

HBox

来自xt_wiki

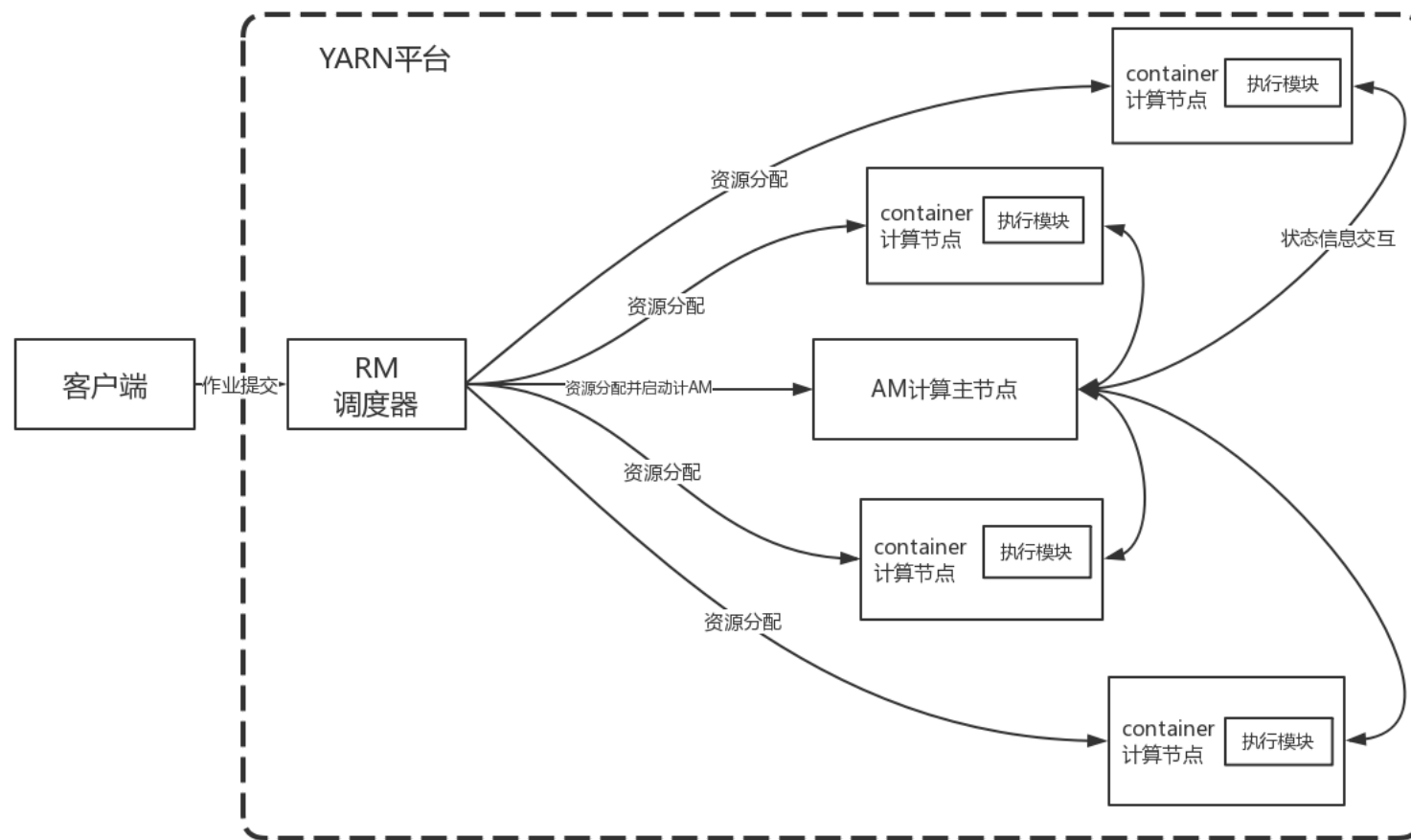
目录

- 1 HBox简介
- 2 HBox平台架构执行流程
- 3 HBox功能特性
- 4 HBox集群资源现状
- 5 HBox作业提交说明
- 6 HBox平台checkpoint
- 7 HBox平台debug功能
- 8 HBox平台提交docker作业
- 9 HBox平台提交digits作业
- 10 HBox作业查看说明
- 11 HBox作业示例Demo
- 12 FAQ
- 13 失败作业常见问题排查
 - 13.1 用户脚本执行错误
 - 13.2 可分配资源不足
 - 13.3 container执行过程中无故退出
 - 13.4 container未成功启动，History页面报出"Not get the container information, please check all containers started successfully !" 或 "Job History Log getting error !"
 - 13.5 container未成功启动，History页面显示container的状态为UNDEFINED
- 14 申请使用
- 15 新功能预告

HBox简介

HBox是一款基于Hadoop Yarn的深度学习作业调度框架，目前完成了对TensorFlow、MXNet、Caffe、Theano、PyTorch、Keras、XGBoost等常用框架的集成，同时具备良好的扩展性和兼容性。

HBox平台架构执行流程



HBox功能特性

1 支持多种深度学习框架

支持TensorFlow、MXNet分布式和单机模式，支持所有的单机模式的深度学习框架，如Caffe、Theano、PyTorch等。对于同一个深度学习框架支持多版本和自定义版本。

2 GPU虚拟机服务

HBox现可提交Docker Container作业，支持用户将在Docker容器内执行作业，进行GPU环境下的作业调试，详情请见HBox平台提交Docker作业部分说明。

3 输入数据读取

在HBox客户端作业提交脚本或系统配置中，用户可通过"--input-strategy"或"hbox.input.strategy"指定输入数据"--input"所采用的读取方式。目前，HBox支持如下三种HDFS输入数据读取方式：

1) **Download**：AM根据用户在提交脚本中所指定的输入数据参数"input"，遍历对应HDFS路径下所有文件，以文件为单位将输入文件数据均匀分配给不同Worker。在Worker中的执行程序对应进程启动之前，Worker会根据对应的文件分配信息将对应需要读取的HDFS文件下载到本地指定路径。**注意：HBox默认采用Download模式对输入数据进行处理。**使用示例说明：

```
HBox作业提交脚本中，指定参数：
--input /home/xitong/input#data \
在此参数示例中，HBox将遍历HDFS路径/home/xitong/input文件夹下的所有文件，根据Worker数目，将文件均匀分配给各Worker，各Worker下载所分配文件至./data（“#”后所指定名称）路径下。由此，用户可在程序中从本地（./data）操作输入文件。
注意：若“#”前为指定HDFS文件，则将该文件下载至“#”后的指定本地目录下。

扩展：
a. 不同HDFS文件下载至同一本地路径，如 输入文件为/home/xitong/input1/file1、/home/xitong/input2/file2，均对应本地路径./data，示例如下：
将所需的不同HDFS文件以“，”分隔传入input参数，最后以“#”来拼接所指定的本地路径名
--input /home/xitong/input1/file1,/home/xitong/input2/file2#data \
或 指定多个input参数，分别将HDFS文件对应本地目录
--input /home/xitong/file1#data \
--input /home/xitong/file2#data \
注意，根据Worker数目，若发现所需下载至同一本地目录下的HDFS文件存在重名现象：
如：“--worker-num 1 --input /home/xitong/input1/file1,/home/xitong/input2/file1#data”，即在Worker中会出现不同HDFS目录下的“file1”文件下载至本地./data路径。
解决：可通过设置提交参数“--isRenameInputFile true”，HBox将根据当前时间对文件进行重命名。（在此情况下，若用户程序存在指定文件名进行文件读取操作，可能需要进行修改）
b. 多个输入文件路径，如 分别将HDFS路径/home/xitong/input1、/home/xitong/input2、/home/xitong/input3下载至本地路径data1、data2、data3，可通过多个input参数进行指定，示例如下：
--input /home/xitong/input1#data1 \
--input /home/xitong/input2#data2 \
--input /home/xitong/input3#data3 \
```

2) **Placeholder**：与Download模式不同，Worker不会直接下载HDFS文件到本地指定路径，而是将所分配的HDFS文件列表通过环境变量`INPUT_FILE_LIST`传给Worker中的执行程序对应进程。执行程序从环境变量`os.environ["INPUT_FILE_LIST"]`中获取需要处理的文件列表，直接对HDFS文件进行读写等操作。该模式要求深度学习框架具备读取HDFS文件的功能，或借助第三方模块库如pydoop等。TensorFlow在0.10版本后已经支持直接操作HDFS文件。使用示例说明：

```
HBox作业提交脚本中，指定参数：
--input /home/xitong/input#data \
--input-strategy PLACEHOLDER \
在此参数示例中，HBox将遍历HDFS路径/home/xitong/input文件夹下的所有文件，根据Worker数目，将文件均匀分配给各Worker，各Worker将所分配的文件列表通过环境变量“INPUT_FILE_LIST”传给各执行程序，执行程序可依赖第三方库或框架自身来对HDFS文件直接操作。
注意：环境变量“INPUT_FILE_LIST”为json格式，其中，key为“input”参数所指定的本地路径，value为所分配的HDFS文件列表（list类型）
根据上述作业提交脚本，输入文件列表获取使用示例如下：
import os
import json
inputfile = json.loads(os.environ["INPUT_FILE_LIST"])
data_file = inputfile["data"]
注意，若输入文件列表太大容易造成执行命令参数过长导致作业失败。此处，若环境变量“INPUT_FILE_LIST”不存在，则说明已超出参数长度上限，Hbox会将环境变量“INPUT_FILE_LIST”应传入的内容写入本地文件“inputFileList.txt”中，用户可从该文件中读取输入文件列表，文件内容
例如：
with open("inputFileList.txt") as f:
    fileStr = f.readline()
    inputfile = json.loads(fileStr)
```

3) **InputFormat**：HBox集成有MapReduce中的InputFormat功能。在AM中，根据split size对所提交脚本中所指定的输入数据进行分片，并均匀的分配给不同worker。在Worker中，根据所分配到的分片信息，以用户指定的InputFormat类读取数据分片，并通过管道将数据传递给Worker中的执行程序进程。使用示例说明：

```
HBox作业提交脚本中，指定参数：
--input /home/xitong/input \
--input-strategy STREAM \
在此参数示例中，HBox将HDFS路径/home/xitong/input文件夹下的所有文件数据进行分片并分发给各Worker。Worker通过管道将数据传给执行程序进程，执行程序可通过标准输入方式进行数据读取。
```

4 输出结果保存

同输入数据读取类似，用户可通过" --output-strategy"或"hbox.output.strategy"指定输出结果"--output"的保存方式。HBox支持如下两种结果输出保存模式：

1) **Upload**：执行程序结束后，Worker根据提交脚本中输出数据参数'--output'，将本地输出路径保存文件上传至对应HDFS路径。**注意：HBox默认采用Upload模式对输出数据进行处理。**使用示例说明：

```
HBox作业提交脚本中，指定参数：
--output /home/xitong/output#model \
在此参数示例中，执行程序进程结束后，将本地./model下的所有文件，上传至HDFS路径/home/xitong/output/${containerID}下，注意，此处${containerID}为Worker所属ContainerID，由HBox自动分配。若需要多个输出路径，可通过设置多个“ --output”参数实现。
```

为方便用户在训练过程中随时将本地输出上传至HDFS，HBox系统在作业执行Web界面提供输出结果中间保存机制。用户作业执行过程中，根据训练进度及设置，将结果保存在"output"参数的对应的本地路径下，在执行中途需要提前将本地输出传至对应HDFS路径时，可在作业执行ApplicationMaster页面点击"Save Model"按钮进行中间输出结果上传。上传成功后，界面显示当前已上传的中间结果列表。

2) **OutputFormat**: HBox集成有MR中的OutputFormat功能。在训练过程中，Worker根据指定的OutputFormat类，将结果输出至HDFS。使用示例说明：

HBox作业提交脚本中，指定参数：
 --output /home/xitong/output \
 --output-strategy STREAM \
在此参数示例中，各Worker将执行程序进程中标准输出内容上传至HDFS路径/home/xitong/output/\${containerID}下。

5 Web展示

为方便用户查看作业信息，HBox提供有Web页面展示作业执行信息，包括Container执行日志、所属机器、执行状态等信息。并且，用户可在Worker执行程序中按照`"report:progress:<float type>"`格式向标准错误打印作业执行进度，HBox系统客户端和Web界面均可根据打印信息进行执行进度展示。对于TensorFlow作业，用户在设置TensorBoard服务开启后，Noah系统中对应Worker会根据用户配置开启TensorBoard服务进程，用户可在作业Web界面通过链接实时查看。作业运行结束后，可浏览页面查看历史日志信息。

6 TensorFlow分布式模式ClusterSpec自动构建

TensorFlow分布式模式默认需要在代码里预先设置各PS和Worker的host和port信息，并定义ClusterSpec。HBox简化了这个流程，实现了ClusterSpec的自动构建，并通过环境变量 TF_CLUSTER_DEF 、 TF_ROLE 、 TF_INDEX 对应的将clusterSpec、job_name、task_index等信息传送给各container（PS或Worker），用户只需在TensorFlow分布式模式程序中，从环境变量中获取对应变量，从而完成ClusterSpec及role、index分配。其中，环境变量TF_CLUSTER_DEF为json格式，其余为字符串格式。例如：

```
import os
import json
cluster_def = json.loads(os.environ["TF_CLUSTER_DEF"])
cluster = tf.train.ClusterSpec(cluster_def)
job_name = os.environ["TF_ROLE"]
task_index = int(os.environ["TF_INDEX"])
```

7 原生框架代码的兼容性

TensorFlow分布式模式的代码需要做三行修改，单机模式和其他深度学习框架代码不用做任何修改即可迁移到HBox上。

8 Restful接口提供

Hbox 1.4版本开始提供简单的Restful接口供用户发请求的方式获取container信息，其中base url为 **`${集群RM地址}/proxy/${APPID}/ws/`**

- 1) **/app**: 获取作业资源申请信息，接收类型为 application/json
- 2) **/containers**: 获取作业运行的container信息，接收类型为 application/json
- 3) **/containers/{containerid}**: 获取指定container的对应信息，接收类型为 application/json
- 4) **/app/evaluator**: 获取evaluator角色的containerID，接收类型为 text/plain
- 5) **/containers/{containerid}/{logtype}**: 获取指定container的日志内容（stdout / stderr），接收类型为 text/plain

注意：日志获取前提是：提交脚本中配置 --conf hbox.container.running.log.enable=true，否则日志不会重定向给restful接口接收对象；作业AM推出后，日志请求将失效；由于此处返回的日志类型为String，建议获取间隔不要太大，避免因内容超限报错丢失

- 6) **/app/signal/{sid}**: 给hbox-cmd中的执行进程发送指定信号，接收类型为 text/plain
 - 7) **/app/savemodel**: 发送中间模型保存请求，接收类型为 text/plain
 - 8) **/app/boardUrl**: 作业运行过程中的board url信息，接收类型为 text/plain
- 文件:Restful API.docx

HBox集群资源现状

HBox由原来的三大集群已经合并成一个集群统一管理，机器配置没有更改，集群主页统一，用户作业会根据指定的队列名提交到相应的机器节点，集群信息如下：

集群	Cluster地址	机器数	机器内存	机器CPU Cores数目	GPU信息
AI_ZZZC	http://m02.ai.zzzc.qihoo.net:8888/cluster	总数:59台 按照标签分布如下： P40: 18台 M40: 19台 K4080: 4台 ADSYS: 4台，为商业化产品部门私有 CPU: 14台	P40: 100GB M40: 195GB K4080: 100G CPU: 250G	P40: 24 cores M40: 24cores K4080: 24cores CPU: 24cores	P40: 4卡，显存22GB M40: 4卡，显存12GB K40: 4卡，显存11GB K80: 8卡，显存11GB

HBox作业提交说明

1) 在客户端通过hbox-submit命令，向HBox平台提交作业，示例如下[以TensorFlow 分布式作业提交为例]：

```
/usr/bin/hadoop/software/hbox/bin/hbox-submit \  
--app-type "tensorflow" \  
--driver-memory 1024 \  
--driver-cores 1 \  
--files tfTestDemo.py,dataDeal.py \  
--worker-memory 2048 \  
--worker-num 4 \  
--worker-cores 1 \  
--worker-gpus 1 \  
--ps-memory 1024 \  
--ps-num 2 \  
--ps-cores 1 \  
--input hdfs://namenode.safe.lycc.qihoo.net:9000/tmp/tensorflowOnYarn/demo/data#data \  
--output /home/xitong/tf-test/outputTest#model \  
--hbox-cmd "python tfTestDemo.py --data_path=./data --save_path=./model --log_dir=./eventLog --training_epochs=10" \  
--board-enable true \  
--priority VERY_LOW \  
--conf hbox.tf.board.log.dir=eventLog \  
--app-name "tensorflowTest"
```

2) 参数说明

HBox作业通用参数

--app-type	指定作业类型，默认为HBox，可根据作业类型，设置为tensorflow、caffe、xgboost、pytorch、keras、theano等
--app-name	指定作业名称
--driver-memory	指定申请的driver内存大小，默认单位为MB，默认大小为3072，用户也可指定单位g，比如3g或3G
--driver-cores	指定申请的driver核数，默认个数为1
--files	指定作业执行所需要的本地文件、文件夹，通过上传至HDFS发送给各container。需要上传多个文件或文件夹，请以“,”分隔形式传入。
--worker-memory	指定申请的worker内存大小，默认单位为MB，默认大小为1024，用户也可指定单位g，比如1g或1G
--worker-num	指定申请的worker数目，默认个数为1
--worker-cores	指定worker申请的核数，默认个数为1
--worker-gpus	指定worker申请所需的GPU卡数，默认个数为0
--hbox-cmd	作业执行命令
--input	输入文件路径，格式为：HDFS路径#本地文件夹名称。若存在多个输入文件，可设置多个input参数，详细使用可见HBox功能特性对应部分。注意：可以不指定input参数，直接从hdfs读取数据。（input参数中文件在各worker上的分布机制请见FAQ部分） （注：使用 --input功能，HBox会先把hdfs文件下载到本地，如果input比较大建议使用--inputformat功能或者TF等框架内置的读取HDFS的方法。另外，我们也提供python和C/C++读取HDFS文件的库，详见FAQ）
--output	输出文件路径，格式为：HDFS路径#本地文件夹名称，若未指定本地文件夹名称，默认将本地output路径下的内容上传至HDFS。若存在多个输出文件，可设置多个output参数，详见HBox功能特性对应部分。注意：若执行程序中直接对HDFS文件进行输出操作，可以不指定output。
--cacheFile	指定作业执行所需的相关HDFS文件或文件夹。若需要指定多项文件或文件夹，请以“,”分隔形式传入。
--cacheArchive	指定作业执行所需的相关HDFS压缩文件，自动解压至各container。若需要使用多项压缩文件，请以“,”分隔形式传入。压缩格式：tgz、zip、jar
--jars	指定作业所需用户自定义jar包文件。若需要指定多项文件或文件夹，请以“,”分隔形式传入。
--user-classpath-first	是否优先加载用户自定义jar包，默认为true。
--priority	作业执行优先级，支持DEFAULT、VERY_LOW、LOW、NORMAL、HIGH、VERY_HIGH
--queue	指定作业提交队列，默认为提交作业的用户名
--conf	用户自定义相关配置项，此处所设置的配置内容优先级最高

Tensorflow、MXNet 分布式扩展参数	
<code>--ps-memory</code>	指定申请的parameterServer的内存大小，默认单位为MB，默认大小为1024
<code>--ps-num</code>	指定申请的parameterServer的数目，默认个数为0
<code>--ps-cores</code>	指定parameterServer所申请的核数，默认个数为1
<code>--ps-gpus</code>	指定parameterServer申请所需的GPU卡数，默认个数为0

—board-enable	是否开启board服务，若开启则在作业页面可直接实时查看board页面，默认为false。其中，TensorFlow类型作业Board服务为tensorboard，其他类型作业Board服务为visualDL。
—board-index	启动board服务的worker index设置，默认为0。
—board-logdir	board日志存放的路径（可使用HDFS路径，若不含HDFS前缀，采用客户端默认HDFS前缀地址）。若未设置，则默认为本地文件路径./eventLog，程序执行完后，上传至集群对应HDFS路径。
—board-historydir	程序执行完成后，board日志存放的hdfs路径，默认为集群配置的/home/yarn/hbox/eventLog/\${applicationID}。
—board-reloadinterval	tensorboard服务中数据加载时间间隔，默认为1。
—board-modeldb	visualDL服务中加载的ONNX格式的模型文件。
—board-cacheTimeout	visualDL服务中缓存失效时间间隔，默认为20s。

```
hbox支持hadoop inputformat/outputformat接口，其详细使用主要参数如下：
—input-strategy      若用户要使用inputformat接口，则需设置此参数为STREAM，用户程序必须从标准输入读数据，默认为DOWNLOAD
—output-strategy     若用户要使用outputformat接口，则需设置此参数为STREAM，用户程序结果必须写入标准输出，默认为UPLOAD(训练模型的作业一般不用，离线inference时可能会用到)
—inputformat         指定inputformat类，默认为org.apache.hadoop.mapred.TextInputFormat(一般用户不需要指定)
—outputformat        指定outputformat类，默认为org.apache.hadoop.mapred.lib.TextMultiOutputFormat(一般用户不需要指定)
—stream-epoch        inputformat方式读数据的epoch数，表示要循环读多少遍数据，默认为1(一般用户不需要指定)

inputformat/outputformat使用demo路径如下：
数据：hdfs://namenode.safe.lycc.qihoo.net:9000/tmp/inputformatDemo/data
程序及脚本：hdfs://namenode.safe.lycc.qihoo.net:9000/tmp/inputformatDemo
```

由于目前hbox作业失败多数由于内存分配过少导致，hbox支持用户通过参数“--conf”配置，控制作业失败重试次数以及内存自动扩增比例，详细如下：

- conf hbox.app.max.attempts=1 若用户希望作业失败后自动重试，配置此参数。默认配置为1，即作业失败后，不进行重试；最大值依赖于集群设置。
- conf hbox.worker.mem.autoscale=0.0 此参数设置作业失败后重试时，worker内存自动扩充倍数（以作业提交设置内存大小为基准）。默认配置为0.0，即不进行自动扩充。
- conf hbox.ps.mem.autoscale=0.0 此参数设置作业失败后重试时，ps内存自动扩充倍数（以作业提交设置内存大小为基准）。默认配置为0.0，即不进行自动扩充。

可通过conf来设置用户自定义环境，

```

--conf hbox.horovod.extra.ld.library.path    horovod可以运行在GPU上，NCCL 2提供针对NVIDIA GPU和各种网络设备（例如RoCE或InfiniBand）优化的allreduce操作。如果已使用nccl- <version> .txz软件包安装了NCCL 2，则应将库路径添加到LD_LIBRARY_PATH环境变量中，或将其注册到/
--conf hbox.horovod.process.num.per.worker  指定运行机器上的进程数，默认为1
--conf hbox.horovod.mpi.threads.disable     默认情况下，提供RDMA功能的 Open MPI `openib` BTL 不支持多线程，为了将RDMA和openib一起使用，必须通过HOROVOD_MPI_THREADS_DISABLE来禁用多线程，默认为禁用。
--conf hbox.horovod.timeline                Horovod有能力记录其活动的时间表，称为Horovod Timeline。为了记录horovod时间轴，需要将“HOROVOD_TIMELINE”环境变量设置为要创建的时间轴文件的位置。
--conf hbox.horovod.fusion.threshold        Horovod支持批量小型allreduce操作的能力，如果使用需要设置融合缓冲区的大小，默认为不使用，若设置了hbox.horovod.fusion.threshold，0表示不启用Tensor Fusion，默认大小为64M。
--conf hbox.horovod.cycle.time              可以使用HOROVOD_CYCLE_TIME环境变量调整线程之间的循环时间
--conf hbox.horovod.stall.check.disable     指定是否不执行stall tensor check，默认为false；
--conf hbox.horovod.hierarchical.allreduce  如果需要使用分层allreduce，需要配置，默认为不启用。

```

horovod作业申请的worker数目应小于队列可用机器数目

HBox平台checkpoint

各个计算框架都有自己的checkpoint机制，所以利用框架本身的checkpoint机制和直接读写hdfs数据可以很容易实现训练恢复，以mxnet和tensorflow为例：

1) mxnet checkpoint:

mxnet 从checkpoint恢复模型主要用到了mx.model.load_checkpoint(model_prefix, load_epoch)接口，该接口主要传两个参数，第一个是模型的路径及前缀名，第二个参数是表示从哪一个迭代epoch开始load，返回网络结构和模型参数,用户代码逻辑可如下写：

load model

```
def load_model(args):
    if 'load_epoch' not in args or args.load_epoch is None:
        return (None, None, None)
    assert args.model_prefix is not None
    sym, arg_params, aux_params = mx.model.load_checkpoint(
        args.model_prefix, args.load_epoch)
    logging.info('Loaded model %s_%04d.params', args.model_prefix, args.load_epoch)
    return (sym, arg_params, aux_params)
```

将返回网络及参数继续训练，指定load epoch

```
lenet_model = mx.mod.Module(symbol=sym, context=[mx.gpu(int(i)) for i in args.gpus.split(',')])
lenet_model.fit(train_iter,
                eval_data = val_iter,
                optimizer = args.optimizer,
                optimizer_params = {'learning_rate':args.lr},
                eval_metric = 'acc',
                batch_end_callback = mx.callback.Speedometer(args.batch_size, 100),
                epoch_end_callback = checkpoint,
                arg_params = arg_params,
                aux_params = aux_params,
                begin_epoch = args.load_epoch if args.load_epoch else 0,
                num_epoch = args.num_epochs)
```

mxnet分布式版本的checkpoint和单机版本基本一致，不同的地方是由于每个worker都保存了模型，所以各个worker恢复自己对应模型，因此用户在save模型的时候加了一级worker rank目录，恢复的时候从各自的rank恢复，代码逻辑如下：

save model,将worker的rank号加入,join成新的模型存储路径，生成checkpoint回调函数mx.callback.do_checkpoint(model_prefix,do_epoch)返回,该函数第一个参数表示模型存储前缀路径，第二个参数表示多少个epoch做一次checkpoint.

```
def _save_model(args, rank=0):
    if args.model_prefix is None:
        return None
    model_prefix = os.path.join(os.path.dirname(args.model_prefix), str(rank), os.path.basename(args.model_prefix))
    return mx.callback.do_checkpoint(model_prefix if rank == 0 else "%s-%d" % (model_prefix, rank), 5)
```

训练时传入回调函数用来checkpoint

```
# run
model.fit(train,
    begin_epoch      = args.load_epoch if args.load_epoch else 0,
    num_epoch        = args.num_epochs,
    eval_data        = val,
    eval_metric       = eval_metrics,
    kvstore           = kv,
    optimizer         = args.optimizer,
    optimizer_params  = optimizer_params,
    initializer       = initializer,
    arg_params        = arg_params,
    aux_params        = aux_params,
    batch_end_callback = batch_end_callbacks,
    epoch_end_callback = checkpoint,
    allow_missing     = True,
    monitor           = monitor)
```

load model

```
def _load_model(args, rank=0):
    if 'load_epoch' not in args or args.load_epoch is None:
        return (None, None, None)
    assert args.model_prefix is not None
    model_prefix = os.path.join(os.path.dirname(args.model_prefix), str(rank), os.path.basename(args.model_prefix))
    if rank > 0:
        model_prefix += "-%d" % (rank)
    sym, arg_params, aux_params = mx.model.load_checkpoint(
        model_prefix, args.load_epoch)
    logging.info('Loaded model %s_%04d.params', model_prefix, args.load_epoch)
    return (sym, arg_params, aux_params)
```

从指定epoch开始恢复模型继续训练

```
# run
model.fit(train,
    begin_epoch      = args.load_epoch if args.load_epoch else 0,
    num_epoch        = args.num_epochs,
    eval_data        = val,
    eval_metric       = eval_metrics,
    kvstore           = kv,
    optimizer         = args.optimizer,
    optimizer_params  = optimizer_params,
    initializer       = initializer,
    arg_params        = arg_params,
    aux_params        = aux_params,
    batch_end_callback = batch_end_callbacks,
    epoch_end_callback = checkpoint,
    allow_missing     = True,
    monitor           = monitor)
```

注: **model_prefix**为hdfs全路径, **mxnet checkpoint demo**路径为:

src: hdfs://namenode.safe.lycc.qihoo.net:9000/tmp/mxnet/checkpoint_demo

data: hdfs://namenode.safe.lycc.qihoo.net:9000/tmp/mxnet/data

model: hdfs://namenode.safe.lycc.qihoo.net:9000/tmp/mxnet/checkpoint_model

2) tensorflow checkpoint:

tensorflow 从checkpoint恢复模型主要用到了tf.train.Saver()接口restore方法，主要传两个参数，第一个是tf session，第二参数是checkpoint模型的路径，利用hbox的cacheFile功能将模型文件download到本地restore，用户提交脚本可如下写：

```
~/usr/bin/hadoop/software/hadoop/bin/hadoop fs -rmr /home/xitong/tf-test/outputfile/model/*
~/usr/bin/hadoop/software/hbox/bin/hbox-submit \
  --app-type "tensorflow" \
  --files train_mnist.py \
  --worker-num 1 \
  --worker-gpus 1 \
  --worker-memory 8192 \
  --boardEnable true \
  --cacheFile hdfs://namenode.safe.lycc.qihoo.net:9000/tmp/tensorflow_checkpoint/model/single_model#checkpoint \
  --hbox-cmd "python train_mnist.py --data_path=./data
              --save_path=hdfs://namenode.safe.lycc.qihoo.net:9000/home/xitong/tf-test/outputfile/model
              --checkpoint_path checkpoint/model.ckpt-2500
              --training_epochs=10" \
  --input hdfs://namenode.safe.lycc.qihoo.net:9000/tmp/tensorflow_checkpoint/data#data \
  --appName "tfdemo_test_single" \
echo "view result"
hadoop fs -ls /home/xitong/tf-test/outputfile/model/*
~
```

用户代码逻辑可如下写：

```
if FLAGS.checkpoint_path:
    saver.restore(sess, FLAGS.checkpoint_path)
```

注：**model 存储直接传hdfs全路径，tensorflow checkpoint demo路径为：**

src: hdfs://namenode.safe.lycc.qihoo.net:9000/tmp/tensorflow_checkpoint/demo

data: hdfs://namenode.safe.lycc.qihoo.net:9000/tmp/tensorflow_checkpoint/data

model: hdfs://namenode.safe.lycc.qihoo.net:9000/tmp/tensorflow_checkpoint/model

HBox平台debug功能

很多时候用户想要在本地图试程序，待本地调试通了再提交到集群跑作业，但是并没有用于调试的gpu机器以及运行环境。针对此情况HBox平台结合docker提供了一种新的作业类型-调试作业，用户申请用于调试的worker数以及相应的资源，提交作业，HBox会返回用于登录各个worker的命令以及密码，用户通过ssh直接登录到申请的worker执行本地操作，比如调试程序，安装缺少的包，下载数据等。与提交其他类型的作业相比，提交脚本只需修改几个参数即可，具体提交脚本及其参数解释如下：

```
#!/bin/sh
~/usr/bin/hadoop/software/hbox/bin/hbox-submit \
  --app-type "vpc" \
  --worker-memory 5g \
  --worker-gpus 1 \
  --worker-cores 1 \
  --worker-num 1 \
  --duration 5m \
  --priority HIGH \
  --app-name "vpc_test"
~
```

HBox调试作业参数说明

--app-type	指定作业类型, hbox调试作业类型必须为“vpc”, 大小写均可
--worker-memory	指定申请用于调试的worker内存大小, 默认单位为MB, 默认大小为1024
--worker-num	指定申请用于调试的worker数目, 默认个数为1
--worker-cores	指定worker申请的cpu核数, 默认个数为1
--worker-gpus	指定worker申请所需的GPU卡数, 默认个数为0
--duration	指定worker需要运行的时间, 单位可为s, m, h, d, 默认为1d(即1天), 若此参数指定为0(无单位)则表示永久运行.
--priority	作业执行优先级, 支持DEFAULT、VERY_LOW、LOW、NORMAL、HIGH、VERY_HIGH
--queue	指定作业提交队列, 默认为提交作业的用户名
--conf	vpc类型的作业用户可以通过conf参数来配置自定义的镜像, 只需指定镜像地址即可, 例如镜像地址为harbor.docker.qihoo.net:5000/ai/deeplearning:v2.8, 则用户需指定参数 --conf yarn.nodemanager.docker-container-executor.image-name=harbor.docker.qihoo.net:5000/ai/deeplearning:v2.8。若不用conf参数指定镜像则会使用hbox平台提供的默认镜像(默认镜像版本为v2.7, 建议用户手动指定使用v2.8版本)
--app-name	用户指定的作业名

vpc类型的作业不需要指定hbox-cmd参数了, 但与其它类型作业一样, 用于用户上传和下载的一些参数如--files, --input等在vpc类型的作业下也都能正常使用.

执行脚本后客户端返回的登录命令和密码如下:

```
17/09/15 16:27:24 INFO Client: Hbox application needs 1 worker container in fact
17/09/15 16:27:29 INFO Client: Waiting for vpc login command and password...
17/09/15 16:27:30 INFO Client: Waiting for vpc login command and password...
17/09/15 16:27:31 INFO Client: Waiting for vpc login command and password...
17/09/15 16:27:32 INFO Client: Waiting for vpc login command and password...
17/09/15 16:27:33 INFO Client: Waiting for vpc login command and password...
17/09/15 16:27:34 INFO Client: Waiting for vpc login command and password...
17/09/15 16:27:35 INFO Client: Waiting for vpc login command and password...
17/09/15 16:27:36 INFO Client: Received vpc login command and password from container_e358_1505442218042_0004_01_000002
17/09/15 16:27:36 INFO Client: Login command:ssh root@gpu07.sys.lycc.qihoo.net -p 59136
17/09/15 16:27:36 INFO Client: Password:xLLVIm
```

使用ssh命令和密码登录worker, 进行本地操作。登录vpc类型作业的worker后, shell会显示为绿色, 并且主机名带有vpc字样, 做个简单的本地操作, 起个python shell执行tensorflow session打印hello world, 结果如下:

```

crwxrwxrwx 1 root root 100 Sep 15 08:37 core-site.xml -> /data00/xitong/localhosts/usercache/xitong/a
drwxrwxrwt 1 root root 28 Sep 15 08:37 tmp
drwxr-xr-x 5 root root 480 Sep 15 08:49 dev
dr-xr-x-- 1 root root 42 Sep 15 08:49 root
[root@vpc-002 /]$ python
Python 2.7.5 (default, Nov 6 2016, 00:28:07)
[GCC 4.8.5 20150623 (Red Hat 4.8.5-11)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import tensorflow as tf
>>> with tf.Session() as sess:
...     print("hello world!")
2017-09-15 08:50:55.490286: W tensorflow/core/platform/cpu_feature_guard.cc:45] The TensorFlow library w
d could speed up CPU computations.
2017-09-15 08:50:55.490365: W tensorflow/core/platform/cpu_feature_guard.cc:45] The TensorFlow library w
d could speed up CPU computations.
2017-09-15 08:50:55.490382: W tensorflow/core/platform/cpu_feature_guard.cc:45] The TensorFlow library w
ould speed up CPU computations.
2017-09-15 08:50:55.490396: W tensorflow/core/platform/cpu_feature_guard.cc:45] The TensorFlow library w
could speed up CPU computations.
2017-09-15 08:50:55.490409: W tensorflow/core/platform/cpu_feature_guard.cc:45] The TensorFlow library w
ould speed up CPU computations.
2017-09-15 08:50:55.868784: I tensorflow/core/common_runtime/gpu/gpu_device.cc:940] Found device 0 with
name: Tesla K40m
major: 3 minor: 5 memoryClockRate (GHz) 0.745
pciBusID 0000:02:00.0
Total memory: 11.17GiB
Free memory: 11.10GiB
2017-09-15 08:50:55.868969: I tensorflow/core/common_runtime/gpu/gpu_device.cc:961] DMA: 0
2017-09-15 08:50:55.868983: I tensorflow/core/common_runtime/gpu/gpu_device.cc:971] 0: Y
2017-09-15 08:50:55.869002: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1030] Creating TensorFlow
hello world!
>>> █

```

注：若采用docker镜像提交作业，建议以默认提供的镜像为基础版本，在此基础上进行扩充，避免因环境变量或服务依赖等缺失造成作业失败。

HBox平台提交docker作业

hbox平台支持用户作业跑在docker容器里，用户可登录到容器查看程序的运行情况及中间结果等，提交方式不变，只需通过--conf指定些参数即可，示例提交脚本如下：

```
#!/bin/sh
/usr/bin/hadoop/software/hadoop/bin/hadoop fs -rmr /home/ouyangwen/tf-test/outputTest_dist
/usr/bin/hadoop/software/hbox/bin/hbox-submit \
  --app-type "tensorflow" \
  --files tfTestDemo.py,dataDeal.py \
  --worker-memory 8192 \
  --worker-num 2 \
  --worker-cores 1 \
  --worker-gpus 1 \
  --ps-memory 2048 \
  --ps-num 2 \
  --hbox-cmd "python tfTestDemo.py --data_path=./data --save_path=./model --log_dir=./eventLog --training_epochs=10" \
  --board-enable true \
  --output /home/ouyangwen/tf-test/outputTest_dist#model \
  --input hdfs://namenode.safe.lycc.qihoo.net:9000/home/xitong/tf-test/inputfile#data \
  --conf yarn.nodemanager.container-executor.type=docker \
  --app-name "tfdemo_dist_docker"
echo "view result"
hadoop fs -ls /home/ouyangwen/tf-test/outputTest_dist/*
```

yarn.nodemanager.container-executor.type参数必须指定为docker,同vpc类型作业一样,若用户想使用自己的docker镜像,则需通过--conf参数指定yarn.nodemanager.docker-container-executor.image-name参数为自己地镜像地址。

执行脚本后客户端返回的登录命令和密码如下:

```
17/09/15 17:06:24 INFO Client: Waiting for vpc login command and password...
17/09/15 17:06:25 INFO Client: Waiting for vpc login command and password...
17/09/15 17:06:26 INFO Client: Waiting for vpc login command and password...
17/09/15 17:06:27 INFO Client: Waiting for vpc login command and password...
17/09/15 17:06:28 INFO Client: Received vpc login command and password from container e358_1505442218042_0005_01_000002
17/09/15 17:06:28 INFO Client: Login command:ssh root@gpu07.sys.lycc.qihoo.net -p 38389
17/09/15 17:06:28 INFO Client: Password:T3oC9G
```

使用ssh命令和密码登录worker,默认会切换到用户执行程序所在目录,如下图:

```
lrwxrwxrwx 1 root root 114 Sep 15 09:21 singleTfTestDemo.py -> /data00/xitong/localdirs/us
lrwxrwxrwx 1 root root 108 Sep 15 09:21 core-site.xml -> /data00/xitong/localdirs/usercach
lrwxrwxrwx 1 root root 108 Sep 15 09:21 AppMaster.jar -> /data00/xitong/localdirs/usercach
drwxr-xr-x 2 root root 4096 Sep 15 09:21 data
drwxr-xr-x 5 root root 480 Sep 15 09:23 dev
drwxrwxrwt 1 root root 50 Sep 15 09:23 tmp
drwxr-xr-x 2 root root 68 Sep 15 09:23 eventLog
drwxr-xr-x 2 root root 4096 Sep 15 09:24 output
[root@vpc-002 /]$
```

HBox平台提交digits作业

digits是nvidia公司开发的可用于在web训练深度学习模型的系统,目前支持Caffe, Torch, and Tensorflow框架。hbox平台支持用户提交digits作业,示例提交脚本如下:

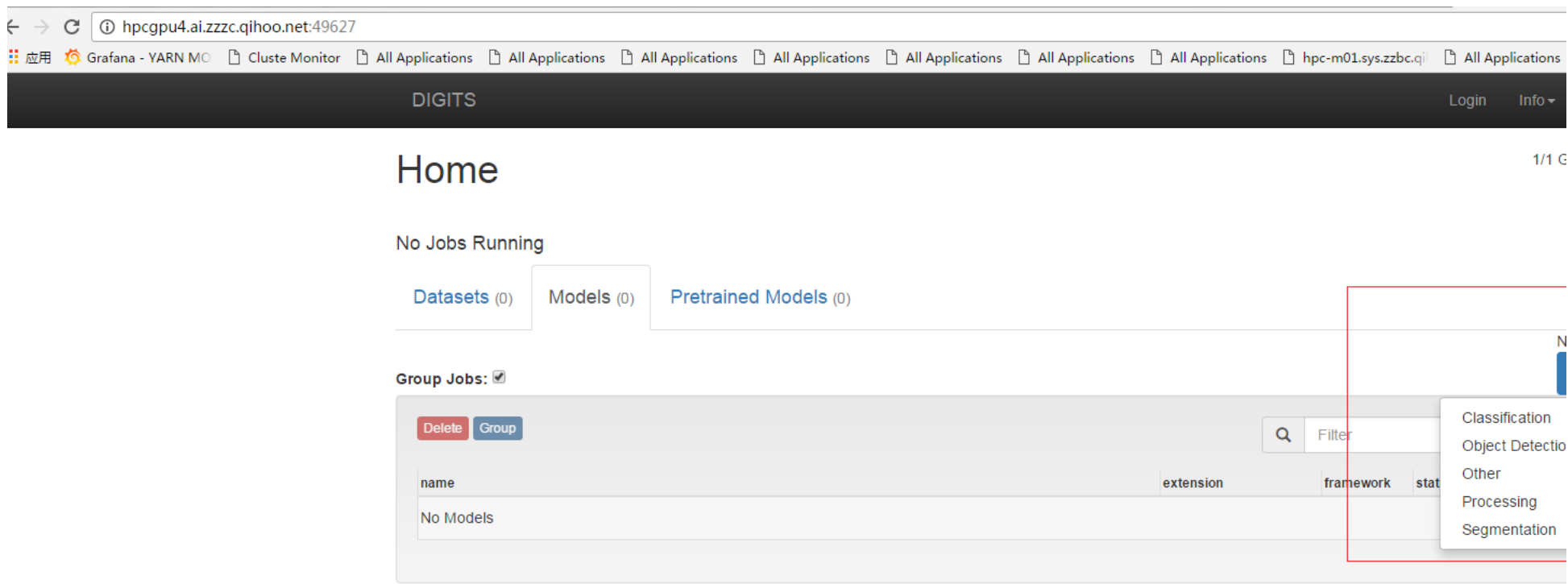
```
#!/bin/sh
/usr/bin/hadoop/software/hbox/bin/hbox-submit \
  --app-type "digits" \
  --worker-memory 5g \
  --worker-gpus 1 \
  --duration 10m \
  --priority HIGH \
  --app-name "digits_test"
~
```

hbox指定的作业类型必须为“**digits**”，大小写均可，使用**--duration**参数指定digits server需要运行的时间，单位可为s, m, h, d, 默认为1d(即1天)，若此参数指定为0(无单位)则表示永久运行..

执行脚本后客户端返回ditits server的的登录命令和密码以及web访问的url地址如下：

```
17/11/08 12:12:48 INFO Client: Waiting for digits server url...
17/11/08 12:12:49 INFO Client: Waiting for vpc login command and password...
17/11/08 12:12:49 INFO Client: Waiting for digits server url...
17/11/08 12:12:50 INFO Client: Received vpc login command and password from container e10_1506323180152_5557_01_000006
17/11/08 12:12:50 INFO Client: Login command:ssh root@hpcgpu4.ai.zzzc.qihoo.net -p 41261
17/11/08 12:12:50 INFO Client: Password:fkPrIt
17/11/08 12:12:50 INFO Client: Received digits server url from container e10_1506323180152_5557_01_000006
17/11/08 12:12:50 INFO Client: digits server url: http://hpcgpu4.ai.zzzc.qihoo.net:49627
```

通过web地址访问digits服务，用户可创建模型，加载数据进行训练，如下图：



HBox作业查看说明

作业提交后，可进入 <http://m03.ai.zzzc.qihoo.net:8888/cluster/apps> 页面查看作业执行状态及日志，如下：



All Applications

Cluster

[About](#)
[Nodes](#)
[Node Labels](#)
[Applications](#)

[NEW](#)
[NEW SAVING](#)
[SUBMITTED](#)
[ACCEPTED](#)
[RUNNING](#)
[FINISHED](#)
[FAILED](#)
[KILLED](#)

[Scheduler](#)

Tools

Cluster Metrics

Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Memory Used	Memory Total	Memory Reserved	VCores Used	VCores Total	VCores Reserved	GCores Used	GCores Total	GCores Reserved	Active Nodes	Decommissioned Nodes
1958	0	5	1953	13	72 GB	500 GB	0 B	13	120	0	16	32	0	5	0

User Metrics for dr.who

Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Containers Pending	Containers Reserved	Memory Used	Memory Pending	Memory Reserved	VCores Used	VCores Pending	VCores Reserved
0	0	0	0	0	0	0	0 B	0 B	0 B	0	0	0

Scheduler Metrics

Scheduler Type	Scheduling Resource Type	Minimum Allocation
Fair Scheduler	[MEMORY, CPU, GPU]	<memory:1024, vCores:1, gCores:0> <memory:51200, vCores:12, gCores:0>

Show 20 entries

ID	User	Name	Application Type	Queue	Application Priority	StartTime	FinishTime	State	FinalStatus	Running Containers	Allocated VCoers	Allocated GCores	Allocated Memory(MB)	% of Queue
application_1495726172438_2027	xitong	hboxtest	TENSORFLOW	root.xitong	VERY_LOW	Thu Jun 29 15:21:02	N/A	RUNNING	UNDEFINED	5	5	2	10240	8.0

提交作业ID，点击进入AM日志查看界面

用户名

作业名称

作业类型

所属队列

执行状态



Application application_1495726172438_2027

Cluster

[About](#)
[Nodes](#)
[Node Labels](#)
[Applications](#)
 NEW
 NEW SAVING
 SUBMITTED
 ACCEPTED
 RUNNING
 FINISHED
 FAILED
 KILLED
[Scheduler](#)

Tools

Kill Application

User: xitong
Name: hboxtest
Application Type: TENSORFLOW
Application Tags:
Application Priority: VERY_LOW (Higher Integer value indicates higher priority)
YarnApplicationState: RUNNING: AM has registered with RM and started running.
FinalStatus Reported by AM: Application has not completed yet.
Started: Thu Jun 29 15:21:02 +0800 2017
Elapsed: 1mins 28sec
Tracking URL: [ApplicationMaster](#) 点击进入container日志界面
Diagnostics:

Used Container Num this application 5
Reserved Container Num this application N?A
Needed Resource for this application: <memory:10240, vCores:5, gCores:2>
Used Resource for this application: <memory:10240, vCores:5, gCores:2>
Reserved Resource for this application: <memory:0, vCores:0, gCores:0>
Total Resource Preempted: <memory:0, vCores:0, gCores:0>
Total Number of Non-AM Containers Preempted: 0
Total Number of AM Containers Preempted: 0
Resource Preempted from Current Attempt: <memory:0, vCores:0, gCores:0>
Number of Non-AM Containers Preempted from Current Attempt: 0
Aggregate Resource Allocation: 851630 MB-seconds, 415 vcore-seconds, 162 gcore-seconds
Cost(Yuan): 13.80 Yuan

Show 20 entries

Attempt ID	Started	Node	Logs	
appattempt_1495726172438_2027_000001	Thu Jun 29 15:21:02 +0800 2017	http://gpu09.sys.lycc.qihoo.net:8042	Logs 点击查看AM日志	0


Showing 1 to 1 of 1 entries

←

→

🔄

gpu07.sys.lycc.qihoo.net:8888/proxy/application_1495726172438_2027



Tools

Configuration

Thread dump

Logs

Metrics

Tensorflow Application application_1495726172438_2027

All Containers:

containerID 列表

container所在机器

GPU分配信息

container所属角色

当前执行状态

container开始时间

Container ID	Container Host	GPU Device ID	Container Role	Container Status	Start Time
container e48 1495726172438 2027 01 000004	gpu06.sys.lycc.qihoo.net	2	worker	RUNNING	Thu Jun 29 15:21:18 CST 2017
container e48 1495726172438 2027 01 000005	gpu07.sys.lycc.qihoo.net	3	worker	RUNNING	Thu Jun 29 15:21:15 CST 2017
container e48 1495726172438 2027 01 000002	gpu06.sys.lycc.qihoo.net	-	ps	RUNNING	Thu Jun 29 15:21:17 CST 2017
container e48 1495726172438 2027 01 000003	gpu07.sys.lycc.qihoo.net	-	ps	RUNNING	Thu Jun 29 15:21:17 CST 2017

View TensorBoard:

Tensorboard Info

<http://gpu06.sys.lycc.qihoo.net:50038>

Save Model

点击后，将本地输出目录当前内容上传至对应hdfs路径下的interResult目录中

saved the model completed!

Saved timeStamp	Saved path
2017-06-29 15:22:46	/home/xitong/tf-test/outputTest/interResult/interResult_2017_06_29_15_22_46

点击进入对应container日志

tensorBoard链接

输出存储信息列表



Logs for container_e48_1495726172438_2027_01_000004

▼ ResourceManager

RM Home

► NodeManager

► Tools

Showing 4096 bytes. Click [here](#) for full log

```
er: (Step: 1603,, Epoch: 2,, Cost: 0.1573,, Time: 35.52ms')
17/06/29 15:22:17 INFO container.HboxContainer: (Step: 1605,, Epoch: 2,, Cost: 0.1581,, Time: 35.69ms')
17/06/29 15:22:17 INFO container.HboxContainer: (Step: 1607,, Epoch: 2,, Cost: 0.1656,, Time: 35.71ms')
17/06/29 15:22:17 INFO container.HboxContainer: (Step: 1609,, Epoch: 2,, Cost: 0.1703,, Time: 35.68ms')
17/06/29 15:22:17 INFO container.HboxContainer: (Step: 1611,, Epoch: 2,, Cost: 0.1657,, Time: 35.66ms')
17/06/29 15:22:17 INFO container.HboxContainer: (Step: 1613,, Epoch: 2,, Cost: 0.1640,, Time: 35.76ms')
17/06/29 15:22:17 INFO container.HboxContainer: (Step: 1615,, Epoch: 2,, Cost: 0.1640,, Time: 35.71ms')
17/06/29 15:22:18 INFO container.HboxContainer: (Step: 1617,, Epoch: 2,, Cost: 0.1625,, Time: 35.88ms')
17/06/29 15:22:18 INFO container.HboxContainer: (Step: 1619,, Epoch: 2,, Cost: 0.1628,, Time: 35.78ms')
17/06/29 15:22:18 INFO container.HboxContainer: (Step: 1621,, Epoch: 2,, Cost: 0.1596,, Time: 35.68ms')
17/06/29 15:22:18 INFO container.HboxContainer: (Step: 1623,, Epoch: 2,, Cost: 0.1542,, Time: 35.73ms')
17/06/29 15:22:18 INFO container.HboxContainer: (Step: 1625,, Epoch: 2,, Cost: 0.1605,, Time: 35.91ms')
17/06/29 15:22:18 INFO container.HboxContainer: (Step: 1627,, Epoch: 2,, Cost: 0.1556,, Time: 35.74ms')
17/06/29 15:22:18 INFO container.HboxContainer: (Step: 1629,, Epoch: 2,, Cost: 0.1653,, Time: 35.79ms')
17/06/29 15:22:18 INFO container.HboxContainer: (Step: 1631,, Epoch: 2,, Cost: 0.1579,, Time: 35.80ms')
17/06/29 15:22:18 INFO container.HboxContainer: (Step: 1633,, Epoch: 2,, Cost: 0.1608,, Time: 35.68ms')
17/06/29 15:22:18 INFO container.HboxContainer: (Step: 1634,, Epoch: 2,, Cost: 0.1646,, Time: 35.59ms')
17/06/29 15:22:18 INFO container.HboxContainer: (Step: 1636,, Epoch: 2,, Cost: 0.1672,, Time: 35.76ms')
17/06/29 15:22:18 INFO container.HboxContainer: (Step: 1638,, Epoch: 2,, Cost: 0.1618,, Time: 35.63ms')
17/06/29 15:22:18 INFO container.HboxContainer: (Step: 1640,, Epoch: 2,, Cost: 0.1632,, Time: 36.68ms')
17/06/29 15:22:18 INFO container.HboxContainer: (Step: 1642,, Epoch: 2,, Cost: 0.1677,, Time: 35.64ms')
17/06/29 15:22:18 INFO container.HboxContainer: (Step: 1644,, Epoch: 2,, Cost: 0.1648,, Time: 35.98ms')
17/06/29 15:22:18 INFO container.HboxContainer: (Step: 1646,, Epoch: 2,, Cost: 0.1725,, Time: 35.60ms')
17/06/29 15:22:18 INFO container.HboxContainer: (Step: 1648,, Epoch: 2,, Cost: 0.1611,, Time: 35.73ms')
17/06/29 15:22:18 INFO container.HboxContainer: (Step: 1650,, Epoch: 2,, Cost: 0.1628,, Time: 35.69ms')
17/06/29 15:22:18 INFO container.HboxContainer: (Step: 1652,, Epoch: 2,, Cost: 0.1615,, Time: 35.76ms')
17/06/29 15:22:18 INFO container.HboxContainer: (Step: 1654,, Epoch: 2,, Cost: 0.1589,, Time: 36.23ms')
17/06/29 15:22:18 INFO container.HboxContainer: (Step: 1656,, Epoch: 2,, Cost: 0.1612,, Time: 35.91ms')
17/06/29 15:22:18 INFO container.HboxContainer: (Step: 1658,, Epoch: 2,, Cost: 0.1621,, Time: 35.94ms')
17/06/29 15:22:18 INFO container.HboxContainer: (Step: 1660,, Epoch: 2,, Cost: 0.1673,, Time: 36.10ms')
17/06/29 15:22:18 INFO container.HboxContainer: (Step: 1662,, Epoch: 2,, Cost: 0.1561,, Time: 35.99ms')
17/06/29 15:22:18 INFO container.HboxContainer: (Step: 1664,, Epoch: 2,, Cost: 0.1599,, Time: 36.01ms')
17/06/29 15:22:18 INFO container.HboxContainer: (Step: 1666,, Epoch: 2,, Cost: 0.1605,, Time: 35.98ms')
17/06/29 15:22:18 INFO container.HboxContainer: (Step: 1668,, Epoch: 2,, Cost: 0.1541,, Time: 36.08ms')
17/06/29 15:22:19 INFO container.HboxContainer: (Step: 1670,, Epoch: 2,, Cost: 0.1587,, Time: 36.06ms')
17/06/29 15:22:19 INFO container.HboxContainer: (Step: 1672,, Epoch: 2,, Cost: 0.1616,, Time: 35.86ms')
```

→ container对应日志信息

作业执行完毕后，可查看作业的历史执行信息



All Applications

- Cluster
 - About
 - Nodes
 - Node Labels
 - Applications
 - NEW
 - NEW SAVING
 - SUBMITTED
 - ACCEPTED
 - RUNNING
 - FINISHED
 - FAILED
 - KILLED
 - Scheduler
- Tools

Cluster Metrics

Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Memory Used	Memory Total	Memory Reserved	VCores Used	VCores Total	VCores Reserved	GCores Used	GCores Total	GCores Reserved	Active Nodes	D
1994	0	4	1990	8	62 GB	500 GB	0 B	8	120	0	14	32	0	5	0

User Metrics for dr.who

Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Containers Pending	Containers Reserved	Memory Used	Memory Pending	Memory Reserved	VCores Used	VCores Pending	VCores
0	0	0	0	0	0	0	0 B	0 B	0 B	0	0	0

Scheduler Metrics

Scheduler Type	Scheduling Resource Type	Minimum Allocation
Fair Scheduler	[MEMORY, CPU, GPU]	<memory:1024, vCores:1, gCores:0> <memory:51200, vCores:12, gCores:0>

Show 20 entries

ID	User	Name	Application Type	Queue	Application Priority	StartTime	FinishTime	State	FinalStatus	Running Containers	Allocated VCo	Allocated GCores	Allocated Memory(MB)	% of Queue
application 1495726172438_2027	xitong	hboxtest	TENSORFLOW	root.xitong	VERY_LOW	Thu Jun 29 15:21:02 +0800 2017	Thu Jun 29 15:25:32 +0800 2017	FINISHED	SUCCEDED	N/A	N/A	N/A	N/A	0.0

Showing 1 to 1 of 1 entries (filtered from 8,496 total entries)

作业ID, 点击进入AM信息界面



Tensorflow Application application_1495726172438_2027

Tools

- Configuration
- Thread dump
- Logs
- Metrics

All Containers: [container列表 点击查看日志信息](#) [container所在机器](#) [GPU分配列表](#) [角色](#) [执行状态](#) [开始执行时间](#) [任务ID](#)

Container ID	Container Host	GPU Device ID	Container Role	Container Status	Start Time	Task ID
container e48 1495726172438_2027 01 000002	gpu06.sys.lycc.qihoo.net	-	ps	SUCCEDED	Thu Jun 29 15:21:17 CST 2017	Thu Jun 29 15:21:17 CST 2017
container e48 1495726172438_2027 01 000003	gpu07.sys.lycc.qihoo.net	-	ps	SUCCEDED	Thu Jun 29 15:21:17 CST 2017	Thu Jun 29 15:21:17 CST 2017
container e48 1495726172438_2027 01 000004	gpu06.sys.lycc.qihoo.net	2	worker	SUCCEDED	Thu Jun 29 15:21:18 CST 2017	Thu Jun 29 15:21:18 CST 2017
container e48 1495726172438_2027 01 000005	gpu07.sys.lycc.qihoo.net	3	worker	SUCCEDED	Thu Jun 29 15:21:15 CST 2017	Thu Jun 29 15:21:15 CST 2017

View TensorBoard:

[tensorfBoard信息](#)

Tensorboard Info
tensorboard --logdir=hdfs://namenode.safe.lycc.qihoo.net:9000/home/yarn/hbox/eventLog/application_1495726172438_2027

Saved Model

[已上传的中间输出列表](#)

Saved timeStamp	Saved path
2017-06-29 15:22:46	/home/xitong/tf-test/outputTest/interResult/interResult_2017_06_29_15_22_46

HBox作业示例Demo

caffe demo所在hdfs路径如下：

```
数据存放地址：  hdfs://namenode.safe.lycc.qihoo.net:9000/home/xitong/hbox/caffe/data
执行程序地址：  hdfs://namenode.safe.lycc.qihoo.net:9000/home/xitong/hbox/caffe/demo
caffe dist包地址：hdfs://namenode.safe.lycc.qihoo.net:9000/home/xitong/hbox/caffe/dist/caffe.tgz
[注意：该caffe dist包为gpu版本，lib包为libccaffe.so.1.0.0-rc5，若版本与本地caffe程序编译版本不一致，请重新编译或使用自己的dist包]
提交脚本中，注意指定 --app-type "caffe"
集群提供的caffe包的版本为：1.0
```

xgboost demo所在hdfs路径如下：

```
数据存放地址：  hdfs://namenode.safe.lycc.qihoo.net:9000/home/xitong/hbox/xgboost/data
执行程序地址：  hdfs://namenode.safe.lycc.qihoo.net:9000/home/xitong/hbox/xgboost/demo
xgboost dist包地址：hdfs://namenode.safe.lycc.qihoo.net:9000/home/xitong/hbox/xgboost/dist/xgboost.tgz
hdfs://namenode.safe.lycc.qihoo.net:9000/home/xitong/hbox/xgboost/dist/xgboost-0.7.tgz
提交脚本中，注意指定 --app-type "xgboost"
用户需要在执行脚本里export环境变量如下：
export PYTHONPATH=xgboost/python-package:$PYTHONPATH
export LD_LIBRARY_PATH=~ /software/hadoop/lib/native:$LD_LIBRARY_PATH
用户在提交脚本增加参数如下：
--cacheArchive hdfs://namenode.safe.lycc.qihoo.net:9000/home/xitong/hbox/xgboost/dist/xgboost.tgz#xgboost
分布式版本的xgboost demo所在hdfs路径如下：
数据存放地址：  hdfs://namenode.safe.lycc.qihoo.net:9000/home/xitong/hbox/xgboost/distData
执行程序地址：  hdfs://namenode.safe.lycc.qihoo.net:9000/home/xitong/hbox/xgboost/distDemo
由于xgboost分布式使用的是dmlc的rabit tracker，所以提交脚本不需要设置ps。此外提交脚本中，注意指定 --app-type "distxgboost"
用户需要在执行脚本里export环境变量如下：
export PYTHONPATH=xgboost/python-package:$PYTHONPATH
export LD_LIBRARY_PATH=~ /software/hadoop/lib/native:$LD_LIBRARY_PATH
用户在提交脚本增加参数如下：
--app-type "distxgboost"
--cacheArchive hdfs://namenode.safe.lycc.qihoo.net:9000/home/xitong/hbox/xgboost/dist/xgboost.tgz#xgboost
集群提供的xgboost包的版本为：0.6
```

tensorflow demo所在hdfs路径如下：

```
数据存放地址：  hdfs://namenode.safe.lycc.qihoo.net:9000/home/xitong/hbox/tensorflowOnYarn/demo/data
执行程序地址：
  hdfs://namenode.safe.lycc.qihoo.net:9000/home/xitong/hbox/tensorflow/demo/src/single 单机多卡模式作业
  hdfs://namenode.safe.lycc.qihoo.net:9000/home/xitong/hbox/tensorflow/demo/src/distribute 分布式模式作业
提交脚本中，注意指定 --app-type "tensorflow"
若集群中没有安装tensorflow或者用户想使用自己编译的tensorflow版本，则可以通过cacheArchive传tensorflow包的方式运行
只需在运行脚本里面加个参数并且在用户的执行脚本里export下PYTHONPATH环境变量就可以了
export PYTHONPATH=./:$PYTHONPATH
--cacheArchive hdfs://namenode.safe.lycc.qihoo.net:9000/home/xitong/hbox/tensorflow/cacheArchive/tensorflow-1.2.1.tgz#tensorflow
集群安装的tensorflow版本为：1.0.0rc2
集群提供的tensorflow的cacheArchive，1.2.1版本的路径为：hdfs://namenode.safe.lycc.qihoo.net:9000/home/xitong/hbox/tensorflow/cacheArchive/tensorflow-1.2.1.tgz
集群提供的tensorflow的cacheArchive，1.3.0版本的路径为：hdfs://namenode.safe.lycc.qihoo.net:9000/home/xitong/hbox/tensorflow/cacheArchive/tensorflow-1.3.0.tgz
集群提供的tensorflow的cacheArchive，1.4.0版本的路径为：hdfs://namenode.safe.lycc.qihoo.net:9000/home/xitong/hbox/tensorflow/cacheArchive/tensorflow-1.4.0.tgz
集群提供的tensorflow的cacheArchive，1.7.0版本的路径为：hdfs://namenode.safe.lycc.qihoo.net:9000/home/xitong/hbox/tensorflow/cacheArchive/tensorflow-1.7.0.tgz
集群提供的tensorflow的cacheArchive，1.8.0版本的路径为：hdfs://namenode.safe.lycc.qihoo.net:9000/home/xitong/hbox/tensorflow/cacheArchive/tensorflow-1.8.0.tgz
集群提供的tensorflow的cacheArchive，1.9.0版本的路径为：hdfs://namenode.safe.lycc.qihoo.net:9000/home/xitong/hbox/tensorflow/cacheArchive/tensorflow-1.9.0.tgz
集群提供的tensorflow的cacheArchive，1.10.0版本的路径为：hdfs://namenode.safe.lycc.qihoo.net:9000/home/xitong/hbox/tensorflow/cacheArchive/tensorflow-1.10.0.tgz
集群提供的tensorflow的cacheArchive，1.10.1(GPU版本)的路径为：hdfs://namenode.safe.lycc.qihoo.net:9000/home/xitong/hbox/tensorflow/cacheArchive/tensorflow-1.10.1-gpu.tgz
集群提供的tensorflow的cacheArchive，1.10.1(CPU版本)的路径为：hdfs://namenode.safe.lycc.qihoo.net:9000/home/xitong/hbox/tensorflow/cacheArchive/tensorflow-1.10.1-cpu.tgz
集群提供的tensorflow的cacheArchive，1.12.0版本的路径为：hdfs://namenode.safe.lycc.qihoo.net:9000/home/xitong/hbox/tensorflow/cacheArchive/tensorflow-1.12.0.tgz
集群提供的tensorflow的cacheArchive，1.13.1(CPU版本)版本的路径为：hdfs://namenode.safe.lycc.qihoo.net:9000/home/xitong/hbox/tensorflow/cacheArchive/tensorflow-1.13.1-cpu.tgz
集群提供的tensorflow的cacheArchive，1.13.1(GPU版本)版本的路径为：hdfs://namenode.safe.lycc.qihoo.net:9000/home/xitong/hbox/tensorflow/cacheArchive/tensorflow-1.13.1.tgz
```

由于1.3.0以上的版本需要cudnn6以上，我们已经将所需要的cudnn库和tensorflow一起打包了，用户需要在执行脚本里export环境变量如下：

```
export PYTHONPATH=./:$PYTHONPATH
export LD_LIBRARY_PATH=./tensorflow/cuda/lib64:./tensorflow/cuda/extras/CUPTI/lib64:$LD_LIBRARY_PATH
用户在提交脚本增加参数如下：
--cacheArchive hdfs://namenode.safe.lycc.qihoo.net:9000/home/xitong/hbox/tensorflow/cacheArchive/tensorflow-x.x.x.tgz#tensorflow
注意：运行tensorflow1.4版本时，--cacheArchive需要添加hdfs://namenode.safe.lycc.qihoo.net:9000/home/xitong/hbox/tensorflow/cacheArchive/enum34.tgz#enum
```

注意：tensorflow1.10.0 以上版本要求google.protobuf版本 > 集群安装版本，--cacheArchive 需添加hdfs://namenode.safe.lycc.qihoo.net:9000/home/xitong/hbox/tensorflow/cacheArchive/protobuf3.6.0.tgz ，并且在执行脚本export环境变量前执行如下操作：

```
mkdir google
cp -r protobuf3.6.0.tgz/* google/
```

注意：tensorflow1.12.0 版本中的keras_applications等模块一起打入tensorflow-1.12.0.tgz，因此设置环境变量时，需要更新：

```
export PYTHONPATH=`pwd`/:`pwd`/tensorflow:$PYTHONPATH
```

注意：tensorflow1.13.1-cpu 版本使用时，--cacheArchive hdfs://namenode.safe.lycc.qihoo.net:9000/home/xitong/hbox/tensorflow/cacheArchive/tensorflow-1.13.1-cpu.tgz#tensorflow 通过以下方式设置环境变量：

```
export PYTHONPATH=`pwd`/tensorflow:$PYTHONPATH
```

注意：tensorflow1.13.1-gpu 版本使用时，其要求cuda-10.0，cudnn-7.5版本，因此提交脚本中指定：

```
--cacheArchive hdfs://namenode.safe.lycc.qihoo.net:9000/home/xitong/hbox/tensorflow/cacheArchive/tensorflow-1.13.1.tgz#tensorflow,hdfs://namenode.safe.lycc.qihoo.net:9000/home/xitong/hbox/cacheArchive/cuda-10.0-cudnn-7.5.tgz#cuda-10.0
```

并通过以下方式设置环境变量：

```
export PYTHONPATH=`pwd`/tensorflow:$PYTHONPATH
export LD_LIBRARY_PATH=`pwd`/cuda-10.0/lib64:$LD_LIBRARY_PATH
```

tensorflow estimator demo 所在hdfs路径如下：

```
hdfs://namenode.safe.lycc.qihoo.net:9000/home/xitong/hbox/tfEstimator
```

其中，分布式版本中的TF_CONFIG构建方式可进行自行扩展

tensor2tensor demo所在hdfs路径如下：

```
hdfs://namenode.safe.lycc.qihoo.net:9000/home/xitong/hbox/distT2T
```

pytorch demo所在hdfs路径如下：

数据存放地址： hdfs://namenode.safe.lycc.qihoo.net:9000/home/xitong/hbox/pytorch/data
执行程序地址： hdfs://namenode.safe.lycc.qihoo.net:9000/home/xitong/hbox/pytorch/demo
提交脚本中，注意指定 --app-type "pytorch"
集群安装的pytorch的版本为：0.1.11.5
集群提供的pytorch的cacheArchive，0.2.0.3版本的路径为： hdfs://namenode.safe.lycc.qihoo.net:9000/home/xitong/hbox/pytorch/cacheArchive/torch-0.2.tgz
用户需要在执行脚本里export环境变量如下：

```
export PYTHONPATH=./:$PYTHONPATH
用户在提交脚本增加参数如下：
--cacheArchive hdfs://namenode.safe.lycc.qihoo.net:9000/home/xitong/hbox/pytorch/cacheArchive/torch-0.2.tgz#torch
```

集群提供的pytorch的cacheArchive，0.4.0版本的路径为： hdfs://namenode.safe.lycc.qihoo.net:9000/home/xitong/hbox/pytorch/cacheArchive/torch-0.4.0.tgz
由于0.4.0以上若使用cudnn6以上，我们已经将所需要的cudnn库和pytorch一起打包了，用户需要在执行脚本里export环境变量如下：

```
export PYTHONPATH=`pwd`/:$PYTHONPATH
export LD_LIBRARY_PATH=`pwd`/torch/cuda-9.0/lib64:~`pwd`/torch/cuda-9.0/extras/CUPTI/lib64:$LD_LIBRARY_PATH
```

集群提供的pytorch-1.1.0 cuda-10 版本： hdfs://namenode.safe.lycc.qihoo.net:9000/home/xitong/hbox/pytorch/cacheArchive/torch-1.1.0-cu100.tgz
使用方法：

```
--cacheArchive hdfs://namenode.safe.lycc.qihoo.net:9000/home/xitong/hbox/pytorch/cacheArchive/torch-1.1.0-cu100.tgz#torch-1.1.0,hdfs://namenode.safe.lycc.qihoo.net:9000/home/xitong/hbox/cacheArchive/cuda-10.0-cudnn-7.5.tgz#cuda-10.0
```

并通过以下方式设置环境变量：

```
export PYTHONPATH=`pwd`/torch-1.1.0:$PYTHONPATH
export LD_LIBRARY_PATH=`pwd`/cuda-10.0/lib64:$LD_LIBRARY_PATH
```

分布式pytorch示例[TCP通信机制]：

```
执行程序地址： hdfs://namenode.safe.lycc.qihoo.net:9000/home/xitong/hbox/pytorch/distDemo
```

其中，分布式需要的rank，world_size，init_method，可通过环境变量获取，方式如下，详情可见提供的示例

```
rank=os.environ["RANK"]
world_size=os.environ["WORLD_SIZE"]
init_method=os.environ["INIT_METHOD"]
```

keras demo[默认基于tensorflow]所在hdfs路径如下：

```
数据存放地址： hdfs://namenode.safe.lycc.qihoo.net:9000/home/xitong/hbox/keras/data/
执行程序地址： hdfs://namenode.safe.lycc.qihoo.net:9000/home/xitong/hbox/keras/demo/
提交脚本中，注意指定 --app-type "keras"
集群安装的keras的版本为： 2.0.3
```

theano demo所在hdfs路径如下：

```
数据存放地址： hdfs://namenode.safe.lycc.qihoo.net:9000/home/xitong/hbox/theano/data/
执行程序地址： hdfs://namenode.safe.lycc.qihoo.net:9000/home/xitong/hbox/theano/demo/
提交脚本中，注意指定 --app-type "theano"
集群安装的theano版本为： 0.9.0
集群提供的theano的cacheArchive，1.0.1版本的路径为： hdfs://namenode.safe.lycc.qihoo.net:9000/home/xitong/hbox/theano/dist/theano-1.0.1.tgz
用户需要在执行脚本里export环境变量如下：
export PYTHONPATH=`pwd`/:$PYTHONPATH
用户在提交脚本增加参数如下：
--cacheArchive hdfs://namenode.safe.lycc.qihoo.net:9000/home/xitong/hbox/theano/dist/theano-1.0.1.tgz#theano
```

mxnet demo所在hdfs路径如下：

```
数据存放地址： hdfs://namenode.safe.lycc.qihoo.net:9000/home/xitong/hbox/mxnet/data/
执行程序地址： hdfs://namenode.safe.lycc.qihoo.net:9000/home/xitong/hbox/mxnet/demo/
分布式版本和单机版本demo都包含了，提交脚本中，注意指定 --app-type "mxnet"
若集群中没有安装mxnet或者用户想使用自己编译的mxnet版本，则可以通过传mxnet包的方式运行
已经安装好的mxnet包路径为： hdfs://namenode.safe.lycc.qihoo.net:9000/home/xitong/hbox/mxnet/dist/mxnet.tgz
执行程序地址为： hdfs://namenode.safe.lycc.qihoo.net:9000/home/xitong/hbox/mxnet/demo_with_user_define_mxnet
集群安装和提供的mxnet包的版本都为： 0.10.0
```

集群提供的sklearn的cacheArchive，0.19.0版本的路径为： hdfs://namenode.safe.lycc.qihoo.net:9000/tmp/sklearn-0.19.0.tgz

用户需要在执行脚本里export环境变量如下：

```
export PYTHONPATH=./:$PYTHONPATH
```

用户在提交脚本增加参数如下：

```
--cacheArchive hdfs://namenode.safe.lycc.qihoo.net:9000/tmp/sklearn-0.19.0.tgz#sklearn
```

分布式版本的lightGBM demo所在hdfs路径如下：

数据存放地址： hdfs://namenode.safe.lycc.qihoo.net:9000/home/xitong/hbox/lightGBM/data

执行程序地址： hdfs://namenode.safe.lycc.qihoo.net:9000/home/xitong/hbox/lightGBM/demo

执行分布式lightGBM需要在用户程序的配置文件中指定机器数目和本地端口号，此处我们可以直接从环境变量中获取到，因此用户需要在执行脚本里将这两个参数写入到配置文件，

由于多个container分配到同一台机器，--files传的文件会软链到同一个文件，所以为了支持一台机器调度多个container，需要复制下conf文件到可执行脚本的当前目录，如下：

```
cp train.conf train_real.conf
chmod 777 train_real.conf
echo "num_machines = $LIGHTGBM_NUM_MACHINE" >> train_real.conf
echo "local_listen_port = $LIGHTGBM_LOCAL_LISTEN_PORT" >> train_real.conf
./LightGBM/lightgbm config=train_real.conf
~
```

此外用户还需要在程序配置文件中指定机器列表文件，该文件我们统一命名为lightGBMlist.txt，会在每个worker的执行目录生成，用户在程序配置文件指定参数如下：

```
machine_list_file = lightGBMlist.txt
```

用户在提交脚本增加参数如下：
--app-type "distlightgbm"
--cacheArchive hdfs://namenode.safe.lycc.qihoo.net:9000/home/xitong/hbox/lightGBM/dist/LightGBM.tgz#LightGBM

分布式版本的lightLDA demo所在hdfs路径如下：

数据存放地址： hdfs://namenode.safe.lycc.qihoo.net:9000/home/xitong/hbox/lightLDA/data, hdfs://namenode.safe.lycc.qihoo.net:9000/home/xitong/hbox/lightLDA/dict
执行程序地址： hdfs://namenode.safe.lycc.qihoo.net:9000/home/xitong/hbox/lightLDA/demo
分布式lightLDA，用户的worker端执行需要指定server的endpoint文件列表，该文件我们统一命名为lightLdaEndPoints.txt，会在每个worker的执行目录生成，用户在执行脚本指定文件名即可。
使用细节请参考demo
用户在提交脚本增加参数如下：
--app-type "distlightlda"
--cacheArchive hdfs://namenode.safe.lycc.qihoo.net:9000/home/xitong/hbox/lightLDA/dist/lightLDA.tgz#lightLDA

xfow demo所在hdfs路径如下：

数据存放地址： hdfs://namenode.safe.lycc.qihoo.net:9000/home/xitong/hbox/xfow/data
执行程序地址： hdfs://namenode.safe.lycc.qihoo.net:9000/home/xitong/hbox/xfow/demo
该框架作业类型如下：
--app-type "xfow"
因程序执行需要libzmq.so文件，需要用户上传并设置环境变量，详情请参见demo

horovod demo所在hdfs路径如下：

数据存放地址： hdfs://namenode.safe.lycc.qihoo.net:9000/home/xitong/horovod-test/data
执行程序地址： hdfs://namenode.safe.lycc.qihoo.net:9000/home/xitong/horovod-test/demo
该框架作业类型如下：
--app-type "horovod"
普通的tensorflow类型的作业可以通过添加以下几行代码变成horovod作业，从而使得分布式性能随着机器增加而线性增长。

```
import horovod.tensorflow as hvd  
hvd.init()  
opt = hvd.DistributedOptimizer(tensorflowOpt)
```


具体代码的添加位置和对tensorflow优化器的封装形式可以参见demo示例和horovod官方文档。

cuda高版本cacheArchive包列表：

hdfs://namenode.safe.lycc.qihoo.net:9000/home/xitong/hbox/cacheArchive/cudnn-6-cuda-8.0.tgz
hdfs://namenode.safe.lycc.qihoo.net:9000/home/xitong/hbox/cacheArchive/cudnn-7.1.4-cuda-9.0.tgz
hdfs://namenode.safe.lycc.qihoo.net:9000/home/xitong/hbox/cacheArchive/cuda-10.0-cudnn-7.5.tgz
注：集群未提供的，用户可自行在[1] (<https://developer.nvidia.com/rdp/cudnn-archive>) 网站下载

FAQ

1) Q：在hbox中，caffe、xgboost等需要的dist包如何传给container？

A：可通过--files 上传本地对应文件或文件夹，--cacheFile 上传hdfs对应的文件或文件夹， --cacheArchive 上传hdfs上对应的压缩文件。注意，文件或文件夹的目录层级，尤其是压缩文件解压后的目录。

2) Q：参数input目录下的文件在各worker上是如何分布的？

A: 根据input目录下的文件数目, 轮询的将各文件从hdfs下载至各worker对应container下的本地文件夹内(即参数设置时“#”后所设的本地文件夹名称), 此处应注意所设worker数目是否合适, 是否会出现文件数目<worker数目的情况。

3) Q: 若参数output未指定本地文件路径该怎么办?

A: 若用户仅指定有output的HDFS文件路径, 则默认将本地文件夹“output”中的内容上传至指定的HDFS路径下。

4) Q: 若存在用户自定义module于其他python文件中, 如何处理?

A: 利用files参数, 添加所需要的所有python文件, 在调用其他自定义模块前, 将python文件所在路径添加至系统路径, 如: `sys.path.append(os.getcwd())`。

5) Q: ApplicationMaster 页面中 “Save Model” 按钮的用处?

A: 如果用户在程序执行过程中希望上传当前output输出目录中的文件至对应HDFS路径, 可点击“Save Model”按钮, 对应文件将输出至用户所设HDFS输出路径下的interResult文件夹中, 自行查看。注意: 由于用户的输出目录结果上传至HDFS的操作是在程序执行结束后进行的, 所以如果用户希望提前终止程序运行, 需要先使用此功能上传, 再进行作业的kill操作。

6) Q: Keras作业执行的默认后端是tensorflow, 如何使用theano作为后端?

A: 可在执行脚本中设置环境变量 "KERAS_BACKEND=theano"。

7) Q: tensorflow分布式模式中, 程序中如何获取clusterSpec、当前节点的 job_name 及 task_index?

A: AM通过环境变量 TF_CLUSTER_DEF、TF_ROLE、TF_INDEX 对应的将clusterSpec、job_name、task_index等信息传送给各container, 详情请见tensorflow的分布式demo示例。

8) Q: tensorboard服务如何开启?

A: 提交脚本时指定参数--board-enable为true, 若用户没指定tensorboard的logdir, 则用户需使用eventLog这个默认目录, 用户可通过--board-logdir自行指定

9) Q: 作业的失败状态是如何判定的?

A: 对于tensorflow分布式模式程序, 若一个worker判定为失败, 则整个作业失败。对于tensorflow单机模式及caffe、xgboost等单机程序, 可通过设置 --conf hbox.container.maxFailures.rate=0.x 参数来指定可容忍的最大worker失败比例【默认为0.5】, 以此来作为作业失败的依据, 即在默认情况下, 若作业启动的worker数目为6, 则当worker失败数目>=3时, 则作业为失败状态。

10) Q: 作业如何查看各个container的运行进度?

A: 需要用户在提交的程序中计算出程序运行的进度打印到标准错误输出, 打印的格式为"reporter progress:0.20\n", 以python程序为例: `sys.stderr.write("reporter progress:%0.2f\n"%progress)`, hbox会将进度信息以进度条的方式显示在web ui。

11) Q: 作业中各container如何获取分配的GPU数目?

A: AM通过环境变量 GPU_NUM 将gpu数目分配信息传送给各container。

12) Q: 如何在python代码中读写hdfs文件?

A: 详见wiki: <http://wiki.sys.corp.qihoo.net/mediawiki/index.php/Pydoop%E4%BD%BF%E7%94%A8%E6%96%87%E6%A1%A3>

13) Q: 如何在C/C++代码中读写hdfs文件?

A: 详见: <http://wiki.sys.corp.qihoo.net/mediawiki/index.php/Libhdfs%E4%BD%BF%E7%94%A8%E6%96%87%E6%A1%A3>

14) 集群使用的cuda和cudnn的版本号是多少?

A: 集群使用的cuda版本号为8.0, cudnn版本号为5.0

15) 如何利用spark和hadoop来处理生成标准的tensorflow数据格式?

A: **tensorflow ecosystem**项目的两个子项目提供了该功能

tensorflow ecosystem hadoop: 利用了hadoop 的InputFormat/OutputFormat接口实现了tensorflow的TFRecords数据格式生成, 同时也能用spark来处理。
tensorflow ecosystem spark-tensorflow-connector: 实现了tensorflow的TFRecords数据格式和Spark SQL DataFrames数据的相互导入与导出。
tensorflow ecosystem具体细节可参考其git hub地址: <https://github.com/tensorflow/ecosystem>
tensorflow ecosystem spark使用demo地址: hdfs://namenode.safe.lycc.qihoo.net:9000/tmp/tfEcosystemSparkDemo
tensorflow ecosystem hadoop使用demo地址: hdfs://namenode.safe.lycc.qihoo.net:9000/tmp/tfEcosystemHadoopDemo
最新版本的基于spark2.3的对应jar包路径如下, demo使用可参见官方github介绍
hdfs://namenode.safe.lycc.qihoo.net:9000/tmp/tfEcosystemSparkDemo/tensorflow-hadoop-1.10.0.jar
hdfs://namenode.safe.lycc.qihoo.net:9000/tmp/tfEcosystemSparkDemo/spark-tensorflow-connector_2.11-1.10.0.jar

16) Tensorflow中, 环境变量TF_CONFIG如何利用已知变量进行构建?

以Tensorflow Estimator分布式中, chief模式下的环境变量TF_CONFIG的构建为例:

```
import os
import json

cluster = json.loads(os.environ["TF_CLUSTER_DEF"])
task_index = int(os.environ["TF_INDEX"])
task_type = os.environ["TF_ROLE"]

# chief: worker 0 as chief, other worker index --
tf_config = dict()
worker_num = len(cluster["wroker"])
if task_type == "ps":
    tf_config["task"] = {"index":task_index, "type":task_type}
elif task_type == "worker":
    if task_index == 0:
        tf_config["task"] = {"index":0, "type":"chief"}
    else:
        tf_config["task"] = {"index":task_index-1, "type":task_type}
elif task_type == "evaluator":
    tf_config["task"] = {"index":task_index, "type":task_type}

if worker_num == 1:
    cluster["chief"] = cluster["worker"]
    del cluster["worker"]
else:
    cluster["chief"] = [cluster["worker"][0]]
    del cluster["worker"][0]

tf_config["cluster"] = cluster
os.environ["TF_CONFIG"] = json.dumps(tf_config)
```

由此, 可利用Tensorflow分布式模式下, Hbox提供的环境变量 TF_CLUSTER_DEF、TF_ROLE、TF_INDEX 对应的来构建所需的环境变量TF_CONFIG。
hbox1.4版本以上, TensorFlow Estimator 分布式在worker数目>1的情况下, 可通过设置提交参数 "--tf-evaluator "为true, 此时, 最后一个worker将视为evaluator角色。

17) Q: HBox客户端如何提交MPI类型作业到safe lycc 集群?

A: 在客户端提交中, 指定集群信息: --cluster "safe-lycc"

18) Q: python程序报出cudnn版本不匹配, 类似 Loaded runtime CuDNN library: 7.0.3 but source was compiled with: 7.1.4 , 如何解决?

A: 目前集群机器及集群所提供的各版本包中所含的cudnn版本为7.0.3, 若出现此问题, 可在--cacheArchive参数中追加集群提供的cudnn高版本包(见demo部分所提供的路径列表), 并优先加载到环境变量中, 例如:

```
--cacheArchive hdfs://namenode.safe.lycc.qihoo.net:9000/home/xitong/hbox/tensorflow/cacheArchive/tensorflow-1.12.0.tgz#tensorflow,hdfs://namenode.safe.lycc.qihoo.net:9000/home/xitong/hbox/tensorflow/cacheArchive/protobuf3.6.0.tgz,hdfs://namenode.safe.lycc.qihoo.net:9000/home/xitong/hbox/tensorflow/cacheArchive/cudnn-7.0.3.tgz#cudnn
在hbox-cmd的执行脚本中, 优先追加 ./cudnn/lib64 至环境变量 LD_LIBRARY_PATH, 如 export LD_LIBRARY_PATH=./cudnn/lib64:./tensorflow/cuda/lib64:./tensorflow/cuda/extras/CUPTI/lib64:$LD_LIBRARY_PATH
```

失败作业常见问题排查

用户脚本执行错误

- 作业执行失败后，可进入History页面，点击containerID，查看container日志信息，判定脚本执行失败问题所在。
- 若脚本执行失败是由于python模块导入失败，如"**ImportError: No module named xxx**":
 - 若模块为单独的python文件，1) 检查是否通过提交参数指定所需文件；2) 检查引用代码中是否添加有sys.path.append(os.getcwd())
 - 若模块为文件夹或压缩文件后的解压文件，1) 检查提交脚本中是否指定有PYTHONPATH，如：“export PYTHONPATH=./:\$PYTHONPATH”，并注意检查模块文件所在的绝对路径是否和指定的PYTHONPATH路径符合（主要出现在压缩文件解压后的文件夹嵌套）。

可分配资源不足

- 日志页面显示有"**Waiting for all containers allocated ...**"，集群可分配资源或队列资源不满足作业资源需求，在等待container全部分配启动，AM日志中有详细container分配进度。

container执行过程中无故退出

- 若作业失败时，container日志处于正常输出状态。可进入Application页面，查看AM日志信息，是否存在日志"**Container xxx timed out after xxx seconds**"。若有此日志，可进入History页面，查看container metrics曲线图，memory是否超出内存申请大小。若因内存超限导致作业失败，请调大"**worker-memory**"或"**ps-memory**"的申请。

container未成功启动，History页面报出"Not get the container information, please check all containers started successfully !" 或 "Job History Log getting error !"

- 在集群cluster页面，点击applicationID，进入Application页面，查看AM日志信息，根据日志"**Container waiting except the allocated expiry time. Maybe the Cluster resource not satisfied the user necessary. Please resubmit !**"判定是否因container资源申请超时而造成作业失败退出。通常该情况下集群资源相对紧张。

container未成功启动，History页面显示container的状态为UNDEFINED

- 在集群cluster页面，点击applicationID，进入Application页面，查看AM日志信息，根据日志"**Container xxx local resource timed out after xxx seconds**"判定是否因container下载所需资源超时导致失败。若为此原因，可在提交参数中设置"**--conf hbox.localresource.timeout=xxx**"来调整超时设置，单位为毫秒。

申请使用

发邮件到 **g-xitong-ai@360.cn**，告知 **hadoop账号** 和 **客户端地址** 即可。

新功能预告

取自 “<http://wiki.sys.corp.qihoo.net:80/mediawiki/index.php?title=HBox&oldid=12317>”

- 本页面最后修改于2019年7月30日 (星期二) 11:18。
- 本页面已经被访问过7,811次。