Deep Learning Models at Scale with Apache Spark™

Joseph K. Bradley (speaker), Xiangrui Meng

May 31, 2019

ASA SDSS

About me

Joseph Bradley

- Software Engineer at Databricks
- Apache Spark committer & PMC member

About Databricks

TEAM

Started Spark project (now Apache Spark) at UC Berkeley in 2009

MISSION

Making Big Data Simple

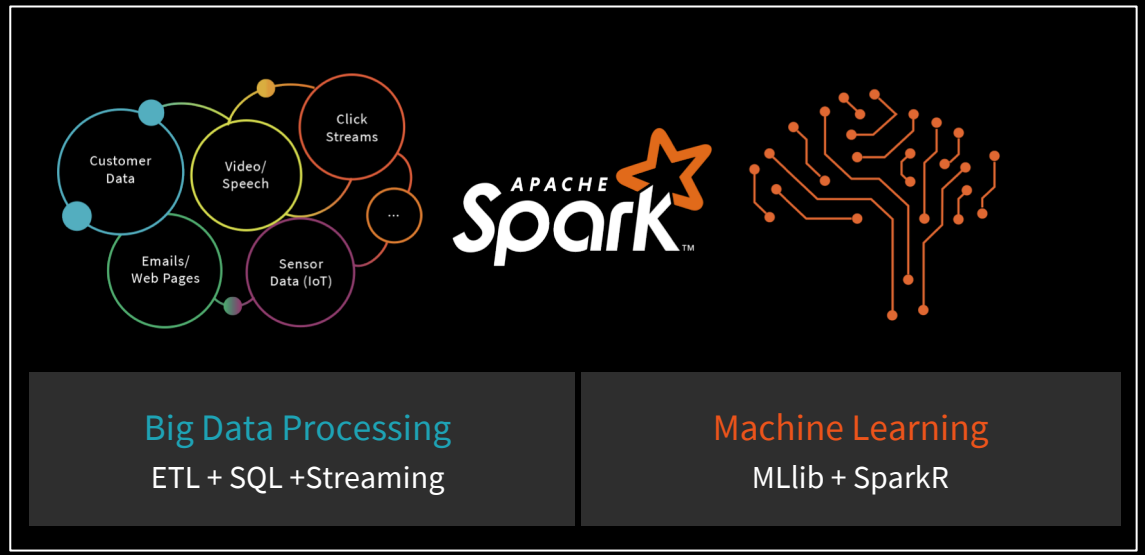
PRODUCT

Unified Analytics Platform

Try for free today.
databricks.com

Apache Spark + AI

Apache Spark:
The First Unified Analytics Engine

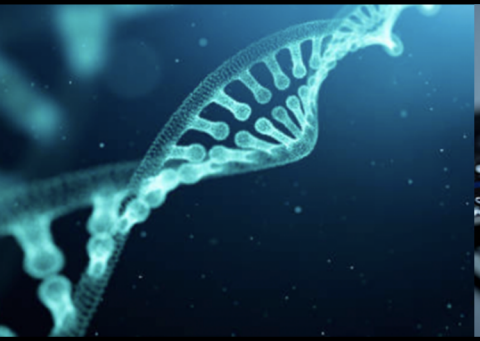


S3 Parquet hadoop kafka Azure

databricks

AI is re-shaping the world

Disruptive innovations are affecting enterprises across the planet



Healthcare and Genomics

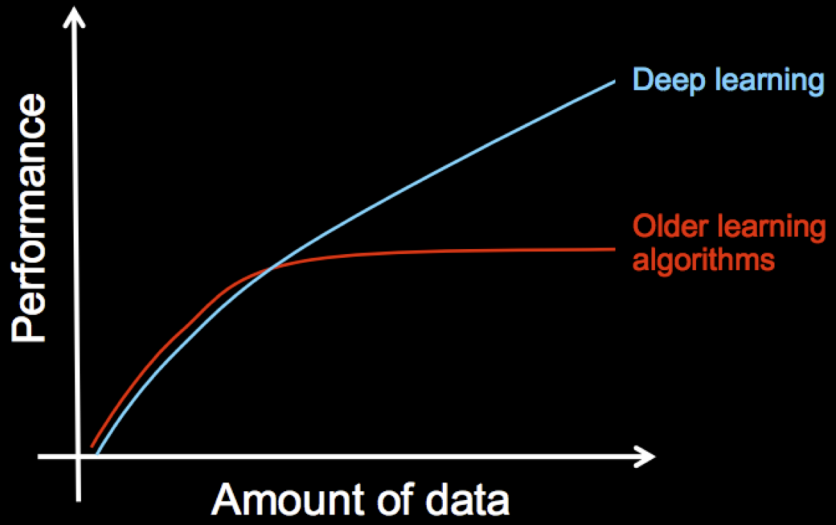
Fraud Prevention

Digital Personalization

Internet of Things

and many more...

Better AI needs more data



When AI goes distributed ...

Larger datasets

➔ More need for distributed training

➔ More open-source offerings: distributed TensorFlow, Horovod, distributed MXNet

This is where Spark and AI meet.

Challenges

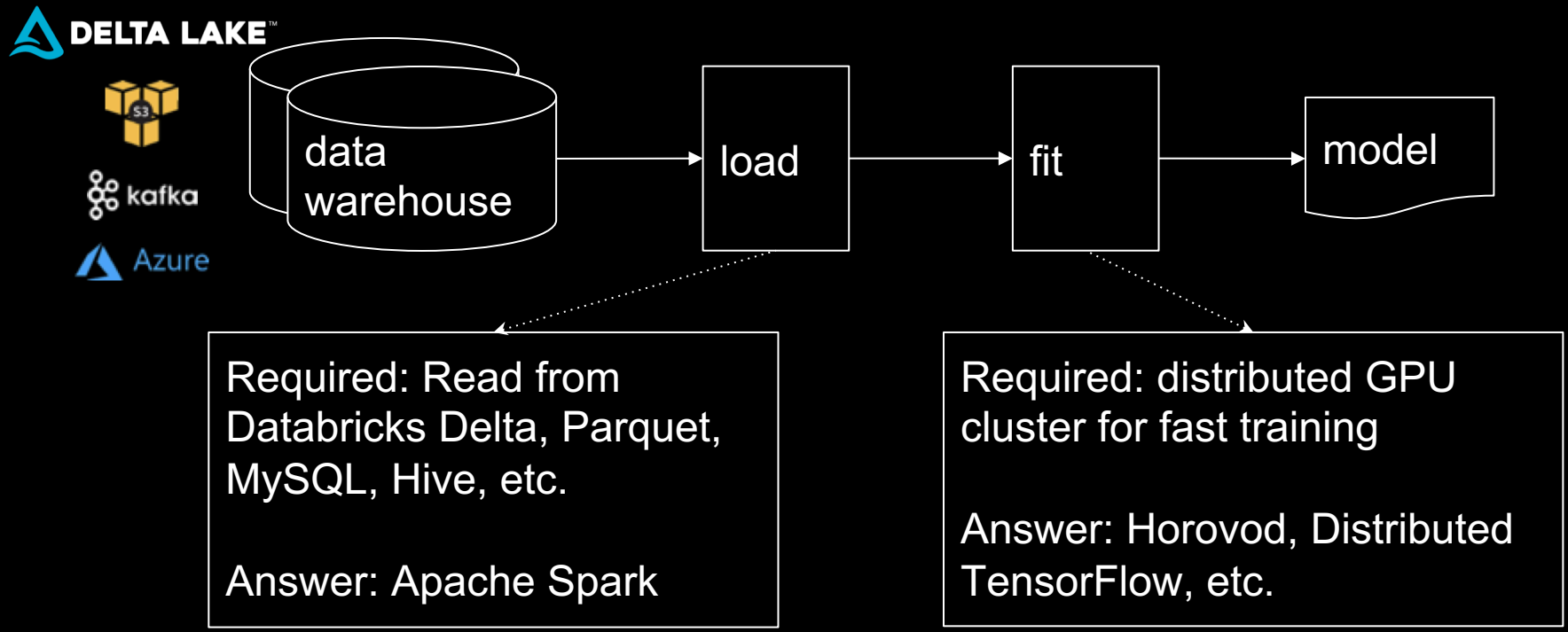
Two user stories

As a data scientist,

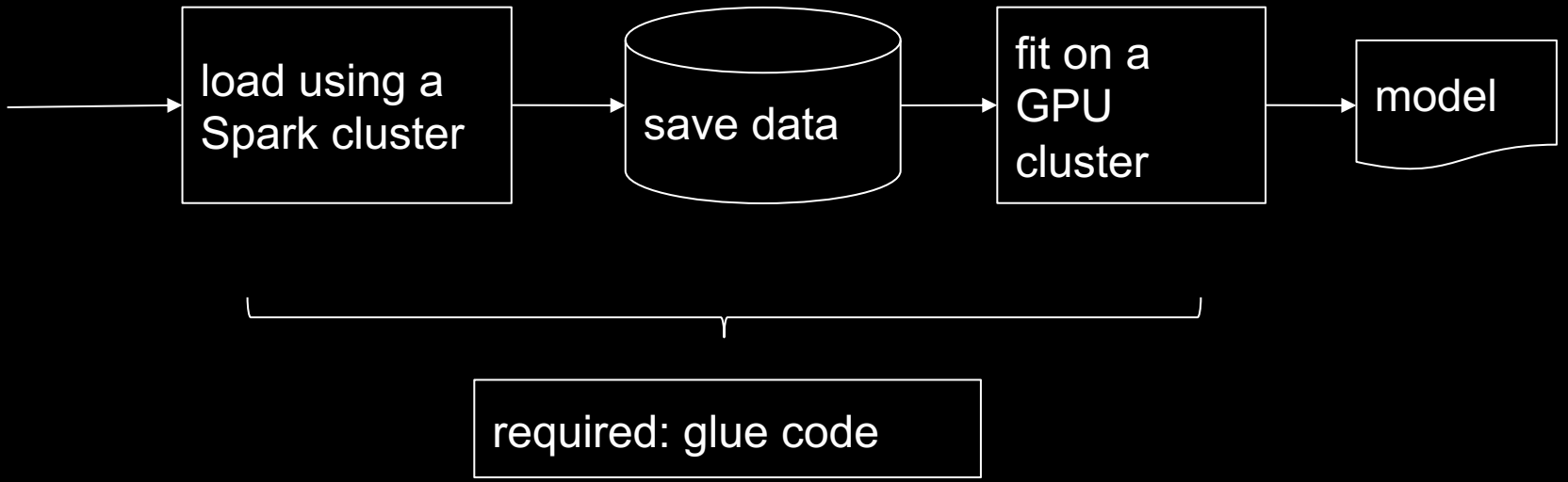
- I can build a pipeline that fetches training data from a production data warehouse and fits a DL model at scale.

- I can apply my DL model to a distributed data stream and augment the data stream with predicted labels.

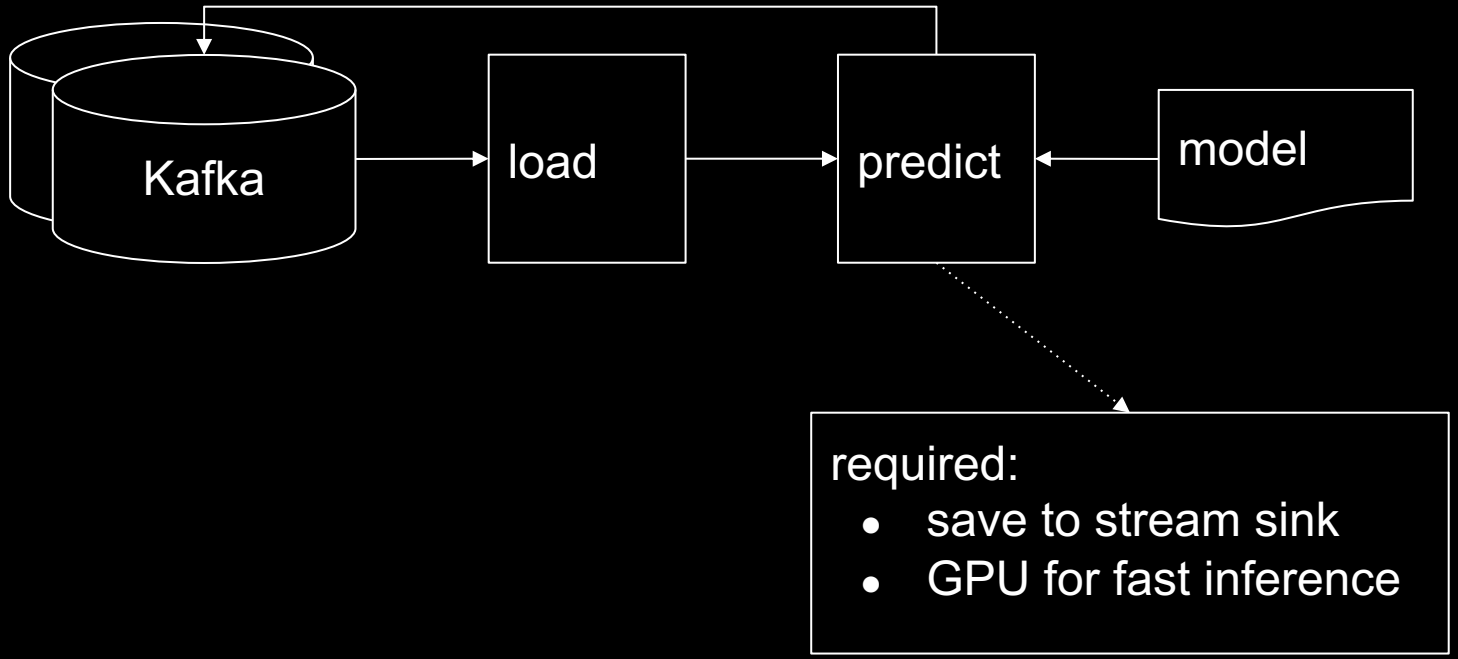
Distributed training



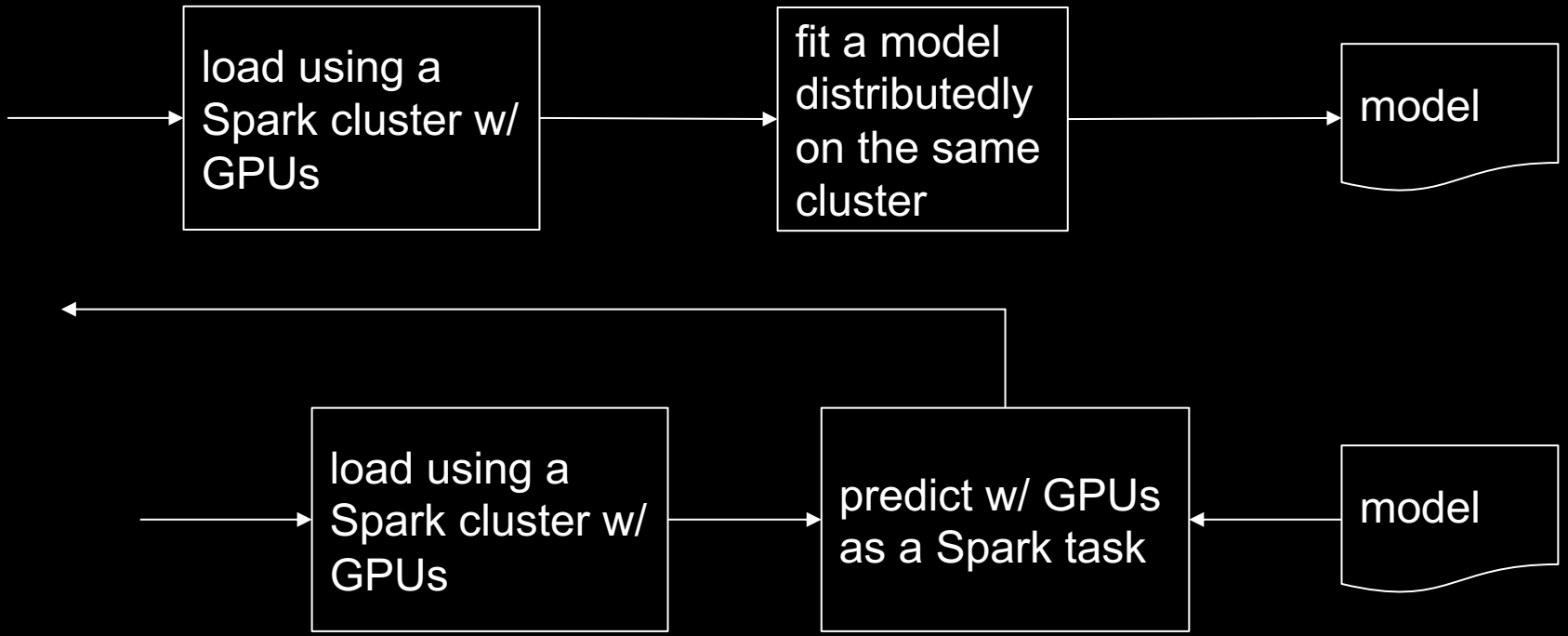
Two separate data and AI clusters?



Streaming model inference



A hybrid Spark and AI cluster?



Unfortunately, it doesn’t work out of the box.

Project Hydrogen

Big data for AI

There are many efforts from the Spark community to integrate Spark with AI/ML frameworks:

- (Yahoo) CaffeOnSpark, TensorFlowOnSpark
- (Intel) BigDL
- (John Snow Labs) Spark-NLP
- (Databricks) TensorFrames, Deep Learning Pipelines, spark-sklearn
- … 80+ ML/AI packages on spark-packages.org

Project Hydrogen to fill the major gaps

Barrier Execution Mode	Accelerator Aware Scheduling	Optimized Data Exchange

In this talk

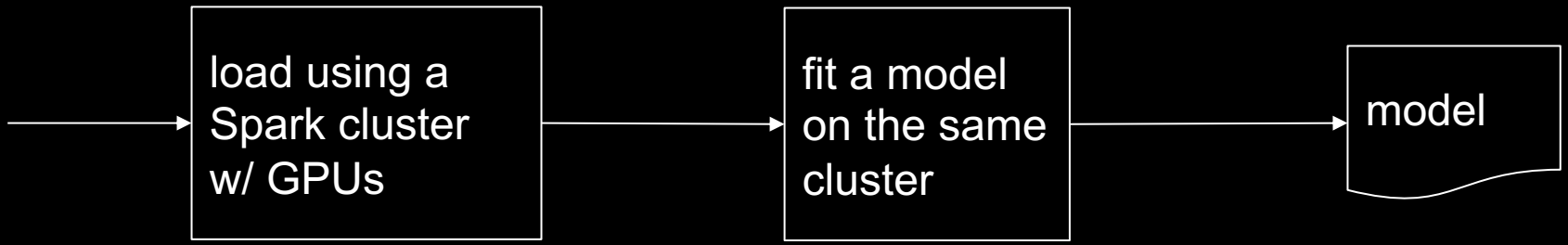
Within Apache Spark:

- Current status of Project Hydrogen features
- Development of new features

From Databricks:

- Our use of Project Hydrogen features
- Lessons learned and best practices

Story #1: Distributed training



Project Hydrogen

Barrier Execution Mode

Accelerator Aware Scheduling

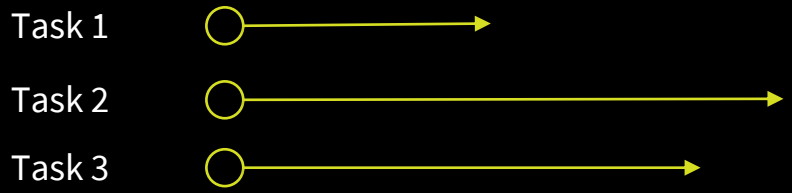
Optimized Data Exchange

Different execution models

Spark (MapReduce)

Tasks are independent of each other

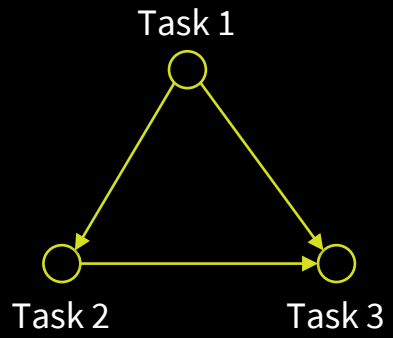
Embarrassingly parallel & massively scalable



Distributed training

Complete coordination among tasks

Optimized for communication



Barrier execution mode

Gang scheduling on top of MapReduce execution model
➔ A distributed DL job can run as a Spark job.

- Coordination: start all tasks together
- Context: provides info to coordinate across tasks
- Failure handling: cancel and restart all tasks in case of failure

JIRA: SPARK-24374 (Spark 2.4)

Barrier mode integration

Horovod (an LF AI hosted project)

Horovod is a distributed training framework for TensorFlow, Keras, PyTorch, and MXNet.

- Little modification to single-node code
- High-performance I/O via MPI and NCCL

Developed at Uber, now an LF AI hosted project at Linux Foundation.

Hydrogen integration with Horovod

HorovodRunner: in Databricks Runtime 5.0 ML

- Runs Horovod under barrier execution mode.
- Hides cluster setup, scripts, MPI command line from users.

```
def train_hvd():
  hvd.init()
  … # train using Horovod

HorovodRunner(np=2).run(train_hvd)
```

Implementation of HorovodRunner

- Pickle and broadcast the train() function.
- Launch a Spark job in barrier execution mode.
- In the first executor, use worker addresses to launch the Horovod MPI job.
- Terminate Horovod if the Spark job got cancelled.

Collaboration on Horovod + Spark

Engineers at Uber and Databricks are improving this integration:

- Merge design and code development into horovod.spark.
- HorovodRunner uses horovod.spark implementation with extra Databricks-specific features.
- Support barrier execution mode and GPU-aware scheduling.

Stay tuned for future announcements!

Project Hydrogen

Barrier Execution Mode

Accelerator Aware Scheduling

Optimized Data Exchange

Accelerator-aware scheduling

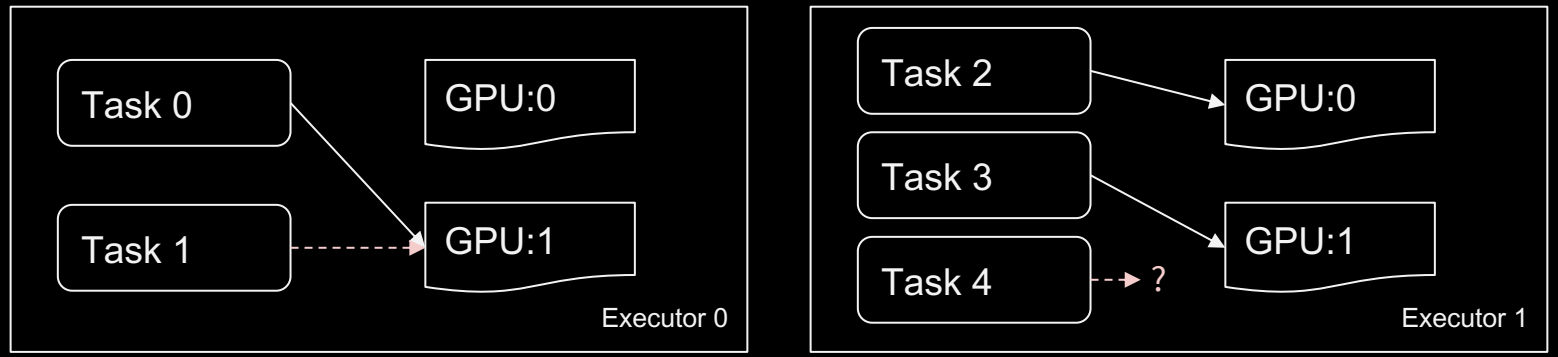
Accelerators (GPUs, FPGAs) are widely used for accelerating specialized workloads like deep learning and signal processing.

Spark is currently unaware of GPUs & other accelerators.

JIRA: SPARK-24615 (ETA: Spark 3.0)

Why does Spark need GPU awareness?

Consider a simple case where one task needs one GPU:



Workarounds (a.k.a hacks)

Limit Spark task slots per node to 1.

- The running task can safely claim all GPUs on the node.
- It might lead to resource waste if the workload doesn't need all GPUs.
- User also needs to write multithreading code to maximize data I/O.

Let running tasks themselves to collaboratively decide which GPUs to use, e.g., via shared locks.

User-facing API

User can retrieve assigned GPUs from task context (#24374)

```
context = TaskContext.get()
assigned_gpu = context.getResources()[“gpu”][0]

with tf.device(assigned_gpu):
  # training code ...
```

Cluster manager support

Standalone

SPARK-27361

YARN

SPARK-27361

Kubernetes

SPARK-27362

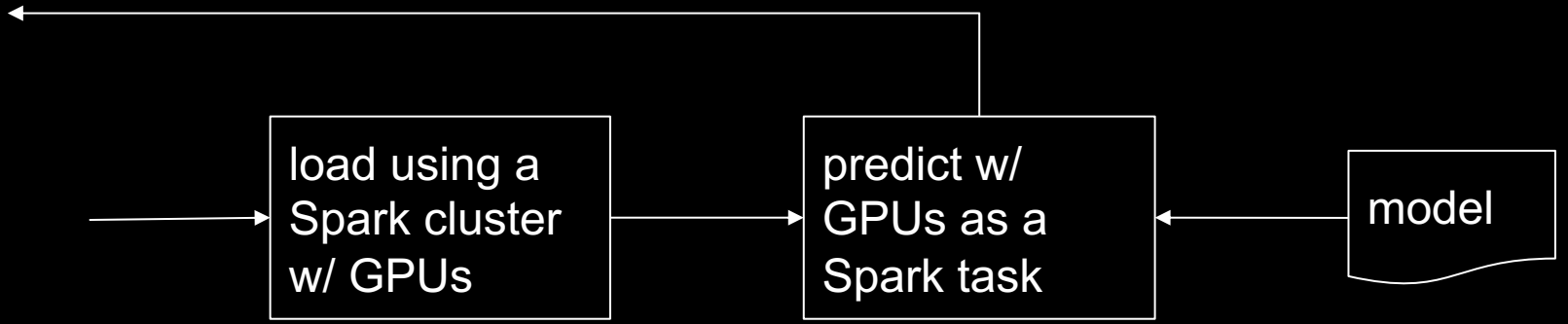
Mesos

SPARK-27363

Future features in discussion

- FPGAs and other accelerators
- Resource request at task level
- Fine-grained scheduling within one GPU
- Affinity and anti-affinity
- ...

Story #2:
Streaming model inference



Project Hydrogen

Barrier Execution Mode

Accelerator Aware Scheduling

Optimized Data Exchange

Optimized data exchange

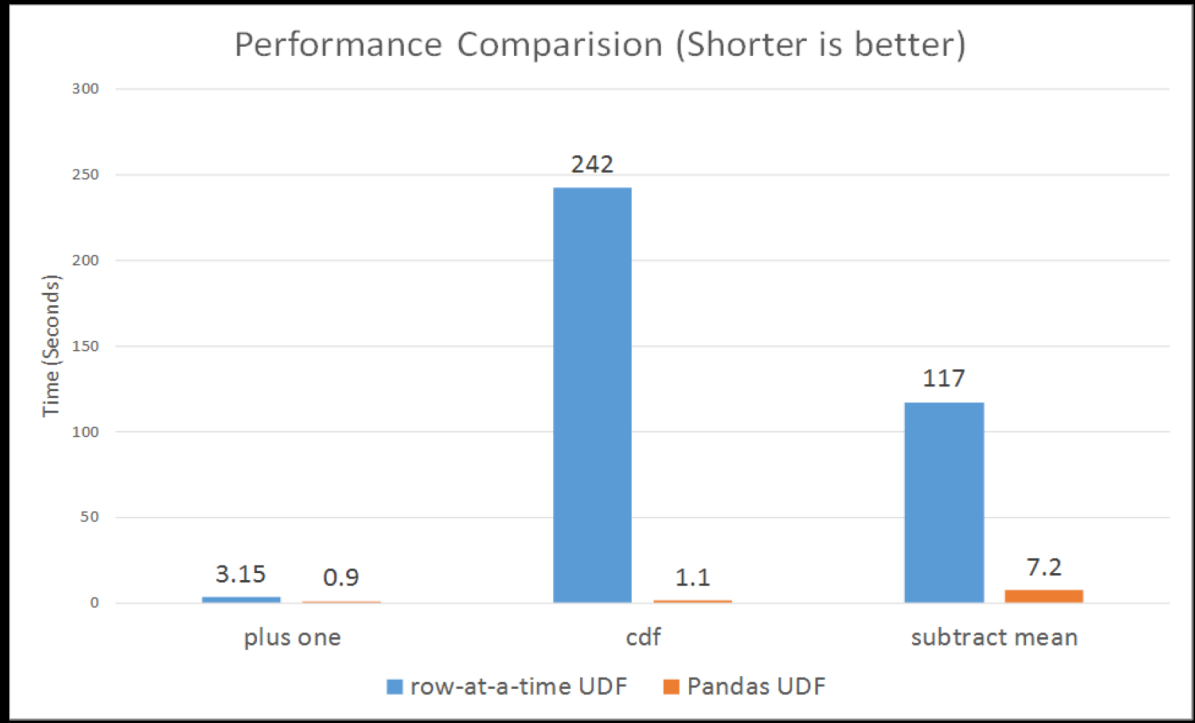
None of the integrations are possible without exchanging data between Spark and AI frameworks. And performance matters.

JIRA: SPARK-24579

Pandas User-Defined Function (UDF)

Pandas UDF was introduced in Spark 2.3

- Pandas for vectorized computation
- Apache Arrow for data exchange



Pandas UDF for distributed inference

Pandas UDF makes it simple to apply a model to a data stream.

```
@pandas_udf(...)
def predict(features):
  ...

spark.readStream(...) \
  .withColumn('prediction', predict(col('features')))
```

Support for complex return types

We improved scalar Pandas UDF to return StructTypes.
E.g., predicted labels and raw scores together

```
@pandas_udf(...)
def predict(features):
  # ...
  return pd.DataFrame({'labels': labels, 'scores': scores})
```

JIRA: SPARK-23836 (Spark 3.0)

Data pipelining

	CPU	GPU
t1	fetch batch #1	
t2		process batch #1
t3	fetch batch #2	
t4		process batch #2
t5	fetch batch #3	
t6		process batch #3

	CPU	GPU
t1	fetch batch #1	
t2	fetch batch #2	process batch #1
t3	fetch batch #3	process batch #2
t4		process batch #3

(pipelining)

Pandas UDF prefetch

Goal: improve throughput

Prefetch next Arrow record batch while executing the Pandas UDF on the current batch.

- Up to 2x for I/O and compute balanced workloads
- Observed 1.5x in real workloads

Enabled by default on Databricks Runtime 5.2.

JIRA: SPARK-27569 (ETA: Spark 3.0)

Per-batch initialization overhead

Loading a model per batch introduces overhead.

New Pandas UDF interface: Load model once.
Then iterate over batches.

```
@pandas_udf(...)
def predict(batches):
  model = … # load model once
  for batch in batches:
    yield model.predict(batch)
```

JIRA: SPARK-26412 (WIP)

Standardize on the Arrow format

Many accelerated computing libraries now support Arrow.

Proposal: Expose the Arrow format in a public interface.

- Simplify data exchange.
- Reduce data copy/conversion overhead.
- Allow pluggable vectorization code.

JIRA: SPARK-27396 (pending vote)

Project Hydrogen

Barrier Execution Mode	Accelerator Aware Scheduling	Optimized Data Exchange

Getting started

Deep Learning on Apache Spark

- Doc sources: docs.databricks.com, github.com/horovod/horovod
- Topics: HorovodRunner, horovod.spark, Pandas UDFs

Project Hydrogen

- Apache Spark JIRA & dev mailing list
- spark.apache.org

Acknowledgements

- Many ideas in Project Hydrogen are based on previous community work: TensorFrames, BigDL, Apache Arrow, Pandas UDF, Spark GPU support, MPI, etc.

- We would like to thank many Spark committers and contributors who helped the project proposal, design, and implementation.

Acknowledgements

- Xingbo Jiang
- Thomas Graves
- Andy Feng
- Alex Sergeev
- Shane Knapp
- Xiao Li
- Li Jin
- Bryan Cutler
- Takuya Ueshin
- Wenchen Fan
- Jason Lowe
- Hyukjin Kwon
- Madhukar Korupolu
- Robert Evans
- Yinan Li
- Felix Cheung
- Imran Rashid
- Saisai Shao
- Mark Hamstra
- Sean Owen
- Yu Jiang
- … and many more!

Thank You!
Questions?

databricks