使用Kubernetes运行MXNet和AutoTVM

苏磊
TuSimple

自我介绍

- 工作背景：
  - 2008~2011：Platform Computing (Acquired by IBM)
  - 2011~2012：腾讯云平台
  - 2012~2017：IBM Spectrum Computing
  - 2017～至今：图森
- 技能领域：
  - 10+年分布式计算研发，将分布式计算应用于高性能计算、云计算、大数据和车载系统等领域
  - 资源管理、资源调度、作业调度
  - 高性能车载中间件、深度学习优化

US 20160306662A1

(19) **United States**

(12) **Patent Application Publication**

Chin et al.

(10) **Pub. No.: US 2016/0306662 A1**

(43) **Pub. Date: Oct. 20, 2016**

(54) **MULTI-DIMENSION SCHEDULING AMONG MULTIPLE CONSUMERS**

(71) Applicant: **International Business Machines Corporation**, Armonk, NY (US)

(72) Inventors: **Alicia E. Chin**, Markham (CA); **Michael Feiman**, Richmond Hill (CA); **Zhenhua Hu**, Toronto (CA); **Jason T. S. Lam**, Markham (CA); **Zhimin Lin**, Scarborough (CA); **Lei Su**, Beijing (CN); **Hao Zhou**, Toronto (CA)

(21) Appl. No.: **14/690,866**

(22) Filed: **Apr. 20, 2015**

Publication Classification

(51) **Int. Cl.**
G06F 9/50 (2006.01)

(52) **U.S. Cl.**
CPC ***G06F 9/50*** (2013.01)

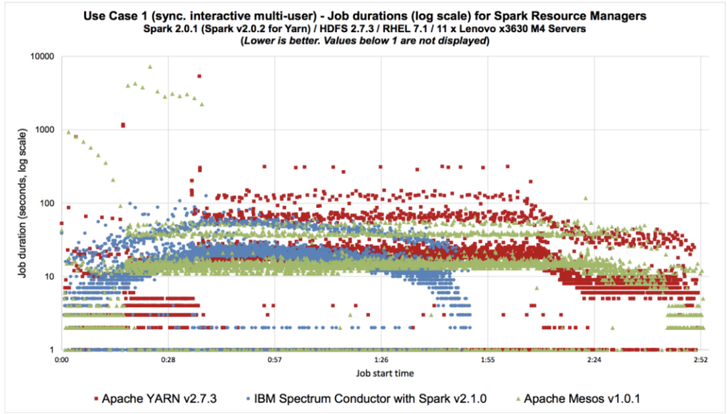
(57) **ABSTRACT**

Embodiments of the present invention provide systems and methods for allocating multiple resources. In one embodiment, a configured resource plan is used to construct a hierarchical tree. The system then identifies a set of unowned resources from the configured resource plan and sends the set of unowned resource to a share pool. The share pool is either a global or local pool and can be accessed by one or more consumers. In response to changes in workload demands, a set of unused resources are lent to a global or local pool.

100

Why Conductor : Performance – most recent STAC Benchmark

Visualization – Use Case 1



Key Points

- Conductor continues to demonstrate performance and throughput leadership driving ROI for client

UC1 Throughput

- 56% higher than YARN
- 57% higher than Mesos

UC2 Throughput

- 30% higher than YARN
- 62% higher than Mesos

UC3 Throughput

- 55% higher than YARN
- 88% higher than Mesos

UC4 –Throughput

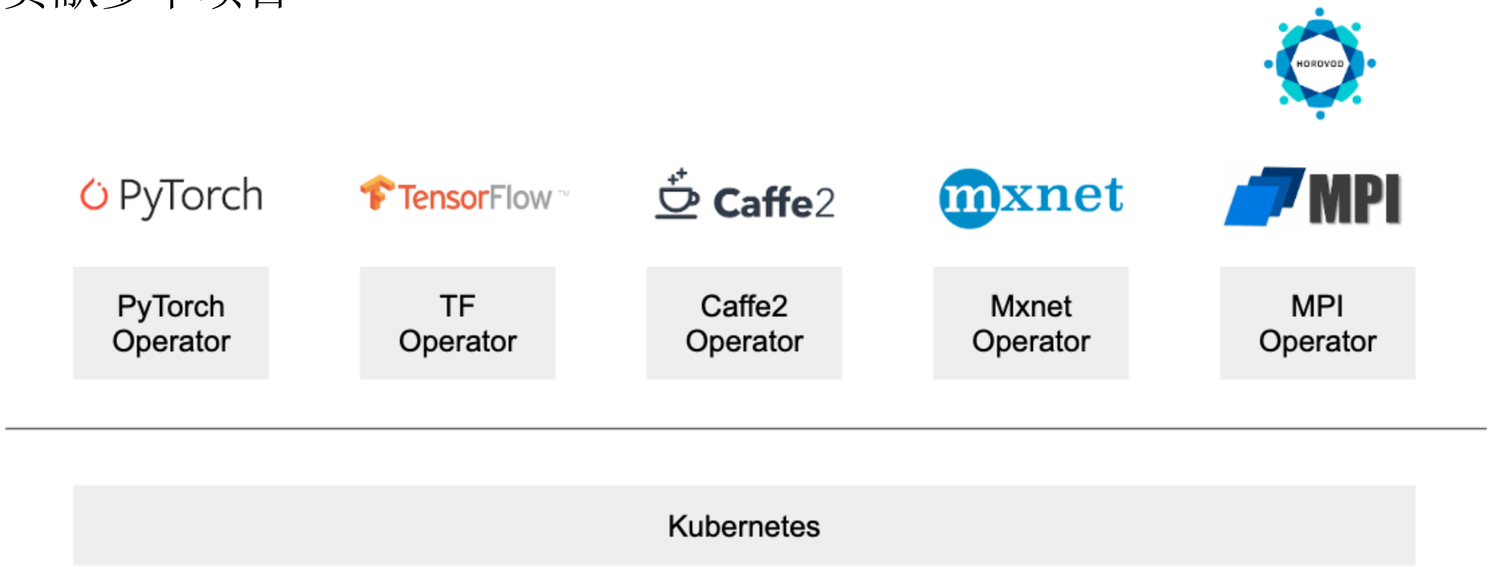
- 224% higher than YARN
- 25% higher than Mesos

- Kubeflow介绍
- MXNet介绍和分布式训练
- 使用mxnet-operator提交training任务
- TVM和AutoTVM
- 使用mxnet-operator提交tuning任务
- mxnet-operator roadmap

什么是Kubeflow

- Kubeflow旨在支持多种机器学习框架运行在 Kubernetes 之上，比如 Tensorflow, Pytorch, MXNet, Caffe等深度学习框架。
  - 包含operator、pipeline、超参数调优、serving 等诸多模块。
  - 通过operator提供作业生命周期管理，满足分布式训练需求，达到一键式部署训练任务的目的。

- 众多国内公司参与、为Kubeflow生态贡献多个项目
  - 图森：mxnet-operator
  - 才云：tf-operator
  - Momenta：caffe2-operator



什么是MXNet

- Apache MXNet是一个深度学习框架，旨在提高效率和灵活性。允许混合符号和命令式编程，以最大限度地提高效率和生产力。MXNet的核心是一个动态依赖调度程序，可以动态地自动并行化符号和命令操作。最重要的图优化层使符号执行更快，内存效率更高。

- 特点
  - 灵活的编程模型：支持命令式和符号式编程模型以最大化效率和性能。
  - 从云端到客户端可移植：可运行于多CPU、多GPU、集群、服务器、工作站甚至移动智能手机。
  - 多语言支持：支持七种主流编程语言，包括C++、Python、R、Scala、Julia、Matlab和JavaScript。
  - 分布式训练：支持在多CPU/GPU设备上的分布式训练，使其可充分利用云计算的规模优势。
  - 性能优化：使用一个优化的C++后端引擎并行I/O和计算，无论使用哪种语言都能达到最佳性能。
  - 云端友好：可直接与S3，HDFS和Azure兼容

- 官网地址
  - https://mxnet.apache.org/

MXNet用户


amazon
web services


Bai du 百度


Dato


intel


NVIDIA


香港科技大學
THE HONG KONG
UNIVERSITY OF SCIENCE
AND TECHNOLOGY


图森 tu Simple


UNIVERSITY of
WASHINGTON


WOLFRAM
COMPUTATION MEETS KNOWLEDGE

深度学习Pipeline

上传

开放数据集

私有标定数据

待标定图片

数据整合

创建训练数据集

数据预处理

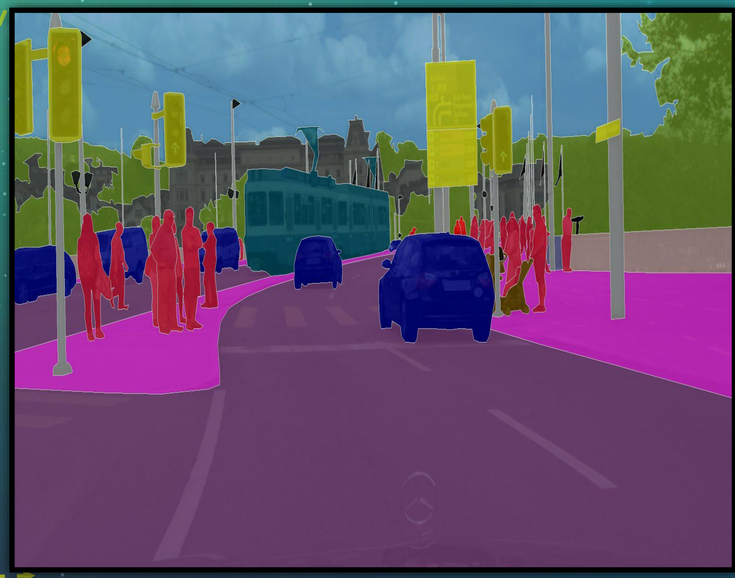
模型训练

算法自动标定

手动校准

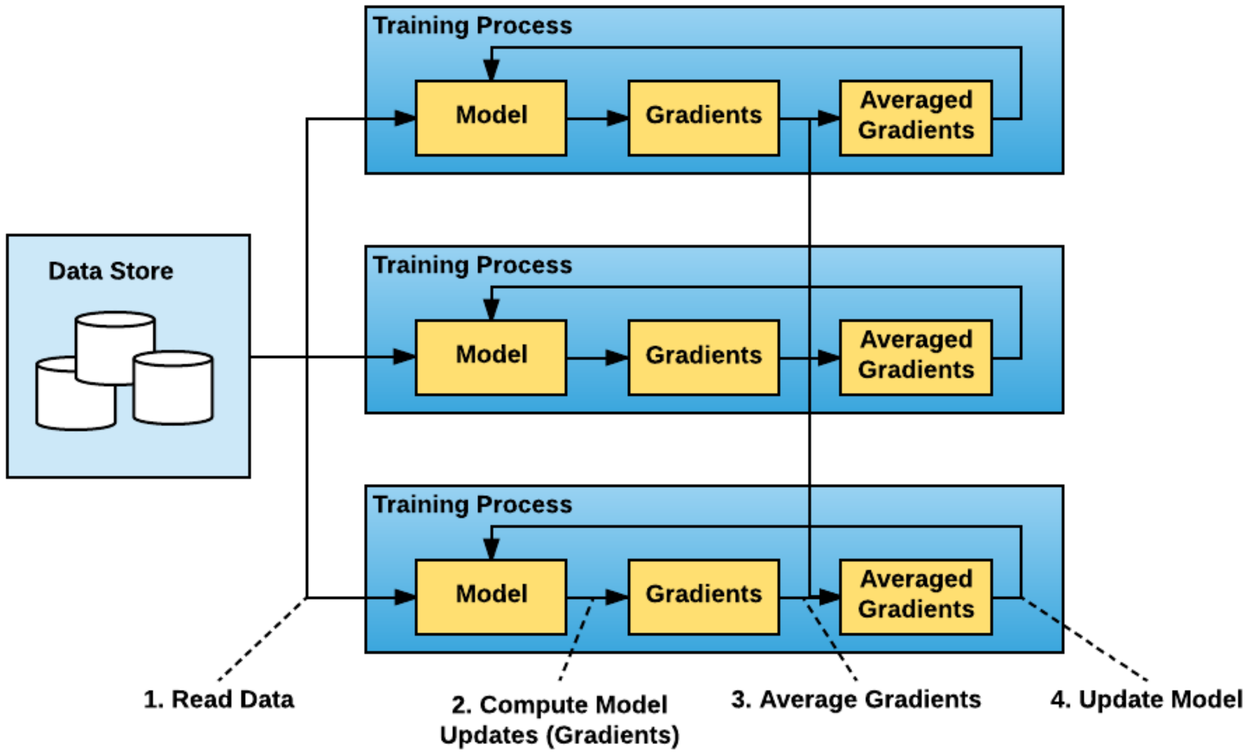
迭代

模型部署



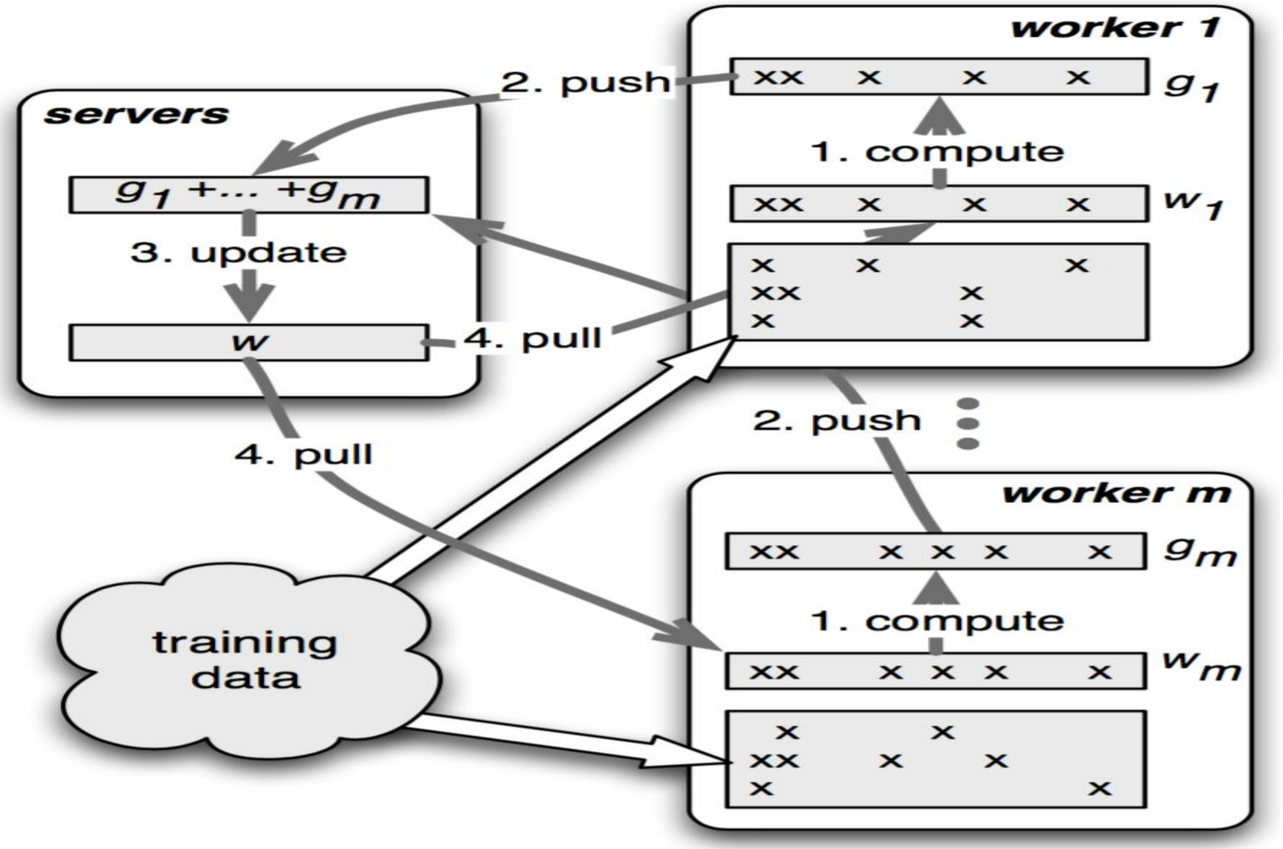
MXNet分布式训练

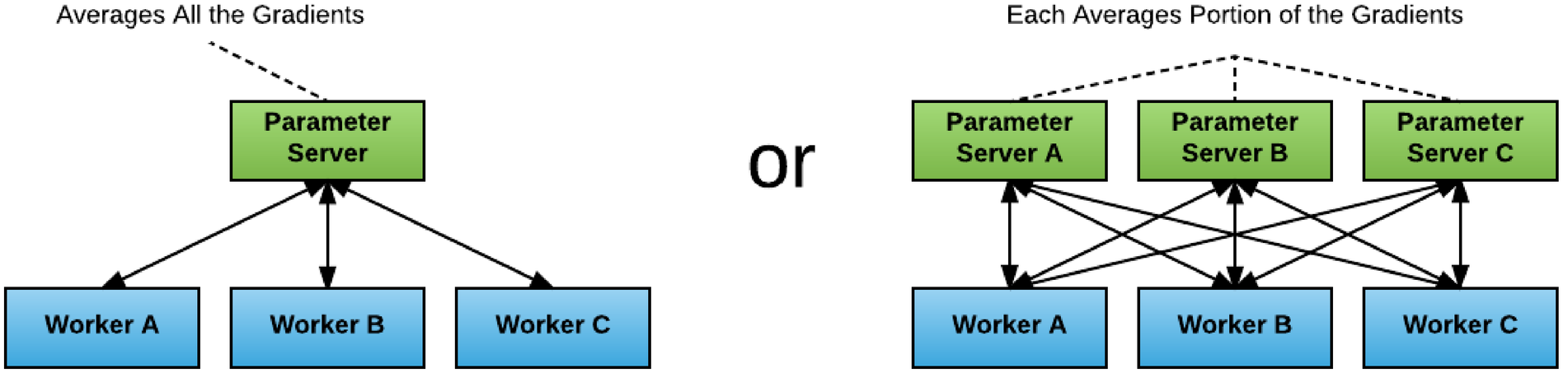
- 并行模式
  - 数据并行化：每个设备(如GPU)有一个完整的模型副本，每个设备训练一小部分数据集，共同更新共享模型，可支持单机多卡和多机多卡
  - 模型并行化：单个设备无法装下整个模型，每个设备只含有模型的一部分并且负责训练这部分模型，当前仅支持单机多卡



MXNet分布式训练

- 通信模式
  - 参数服务器





MXNet分布式训练

- 组件：
  - Server：保存模型参数，聚合梯度更新参数。一个或多个。
  - Worker：对训练样本进行实际训练。训练前，从参数服务器拉取参数；训练后，发送梯度到参数服务器。一个或多个。
  - Scheduler：负责建立通信并分配和监控任务。有且仅有一个。

- 启动方式：

```
export COMMAND='python example/gluon/image_classification.py --dataset cifar10 --model vgg11
--epochs 1 --kvstore dist_sync'
DMLC_ROLE=server DMLC_PS_ROOT_URI=127.0.0.1 DMLC_PS_ROOT_PORT=9092 DMLC_NUM_SERVER=2
DMLC_NUM_WORKER=2 $COMMAND &
DMLC_ROLE=server DMLC_PS_ROOT_URI=127.0.0.1 DMLC_PS_ROOT_PORT=9092 DMLC_NUM_SERVER=2
DMLC_NUM_WORKER=2 $COMMAND &
DMLC_ROLE=scheduler DMLC_PS_ROOT_URI=127.0.0.1 DMLC_PS_ROOT_PORT=9092 DMLC_NUM_SERVER=2
DMLC_NUM_WORKER=2 $COMMAND &
DMLC_ROLE=worker DMLC_PS_ROOT_URI=127.0.0.1 DMLC_PS_ROOT_PORT=9092 DMLC_NUM_SERVER=2
DMLC_NUM_WORKER=2 $COMMAND &
DMLC_ROLE=worker DMLC_PS_ROOT_URI=127.0.0.1 DMLC_PS_ROOT_PORT=9092 DMLC_NUM_SERVER=2
DMLC_NUM_WORKER=2 $COMMAND
```

使用mxnet-operator提交training任务

```
apiVersion:
"kubeflow.org/v1alpha2"
kind: "MXJob"
metadata:
  name: "mxnet-job"
spec:
  jobMode: MXTrain
  mxReplicaSpecs:
    Scheduler:
      replicas: 1
      restartPolicy: Never
      template:
        spec:
          containers:
            - name: mxnet
              image:
mxjob/mxnet:gpu
```

```
Worker:
  replicas: 1
  restartPolicy: Never
  template:
    spec:
      containers:
        - name: mxnet
          image: mxjob/mxnet:gpu
          command: ["python"]
          args: ["/incubator-
mxnet/example/image-
classification/train_mnist.py","--
num-epochs","1","--num-
layers","2","--kv-
store","dist_device_sync","--
gpus","0"]
          resources:
            limits:
              nvidia.com/gpu: 1
```

```
Server:
  replicas: 1
  restartPolicy: Never
  template:
    spec:
      containers:
        - name: mxnet
          image:
mxjob/mxnet:gpu
```

使用mxnet-operator提交training任务

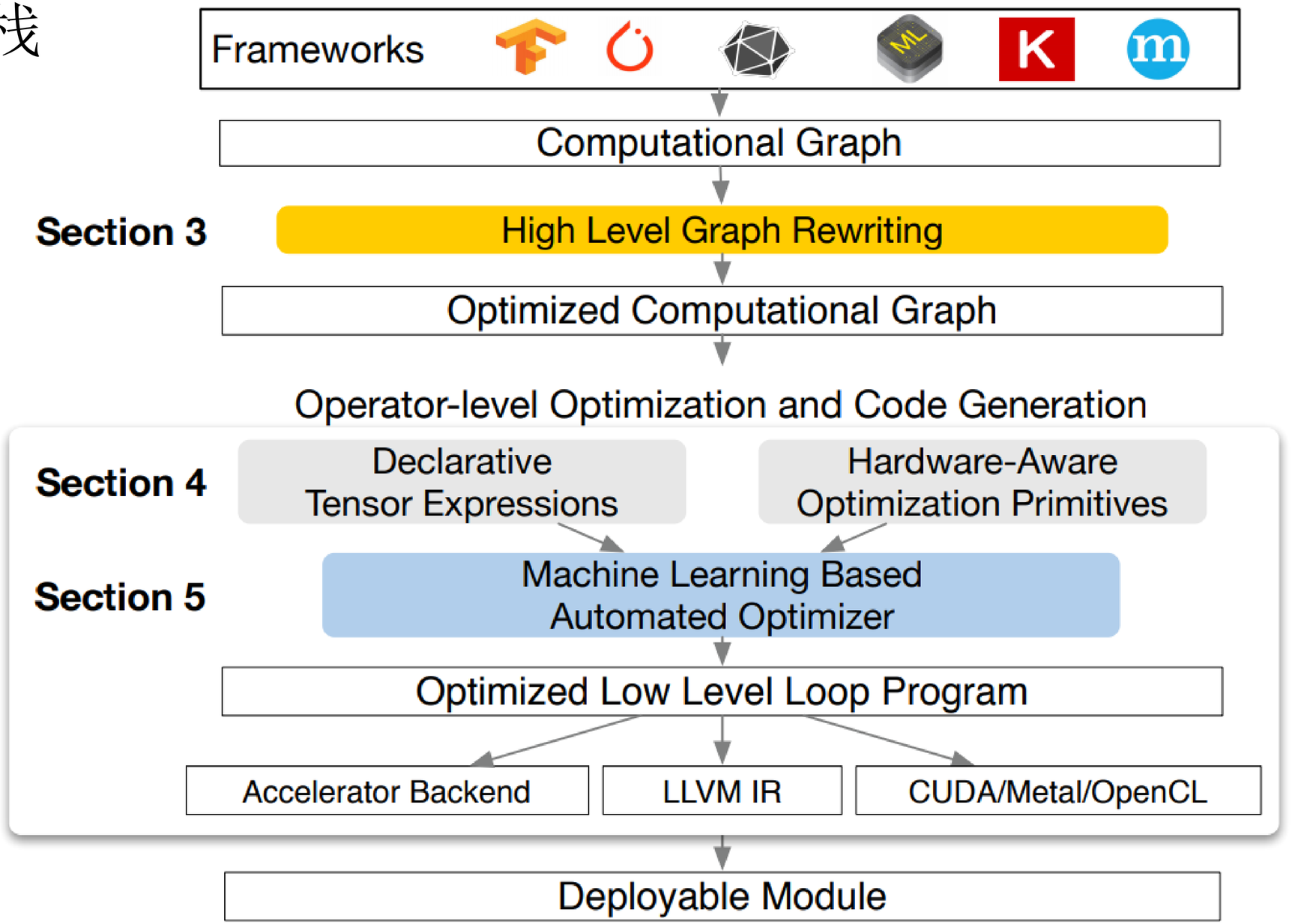
```
mxnet-job-scheduler-0   1/1       Running   0          23s
mxnet-job-server-0      1/1       Running   0          23s
mxnet-job-worker-0      1/1       Running   0          23s
leisu@desk-73:/extend/Workspace/myproj/src/github.com/kubeflow/mxnet-operator/examples/mxnet-operator.v2/train$ kubectl logs mxne
-0
INFO:root:start with arguments Namespace(add_stn=False, batch_size=64, disp_batches=10, dtype='float32', gc_threshold=0.5, gc_typ
us='0', initializer='default', kv_store='dist_device_sync', load_epoch=None, loss='', lr=0.05, lr_factor=0.1, lr_step_epochs='10'
_size=0, model_prefix=None, mom=0.9, monitor=0, network='mlp', num_classes=10, num_epochs=10, num_examples=60000, num_layers=2, c
d', test_io=0, top_k=0, warmup_epochs=5, warmup_strategy='linear', wd=0.0001)
DEBUG:urllib3.connectionpool:Starting new HTTP connection (1): yann.lecun.com
DEBUG:urllib3.connectionpool:http://yann.lecun.com:80 "GET /exdb/mnist/train-labels-idx1-ubyte.gz HTTP/1.1" 200 28881
DEBUG:urllib3.connectionpool:Starting new HTTP connection (1): yann.lecun.com
DEBUG:urllib3.connectionpool:http://yann.lecun.com:80 "GET /exdb/mnist/train-images-idx3-ubyte.gz HTTP/1.1" 200 9912422
DEBUG:urllib3.connectionpool:Starting new HTTP connection (1): yann.lecun.com
DEBUG:urllib3.connectionpool:http://yann.lecun.com:80 "GET /exdb/mnist/t10k-labels-idx1-ubyte.gz HTTP/1.1" 200 4542
DEBUG:urllib3.connectionpool:Starting new HTTP connection (1): yann.lecun.com
DEBUG:urllib3.connectionpool:http://yann.lecun.com:80 "GET /exdb/mnist/t10k-images-idx3-ubyte.gz HTTP/1.1" 200 1648877
INFO:root:Epoch[0] Batch [10]   Speed: 31347.93 samples/sec     accuracy=0.332386
INFO:root:Epoch[0] Batch [20]   Speed: 41367.77 samples/sec     accuracy=0.709375
INFO:root:Epoch[0] Batch [30]   Speed: 42556.11 samples/sec     accuracy=0.775000
INFO:root:Epoch[0] Batch [40]   Speed: 43240.25 samples/sec     accuracy=0.831250
INFO:root:Epoch[0] Batch [50]   Speed: 37664.58 samples/sec     accuracy=0.873437
INFO:root:Epoch[0] Batch [60]   Speed: 39177.36 samples/sec     accuracy=0.853125
INFO:root:Epoch[0] Batch [70]   Speed: 36217.31 samples/sec     accuracy=0.860938
INFO:root:Epoch[0] Batch [80]   Speed: 36088.77 samples/sec     accuracy=0.865625
INFO:root:Epoch[0] Batch [90]   Speed: 35676.75 samples/sec     accuracy=0.884375
INFO:root:Epoch[0] Batch [100]  Speed: 39835.49 samples/sec     accuracy=0.909375
INFO:root:Epoch[0] Batch [110]  Speed: 38521.27 samples/sec     accuracy=0.892188
INFO:root:Epoch[0] Batch [120]  Speed: 36256.44 samples/sec     accuracy=0.893750
INFO:root:Epoch[0] Batch [130]  Speed: 36556.65 samples/sec     accuracy=0.906250
INFO:root:Epoch[0] Batch [140]  Speed: 34758.38 samples/sec     accuracy=0.921875
INFO:root:Epoch[0] Batch [150]  Speed: 39165.36 samples/sec     accuracy=0.917188
INFO:root:Epoch[0] Batch [160]  Speed: 38720.19 samples/sec     accuracy=0.907813
INFO:root:Epoch[0] Batch [170]  Speed: 36834.55 samples/sec     accuracy=0.900000
INFO:root:Epoch[0] Batch [180]  Speed: 37426.69 samples/sec     accuracy=0.917188
INFO:root:Epoch[0] Batch [190]  Speed: 36872.50 samples/sec     accuracy=0.925000
INFO:root:Epoch[0] Batch [200]  Speed: 40973.13 samples/sec     accuracy=0.914062
INFO:root:Epoch[0] Batch [210]  Speed: 42677.90 samples/sec     accuracy=0.907813
INFO:root:Epoch[0] Batch [220]  Speed: 38584.39 samples/sec     accuracy=0.917188
```

什么是TVM

- 一个端到端的深度学习编译软件栈
  - 为CPU, GPU和其他硬件加速器优化深度学习计算负载
  - 自动转换计算图以最小化内存使用，优化数据布局，算子融合
  - 提供从现有的前端框架到裸机硬件的端到端编译

- 官网地址：
  - https://tvm.ai/

TVM软件栈


Frameworks
Computational Graph
Section 3
High Level Graph Rewriting
Optimized Computational Graph
Operator-level Optimization and Code Generation
Section 4
Declarative
Tensor Expressions
Hardware-Aware
Optimization Primitives
Section 5
Machine Learning Based
Automated Optimizer
Optimized Low Level Loop Program
Accelerator Backend
LLVM IR
CUDA/Metal/OpenCL
Deployable Module

TVM使用

Typical Workflow of NNVM Compiler

model from framework

```
graph, params = nnvm.frontend.from_xyz(...)
```

compile

```
graph, lib, params = nnvm.compiler.build(
      graph, target="cuda", {"data", data_shape}, params=params)
```

deploy

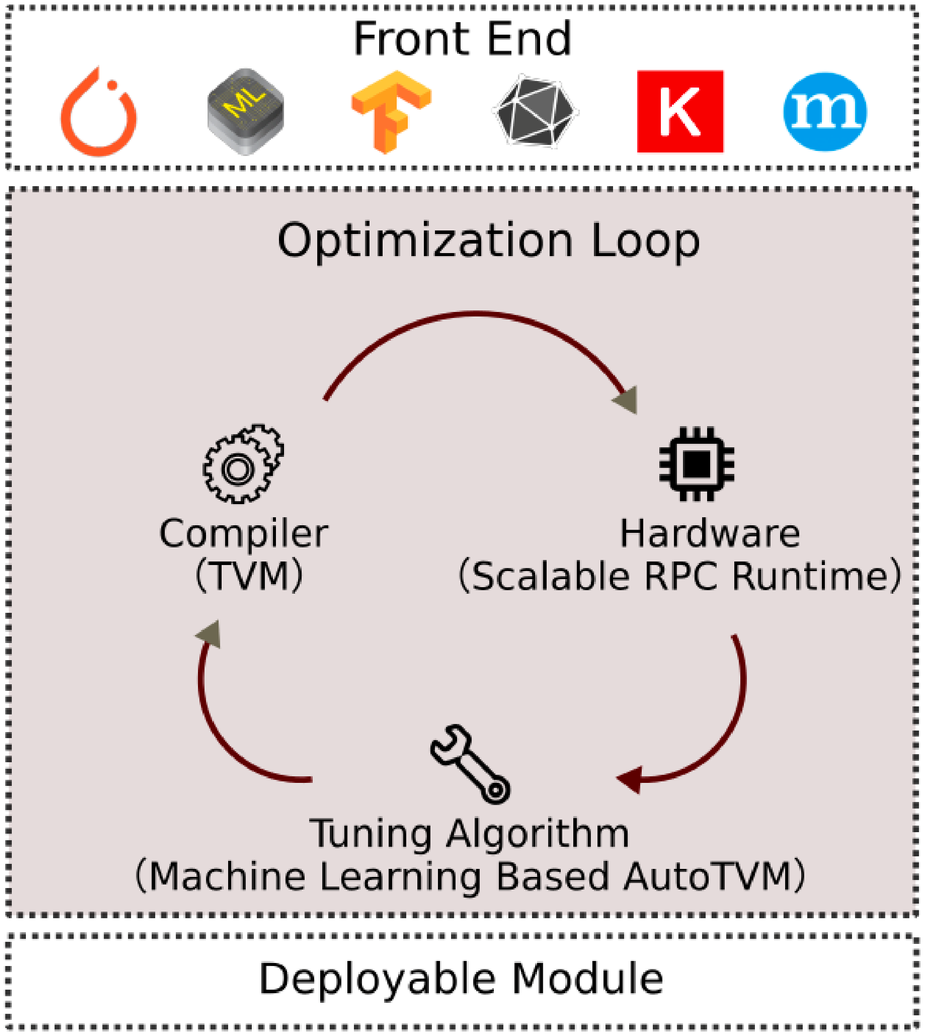
```
module = graph_runtime.create(graph, lib, tvm.gpu(0))
module.set_input(**params)
module.run(data=data_array)
output = tvm.nd.empty(out_shape, ctx=tvm.gpu(0))
module.get_output(0, output)
```

什么是AutoTVM

- 痛点：
  - 高效的operator受多种因素影响：硬件设备的类型、输入大小、tensor layout等，手工优化费时费力。
  - 参数搜索空间巨大

- 方案：
  - 采用机器学习方法自动寻找较优的参数配置

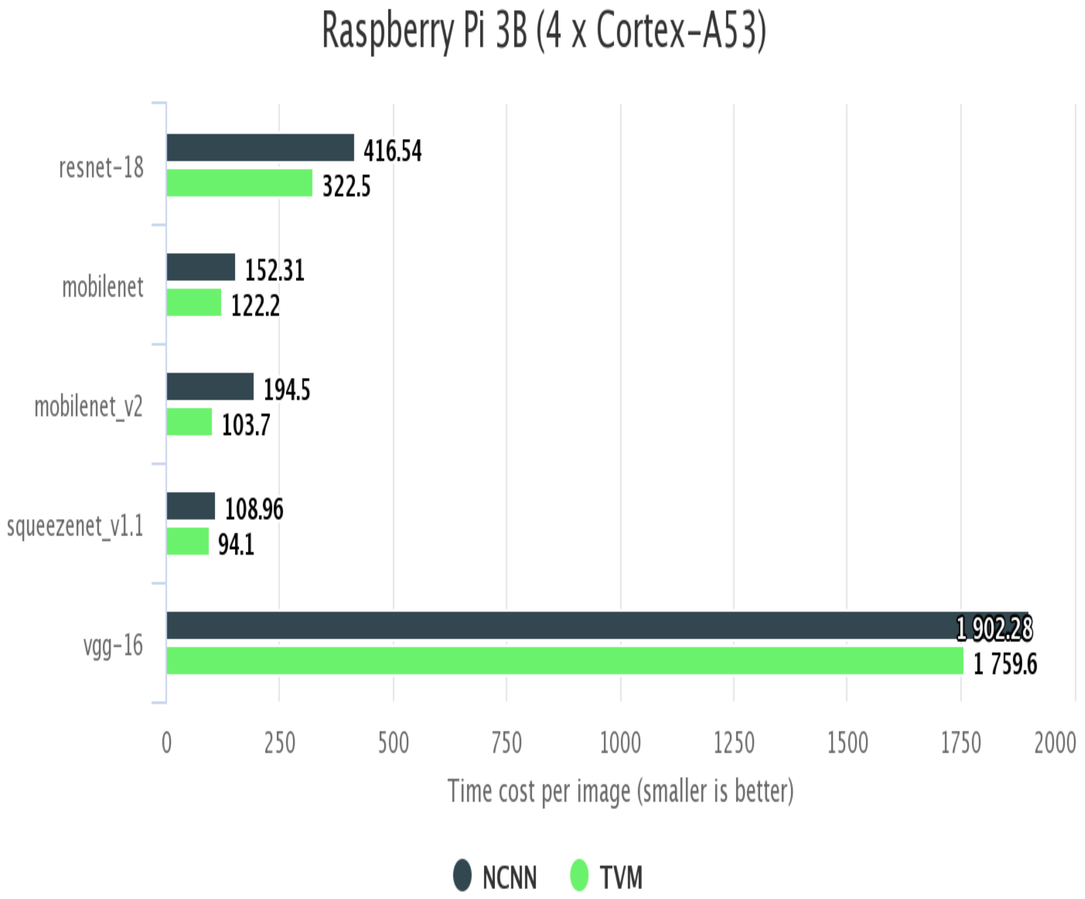
AutoTVM机制

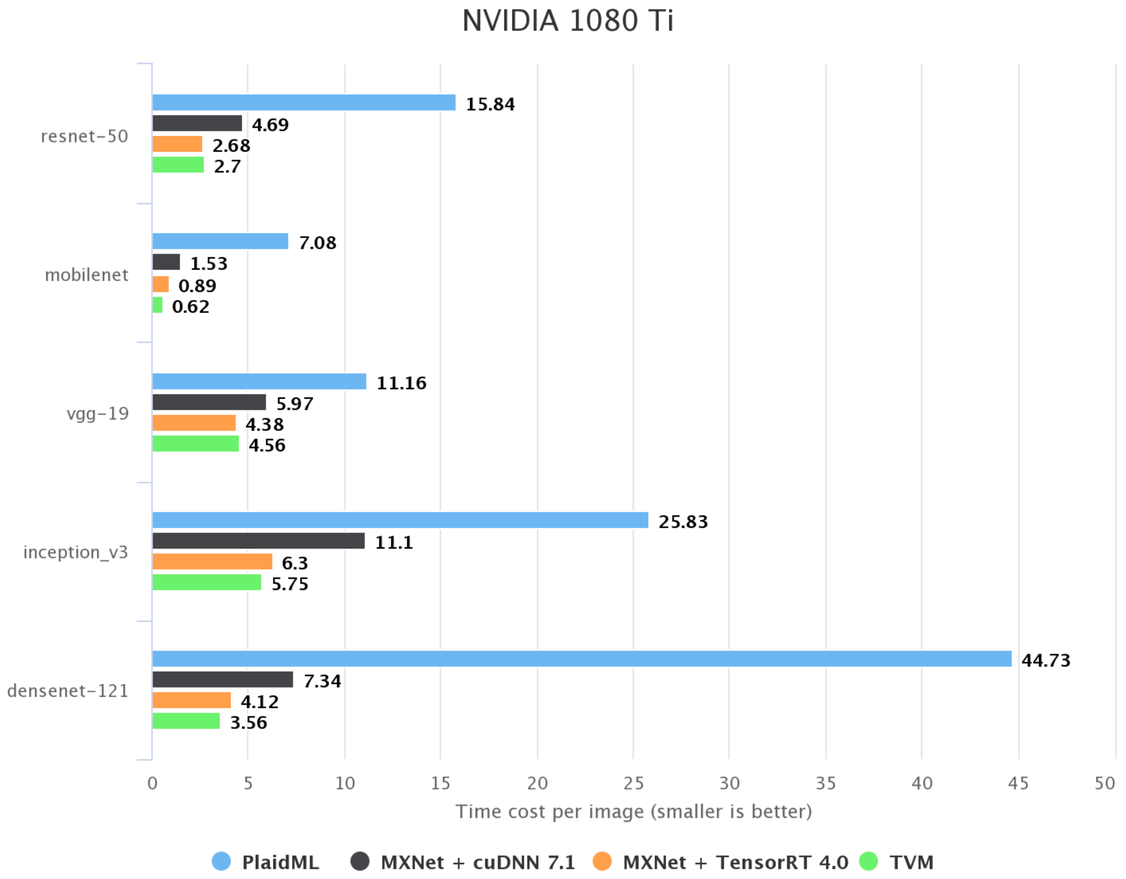


- 基本流程
  - 从前端框架获取和生成所有操作符的kernel
  - Tuner从可调参数搜索空间里寻找并编译出一批有提速潜质的kernel，通过RPC调用在硬件上执行这些kernel
  - Tuner获得相应kernel的性能数据并作为训练数据训练新的预测模型
  - 获得新的预测模型后，Tuner根据新的预测结果寻找下一批有提速潜质的kernel实现，并循环上述过程

- 组件：
  - Tunertracker：将任务调度至Tunerserver
  - Tunerserver：在实际硬件上执行kernel并返回性能数据
  - Tuner：实际的tuning程序，基于机器学习算法寻找较优配置并生成kernel

AutoTVM效果


Raspberry Pi 3B (4 x Cortex-A53)
resnet-18
416.54
322.5
mobilenet
152.31
122.2
mobilenet_v2
194.5
103.7
squeezenet_v1.1
108.96
94.1
vgg-16
1 902.28
1 759.6
0
250
500
750
1000
1250
1500
1750
2000
Time cost per image (smaller is better)
NCNN
TVM


NVIDIA 1080 Ti
resnet-50
15.84
4.69
2.68
2.7
mobilenet
7.08
1.53
0.89
0.62
vgg-19
11.16
5.97
4.38
4.56
inception_v3
25.83
11.1
6.3
5.75
densenet-121
44.73
7.34
4.12
3.56
0
5
10
15
20
25
30
35
40
45
50
Time cost per image (smaller is better)
PlaidML
MXNet + cuDNN 7.1
MXNet + TensorRT 4.0
TVM

使用mxnet-operator提交auto-tuning任务

```
apiVersion:
"kubeflow.org/v1alpha2"
kind: "MXJob"
metadata:
  name: "auto-tuning-job"
spec:
  jobMode: MXTune
  mxReplicaSpecs:
    TunerTracker:
      replicas: 1
      restartPolicy: Never
      template:
        spec:
          containers:
          - name: mxnet
            image: mxjob/auto-
tuning:gpu
            command: ["python3"]
            args:
["/home/scripts/start-job.py"]
```

```
    TunerServer:
      label: 2080ti
      replicas: 1
      restartPolicy: Never
      template:
        spec:
          containers:
          - name: mxnet
            image:
mxjob/auto-tuning:gpu
            command:
["python3"]
            args:
["/home/scripts/start-
job.py"]
            resources:
              limits:

nvidia.com/gpu: 1
```

```
Tuner:
      replicas: 1
      restartPolicy: Never
      template:
        spec:
          containers:
          - name: mxnet
            image:
mxjob/auto-tuning:gpu
            command:
["python3"]
            args:
["/home/scripts/start-
job.py"]
```

使用mxnet-operator提交auto-tuning任务

```
if __name__ == '__main__':
    mx_config = json.loads(os.environ.get('MX_CONFIG') or '{}')
    cluster_config = mx_config.get('cluster', {})
    labels_config = mx_config.get('labels', {})
    task_config = mx_config.get('task', {})
    task_type = task_config.get('type')
    task_index = task_config.get('index')

    if task_type == "":
        print("No task_type, Error")
    elif task_type == "tunertracker":
        addr = cluster_config["tunertracker"][0]
        command = "python3 -m tvm.exec.rpc_tracker --port={0}".format(addr.get('port'))
        print("DO: " + command)
        os.system(command)
    elif task_type == "tunerserver":
        time.sleep(5)
        addr = cluster_config["tunertracker"][0]
        label = labels_config["tunerserver"]
        command = "python3 -m tvm.exec.rpc_server --tracker={0}:{1} --key={2}".format(addr.get('url'), addr.get('port'), label)
        print("DO: " + command)
        os.system(command)
    elif task_type == "tuner":
        time.sleep(5)
        addr = cluster_config["tunertracker"][0]
        label = labels_config["tunerserver"]
        command = "python3 /home/scripts/auto-tuning.py --tracker {0} --tracker_port {1} --server_key {2}".format(addr.get('url'),
addr.get('port'), label)
        print("DO: " + command)
        os.system(command)
    else:
        print("Unknow task type! Error")
```

使用mxnet-operator提交auto-tuning任务

```
leisu@desk-73:/extend/Workspace/myproj/src/github.com/kubeflow/mxnet-operator/examples/mxnet-operator.v2/tune$ kubectl get pods
NAME                           READY     STATUS    RESTARTS   AGE
auto-tuning-job-tuner-0        1/1       Running   0          9m
auto-tuning-job-tunerserver-0  1/1       Running   0          9m
auto-tuning-job-tunertracker-0 1/1       Running   0          9m
leisu@desk-73:/extend/Workspace/myproj/src/github.com/kubeflow/mxnet-operator/examples/mxnet-operator.v2/tune$ kubectl logs auto-tuning-job-tuner-0
Extract tasks...
...100%, 0.02 MB, 23 KB/s, 1 seconds passed
Tuning...
[Task  1/12]  Current/Best: 2343.76/2632.19 GFLOPS | Progress: (100/100) | 273.24 s Done.
[Task  2/12]  Current/Best:  285.03/ 392.79 GFLOPS | Progress: (100/100) | 123.07 s Done.
```

Mxnet-operator Roadmap

- 提高AutoTVM运行效率
- MXJob支持MXBoard
- 集成Kube-batch

致谢

姜宗沛
清华大学

杨瑞恒
上海交通大学

Q & A