH}
export PYSPARK_PYTHON=${PYTHON_ROOT}/bin/python
export PYSPARK_DRIVER_PYTHON=${PYTHON_ROOT}/bin/python
export SPARK_YARN_USER_ENV="PYSPARK_PYTHON=${PYTHON_ROOT}/bin/python"
export PATH=${PYTHON_ROOT}/bin/:$PATH
#export QUEUE=gpu

# set paths to libjvm.so, libhdfs.so, and libcuda*.so
export LIB_HDFS=/opt/cloudera/parcels/CDH/lib64              # path to libhdfs.so, for TF acccess to HDFS
export JAVA_HOME=/usr/java/jdk1.8.0_162
export LIB_JVM=$JAVA_HOME/jre/lib/amd64/server               # path to libjvm.so
#export LIB_CUDA=/usr/local/cuda/lib64                       # for GPUs only

# for CPU mode:
export QUEUE=default
# remove references to $LIB_CUDA

# hadoop fs -rm -r slim_train
export NUM_GPU=2
export MEMORY=$((NUM_GPU * 27))
# use SPARK 2
/usr/bin/spark2-submit \
--master yarn \
--deploy-mode cluster \
--queue ${QUEUE} \
--num-executors 4 \
--executor-memory ${MEMORY}G \
--py-files slim.zip \
--conf spark.dynamicAllocation.enabled=false \
--conf spark.yarn.maxAppAttempts=1 \
--conf spark.ui.view.acls=* \
--conf spark.executorEnv.LD_LIBRARY_PATH=$LIB_JVM:$LIB_HDFS \
--conf spark.yarn.appMasterEnv.PYSPARK_PYTHON=/usr/bin/python \
--conf spark.yarn.appMasterEnv.PYSPARK_DRIVER_PYTHON=/usr/bin/python \
${TFoS_HOME}/examples/slim/train_image_classifier.py \
--dataset_dir hdfs://default${DATASET_DIR} \
--train_dir hdfs://default${TRAIN_DIR} \
--dataset_name cifar10 \
--dataset_split_name train \
--model_name inception_v3 \
--max_number_of_steps 1000 \
--batch_size 32 \
--num_ps_tasks 1 \
--num_gpus 0

# add or replace the following options if you use GPUs
# --num_gpus ${NUM_GPU} \
# --conf spark.executorEnv.LD_LIBRARY_PATH="$LIB_CUDA:$LIB_JVM:$LIB_HDFS" \
# --driver-library-path="$LIB_CUDA" \
#
```

This training example successfully ran on a Hadoop cluster with 4 data nodes but without GPUs.

Appendix

References

[1] https://deeplearning4j.org/spark#gpusspark
[2] https://community.cloudera.com/t5/Batch-Processing-and-Workflow/Node-Labels/td-p/37044
[3] https://thenewstack.io/docker-hadoop-theres-good-bad-ugly
[4] https://issues.apache.org/jira/browse/SPARK-3785
[5] https://blog.cloudera.com/blog/2017/04/deep-learning-frameworks-on-cdh-and-cloudera-data-science-workbench
[6] http://blog.cloudera.com/blog/2017/06/deep-learning-on-apache-spark-and-hadoop-with-deeplearning4j
[7] https://blog.cloudera.com/blog/2017/07/prophecy-fulfilled-keras-and-cloudera-data-science-workbench
[8] https://www.cloudera.com/documentation/data-science-workbench/latest/topics/cdsw_gpu.html
[9] https://hortonworks.com/blog/distributed-tensorflow-assembly-hadoop-yarn
[10] https://community.hortonworks.com/articles/95570/distributed-training-of-neural-networks-on-gpus.html
[11] https://mapr.com/blog/distributed-deep-learning-caffe-using-mapr-cluster
[12] https://mapr.com/blog/deep-learning-qss-1
https://mapr.com/blog/deep-learning-qss-2
[13] https://arxiv.org/abs/1511.06435v3, Soheil Bahrampour, Naveen Ramakrishnan, Lukas Schott, Mohak Shah, "Comparative Study of Deep Learning Software Frameworks", March 30 2016
[14] https://arxiv.org/abs/1608.07249v7, Shaohuai Shi, Qiang Wang, Pengfei Xu, Xiaowen Chu, “Benchmarking State-of-the-Art Deep Learning Software Tools”, February 17 2017
[15] https://en.wikipedia.org/wiki/Comparison_of_deep_learning_software
[16] https://deeplearning4j.org/compare-dl4j-torch7-pylearn
[17] https://www.anandtech.com/show/11942/intel-shipping-nervana-neural-network-processor-first-silicon-before-year-end
[18] http://scikit-learn.org/stable/modules/neural_networks_supervised.html
[19] https://arxiv.org/pdf/1711.03386.pdf
[20] https://www.cloudera.com/documentation/data-science-workbench/latest/topics/cdsw_gpu.html#enable_gpu_support
[21] https://blog.thedataincubator.com/2017/10/ranking-popular-deep-learning-libraries-for-data-science/
[22] https://devblogs.nvidia.com/tensorrt-integration-speeds-tensorflow-inference/