mlbench Documentation

Release 0.1.1

Ralf Grubenmann

Dec 06, 2018

Contents:

DEPRECATED!!!!!!!!!!

This repository is not used anymore. The code for MLBench is now found in https://github.com/mlbench/mlbench-core https://github.com/mlbench/mlbench-benchmarks https://github.com/mlbench/mlbench-dashboard https://github.com/mlbench/mlbench-helm https://github.com/mlbench/mlbench-docs

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

Community

About us: See *Authors*

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

Contact Email: mlbench-contact@googlegroups.com

2.1 Prerequisites

2.1.1 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.

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
```

2.1.2 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
```

2.2 Installation

Make sure to read *Prerequisites* before installing mlbench.

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

2.2.1 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 *charts/mlbench/values.yaml* file to *myvalues.yaml*.

```
limits:
  workers:
  cpu:
  bandwidth:
  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.bandwidth` is the maximum network bandwidth available between workers, in mbit per second. This is the default bandwidth used and the maximum number selectable 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.

2.2.2 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} charts/mlbench
```

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␣
↪charts/mlbench
  [...]
  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
```

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.

2.2.3 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} charts/mlbench
```

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
```

Hint: If you want to build the docker images yourself and host it in the GC registry, follow these steps:

Authenticate with GC registry:

```
$ gcloud auth configure-docker
```

Build docker images (Replace **<gcloud project name>** with the name of your project):

```
$ make publish-docker component=master docker_registry=gcr.io/<gcloud project name>
$ make publish-docker component=worker docker_registry=gcr.io/<gcloud project name>
```

Use the following settings for your *myvalues.yaml* file when installing with helm:

```
master:

  image:
    repository: gcr.io/<gcloud project name>/mlbench_master
    tag: latest
    pullPolicy: Always


worker:

  image:
    repository: gcr.io/<gcloud project name>/mlbench_worker
    tag: latest
    pullPolicy: Always
```

2.2.4 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*

First build docker images and push them to private registry *localhost:5000*.

```
$ make publish-docker component=master docker_registry=localhost:5000
$ make publish-docker component=worker docker_registry=localhost:5000
```

Then start minikube cluster

```
$ minikube start
```

Use tcp-proxy to forward node's 5000 port to host's port 5000 so that one can pull images from local registry.

```
$ minikube ssh
$ docker run --name registry-proxy -d -e LISTEN=':5000' -e TALK="$(/sbin/ip route|awk
↪'/default/ { print $3 }'):5000" -p 5000:5000 tecnativa/tcp-proxy
```

Now we can pull images from private registry inside the cluster, check `docker pull localhost:5000/mlbench_master:latest`.

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} charts/mlbench
```

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}`

2.2.5 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, build and push the `master` and `worker` images:

```
$ make publish-docker component=master docker_registry=localhost:5000
$ make publish-docker component=worker docker_registry=localhost:5000
```

Also, 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
```

(continues on next page)

(continued from previous page)

```
  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␣
↪charts/mlbench
  [...]
  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.

2.3 Component Overview

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

2.3.1 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.

Dashboard

MLBench comes with a dashboard to manage and monitor the cluster and jobs.

Main Page

The main view shows all MLBench worker nodes and their current status

Runs Page

The Runs page allows you to start a new experiment on the worker nodes. You can select how many workers to use, how many CPU Cores each worker can utilize and limit the network bandwidth available to workers.

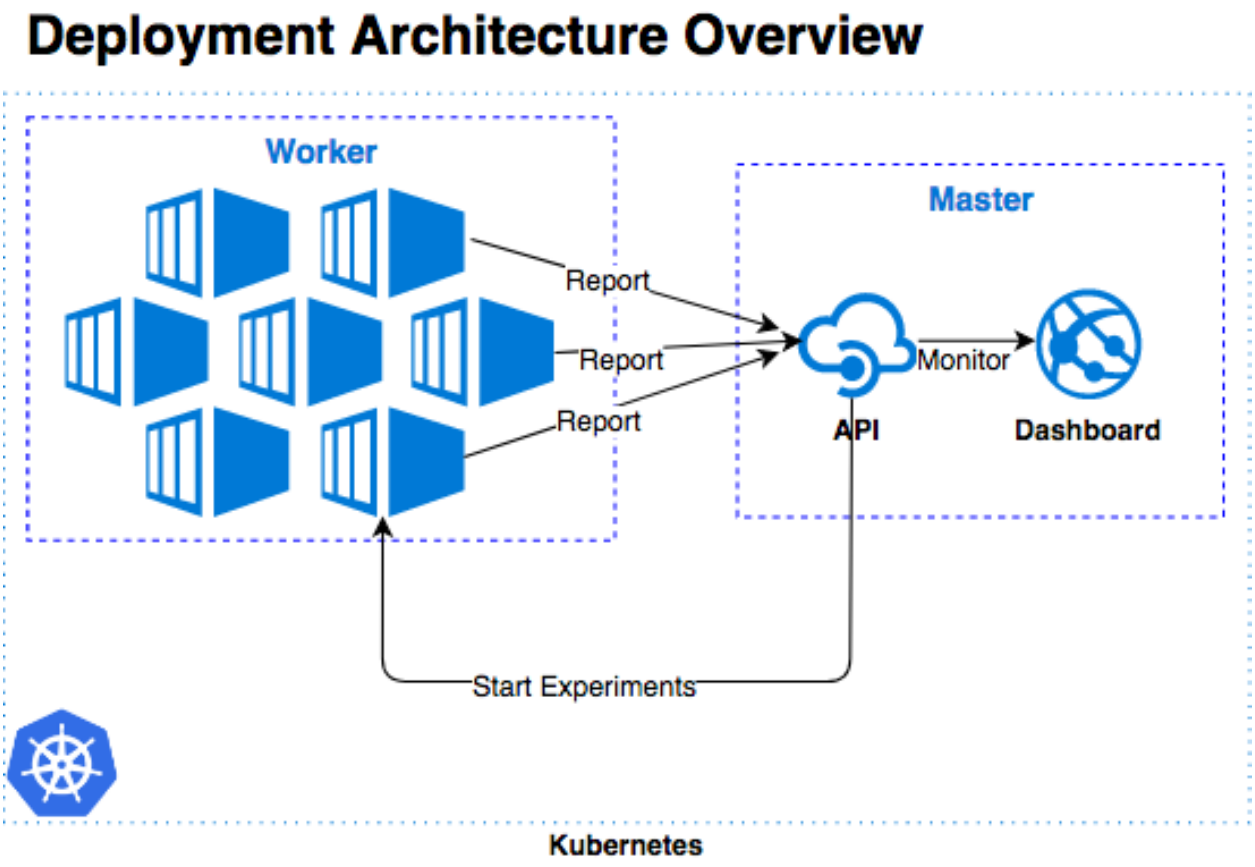


Fig. 1: Deployment Overview: Relation between Worker and Master

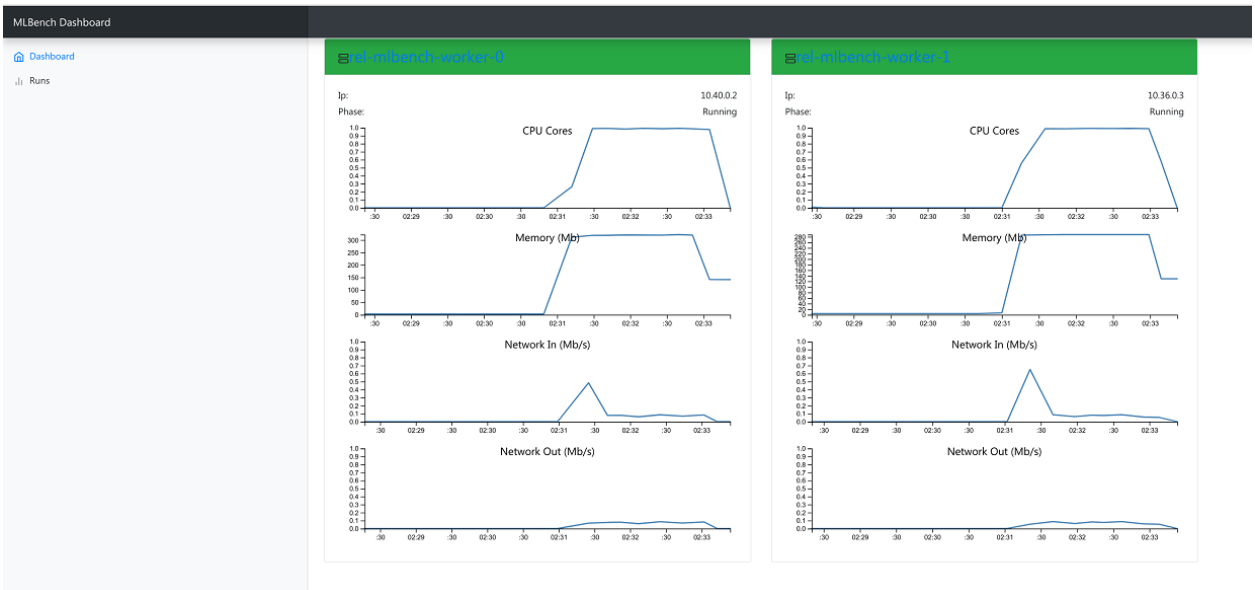


Fig. 2: Dashboard Main Page

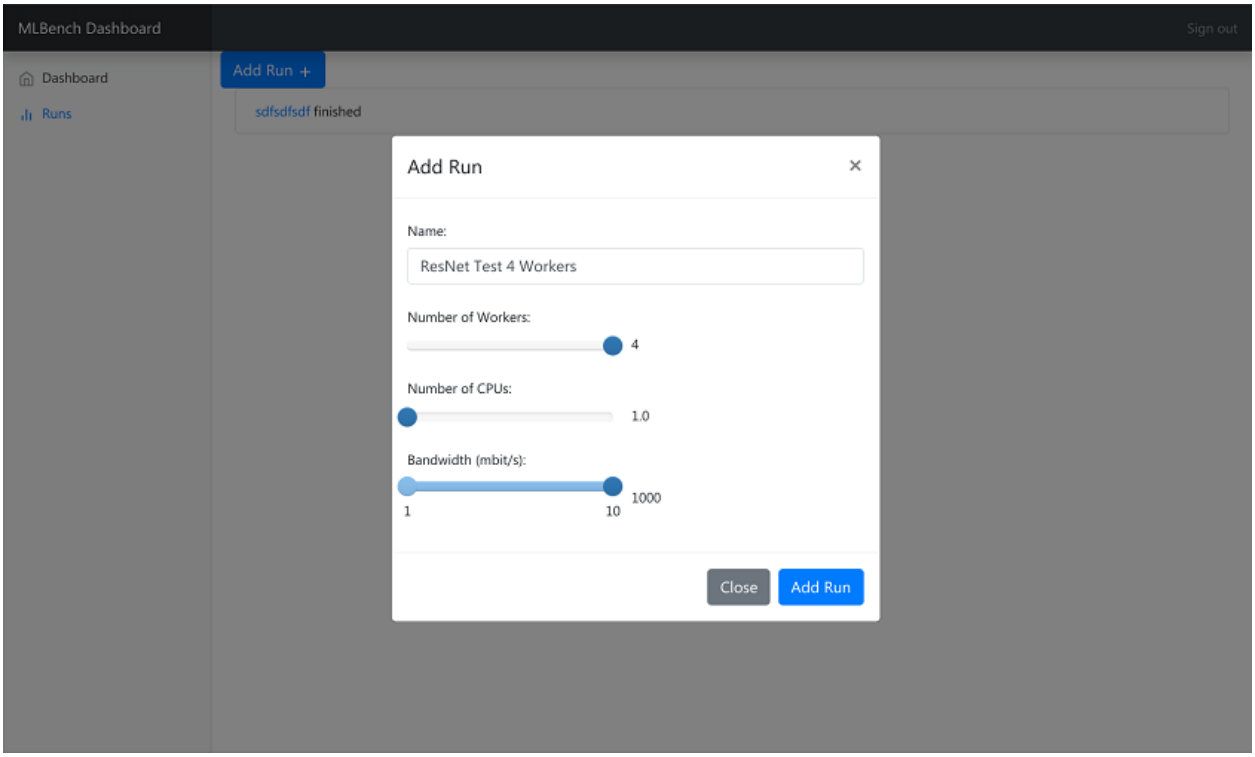


Fig. 3: Dashboard Runs Page

Run Details Page

The Run Details page shows the progress and result of an experiment. You can track metrics like `train loss` and `validation accuracy` as well as see the `stdout` and `stderr` logs of all workers.

It also allows you to download all the metrics of a run as well as resource usage of all workers participating in the run as json files.

REST API

MLBench provides a basic REST Api though which most functionality can also be used. It's accessible through the /api/ endpoints on the dashboard URL.

Pods

`GET /api/pods/`
All Worker-Pods available in the cluster, including status information

Example request:

```
GET /api/pods HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
```

Example response:

Fig. 4: Dashboard Run Details Page

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: text/javascript

[
  {
    "name":"worn-mouse-mlbench-worker-55bbdd4d8c-4mxh5",
    "labels":"{'app': 'mlbench', 'component': 'worker', 'pod-template-hash':
↪'1166880847', 'release': 'worn-mouse'}",
    "phase":"Running",
    "ip":"10.244.2.58"
  },
  {
    "name":"worn-mouse-mlbench-worker-55bbdd4d8c-bwwsp",
    "labels":"{'app': 'mlbench', 'component': 'worker', 'pod-template-hash':
↪'1166880847', 'release': 'worn-mouse'}",
    "phase":"Running",
    "ip":"10.244.3.57"
  }
]
```

Request Headers

- Accept – the response content type depends on *Accept* header

Response Headers

- Content-Type – this depends on *Accept* header of request

Status Codes

- 200 OK – no error

Metrics

GET /api/metrics/
Get metrics (Cpu, Memory etc.) for all Worker Pods

Example request:

```
GET /api/metrics HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: text/javascript

{
  "quiet-mink-mlbench-worker-0": {
      "container_cpu_usage_seconds_total": [
          {
              "date": "2018-08-03T09:21:38.594282Z",
              "value": "0.188236813"
          },
          {
              "date": "2018-08-03T09:21:50.244277Z",
              "value": "0.215950298"
          }
      ]
  },
  "quiet-mink-mlbench-worker-1": {
      "container_cpu_usage_seconds_total": [
          {
              "date": "2018-08-03T09:21:29.347960Z",
              "value": "0.149286015"
          },
          {
              "date": "2018-08-03T09:21:44.266181Z",
              "value": "0.15325329"
          }
      ],
      "container_cpu_user_seconds_total": [
          {
              "date": "2018-08-03T09:21:29.406238Z",
              "value": "0.1"
          },
          {
              "date": "2018-08-03T09:21:44.331823Z",
              "value": "0.1"
          }
      ]
  }
}
```

Request Headers

- Accept – the response content type depends on *Accept* header

Response Headers

- Content-Type – this depends on *Accept* header of request

Status Codes

- 200 OK – no error

GET /api/metrics/(str: *pod_name_or_run_id*)/
Get metrics (Cpu, Memory etc.) for all Worker Pods

Example request:

```
GET /api/metrics HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: text/javascript

{
  "container_cpu_usage_seconds_total": [
      {
          "date": "2018-08-03T09:21:29.347960Z",
          "value": "0.149286015"
      },
      {
          "date": "2018-08-03T09:21:44.266181Z",
          "value": "0.15325329"
      }
  ],
  "container_cpu_user_seconds_total": [
      {
          "date": "2018-08-03T09:21:29.406238Z",
          "value": "0.1"
      },
      {
          "date": "2018-08-03T09:21:44.331823Z",
          "value": "0.1"
      }
  ]
}
```

Query Parameters

- **since** – only get metrics newer than this date, (Default *1970-01-01T00:00:00.000000Z*)
- **metric_type** – one of *pod* or *run* to determine what kind of metric to get (Default: *pod*)

Request Headers

- Accept – the response content type depends on *Accept* header

Response Headers

- Content-Type – this depends on *Accept* header of request

Status Codes

- 200 OK – no error

POST /api/metrics

Save metrics. "pod_name" and "run_id" are mutually exclusive. The fields of metrics and their types are defined in *mlbench/api/models/kubemetrics.py*.

Example request:

```
POST /api/metrics HTTP/1.1
Host: example.com
Accept: application/json, text/javascript

{
  "pod_name": "quiet-mink-mlbench-worker-1",
  "run_id": 2,
  "name": "accuracy",
  "date": "2018-08-03T09:21:44.331823Z",
  "value": "0.7845",
  "cumulative": False,
  "metadata": "some additional data"
}
```

Example response:

```
HTTP/1.1 201 CREATED
Vary: Accept
Content-Type: text/javascript

{
  "pod_name": "quiet-mink-mlbench-worker-1",
  "name": "accuracy",
  "date": "2018-08-03T09:21:44.331823Z",
  "value": "0.7845",
  "cumulative": False,
  "metadata": "some additional data"
}
```

Request Headers

- Accept – the response content type depends on `Accept` header

Response Headers

- Content-Type – this depends on `Accept` header of request

Status Codes

- 201 Created – no error

Runs

GET /api/runs/

Gets all active/failed/finished runs

Example request:

```
GET /api/runs/ HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: text/javascript

[
  {
    "id": 1,
    "name": "Name of the run",
    "created_at": "2018-08-03T09:21:29.347960Z",
    "state": "STARTED",
    "job_id": "5ec9f286-e12d-41bc-886e-0174ef2bddae",
    "job_metadata": {...}
  },
  {
    "id": 2,
    "name": "Another run",
    "created_at": "2018-08-02T08:11:22.123456Z",
    "state": "FINISHED",
    "job_id": "add4de0f-9705-4618-93a1-00bbc8d9498e",
    "job_metadata": {...}
  },
]
```

Request Headers

- Accept – the response content type depends on *Accept* header

Response Headers

- Content-Type – this depends on *Accept* header of request

Status Codes

- 200 OK – no error

GET /api/runs/(int: *run_id*)/
Gets a run by id

Example request:

```
GET /api/runs/1/ HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: text/javascript

{
  "id": 1,
  "name": "Name of the run",
```

(continues on next page)

(continued from previous page)

```
  "created_at": "2018-08-03T09:21:29.347960Z",
  "state": "STARTED",
  "job_id": "5ec9f286-e12d-41bc-886e-0174ef2bddae",
  "job_metadata": {...}
}
```

:run_id The id of the run

Request Headers

- Accept – the response content type depends on *Accept* header

Response Headers

- Content-Type – this depends on *Accept* header of request

Status Codes

- 200 OK – no error

POST /api/runs/

Starts a new Run

Example request:

```
POST /api/runs/ HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
```

Request JSON Object

- **name** (*string*) – Name of the run
- **num_workers** (*int*) – Number of worker nodes for the run
- **num_cpus** (*json*) – Number of Cores utilized by each worker
- **max_bandwidth** (*json*) – Maximum egress bandwidth in mbps for each worker

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: text/javascript

{
  "id": 1,
  "name": "Name of the run",
  "created_at": "2018-08-03T09:21:29.347960Z",
  "state": "STARTED",
  "job_id": "5ec9f286-e12d-41bc-886e-0174ef2bddae",
  "job_metadata": {...}
}
```

Request Headers

- Accept – the response content type depends on *Accept* header

Response Headers

- Content-Type – this depends on *Accept* header of request

Status Codes

- 200 OK – no error
- 409 Conflict – a run is already active

2.3.2 Worker

The worker image contains all the boilerplate code needed for a distributed ML model and as well as the actual model code. It takes care of training the distributed model, depending on the settings provided by the Master. So the Master informs the Worker nodes what experiment the user would like to run and they run the relevant code by themselves.

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

2.4 Benchmarking Tasks

2.4.1 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

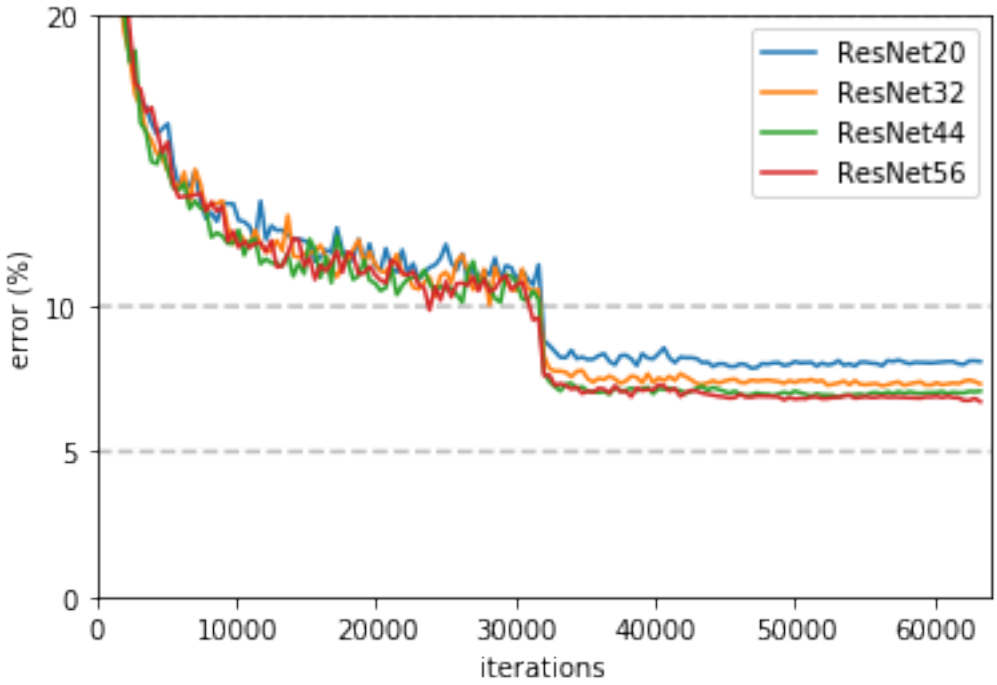
Accuracy after Time

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.

Here is a plot of validation error training iterations for ResNet on **'CIFAR-10<http://www.cs.toronto.edu/~kriz/cifar.html>'_** using the settings from Deep Residual Learning for Image Recognition.



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.

2.4.2 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 *goyal2017accurate*. 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.

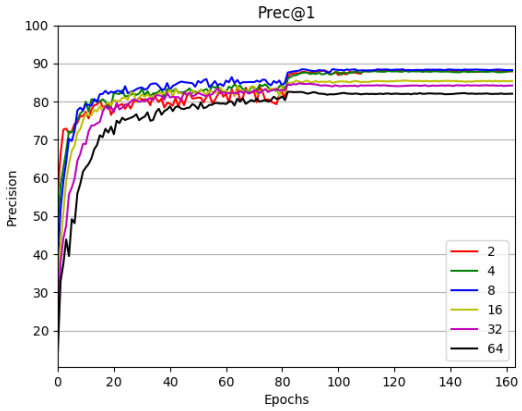
Implementation details:

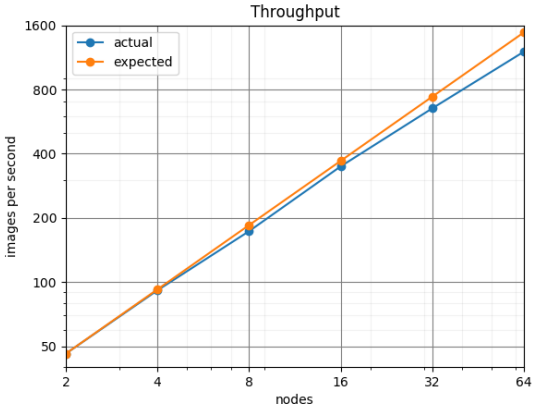
1. **Data Preprocessing** We followed the same approach as mentioned here.
2. **Selection of Framework & Systems** While our initial reference implementation is currently PyTorch, we will aim to provide the same algorithm in more frameworks very soon, starting with Tensorflow. For the systems, kubernetes allows easy transferability of our code. While initial results reported are from google kubernetes engine, AWS will be supported very soon.
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.

Results

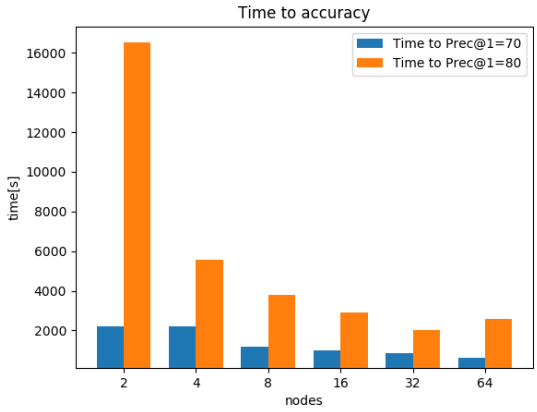
Here we present the results for scaling task.

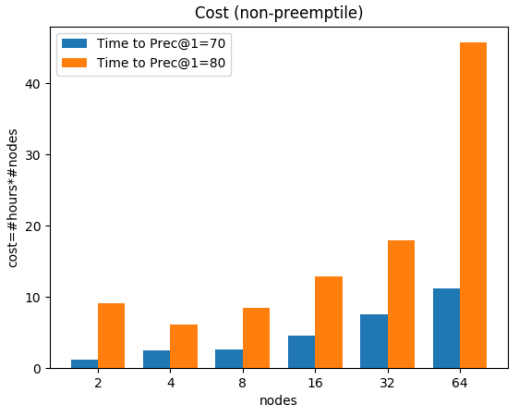
- The left figure is an epoch to accuracy curve. For 2, 4, 8 nodes, scaling the size of cluster gives same accuracy. For 16 or more nodes, the accuracy gradually drops.
- The right hand side compares expected throughput with the actual throughput. From the figure, we can see the actual throughput is marginally below ideal scaling.





- The left figure hand side figure compares the time to 70% and 80% accuracy for different number of nodes. 70% accuracy is easy to reach for all of the tests and the time-to-accuracy decreases with the number of nodes. For time-to-80%-accuracy, however, it spends more time on 64 nodes rather than 32 nodes.
- The right figure compares the cost of experiment. Note that a regular n1-standard-4 instance costs $0.1900 per hour and a preemptible one costs only $0.04. For experiments with 16 nodes or more, the task finishes with 24 hours and thus we can use preemptible instance. The cost can be reduced correspondingly.





1b. Image Classification (ResNet, ImageNet)

TODO (again synchr SGD as main baseline)

2. Linear Learning (Generalized Linear Models for Regression and Classification)

TODO (more data intensive compared to deep learning. again synchr SGD as main baseline)

2.5 Developer Guide

2.5.1 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

2.5.2 Code Style

Python code should follow PEP8 guidelines. flake8 checks PEP8 compliance

2.6 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:

2.6.1 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 :)

2.6.2 Get Started!

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

1. Fork the *mlbench* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/mlbench.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv mlbench
$ cd mlbench/
$ python setup.py develop
```

4. 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.

2.6.3 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.

2.6.4 Tips

To run a subset of tests:

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

2.6.5 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
```

2.7 Authors

2.7.1 Core Contributors

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

2.7.2 Contributors

None yet. Why not be the first?

2.8 Credits

- docker.openmpi for good open-mpi templates
- torchvision for model and metrics templates.
- mlperf for some of the argument parsing options.
- jekyllrb for good blog templates
- kubernetes-nfs-volume-on-gke for NFS volume on GKE

This package was created with Cookiecutter and the audreyr/cookiecutter-pypackage project template.

2.9 Change Log

2.9.1 0.1.1 (2018-09-24)

Full Changelog

Implemented enhancements:

- Add scripts and helm charts to setup NFS in Gcloud. #69

Fixed bugs:

- Automated Build broken due to Travis-Ci Memory limits #67

Closed issues:

- Potential Bottleneck: Throughput of Persistent Disk for NFS #72

2.9.2 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

- add benchmark code for MPI #7
- add benchmark code for tensorflow #6
- add benchmark code for apache reef #5
- add benchmark code for apache flink #4
- get initial benchmark numbers (spark reference implementation and mllib/ml) #3
- evaluate script (framework-independent) and algorithm output format #2
- bench-spark: remove prepare-data for now, comment on solver prequisites #1

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

2.10 Indices and tables

- genindex
- modindex
- search

Bibliography

[goyal2017accurate] Goyal, Priya, et al. Accurate, large minibatch SGD: training imagenet in 1 hour.

HTTP Routing Table

/api