Build the A.I. Cloud

Provision and manage Tensorflow cluster with OpenStack

Layne Peng & Accela Zhao
Who are we & Why are we here?

Layne Peng, Principal Technologist at DellEMC OCTO, experienced in cloud computing areas, responsible for cloud computing related initiatives since joint in 2011, 10+ patents and a big data and cloud computing book author.

Email: layne.peng@dell.com
Twitter: @layne_peng

Accela Zhao, (former) Technologist at DellEMC OCTO, active OpenStack community contributor, experienced in cloud scheduling and container technologies.

Email: accelazh@gmail.com
Who are we & Why are we here?
Who are we & Why are we here?

DellEMC OCTO ARD

- Hardware
- Fast Data
- SW Systems
- Data Science

Building Infrastructure for analytic

© Copyright 2016 Dell Inc.
Who are we & Why are we here?

DellEMC OCTO ARD

Hardware  Fast Data  SW Systems  Data Science

Building Infrastructure for analytic

Since Essex

a. Heterogeneous:  
   - Various accelerators
   - A lot of machines
   - On demand provision
   - Multiple projects
A.I. & Deep Learning

Birth 1952–1956
• Cybernetics and early neural networks
• Symbolic reasoning and the Logic Theorist
• Dartmouth Conference 1956: the birth of AI

Golden years 1956–1974
• Reasoning as search
• Natural language
• Micro-worlds

Boom 1980–1987
• Expert systems
• Hopfield net
• Backpropagation

Optimism & Winter comes time by time:
• Limit computer power
• Intractability and the combinatorial explosion
• Moravec's paradox

* Image from: http://www.lalalandrecords.com
A.I. & Deep Learning

Birth 1952–1956
- Cybernetics and early neural networks
- Symbolic reasoning and the Logic Theorist
- Dartmouth Conference 1956: the birth of AI

Golden years 1956–1974
- Reasoning as search
- Natural language
- Micro-worlds

Boom 1980–1987
- Expert systems
- Hopfield net
- Backpropagation

Optimism & Winter comes time by time:
- Limit computer power
- Intractability and the combinatorial explosion
- “Impossible” tasks: Moravec's paradox

But, cloud computing, large-scale cluster, new hardware, deep learning techs bring new lights to AI area

* Image from: http://www.lalalandrecords.com

* Slide by Andrew Ng, all rights reserved.
**Tensorflow**

Deep Learning framework from Google
- GPU/CPU/TPU, heterogeneous platform
- C++, Python
- Distributed training and serving
- DNN building block, ckpt/queue/…
- Docker and Kubernetes supported
Tensorflow

Deep Learning framework from Google
- GPU/CPU/TPU, heterogeneous platform
- C++, Python
- Distributed training and serving
- DNN building block, ckpt/queue/…
- Docker and Kubernetes supported

Most popular Deep Learning framework since it open source

<table>
<thead>
<tr>
<th></th>
<th>Stars</th>
<th>Forks</th>
<th>Contributoers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tensorflow</td>
<td>26995</td>
<td>10723</td>
<td>286</td>
</tr>
<tr>
<td>Caffe</td>
<td>10973</td>
<td>6575</td>
<td>196</td>
</tr>
<tr>
<td>CNTK</td>
<td>5699</td>
<td>1173</td>
<td>69</td>
</tr>
<tr>
<td>Torch</td>
<td>4852</td>
<td>1360</td>
<td>100</td>
</tr>
<tr>
<td>Theano</td>
<td>4022</td>
<td>1448</td>
<td>234</td>
</tr>
<tr>
<td>MXNet</td>
<td>4173</td>
<td>1515</td>
<td>152</td>
</tr>
<tr>
<td>Apache SINGA</td>
<td>607</td>
<td>211</td>
<td>18</td>
</tr>
</tbody>
</table>

*A more detailed summary and comparison [https://github.com/zer0n/deepframeworks](https://github.com/zer0n/deepframeworks)*
Tensorflow

• **Flexible** to construct the compute, define the operators
• **Auto-Differentiation** for difficult algorithms
• **Portable** to run in PC or cloud, different hardware such as CPU, GPU or other cards
• **Connect Research and Production** by providing *Training-* -> *Serving* model
• **Distributed** training and serving
Tensorflow

- **Flexible** to construct the compute, define the operators, support different language
- **Auto-Differentiation** for difficult algorithms
- **Portable** to run in PC or cloud, different hardware such as CPU, GPU or other cards
- **Connect Research and Production** by providing *Training-* > *Serving* model
- **Distributed** training and serving
TensorFlow

- Flexible to construct the compute, define the operators, support different language
- Auto-Differentiation for difficult algorithms
- **Portable** to run in PC or cloud, different hardware such as CPU, GPU or other cards
- Connect Research and Production by providing *Training-* > *Serving* model
- Distributed training and serving

Portable => support variable hardware to accelerate the computing

- Add new features by adding new Ops
- Add specified hardware accelerators by add new Kernels to the Ops
  - ✓ Current support: CPU and GPU
  - ✓ We are adding more…

*Cifar10 training, Tensorflow v0.9
1xTesla K40c vs. 4xE5-2660 v2
Known issue*, < 40% performance
Tensorflow

- Flexible to construct the compute, define the operators, support different language
- Auto-Differentiation for difficult algorithms
- Portable to run in PC or cloud, different hardware such as CPU, GPU or other cards
- **Connect Research and Production** by providing *Training-* > *Serving* model
- Distributed training and serving

Connect Research and Production

![Diagram showing the process of training and serving with two clusters](image-url)
Tensorflow

- Flexible to construct the compute, define the operators, support different language
- Auto-Differentiation for difficult algorithms
- Portable to run in PC or cloud, different hardware such as CPU, GPU or other cards
- Connect Research and Production by providing Training -> Serving model
- Distributed training and serving

Distributed (since v0.8)

* Image from: Large Scale Distributed Deep Networks
A small training cluster

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"
  ]})
Deep Learning & Cloud

A small training cluster

Cluster spec

Cluster

Workers

Task 1

Task 2

Worker Service

Parameters

gRPC

A small training cluster

Cluster spec

Worker Service

Parameters

gRPC

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"
  ]})

But, in production environment, there are thousands of servers need to be coordinated, complex environments …

* Benchmark from: https://research.googleblog.com/2016/04/announcing-tensorflow-08-now-with.html
But, in production environment, there are thousands of servers need to be coordinated, complex environments …

We need
OpenStack Integration Options

- Scalabilities?
- Can support **heterogeneous** environment?
- Flexible enough for **extending** new features?
- Can hide the plumbing for system engineers and data scientists?
OpenStack Integration Options

- Scalabilities?
- Can support *heterogeneous* environment?
- Flexible enough for *extending* new features?
- Can hide the plumbing for system engineers and data scientists?

Option 1: Integrated by Magnum
Option 2: Integrated by Sahara
OpenStack Integration Options 1 - Magnum

Magnum – OpenStack Container Orchestration Solution

- Provision of popular container platform
  - Kubernetes
  - Mesos
  - Swarm
- Abstract cluster management
  - Baymodel
  - Bay
- Integrated with Cinder to provision volume service for container
  - Massive dataset to train
Magnum Architecture
OpenStack Integration Options 1 - Magnum

Magnum – OpenStack Container Orchestration Solution

- Provision of popular container platform
  - Kubernetes
  - Mesos
  - Swarm
- Abstract cluster management
  - Baymodel
  - Bay
- Integrated with Cinder to provision volume service for container
  - Massive dataset to train
OpenStack Integration Options 1 - Magnum

Why? - Tensorflow (both Training and Serving) **officially** supported integrated with Kubernetes

Step 1: Package Workers & Parameters
Server or Serving node into images

- tf_worker:v1
- tf_ps:v1
- tf_inception:v1

Step 2: Create the clusters according to Tensorflow’s cluster spec (training)

```python
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"
    ]
)
```
OpenStack Integration Options 1 - Magnum

Why? - Tensorflow (both Training and Serving) **officially** supported integrated with Kubernetes

---

**Step 1:** Package Workers & Parameters
Server or Serving node into images

- tf_worker:v1
- tf_ps:v1
- tf_incenption:v1

**Step 2:** Create the clusters according to Tensorflow's cluster spec (training)

```python
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"
    ]
})
```
## Magnum Integration – Pros & Cons

<table>
<thead>
<tr>
<th>Pros</th>
<th>Cons</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Tensorflow has good support on Kubernetes, and well provisioned by Magnum&lt;br&gt;• Ready to use. No need to implement extra plugins or drivers&lt;br&gt;• Benefit from Magnum features such as tenant management, volumes, scaling, baremetal supported, which fully leverage OpenStack ecosystem</td>
<td>• OpenStack doesn’t have direct control/integration to Tensorflow, which is managed by upper-level container platform&lt;br&gt;• No one-step deployment&lt;br&gt;• Tensorflow deep learning is not made the first-class OpenStack citizen</td>
</tr>
</tbody>
</table>
Additional Pros – Simple to extend Scheduler

One of the many “black magic” in Kubernetes:
1. Node selector
2. Node affinity (since v1.2)

Ref: [http://kubernetes.io/docs/user-guide/node-selection](http://kubernetes.io/docs/user-guide/node-selection)

Add “nodeSelector” field to pod configuration:

```
"nodeSelector": {
  "octo.emc.kubernetes.io/gpu": "true",
  "octo.emc.kubernetes.io/offload": "true"
}
```

Extra benefit:
Integrated with Mesos, which can provide more intelligent scheduling features
Additional Pros – Rolling Update

Very IMPORTANT in Tensorflow Serving…
1. New trained model improve the effects
2. Minimize service impact.

Sample

Simply “Apply” updated spec:

```
# kubectl apply -f inception-v2.json --validate=false
```

Verify

Check the progress:

```
# kubectl get pods --namespace="kube-system" -o wide
```

<table>
<thead>
<tr>
<th>NAME</th>
<th>STATUS</th>
<th>RESTARTS</th>
<th>AGE</th>
<th>NODE</th>
</tr>
</thead>
<tbody>
<tr>
<td>inception.com-2008606290-0w8w2</td>
<td>Terminating</td>
<td>0</td>
<td>8m</td>
<td>am-tower1</td>
</tr>
<tr>
<td>inception.com-2008606290-2kxou</td>
<td>Terminating</td>
<td>0</td>
<td>37m</td>
<td>am-tower0</td>
</tr>
<tr>
<td>inception.com-2008606290-vq977</td>
<td>Terminating</td>
<td>0</td>
<td>6m</td>
<td>am-tower2</td>
</tr>
<tr>
<td>inception.com-2103174739-1cjc</td>
<td>Running</td>
<td>0</td>
<td>18s</td>
<td>am-tower0</td>
</tr>
<tr>
<td>inception.com-2103174739-1cjic</td>
<td>Running</td>
<td>0</td>
<td>14m</td>
<td>am-tower0</td>
</tr>
<tr>
<td>inception.com-2103174739-4kwa4</td>
<td>Running</td>
<td>0</td>
<td>34s</td>
<td>am-tower1</td>
</tr>
<tr>
<td>inception.com-2103174739-ypwlc</td>
<td>Running</td>
<td>0</td>
<td>4m</td>
<td>am-tower2</td>
</tr>
</tbody>
</table>
Additional Pros – Auto-scaling

Auto collect metrics and scale algorithms…

**Sample**

1. Heapster (Influxdb) collect the metrics:

   ```
   # kubectl get pods --all-namespaces --o wide
   NAME             READY   STATUS    AGE     NODE     SIZE   VCPU   CPU   MEMORY   IMAGE
   heapster-v1.0.2-3098241085-lyv64  4/4     Running 1h     am-tower2  443Mi   1v     50%    1Gi
   monitoring-influxdb-grafana-v3-pjib8  2/2     Running 2h     am-tower1  1Gi    2v     50%    1Gi
   inception.com-2008606290-16akw  1/1     Running 29s     am-tower1  1Gi    2v     50%    1Gi
   inception.com-2008606290-a1uec  1/1     Running 29s     am-tower2  1Gi    2v     50%    1Gi
   inception.com-2008606290-va0co  1/1     Running 29s     am-tower2  1Gi    2v     50%    1Gi
   kube-controller-manager-am-tower0  1/1     Running 14d     am-tower0  292Mi  2v     50%    292Mi
   ``

2. Add auto-scaling capability to deployment:

   ```
   #… --min=1 --max=3 --cpu-percent=50 --namespace="kube-system"
   ```

**Verify**

1. Increase workload…

   ```
   # kubectl hpa --namespace=kube-system
   NAME  REFERENCE                   TARGET   CURRENT
   inception.com Deployment/inception.com/scale 50%  71%
   ```

2. Check the deployment:

   ```
   LastScan  Count  Reason  Message
   --------  -----  -------  ---------
   m   1  ScalingReplicaSet  Scaled up replica set inception.com
   ```
OpenStack Integration Options 2 - Sahara

Sahara is official OpenStack Data Processing Solution

- Self-service provisioning of big data clusters
  - Vanilla Hadoop
  - Hortonworks
  - Spark
  - Cloudera
  - and more
- Elastic Data Processing (EDP) for workflow execution
  - Data focus solution vs. Resource focus solution
  - Data is the first citizen
- UI integration with Horizon

Sahara Architecture

- Auth. components
- Data Access Layer (DAL)
- Secure Storage Access Layer
- Provisioning Engine
- Vendor plugins
- Elastic Data Processing (EDP)
- REST API
- Python Sahara Client
- Sahara pages
## Sahara Integration – Pros & Cons

<table>
<thead>
<tr>
<th><strong>Pros</strong></th>
<th><strong>Cons</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>• Sahara is the OpenStack-native big data provisioning and analytic-as-a-service solution</td>
<td>• Not much community support</td>
</tr>
<tr>
<td>• Manage Tensorflow provisioning and job workflow by extending Sahara plugin framework and EDP framework. Provides high level abstraction to data scientists</td>
<td>• Need to implement Tensorflow plugin and EDP for Sahara</td>
</tr>
</tbody>
</table>
| • Benefits from Sahara cluster scheduling, scaling, storage management, integration with Horizon UI and more | • Miss all existed community work of collaboration between Tensorflow and Kubernetes:  
  ✓ Rolling update  
  ✓ Auto-scale |
OpenStack Integration Options

Chosen option: Integrated by Magnum

- Very simple to implement and solve problem
- Easy to extend and modification
- Enough for not a multi-tenant environment

So does it solve every problem?
Hardware Functions & Virtualization

- Heterogeneous hardware (GPU, Offload Card, NVMe RAM, Multi-core CPU …) provides us acceleration capabilities for Deep Learning;
- OpenStack provides capabilities:
  - Flexible for adding new features;
  - As a Service APIs
  - Ecosystem to leverage

But...
- Traditional OpenStack deployment is based on Virtualization;
- Not all the hardware functions support VT, even some declared they supported.

Temporary (or not) proposal solutions:
- Bare metal & virtualization hybrid environment?
- Containerization & virtualization hybrid environment?
Bare metal & Virtualization Hybrid Environment

Ironic is a “skin” to make bare metal work like virtual machine in OpenStack!

Key components:
• ironic-api
• ironic-conductor
• ironic-python-agent
• Nova-driver

k8s_fedora_ironic_v1 driver in Magnum!

Magnum and Sahara are able to work on hybrid environment contains bare metal and virtual machines:
2-level Scheduling and Scaling

Chosen option: Integrated by Magnum
2-level Scheduling and Auto-scaling

Bare metal

Nova Filter Scheduler*:
  • Bare metal or virtual machine
  • Boot the host with what kind of hardware functions

Auto-scale the Kubernetes Cluster:
  • Basically, it is Heat…

Container

Kubernetes node selector & node affinity:
  • Add hardware functions sensor
  • Extend build-in node labels

Kubernetes auto-scaling support:
  • Notified to scale

Pass-through the hardware functions sense.

* Ref: http://docs.openstack.org/developer/nova/filter_scheduler.html
New Trends?

Mirantis Collaborates with Intel and Google to Enable OpenStack on Kubernetes

Mirantis, the pure-play OpenStack company, today announced a collaboration with Google and Intel to evolve the architecture of the leading purpose-built lifecycle management tool for OpenStack, Fuel, and related OpenStack projects, to enable the use of Kubernetes as their underlying orchestration engine. The companies will work with the OpenStack community to package OpenStack into Docker containers to be managed by Kubernetes. The companies will jointly discuss the details of this collaboration at the upcoming OpenStack Days Silicon Valley on August 9-10.

OpenStack centric to Kubernetes centric?

Can we run part of Tensorflow in Kubernetes, part in OpenStack?

- Worker nodes in containers managed by Kubernetes;
- Coordinator nodes in virtual machines managed by OpenStack;
- Use Kuryr for connecting those parts?