

Spark & Tensorflow 解决方案

Ruichen Wang

May 9, 2019

1 应用场景& 问题描述

1.1 应用场景

基于深度学习的推荐系统。应用CNN,RNN, DNN等相关深度网络结构, 如Deep and Wide, deepFM, xdeepFM, DCN等等。

1.2 问题描述

现在的测试or生产环境中, 要么就是不用深度学习相关的框架, 采用spark自带的mllib或者sklearn来处理数据。要么就是先把数据collect()到某一台机器上的内存上, 再在这节点单机来运行。

采用collect这种方法主要有几个坏处, 一个是collect操作非常耗时, 其他节点都基本处于空闲阶段, 造成资源浪费。二是如果数据量比较大, collect到一台机器的内存上也是不现实的。@jufeng也已经提到过类似的问题。

2 目前解决方案

最好的解决方案是分布式地数据输入模型+ 分布式地训练模型。

2.1 分布式地数据输入模型

目前能结合spark从hdfs读写数据的官方API仅有tensorflow。Pytorch目前官方并没有读写hdfs的支持。也不像tf那样与spark能够直接结合。

数据流主要步骤: hive table → spark dataframe → hdfs格式的tfrecord

tfrecord是tensorflow官方的一支持流格式的二进制数据。一般建议每个tfrecord大

小在100~200MB之间。能够保证数据读取的线性性能。可以用来存储任务预处理的数据。

使用tfrecord的优势：模型读取数据不需要collect到单节点，可以直接读hdfs上的tfrecord。spark dataframe转成tfrecord也是采用分布式写，大概2000w行的dataframe，大小约5个G，只需要5秒钟。tensorflow模型可以直接读取hdfs上的tfrecord，不需要下载到运行节点或者先load到内存，而是在训练时按batch size流式的取。

相关的包和环境已经配置好了，目前测试环境已经支持，线上环境还没有配置，待运维部署。

2.2 示例代码

```
>>> df=ss.sql('select * from g_tfrecord_test')
>>> df
DataFrame[id: int, IntegerCol: int, LongCol: bigint, FloatCol: float, DoubleCol: double, VectorCol: vector, StringCol: string]
>>> df.show()
+-----+-----+-----+-----+-----+-----+-----+
| id|IntegerCol|LongCol|FloatCol|DoubleCol| VectorCol|StringCol|
+-----+-----+-----+-----+-----+-----+-----+
| 11|         1|    23|   10.0|    14.0|[1.0, 2.0]|      r1|
| 21|         2|    24|   12.0|    15.0|[2.0, 2.0]|      r2|
+-----+-----+-----+-----+-----+-----+-----+
```

Figure 1: hive table.

spark写tfrecord，tfrecord其实也是key value的格式，key对应dataframe里的col name. value 自然就是数据本身了。

```
df.repartition(10).write.format("tfrecords").option("recordType",
    "Example").save(path)
```

Home / user / wangrc / mnist.tfrecord / train

<input type="checkbox"/>	Name	Size
<input type="checkbox"/>	↑	
<input type="checkbox"/>	.	
<input type="checkbox"/>	_SUCCESS	0 bytes
<input type="checkbox"/>	part-r-00000	5.4 MB
<input type="checkbox"/>	part-r-00001	5.4 MB
<input type="checkbox"/>	part-r-00002	5.4 MB
<input type="checkbox"/>	part-r-00003	5.4 MB
<input type="checkbox"/>	part-r-00004	5.4 MB
<input type="checkbox"/>	part-r-00005	5.4 MB
<input type="checkbox"/>	part-r-00006	5.4 MB
<input type="checkbox"/>	part-r-00007	5.4 MB
<input type="checkbox"/>	part-r-00008	5.4 MB
<input type="checkbox"/>	part-r-00009	5.4 MB

Figure 2: hdfs tfrecord

spark读tfrecord，可以直接读回来成为dataframe

```
df = spark.read.format("tfrecords").option("recordType",
    "Example").load(path)
```

注意：使用tensorflow读取hdfs时需要确保CLASSPATH变量设置好，基于安全等一些因素考虑目前没有设置为默认。需要在sh脚本里添加两行export 代码，加在spark-submit之前就可以了：

```
#!/bin/bash
source /etc/profile
export HADOOP_HOME=/tmp/wangrc/hadoop-2.7.3
export CLASSPATH=${HADOOP_HOME}/bin/hadoop classpath --glob)
spark-submit \
--master yarn \
--conf spark.network.timeout=600 \
--conf spark.sql.shuffle.partitions=10 \
--conf spark.executor.memoryOverhead=2048 \
--conf spark.driver.memoryOverhead=2048 \
--executor-cores 1 \
--num-executors 1 \
--executor-memory 4g \
--driver-memory 10g \
train_from_tfrecord.py
```

tensorflow读tfrecord-推荐使用这种方法, tf dataset 有map方法可以进行decode和一些的特征、数值处理

```
import tensorflow as tf
tf.enable_eager_execution()
raw_dataset =
    tf.data.TFRecordDataset(['hdfs://cluster/user/wangrc/test1.tfrecord/part-r-00000'])
for raw_record in raw_dataset.take(2):
    print(repr(raw_record))
```



```
>>> for raw_record in raw_dataset.take(2):
...     print(repr(raw_record))
...  # 用 tar -xvf 解压
<tf.Tensor: id=14, shape=(), dtype=string, numpy=b'\n\x91\x01\n\x0b\n\x02id\x12\x05\x1a\x03\n\x01\x0b\n\x13\n\nIntegerCol\x12\x05\x1a\x03\n\x01\x01\n\x10\n\x07LongCol\x12\x05\x1a\x03\n\x01\x17\n\x14\n\x08FloatCol\x12\x08\x12\x06\n\x04\x00\x00 A\n\x15\n\tDoubleCol\x12\x08\x12\x06\n\x04\x00\x00`A\n\x19\n\tVectorCol\x12\x0c\x12\n\n\x08\x00\x00\x80?\x00\x00\x00@\n\x13\n\tStringCol\x12\x06\n\x04\n\x02r1'>
<tf.Tensor: id=16, shape=(), dtype=string, numpy=b'\n\x91\x01\n\x0b\n\x02id\x12\x05\x1a\x03\n\x01\x15\n\x13\n\nIntegerCol\x12\x05\x1a\x03\n\x01\x02\n\x10\n\x07LongCol\x12\x05\x1a\x03\n\x01\x18\n\x14\n\x08FloatCol\x12\x08\x12\x06\n\x04\x00\x00@A\n\x15\n\tDoubleCol\x12\x08\x12\x06\n\x04\x00\x00pA\n\x19\n\tVectorCol\x12\x0c\x12\n\n\x08\x00\x00\x00@\x00\x00\x00@\n\x13\n\tStringCol\x12\x06\n\x04\n\x02r2'>
```

Figure 3: 二进制tfrecord

tensorflow读tfrecord-似乎已经DEPRECATED, 方便大家调试的时候用, 可以直接看到json格式的key, value。

```
import tensorflow as tf
record_iterator =
    tf.python_io.tf_record_iterator(path='hdfs://cluster/user/wangrc/test1.tfrecord/part-
for string_record in record_iterator:
    example = tf.train.Example()
    example.ParseFromString(string_record)
    print(example)
```

```

feature {
  key: "FloatCol"
  value {
    float_list {
      value: 10.0
    }
  }
}
feature {
  key: "IntegerCol"
  value {
    int64_list {
      value: 1
    }
  }
}

```

Figure 4: json tfrecord

3 模型Parse & Train

建议使用第一种形式读取，tensorflow提供了非常易用的dataset API, 支持源生的tf训练和keras model fit.

```

feature_description = {
    'label': tf.FixedLenFeature([], tf.int64),
    'features': tf.FixedLenFeature([784], tf.int64),
}

def _parse_function(example_proto):
    parsed_feature= tf.parse_single_example(example_proto,
        feature_description)
    return tf.reshape(tf.cast(parsed_feature['features'],
        tf.float32), [28,28,1]), \
        tf.cast(parsed_feature['label'], tf.float32)
dataset = raw_dataset.map(_parse_function,num_parallel_calls=4)

```

Layer (type)	Output Shape	Param #	Connected to						
input_1 (InputLayer)	(None, 28, 28, 1)	0							
lambda (Lambda)	(None, 28, 28, 1)	0	input_1[0][0]						
lambda_1 (Lambda)	(None, 28, 28, 1)	0	input_1[0][0]						
model (Model)	(None, 10)	1386506	lambda[0][0] lambda_1[0][0]						
dense_1 (Concatenate)	(None, 10)	0	model[1][0] model[2][0]						
Total params: 1,386,506									
Trainable params: 1,386,506									
Non-trainable params: 0									
Epoch 1/10									
100/100 [=====]									
Epoch 2/10									
100/100 [=====]									
Epoch 3/10									
100/100 [=====]									
Epoch 4/10									
100/100 [=====]									

Figure 5: train

```

Stdoutoutput ['/device:GPU:0', '/device:GPU:1']
Stdoutoutput 2019-05-08 17:03:30.496383: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1511] Adding visible gpu de
Stdoutoutput 2019-05-08 17:03:30.496488: I tensorflow/core/common_runtime/gpu/gpu_device.cc:982] Device interconnect St
Stdoutoutput 2019-05-08 17:03:30.496500: I tensorflow/core/common_runtime/gpu/gpu_device.cc:988] 0 1
Stdoutoutput 2019-05-08 17:03:30.496508: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1001] 0: N Y
Stdoutoutput 2019-05-08 17:03:30.496515: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1001] 1: Y N
Stdoutoutput 2019-05-08 17:03:30.496882: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1115] Created TensorFlow de
Stdoutoutput 2019-05-08 17:03:30.496996: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1115] Created TensorFlow de
Stdoutoutput 2019-05-08 17:03:44.992563: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1511] Adding visible gpu de
Stdoutoutput 2019-05-08 17:03:44.992694: I tensorflow/core/common_runtime/gpu/gpu_device.cc:982] Device interconnect St
Stdoutoutput 2019-05-08 17:03:44.992706: I tensorflow/core/common_runtime/gpu/gpu_device.cc:988] 0 1
Stdoutoutput 2019-05-08 17:03:44.992714: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1001] 0: N Y
Stdoutoutput 2019-05-08 17:03:44.992721: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1001] 1: Y N
Stdoutoutput 2019-05-08 17:03:44.993118: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1115] Created TensorFlow de
Stdoutoutput 2019-05-08 17:03:44.993390: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1115] Created TensorFlow de
Stdoutoutput
Stdoutoutput Layer (type) Output Shape Param # Connected to
Stdoutoutput =====
Stdoutoutput input_1 (InputLayer) (None, 28, 28, 1) 0
Stdoutoutput
Stdoutoutput lambda (Lambda) (None, 28, 28, 1) 0 input_1[0][0]
Stdoutoutput
Stdoutoutput lambda_1 (Lambda) (None, 28, 28, 1) 0 input_1[0][0]
Stdoutoutput
Stdoutoutput model (Model) (None, 10) 1386506 lambda[0][0]
Stdoutoutput lambda_1[0][0]
Stdoutoutput
Stdoutoutput dense_1 (Concatenate) (None, 10) 0 model[1][0]
Stdoutoutput model[2][0]
Stdoutoutput =====
Stdoutoutput Total params: 1,386,506
Stdoutoutput Trainable params: 1,386,506
Stdoutoutput Non-trainable params: 0
Stdoutoutput
Stdoutoutput Epoch 1/10
Stdoutoutput
Stdoutoutput 1/100 [.....] - ETA: 7:46 - loss: 13.6336 - sparse_categorical_accuracy: 0.1094
Stdoutoutput Heart beat
Stdoutoutput 5/100 [>.....] - ETA: 1:30 - loss: 12.7988 - sparse_categorical_accuracy: 0.1781
Stdoutoutput 9/100 [=>.....] - ETA: 48s - loss: 12.0571 - sparse_categorical_accuracy: 0.2248
Stdoutoutput 13/100 [==>.....] - ETA: 32s - loss: 11.4142 - sparse_categorical_accuracy: 0.2689
Stdoutoutput 17/100 [====>.....] - ETA: 24s - loss: 10.9049 - sparse_categorical_accuracy: 0.3031
Stdoutoutput 21/100 [=====>.....] - ETA: 18s - loss: 10.5704 - sparse_categorical_accuracy: 0.3253
Stdoutoutput 23/100 [=====>.....] - ETA: 17s - loss: 10.4368 - sparse_categorical_accuracy: 0.3342
Stdoutoutput 26/100 [=====>.....] - ETA: 14s - loss: 10.1687 - sparse_categorical_accuracy: 0.3511

```

Figure 6: train

示例代码: `/user/wangrc/distribute_demo`

4 说好的分布式训练呢？

我们理想中的分布式训练框架应该有哪些功能呢？

- 支持spark,hive数据读写
- 支持任务部署模式,有机制处理挂起或者失败
- 支持gpu资源弹性分配, 任务级别或者更细节的显卡级别

目前有成熟的解决方案吗？ Not Found

GPU训练： 首先在我们目前的环境中，没有docker, k8s等来做GPU资源管理。在hadoop3.1以后，Databricks、NVIDIA、Google 以及阿里巴巴正在为Apache

Spark添加gpu原生支持。相信目前还有很多问题。目前我们只有spark,hadoop2.7.3,是没有gpu资源管理的。

TensorFlow、PyTorch、XGBoost、LibLinear 这些AI引擎的分布式计算能力都有一些问题。TensorFlow 原生支持分布式训练，但不支持容错，一个进程挂了，整个作业就挂了。虽然这还可以通过checkpointing 解决，但是不容错就不能弹性调度，不能弹性调度就意味着集群利用率可能极差。比如一个有N个GPU 的集群上在运行一个作业，使用了一个GPU；此时一个新提交的作业要求使用N 个GPU，因为空闲GPU 个数是N-1，所以这个新的作业不能开始执行，而是得一直等数小时甚至数天，直到前一个作业结束、释放那个被占用的GPU。这么长时间里，集群利用率 $\leq 1/N$ 。关于这个问题的解决方案，百度PaddleEDL和阿里集团的XDL做了一些很有益的探索。

实际上，databricks也建议大家尽量在集群中使用单机进行训练。（我为我自己开脱！）

Distributed Training

When possible, Databricks recommends that you train neural networks on a single machine; distributed code for training and inference is more complex than single-machine code and slower due to communication overhead. However, you should consider distributed training and inference if your model or your data are too large to fit in memory on a single machine.

Figure 7: Databricks

4.1 Uber的Horovod

Horovod是uber团队开源一个方便分布式训练的项目。能够非常方便的从单机扩展到多机多卡训练。



Horovod is a distributed training framework for TensorFlow, Keras, PyTorch, and MXNet. The goal of Horovod is to make distributed Deep Learning fast and easy to use.

Figure 8: hvd

```
>>> def fn(magic_number):
...     import horovod.torch as hvd
...     hvd.init()
...     print('Hello, rank = %d, local_rank = %d, size = %d, local_size = %d, magic_number = %d' %
...           (hvd.rank(), hvd.local_rank(), hvd.size(), hvd.local_size(), magic_number))
...     return hvd.rank()
...
>>> import horovod.spark
>>> horovod.spark.run(fn, args=(42,))
Running 8 processes (inferred from spark.default.parallelism)...
[Stage 0:->
[1,2]<stdout>:Hello, rank = 2, local_rank = 2, size = 8, local_size = 8, magic_number = 42
[1,0]<stdout>:Hello, rank = 0, local_rank = 0, size = 8, local_size = 8, magic_number = 42
[1,1]<stdout>:Hello, rank = 1, local_rank = 1, size = 8, local_size = 8, magic_number = 42
[1,4]<stdout>:Hello, rank = 4, local_rank = 4, size = 8, local_size = 8, magic_number = 42
[1,7]<stdout>:Hello, rank = 7, local_rank = 7, size = 8, local_size = 8, magic_number = 42
[1,3]<stdout>:Hello, rank = 3, local_rank = 3, size = 8, local_size = 8, magic_number = 42
[1,5]<stdout>:Hello, rank = 5, local_rank = 5, size = 8, local_size = 8, magic_number = 42
[0, 1, 2, 3, 4, 5, 6, 7]
```

Figure 9: 历尽艰难终于在ha01跑通

踩过的坑

- 需要openmpi支持，首先openmpi 必须是3.1.2，官方回复：高级版本据说会有卡死问题。
- tensorflow和hvd必须使用同一版本gcc编译。conda包里的tf与pip hvd不同的gcc不兼容。需要自己重启编译。
- pip包还存在一个已经bug，有些bash命令解析时会多一个括号，几天前刚刚在github源码中fix。而我非常幸运地刚刚好就遇到了这个bug。。所以需要自己下载源码重启编译，其中遇到的细小问题都记了好几页了。
- 另外，hvd没有自己的读取hdfs api,需要petastorm的支持。
- hvd号称自己支持tensorflow_v=1.12, 然而1.13就会报reference error. 直到前天，hvd还在修复一些pytorch nn 的bug。。

不谈hvd api语法等等的学习成本，光是这些千奇百怪的错就已经很折磨人了。

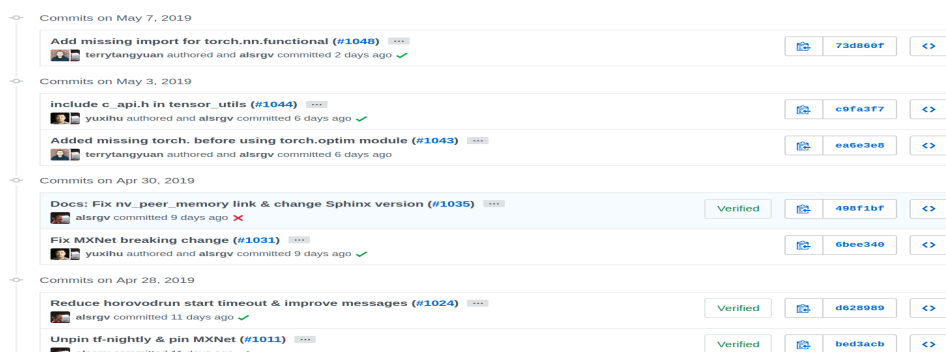


Figure 10: hvd commit list

再看它的多机多卡需求, 要求所有机器都能互相无密码无权限ssh通, 不能有堡垒机的概念了:

2. To run on 4 machines with 4 GPUs each:

```
$ horovodrun -np 16 -H server1:4,server2:4,server3:4,server4:4 python train.py
```

Failures due to SSH issues

The host where `horovodrun` is executed must be able to SSH to all other hosts without any prompts.

If `horovodrun` fails with permission error, verify that you can ssh to every other server without entering a password or answering questions like this:

Figure 11: what??

这和我一台一台机器ssh上去手动跑命令有什么区别???

4.2 多机多卡训练

其实，在推荐系统领域，并没有很庞大的深度学习网络。不像CNN或者RNN一个kernel就有很多参数。一块GPU 训练FFM跑10亿的数据，也就15分钟以内就跑完一轮了。

如果大家真的有需求，需要训练一个超大的深度学习网络，大到2块GPU不能满足你训练要求。推荐使用原生的tensorflow 分布式训练方法：在测试环境里写好master，worker节点和端口，到每台机器上去手动运行，先训练好你的模

型。保存好模型到线上hdfs, 线上单节点预测。在测试环境里随便折腾, 5台机器7块GPU都可以一起训练。

Q: 我不死心, 我就要用多块GPU。我可以写一个sh脚本, 在一个机器上操作吗? 我可以建立定时任务大家一起跑分布式吗?

A: 建议不要。写sh脚本基本不可能。涉及到权限问题, 线上不可能让一个sh脚本操作整个集群的。定时任务也不好, 最主要的还是没有资源管理, 如何知道你的任务成功与否? 跑死机了谁来解决?