Challenges in Optimizing Job Scheduling on Mesos

Alex Gaudio
Who Am I?
Who Am I?

- Data Scientist and Engineer at Sailthru
- Mesos User
- Creator of Relay.Mesos
Who Am I?

● Data Scientist and Engineer at Sailthru
  ○ Distributed Computation and Machine Learning

● Mesos User
  ○ 1 year

● Creator of Relay.Mesos
  ○ intelligently auto-scale Mesos tasks
Goals

1.

2. 

3. 
What are the goals of this talk?

1. Understand the problem of job scheduling using basic principles
What are the goals of this talk?

1. Understand the problem of job scheduling using basic principles

2. Learn ways to think about, use or develop **Mesos** more effectively
What are the goals of this talk?

1. Understand the problem of job scheduling using basic principles

2. Learn how to think about and use or develop Mesos more effectively

3. Have some fun along the way!
Contents

- The Problem of Utilization

- How does Mesos do (or not do) Job Scheduling?
The Problem of Utilization

Here’s a Box
The Problem of Utilization

It has 3 dimensions
What can you do with a box that has 3 dimensions?
What does this mean?!
The Problem of Utilization

Stuff the box
The Problem of Utilization

Unpack the box
The Problem of Utilization

Box in a box
The Problem of Utilization

Carry the box
The Problem of Utilization
The Problem of Utilization

Is really ...

All about the box!
The Problem of Utilization

By Example:

Please efficiently pack these stolen boxes into my get-away car!
The Problem of Utilization

Explained By Analogy
The Problem of Utilization

Box  Computer

A Computer is really just a Box
The Problem of Utilization

We can represent a box with 3 dimensions.
The Problem of Utilization

... If we relabel the dimensions
The Problem of Utilization

A computer, like a box, is a multi-dimensional object.
The Problem of Utilization

A computer, is just a collection of resources
The Problem of Utilization

If we put things in boxes,

What can we put in our computer?
The Problem of Utilization

What can we put in our computer?

Processes!
$ pstree

Output of a computer’s Process Tree
This is an interesting slide!

$ pstree
Why is the **pstree** slide interesting?

1. It introduces the concept of a **process**.

A process is an instance of code that accesses resources over time.
Why is the pstree slide interesting?

1. It introduces the concept of a process.

A process may use, share, steal, lock or release resources
Why is the *pstree* slide interesting?

2. It shows a computer with **multiple processes** running on it.
Why is the `pstree` slide interesting?

2. It shows a computer with **multiple processes** running on it.

- The processes access the same pool of resources.
Why is the pstree slide interesting?

2. It shows a computer with multiple processes running on it.

- Shared access to same pool of resources.
- Processes are categorized into a hierarchical structure.
At this point, we can ask a couple great questions!
At this point, we can ask a couple great questions!

- Why don't computers just have 1 process per box?
At this point, we can ask a couple great questions!

- Why don't computers just have 1 process per box?
- Is it inefficient to have so many processes on one box?
At this point, we can ask a couple great questions!

- Why don't computers just have 1 process per box?
- Is it inefficient to have so many processes on one box?
- Aren’t processes just another kind of box?
The Problem of Utilization

Let’s try to answer these questions!
The Problem of Utilization

Imagine a computer with only 2 resources.
The Problem of Utilization

Imagine a computer with only 2 resources.

Only 3 distinct process types run on this computer.
The Problem of Utilization

There is a fixed number of ways we can use up the computer’s resources.
The Problem of Utilization

There is a fixed number of ways we can use up the computer’s resources.

1 process at a time.
Could be great if all processes were the size of the computer
There is a fixed number of ways we can use up the computer’s resources.

2+ processes
Sharing resources
New Concept: Shared State
The Problem of Utilization

Different Utilization Strategies

Maximum Variation
Under-utilized
The Problem of Utilization

There is a fixed number of ways we can use up the computer's resources.

Maximum Utilization
No Variation
The Problem of Utilization

There is a fixed number of ways we can use up the computer’s resources.

Over-provisioned and Under-utilized
The Problem of Utilization

Competing for shared resources. Unclear consequences.

Over-provisioned and Under-utilized
The Problem of Utilization

A multi-dimensional problem!

And

very complicated!
Many ways we can use a computer’s resources.

Many different factors inform how we choose to utilize a set of resources.
Benefits of Shared State

- increased utilization
- flexibility to do different things simultaneously
- exposes a lot of interesting problems to solve
Drawbacks of Shared State

- **resource competition**
  - network and io congestion
  - context switching
  - out of memory errors

- **less predictable**
  - constantly changing dynamic systems
  - non-deterministic waiting
  - feedback loops
One machine, a host of problems

- Operating systems are complicated!
- Your laptop’s kernel solves these scheduling problems well.
WAIT A MINUTE

HOLD UP BOO BOO

Wait!
The Problem of Utilization

- Thus far, we’ve discussed resource utilization on 1 machine.
- Is 1 machine enough?
- And what about Mesos?
Obviously, 1 machine isn’t enough

- Problems of scale:
  - Too much data
  - Not enough compute power
  - Everything can’t connect to 1 node

- Problems of reliability and availability:
  - 1 machine is a Single Point of Failure
  - No redundancy
Many machines, then?
Mesos!
Recall the Box...

Box ➔ Computer

A Computer is really just a Box
Mesos is really just a box, too
AND Mesos is just a Computer

Double Analogy
Mesos is a Distributed Computer
Mesos is a Distributed Computer

- a lot of machines
- all solving the similar problems
Mesos is a Distributed Computer

- a lot of machines
- all solving the similar problems
- We need ways to tell each machine what to do.
Must rebuild all elements of an operating system in context of a distributed system!
Must rebuild all elements of an operating system in context of a distributed system!

IF HISTORY REPEATS ITSELF, I'M GETTING A DINOSAUR

Same old problems
Awesome new technology
Part 2:
Part 2:
How does Mesos do Job Scheduling?
How Mesos does Job Scheduling

A very big box
Let’s call it “Grid”
How Mesos does Job Scheduling

The “Grid” holds a lot of smaller boxes.

The little boxes are “Slaves”

Mesos Slaves
(aka computers or boxes)
How Mesos does Job Scheduling

Each slave is a partitioned pool of resources

Mesos Slaves
How Mesos does Job Scheduling

- Slaves advertise resources to Master
- Master packages resources into resource offers.
How Mesos does Job Scheduling

Mesos Slaves

Master offers resources to frameworks

Mesos Master

Frameworks
How Mesos does Job Scheduling

Mesos Slaves

Mesos Master

Frameworks

Frameworks accept or reject resource offers.
How Mesos does Job Scheduling

Accepted offers result in tasks that do useful work.
3 Types of Scheduling Architectures
(aka 3 Types of Distributed Kernels)

Monolithic  Two-level  Shared state

Mesos has a two-level architecture.
3 Types of Scheduling Architectures

Mesos Master
(manage resource and framework state)

Mesos Frameworks
(manage task state)

Two-level

subset

cluster state information
cluster machines

from the Google Omega Whitepaper
3 Types of Scheduling Architectures

Monolithic
- scheduling logic
- no concurrency
- cluster state information
- cluster machines

Two-level
- subset
- pessimistic concurrency (offers)

Shared state
- full state
- optimistic concurrency (transactions)

from the Google Omega Whitepaper
# 3 Types of Scheduling Architectures

(aka 3 Types of Distributed Kernels)

<table>
<thead>
<tr>
<th>Monolithic</th>
<th>Two-level</th>
<th>Shared state</th>
</tr>
</thead>
</table>

[Diagram showing a red arrow pointing from Monolithic to Two-level, with a goal flag on the right side.]
# 3 Types of Scheduling Architectures

(aka 3 Types of Distributed Kernels)

<table>
<thead>
<tr>
<th>Monolithic</th>
<th>Two-level</th>
<th>Shared state</th>
</tr>
</thead>
</table>

![Diagram of scheduling architectures](image)
3 Types of Scheduling Architectures

(aka 3 Types of Distributed Kernels)

- **Monolithic**: Borg (Google)
- **Two-level**: Hadoop YARN
- **Shared state**: Google
Remainder of this talk...

Point out weaknesses with Mesos that

1. Prevent it from being a shared state kernel.

2. Can make Mesos challenging to use.
Remainder of this talk...

1. Optimistic Vs Pessimistic Offers

2. DRF Algorithm and Framework Sorters

3. Missing APIs / Enhancements
Optimistic Vs Pessimistic Offers

We Trust Everyone!

Trust No One
Everyone promised not to take my spot.

Protect my spot from thieves!
Optimistic Vs Pessimistic Offers

Trust No One
Optimistic Vs Pessimistic Offers

- 2 frameworks sharing the same resources is not safe
Optimistic Vs Pessimistic Offers

- 2 frameworks sharing the same resources is not safe
- A chunk of resources is only offered to a single framework scheduler at a time.
Why is this a problem?

When a Framework receives resource offers, it has 2 options:

- Hold onto the offer forever in a state of indecision
- Make an immediate decision
Why is this a problem?

When a Framework receives resource offers, it has 2 options:

- Hold onto the offer forever in a state of indecision
- Make an immediate decision
Why is this a problem?

Under-utilization

If the framework holds the offer forever, those resources can’t be used.

... or eaten!
Why is this a problem?

Under-utilization

Can be hard to schedule large tasks
Why is this a problem?

Gaming the System

If it’s hard to schedule large tasks, frameworks might hold onto tons of offers until it can schedule its huge task.
Why is this a problem?

Gaming the System:

One could create many instances of a framework to trick Mesos to let it hoard more offers!
Workarounds / Solutions

- **--offer_timeout**  Set short timeouts to penalize slow frameworks

- **MESOS-1607**: Wait for optimistic offers!
  - Submit one offer to multiple frameworks, but rescind the offer when necessary.
  - Encourages more sophisticated allocation algorithms
Remainder of this talk...

1. Optimistic Vs Pessimistic Offers
2. DRF Algorithm and Framework Sorter
3. Missing APIs / Enhancements
DRF and Framework Sorter
DRF and Framework Sorter

Mesos Master must choose which Frameworks to give offers to first.
Mesos Master must choose which Frameworks to give offers to first.

In a pessimistic system, this is very important!
What is DRF?

“Dominant Resource Fairness” Algorithm
What is DRF?

“Dominant Resource Fairness” Algorithm

- A method for prioritizing which frameworks to give a resource offer to first.
What is DRF?

“Dominant Resource Fairness” Algorithm

We can represent a framework by how many resources it uses.
What is DRF?

“Dominant Resource Fairness” Algorithm

We can represent a framework by how many resources it uses.

For example:
- 30% of total RAM
- 12% of total CPU
What is DRF?

“Dominant Resource Fairness” Algorithm

Framework XYZ's Dominant Resource is the 30% RAM
How does DRF work?

“Dominant Resource Fairness” Algorithm

Identify all frameworks by their dominant resource
How does DRF work?

“Dominant Resource Fairness” Algorithm

Out of all frameworks (F1, F2 and F3),

F2 has the minimum dominant share of resources.
How does DRF work?

“Dominant Resource Fairness” Algorithm

DRF says that as long as resources are available,

Mesos should offer resources to F2 first, F3 second, and F1 last.
How does DRF work?

Weighted DRF

Per-framework weights, if defined, adjust the dominant share for each framework.
How does DRF work?

Weighted DRF

Per-framework weights, if defined, adjust the dominant share for each framework.

Weighting informs Mesos that it should generally prefer some Frameworks over others.
DRF is great if...
DRF is great if...

- All frameworks have work to do
DRF is great if...

- All frameworks have work to do
- A framework’s “hunger” for more resources does not change over its lifetime
DRF is great if...

- All frameworks have work to do
- A framework’s “hunger” for more resources does not change over its lifetime
- You know apriori that specific frameworks to use more or less resources
DRF is bad if...
DRF is bad if...

- Some frameworks don’t want any more tasks, while others do.
DRF is bad if...

- Some frameworks don’t want any more tasks, while others do.
- The framework's "hunger" for resources changes over its lifetime (perhaps based on queue size or pending web requests)
DRF Examples

Framework 1
1 task

Framework 2
6 tasks

Framework 3
1 task

Framework 4
30 tasks

Framework 5
1 task

Framework 6
50 tasks

Framework 4 always wants 30 tasks
DRF with weights is great **IF** these expected ratios never change.

Framework 4 always wants 30 tasks.
Sometimes frameworks don’t want to do work.
Sometimes frameworks don’t want to do work.

- DRF gives preference to the “0 tasks” frameworks.
- Framework 6 gets starved for resources!
Sometimes frameworks don’t want to do work.

- DRF gives preference to the “0 tasks” frameworks.
- Framework 6 gets starved for resources!
Real-world Examples of Bad DRF

Any Framework that declines usable offers suggests DRF isn’t working well

- Consumer Framework that consumes an occasionally empty queue
- Web Server Framework that sometimes doesn’t get a lot of requests
- Database Framework that doesn’t have a lot to do sometimes
Workarounds / Solutions

● Ensure all your frameworks always want more tasks
  ○ Can be very hard, perhaps impossible, to do.
  ○ ie. What if a framework just maintains N services?
  ○ Might encourage sloppy or inefficient frameworks.
Workarounds / Solutions

- Write your own allocation algorithm!
  - See Li Jin’s 11:50 talk, "Preemptive Task Scheduling in Mesos Framework"
  - Maybe other talks?
Workarounds / Solutions

- wait for optimistic offers to make this less of an issue
- allow frameworks to periodically restart themselves and define a different DRF weighting every time they restart
DRF Speculation

- A really good dynamic weighting algorithm would benefit by knowledge of the current distribution of weights by other frameworks across the system.
  - Frameworks could compete with each other based on this information
  - Makes Mesos more like a shared-state scheduler
Remainder of this talk...

1. Optimistic Vs Pessimistic Offers
2. DRF Algorithm and Framework Sorter
3. Missing APIs / Enhancements
The Disclaimer
These are my opinions

Not sure whether others will agree

If you have opinions too, let’s get beers tonight!
Missing APIs / Enhancements

- In my opinion, different framework sorter algorithms and even optimistic offers, will only take us so far.
Missing APIs / Enhancements

- Frameworks should more actively leverage statistics about resource utilization to inform Mesos master about how it should be allocated.
Missing APIs / Enhancements

- Frameworks should more actively leverage statistics about resource utilization to inform mesos master about how it should be allocated.
  - Frameworks know their resource needs better than the Master.
  - Some frameworks can make simple decisions
  - Others can be smart in how they wish to populate the grid
Missing APIs / Enhancements

● Frameworks should be able to tell mesos what they will want in the future (and how badly they want it)
  ○ Let the framework developer community play the game to “optimize this scheduling problem”

● The DRF algorithm, or hierarchical allocator in general, should leverage historical data.
For more about our story, check out this talk at 4:50!

Building A Machine Learning Platform to Predict User Behavior on Mesos - Jeremy Stanley,