Container + DeepLearning: from working to scaling

Frank Zhao

Software Architect, EMC CTO Office

zhaojp@gmail.com

WHO AM I?

- Frank Zhao, 赵军平, Junping Zhao
- Software Architect @ EMC CTO Office

- 10+year engineering in storage, virtualization, flash
  - 30+ patents (3 granted in U.S.)
- Now, working areas:
  - In-mem processing and analytics, perf acceleration
  - Micro-service, container
  - Streaming processing system
  - Software defined storage …

AGENDA

- Motivations
- Deep learning and TensorFlow
- Distributed DL training
  - Docker + K8S + Tensorflow as example
- Distributed DL serving
  - Docker + K8S + Tensorflow as example
- Summary
- Outlook

Motivations

- Container + DL for
  - Easier deployment & management
  - Scalability: from infrastructure to typical DL framework
  - Smooth transition from training to serving, or Dev&Ops
  - Least performance overhead (vs. VM) especially CPU/mem
    - Maximize heterogeneous env, computing, memory, networking etc

- → **“DL as a Service” by container ecosystem?**
  - In cloud env, scale-out & heterogeneous

- Focus on DL workload, practices sharing, opportunities
  - Non-goal: TF internal or deep dive

Deep learning

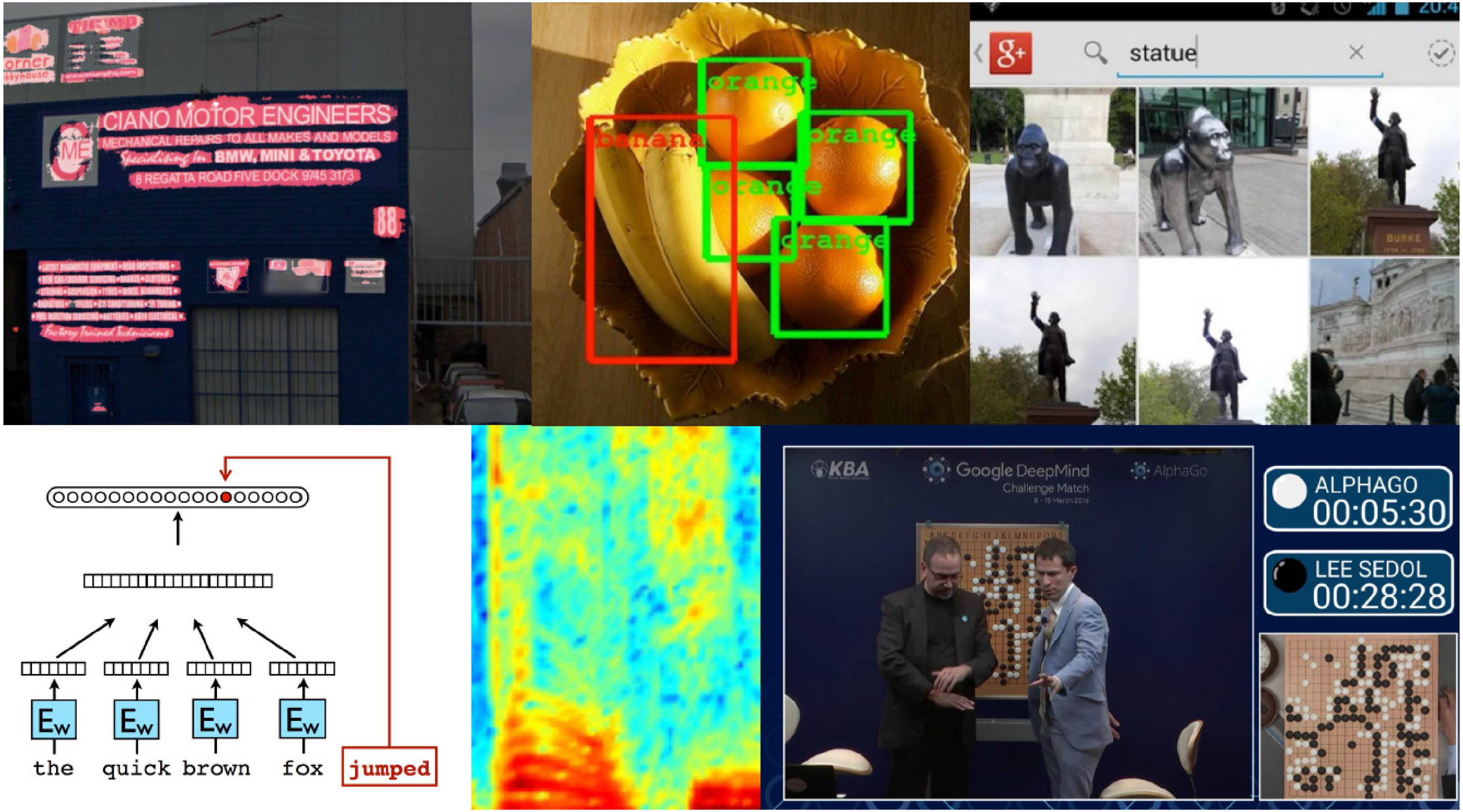


Photo from Google intro: "Diving into ML through Tensorflow"

Open-source DL frameworks

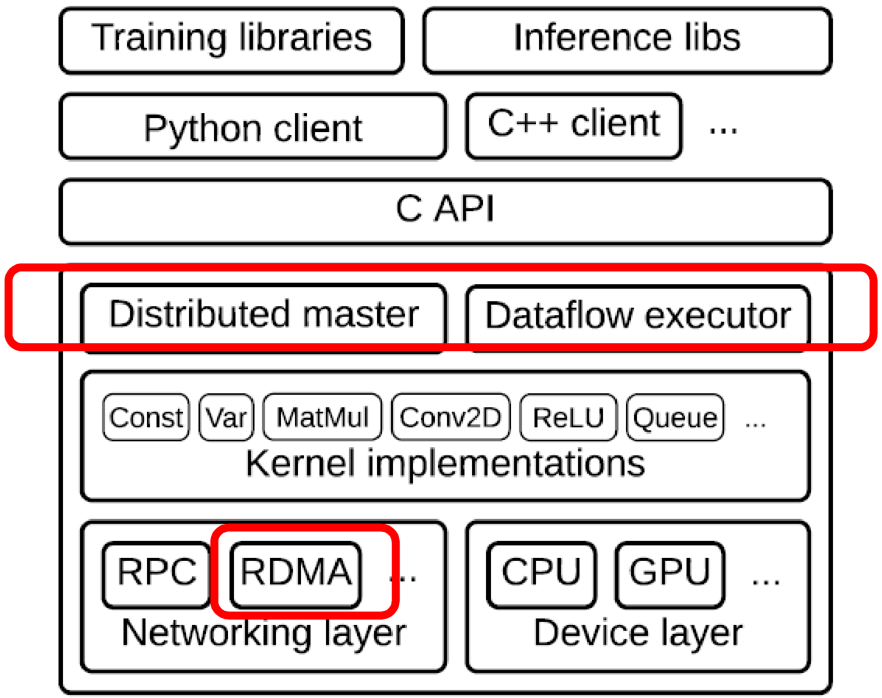
By 6/28/2016

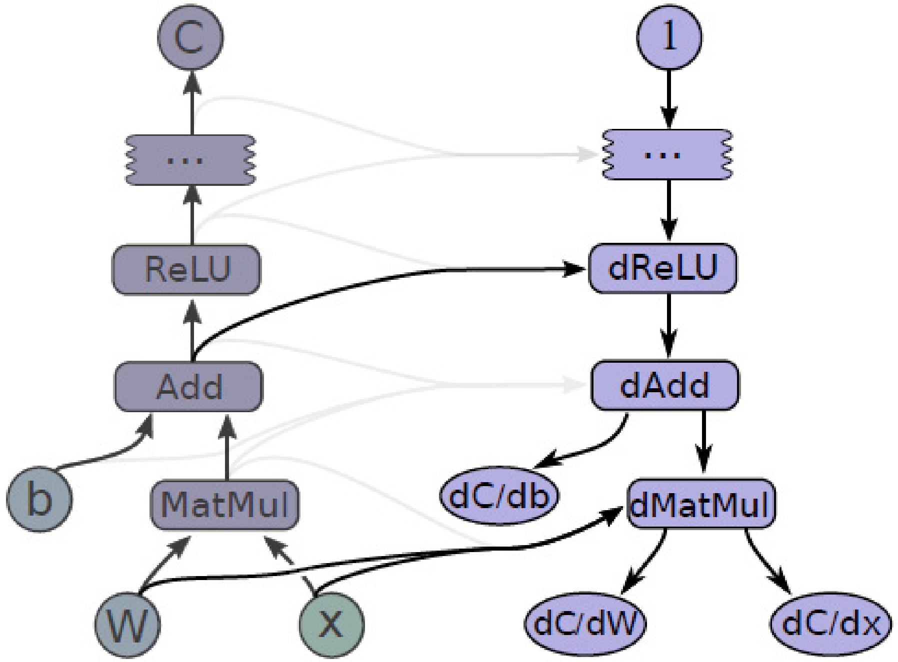
	Star	Forks	Contributor
Tensorflow	**26995**	**10723**	**286**
Caffe	10973	6575	196
CNTK	5699	1173	69
Torch	4852	1360	100
Theano	4022	1448	234
MXNet	4173	1515	152
Apache SINGA	607	211	18

A more detailed summary and comparison @ https://github.com/zer0n/deepframeworks

TensorFlow

- DL framework from Google
  - GPU/CPU/TPU, heterogeneous: desktop, server, mobile etc
  - C++, Python; Distributed training and serving
  - DNN building blocks, ckpt/queue/ …, TensorBoard
  - Good docker/K8S supported

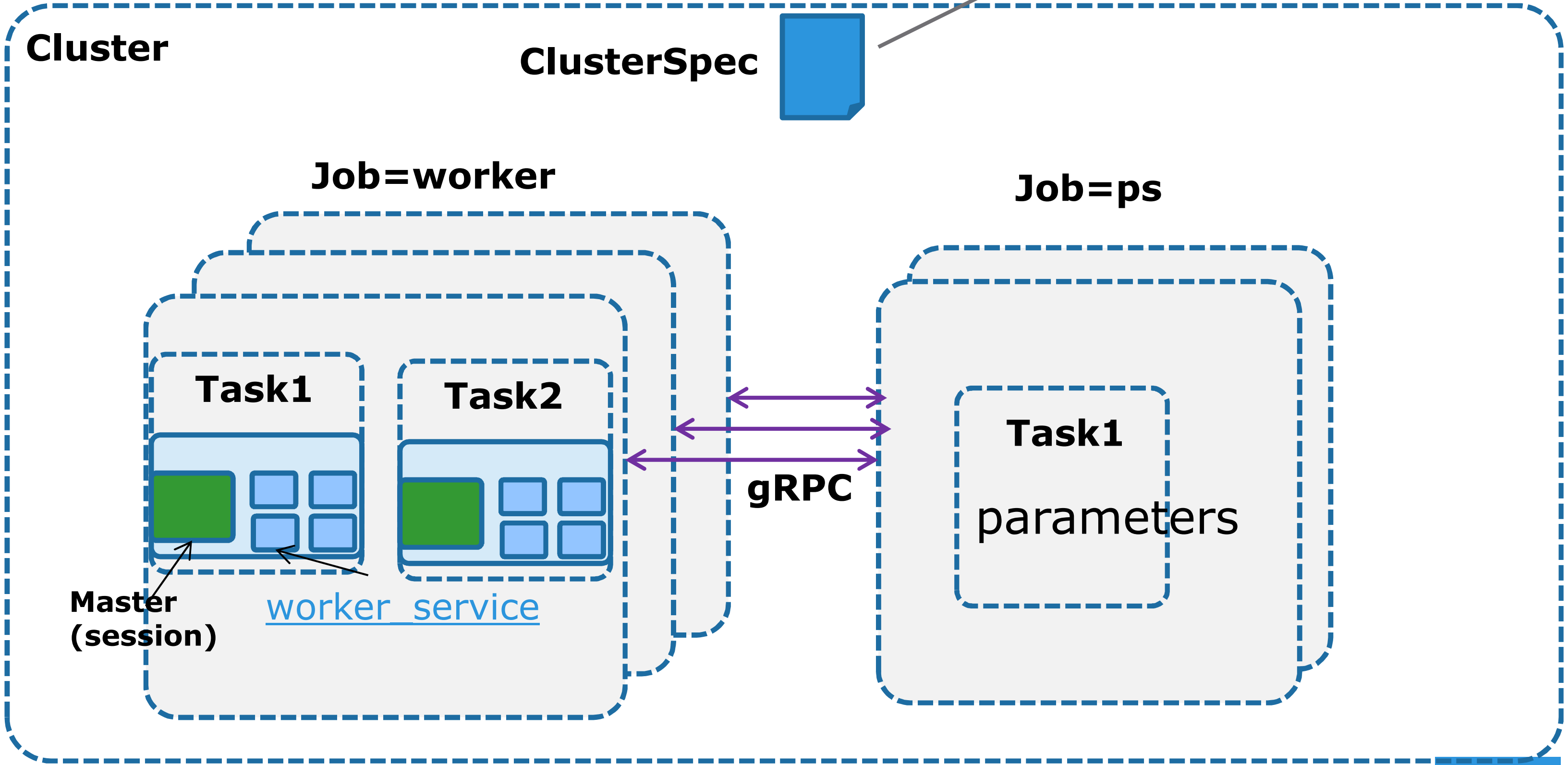




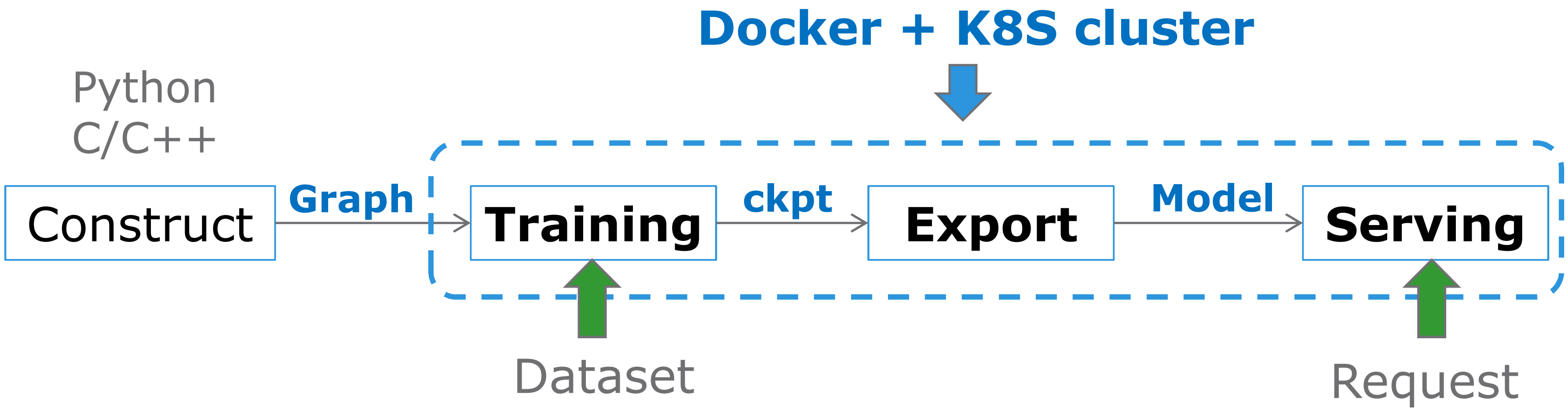
TF distribution overview

- **Cluster**
  - **Jobs (ps, worker)**
    - **Tasks**
      - **Devices** (CPU, GPU)

```
tf.train.ClusterSpec({
   "worker": [
      "worker0.example.com:2222",
      "worker1.example.com:2222",
      "worker2.example.com:2222"
   ],
   "ps": [
      "ps0.example.com:2222",
      "ps1.example.com:2222"
   ]})
```



Workflow & dataset



Dataset	Type	Size
MNIST	Handwritten digit recognition	10 class. 60K training + 10K test
CIFAR-10/100	Image classification	10/100 classes; (50K + 10K test) 160+MB
Inception-V3	Image classification	1000 classes; ~500GB dataset

Test environment

- 3-node bare metal (Dell?) servers
  - CPU only; GPU+CPU cluster is being setup
  - Docker images from Tensorflow official w/ modifications

TF: 0.8

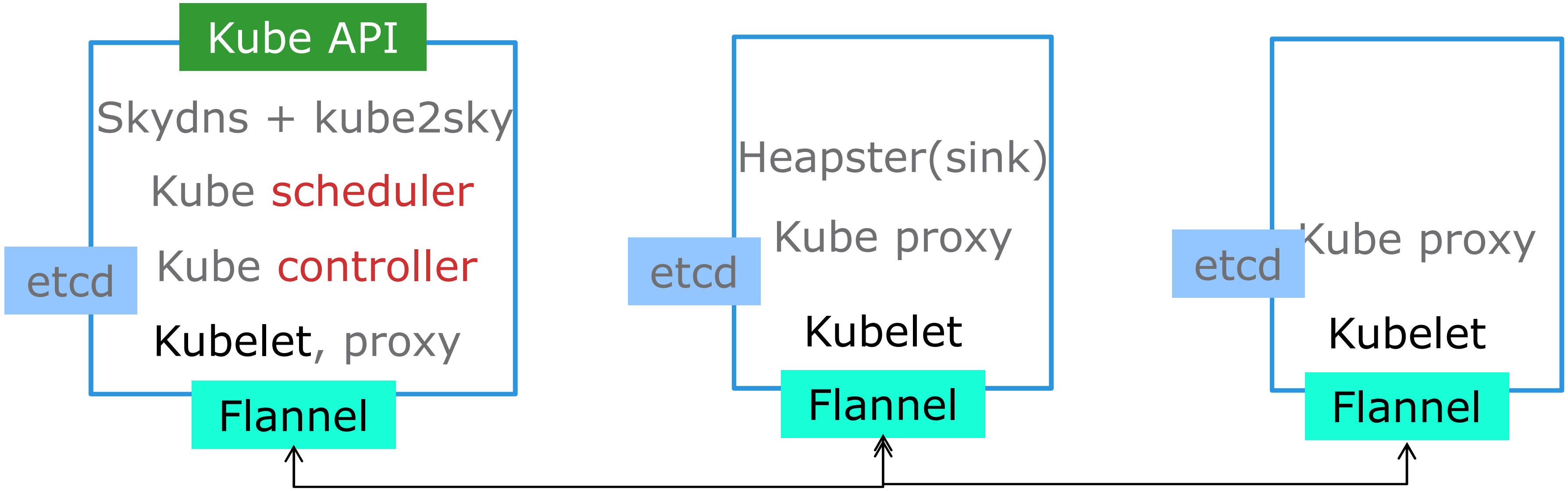
K8S: 1.2.4

Docker: 1.10.3; AUFS

Ubuntu 15.10 (4.2.0-16)

E5-2643 v3 @ 3.40GHz,
2 *12 cores; 256GB DDR3
1000 Mbps

K8S cluster




```
root@sm-tower1:~# docker ps
CONTAINER ID   IMAGE                                                    COMMAND                      CREATED
6605cec68471   gcr.io/google_containers/heapster_grafana:v2.6.0-2      "/bin/sh -c /run.sh"         3 days ago
05102147185b   gcr.io/google_containers/heapster_influxdb:v0.5         "influxd --config /et"       3 days ago
e42bf1d27ce5   gcr.io/google_containers/pause:2.0                      "/pause"                     3 days ago
90b53bc1341e   gcr.io/google_containers/skydns:1.0                     "/skydns"                    3 days ago
527d600b9259   gcr.io/google_containers/kube2sky:1.15                  "/kube2sky"                  3 days ago
de3633ab3fb0   quay.io/coreos/flannel:0.5.5                            "/bin/sh -c '/opt/bin"       4 days ago
ff9c894c47e0   gcr.io/google_containers/flannel-server-helper:0.1      "/flannel_helper --ne"       4 days ago
8b20a0d4a34b   quay.io/smana/kubernetes-hyperkube:v1.2.4               "/hyperkube proxy --v"       4 days ago
373f4ad80b61   quay.io/smana/kubernetes-hyperkube:v1.2.4               "/hyperkube scheduler"       4 days ago
c661aa61186b   quay.io/smana/kubernetes-hyperkube:v1.2.4               "/hyperkube controlle"       4 days ago
4a4b30c7ccfb   gcr.io/google_containers/pause:2.0                      "/pause"                     4 days ago
04ec408af28e   gcr.io/google_containers/pause:2.0                      "/pause"                     4 days ago
bc41b8f1f32b   gcr.io/google_containers/pause:2.0                      "/pause"                     4 days ago
93e8cf10610c   gcr.io/google_containers/pause:2.0                      "/pause"                     4 days ago
```

Cluster setup

- Still not very easy!
  - TF official setup script doesn't work in my Ubuntu15

- Tips
  - DNS (skydns) service is highly recommended
  - Use Docker 1.10 if possible
    - Docker1.11 (RunC) may not work well
  - Kargo likely helps setup easier
    - https://docs.kubespray.io/ including K8S, etcd, flanneld, etc

- Close look at DL workload
  - Workload profiling
  - Scaling w/ CPU/GPU

- Distribution efficiency?
  - Communication overhead
  - Synchronization the parameter overhead
- Test plan: containers in **1**-node vs. in **3**-node
  - To isolate networking overhead
  - Async vs. sync

DL workload profiling

CIFAR10 dataset, CNN, Intel E5 v3 CPU*24 core

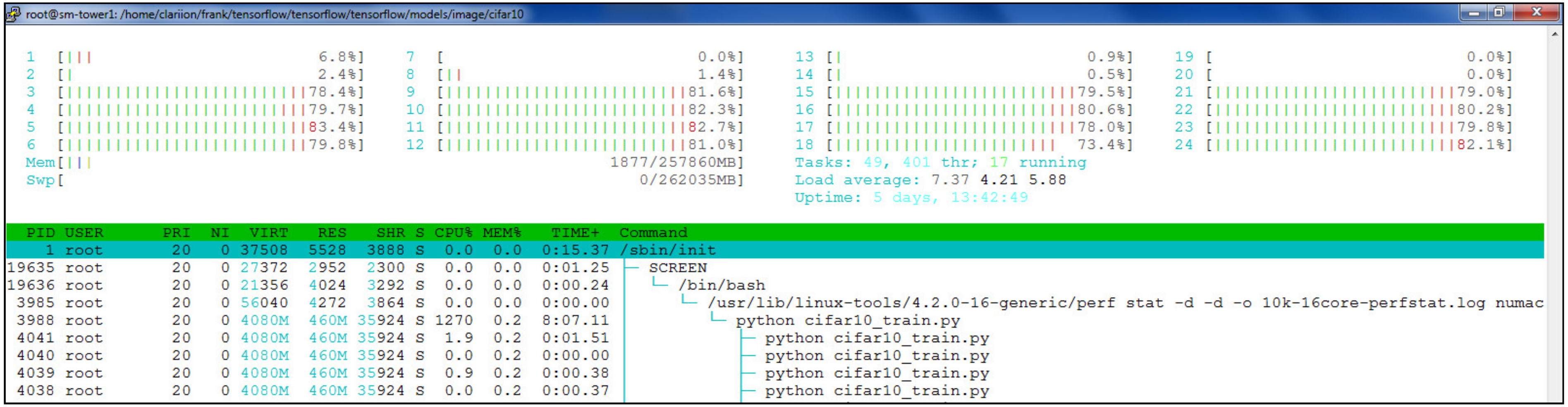
```
Performance counter stats for 'python cifar10_train.py':

   5263014.248826      task-clock (msec)         #   17.133 CPUs utilized
       22,264,167      context-switches          #    0.004 M/sec
        1,769,718      cpu-migrations            #    0.336 K/sec
      126,755,898      page-faults               #    0.024 M/sec
18,645,619,419,538     cycles                    #    3.543 GHz                      (28.56%)
  <not supported>      stalled-cycles-frontend
  <not supported>      stalled-cycles-backend
21,654,529,625,343     instructions              #    1.16  insns per cycle          (36.24%)
 1,170,696,486,325     branches                  #  222.438 M/sec                    (35.92%)
    11,265,428,007     branch-misses             #    0.96% of all branches          (35.80%)
 4,000,366,119,368     L1-dcache-loads           #  760.090 M/sec                    (21.86%)
   327,032,499,068     L1-dcache-load-misses     #    8.18% of all L1-dcache hits    (16.77%)
    53,063,949,994     LLC-loads                 #   10.082 M/sec                    (16.70%)
    11,964,635,507     LLC-load-misses           #   45.10% of all LL-cache hits     (21.86%)
  <not supported>      L1-icache-loads
    12,139,313,035     L1-icache-load-misses     #    2.307 M/sec                    (28.53%)
 3,996,507,473,413     dTLB-loads                #  759.357 M/sec                    (21.46%)
     4,198,057,989     dTLB-load-misses          #    0.11% of all dTLB cache hits   (18.96%)
     1,276,700,163     iTLB-loads                #    0.243 M/sec                    (16.68%)
       624,640,855     iTLB-load-misses          #   48.93% of all iTLB cache hits   (21.60%)
```

- **CPU intensive, consume 17+ cores**
- **Cache locality is not good**

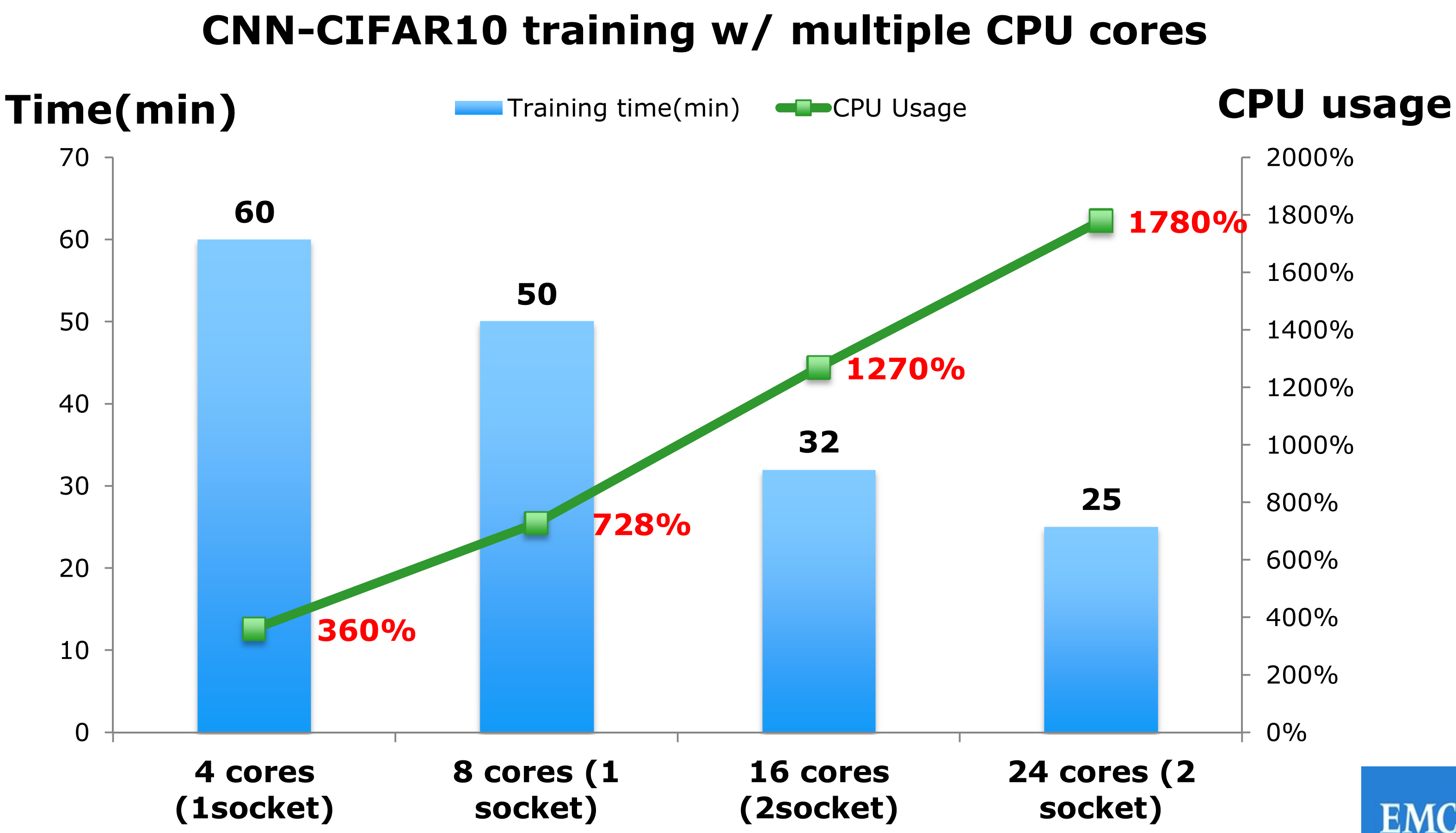
Balancing btw CPUs

- Very good balancing btw cores



Performance Scaling w/ CPU

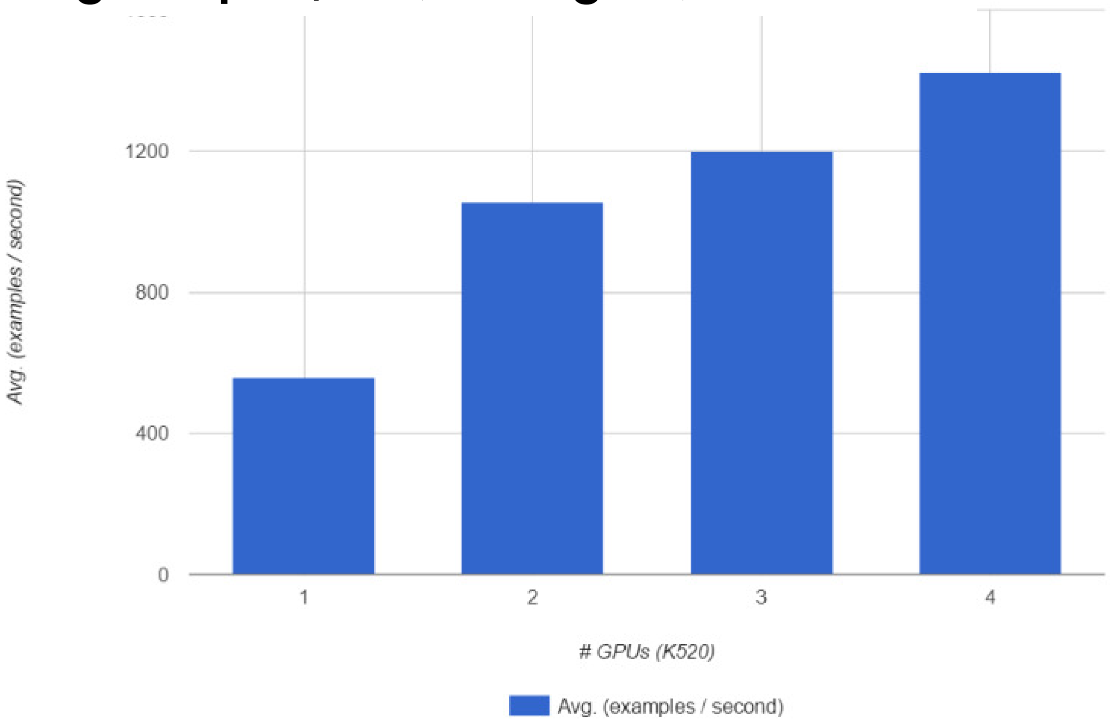
- 4X cores gains 1.8X faster
  - looks ok, but <u>not quite linear</u>. rooms to improve?



Performance Scaling w/ GPU

- 4X GPU achieves 2.4X faster
  - AWS g2.8xlarge, 4*K520 GPU

Avg samples/sec, the higher, the better



benchmark from http://www.bitfusion.io/2016/05/09/easy-tensorflow-model-training-aws/

Distribution: Parallelism & parameter update

Data parallel (Full-graph)	**Model parallel (Btw-graph)**
Deterministic model. Same codes in nodes/workers	fine-grained control of ops on node/workers; may different
Simple, easy to implement & control	Complex, may with high accuracy

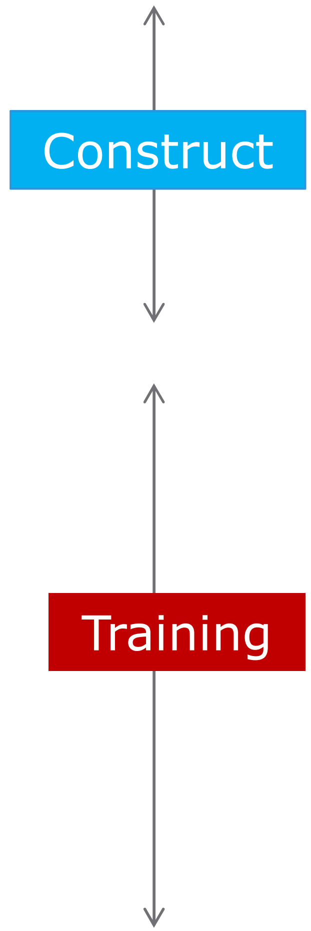
Synchronous update	**Asynchronous update**
sync-point between workers. Prevents from "falling behind"	Worker runs as fast as they can w/o sync-point
high quality/accuracy, but slower* (or to optimize)	Usually fast, but relative lower accuracy due to fall-behind

EMC²

Distributed training steps, by Docker+K8S

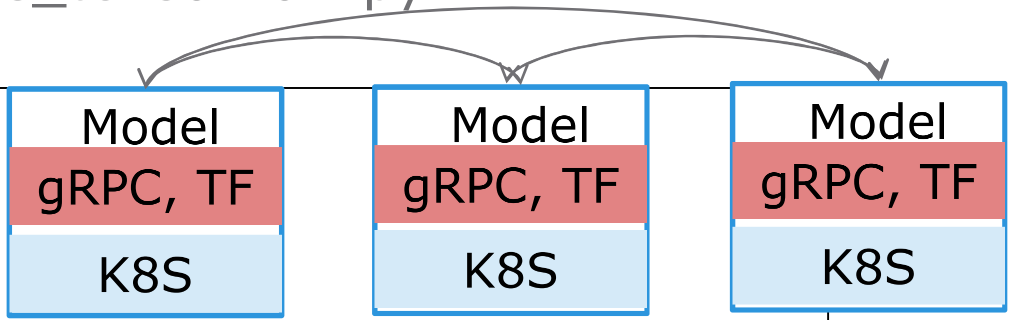
1. Construct the graph,
   - full-graph (data parallel) for Inception and MNIST
   - Sync or async; place worker on CPU or GPU
   - Setup termination condition (training steps or accuracy)

2. Package as container
   - Graph model, TF running lib, gRPC, or even dataset
3. Deploy by K8S
   - Specify job/worker and index
4. Feed dataset and start training
   - Container bind-volume, and batch load
5. Monitoring, visualization & report



Training cluster config

- K8S config yaml
  - tools/dist_test/scripts/k8s_tensorflow.py



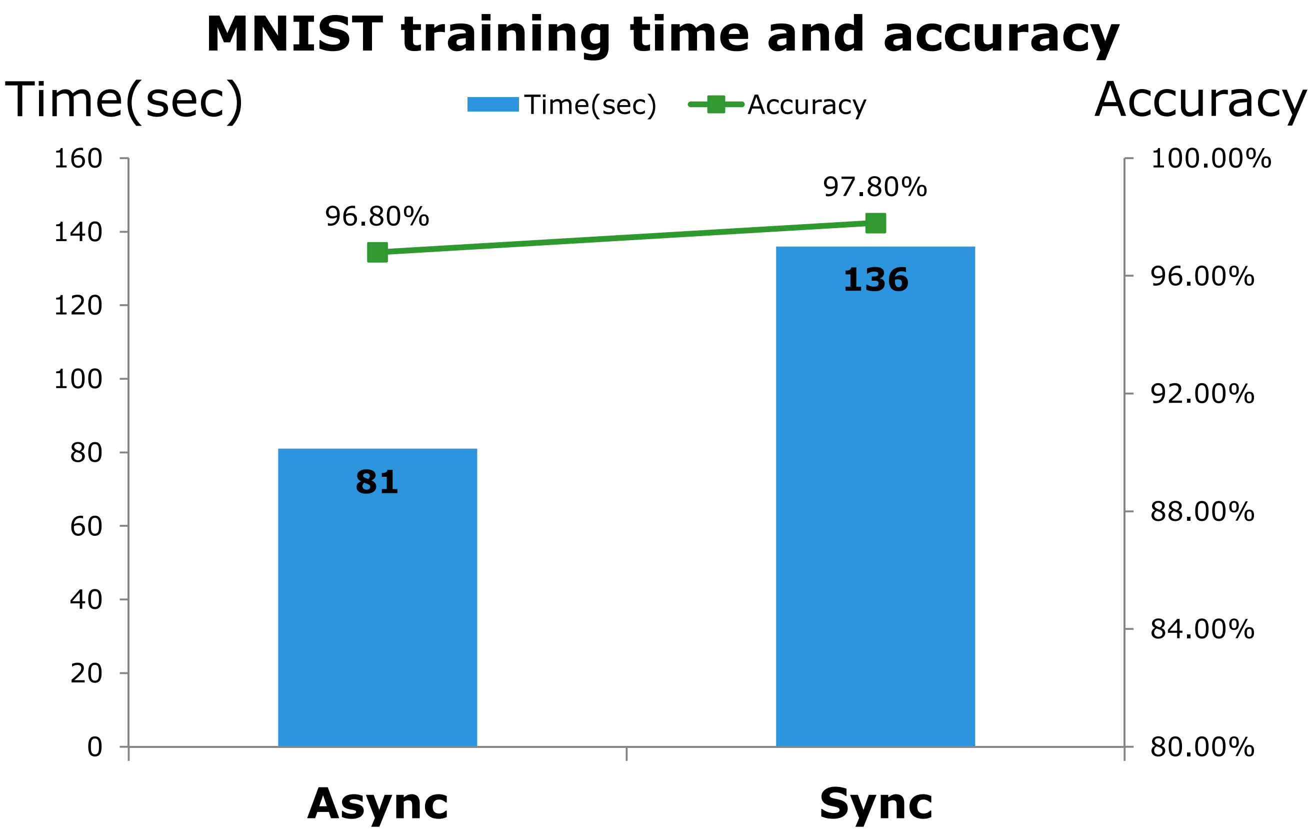

```yaml
apiVersion: v1
kind: ReplicationController
metadata:
  name: tf-worker0
spec:
  replicas: 1
  template:
    metadata:
      labels:
        tf-worker: "0"
    spec:
      containers:
      - name: tf-worker0
        image: tensorflow/tf_grpc_test_server
        args:
          - --cluster_spec=worker|tf-worker0:2222;tf-worker1:2222;tf-worker2:2222,
          - --job_name=worker
          - --task_id=0
        ports:
        - containerPort: 2222
      nodeSelector:
        nodeName: "tower1"
```

Docker image w/
model + TF+gRPC

Cluster spec;
specify PS|worker and index

Distribute containers in **1** node

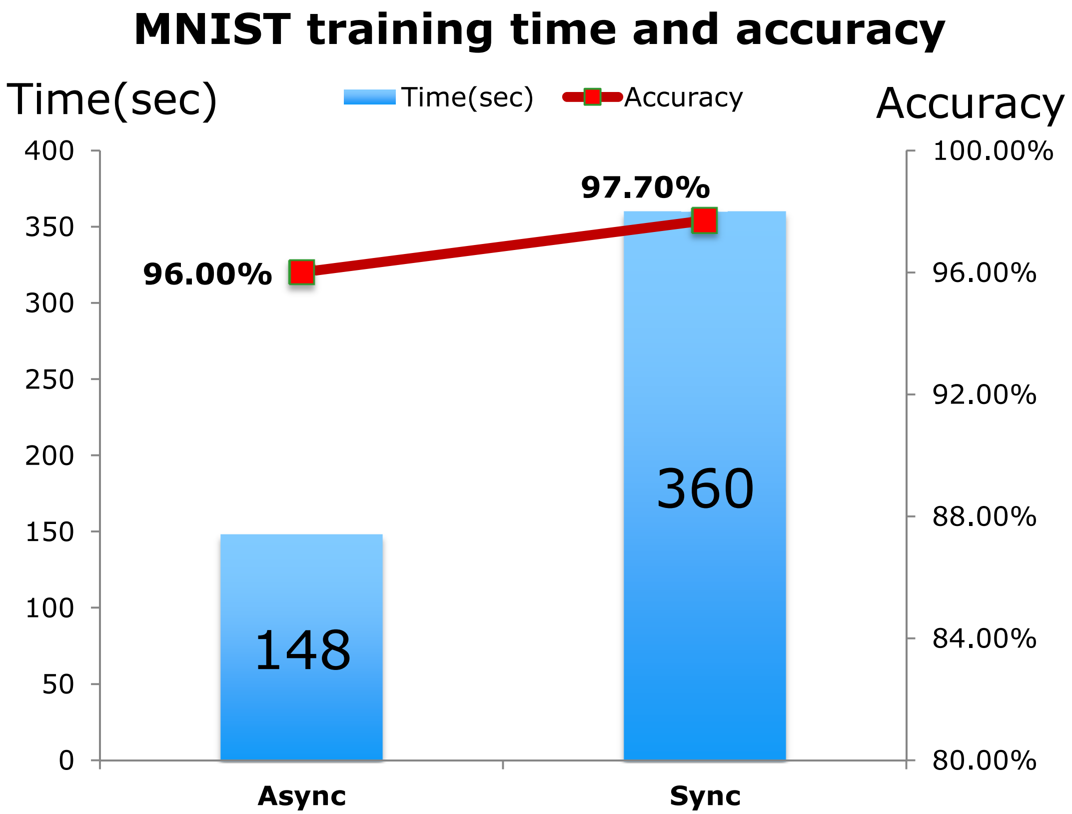
- Scheduled 3 containers in 1 node using K8S "nodeSelector"
- Async vs. sync

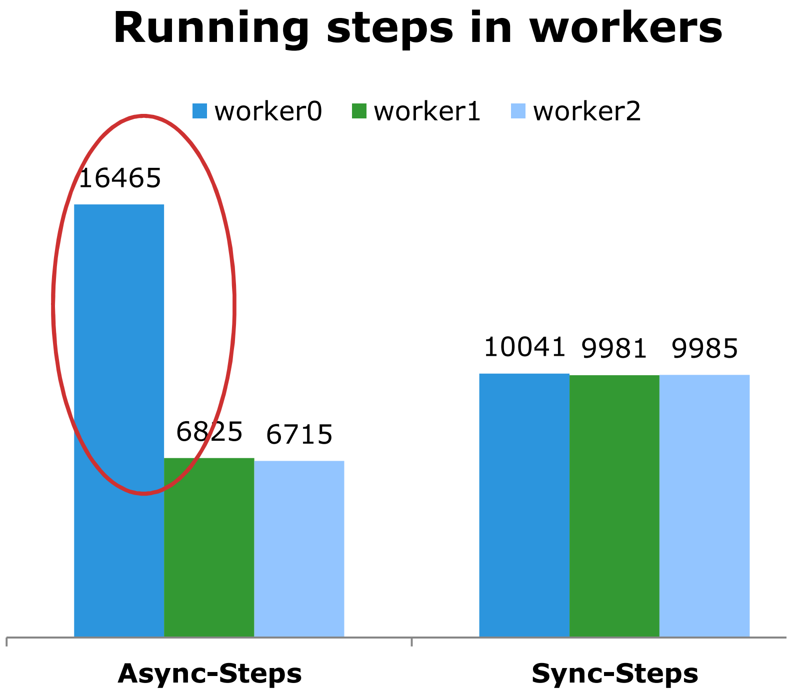


Distribute containers in 3 nodes

Each node has 1 worker container

- Async vs. sync





Observation and Considerations

- Async vs. Sync, Efficiency
  - Async is much faster than sync (**68%** in one node)
    - 143% in 3 nodes, but network is weak and stack is not fully optimized
  - Communication and sync overhead are bottlenecks
    - Fine tuned distributed training can achieve **13X faster w/ 16 nodes***
  - **RDMA,** parameter **local cache** + **bulk async update** back
    - Spark and Nvidia HPC etc claimed get 50+% higher perf
- Data parallel
  - Partition dataset across workers and proper mini-batch
    - shared bind-volume across workers
  - Not balanced in async way - to dig out
- Pod placement on nodes
  - Automatic, or use K8S nodeSelector/nodeAffinity
- Ops placement on CPU/GPU device
  - Automatic, or explicit TF API: with tf.device(name): {ops}

Observation and Considerations(2)

- Instance number
  - Moderate num per node to reduce communication overhead
- CPU/Mem throttling
  - Config "Request/Limit" only if necessary i.e., resource-shared env.
    - Container level, K8S pod, K8S replica or quota
  - Carefully set batch size to avoid OOM
  - Fully exploit GPU mem at first
- Networking concern, Flannel VLAN overhead
  - Use "--net=host" with native network and map to ports
  - RDMA is recommended; need powerful HW/driver
- With GPU?
  - Nvidia-Docker-plugin
  - GPU memory is fast but capacity is limited - a few GB
    - **Mem tiering**: GPU mem as Tier1, swap to CPU mem
    - Reference: "Scalable deep learning on distributed GPUs with a GPU-specialized parameter server" EuroSys16

Export model

- Checkpoint then export it
  - saver() : create a ckpt into files
  - Exporter(): set signature, version etc
  - CreateSessionBundle(): to load model at specific path

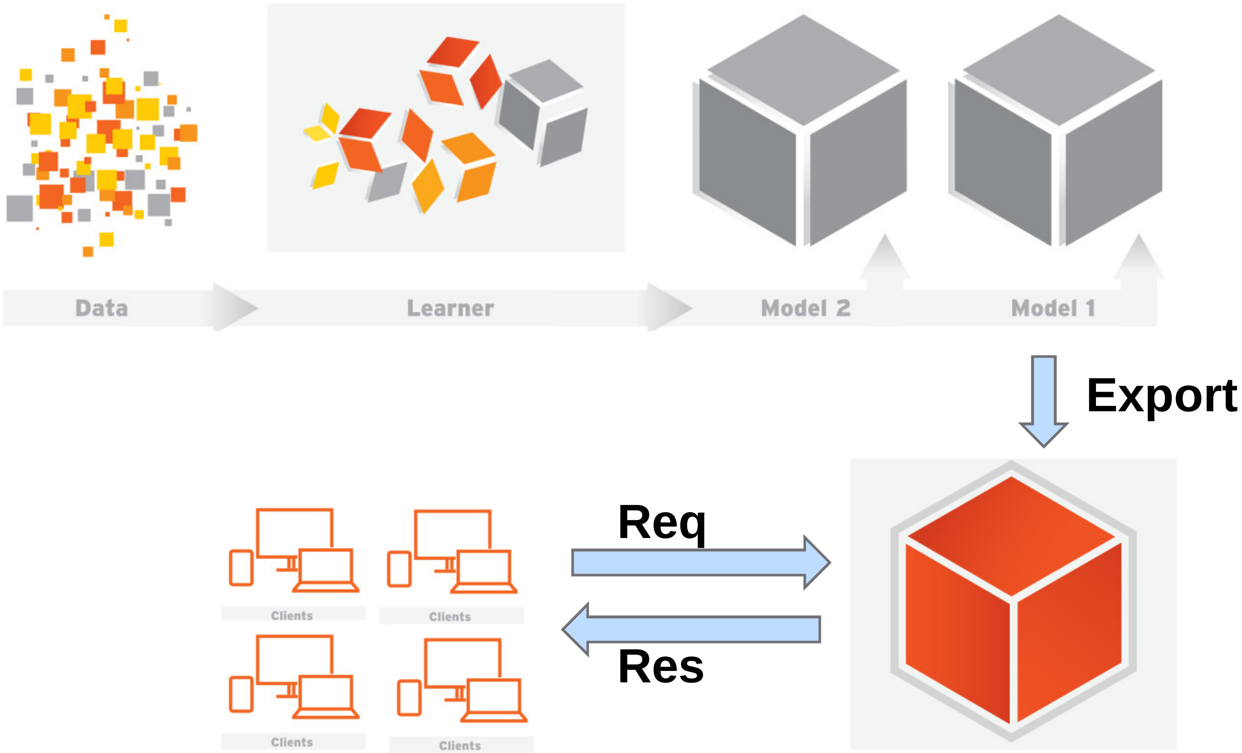
```
root@:/serving# ll inception-export
total 107M
-rw-r--r--   69 Jun 21 09:25 checkpoint
-rw-r--r-- 107M Jun 21 09:25 export
-rw-r--r-- 140K Jun 21 09:25 export.meta
```

DL serving

- Online service to the world based on trained model
  - No label dataset, no lose function and backpropagation

Continuous Training

Online Serving



Distributed Serving by Docker + K8S

- K8S features for deployment and management
  - Deployment
  - Auto-scale
  - Rolling-upgrade

Prepare the image
+ Include tensorflow,
+ tensorflow_serving
+ exported model/parameter

Export service port
Example: gcr.io/tensorflow-serving/inception

K8S: "deployment"

- **Highly recommended,** rather than k8s RC, pod
  - Auto manage desired state, i.e, autoscale (HPA)
    - With service and Replicaset
  - Easy to rolling upgrade on the fly

```
{
 "apiVersion": "extensions/v1beta1",
 "kind": "Deployment",
 "metadata": {
  "name": "inception.com",
  "namespace": "kube-system"
 },
 "spec": {
  "replicas": 1,
  "template": {
     "spec": {
     "containers": [
```

Same ns w/ heapster

```
"name": "inception-container",
"image": "inception_serving:v1",
 "imagePullPolicy": "IfNotPresent",
       "command": [
          ],
  "resources": {
     "requests": {
       "cpu": "6000m"
    }
```

Required for auto-scale

K8S: auto-scale

- Need Heapster (Influxdb) to collect matrix

```
# kubectl get pods --all-namespaces -o wide
NAME                                      READY     STATUS    AGE    NODE
heapster-v1.0.2-3098241085-1yv64          4/4       Running   1h     sm-tower2
monitoring-influxdb-grafana-v3-pjib8      2/2       Running   2h     sm-tower1
inception.com-2008606290-16akw            1/1       Running   29s    sm-tower1
inception.com-2008606290-a1uae            1/1       Running   29s    sm-tower2
inception.com-2008606290-va0co            1/1       Running   29s    sm-tower0
kube-controller-manager-sm-tower0         1/1       Running   14d    sm-tower0
```

kubectl **autoscale** deployment inception.com
--min=1 --max=3 --cpu-percent=50 --namespace="kube-system"

K8S: auto-scale result

- Increase workload, CPU get hot

```
# kubectl get hpa --namespace="kube-system"
NAME              REFERENCE                        TARGET    CURRENT
inception.com     Deployment/inception.com/scale   50%       878%
```

Auto-scale# ← in 4+ minutes

kubectl **describe deployments** inception.com

```
LastSeen   Count    Reason              Message
--------   -----    ------              -------
  4m         1    ScalingReplicaSet   Scaled up replica set inception.co
```

K8S: rolling upgrade

- Minimize service impact via online rolling upgrade
- ”apply”
  - Replace instance in graceful pace

```
"spec": {
  "containers": [
    {
      "name": "inception-container",
      "image": "inception_serving:v2",
      "imagePullPolicy": "IfNotPresent",
```

```
# kubectl apply -f inception-v2.json --validate=false
deployment "inception.com" configured
service "inception-service" configured
```

K8S: rolling-upgrade in-progress

- kubectl **get pods** --namespace="kube-system" -o wide

```
NAME                              STATUS        RESTARTS   AGE   NODE
inception.com-2008606290-0w8w2    Terminating   0          8m    sm-tower1
inception.com-2008606290-2kxou    Terminating   0          37m   sm-tower0
inception.com-2008606290-vq977    Terminating   0          8m    sm-tower2
inception.com-2103174739-1cjic    Running       0          18s   sm-tower0

...

inception.com-2103174739-1cjic    Running       0          14m   sm-tower0
inception.com-2103174739-4kws4    Running       0          34s   sm-tower1
inception.com-2103174739-ypwlc    Running       0          4m    sm-tower2
```

Considerations, and gaps

- Overall, K8S deployment is wonderful for serving
- Pod placement, re-scheduling
  - Use nodeSelector or nodeAffinity, label based
  - Gap: only for initial placement, not for dynamic execution

- Auto-scaling matrix and rule
  - Now CPU usage based, need heapster config
  - Gap: need GPU support and advanced rules

- Auto scale-down?
  - K8S supports, but not stable in my test.
  - Use "resize" cmd to manually adjust
  - What if scaling meets auto-upgrade??

Summary

- Explore typical DL workload
- Practice end-end DL distributed training and serving
  - Docker + K8S + TF
  - More deep optimization work are in-progress, see next
- TF, as one of hottest DL frameworks, provides good docker/K8S support, distributed version is promising but still in early stage
  - Perf scaling w/ multiple CPU/GPU can be improved
  - Data distribution is not always balanced
  - Lack of deep guideline, and profiling/tuning module (“EEG”) is not open-sourced
- K8S: auto-scale based on GPU usage or mixed?
-

In-progress & outlook

- Community
  - TF and Google
    - C++/Python optimization; More platforms (iOS, Windows, OpenCL etc)
    - Optimization for distribution, memcpy btw host and GPU etc
    - Google services: Datalab, dataflow(Apache Beam), cloud ML
  - K8S: advanced matrix for auto-scale; node affinity

- Our in-progress/plan
  - Heterogeneous cluster w/ powerful HW
    - GPU + CPU, 100GbE+RDMA/GPUDirect, NVRAM/NVMe
  - Unified memory and tiering for parameter update
  - Smart container/ops placement and (re-)scheduling
  - Efficient data partition/paralleling, like HPC

→ **"DL as a Service"**

References

- TensorFlow: A system for large-scale machine learning; on Heterogeneous Distributed Systems
  - http://arxiv.org/abs/1605.08695 and http://bit.ly/tf-workshop-slides
- K8S and TF serving
  - http://blog.kubernetes.io/2016/03/scaling-neural-network-image-classification-using-Kubernetes-with-TensorFlow-Serving.html
- GeePS: Scalable deep learning on distributed GPUs with a GPU-specialized parameter server EuroSys 2016
- Google cloud DL services, TPU
  - http://conferences.oreilly.com/strata/hadoop-big-data-ca/public/schedule/detail/50445
  - https://cloudplatform.googleblog.com/2016/05/Google-supercharges-machine-learning-tasks-with-custom-chip.html
- Scaling TF on Spark
  - https://spark-summit.org/east-2016/events/distributed-tensor-flow-on-spark-scaling-googles-deep-learning-library/
- Docker SWARM vs K8s
  - https://blog.docker.com/2016/03/swarmweek-docker-swarm-exceeds-kubernetes-scale/

Thank You! 谢谢！

ありがとう！

Questions?