Systems plus ML:
When the sum is greater than its parts

Ion Stoica
UC Berkeley, Director of RISELab
November 5, 2019
Studies the design of real-time, intelligent, secure, and explainable algorithms and systems.
Studies the design of real-time, intelligent, secure, and explainable algorithms and systems that can respond in seconds/milliseconds using sophisticated models/algos. Protecting the data and system auditable and verifiable.
Studies the design of **real-time, intelligent, secure, and explainable** algorithms and systems.
Positive feedback loop to accelerate ML and Systems
Systems

- Ray

ML

Systems

- Deep queries
- Neurocuts
- Autopandas
Trends

Apps becoming distributed

Apps becoming more complex
Apps becoming **distributed**

Moore’s law & Denard scaling have ended!

Training largest models: doubles every 3.5 months (**35x** over 18 months)!

No choice but to distribute apps
Apps becoming more complex

Virtually all apps will become AI centric
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!

Data Ingestion & Featurization

Training
Hyperparam. Tuning

Serving
A/B testing
Even more complex patterns!

Data Ingestion & Featurization

Training Hyperparam. Tuning

Serving A/B testing

Reinforcement Learning
Today’s ML Ecosystem

<table>
<thead>
<tr>
<th>Distributed systems</th>
<th>Distributed systems</th>
<th>Distributed systems</th>
<th>Distributed systems</th>
<th>Distributed systems</th>
<th>Distributed systems</th>
</tr>
</thead>
<tbody>
<tr>
<td>Training</td>
<td>Model Serving</td>
<td>Hyperparam. Tuning</td>
<td>Streaming</td>
<td>Simulation</td>
<td>Featurization</td>
</tr>
</tbody>
</table>

- PyTorch
- TensorFlow
- SIGOPT
- Apache Spark
- Kafka
- Flink
- MPI
- Orleans
- Apache Hadoop
Ray: A Distributed Framework for Emerging AI Applications, Philipp Moritz et al, OSDI 2018

Libraries

- Training
- Model Serving
- Hyperparam. Tuning
- Streaming
- Simulation
- Featurization

General-purpose distributed computing framework for Python (and Java)
Function

```python
def read_array(file):
    # read ndarray “a”
    # from “file”
    return a

def add(a, b):
    return np.add(a, b)

a = read_array(file1)
b = read_array(file2)
sum = add(a, b)
```

Object

```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()
```
Function ➔ Task

```python
@ray.remote
def read_array(file):
    # read ndarray “a”
    # from “file”
    return a

@ray.remote
def add(a, b):
    return np.add(a, b)

a = read_array(file1)
b = read_array(file2)
sum = add(a, b)
```

Object ➔ Actor

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

c = Counter()
c.inc()
c.inc()
```
Function → Task

```python
@ray.remote
def read_array(file):
    # read ndarray "a"
    # from "file"
    return a

@ray.remote
def add(a, b):
    return np.add(a, b)
```

```python
id1 = read_array.remote(file1)
id2 = read_array.remote(file2)
id = add.remote(id1, id2)
sum = ray.get(id)
```

Object → Actor

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

```python
c = Counter.remote()
id4 = c.inc.remote()
id5 = c.inc.remote()
```
Task API

```python
def read_array(file):
    # read ndarray “a”
    # from “file”
    return a

def add(a, b):
    return np.add(a, b)

id1 = read_array.remote(file1)
id2 = read_array.remote(file2)
id = add.remote(id1, id2)
sum = ray.get(id)
```

- Blue variables are Object IDs
- Similar to futures

Return id1 immediately, before read_array() finishes
Task API

```python
@ray.remote
def read_array(file):
    # read ndarray “a”
    # from “file”
    return a

@ray.remote
def add(a, b):
    return np.add(a, b)

id1 = read_array.remote(file1)
id2 = read_array.remote(file2)
id = add.remote(id1, id2)
sum = ray.get(id)
```

- Blue variables are **Object IDs**
- Similar to futures

Dynamic task graph: build at runtime
Task API

```python
@ray.remote
def read_array(file):
    # read ndarray “a”
    # from “file”
    return a

@ray.remote
def add(a, b):
    return np.add(a, b)

id1 = read_array.remote(file1)
id2 = read_array.remote(file2)
id = add.remote(id1, id2)
sum = ray.get(id)
```

- Blue variables are **Object IDs**
- Similar to futures

Every task scheduled, but not finished yet

Node 1
- `file1`
- `read_array`
- `id1`

Node 2
- `file2`
- `read_array`
- `id2`

Node 3
- `add`
- `id`
Task API

```python
@ray.remote
def read_array(file):
    # read ndarray "a"
    # from "file"
    return a

@ray.remote
def add(a, b):
    return np.add(a, b)

id1 = read_array.remote(file1)
id2 = read_array.remote(file2)
id = add.remote(id1, id2)
sum = ray.get(id)
```

- Blue variables are **Object IDs**
- Similar to futures

Node 1
- file1
  - read_array
  - id1

Node 2
- file2
  - read_array
  - id2

Node 3
- add
  - id

- ray.get() block until result available
 Task API

```python
@ray.remote
def read_array(file):
    # read ndarray "a"
    # from "file"
    return a

@ray.remote
def add(a, b):
    return np.add(a, b)

id1 = read_array.remote(file1)
id2 = read_array.remote(file2)
id = add.remote(id1, id2)
sum = ray.get(id)
```

- Blue variables are **Object IDs**
- Similar to futures

Task graph executed to compute `sum`
Example: In-app promotion

- Model updated every **1 day**
- + 5% CTR
- Model updated every **1 hour** (using state-of-the-art solution)
- ?
Ray: unified platform for distributed apps

- Model updated every 1 day
- Model updated every 1 hour (using state-of-the-art solution)
- Model updated every 5 min using Ray

ACME

Data Ingestion & Featurization

Ray

Serving
Ray Architecture

In-memory obj. store
- Immutable objects

Serialization using Apache Arrow
Ray Architecture

In-memory obj. store
  • Immutable objects

Distributed scheduler
Ray Architecture

In-memory obj. store
• Immutable objects

Distributed scheduler
Ray Architecture

In-memory obj. store
- Immutable objects

Distributed scheduler

Central control store (GCS)
- Stateless components
Ray Architecture

In-memory obj. store
- Immutable objects

Distributed scheduler

Central control store (GCS)
- Stateless components
Scalability

Decentralized scheduler

Sharded GCS

Any worker can submit tasks
  • Driver not a bottleneck
Scalability

Decentralized scheduler

Sharded GCS

Any worker can submit tasks
  • Driver not a bottleneck

Actors arguments sent by reference instead inline
  • Avoid unnecessary copies
  • Can optimize transfers (e.g., parallel transfers, multicast)
Fault Tolerance

Lineage based
- Replay computation to reconstruct lost objects

Classic solution
- Store lineage before invocation
- Insert delay on critical path
Fault Tolerance

Lineage based
• Replay computation to reconstruct lost objects

Classic solution
• Store lineage before invocation
• Insert delay on critical path

Lineage stash*
• Store lineage in task call and flush it

**"Lineage Stash: Fault Tolerance Off the Critical Path”, Stephanie Wang et al, SOSP ’19
Scalability & Performance

Latency of local task execution: \(~300\) us
Latency of remote task execution: \(~1\) ms

1.8 million tasks/sec
Ray vs specialized systems

**Serving**
- Ray: ~7000 queries/sec
- Clipper: ~4000 queries/sec

**Training**
- Ray + TF: ~6000 iterations/sec
- Horovod + TF: ~5000 iterations/sec

**Simulation**
- Ray: ~4500 steps/sec (x1000)
- MPI: ~1500 steps/sec (x1000)

*Match performance of specialized systems*
Ongoing work

Flexible scheduling policies
• Need to support conflicting policies, e.g., affinity, antiaffinity, locality, gang scheduling, …

Improved garbage collection for object store
• Currently LRU, but not "good enough"
• Ideally, global reference counting
General-purpose distributed computing framework for Python (and Java)
Growing adoption

200 contributors from 40+ companies

Sold out tutorials at O’Reilly AI

Included in AWS Sage Maker
Systems

- Ray
- Deep queries
- Neurocuts
- Autopandas
• Ray
• Deep queries
• Neurocuts
• Autopandanas
“Classic” DL/RL apps

Speech recognition
Video recognition
Language translation

Human tasks: 100% accuracy not expected
Systems problems

Program synthesis
Mortgage decisions
Robotic surgery

Must ensure correctness and explainability!
Three approaches

- **Correct by construction**: Build a solution that is provably correct.
- **Iterative optimization**: Sequence of transformations w/o impacting correctness.
- **Generate & Verify**: Generate solutions whose correctness can be verified.
Database joins

Calculate tax owned by “Manager I” employees

```
SELECT SUM(sal.salary*tax.rate)
FROM emp, sal, tax
WHERE emp.position = sal.position AND
tax.country = sal.country AND
emp.position = 'Manager I'
```
Join Optimization

\[
\text{SELECT SUM(sal.salary*tax.rate)}
\]
\[
\text{FROM emp, sal, tax}
\]
\[
\text{WHERE emp.position = sal.position AND}
\]
\[
\text{tax.country = sal.country AND}
\]
\[
\text{emp.position = 'Manager I'}
\]

In what order do you perform joins?

[Diagram showing two join trees with emp and sal on the left and tax on the right, with the cost of the left tree being 1200 and the right tree being 120.]
Join Optimization

```
SELECT SUM(sal.salary*tax.rate)
FROM emp, sal, tax
WHERE emp.position = sal.position AND
tax.country = sal.country AND
emp.position = 'Manager I'
```

In what order do you perform joins?

Exponential in number of queries!
40 years of heuristics!

**Left-deep**: maximize index usage

**Right-deep**: maximize hash table re-use

**Zig-zag**: union of LD and RD

**Greedy**: exploit linear cost models

**IK-KBZ**: exploit Star Schemas

**GEQO**: genetic algorithms in Postgres

Can ML replace programmed heuristics with efficient and data-driven strategies?
Reinforcement Learning (RL)

Agent continually learning by interacting with env. Compute policy (i.e., state $\rightarrow$ action) to maximize reward
Deep Query*: Join plans with Deep RL

Why reinforcement learning (RL)?

- Natural formulation
  - **State**: set of tables joined so far
  - **Action**: table to join next
  - **Reward**: negative of estimated cost
- Use Deep Q-Learning

Deep Query: Build join plans with Deep RL

Why reinforcement learning (RL)?

• Natural formulation
Deep Query: Build join plans with Deep RL

Why reinforcement learning (RL)?

**Correct by construction:** the "join" operation is commutative and associative so result’s correctness is guaranteed once we join all tables
Deep Query: Properties

Generalize to unseen queries

Adapt to workload and hardware characteristics

Efficient planning (order of magnitude faster)
113 queries, IMDb dataset, 21 tables
4–17 way joins (avg ~8 relations per query)
Cost models:
- Model 1: Index Mostly
- Model 2: Hybrid Hash
113 queries, IMDb dataset, 21 tables
4–17 way joins (avg ~8 relations per query)
Cost models:
  • Model 1: Index Mostly
  • Model 2: Hybrid Hash
Systems ML
• Ray

Systems ML
• Deep queries
• **Neurocuts**
• Autopandas
Packet Classification

Fundamental problem in networking
• Building block for access control, QoS, defense against attacks
The problem

Similar to point-location in a hypercube

Hard time-space tradeoff:

- $O(\log N)$ time and $O(N^d)$ space
- $O(\log^{d-1} N)$ time and $O(N)$ space

$N$: # of rules; $d$: # of attributes

- In our case: $N \approx 100K$, $d = 5$

But harder: rules overlap and have priorities
20+ years of work

Hardware
  • Expensive and power hungry $\rightarrow$ prohibitive for large classifiers

Software
  • Build a multi-dimensional "decision tree" – really a k-d index
  • HiCuts ('99), HyperCuts ('03), EffiCuts ('10), CutSplit ('15)
    – All rely on hand-tuned heuristics, which are brittle and not optimal
Two approaches

1. End-to-end solution
Two approaches

1. End-to-end solution

2. Build classification tree
NeuroCuts*: Building decision trees with Deep RL

*”Neural Packet Classification”, Eric Liang et al, SIGOMM 2019
NeuroCuts: Building decision trees with Deep RL

Why reinforcement learning (RL)?
- Simulation is reality: build a tree in simulation → deploy to prod.
- Naive MDP (Markov Decision Process) formulation

Iterative optimization: start from a single-node and iteratively refine (build) the tree; an iteration doesn’t affect three correctness

\[
\text{Reward} = - \text{depth[tree]} - \alpha \times \text{size[tree]}
\]
Results

Classification time: median 18% faster than state-of-the-art

3x better either in space or time than any previous solution
Systems
• Ray

Systems ML
• Deep queries
• Neurocuts
• Autopandas
API Explosion!
How to cope? *StackOverflow*

How do I turn this:

<table>
<thead>
<tr>
<th></th>
<th>weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>kg</td>
<td>lbs</td>
</tr>
<tr>
<td>cat</td>
<td>1 2</td>
</tr>
<tr>
<td>dog</td>
<td>2 4</td>
</tr>
</tbody>
</table>

into this:

<table>
<thead>
<tr>
<th></th>
<th>weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>cat</td>
<td>kg 1</td>
</tr>
<tr>
<td></td>
<td>lbs 2</td>
</tr>
<tr>
<td>dog</td>
<td>kg 2</td>
</tr>
<tr>
<td></td>
<td>lbs 4</td>
</tr>
</tbody>
</table>

in pandas?
Problems with *StackOverflow*

How do I turn this:

<table>
<thead>
<tr>
<th></th>
<th>weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>cat kg</td>
<td>1</td>
</tr>
<tr>
<td>cat lbs</td>
<td>2</td>
</tr>
<tr>
<td>dog kg</td>
<td>2</td>
</tr>
<tr>
<td>dog lbs</td>
<td>4</td>
</tr>
</tbody>
</table>

into this:

<table>
<thead>
<tr>
<th></th>
<th>weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>cat kg</td>
<td></td>
</tr>
<tr>
<td>cat lbs</td>
<td>2</td>
</tr>
<tr>
<td>dog kg</td>
<td>2</td>
</tr>
<tr>
<td>dog lbs</td>
<td>4</td>
</tr>
</tbody>
</table>

in pandas?

**Inefficient Solutions**

Well, you need to start by building the index `pd.muliIndex(...)`. 
Problems with *StackOverflow*

How do I turn this:

<table>
<thead>
<tr>
<th>weight</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>kg</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>lbs</td>
<td>2</td>
<td>4</td>
</tr>
</tbody>
</table>

into this:

<table>
<thead>
<tr>
<th>weight</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>kg</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>lbs</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

in pandas?

**Inefficient Solutions**

4 days later!

Just use the stack function
Goal: StackOverflow for APIs via Program Synthesis

How do I turn this:

<table>
<thead>
<tr>
<th></th>
<th>weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>kg</td>
<td>lbs</td>
</tr>
<tr>
<td>cat</td>
<td>1</td>
</tr>
<tr>
<td>dog</td>
<td>2</td>
</tr>
</tbody>
</table>

into this:

<table>
<thead>
<tr>
<th></th>
<th>weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>kg</td>
<td>lbs</td>
</tr>
<tr>
<td>cat</td>
<td>1</td>
</tr>
<tr>
<td>lbs</td>
<td>2</td>
</tr>
<tr>
<td>dog</td>
<td>2</td>
</tr>
<tr>
<td>lbs</td>
<td>4</td>
</tr>
</tbody>
</table>

in pandas?

output = input.stack(
    level=[1],
    dropna=True
)
Target API: **pandas** (DataFrame transformations)

Premier library for data scientists

Avg. number of single-function programs: $10^{17}$ 😱
Autopandas*

Input:

<table>
<thead>
<tr>
<th></th>
<th>weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>cat</td>
<td>kg</td>
</tr>
<tr>
<td></td>
<td>lbs</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>dog</td>
<td>kg</td>
</tr>
<tr>
<td></td>
<td>lbs</td>
</tr>
</tbody>
</table>

Output:

<table>
<thead>
<tr>
<th></th>
<th>weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>cat</td>
<td>kg</td>
</tr>
<tr>
<td></td>
<td>lbs</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>dog</td>
<td>kg</td>
</tr>
<tr>
<td></td>
<td>lbs</td>
</tr>
</tbody>
</table>

Generate program

Check if matches input-output

Yes

Output program

No

“AutoPandas: Neural-Backed Generators for Program Synthesis”, Rohan Bavishi et al, OOPSLA 2019
Predict program

input:

<table>
<thead>
<tr>
<th></th>
<th>weight</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>cat</td>
<td>kg</td>
<td>lbs</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>dog</td>
<td>kg</td>
<td>lbs</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>4</td>
</tr>
</tbody>
</table>

output:

<table>
<thead>
<tr>
<th></th>
<th>weight</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>cat</td>
<td>kg</td>
<td>lbs</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>dog</td>
<td>kg</td>
<td>lbs</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>4</td>
</tr>
</tbody>
</table>

Generate program

check if matches input-output

Huge search space! 
(10^{17} for one function)

Graph Neural Networks*

*M. Allamanis, M. Brockschmidt, and M. Khademi, Learning to represent programs with graphs, ICLR 2018
def generate_program(input, output):
    fn_seq = OrderedSequence(functions)
    for fn in fn_seq:
        if fn == pivot:
            arg_col = Select(input.cols)
            arg_idx = Select(input.cols - {arg_col})
            fn.add_args(arg_col, arg_idx)
        elif fn == drop:
            arg_ax = Select({0,1})
            arg_lbl = Select(input.rows) if arg_ax else Select(input.cols)
            fn.add_args(arg_ax, arg_lbl)
        # <omitted code...>
    return fn_seq

Neural-backed program generators

<table>
<thead>
<tr>
<th>input:</th>
<th>weight</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>kg</td>
</tr>
<tr>
<td>cat</td>
<td>1</td>
</tr>
<tr>
<td>dog</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>output:</th>
<th>weight</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>kg</td>
</tr>
<tr>
<td>cat</td>
<td>1</td>
</tr>
<tr>
<td>dog</td>
<td>2</td>
</tr>
</tbody>
</table>
Neural-backed program generators

```python
def generate_program(input, output):
    fn_seq = OrderedSequence(functions)
    for fn in fn_seq:
        if fn == pivot:
            arg_col = Select(input.cols)
            arg_idx = Select(input.cols - {arg_col})
            fn.add_args(arg_col, arg_idx)
        elif fn == drop:
            arg_ax = Select({0,1})
            arg_lbl = Select(input.rows) if arg_ax else Select(input.cols)
            fn.add_args(arg_ax, arg_lbl)
        elif ...
    return fn_seq
```

Describes search space

Can be arbitrary Python!
Neural-backed program generators

```python
def generate_program(input, output):
    fn_seq = OrderedSequence(functions)
    for fn in fn_seq:
        if fn == pivot:
            arg_col = Select(input.cols)
            arg_idx = Select(input.cols - {arg_col})
            fn.add_args(arg_col, arg_idx)
        elif fn == drop:
            arg_ax = Select({0,1})
            arg_lbl = Select(input.rows)
            if arg_ax
                fn.add_args(arg_ax, arg_lbl)
            else:
                fn.add_args(arg_ax, input.cols)
        elif ...
    return fn_seq
```

For one function reduce search space by $10^{12}$
(from $10^{17}$ to $10^{5}$)
Generate and verify: repeatedly generate solutions (e.g., programs) and verify these solutions until we get a correct solution.
- 70 benchmarks from StackOverflow, “Python for Data Analysis”, “Data School” videos
- ~95% of code snippets in StackOverflow have length <= 3 functions
Summary

**ML needs systems**: scale algorithms, easy to program

**Systems need ML**: improve state-of-the-art heuristics

Putting the two together → explosive growth

Systems → better ML → better Systems → …