Inference at Scale with Apache Beam

Danny McCormick

docs.google.com/presentation/d/1JJiLxXEPJgxspDsvpWvnccaGQU2o3xVYPRdZZ60mWpE
(or shorturl.at/qxZ38)
1. Beam History + Overview
2. Basic Inference
3. Problems/Solutions
   - Model Freshness
   - Large Models
   - Specialty Hardware
4. Where Next?
What is Apache Beam
Data got big:
And neverending!
In the beginning, there was MapReduce
In the beginning, there was MapReduce
Then came Flume
From Flume came Beam

Datastore → Map → Group by Key (Reduce) → Map → Combine → Datastore

Datastore → Map → Combine → Datastore

Datastore → Map → Map → Datastore
Batch processing is a special case of stream processing.

Batch + Stream = Beam
Build your pipeline in whatever language(s) you want…

Group by Key
... with whatever execution engine you want

- Cloud Dataflow
- Apache Flink
- Apache Spark
- Dask
- Ray
- Twister2
- Apache Samza
- Prism/Local
Beam Basics
- **PCollection** - distributed multi-element dataset
- **Transform** - operation that takes N PCollections and produces M PCollections
- **Pipeline** - directed acyclic graph of Transforms and PCollections
Basic Beam Graph

- Source Transform
- Map Transform
- Combine Transform
- Sink Transform

Diagram showing the flow of data through various transforms, including source, map, combine, and sink transforms.
def add_one(element):
    return element + 1

import apache_beam as beam

with beam.Pipeline() as pipeline:
    pipeline | beam.io.ReadFromText('gs://some/inputData.txt')
    | beam.Map(add_one)
    | beam.io.WriteToText('gs://some/outputData')
Beam ML
Inference with Beam
Challenges of Distributed Inference

- Efficiently loading models
- Batching
- Model Updates
- Using multiple models
Distributed Inference with Beam

- Beam takes care of all of this with the `RunInference` transform
- Loads model, batches inputs, handles updates, and plugs into DAG

```python
RunInference(model_handler=<config>)
```
```python
>>> data = numpy.array([10, 40, 60, 90],
...                     dtype=numpy.float32).reshape(-1, 1)

>>> model_handler = PytorchModelHandlerTensor(
...    model_class=LinearRegression,
...    model_params={'input_dim': 1, 'output_dim': 1},
...    state_dict_path='gs://path/to/model.pt')

>>> with beam.Pipeline() as p:
...     predictions = (p
...                    | beam.Create(data)
...                    | beam.Map(torch.Tensor)  # Map np array to Tensor
...                    | RunInference(model_handler=model_handler)
...                    | beam.Map(print))
```
Basic Inference Demo

colab.sandbox.google.com/github/apache/beam/blob/master/examples/notebooks/beam-ml/run_inference_huggingface.ipynb
(shorturl.at/brvN9)
Automatic Model Refresh
You’ve deployed your model! Now what?

- New data
- New training algorithms
- New models
Options

- Stop and start your pipeline
- Pipeline drain/update
- Automatic model refresh
Automatic Model Refresh

- Hot swaps model in live pipeline
- Manages memory for you
- No pipeline down time (though maybe some inference down time)
Automatic Model Refresh

side_input_pcoll = (pipeline
| "WatchFilePattern" >> WatchFilePattern(file_pattern=file_pattern,
interval=side_input_fire_interval,
stop_timestamp=end_timestamp))

inferences = (image_data
| "ApplyWindowing" >> beam.WindowInto(beam.window.FixedWindows(10))
| "RunInference" >> RunInference(model_handler=model_handler,
model_metadata_pcoll=side_input_pcoll))
Large Models
Distributed Runner Architecture*
Ideal small model configuration

VM

I/O & Model

I/O & Model

I/O & Model

I/O & Model

I/O & Model

I/O & Model

I/O & Model

I/O & Model
Ideal Large Model Configuration

![Diagram of an Ideal Large Model Configuration]

- VM
- I/O
- Model
- I/O
- I/O
- I/O
- I/O
- I/O

The diagram illustrates the configuration of an ideal large model, with VM at the top, connected to multiple I/O components and a central Model node.
Ideal Multi Large Model Configuration
How do we map ideal model configurations to this?
Ideal small model configuration

- VM
  - I/O & Model
  - I/O & Model
  - I/O & Model
  - I/O & Model
  - I/O & Model
  - I/O & Model
Aka the easy case

```python
>>> model_handler = PytorchModelHandlerTensor(
...    model_class=LinearRegression,
...    model_params={'input_dim': 1, 'output_dim': 1},
...    state_dict_path='gs://path/to/model.pt')

>>> pcoll | RunInference(model_handler=model_handler)
```
Ideal Large Model Configuration
Ideal Large Model Configuration

VM

Worker Process
Worker Process
Worker Process
Worker Process

Inference Process

Worker Process
Worker Process
Worker Process
Worker Process
Optional: serve a single model for all processes

- Reduce memory at cost of interprocess communication, minimized parallelism

```python
>>> model_handler = PytorchModelHandlerTensor(
...    model_class=LinearRegression,
...    large_model=True,
...    model_params={'input_dim': 1, 'output_dim': 1},
...    state_dict_path='gs://path/to/model.pt')

>>> pcoll | RunInference(model_handler=model_handler)
```
Ideal Multi Large Model Configuration

Model 1

Model 2

Model 3
Ideal Large Model Configuration

VM

- Worker Process
- Worker Process
- Worker Process
- Worker Process
- Model Manager
- Worker Process
- Worker Process
- Worker Process
- Worker Process
Optional: serve a single model for all processes

- Model Manager empowered to load/unload models in order to make optimal use of memory

```python
>>> per_key_mhs = [
    ... KeyModelMapping(['key1', 'key2', 'key3'], model_handler_1),
    ... KeyModelMapping(['foo', 'bar', 'baz'], model_handler_2)
]

>>> mh = KeyedModelHandler(per_key_mhs)

>>> pcoll | RunInference(model_handler=mh)
```
Large Model Demo

colab.sandbox.google.com/github/apache/beam/blob/master/examples/notebooks/beam-ml/per_key_models.ipynb
(shorturl.at/pKNU2)
Specialty Hardware
GPU/TPU Support

- Hardware availability dependent on runner
- Beam has some primitives that help
● Resource hints for heterogeneous pools
● Built in detection + framework specific responses to GPUs at the ModelHandler level
● Large model setting (revisited)
Ideal Large Model Configuration

- VM
  - Worker Process
  - Worker Process
  - Worker Process
  - Worker Process
  - Inference Process
  - Worker Process
  - Worker Process
  - Worker Process
  - Worker Process
Where next?

(opportunities I see, not representative of the whole community)
Inference Space

- More frameworks
- Better performance testing/profiling
- Model Manager Improvements
Beyond Inference

- MLTransform for data prep and pre/postprocessing
- Feature Store Enrichment
- Higher level ML support (e.g. anomaly detection)
Come join our community!
Questions?

Contact - Danny McCormick (dannymccormick@google.com)

Slides - https://shorturl.at/jzEQ6