Heterogeneous Resource Scheduling Using Apache Mesos for Cloud Native Frameworks

Sharma Podila
Senior Software Engineer
Netflix

Aug 20th
MesosCon 2015
Agenda

- Context, motivation
- Fenzo scheduler library
- Usage at Netflix
- Future direction
Why use Apache Mesos in a cloud?

Resource granularity

Application start latency
A tale of two frameworks

Reactive stream processing

Container deployment and management
Reactive stream processing, Mantis

- Cloud native
- Lightweight, dynamic jobs
  - Stateful, multi-stage
  - Real time, anomaly detection, etc.
- Task placement constraints
  - Cloud constructs
  - Resource utilization
- Service and batch style
Mantis job topology

A job is set of 1 or more stages
A stage is a set of 1 or more workers
A worker is a Mesos task
Container management, Titan

- Cloud native
- Service and batch workloads
- Jobs with multiple sets of container tasks
- Container placement constraints
  - Cloud constructs
  - Resource affinity
  - Task locality
Container scheduling model

Co-locate tasks from multiple task sets
Why develop a new framework?
Easy to write a new framework?
Easy to write a new framework?

What about scale?
Performance?
Fault tolerance?
Availability?
Easy to write a new framework?

What about scale?
Performance?
Fault tolerance?
Availability?

And scheduling is a hard problem to solve
Long term justification is needed to create a new Mesos framework
Our motivations for new framework

- Cloud native
  (autoscaling)
Our motivations for new framework

- Cloud native
  (autoscaling)

- Customizable task placement optimizations
  (Mix of service, batch, and stream topologies)
Cluster autoscaling challenge

For long running stateful services

Host 1
Host 2
Host 3
Host 4

VS.

Host 1
Host 2
Host 3
Host 4
Cluster autoscaling challenge

For long running stateful services

Host 1  Host 2  Host 3  Host 4

---

VS.

Host 1  Host 2  Host 3  Host 4
Components of a mesos framework

API for users to interact
Components of a mesos framework

API for users to interact

Be connected to Mesos via the driver
Components of a mesos framework

API for users to interact

Be connected to Mesos via the driver

Compute resource assignments for tasks
Components of a mesos framework

API for users to interact

Be connected to Mesos via the driver

Compute resource assignments for tasks
Fenzo

A common scheduling library for Mesos frameworks
Fenzo usage in frameworks

- **Mesos master**
- **Mesos framework**
- **Fenzo task scheduler**
- **Persistence**

- **Tasks requests**
- **Available resource offers**

**Task assignment result**
- Host1
  - Task1
  - Task2
- Host2
  - Task3
  - Task4
Fenzo scheduling library

- Heterogeneous resources
- Heterogeneous task requests
- Autoscaling of cluster
- Plugins for Constraints, Fitness
- Visibility of scheduler actions
- High speed
Announcing availability of Fenzo in Netflix OSS suite
Scheduling problem

$N$ tasks to assign from $M$ possible slaves
Scheduling optimizations

First fit assignment
\[ \sim O(1) \]

Optimal assignment
\[ \sim O(N * M) \]

Real world trade-offs

1 Assuming tasks are not reassigned
Scheduling strategy

For each task
On each host
  Validate **hard constraints**
  Eval **fitness** and **soft constraints**
Until fitness good enough, and
  A minimum #hosts evaluated
Task constraints

Soft
Hard
Task constraints

Soft
Hard
Extensible
Built-in Constraints
Host attribute value constraint

Task
HostAttrConstraint:instanceType=r3

Fenzo

Host1
Attr:instanceType=m3

Host2
Attr:instanceType=r3

Host3
Attr:instanceType=c3
Unique host attribute constraint

Task
UniqueAttr:zone

Fenzo

Host1
Attr:zone=1a

Host2
Attr:zone=1a

Host3
Attr:zone=1b
Balance host attribute constraint

Job with 9 tasks, BalanceAttr:zone

- Host1
  Attr: zone=1a

- Host2
  Attr: zone=1b

- Host3
  Attr: zone=1c

Fenzo
Fitness evaluation

Degree of fitness

Composable
Bin packing fitness calculator

fitness = usedCPUs / totalCPUs
Bin packing fitness calculator

\[ \text{fitness} = \frac{\text{usedCPUs}}{\text{totalCPUs}} \]

Fitness for:

- **Host1**: 0.25
- **Host2**: 0.5
- **Host3**: 0.75
- **Host4**: 1.0
- **Host5**: 0.0
Bin packing fitness calculator

\[ \text{fitness} = \frac{\text{usedCPUs}}{\text{totalCPUs}} \]

Fitness for:

- Host1: 0.25
- Host2: 0.5
- Host3: 0.75
- Host4: 1.0
- Host5: 0.0 (Circled with a red strike through it)
Composable fitness calculators

Fitness

\[ \text{Fitness} = \frac{\text{BinPackFitness} \times \text{BinPackWeight} + \text{RuntimePackFitness} \times \text{RuntimeWeight}}{2.0} \]
Cluster autoscaling in Fenzo

- ASG/cluster: computeCluster
  - MinIdle: 8
  - MaxIdle: 20
  - CooldownSecs: 360

ScaleUp action: Cluster, N

ScaleDown action: Cluster, HostList
Rules based cluster autoscaling

- Set up rules per host attribute value
  - E.g., one autoscale rule per ASG/cluster, one cluster for network-intensive jobs, another for CPU/memory-intensive jobs

- Sample:

<table>
<thead>
<tr>
<th>Cluster Name</th>
<th>Min Idle Count</th>
<th>Max Idle Count</th>
<th>Cooldown Secs</th>
</tr>
</thead>
<tbody>
<tr>
<td>NetworkClstr</td>
<td>5</td>
<td>15</td>
<td>360</td>
</tr>
<tr>
<td>ComputeClstr</td>
<td>10</td>
<td>20</td>
<td>300</td>
</tr>
</tbody>
</table>
Shortfall analysis based autoscaling

- Rule-based scale up has a cool down period
  - What if there’s a surge of incoming requests?
- Pending requests trigger shortfall analysis
  - Scale up happens regardless of cool down period
  - Remembers which tasks have already been covered
Usage at Netflix
Cluster autoscaling
# Scheduler run time (milliseconds)

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Average</strong></td>
<td>2 mS</td>
</tr>
<tr>
<td><strong>Maximum</strong></td>
<td>38 mS</td>
</tr>
</tbody>
</table>

Note: times can vary depending on # of tasks, # and types of constraints, and # of hosts
Experimenting with Fenzo

Note: Experiments can be run without requiring a physical cluster
A bin packing experiment

- Tasks with cpu=1
- Tasks with cpu=3
- Tasks with cpu=6

Iteratively assign Bunch of tasks

Start with idle cluster

Mesos Slaves
- Host 1
- Host 2
- Host 3

Host 3000
Bin packing sample results

Bin pack tasks using Fenzo’s built-in CPU bin packer
Bin packing sample results

Bin pack tasks using Fenzo’s built-in CPU bin packer
Task runtime bin packing sample

Bin pack tasks based on custom fitness calculator to pack short vs. long run time jobs separately
# Scheduler speed experiment

**Hosts:** 8-CPUs each  
**Task mix:** 20% running 1-CPU jobs, 40% running 4-CPU, and 40% running 6-CPU jobs  
**Goal:** starting from an empty cluster, assign tasks to fill all hosts  
**Scheduling strategy:** CPU bin packing

<table>
<thead>
<tr>
<th># of hosts</th>
<th># of tasks to assign each time</th>
<th>Avg time</th>
<th>Avg time per task</th>
<th>Min time</th>
<th>Max time</th>
<th>Total time</th>
</tr>
</thead>
</table>


Scheduler speed experiment

**Hosts:** 8-CPUs each

**Task mix:** 20% running 1-CPU jobs, 40% running 4-CPU, and 40% running 6-CPU jobs

**Goal:** starting from an empty cluster, assign tasks to fill all hosts

**Scheduling strategy:** CPU bin packing

<table>
<thead>
<tr>
<th># of hosts</th>
<th># of tasks to assign each time</th>
<th>Avg time</th>
<th>Avg time per task</th>
<th>Min time</th>
<th>Max time</th>
<th>Total time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,000</td>
<td>1</td>
<td>3 mS</td>
<td>3 mS</td>
<td>1 mS</td>
<td>188 mS</td>
<td>9 s</td>
</tr>
<tr>
<td>1,000</td>
<td>200</td>
<td>40 mS</td>
<td>0.2 mS</td>
<td>17 mS</td>
<td>100 mS</td>
<td>0.5 s</td>
</tr>
</tbody>
</table>
Scheduler speed experiment

**Hosts:** 8-CPU each  
**Task mix:** 20% running 1-CPU jobs, 40% running 4-CPU, and 40% running 6-CPU jobs  
**Goal:** starting from an empty cluster, assign tasks to fill all hosts  
**Scheduling strategy:** CPU bin packing

<table>
<thead>
<tr>
<th># of hosts</th>
<th># of tasks to assign each time</th>
<th>Avg time per task</th>
<th>Avg time</th>
<th>Min time</th>
<th>Max time</th>
<th>Total time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,000</td>
<td>1</td>
<td>3 mS</td>
<td>3 mS</td>
<td>1 mS</td>
<td>188 mS</td>
<td>9 s</td>
</tr>
<tr>
<td>1,000</td>
<td>200</td>
<td>40 mS</td>
<td>0.2 mS</td>
<td>17 mS</td>
<td>100 mS</td>
<td>0.5 s</td>
</tr>
<tr>
<td>10,000</td>
<td>1</td>
<td>29 mS</td>
<td>29 mS</td>
<td>10 mS</td>
<td>240 mS</td>
<td>870 s</td>
</tr>
<tr>
<td>10,000</td>
<td>200</td>
<td>132 mS</td>
<td>0.66 mS</td>
<td>22 mS</td>
<td>434 mS</td>
<td>19 s</td>
</tr>
</tbody>
</table>
Accessing Fenzo

Code at
https://github.com/Netflix/Fenzo

Wiki at
https://github.com/Netflix/Fenzo/wiki
Future directions

- Task management SLAs
- Support for newer Mesos features
- Collaboration
To summarize...
Fenzo: scheduling library for frameworks

- Heterogeneous resources
- Autoscaling of cluster
- Visibility of scheduler actions
- Plugins for Constraints, Fitness
- Heterogeneous task requests
- High speed
Fenzo is now available in Netflix OSS suite at
https://github.com/Netflix/Fenzo
Questions?

Heterogeneous Resource Scheduling Using Apache Mesos for Cloud Native Frameworks

Sharma Podila
spodila @ netflix . com

@podila