Cloud based Shared Cluster for HPC/BigData

2016-07-14

Namgon Lucas Kim
namgon@smartx.kr

Networked Computing Systems Lab,
Gwangju Institute of Science and Technology (GIST)
SmartX Shared Cluster based on Apache Mesos

Resource Orchestration

Workload

Profiling (Benchmark)

HPC/Big Data Apps

Workflow

Operational Data

2016-07-01 KCC 2016
Ironic Deploy and Running Environment (Concept)

Ironic Baremetal Provisioning:
- pBox
- (p+v)Box

VM Provisioning:
- pBox
- (p+v)Box

Ironic Python Agent (IPA)
Ironic Deploy Image

User Image

Hypervisor

VM

Hypervisor
Ironic Bare-metal Deployment Steps

Figure 1.3.3. Bare Metal Deployment Steps

1. Nova boot
2. Apply filters & find available compute host node
3. Compute Manager calls driver.spawn()
4. Get info and claim bare metal node
5. Fetch images
6. Plug VIFs
7. Deploy bare metal node
8. Deploy (active boot loader)
9. Power on bare metal node
10. Write image
11. Reboot
12. Update status of bare metal node
PXE Boot and Direct Deploy Process

- Nova
  - Set instance info
  - Set provision state
  - Image source: root.qb, etc.
  - do_node_deploy()

- API
  - API call

- Conductor
  - Cache images
  - Update pxelinux
  - Update DHCPBOOT
  - power on

- Neutron
  - next-server = Conductor
  - Attempts tftpboot

- TFTP/HTTP
  - Attempts tftpboot
  - Send deploy kernel, ramdisk and config

- Node
  - DHCP request
  - Sends agent ramdisk
  - Lookup()

- Pass UUID

- Heartbeat
  - Continue deploy: Pass image, disk info

- Node downloads image, writes to disk

- Heartbeat periodically

- Is deploy done yet?

- Still working

- When deploy is done

- Clear DHCPBOOT
  - Set bootdev HDD

- Reboot

- Reboots into user instance
What is Mesos?

• Let us treat a cluster of nodes as one big computer
• **A cluster resource negotiator** (manager)
• Scalable to 10,000s of nodes
• A top-level Apache project (July. 2013~)
What is Mesos? (Cont’d)
What is Mesos? (Cont’d)

• Compare with Cloud Services

  PaaS
  Deploy and manage Applications/services

  Mesos
  Build and run Distributed systems using resources

  IaaS
  Provision and manage machines
Powered by Mesos

Twitter
Airbnb
Vimeo
OpenTable
HubSpot
Ignidata
xogito
PayPal
Atlassian
Cloud Physics
DeviceScape
Mesosphere

http://mesosphere.github.io/presentations
The Berkeley Data Analytics Stack

- An open source software stack that integrates software components being built by the AMPLab to make sense of Big Data.

<table>
<thead>
<tr>
<th>In-house Apps</th>
<th>Cancer Genomics</th>
<th>Energy Debugging</th>
<th>Smart Buildings</th>
</tr>
</thead>
<tbody>
<tr>
<td>Access and Interfaces</td>
<td>Sample Clean</td>
<td>SparkR</td>
<td>MLBase</td>
</tr>
<tr>
<td></td>
<td>G-OLA</td>
<td>GraphX</td>
<td>ML Pipelines</td>
</tr>
<tr>
<td></td>
<td>BlinkDB</td>
<td>Splash</td>
<td>MLLib</td>
</tr>
<tr>
<td></td>
<td>SparkSQL</td>
<td></td>
<td>Velox</td>
</tr>
<tr>
<td>Processing Engine</td>
<td>Spark Streaming</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Spark Core</td>
<td></td>
</tr>
<tr>
<td>Storage</td>
<td>Succinct</td>
<td>Tachyon</td>
<td></td>
</tr>
<tr>
<td></td>
<td>HDFS, S3, Ceph</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Resource Virtualization</td>
<td>Mesos</td>
<td></td>
<td>Hadoop Yarn</td>
</tr>
</tbody>
</table>

[Diagram showing the stack with colors indicating AMPLab Developed, Spark Community, 3rd Party, In Development]
Understanding Mesos

• Provides a platform for different frameworks
• Want to run multiple frameworks in a single cluster
  – ...to *maximize utilization*
  – ...to *share data* between frameworks

Ref: NSDI 2011 Presentation
Element 1: Fine-Grained Sharing

Coarse-Grained Sharing (HPC):

- Framework 1
- Framework 2
- Framework 3
- Storage System (e.g. HDFS)

Fine-Grained Sharing (Mesos):

- Fw. 1
- Fw. 2
- Fw. 3
- Storage System (e.g. HDFS)

+ Improved utilization, responsiveness, data locality

Ref: NSDI 2011 Presentation
Element 2: Resource Offers

• Mesos: Resource offers
  – Offer available resources to frameworks, let them pick which resources to use and which tasks to launch

+ Keep Mesos simple, make it possible to support future frameworks

- Decentralized decisions might not be optimal

Ref: NSDI 2011 Presentation
Mesos Resource Allocation

- Dominant Resource Fairness (DRF): a generalization of max-min fairness
- Strict priorities
  - Quincy: a centralized scheduling algorithm for Dryad’s DAG-based programming model
  - Delay scheduling: to achieve data locality

- **Apply max-min fairness to dominant shares**
- Equalize the dominant share of the users

- Example:
  - Total resources: <9 CPU, 18 GB>
  - User 1 demand: <1 CPU, 4 GB> dom res: mem
  - User 2 demand: <3 CPU, 1 GB> dom res: CPU
Dominant Resource Fairness (DRF) [NSDI 2011]

- NSDI 2011, Basic idea of resource sharing

```
Algorithm 1 DRF pseudo-code

\( R = (r_1, \ldots, r_m) \) \quad \triangleright \text{total resource capacities}
\( C = (c_1, \ldots, c_m) \) \quad \triangleright \text{consumed resources, initially 0}
\( s_i \) \quad (i = 1..n) \quad \triangleright \text{user } i\text{'s dominant shares, initially 0}
\( U_i = (u_{i,1}, \ldots, u_{i,m}) \) \quad (i = 1..n) \quad \triangleright \text{resources given to user } i, \text{initially 0}

pick user \( i \) with lowest dominant share \( s_i \)
\( D_i \leftarrow \text{demand of user } i\text{'s next task} \)
if \( C + D_i \leq R \) then
    \( C = C + D_i \) \quad \triangleright \text{update consumed vector}
    \( U_i = U_i + D_i \) \quad \triangleright \text{update } i\text{'s allocation vector}
    \( s_i = \max_{j=1}^m \{u_{i,j} / r_j\} \)
else
    return \quad \triangleright \text{the cluster is full}
end if
```

User 1: <1 CPU, 4 GB> User 2: <3 CPU, 1 GB>

↓

User 1: <1 CPU, 4 GB> User 2: <3 CPU, 2 GB>
Overview of Mesos Execution

Framework = scheduler + executor
Mesos Tools and Apps

• Ops Tools
  – Collectd, Deploy script, Chef cookbook, Puppet Module, Vagrant setup

• Developer Tools
  – Go Bindings, Framework Templeteate, Xcode Workspace
Mesos Architecture and Resource Offer Example

Task distribution, launching, monitoring, failure detection, killing, and cleanup

Resource isolation with containers
Containerization in Mesos
(Automated) Mesos Cluster Provisioning

- Supported OS:
  - Ubuntu 14.04~
  - CentOS 7.2 (Planned)
- Target SW:
  - Mesos
  - Marathon
  - Zookeeper

DevOps Tower: Mesos Master, ...

- mesos-configuration-master.sh
- mesos-configuration-slave.sh (through SSH)

sx-box-m1
sx-box-m2
sx-box-m3
Running MPI workload on Mesos Cluster

- (Popular) MPI Libraries: **MPICH**, OpenMPI, Intel MPI
- MPI Process Manager
  - **hydra**: Default of MPICH3, launches processes using the native daemons present on the system such as ssh, slurm, pbs, etc.
  - **gforker**: a simple process manager that runs all processes on a single node
- Mesos MPI Framework
  - **MPICH2 1.2** (using mpd)
  - **MPICH2-hydra**
Spark: a specialized framework of Mesos

• Optimized iterative jobs for machine learning
• Uses the **long-lived nature of Mesos executors** to cache a slice of the dataset in memory at each executor, and then run multiple iterations on this cached data.
• Example (Fig. 4 in NSDI 2011 paper)
  – Logistic regression (from T. Hastie et al. The Elements of Statistical Learning: Data Mining, Inference, and Prediction.)

Figure 4: Data flow of a logistic regression job in Dryad vs. Spark. Solid lines show data flow within the framework. Dashed lines show reads from a distributed file system. Spark reuses in-memory data across iterations to improve efficiency.
Mesos Framework for SmartX
Type-M (example)

Register, Resource Request, Launch Task

Launch Task (Call executor)

Resource Offer, Task Status

Task status

ZooKeeper

Mesos Master

Marathon
Scheduler
Driver (Libmesos)

Spark
Scheduler
Driver (Libmesos)

Mesos Slave
Mesos-docker-executor
Spark-executor
Task

Mesos Slave
Mesos-docker-executor
Spark-executor
Task

Mesos Slave
Mesos-docker-executor
Spark-executor
Task

Mesos Slave
Mesos-docker-executor
Spark-executor
Task

2016-07-01 KCC 2016
SmartX Box Cluster Configuration

< Hardware >

Master

Dell R210:
- Xeon x3450@2.66Ghz 4C8T, DDR3 16GB, 320GB HDD, 2-port 10GbE, 2-port 1GbE

(M1~M2):
- 2 of Xeon E5-2650v3@2.3Ghz, 10C20T, DDR4 128GB,
- 400 GB NVMe (PCIe SSD), 2 of 3TB HDD, 2-port 10GbE, 2-port 1GbE,
- *Nvidia Tesla K40m*

< Software >

Type-M Master

- OS: Ubuntu 14.04.4 (Kernel 4.4)
- Mesos 0.28.1
- Docker 1.11.0
- Marathon
- Spark 1.5.2 (Master)

Type-M Slaver/Worker

- OS: Ubuntu 14.04.4 (Kernel 4.4)
- Mesos 0.28.1 / Docker 1.11.0
- Spark 1.5.2 (Slave)
- HDFS (Hadoop) 2.7.1: DataNode
- *CUDA 7.5*
Benchmarking Experiments with BigData Workloads

- Spark-perf workflow and parameters for K-means test

```
num-examples 1,000,000
num-features 10,000
can_scale
num-centers 20
can_scale
num-iterations 20
can_scale
num-partitions 128
Random-seed 5
SCALE_FACTOR 1.0 (default)
```

If `can_scale` == true max(1, val*SCALE_FACTOR)
Mesos Run Modes in Spark

- **Fine-grained (default)**
  - each Spark task runs as a separate Mesos task
  - allows multiple instances of Spark (and other frameworks) to share machines at a very fine granularity
  - it comes with an additional overhead in launching each task

- **Coarse-grained**
  - launch only *one* long-running Spark task on each Mesos machine, and dynamically schedule its own “mini-tasks” within it
  - is much lower startup overhead, but at the cost of reserving the Mesos resources
Result of Benchmarking on SmartX Mesos Cluster

- K-means Test using Spark-Perf on Type-M Cluster (M1+M2)
  - Scale-factor: 0.001, 0.001, 0.1, 0.5, 1, 5, 10, 100
  - Scale-factor 1 = 20 x AWS EC2 m1.xlarge (4 vCPU, 15 GB RAM, 1.6TB)
  - Coarse-grained or Fine-grained for each scale-factor

![K-Means Test Result on Mesos Cluster (SmartX Type-M)](chart1)

- 0.1 22.5% faster than 0.5

![K-Means Test Result on Mesos Cluster (SmartX Type-M)](chart2)
Q & A

• Thank you!
Linux Container and Docker

- Docker uses libcontainer as the default execution driver from version 0.9.

http://blog.docker.com/2014/03/docker-0-9-introducing-execution-drivers-and-libcontainer/

https://medium.com/aws-activate-startup-blog/a-better-dev-test-experience-docker-and-aws-291da5ab1238
VMs vs Containers

- Containers are isolated, but share OS
- Hypervisor provides hardware-level virtualization

- Containers provides OS-level virtualization
Open Platform for Containers: Docker

- **Docker** is an open platform to easily create lightweight, portable, self-sufficient containers from any application that will run virtually anywhere

  - **Features**
    - Based on Linux containers
    - Minimal overhead
    - Uses a layered filesystem to save space (Another Union File System)
    - Uses a copy-on-write filesystem to track changes
    - Can run on any Linux system that supports LXC ...
    - Docker container contains everything it needs to run

  - First-class citizen in Mesos and the Mesosphere stack
  - Kubernetes employs and promotes Docker
Dependency Hell

Static website
- nginx 1.5 + modsecurity + openssl + bootstrap 2

Background workers
- Python 3.0 + celery + pyredis + libcurl + ffmpeg + libopencv + nodejs + phantomjs

Web frontend
- Ruby + Rails + sass + Unicorn

User DB
- postgresql + pgv8 + v8

Queue
- Redis + redis-sentinel

Analytics DB
- hadoop + hive + thrift + OpenJDK

API endpoint
- Python 2.7 + Flask + pyredis + celery + psycopg + postgresql-client

Do services and apps interact appropriately?

Can I migrate smoothly and quickly?

Multiplicity of Stacks

Development VM

QA server

Public Cloud

Disaster recovery

Production Cluster

Customer Data Center

Production Servers

Contributor’s laptop
Docker eliminates the matrix from hell

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Static website</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Web frontend</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Background workers</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>User DB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Analytics DB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Queue</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Marathon

• Distributed "init" for long-running services

- Start, stop, scale, update apps
- Nice web interface, API
- Highly available, no single point of failure (SPoF)
- Native Docker support
- Fully featured REST API
- Pluggable event bus
- Rolling deploy / restart
- Application health checks
- Artifact staging
### Marathon (Cont’d)

<table>
<thead>
<tr>
<th>ID</th>
<th>Memory (MB)</th>
<th>CPUs</th>
<th>Tasks / Instances</th>
<th>Health</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>/chronos</td>
<td>512</td>
<td>0.5</td>
<td>1 / 1</td>
<td>Green</td>
<td>Running</td>
</tr>
<tr>
<td>/cpu-waster</td>
<td>16</td>
<td>0.5</td>
<td>0 / 0</td>
<td>Green</td>
<td>Suspended</td>
</tr>
<tr>
<td>/dcos/service/history</td>
<td>512</td>
<td>0.5</td>
<td>0 / 0</td>
<td>Green</td>
<td>Suspended</td>
</tr>
<tr>
<td>/dispatch</td>
<td>128</td>
<td>0.5</td>
<td>1 / 1</td>
<td>Green</td>
<td>Running</td>
</tr>
<tr>
<td>/em/apollo</td>
<td>1024</td>
<td>1</td>
<td>0 / 0</td>
<td>Green</td>
<td>Suspended</td>
</tr>
<tr>
<td>/em/artemis</td>
<td>1024</td>
<td>1</td>
<td>0 / 0</td>
<td>Green</td>
<td>Suspended</td>
</tr>
<tr>
<td>/em/semaphore</td>
<td>16</td>
<td>0.1</td>
<td>1 / 1</td>
<td>Green</td>
<td>Running</td>
</tr>
<tr>
<td>/gollumwiki</td>
<td>256</td>
<td>0.01</td>
<td>0 / 0</td>
<td>Green</td>
<td>Suspended</td>
</tr>
<tr>
<td>/hdfs</td>
<td>512</td>
<td>1</td>
<td>1 / 1</td>
<td>Green</td>
<td>Running</td>
</tr>
<tr>
<td>/history</td>
<td>256</td>
<td>0.1</td>
<td>1 / 1</td>
<td>Green</td>
<td>Running</td>
</tr>
<tr>
<td>/jenkins</td>
<td>1024</td>
<td>1</td>
<td>1 / 1</td>
<td>Green</td>
<td>Running</td>
</tr>
</tbody>
</table>
Marathon (Cont’d)
Marathon (Cont’d)
Running Docker with Marathon

- Make JSON file as below

```json
{
  "container": {
    "type": "DOCKER",
    "docker": {
      "image": "ubuntu"
    }
  },
  "id": "ubuntu",
  "instances": "1",
  "cpus": "0.5",
  "mem": "512",
  "uris": [],
  "cmd": "while sleep 10; do date -u +%T; done"
}
```

- `curl -X POST -H "Content-Type: application/json" http://<master>:8080/v2/apps -d@Docker.json`
Running Docker with Marathon (Cont’d)
Kubernetes (K8S)

- *Inspired by Google's systems and experience:* Manage Containers, not Machines
- *Efficient & Robust:*  
  - optimized packing  
  - active monitoring  
  - self healing
- *Organizationally Scalable:*  
  - Enable micro-services  
  - active scaling
- *Modern Open Source:*  
  - Extensible & portable, can run anywhere  
  - Written in Go  
  - Hosted on github  
  - Apache 2.0 licensed

**Kubernetes Architecture**

**Kubelet**: the logical successor of the Container Agent that is part of the Compute Engine image

**Pod**: a relatively tightly coupled group of containers that are scheduled onto the same host
Kubernetes-Mesos

• enables the **Pod abstraction**, along with **Pod labels** for service discovery, load-balancing, and replication control

• Components
  – Executor: Package executor includes a mesos executor, which contains a kubernetes as its member to manage containers.
  – kubernetes-executor
  – kubernetes-mesos: apiserver is the main api server and master for the cluster.
  – Scheduler: Package framework includes a kubernetes framework.
Mesosphere DCOS

User Interface & CLI
Control your entire DCOS-powered datacenter from a single command line and a single web user interface.

Your Apps & Workloads
Run all your apps and workloads, from PaaS-based microservices to big data to databases — and everything in between.

Apache Mesos
Built in to the DCOS, Mesos pools your infrastructure, automatically allocating resources and scheduling tasks based on demands and policy.

Your Infrastructure
The DCOS runs in any modern Linux environment, public and private cloud, virtual machines and bare metal.
Mesosphere DCOS (Cont’d)

**DCOS deployed a PaaS**

Install Mesosphere DCOS on Your Favorite Cloud

Mesosphere DCOS can run on industry leading cloud providers or directly on-location. Select your provider or contact us.

Try It Now

Amazon AWS

Sign Up Today

Microsoft Azure

Sign Up Today

Google Cloud Platform

Need to run on-premises or around-the-clock support? Try our Enterprise Edition.

Sign Up Today
Mesos+Spark and Cloud Storage

• Spark atop Mesos on Google Cloud Platform.

https://www.concur.com/blog/en-us/spark-mesos-on-google-cloud
Make Own Mesos Framework

- Using Mesos API
Install Mesos on Ubuntu 14.04

• sudo apt-key adv --keyserver keyserver.ubuntu.com --recv E56151BF
• DISTRO=$(lsb_release -is | tr '[:upper:]' '[:lower:]')
• CODENAME=$(lsb_release -cs)
• echo "deb http://repos.mesosphere.io/${DISTRO} ${CODENAME} main" | sudo tee /etc/apt/sources.list.d/mesosphere.list
• sudo apt-get -y update
• [M] sudo apt-get -y install mesos marathon
• [S] sudo apt-get -y install mesos
• sudo reboot

https://mesosphere.io/learn/install_ubuntu_debian/
Mesos Configuration (Master)

• Disable slave
  – sudo service mesos-slave stop
  – echo manual > /etc/init/mesos-slave.override

• Zookeeper
  – change the /etc/mesos/zk file (IP address)

• Restart services
  – sudo service zookeeper restart
  – sudo service mesos-master restart
  – sudo service marathon restart

http://mesosphere.io/docs/getting-started/cloud-install/
Mesos Configuration (Slave)

- Disable Zookeeper
  - `sudo service zookeeper stop`
  - `echo manual > /etc/init/zookeeper.override`
- Disable Master
  - `sudo service mesos-master stop`
  - `echo manual > /etc/init/mesos-master.override`
- Edit zk file (configure Master)
  - `sudo vim /etc/mesos/zk`
- Configure IP when use public IP
  - `sudo echo {node IP} > /etc/mesos-slave/ip`
- Restart services
  - `sudo service mesos-slave restart`

http://mesosphere.io/docs/getting-started/cloud-install/
Install Docker on Ubuntu 14.04

• sudo apt-get update
• sudo apt-get -y install docker.io
• sudo ln -sf /usr/bin/docker.io /usr/local/bin/docker
Install Kubernetes

- `sudo aptitude install golang libprotobuf-dev mercurial`
- `cd $GOPATH` # If you don't have one, create directory and set GOPATH accordingly.
- `mkdir -p src/github.com/mesosphere/kubernetes-mesos`
- `git clone git@github.com:mesosphere/kubernetes-mesos.git src/github.com/mesosphere/kubernetes-mesos`
- `cd src/github.com/mesosphere/kubernetes-mesos` && `godep restore`
- `go install github.com/GoogleCloudPlatform/kubernetes/cmd/proxy`
- `go install github.com/mesosphere/kubernetes-mesos/kubernetes-{mesos,executor}`
Running Kubernetes

• `etcd -addr=210.125.84.150:4001 -peer-addr=210.125.84.150:7001`

• `./kubernetes-mesos --machines=210.125.84.150 --mesos_master=210.125.84.150:5050 --etcd_servers=http://210.125.84.150:4001 --executor_path=/root/Go/bin/kubernetes-executor --proxy_path=/root/Go/bin/proxy --address=210.125.84.150 --port=8088`
Appendix

APPENDIX-TROUBLE SHOOT
Troubleshooting

• LLDP error when MAAS commissioning (to use public IPs)
  – Add ‘210.125.84.0/24’ into /etc/squid-deb-proxy/allowed-networks-src.acl
  – Then, service sudo /etc/init.d/squid-deb-proxy restart

• Networking error BTW Master and Slaves
  – Setting Slave IP (for each Mesos-slave nodes)
    sudo echo {node IP} > /etc/mesos-slave/ip