Elastic resource scheduling for Netflix's scalable container cloud

Sharma Podila, Andrew Spyker, Tomasz Bak
Feb 7th 2017
Topics

- Motivations for containers on AWS EC2
- Scheduling using Apache Mesos
- Fenzo deep dive
- Future plans
Containers add to our VM infrastructure

Already in VM’s have ...

- microservice driven,
- cloud native,
- CI/CD devops enabled,
- resilient,
- elastically scalable,
- environment
Container Provides Innovation Velocity

- Iterative local development, deploy when ready
- Manage app and dependencies easily and completely
- Simpler way to express resources, let system manage
Sampling of container usage

NodeJS UI Services

Operations and General Stream Processing

Media Encoding

Digital Watermarking

Reporting

Service

Batch
Sampling of realized container benefits

- **Media Encoding - encoding research development time**
  - VM’s platform to container platform - 1 month vs. 1 week

- **Continuous Integration Testing**
  - Build all Netflix codebases in hours
  - Saves development 100’s of hours of debugging

- **Netflix API Re-architecture using NodeJS**
  - Focus returns to app development
  - Provided reliable smaller instances
  - Simplifies, speeds test and deployment
Scheduling use cases
Reactive stream processing: Mantis

- Configurable message delivery guarantees
- Heterogeneous workloads
  - Real-time dashboarding, alerting
  - Anomaly detection, metric generation
  - Interactive exploration of streaming data
Current Mantis usage

- **At peak:**
  - 2,300 EC2 instances of M3.2xlarge instances
- **Peak of 900 concurrent jobs**
- **Peak of 5,200 concurrent containers**
  - Trough of 4,000 containers
  - Job sizes range from 1 to 500 containers
- **Mix of perpetual and interactive exploratory jobs**
- **Peak of 13 Million events / sec**
Container deployment: Titus

- Titus Job Control
- Cloud Platform (metrics, IPC, health)
- Batch Containers
- VPC
- EC2
- Atlas & Insight
- Eureka
- Edda
Current Titus usage

- Peak of ~1,800 instances
  - Mix of m4.4xl, r3.8xl, p2.8xl
  - ~800 instances at trough
- Mix of batch, stream processing, and some microservices

#Containers (tasks) for the week of 11/7 in one of the regions
Core architectural components

- AWS EC2
- Apache Mesos
- Fenzo at https://github.com/Netflix/Fenzo
- Batch Job Mgr
- Service Job Mgr

Apache Mesos at http://mesos.apache.org/
Scheduling using Apache Mesos
Mesos Architecture
Motivation for a new Mesos scheduler

- Cloud native (cluster autoscaling)
- Customizable task placement optimizations
  - Mix of service, batch, and stream topologies
What does a Mesos scheduler do?

- API for users to interact
- Mesos interaction via the driver
- Compute resource assignments for tasks
What does a Mesos scheduler do?

- API for users to interact
- Be connected to Mesos via the driver
- Compute resource assignments for tasks
  - NetflixOSS Fenzo
    - https://github.com/Netflix/Fenzo
Fenzo deep dive
Scheduling optimizations

- Speed
- Accuracy

First fit assignment
- Real world trade-offs
- Optimal assignment
Scheduling problem

\( N \) tasks to assign from \( M \) possible agents
Scheduling optimizations

Resource assignments
Scheduling optimizations

- Bin packing
  - By resource usage
  - By job types
- Ease deployment of new agent AMIs
- Ease server maintenance and upgrades
Scheduling optimizations

- Task locality, anti-locality (noisy neighbors?, etc.)
- Resource affinity
- Task balancing across racks/AZs hôsts
Scheduling optimizations

- Save cloud footprint costs
- Right instance types
- Save power, cooling costs
- Does everything need to run right away?
Scheduling optimizations

DC/Cloud operator

Application owner

Cost

Security

Security aspects of multi-tenant applications on a host
Scheduling optimizations

- DC/Cloud operator
- Application owner
- Cost
- Security

Proceed quickly in the generally right direction, adapting to changes
Fenko goals

- Extensible
- Cloud native
- Ease of experimentation
- Scheduling decisions visibility
Fenzo scheduling strategy

For each (ordered) task
   On each available host
      Validate **hard constraints**
      Score **fitness** and **soft constraints**
   Until score good enough, and
      A minimum #hosts evaluated
   Pick host with highest score
Experimentation with Fenzo

- Abstractions of tasks and servers (VMs)
- Create various strategies with custom fitness functions and constraints
  - For example, dynamic task anti-locality
- “Good enough” can be dynamic
  - Based on pending task set size, task type, etc.
- Ordering of servers for allocation based on task type
Experimentation with Fenzo

Resource bin packing sample results

Task runtime bin packing sample results
Fitness function Vs. constraints

- **Fitness: site policies**
  - Bin packing for utilization, reduce fragmentation
  - Segregate hosts by task types, e.g., service Vs batch

- **Constraints: user preferences**
  - Resource affinity
  - Task locality
  - Balance tasks across racks or availability zones
Fitness evaluation

- Degree of fitness, score of 0.0 - 1.0

- Composable
  - Multiple weighted fitness functions

- Extensible
  - Combine existing ones with custom plugins
CPU bin packing fitness function

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

\[
\text{fitness} = \frac{\text{usedCPUs}}{\text{totalCPUs}}
\]
CPU bin packing fitness function

fitness = \frac{\text{usedCPUs}}{\text{totalCPUs}}
Current fitness evaluator in Titus

Combines resource request bin packing with task type bin packing

\[
\text{resBinpack} = \frac{\text{cpuFit} + \text{memFit} + \text{networkFit}}{3.0} \\
\text{taskTypePack} = \frac{\text{numSameType}}{\text{totTasks}}
\]

\[
\text{fitness} = \text{resBinpack} \times 0.4 + \text{taskTypePack} \times 0.6
\]
Fenzo constraints

- **Common constraints built-in**
  - Host attribute value
  - Host with unique attribute value
  - Balance across hosts’ unique attribute value

- **Can be used as “soft” or “hard” constraint**
  - Soft evaluates to 0.0 - 1.0
  - Hard evaluates to `true/false`

- **Additional custom plugins**
  - Global constraint to send only GPU requiring tasks to GPU hosts
  - Global constraint to limit EC2 instance types to certain tasks
Fenzo supported resources

- CPU
- Memory
- Disk
- Ports
- Network bandwidth
- Scalar (used for GPU)
- Security groups and IP per container
Why is a task failing to launch?

```
{
  "ConstraintFailures": {
    "GlobalAgentClusterConstraint": {
      "Only runs on agent clusters: [r3.8xlarge, p2.8xlarge]": [...], // 283 items
      "Agent does not run non-GPU tasks": [
        "100.66.26.155",
        "100.66.47.192",
        "100.66.13.102",
        "100.66.31.2"
      ]
    }
  },
  "ResourceFailures": {
    "Memory": [
      "100.66.24.39(-201792.0)"
    ],
    "Network": [
      "100.66.29.87(-112.0)"
    ],
    "CPU": [
      "100.66.24.39(-27.0)",
      "100.66.5.129(-30.0)"
    ]
  }
}
```
Fenzo cluster autoscaling
Fenzo cluster autoscaling

Host 1

Host 2

Host 3

Host 4

VS.

Host 1

Host 2

Host 3

Host 4
Fenzo cluster autoscaling

Host 1
Host 2
Host 3
Host 4

vs.

Host 1
Host 2
Host 3
Host 4

VS.
Fenzo cluster autoscaling

- Threshold based
- Shortfall analysis based

Host 1 vs. Host 1
Host 2 vs. Host 2
Host 3 vs. Host 3
Host 4 vs. Host 4
Autoscaling multiple agent clusters

Grouping agents by instance type let’s us autoscale them independently

Titus

m4.4xlarge agents

r3.8xlarge agents

Min

Desired

Max

Min

Desired

Max
Threshold based autoscaling

- Set up rules per agent attribute value
- Sample:

<table>
<thead>
<tr>
<th>Cluster Name</th>
<th>Min Idle</th>
<th>Max Idle</th>
<th>Cooldown Secs</th>
</tr>
</thead>
<tbody>
<tr>
<td>MemosyClstr</td>
<td>2</td>
<td>5</td>
<td>360</td>
</tr>
<tr>
<td>ComputeClstr</td>
<td>5</td>
<td>10</td>
<td>300</td>
</tr>
</tbody>
</table>
Shortfall analysis based scale up

- 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
- Shortcoming: scale can be too aggressive for short periods of time
Capacity guarantees
Capacity guarantees

● Guarantee capacity for timely job starts
  ○ Mesos supports quotas, but, inadequate at this time
● Generally, optimize throughput for batch jobs and start latency for service jobs
● Categorize by expected behavior
  ○ For example, some service style jobs may be less important
● **Critical** versus **Flex** (flexible) scheduling requirements
Capacity guarantees

- Critical
- Flex

Quotas
Capacity guarantees

Quotas vs. Priorities

Critical Quotas
Flex Quotas

Critical Priorities
Flex Priorities

Resource Allocation Order
Capacity guarantees: hybrid view

Resource Allocation Order

Critical

Flex
Capacity guarantees: hybrid view

Tier Capacity = SUM (App1-cap + App2-cap + … + AppN-cap) + BUFFER

BUFFER:
- Accommodate some new or ad hoc jobs with no guarantees
- Red-black pushes of apps temporarily double app capacity
Fenzo supports multi-tiered task queues
   Can have arbitrary number of tiers

Per tier DRF across multiple queues
Sizing clusters for capacity guarantees

**Tier 0:**
- Used capacity
- Idle capacity
- Autoscaled

Cluster min size (guaranteed capacity)
Cluster max Size

**Tier 1:**
- Used capacity
- Autoscaled

Cluster desired size
Cluster max Size

(Idle size kept near zero)
Netflix container execution values
Netflix container execution values

- Consistent cloud infrastructure with VM’s
  - Virtualize and deeply re-use AWS features
- User and operator tooling common to VM’s
  - IPC and service discovery, telemetry and monitoring
  - Spinnaker integration for CI/CD
- Unique Features
  - Deep Amazon and Netflix infrastructure integration
  - VPC IP per container
  - Advanced security (sec groups, IAM Roles)
Elastic Network Interfaces (ENI)

- Each EC2 instance in VPC has 2 or more ENIs
- Each ENI can have 2 or more IPs
- Security Groups are set on the ENI
Network bandwidth isolation

Each container gets an IP on one of the ENIs

**Linux tc** policies used on virtual Ethernet
  For both incoming and outgoing traffic

Bandwidth limited to the requested value
  No bursting into unused bandwidth
GPU Enablement

Personalization and recommendations
- Deep learning with neural nets/mini batch
- Makes model training infrastructure self-service

Executor takes Scheduler resource definition
- Maps p2.8xl GPU’s using nvidia-docker-plugin
- Mounts drivers and devices into container
Ongoing, and future scheduling work

- **Fine grain capacity guarantees**
  - DRF adapted to elastic clusters
  - Preemptions to improve resource usage efficiency
  - Hierarchical sharing policies via h-DRF
  - Leveraging “Internal spot market”, aka the trough

- **Onboarding new applications**
  - Scale continues to grow
Elastic resource scheduling for Netflix's scalable container cloud

Questions?

Sharma Podila, Andrew Spyker, Tomasz Bak
@podila @aspyker @tomaszbak1974