The Ray Beam Runner Project

A Vision for Unified Batch, Streaming, and ML

By Patrick Ames, Jiajun Yao, and Chandan Prasad

BEAM SUMMIT Austin, 2022

Why do we need another Beam Runner?

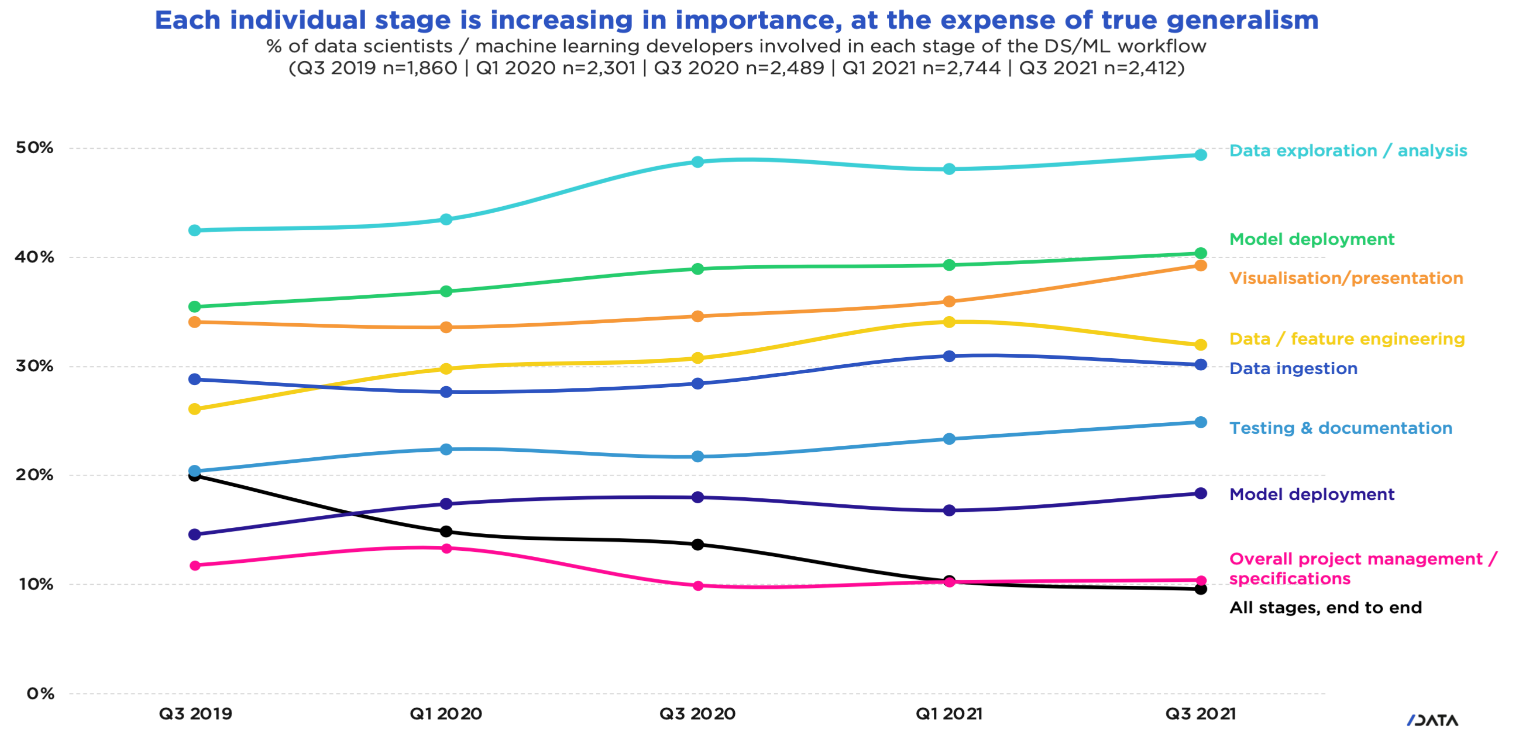
Recent Machine Learning Trends

ML workflows are increasingly distributed and integrated with data analytics:

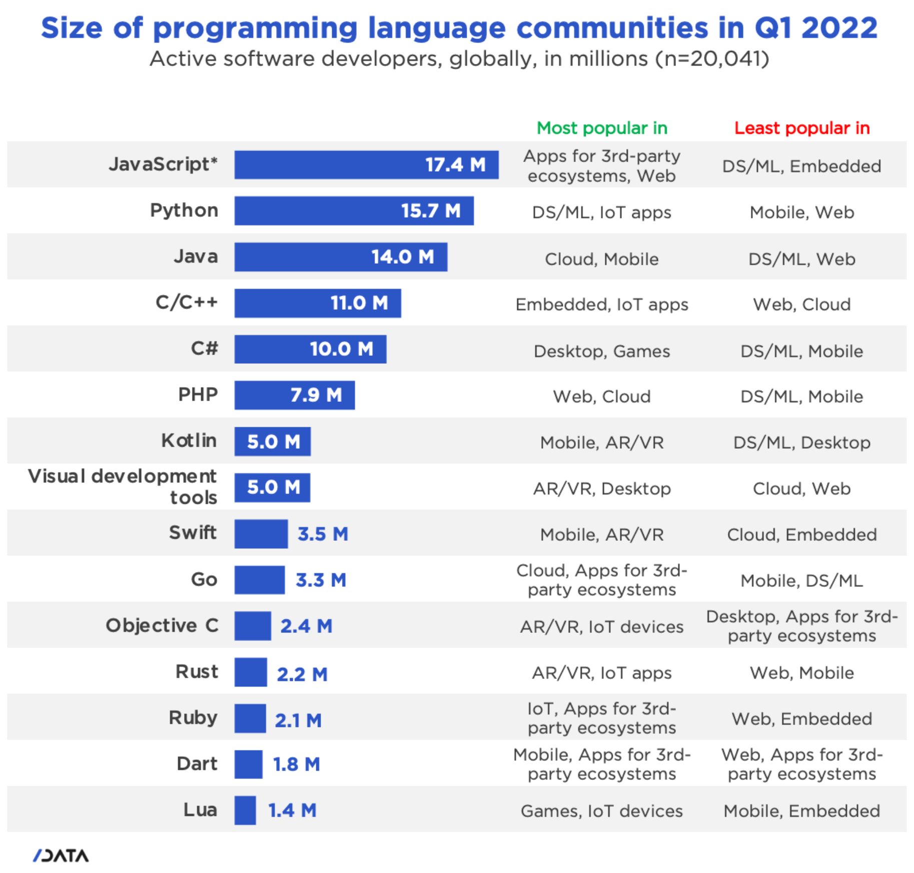
- 2012-2018: 12.5X yearly increase in ML compute demands (from ~0.001 petaflop/s-day to >1000 petaflop/s-day)
- 2015: Moore's Law is slowing (doubling transistor counts every ~20 years)
- 2015 to 2019: 10X/1.5 year increase in ML memory demands
- 2019—2022: An increasing percentage of ML and data science developers are doing data exploration and analysis

ML and data science developers also overwhelmingly prefer to work in Python. Good thing we have Apache Beam's portability framework!

ML Workflow Stage Involvement



Graph from SlashData, licensed under CC 4.0



Size of programming language communities in Q1 2022

Active software developers, globally, in millions (n=20,041)

		Most popular in	Least popular in
JavaScript*	17.4 M	Apps for 3rd-party ecosystems, Web	DS/ML, Embedded
Python	15.7 M	DS/ML, IoT apps	Mobile, Web
Java	14.0 M	Cloud, Mobile	DS/ML, Web
C/C++	11.0 M	Embedded, IoT apps	Web, Cloud
C#	10.0 M	Desktop, Games	DS/ML, Mobile
PHP	7.9 M	Web, Cloud	DS/ML, Mobile
Kotlin	5.0 M	Mobile, AR/VR	DS/ML, Desktop
Visual development tools	5.0 M	AR/VR, Desktop	Cloud, Web
Swift	3.5 M	Mobile, AR/VR	Cloud, Embedded
Go	3.3 M	Cloud, Apps for 3rd-party ecosystems	Mobile, DS/ML
Objective C	2.4 M	AR/VR, IoT devices	Desktop, Apps for 3rd-party ecosystems
Rust	2.2 M	AR/VR, IoT apps	Web, Mobile
Ruby	2.1 M	IoT, Apps for 3rd-party ecosystems	Web, Embedded
Dart	1.8 M	Mobile, Apps for 3rd-party ecosystems	Web, Apps for 3rd-party ecosystems
Lua	1.4 M	Games, IoT devices	Mobile, Embedded

/DATA

Graph from SlashData, licensed under CC 4.0

- Python is the 2nd most popular language in 2022.
- Python is most popular with data science and ML developers.
- Java is least popular with data science and ML developers.
- We want to meet DS and ML developers where they're at.

BEAM SUMMIT

Portable & Unified Data Analytics

We'd like to close these data exploration and analysis gaps with Apache Beam because:

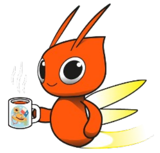
- Portability of code to and from existing Runners reduces barriers to adoption
- We want a unified API to integrate ML workflows with batch and streaming
- Leveraging the Apache Beam SDK reduces undifferentiated heavy-lifting
- There was demand from the Ray open source community

A Runner for DS and ML Devs

We want our Beam Runner to integrate natively with Pythonic tools that data science and ML developers know and love like:

- Pandas
- NumPy
- Dask
- PyArrow
- Scikit-learn
- PyTorch
- TensorFlow
- And many more…

Unified Batch, Streaming, and ML



We also want a unified distributed compute framework to author and run mixed purpose batch, streaming, and ML workflows that let's us:

- Run batch, streaming, and ML on the same cluster with shared memory.
- Write distributed batch, streaming, and ML code in the same Python program.
- Run the same code locally and on a distributed cluster.
- Remove inefficiencies inherent to integrating multiple distributed systems.
- Reuse solutions to common problems like task scheduling, fault tolerance, SerDe, object storage, and cluster autoscaling/management/portability.

We didn't feel like any existing Runner completely met our wish list…

Why Ray?

High-Level Overview

Ray provides a rich set of open source libraries for general purpose distributed computing, machine learning, data science, and workflow management:

- Ray Core
  - General Purpose Distributed Programming
- Ray Libraries
  - End-to-end Distributed Machine Learning Workflow Development
  - Cluster Management and Autoscaling (on AWS, GCP, Azure, Kubernetes, YARN, Slurm)
  - Integrated with Tensorflow, PyTorch, and more.
- Ray Ecosystem
  - Machine Learning (Scitkit-learn, XGBoost, Horovod, Hugging Face, Ludwig)
  - Data Science (Dask, Mars, Modin)
  - Workflow Management (Airflow)

Unified Distributed Compute

Ray Core provides simple but powerful building blocks for distributed computing:

- Tasks
  - Stateless distributed functions.
- Actors
  - Stateful distributed classes with mutable attributes.
- Distributed Object Store (Plasma)
  - Immutable Object Storage
  - Zero-Copy Intranode Object Exchange
- Bottom-up Distributed Scheduler
  - Horizontally Scalable
  - Supports Dynamic Task Graphs
  - Favors Local Scheduling First

Example Task

```
import ray
ray.init()

@ray.remote
def f(x):
    return x * x

results = ray.get([f.remote(i) for i in range(1_000_000)])
print(results)
```

Example Actor

```
@ray.remote
class Counter(object):
    def __init__(self, start):
        self.n = start

    def increment(self):
        self.n += 1
        return self.n

counter = Counter.remote(0)
results = ray.get([counter.increment.remote() for i in range(10)])
print(results)
```

Promising for Batch & Streaming

Early experiments with Ray have shown impressive potential in the domain of batch and streaming data processing:

- >90% latency and efficiency improvements vs. Spark for petabyte-scale batch change-data-capture workloads (CDC)
  - 2021 Ray Summit Presentation: Petabyte Scale Datalake Table Management with Ray
- Ability to scale production workloads across 200,000-CPU clusters in Ant Financial's unified batch/streaming/ML Fusion Engine.
  - 2020 Ray Summit Presentation: Buidling a Fusion Engine With Ray
- 1.8X performance improvement vs. Spark on the 100TB TeraSort benchmark.
  - Exoshuffle Paper

When can I use it?

Current Project Status

Please pardon our dust. Here's what we're up to:

- Building the Ray Runner in Python with Apache Beam's Portability Framework
- Currently test driving a single-process prototype around GitHub
- Developing foundational CI/CD pipelines and test suites
- Continuing to improve Ray Datasets and data ecosystem integrations

Next steps

- Transition our single-process runner prototype to distributed multi-process
- Latency, efficiency, and scalability optimizations for the distributed runner
- Ray ←→ Beam Connectors
- Documentation, user guides, and blog posts
- Get a 1.0 release out the door in 2023!?

Getting Involved

GitHub

Visit us on GitHub:

- Project Home: https://github.com/ray-project/ray_beam_runner
- Feel free to pick up an open issue
- Or create/review a pull request

Slack

Or visit us on the Ray Community Slack:

- Join the Ray Community Slack
- Chat with us on the #beam channel
- Review the latest updates in our pinned progress and design docs

Thanks!