Twine: A Unified Cluster Management System for Shared Infrastructure

Ashvin Goel

Electrical and Computer Engineering
University of Toronto

ECE1724

Authors: Facebook Infrastructure
Motivation

• A region consists of a few closely-located data centers
  • Contains ~1 million machines!

• Need a scalable, fault tolerant resource allocator for running jobs efficiently on all the machines in a region (across data centers)
We Looked at Mesos

Mesos Master

Allocate containers to tasks

Organization policies

Framework Scheduler

Schedule tasks in containers

Task schedule

Resource availability

Allocate containers to tasks
Why Would Mesos Not Work Well?

Problem 1: Single Mesos master will not scale well
What about This Approach?

Data Center 1

Resource availability

Mesos Master

Framework Scheduler

Task schedule

Data Center 2

Resource availability

Mesos Master

Framework Scheduler

Task schedule

Data Center 3

Resource availability

Mesos Master

Framework Scheduler

Task schedule
How Does Mesos Handle Failures?

Problem 2: Pushes node failure handling to framework, not ideal ...
Million Node Resource Management

- Scaling resource management is a clearly a challenge
- But there are many other issues
  - Hardware failures
    - Nodes may fail, multiple racks may fail due to power failures
  - Maintenance operations
    - Node hardware needs upgrades
    - Node software, e.g., kernel, libraries, etc., need updates
  - Software management
    - Application software needs bug fixes, new software releases
    - Custom kernel installations
  - Job resizing, relocation
    - Job tasks need to be added/removed, moved for data locality
Twine Architecture In a Nutshell

All components are sharded for scalability

Data Center 1

Resource Broker (per DC)

Resource availability

Data Center 2

Resource Broker (per DC)

Data Center 3

Resource Broker (per DC)

Track resource availability

Allocate containers to tasks

Create, manage, move tasks

Schedule, run tasks in containers

Allocators

Schedulers

App Schedulers

Task schedule

Task schedule

Task schedule

Task schedule
The next set of slides are a subset from the original Twine talk
Data center geographic regions

- Papillon, NE
- Prineville, OR
- Eagle Mountain, UT
- Los Lunas, NM
- Fort Worth TX
- Alcona, IA
- DeKalb, IL
- New Albany, OH
- Gallatin, TN
- Henrico, VA
- Forest City, NC
- Newton County, GA
- Huntsville, AL
- Luleå, Sweden
- Odense, Denmark
- Clonee, Ireland
- Singapore

New construction
Serving traffic
Cluster management systems help manage all of our services and machines.
What design decisions did Twine make differently?

<table>
<thead>
<tr>
<th>Decision 1</th>
<th>Decision 2</th>
<th>Decision 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dynamic machine partitioning over</td>
<td>Customization in shared infrastructure over</td>
<td>Small machines over big machines</td>
</tr>
<tr>
<td>static clusters</td>
<td>private pools</td>
<td></td>
</tr>
</tbody>
</table>
What design decisions did Twine make differently?

**Decision 1**

Dynamic machine partitioning over static clusters
What if we used Kubernetes?

Building large clusters

Support

At v1.19, Kubernetes supports clusters with up to 5000 nodes. More specifically, we support configurations that meet all of the following criteria:
What if we used Kubernetes?
What if we used Kubernetes?

Stranded capacity: M6, M7, M8 are available, but jobs in Cluster 1 cannot use them
What if we used Kubernetes?

**Stranded capacity:** M6, M7, M8 are available, but jobs in Cluster 1 cannot use them

**Intuition:** Make machine assignment dynamic.
How does Twine avoid stranded capacity?
How does Twine avoid stranded capacity?
How does Twine perform fleet-wide optimization?
How does Twine perform fleet-wide optimization?

Use M5 in newly-constructed Data Center 3 to improve spread for fault tolerance.
How does Twine perform fleet-wide optimization for an entire geographic region?
How does Twine perform fleet-wide optimization for an entire geographic region?

Shard Twine Scheduler by entitlements
How well does the Twine scheduler scale?
What design decisions did Twine make differently?

<table>
<thead>
<tr>
<th>Decision 1</th>
<th>Decision 2</th>
<th>Decision 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dynamic machine</td>
<td>Customization in shared infrastructure</td>
<td>Small machines over big machines</td>
</tr>
<tr>
<td>partitioning over</td>
<td>over</td>
<td></td>
</tr>
<tr>
<td>static clusters</td>
<td>private pools</td>
<td></td>
</tr>
</tbody>
</table>
Challenge: Tasks are not homogenous

Need at least 1 replica of each shard up at all times
Challenge: Tasks are not homogenous

Task 4 becomes unavailable.
Challenge: Tasks are not homogenous

Task 1 restarted for software release. Shard C unavailable!
Challenge: Tasks are not homogenous

**Intuition:** Collaborate with applications to handle lifecycle events.

Task A restarted for software release.
Shard C unavailable!
How does Twine collaborate with applications?
How does Twine collaborate with applications?

2 and 3 have no overlapping shards with 4, but overlap in B. Can update only one of these.
How does Twine collaborate with applications?

TaskController State

A
C

1

2

3

4 unavailable

Scheduler

Time

S0 ← Update Job
request=[1,2,3,4] completed=[ ]

S1

request=[1,3,4] completed=[2]

ack=[3]

S2

ack=[2]

S3

Can only update 3 to maintain C’s availability

TaskController
How does Twine collaborate with applications?
How does Twine collaborate with applications?

Scheduler

Time

S0

Update Job

request=[1,2,3,4] completed=[]

S1

ack=[2]

request=[1,3,4] completed=[2]

S2

ack=[3]

request=[1,4] completed=[3]

S3

ack=[1]

TaskController

Can update 1 or 4, but they overlap in C. Can update only one of these
What is our shared infrastructure adoption?
twshared: shared infrastructure for compute

- Web fully on twshared
- New compute capacity in twshared only
- Host profiles created
- TaskControl created

<table>
<thead>
<tr>
<th>Year</th>
<th>twshared / total fleet</th>
</tr>
</thead>
<tbody>
<tr>
<td>2013</td>
<td></td>
</tr>
<tr>
<td>2014</td>
<td></td>
</tr>
<tr>
<td>2015</td>
<td></td>
</tr>
<tr>
<td>2016</td>
<td></td>
</tr>
<tr>
<td>2017</td>
<td></td>
</tr>
<tr>
<td>2018</td>
<td></td>
</tr>
<tr>
<td>2019</td>
<td></td>
</tr>
<tr>
<td>2020</td>
<td></td>
</tr>
</tbody>
</table>
Conclusion
Evolving Twine over the past 10 years

Dynamic machine partitioning
Avoids stranded capacity in isolated clusters and enables fleet-wide optimizations

Customization in shared infrastructure
Support ubiquitous shared infrastructure to improve efficiency without sacrificing workload performance or capability

Small machines
Achieve higher power efficiency globally
Discussion
Q1

• Mesos uses a resource offer model to allocate containers for tasks. Twine requires app schedulers to specify the number of tasks and machines. Why the difference?
Q2

- Mesos is a two-level scheduler (Master, Frameworks). Why does Twine call itself a three-level scheduler?
Q3

• What could be a drawback of assigning tasks to a logical cluster (entitlement) instead of a physical cluster? How does Twine address this drawback?
Q4

- Twine uses sharding instead of federation to scale resource allocation across data centers. What are the tradeoffs?