CS162
Operating Systems and Systems Programming
Lecture 27

Special Topics: Apache Spark and Ray

December 2nd, 2021
Prof. Ion Stoica
http://cs162.eecs.Berkeley.edu
The Berkeley AMPLab

- January 2011 – 2017
  - 8 faculty
  - > 50 students
  - 3 software engineer team
- Organized for collaboration

AMPCamp (since 2012)

3 day retreats (twice a year)

400+ campers (100s companies)
A Short History

• Started at UC Berkeley in 2009

• Open Source: 2010

• Apache Project: 2013

• Databricks started: 2013

• Today: most popular big data project
2009: State-of-the-art in Big Data

Apache Hadoop
• Large scale, flexible data processing engine
• Batch computation (e.g., 10s minutes to hours)
• Open Source

Getting rapid industry traction:
• High profile users: Facebook, Twitter, Yahoo!, …
• Distributions: Cloudera, Hortonworks
• Many companies still in austerity mode
Input data and intermediate data stored on disk

Hadoop

Machine

stage

stage

Map

Shuffle

Reduce
2009: Berkeley

Lester
Hadoop: Sloooow…

- Machine Learning (ML) are iterative algorithms
- Each iteration a map-reduce job
- Between jobs, data is stored and read from the disk
- So each iteration requires to read/write lots of data from/to disk
2009: Application Trends

Iterative computations, e.g., Machine Learning
  • More and more people aiming to get insights from data

Interactive computations, e.g., ad-hoc analytics
  • SQL engines like Hive and Pig drove this trend
2009: Application Trends

Despite huge amounts of data, many working sets in big data clusters fit in memory
2009: Application Trends

<table>
<thead>
<tr>
<th>Memory (GB)</th>
<th>Facebook (% jobs)</th>
<th>Microsoft (% jobs)</th>
<th>Yahoo! (% jobs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>69</td>
<td>38</td>
<td>66</td>
</tr>
<tr>
<td>16</td>
<td>74</td>
<td>51</td>
<td>81</td>
</tr>
<tr>
<td>32</td>
<td><strong>96</strong></td>
<td><strong>82</strong></td>
<td><strong>97.5</strong></td>
</tr>
<tr>
<td>64</td>
<td>97</td>
<td>98</td>
<td>99.5</td>
</tr>
<tr>
<td>128</td>
<td>98.8</td>
<td>99.4</td>
<td>99.8</td>
</tr>
<tr>
<td>192</td>
<td>99.5</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>256</td>
<td>99.6</td>
<td>100</td>
<td>100</td>
</tr>
</tbody>
</table>

*G Ananthanarayanan, A. Ghodsi, S. Shenker, I. Stoica, "Disk- Locality in Datacenter Computing Considered Irrelevant", HotOS 2011*
## 2009: Application Trends

<table>
<thead>
<tr>
<th>Memory (GB)</th>
<th>Facebook (% jobs)</th>
<th>Microsoft (% jobs)</th>
<th>Yahoo! (% jobs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>69</td>
<td>38</td>
<td>66</td>
</tr>
<tr>
<td>16</td>
<td>74</td>
<td>51</td>
<td>81</td>
</tr>
<tr>
<td>32</td>
<td>96</td>
<td>82</td>
<td>97.5</td>
</tr>
<tr>
<td><strong>64</strong></td>
<td><strong>97</strong></td>
<td><strong>98</strong></td>
<td><strong>99.5</strong></td>
</tr>
<tr>
<td>128</td>
<td>98.8</td>
<td>99.4</td>
<td>99.8</td>
</tr>
<tr>
<td>192</td>
<td>99.5</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>256</td>
<td>99.6</td>
<td>100</td>
<td>100</td>
</tr>
</tbody>
</table>

What Is Spark?

- Parallel execution engine for big data processing
- **General**: efficient support for multiple workloads
- **Easy** to use: 2-5x less code than Hadoop MR
  - High level API’s in Python, Java, and Scala
- **Fast**: up to 100x faster than Hadoop MR
  - Can exploit in-memory when available
  - Low overhead scheduling, optimized engine
General

• Unifies *batch, interactive* workloads
General

- Unifies *batch, interactive, streaming* workloads
General

- Unifies **batch, interactive, streaming** workloads
- Easy to build sophisticated applications
  - Support iterative, graph-parallel algorithms
  - Powerful APIs in Scala, Python, Java, R

Spark Core

SparkSQL  Spark Streaming  MLlib  GraphX  SparkR
Easy to Write Code

WordCount in 3 lines of Spark

val f = sc.textFile(inputPath)
val w = f.flatMap(l => l.split(" ")).map(word => (word, 1)).cache()
w.reduceByKey(_ + _).saveAsText(outputPath)

WordCount in 50+ lines of Java MR
RDD: Resilient Distributed Datasets

• Collections of objects partition & distributed across a cluster
  – Stored in RAM or on Disk
  – Resilient to failures

• Operations
  – Transformations
  – Actions

• Execution model: similar to SIMD (Single Instruction Multiple Data)
Spark, as a BSP System

![Diagram of Spark as a BSP System]

- RDD
- tasks (processors)
- Shuffle
- RDD
- tasks (processors)

stage (super-step)

---

Lec 27.19

12/2/21

CS162 © UCB Fall 2021
Spark, as a BSP System

- all tasks in same stage impl. same operations,
- single-threaded, deterministic execution

**Immutable dataset**

**Barrier implicit by data dependency**

stage (super-step)  |  tasks (processors)  |  Shuffle  |  tasks (processors)  |  stage (super-step)

RDD  |  RDD
Operations on RDDs

• Transformations \( f(RDD) \rightarrow RDD \)
  - Lazy (not computed immediately)
  - E.g., “map”, “filter”, “groupBy”

• Actions:
  - Triggers computation
  - E.g. “count”, “collect”, “saveAsTextFile”
Spark for ML: Very fast

- Machine Learning (ML) are iterative algorithms
- Each iteration one or two Spark stages (not jobs)
- Between stages, data is passed through memory (not disk), so much faster!
<table>
<thead>
<tr>
<th>Rank</th>
<th>Team Name</th>
<th>Best Test Score</th>
<th>% Improvement</th>
<th>Best Submit Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>BellKor's Pragmatic Chaos</td>
<td>0.8567</td>
<td>10.06</td>
<td>2009-07-26 18:18:28</td>
</tr>
<tr>
<td>2</td>
<td>The Ensemble</td>
<td>0.8567</td>
<td>10.06</td>
<td>2009-07-26 18:38:22</td>
</tr>
<tr>
<td>3</td>
<td>Grand Prize Team</td>
<td>0.8582</td>
<td>9.90</td>
<td>2009-07-10 21:24:40</td>
</tr>
<tr>
<td>4</td>
<td>Opera Solutions and Vandelay United</td>
<td>0.8588</td>
<td>9.84</td>
<td>2009-07-10 01:12:31</td>
</tr>
<tr>
<td>5</td>
<td>Vandelay Industries !</td>
<td>0.8591</td>
<td>9.81</td>
<td>2009-07-10 00:32:20</td>
</tr>
<tr>
<td>6</td>
<td>PragmaticTheory</td>
<td>0.8594</td>
<td>9.77</td>
<td>2009-06-24 12:06:56</td>
</tr>
<tr>
<td>7</td>
<td>BellKor in BigChaos</td>
<td>0.8601</td>
<td>9.70</td>
<td>2009-05-13 08:14:09</td>
</tr>
<tr>
<td>8</td>
<td>Dace</td>
<td>0.8612</td>
<td>9.59</td>
<td>2009-07-24 17:18:43</td>
</tr>
</tbody>
</table>

The team "BellKor's Pragmatic Chaos" is tied for best score and submitted 20 minutes late.
Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns
Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns.
Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```python
lines = spark.textFile("hdfs://...")
```
Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```python
lines = spark.textFile("hdfs://...")
```
Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```python
lines = spark.textFile("hdfs://...")
errors = lines.filter(lambda s: s.startswith("ERROR"))
```
Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```python
lines = spark.textFile("hdfs://...")
errors = lines.filter(lambda s: s.startswith("ERROR"))
```
Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```python
lines = spark.textFile("hdfs://...")
errors = lines.filter(lambda s: s.startswith("ERROR"))
messages = errors.map(lambda s: s.split("\t")[2])
messages.cache()

messages.filter(lambda s: "mysql" in s).count()
```
Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```python
lines = spark.textFile("hdfs://...")
errors = lines.filter(lambda s: s.startswith("ERROR"))
messages = errors.map(lambda s: s.split("\t")[2])
messages.cache()

messages.filter(lambda s: "mysql" in s).count()
```
Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```python
lines = spark.textFile("hdfs://...")
errors = lines.filter(lambda s: s.startswith("ERROR"))
messages = errors.map(lambda s: s.split("\t") [2])
messages.cache()

messages.filter(lambda s: "mysql" in s).count()
```
Load error messages from a log into memory, then interactively search for various patterns

```
lines = spark.textFile("hdfs://...")
errors = lines.filter(lambda s: s.startswith("ERROR"))
messages = errors.map(lambda s: s.split("\t") [2])
messages.cache()

messages.filter(lambda s: "mysql" in s).count()
```
Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns.

```python
lines = spark.textFile("hdfs://...")
errors = lines.filter(lambda s: s.startswith("ERROR"))
messages = errors.map(lambda s: s.split("\t")[2])
messages.cache()

messages.filter(lambda s: "mysql" in s).count()
```
Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```python
lines = spark.textFile("hdfs://...")
errors = lines.filter(lambda s: s.startswith("ERROR"))
messages = errors.map(lambda s: s.split("\t") [2])
messages.cache()

messages.filter(lambda s: "mysql" in s).count()
```
Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```python
lines = spark.textFile("hdfs://...")
errors = lines.filter(lambda s: s.startswith("ERROR"))
messages = errors.map(lambda s: s.split("\t")[2])
messages.cache()

messages.filter(lambda s: "mysql" in s).count()
```
Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```python
lines = spark.textFile("hdfs://...")
errors = lines.filter(lambda s: s.startswith("ERROR"))
messages = errors.map(lambda s: s.split("\t")[2])
messages.cache()

messages.filter(lambda s: "mysql" in s).count()
messages.filter(lambda s: "php" in s).count()
```
Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```python
lines = spark.textFile("hdfs://...")
errors = lines.filter(lambda s: s.startswith("ERROR"))
messages = errors.map(lambda s: s.split("\t") [2])
messages.cache()

messages.filter(lambda s: "mysql" in s).count()
messages.filter(lambda s: "php" in s).count()
```
Load error messages from a log into memory, then interactively search for various patterns

```python
lines = spark.textFile("hdfs://...")
errors = lines.filter(lambda s: s.startswith("ERROR"))
messages = errors.map(lambda s: s.split("\t") [2])
messages.cache()

messages.filter(lambda s: "mysql" in s).count()
messages.filter(lambda s: "php" in s).count()
```
Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```python
lines = spark.textFile("hdfs://...")
errors = lines.filter(lambda s: s.startswith("ERROR"))
messages = errors.map(lambda s: s.split("\t")[2])
messages.cache()

messages.filter(lambda s: "mysql" in s).count()
messages.filter(lambda s: "php" in s).count()
```
Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```python
lines = spark.textFile("hdfs://...")
errors = lines.filter(lambda s: s.startswith("ERROR"))
messages = errors.map(lambda s: s.split("\t")[2])
messages.cache()

messages.filter(lambda s: "mysql" in s).count()
messages.filter(lambda s: "php" in s).count()
```

Cache your data ➔ Faster Results

*Full-text search of Wikipedia*
- 60GB on 20 EC2 machines
- 0.5 sec from mem vs. 20s for on-disk
Language Support

### Standalone Programs
- Python, Scala, & Java

### Interactive Shells
- Python & Scala

### Performance
- Java & Scala are faster due to static typing
- …but Python is often fine

---

**Python**

```python
def count_error_lines(sc):
    lines = sc.textFile(...)  
    return lines.filter(lambda s: "ERROR" in s).count()
```

**Scala**

```scala
val count_error_lines = (sc: SparkSession) => 
  sc.textFile(...)  
  .filter(_.contains("ERROR"))  
  .count
```

**Java**

```java
public static long countErrorLines(JavaRDD<String> lines) {
    return lines.filter(new Function<String, Boolean>() {
        @Override
        public Boolean call(String s) {
            return s.contains("error");
        }
    }).count();
```
Expressive API

- map
- filter
- groupBy
- sort
- union
- join
- leftOuterJoin
- rightOuterJoin
- reduce
- count
- fold
- reduceByKey
- groupByKey
- cogroup
- cross
- zip
- sample
- sample
- take
- first
- partitionBy
- mapWith
- pipe
- save
- ...
Fault Recovery: Design Alternatives

• Replication:
  – Slow: need to write data over network
  – Memory inefficient

• Backup on persistent storage:
  – Persistent storage still (much) slower than memory
  – Still need to go over network to protect against machine failures

• Spark choice:
  – Lineage: track seq. of operations to efficiently reconstruct lost RRD partitions
  – Enabled by determinist execution and data immutability
Fault Recovery Example

• Two-partition RDD $A = \{A_1, A_2\}$ stored on disk
  1) filter and cache $\rightarrow$ RDD $B$
  2) join $\rightarrow$ RDD $C$
  3) aggregate $\rightarrow$ RDD $D$
Fault Recovery Example

- $C_1$ lost due to node failure before reduce finishes
Fault Recovery Example

- $C_1$ lost due to node failure before reduce finishes
- Reconstruct $C_1$, eventually, on different node
Fault Recovery Results

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Iteration time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>119</td>
</tr>
<tr>
<td>2</td>
<td>57</td>
</tr>
<tr>
<td>3</td>
<td>56</td>
</tr>
<tr>
<td>4</td>
<td>58</td>
</tr>
<tr>
<td>5</td>
<td>58</td>
</tr>
<tr>
<td>6</td>
<td>81</td>
</tr>
<tr>
<td>7</td>
<td>57</td>
</tr>
<tr>
<td>8</td>
<td>59</td>
</tr>
<tr>
<td>9</td>
<td>57</td>
</tr>
<tr>
<td>10</td>
<td>59</td>
</tr>
</tbody>
</table>

Failure happens at Iteration 6.
Announcements

Congrats for taking the last midterm!

Project 3 Party on Sunday 12/5 from 10AM-2PM at Woz Lounge

Discussions converted to office hours this week

Office hours will continue into dead week (with possibly modified schedule)

Homework 6 due Friday 12/3 11:59 PM

Project 3 due Wednesday 12/8 11:59 PM
Ray
Trends

AI demands exploding

The end of Moore’s Law
Compute demands, 2012-2019 (AI)

35x every 18 months
Compute demands, 2012-2020 (AI)

No sign of slow down...

Compute demands, 2012-2020 (AI)

No sign of slow down...

Trends

AI demands exploding

The end of Moore’s Law
The end of Moore’s Law

From $2x$ every 18 months to $1.05x$ every 18 months.
Growing gap between demand and supply

Moore’s Law (2x every 18 months)

35x every 18 months

https://openai.com/blog/ai-and-compute/
Specialized hardware is not enough

Moore's Law (2x every 18 months)

35x every 18 months
Example: In-app promotion

A **real** use case:

- Recommend services, products
- Largest fintech company in the world
Example: In-app promotion
Example: In-app promotion

Two questions:
- How fast can we update the model?
- How much does it matter?
Example: In-app promotion

Model updated every 1 day
- + 5% CTR (Click Through Rate)

Model updated every 1 hour (state-of-the-art solution)

Want to get lower, but how?
Example: In-app promotion

Previous solution: integrate best-of-breed frameworks

Challenges: end-to-end delay, development, management cost
Even more complex patterns!
Even more complex patterns!

Simulation

Data Ingestion & Featurization

Training Hyperparam. Tuning

Serving A/B testing

Reinforcement Learning
Today’s ML Ecosystem

Building an MP app/ workflow → stitch together a set of hodge-podge libraries (systems)

Hard to
- Hard to build: different APIs
- Hard to deploy: manage different distributed system
- Slow: high overhead to move data from one system to another
Ray

Universal framework for distributed computing

Libraries
- Training
- Model Serving
- Hyperparam. Tuning
- Streaming
- Simulation
- Featurization
Ray: a short history

2016: Started as a class project
  - Initial goal: scale distributed deep neural network training and reinforcement learning
2017: RLlib and Ray Tune released
2019: Anyscale founded (company behind Ray)
2020: Ray v1.0 release; Ray Serve released
### Minimalist API

#### ray.init()
- Initialize Ray context.

#### @ray.remote
- Function or class decorator specifying that the function will be executed as a task or the class as an actor in a different process.

#### .remote
- Postfix to every remote function, remote class declaration, or invocation of a remote class method. Remote operations are *asynchronous*.

#### ray.put()
- Store object in object store, and return its ID. This ID can be used to pass object as an argument to any remote function or method call. This is a *synchronous* operation.

#### ray.get()
- Return an object or list of objects from the object ID or list of object IDs. This is a *synchronous* (i.e., blocking) operation.

#### ray.wait()
- From a list of object IDs returns (1) the list of IDs of the objects that are ready, and (2) the list of IDs of the objects that are not ready yet.
Ray in a nutshell

Support both **stateless** and **stateful** computations
- Functions → Tasks (stateless)
- Classes → Actors (stateful)

**Future-like abstraction**: enables parallelism

**In-memory distributed object store**: enable passing args/results by reference

**Extension of existing languages**, rather than a new language
Python example

```python
def f(x):
    # compute...
    return r
```
Python example

def f(x):
    # compute...
    return r

x = f(a)  # 1 second
Python example

```python
def f(x):
    # compute...
    return r

x = f(a)  # 1 second
y = f(b)  # 1 second
```
@ray.remote
def f(x):
    # compute...
    return r
def f(x):
    # compute...
    return r

x_id = f.remote(a)
y_id = f.remote(b)
ray.get([x_id, y_id])
Ray: Function ➔ Task

@ray.remote
def f(x):
    # compute...
    return r

x_id = f.remote(a)
y_id = f.remote(b)
ray.get([[x_id, y_id]])
```python
@ray.remote
def f(x):
    # compute...
    return r

x_id = f.remote(a)
y_id = f.remote(b)
ray.get([x_id, y_id])
```
@ray.remote
def f(x):
    # compute...
    return r

x_id = f.remote(a)
y_id = f.remote(b)
ray.get([x_id, y_id])
@ray.remote
def f(x):
    # compute...
    return r

x_id = f.remote(a)

y_id = f.remote(b)

ray.get([x_id, y_id])
@ray.remote
def f(x):
    # compute...
    return r

x_id = f.remote(a)
y_id = f.remote(b)
ray.get([x_id, y_id])
@ray.remote
def f(x):
    # compute...
    return r

x_id = f.remote(a)
y_id = f.remote(b)
ray.get([x_id, y_id])
Ray: Function ➔ Task

@ray.remote
def f(x):
    # compute...
    return r

x_id = f.remote(a)
y_id = f.remote(b)
ray.get([x_id, y_id])
class Counter(object):
    def __init__(self):
        self.value = 0
    def inc(self):
        self.value += 1
        return self.value

c = Counter()
c.inc()
c.inc()
c.inc()
```python
class Counter(object):
    def __init__(self):
        self.value = 0
    def inc(self):
        self.value += 1
        return self.value

c = Counter()
c.inc()
c.inc()
```

```python
@ray.remote
class Counter(object):
    def __init__(self):
        self.value = 0
    def inc(self):
        self.value += 1
        return self.value

c = Counter.remote()
c.inc.remote()
id4 = c.inc.remote()
id5 = c.inc.remote()
```
Python Class

```python
class Counter(object):
    def __init__(self):
        self.value = 0
    def inc(self):
        self.value += 1
        return self.value

C = Counter()
C.inc()
C.inc()
```

Ray: Class → Actor

```python
@ray.remote(num_cpus=2, num_gpus=1)
class Counter(object):
    def __init__(self):
        self.value = 0
    def inc(self):
        self.value += 1
        return self.value

c = Counter()
c.inc.remote()
id4 = c.inc.remote()
id5 = c.inc.remote()
```
can specify resource demands; support heterogeneous hardware
Distributed object store

Distributed object store

@ray.remote
def f():
    # compute...
    return x

@ray.remote
def g(a):
    # compute...
    return y

id_x = f.remote()
id_y = g.remote(id_x)
Distributed object store

@ray.remote
def f():
    # compute...
    return x

@ray.remote
def g(a):
    # compute...
    return y

id_x = f.remote()
id_y = g.remote(id_x)

Only x’s id (id_x) is returned, not X’s value
@ray.remote
def f():
    # compute...
    return x

@ray.remote
def g(a):
    # compute...
    return y

id_x = f.remote()

id_y = g.remote(id_x)
Distributed object store

```python
@ray.remote
def f():
    # compute...
    return x

@ray.remote
def g(a):
    # compute...
    return y

id_x = f.remote()
id_y = g.remote(id_x)
```

node 1

node 2

node 3
Distributed object store

@ray.remote
def f():
    # compute...
    return x

@ray.remote
def g(a):
    # compute...
    return y

id_x = f().remote()
id_y = g.remote(id_x)

Without object store x would have been transferred **twice**: node 2 → node 1 and node 1 → node 3!
Example: In-app promotion

- **Data Ingestion & Featurization**
  - Spark
  - Kafka
  - Flink

- **Training**
  - PyTorch
  - Horovod

- **Serving**
  - Seldon

- **Model updated every 1 day**
- **+ 5% CTR**
- **Model updated every 1 hour** (using state-of-the-art solution)

12/2/21
Ray: unified platform for distributed apps

Model updated every 1 day

- + 5% CTR

Model updated every 1 hour (using state-of-the-art solution)

- + 1% CTR

Model updated every 5 min using Ray

Data Ingestion & Featurization

5 min joining

Serving

ACME
Ray Architecture

Distributed object store
  • Immutable objects
Ray Architecture

Distributed object store
  • Immutable objects

Distributed scheduler
Ray Architecture

**Distributed object store**
- Immutable objects

**Distributed scheduler**
- Direct calls
Ray Architecture

Distributed object store
  • Immutable objects

Distributed scheduler
  • Direct calls

Central control store (GCS)
  • Stateless components
Scalability

Decentralized scheduler

Sharded GCS

Any worker can submit tasks
  • Driver not a bottleneck
Rich Distributed Library Ecosystem

Native Libraries

- rllib
- ray
- rayoptuna
- rayserve

3rd Party Libraries

- spaCy
- Optuna
- Azure Machine Learning
- Hyperopt
- ModelArts
- Dask
- Modin
- Analytics Zoo
- Unity
- MARS
- Classy Vision

RAY - universal framework for distributed computing

- AWS
- Microsoft Azure
- Google Cloud
- Kubernetes
- Laptop
Best-in-class adoption: GitHub stars

Spark
Kafka
Ray
Flink
Kubeflow
MLFlow
Dask

17.9K
Best-in-class adoption: Contributors (550+)

Significant community contributions
We built **streaming, serving and iterative** calculations based on Ray. ... The ... solution based on Ray has been running stably in Ant Group, successfully supporting large-scale activities such as Double 11 and Double 12....
By leveraging Ray, we can **combine the preprocessing, distributed training, and hyperparameter search** all within a single job running a **single training script**.
Winning America’s Cup

Built a “Boat Designer” using RLlib
- Test 10x as many designs

“AI Sailor” using RLlib
- Outperformed human sailors

Won America Cup 2021
Good luck with the end of the semester!

Happy Holidays!