Optimizing ML Workloads on Dataflow
Mindful Chef
Back to healthy
Beatrix returns in September
healthy, delicious meals
mindsfuleat.com
Apache Beam at Trustpilot
Apache Beam at Trustpilot

- NLP
  - B2B and B2C analytics
  - Sentiment analysis
  - Topic modelling
Apache Beam at Trustpilot

- **NLP**
  - B2B and B2C analytics
  - Sentiment analysis
  - Topic modelling

- **Platform integrity**
  - Scam/spam
  - Fake reviews
  - Bad actors
Apache Beam at Trustpilot

- **NLP**
  - B2B and B2C analytics
  - Sentiment analysis
  - Topic modelling

- **Platform integrity**
  - Scam/spam
  - Fake reviews
  - Bad actors

- **Feature Store**
  - Online store (real time stream)
  - Offline store (batch)
  - User events, review activity
Apache Beam at Trustpilot

- **NLP**
  - B2B and B2C analytics
  - Sentiment analysis
  - Topic modelling

- **Platform integrity**
  - Scam/spam
  - Fake reviews
  - Bad actors

- **Feature Store**
  - Online store (real time stream)
  - Offline store (batch)
  - User events, review activity

- **All on GCP Dataflow**
Dataflow Prime cost savings
  - Saving 40% on our large batch ML jobs
Agenda

- Dataflow Prime cost savings
  - Saving 40% on our large batch ML jobs

- Multi-model GPU sharing
  - Multiple embedding vectors for LLM applications
Dataflow Prime cost savings
  - Saving 40% on our large batch ML jobs

Multi-model GPU sharing
  - Multiple embedding vectors for LLM applications

**JAX for lin. alg. speedup**
  - Speeding up linear algebra operations in Beam
• Dataflow Prime cost savings
  ○ Saving 40% on our large batch ML jobs

• Multi-model GPU sharing
  ○ Multiple embedding vectors for LLM applications

• JAX for lin. alg. speedup
  ○ Speeding up linear algebra operations in Beam

• **RunInference** for ML inference
  ○ Improving codebase maintainability
Dataflow Prime

- Enable pipeline step-level resource specification
Dataflow Prime

- Enable pipeline step-level resource specification

Example ML pipeline

- Prepare data in BigQuery for processing
- Text preprocessing
- Feature engineering
- ML model inference
- Launch load of data into BigQuery
- Enable pipeline step-level resource specification
- Example ML pipeline
- Imbalanced resource requirements
  - Each step has differing resource requirements

Example pipeline:

- **Input I/O**
  - Succeeded
  - 4 of 4 stages succeeded
  - **Prepare data in BigQuery for processing**
    - **Low CPU, low RAM**

- **RAM-intensive preprocessing**
  - Succeeded
  - 1 of 1 stage succeeded
  - **Text preprocessing**
    - **Feature engineering**
    - **High CPU, high RAM**

- **GPU-intensive inference**
  - Succeeded
  - 3 of 3 stages succeeded
  - **ML model inference**
    - **Mid CPU, mid RAM, GPU**

- **Output I/O**
  - Succeeded
  - 8 of 8 stages succeeded
  - **Launch load of data into BigQuery**
    - **Low CPU, low RAM**
Dataflow Prime

- Enable pipeline step-level resource specification
- Example ML pipeline
- **Imbalanced resource requirements**
  - Each step has differing resource requirements

---

**Example pipeline**

- Prepare data in BigQuery for processing
  - Low CPU, low RAM
- Text preprocessing
  - Feature engineering
  - High CPU, high RAM
- ML model inference
  - Mid CPU, mid RAM, GPU
- Launch load of data into BigQuery
  - Low CPU, low RAM
- Enable pipeline step-level resource specification
- Example ML pipeline
- Imbalanced resource requirements - Each step has differing resource requirements
- Forced to take the "argmax" over each resource type

Input I/O
- Succeeded - 4 of 4 stages succeeded

RAM-intensive preprocessing
- Succeeded - 1 of 1 stage succeeded

GPU-intensive inference
- Succeeded - 3 of 3 stages succeeded

Output I/O
- Succeeded - 8 of 8 stages succeeded

- Prepare data in BigQuery for processing
  - Low CPU, low RAM
- Text preprocessing
  - Feature engineering
    - High CPU, high RAM
- ML model inference
  - Mid CPU, mid RAM, GPU
- Launch load of data into BigQuery
  - Low CPU, low RAM

custom-4-32000-ext 1x T4 GPU
Dataflow Prime

- Enable pipeline step-level resource specification
- Example ML pipeline
- Imbalanced resource requirements
  - Each step has differing resource requirements
- Forced to take the "argmax" over each resource type

With resource hint specification

- Prepare data in BigQuery for processing
  - Low CPU, low RAM
  - 4GB RAM
- Text preprocessing
  - Feature engineering
  - High CPU, high RAM
  - 32GB RAM
- ML model inference
  - Mid CPU, mid RAM, GPU
  - 8GB RAM, 1xT4 GPU
- Launch load of data into BigQuery
  - Low CPU, low RAM
  - 4GB RAM
with beam.Pipeline(options=PipelineOptions(  
)) as pipeline:  
  _ = (  
    pipeline  
      | inputPTransform()  
      | preprocPTransform()  
      | inferencePTransform()  
      | inferencePTransform()  
  )

With resource hint specification:

- **Prepare data in BigQuery for processing**
  - Low CPU, low RAM
  - 4GB RAM

- **Text preprocessing**
  - Feature engineering
  - High CPU, high RAM
  - 32GB RAM

- **ML model inference**
  - Mid CPU, mid RAM, GPU
  - 8GB RAM, 1xT4 GPU

- **Launch load of data into BigQuery**
  - Low CPU, low RAM
  - 4GB RAM
with beam.Pipeline(options=PipelineOptions(dataflow_service_options=['enable_prime'])) as pipeline:
    _ = (pipeline |
        inputPTransform().with_resource_hints(min_ram="4GB", accelerator=None) |
        preprocPTransform().with_resource_hints(min_ram="32GB", accelerator=None) |
        inferencePTransform().with_resource_hints(min_ram="8GB", accelerator="type:nvidia-tesla-t4;count:1;install-nvidia-driver") |
        inferencePTransform().with_resource_hints(min_ram="4GB", accelerator=None))
Dataflow Prime cost savings

Dataflow Prime cost as % of default Dataflow cost for batch NLP job

% of default Dataflow cost

million rows

0 20 40 60 80 100 120

0.0 2.5 5.0 7.5 10.0 12.5 15.0 17.5 20.0
Review

- I/O can take a lot of time
  - In particular the `WriteToBigQuery PTransform` is I/O-bound
  - Can take a long time:
    - `TriggerLoadJobs DoFn`
    - `WriteRecordsToFile DoFn`
  - Don't need special resources for these steps
  - With `Prime`, just specify to run with lower resources

- GPU steps are dependent on CPU-bound steps
  - Separating these steps enables more efficient use of GPU
  - Ensures higher batch size can be realized == higher throughput
  - Breaking the dependency ensures higher GPU utilization
- Benchmark your pipelines to measure expected savings
  - It may not be the case that you will end up better off

- Break fusion to discretize worker pools
  - Dataflow will sometimes fuse together steps you want to keep separate
  - Insert a Reshuffle PTransform
- LLM-based products and applications
  - chat, agents
- Retrieval augmentation
  - add relevant context to prompt to reduce hallucinations
- Embed data for similarity lookup
  - vector store
- LLM-based products and applications
  - chat, agents

- Retrieval augmentation
  - add relevant context to prompt to reduce hallucinations

- Embed data for similarity lookup
  - vector store

- Multiple ways to embed data
  - semantic
  - fraud detection
  - recommendations

Multi-model GPU sharing

What do users say about our documentation?
- Clear and precise documentation: Users have mentioned that the documentation provided by the business is clear and precise, making it easy to understand and follow [1].
- Plenty of examples: The documentation includes plenty of examples that help users in setting up and integrating the services [2].
- Some users desire more tutorials: Although the documentation is appreciated, some users have expressed the need for more tutorials on how to perform certain tasks [3].

How can we improve our documentation to make it even more user-friendly and comprehensive for our customers?

What specific topics or areas do our users want more tutorials on?
- Currently: Difficult to share GPU
- Forthcoming: Load multiple models onto GPU
- Sink to vector store
  - Multi-embedding index
Speeding up linalg operations

- Linear algebra operations
  - Matrix multiplication
  - PCA
  - Distance metrics
  - Broadcasted vector operations
  - Even neural networks
import numpy as np

def _euclidean_dist_np (X,Y):
    squared_diffs = np.power(X[:,None] - Y, 2)
    summed = np.sum(squared_diffs, axis=-1)

    return np.sqrt(summed)
from jax import numpy as jnp

def _euclidean_dist_jax(X, Y):
    squared_diffs = jnp.power(X[:, None] - Y, 2)
    summed = jnp.sum(squared_diffs, axis=-1)

    return jnp.sqrt(summed)

# JIT compile
euclidean_dist_jax = jit(_euclidean_dist_jax)
① Define a pure function
- `@staticmethod`
- Linear algebra operations

```python
class RBFKernel(DoFn):
    def __init__(self):
        self.rbf_jax = jit(self._rbf)

    @staticmethod
    def _rbf(X, Y, gamma):
        def distance(X, Y):
            return jnp.sqrt(jnp.sum(jnp.power(X[:, None] - Y, 2), axis=-1))

        d = distance(X, Y)
        return jnp.exp(-gamma * d**2)
```

① Define a pure function
Set up JAX in Beam

① Define a pure function
- @staticmethod
- Linear algebra operations

② JIT compile
- In __init__
- Compile once

Code

```python
class RBFKernel(DoFn):
    def __init__(self):
        self.rbf_jit = jit(self._rbf)

    @staticmethod
def _rbf(X, Y, gamma):
        def distance(X, Y):
            return jnp.sqrt(jnp.sum(jnp.power(X[:, None] - Y, 2), axis=-1))
        
        d = distance(X, Y)
        return jnp.exp(-gamma * d**2)
```

① Linear algebra operations
② JIT compile
③ Compile once
Set up JAX in Beam

① Define a pure function
   - @staticmethod
   - Linear algebra operations

② JIT compile
   - In __init__
   - Compile once

③ Use the compiled function

Code

class RBFKernel(DoFn):
    def __init__(self):
        self.rbf_jit = jit(self._rbf)

@staticmethod
def _rbf(X, Y, gamma):
    def distance(X, Y):
        return jnp.sqrt(
            jnp.sum(jnp.power(X[:, None] - Y, 2), axis=-1))

    d = distance(X, Y)
    return jnp.exp(-gamma * d**2)

def process(self, elem):
    ...
    yield key, self.rbf_jit(X, Y, gamma)
Set up JAX in Beam

① Define a pure function
- @staticmethod
- Linear algebra operations

② JIT compile
- In __init__
- Compile once

③ Use the compiled function
- PCA + distance metric faster by ~10x compared to sklearn implementation (CPU)

Code

```python
class RBFKernel(DoFn):
    def __init__(self):
        self.rbf_jit = jit(self._rbf)

@staticmethod
def _rbf(X, Y, gamma):
    def distance(X, Y):
        return jnp.sqrt(jnp.sum(jnp.power(X[:, None] - Y, 2), axis=-1))

    d = distance(X, Y)
    return jnp.exp(-gamma * d**2)

def process(self, elem):
    ... yield key, self.rbf_jit(X, Y, gamma)
```

① ② ③
- **RunInference**
  - *PTransform* in the Python SDK for running ML inference
Custom inference (previous)

- Create weak references to model object
- Set shared handle on model init. func.
- Manual device/CUDA management
- Dealing with different APIs for each ML library

RunInference

- Choose a ModelHandler and provide model URI
- Model object sharing is handled inside RunInference
- Device set as part of ModelHandler args (PyTorch)
- Pick among many supported ModelHandlers
"Benchmarking" codebase improvements

~/go/bin/scc src/ml/sentiment.py

<table>
<thead>
<tr>
<th>Language</th>
<th>Files</th>
<th>Lines</th>
<th>Blanks</th>
<th>Comments</th>
<th>Code Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Python</td>
<td>1</td>
<td>212</td>
<td>32</td>
<td>68</td>
<td>112</td>
</tr>
</tbody>
</table>

Total

<table>
<thead>
<tr>
<th>Language</th>
<th>Files</th>
<th>Lines</th>
<th>Blanks</th>
<th>Comments</th>
<th>Code Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>212</td>
<td>32</td>
<td>68</td>
<td>112</td>
</tr>
</tbody>
</table>

Estimated Cost to Develop (organic) $2,711
Estimated Schedule Effort (organic) 1.46 months
Estimated People Required (organic) 0.17

Processed 7007 bytes, 0.007 megabytes (SI)
"Benchmarking" codebase improvements

```
~/go/bin/scc ./v2/experimental/transforms/ml/sentiment.py
```

<table>
<thead>
<tr>
<th>Language</th>
<th>Files</th>
<th>Lines</th>
<th>Blanks</th>
<th>Comments</th>
<th>Code Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Python</td>
<td>1</td>
<td>122</td>
<td>16</td>
<td>52</td>
<td>54</td>
</tr>
<tr>
<td>Total</td>
<td>1</td>
<td>122</td>
<td>16</td>
<td>52</td>
<td>54</td>
</tr>
</tbody>
</table>

Estimated Cost to Develop (organic) $1,260  
Estimated Schedule Effort (organic) 1.09 months  
Estimated People Required (organic) 0.10

Processed 3972 bytes, 0.004 megabytes (SI)

GitHub: boyter/scc  
GitHub: boyter/scc—Complexity  
Wikipedia: Cyclomatic complexity
Recap

- **Use case 1: Large batch pipelines with imbalanced resources**
  - Dataflow Prime cost saving 40%

- **Use case 2: Running pipelines with multiple embedding models**
  - Forthcoming on Dataflow

- **Use case 3: Speeding up linear algebra operations in Beam**
  -

- **Use case 4: Improve codebase maintainability**
  - Replace custom inference implementations with `RunInference`
Resources

- https://cloud.google.com/dataflow/docs/guides/enable-dataflow-prime
- https://beam.apache.org/documentation/runtime/resource-hints/
- Qdrant—Storing multiple vectors per object in Qdrant
- https://github.com/google/jax
- https://beam.apache.org/documentation/transforms/python/elementwise/runinference/
- GitHub: boyter/scc
- GitHub: boyter/scc—Complexity
- Wikipedia: Cyclomatic complexity