MLBench Documentation

Release 2018

MLBench development team

Apr 30, 2020

MLBench

1 Features **3**

2 Repositories **5**

3 Community **7**

3.1 Prerequisites . . . 7
3.2 Installation . . . 8
3.3 Component Overview . . . 13
3.4 Benchmarking Tasks . . . 14
3.5 Developer Guide . . . 19
3.6 Contributing . . . 19
3.7 Change Log . . . 21
3.8 Authors . . . 26
3.9 Indices and tables . . . 27

Bibliography **29**

A public and reproducible collection of reference implementations and benchmark suite for distributed machine learning algorithms, frameworks and systems.

- Project website: https://mlbench.github.io/
- Free software: Apache Software License 2.0
- Documentation: https://mlbench.readthedocs.io.

CHAPTER 1

Features

- For reproducibility and simplicity, we currently focus on standard **supervised ML**, including standard deep learning tasks as well as classic linear ML models.
- We provide **reference implementations** for each algorithm, to make it easy to port to a new framework.
- Our goal is to benchmark all/most currently relevant **distributed execution frameworks**. We welcome contributions of new frameworks in the benchmark suite.
- We provide **precisely defined tasks** and datasets to have a fair and precise comparison of all algorithms, frameworks and hardware.
- Independently of all solver implementations, we provide universal **evaluation code** allowing to compare the result metrics of different solvers and frameworks.
- Our benchmark code is easy to run on **public clouds**.

CHAPTER 2

Repositories

MLBench consists of 5 Github repositories:

- Documentation: http://github.com/mlbench/mlbench-docs
- Helm Charts for Kubernetes: http://github.com/mlbench/mlbench-helm
- Python Core Library: http://github.com/mlbench/mlbench-core
- Closed-Division Benchmark Implementations: http://github.com/mlbench/mlbench-benchmarks
- Dashboard: http://github.com/mlbench/mlbench-dashboard

CHAPTER 3

Community

About us: See *Authors*

Mailing list: https://groups.google.com/d/forum/mlbench

Contact Email: mlbench-contact@googlegroups.com

3.1 Prerequisites

3.1.1 Kubernetes

mlbench uses Kubernetes as basis for the distributed cluster. This allows for easy and reproducible installation and use of the framework on a multitude of platforms.

Since mlbench manages the setup of nodes for experiments and uses Kubernetes to monitor the status of worker pods, it needs to be installed with a service-account that has permission to manage and monitor `Pods` and `StatefulSets`.

Additionally, *helm* requires a kubernetes user account with the `cluster-admin` role to deploy applications to a kubernetes cluster.

Google Cloud

For Google Cloud see: Creating a Cluster and Kubernetes Quickstart.

Cluster setup in Google Cloud is handled by running a setup script detailed in the Installation section. Concerning this, please refer to *Google Cloud and Cluster Setup*.

If you're planning to use GPUs in your cluster, see the GPUs article, especially the "Installing NVIDIA GPU device drivers" section.

When creating a GKE cluster, make sure to use version `1.10.9` or above of kubernetes, as there is an issue with DNS resolution in earlier version. You can do this with the `--cluster-version=1.10.9` flag for the `gcloud container clusters create` command.

Make sure credentials for your cluster are installed correctly (use the correct zone for your cluster):

```
gcloud container clusters get-credentials ${CLUSTER_NAME} --zone us-central1-a
```

3.1.2 Helm

Helm charts are like recipes to install Kubernetes distributed applications. They consist of templates with some logic that get rendered into Kubernetes deployment *.yaml* files They come with some default values, but also allow users to override those values.

Helm can be found here

Helm needs to be set up with service-account with `cluster-admin` rights:

```
kubectl --namespace kube-system create sa tiller
kubectl create clusterrolebinding tiller --clusterrole cluster-admin --
↪serviceaccount=kube-system:tiller
helm init --service-account tiller
```

3.2 Installation

Make sure to read *Prerequisites* before installing mlbench.

All guides assume you have checked out the mlbench-helm github repository and have a terminal open in the checked-out `mlbench-helm` directory.

3.2.1 Google Cloud and Cluster Setup

This project provides a script to make all the Google Cloud and Cluster setup. In order to do so, please run the following commands:

```
$ ./google_cloud_setup.sh create-cluster
$ ./google_cloud_setup.sh install-chart
```

To delete cluster and cleanup:

```
$ ./google_cloud_setup.sh delete-cluster
```

To uninstall chart:

```
$ ./google_cloud_setup.sh uninstall-chart
```

For general information on the available commands, please run:

```
$ ./google_cloud_setup.sh help
```

3.2.2 Helm Chart values

Since every Kubernetes is different, there are no reasonable defaults for some values, so the following properties have to be set. You can save them in a yaml file of your chosing. This guide will assume you saved them in *myvalues.yaml*. For a reference file for all configurable values, you can copy the *values.yaml* file to *myvalues.yaml*.

```
limits:
  workers:
  cpu:
  gpu:

gcePersistentDisk:
  enabled:
  pdName:
```

- `limits.workers` is the maximum number of worker nodes available to mlbench. This sets the maximum number of nodes that can be chosen for an experiment in the UI. By default mlbench starts 2 workers on startup.
- `limits.cpu` is the maximum number of CPUs (Cores) available on each worker node. Uses Kubernetes notation (*8* or *8000m* for 8 cpus/cores). This is also the maximum number of Cores that can be selected for an experiment in the UI
- `limits.gpu` is the number of gpus requested by each worker pod.
- `gcePersistentDisk.enabled` create resources related to NFS persistentVolume and persistentVolumeClaim.
- `gcePersistentDisk.pdName` is the name of persistent disk existed in GKE.

Caution: If you set `workers`, `cpu` or `gpu` higher than available in your cluster, Kubernetes will not be able to allocate nodes to mlbench and the deployment will hang indefinitely, without throwing an exception. Kubernetes will just wait until nodes that fit the requirements become available. So make sure your cluster actually has the requirements avilable that you requested.

Note: To use `gpu` in the cluster, the nvidia device plugin should be installed. See *Plugins* for details

Note: Use commands like `gcloud compute disks create --size=10G --zone=europe-west1-b my-pd-name` to create persistent disk.

Note: The GCE persistent disk will be mounted to */datasets/* directory on each worker.

3.2.3 Basic Install

Set the *Helm Chart values*

Use helm to install the mlbench chart (Replace `${RELEASE_NAME}` with a name of your choice):

```
$ helm upgrade --wait --recreate-pods -f values.yaml --timeout 900 --install $
↪{RELEASE_NAME} .
```

Follow the instructions at the end of the helm install to get the dashboard URL. E.g.:

```
$ helm upgrade --wait --recreate-pods -f values.yaml --timeout 900 --install rel .
  [...]
  NOTES:
```

(continues on next page)

(continued from previous page)

```
1. Get the application URL by running these commands:
    export NODE_PORT=$(kubectl get --namespace default -o jsonpath="{.spec.ports[0].
↪nodePort}" services rel-mlbench-master)
    export NODE_IP=$(kubectl get nodes --namespace default -o jsonpath="{.items[0].
↪status.addresses[0].address}")
    echo http://$NODE_IP:$NODE_PORT
```

This outputs the URL the Dashboard is accessible at.

Plugins

In `values.yaml`, one can optionally install Kubernetes plugins by turning on/off the following flags:

- `weave.enabled`: If true, install the weave network plugin.
- `nvidiaDevicePlugin.enabled`: If true, install the nvidia device plugin.

3.2.4 Google Cloud / Google Kubernetes Engine

Set the *Helm Chart values*

Important: Make sure to read the prerequisites for *Google Cloud*

Please make sure that `kubectl` is configured correctly.

Caution: Google installs several pods on each node by default, limiting the available CPU. This can take up to 0.5 CPU cores per node. So make sure to provision VM's that have at least 1 more core than the amount of cores you want to use for you mlbench experiment. See here for further details on node limits.

Install mlbench (Replace `${RELEASE_NAME}` with a name of your choice):

```
$ helm upgrade --wait --recreate-pods -f values.yaml --timeout 900 --install $
↪{RELEASE_NAME} .
```

To access mlbench, run these commands and open the URL that is returned (**Note**: The default instructions returned by *helm* on the commandline return the internal cluster ip only):

```
$ export NODE_PORT=$(kubectl get --namespace default -o jsonpath="{.spec.ports[0].
↪nodePort}" services ${RELEASE_NAME}-mlbench-master)
$ export NODE_IP=$(gcloud compute instances list|grep $(kubectl get nodes --namespace␣
↪default -o jsonpath="{.items[0].status.addresses[0].address}") |awk '{print $5}')
$ gcloud compute firewall-rules create --quiet mlbench --allow tcp:$NODE_PORT,tcp:
↪$NODE_PORT
$ echo http://$NODE_IP:$NODE_PORT
```

Danger: The last command opens up a firewall rule to the google cloud. Make sure to delete the rule once it's not needed anymore:

```
$ gcloud compute firewall-rules delete --quiet mlbench
```

3.2.5 Minikube

Minikube allows running a single-node Kubernetes cluster inside a VM on your laptop, for users looking to try out Kubernetes or to develop with it.

Installing mlbench to minikube.

Set the *Helm Chart values*

Start minikube cluster

```
$ minikube start
```

Next install or upgrade a helm chart with desired configurations with name *${RELEASE_NAME}*

```
$ helm init --kube-context minikube --wait
$ helm upgrade --wait --recreate-pods -f myvalues.yaml --timeout 900 --install $
↪{RELEASE_NAME} .
```

Note: The minikube runs a single-node Kubernetes cluster inside a VM. So we need to fix the `replicaCount=1` in *values.yaml*.

Once the installation is finished, one can obtain the url

```
$ export NODE_PORT=$(kubectl get --namespace default -o jsonpath="{.spec.ports[0].
↪nodePort}" services ${RELEASE_NAME}-mlbench-master)
$ export NODE_IP=$(kubectl get nodes --namespace default -o jsonpath="{.items[0].
↪status.addresses[0].address}")
$ echo http://$NODE_IP:$NODE_PORT
```

Now the mlbench dashboard should be available at `http://${NODE_IP}:${NODE_PORT}`.

Note: To access `http://$NODE_IP:$NODE_PORT` outside minikube, run the following command on the host:

```
$ ssh -i ${MINIKUBE_HOME}/.minikube/machines/minikube/id_rsa -N -f -L localhost:$
↪{NODE_PORT}:${NODE_IP}:${NODE_PORT} docker@$(minikube ip)
```

where `$MINIKUBE_HOME` is by default `$HOME`. One can view mlbench dashboard at `http://localhost:${NODE_PORT}`

3.2.6 Docker-in-Docker (DIND)

Docker-in-Docker allows simulating multiple nodes locally on a single machine. This is useful for development.

Hint: For development purposes, it makes sense to use a local docker registry as well with DIND.

Describing how to set up a local registry would be too long for this guide, so here are some pointers:

- You can find a guide here.
- This page details setting up an image pull secret.
- This details adding an image pull secret to a kubernetes service account.

- You can use `dind-proxy.sh` in the mlbench repository to forward the registry port (5000) to kubernetes DIND.

Download the kubeadm-dind-cluster script.

```
$ wget https://cdn.rawgit.com/kubernetes-sigs/kubeadm-dind-cluster/master/fixed/dind-
↪cluster-v1.11.sh
$ chmod +x dind-cluster-v1.11.sh
```

For networking to work in DIND, we need to set a CNI Plugin. In our experience, `weave` works well with DIND.

```
$ export CNI_PLUGIN=weave
```

Now we can start the local cluster with

```
$ ./dind-cluster-v1.11.sh up
```

This might take a couple of minutes.

Hint: If you're using a local docker registry, run `dind-proxy.sh` after the previous step.

Install `helm` (See *Prerequisites*) and set the *Helm Chart values*.

Hint: For a local registry, make sure you have an `imagePullSecret` added to the kubernetes serviceaccount and set the repository and secret in the `values.yaml` file (`regcred` in this example):

```
master:
  imagePullSecret: regcred

  image:
    repository: localhost:5000/mlbench_master
    tag: latest
    pullPolicy: Always


worker:
  imagePullSecret: regcred

  image:
    repository: localhost:5000/mlbench_worker
    tag: latest
    pullPolicy: Always
```

Install mlbench (Replace `${RELEASE_NAME}` with a name of your choice):

```
$ helm upgrade --wait --recreate-pods -f values.yaml --timeout 900 --install rel .
  [...]
  NOTES:
  1. Get the application URL by running these commands:
     export NODE_PORT=$(kubectl get --namespace default -o jsonpath="{.spec.ports[0].
↪nodePort}" services rel-mlbench-master)
     export NODE_IP=$(kubectl get nodes --namespace default -o jsonpath="{.items[0].
↪status.addresses[0].address}")
     echo http://$NODE_IP:$NODE_PORT
```

Run the 3 commands printed by the last command. This outputs the URL the Dashboard is accessible at.

3.3 Component Overview

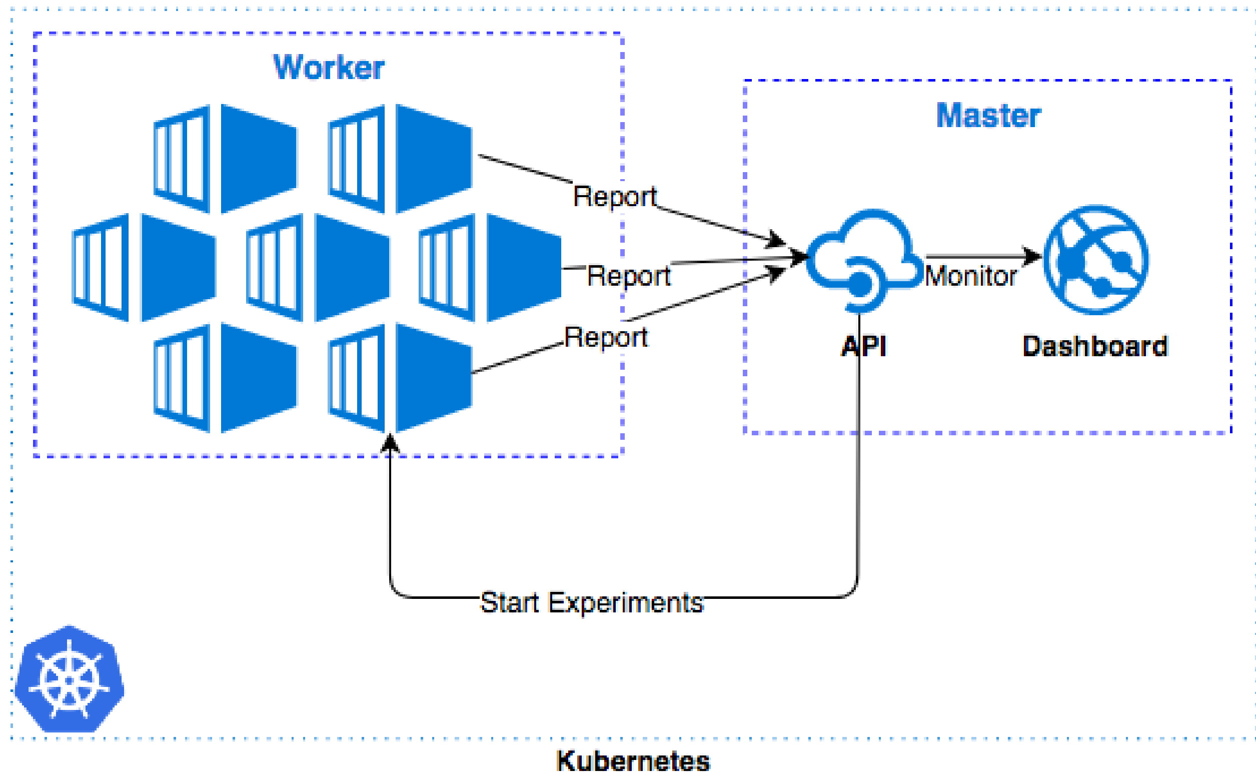


Fig. 1: Deployment Overview: Relation between Worker and Master

mlbench consists of two components, the **Master** and the **Worker** Docker containers.

3.3.1 Master

The master contains the Dashboard, the main interface for the project. The dashboard allows you to start and run a distributed ML experiment and visualizes the progress and result of the experiment. It allows management of the mlbench nodes in the Kubernetes cluster and for most users constitutes the sole way they interact with the mlbench project.

It also contains a REST API that can be use instead of the Dashboard, as well as being used for receiving data from the Dashboard.

The Master also manages the `StatefulSet` of worker through the Kubernetes API.

The code for the Master can be found in the mlbench-dashboard repository.

3.3.2 Worker

The worker images contain all the boilerplate code needed for a distributed ML model as well as the actual model code. There is a docker image for each Benchmark implementation. They take care of training the distributed model, with some configuration supplied by the Master.

Worker nodes send status information to the metrics API of the Master to inform it of the progress and state of the current run.

All official reference implementations along with useful base images can be found in the mlbench-benchmarks repository.

3.3.3 mlbench-core

mlbench-core is a Python package that contains functionality to interact with the Master node, such as writing metrics. It also contains common code used across multiple benchmark implementations and implementation independent helper functions.

The code can be found in the mlbench-core repository.

3.3.4 Helm Chart

The Helm chart allows automated installation of the MLBench framework into a Kubernetes cluster.

It can be found in the mlbench-helm repository.

3.4 Benchmarking Tasks

The results of the benchmarks can be found here: *benchmark-task-results*

3.4.1 Benchmark Divisions

There are two divisions of benchmarking, the closed one which is restrictive to allow fair comparisons of specific training algorithms and systems, and the open divisions, which allows users to run their own models and code while still providing a reasonably fair comparison.

Closed Division

The Closed Division encompasses several subcategories to compare different dimensions of distributed machine learning. We provide precise reference implementations of each algorithm, including the communication patterns, such that they can be implemented strictly comparable between different hardware and software frameworks.

The two basic metrics for comparison are *Accuracy after Time* and *Time to Accuracy* (where accuracy will be test and/or training accuracy)

Variable dimensions in this category include:

- Algorithm - limited number of prescribed standard algorithms, according to strict reference implementations provided
- Hardware - GPU - CPU(s) - Memory
- Scalability - Number of workers

- Network - Impact of bandwidth and latency

Accuracy after Time

The system has a certain amount ot time for training (2 hours) and at the end, the accuracy of the final model is evaluated. The higher the better

Time to Accuracy

A certain accuracy, e.g. 97% is defined for a task and the training time of the system until that accuracy is reached is measured. The shorter the better.

Open Division

The Open Division allows you to implement your own algorithms and training tricks and compare them to other implementations. There's little limit to what can be changed by you and as such, it is up to you to make sure that comparisons are fair.

In this division, mlbench merely provides a platform to easily perform and measure distributed machine learning experiments in a standardized way.

The Open Division is **a work in progress** and as such not yet publicly available.

3.4.2 Benchmark Task Descriptions

We here provide precise descriptions of the official benchmark tasks. The task are selected to be representative of relevant machine learning workloads in both industry and in the academic community. The main goal here is a fair, reproducible and precise comparison of most state-of-the-art algorithms, frameworks, and hardware.

For each task, we provide a reference implementation, as well as benchmark metrics and results for different systems.

1a. Image Classification (ResNet, CIFAR-10)

Image classification is one of the most important problems in computer vision and a classic example of supervised machine learning.

1. **Model** We benchmark two model architectures of Deep Residual Networks (ResNets) based on prior work by He et al. The first model (m1) is based on the ResNets defined in this paper. The second version (m2) is based on the ResNets defined here. For each version we have the network implementations with 20, 32, 44, and 56 layers.

   TODO: only benchmark two most common architectures say (can support more, but they are not part of the official benchmark task)

2. **Dataset** The CIFAR-10 dataset containing a set of images used to train machine learning and computer vision models. It contains 60,000 32x32 color images in 10 different classes, with 6000 images per class. The 10 different classes represent airplanes, cars, birds, cats, deer, dogs, frogs, horses, ships, and trucks.

   The train / test split as provided in the dataset is used. The test dataset contains 10,000 imagest with exactly 1000 randomly-selected images per each class. The rest 50,000 images are training samples.

3. **Training Algorithm** We use standard synchronous SGD as the optimizer (that is distributed mini-batch SGD with synchronous all-reduce communication after each mini-batch).

- number of machines k: 2, 4, 8, 16, 32
- minibatch size per worker b: 32
- maximum epochs: 164
- learning rate
  - learning rate per sample η : 0.1 / 256
  - decay: similar to Deep Residual Learning for Image Recognition, we reduce learning rate by 1/10 at the 82-th and 109-th epoch.
  - scaling and warmup: apply `linear scaling rule` mentioned in [GDollarG+17]. The learning rate per worker is scaled from $\eta \times b$ to $\eta \times b \times k$ within the first 5 epochs.
- momentum: 0.9
- nesterov: True
- weight decay: 0.0001

Besides, in each round workers access disjoint set of datapoints.

Implementation details:

1. **Data Preprocessing** We followed the same approach as mentioned here.
2. **Selection of Framework & Systems** We aim to provide the same algorithm in multiple frameworks, primarily focussing on PyTorch and Tensorflow. For the systems, kubernetes allows easy transferability of our code. While initial results reported are from google kubernetes engine, AWS will be supported very soon.
3. **Environments for Scaling Task** For the scaling task, we use n1-standard-4 type instances with 50GB disk size. There is only one worker per node; each worker uses 2.5 cpus. The bandwidth between two nodes is around 7.5Gbit/s. Openmpi is used for communication. No accelerators are used for this task.

1b. Image Classification (ResNet, ImageNet)

TODO (again synchr SGD as main baseline)

2a. Linear Learning (Logistic Regression, epsilon)

1. **Model** We benchmark Logistic Regression with L2 regularization.
2. **Dataset** The epsilon dataset is an artificial and dense dataset which is used for Pascal large scale learning challenge in 2008. It contains 400,000 training samples and 100,000 test samples with 2000 features.
3. **Training Algorithm** We use standard synchronous SGD as the optimizer (that is distributed mini-batch SGD with synchronous all-reduce communication after each mini-batch).
   - minibatch size per worker b: 100[1]
   - **learning rate** [$\frac{\alpha}{\sqrt{t}}$[2]] Here are the values of alpha we choose for various number of workers:

[1] Here is how we select this value: We train the model with different batch sizes ([1,..,1000]) and in the end we select the batch size that enables the trained model to reach to 89% accuracy on the validation set in less time. we use 80% of the dataset to train the model, and the remaining 20% is used as the validation set.

[2] α is tuned for each cluster size separately. To do so, we use 80% of the dataset to train the model, and the remaining 20% is used as the validation set. We do a grid search to find the best value for alpha: for each value in the grid ([0.001,..,1000]), the model is trained until it reaches to 89% accuracy on the validation set. Finally, we select the value that enables the model to reach the target accuracy value faster.

nodes	α
1	200
2	400
4	600
8	700
16, 32, 64	800

- momentum: 0
- weight decay: 0
- regularization rate $= 0.0000025$

Implementation details:

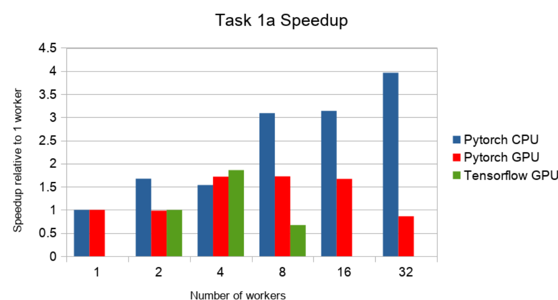
1. **Selection of Framework & Systems** While our initial reference implementation is currently PyTorch, we will aim to provide the same algorithm in more frameworks very soon, starting with Tensorflow. For the systems, kubernetes allows easy transferability of our code. While initial results reported are from google kubernetes engine, AWS will be supported very soon.
2. **Environments for Scaling Task** For the scaling task, we use n1-standard-4 type instances with 50GB disk size. There is only one worker per node; each worker uses 2.5 cpus. The bandwidth between two nodes is around 7.5Gbit/s. Openmpi is used for communication. No accelerators are used for this task.

3.4.3 Benchmark Task Results

1a. Image Classification (ResNet, CIFAR-10)

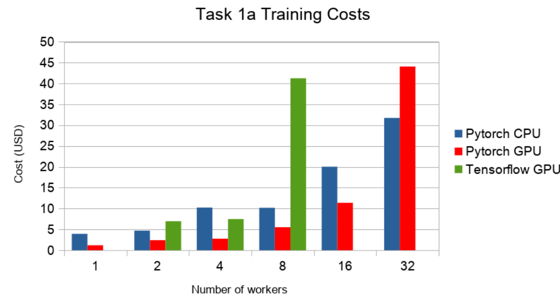
Here we present the results for scaling task. All results were generated on the Google Cloud Kubernetes Engine. *n1-standard-4* instances were used for training, with *NVIDIA® Tesla® K80* GPUs used for GPU training.

- The next figure shows the speedup in training times to 80% accuracy relative to training on one node[3]. The baseline time for 1 worker for the PyTorch CPU implementation is 5895 s, for the PyTorch GPU implementation 407 s and for the Tensorflow GPU implementation 1191 s.



- The next figure compares the cost of experiment. Note that a regular *n1-standard-4* instance costs $0.19 per hour and a preemptible one costs only $0.04. *NVIDIA® Tesla® K80* GPUs (preemtpible) cost $0.135 per hour. All costs shown are for premtible instances.

[3] Training on CPU shows speedup with increasing number of nodes up to 32 nodes. For the Pytorch implementation on the GPU, speedups plateau at 4 nodes and decrease for 32 nodes. Tensorflow GPU numbers are only available up to 8 nodes, as more nodes lead to an Out-Of-Memory error on the GPU. This benchmark is still a work in progress and this issue will be fixed in a future release. Also since Tensorflow requires at least one parameter-server and a worker to run, it can't be run on a single machine. As such, the results between PyTorch and Tensorflow are not directly comparable. Tuning the Tensorflow parameter-server in size when growing the number of total machines might require further tuning



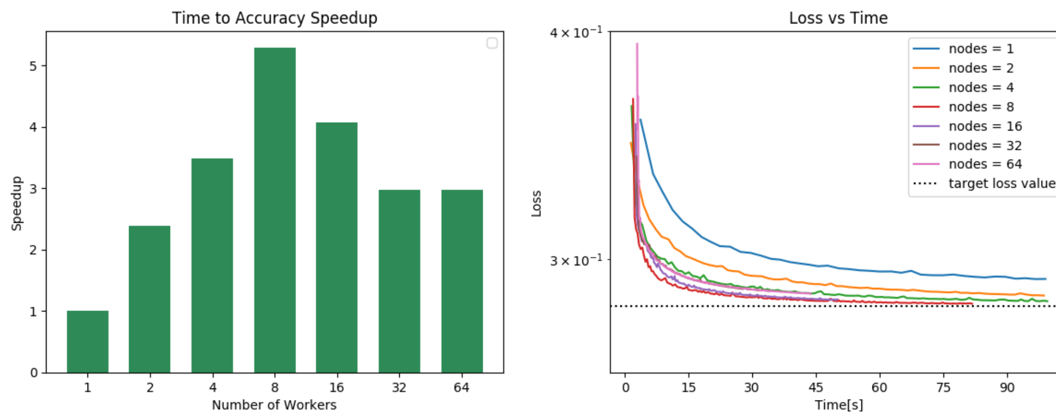
1b. Image Classification (ResNet, ImageNet)

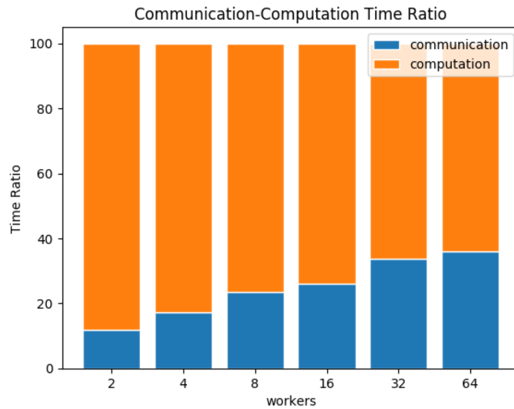
TODO

2a. Linear Learning (Logistic Regression, epsilon)

Here we present the results for the scaling task.

- First figure shows the speedup of time to accuracy, for test accuracy of 89%, as the size of the cluster increases. Even though initially the speedup grows with the number of nodes added to the cluster, the benefit starts dropping for a cluster bigger than 16 nodes. This is mostly due to the issue of large-batch training. As the local batch-size of each worker is fixed, the global batch-size increases with the number of workers. Hence, while increasing batch size up to a point makes the training faster, beyond a certain point it will no longer reduce the number of training steps required, making it slower to reach the same accuracy.
- Second figure illustrates how the loss value drops over time for various number of nodes. The black dotted line shows the target loss value, which is 0.2828 for this particular dataset.
- Last figure shows the average communication-computation time ratio for a node in the cluster. As we expected, the more workers we have, the more time is spent in communication.





3.4.4 Benchmark Task Implementations

For details on the available Benchmark implementations, please see Benchmarking Implementations .

References

3.5 Developer Guide

3.5.1 Development Workflow

Git Flow is used for features etc. This automatically handles pull requests. Make sure to install the commandline tool at the link above

3.5.2 Code Style

Python code should follow PEP8 guidelines. flake8 checks PEP8 compliance

3.6 Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

3.6.1 Types of Contributions

Report Bugs

Report bugs at https://github.com/mlbench/mlbench/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

Write Documentation

mlbench could always use more documentation, whether as part of the official mlbench docs, in docstrings, or even on the web in blog posts, articles, and such.

Submit Feedback

The best way to send feedback is to file an issue at https://github.com/mlbench/mlbench/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

3.6.2 Get Started!

Ready to contribute? Here's how to set up *mlbench* for local development.

1. Install the Prerequisites;
2. Follow the installation guide;
3. Create a branch for local development:

   ```
   $ git checkout -b name-of-your-bugfix-or-feature
   ```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

   ```
   $ flake8 mlbench tests
   $ python setup.py test or py.test
   $ tox
   ```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

   ```
   $ git add .
   $ git commit -m "Your detailed description of your changes."
   $ git push origin name-of-your-bugfix-or-feature
   ```

7. Submit a pull request through the GitHub website.

3.6.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 2.7, 3.4, 3.5 and 3.6, and for PyPy. Check https://travis-ci.org/mlbench/mlbench/pull_requests and make sure that the tests pass for all supported Python versions.

3.6.4 Tips

To run a subset of tests:

```
$ py.test tests.test_mlbench
```

3.6.5 Deploying

A reminder for the maintainers on how to deploy. Make sure all your changes are committed (including an entry in HISTORY.rst). Then run:

```
$ bumpversion patch # possible: major / minor / patch
$ git push
$ git push --tags
```

3.7 Change Log

3.7.1 v2.0.0

MLBench Helm

Implemented enhancements:

- Added GKE and AWS Setup Scripts

MLBench Benchmarks

Implemented enhancements:

- Added Goals to PyTorch Benchmark
- Updated PyTorch Tutorial code
- Changed all images to newest `mlbench-core` version.

MLBench Dashboard

Implemented enhancements:

- Added Download of Task Goals
- Fixed some performance issues

MLBench Core

[v2.0.0](https://github.com/mlbench/mlbench-core/tree/v2.0.0 (2019-06-13)

Full Changelog

3.7.2 v1.4.4

MLBench Core

v1.4.4 (2019-05-28)

Full Changelog

3.7.3 v1.4.3

MLBench Core

v1.4.3 (2019-05-23)

Full Changelog

3.7.4 v1.4.2

MLBench Core

v1.4.2 (2019-05-21)

Full Changelog

3.7.5 v1.4.1

MLBench Core

v1.4.1 (2019-05-16)

Full Changelog

3.7.6 v1.4.0

MLBench Core

v1.4.0 (2019-05-02)

Full Changelog

Implemented enhancements:

- Split Train and Validation in Tensorflow #22

3.7.7 v1.3.4

MLBench Core

v1.3.4 (2019-03-20)

Full Changelog

Implemented enhancements:

- in controlflow, don't mix train and validation #20

Fixed bugs:

- Add metrics logging for Tensorflow #19

3.7.8 v1.3.3

MLBench Core

v1.3.3 (2019-02-26)

Full Changelog

3.7.9 v1.3.2

MLBench Core

v1.3.2 (2019-02-13)

Full Changelog

3.7.10 v1.3.1

MLBench Core

v1.3.1 (2019-02-13)

Full Changelog

3.7.11 v1.3.0

MLBench Core

v1.3.0 (2019-02-12)

Full Changelog

3.7.12 v1.2.1

MLBench Core

v1.2.1 (2019-01-31)

Full Changelog

3.7.13 v1.2.0

MLBench Core

v1.2.0 (2019-01-30)

Full Changelog

3.7.14 v1.1.1

MLBench Core

v1.1.1 (2019-01-09)

Full Changelog

3.7.15 v1.1.0

MLBench Core

v1.1.0 (2018-12-06)

Full Changelog

Fixed bugs:

- Bug when saving checkpoints #13

Implemented enhancements:

- Adds Tensorflow Controlflow, Dataset and Model code
- Adds Pytorch linear models
- Adds sparsified and decentralized optimizers

MLBench Benchmarks

Implemented enhancements:

- Added Tensorflow Benchmark

MLBench Dashboard

Implemented enhancements:

- Added new Tensorflow Benchmark Image
- Remove Bandwidth limiting
- Added ability to run custom images in dashboard

MLBench Helm

Nothing

3.7.16 v1.0.0

MLBench Core

1.0.0 (2018-11-15)

Implemented enhancements:

- Add API Client to mlbench-core #6
- Move to google-style docs #4
- Add Imagenet Dataset for pytorch #3
- Move worker code to mlbench-core repo #1

3.7.17 v0.1.0

Main Repo

0.1.0 (2018-09-14)

Implemented enhancements:

- Add documentation in reference implementation to docs #46
- Replace cAdvisor with Kubernetes stats for Resource usage #38
- Rename folders #31
- Change docker image names #30
- Add continuous output for mpirun #27
- Replace SQlite with Postgres #25
- Fix unittest #23

- Add/Fix CI/Automated build #22
- Cleanup unneeded project files #21
- Remove hardcoded values #20
- Improves Notes.txt #19
- Rename components #15

Fixed bugs:

- 504 Error when downloading metrics for long runs #61

Closed issues:

- small doc improvements for first release #54
- Check mlbench works on Google Cloud #51
- learning rate scheduler #50
- Add Nvidia k8s-device-plugin to charts #48
- Add Weave to Helm Chart #41
- Allow limiting of resources for experiments #39
- Allow downloading of Run measurements #35
- Worker Details page #33
- Run Visualizations #32
- Show experiment history in Dashboard #18
- Show model progress in Dashboard #13
- Report cluster status in Dashboard #12
- Send metrics from SGD example to metrics api #11
- Add metrics endpoint for experiments #10
- Let Coordinator Dashboard start a distributed Experiment #9
- Add mini-batch SGD model experiment #8

* *This Change Log was automatically generated by ‘github_changelog_generator <https://github.com/skywinder/Github-Changelog-Generator>‘__*

3.8 Authors

3.8.1 Core Contributors

- Ralf Grubenmann
- Lie He
- Tao Lin
- Fabian Pedregosa
- Martin Jaggi

3.8.2 Contributors

- Luciana Marques

3.9 Indices and tables

- genindex
- modindex
- search

Bibliography

[GDollarG+17] Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch sgd: training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017.