Beam for Large-Scale, Accelerated ML Inference

Presenter: Uday Kalra
Software Engineer @ Google

BEAM SUMMIT

September 4-5, 2024

Sunnyvale, CA. USA

Terminology

Words I Will Use and What I Mean

Terminology

- **Beam** → Front-end programming model for batch and streaming data processing.
    - Specify pipelines typically composed of **PTransforms** applied to **PCollections**.
- **Engine** → Backend Beam Runner (Dataflow, Flink, etc.)
- **Accelerator** → Referring to Cloud TPUs/GPUs.
    - Typically linked to a serving host machine to execute workloads.
- **JAX** → Computation library popular for ML engineering/research throughout Google.

Bulk Inference

- Offline ML Prediction on a **known** and **potentially vast** collection of data.
    - Enough to **saturate your accelerators** for long periods of time (>1 hour).
    - O(1000) slow predictions or O(1M) quick predictions.

- **Throughput** (Overall Generations/Sec) favored over per-input latency.

- Typical Use Cases:
    - Large-Scale Model Evaluation
    - Dataset Prediction Statistics
    - Teacher Model Distillation

The Game Plan

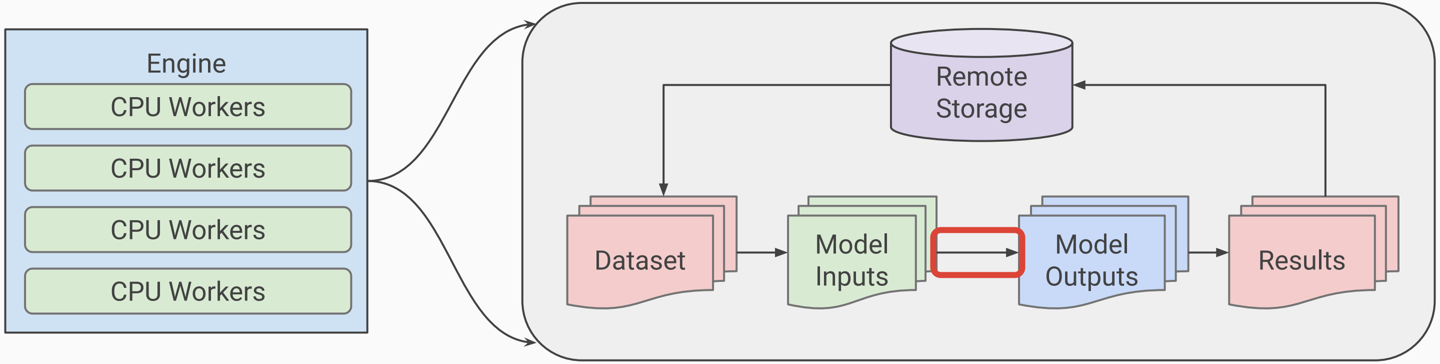
What do Google's engines prioritize for B.I.?

The Goal

- Optimize for **Cost** and **Developer Velocity.**
  - **Efficiency**
    - Accelerators have high power-consumption and cost.
    - Poor utilization/saturation → Expensive and potentially wasteful.
  - **Ergonomics**
    - Engineers/Researchers in ML like Beam for scaling transformations.
    - Inference → A transformation from input to prediction.

The Goal

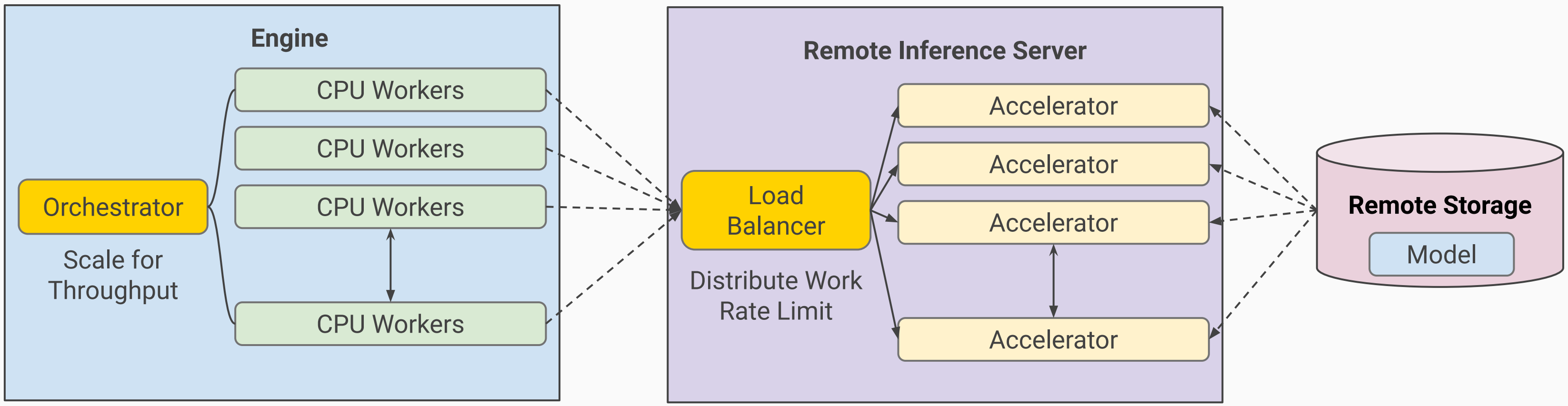
```
generations = (
  read_remote_dataset('/path/to/data')
  | 'Preprocess' >> beam.ParDo(PreprocessData())
  | 'RunInference' >> PredictionPTransform()
  | 'Postprocess' >> beam.ParDo(PostprocessData())
)
generations | 'WriteResults' >>  write_dataset('path/for/outputs')
```



BEAM SUMMIT

Inference Stage Design

Remote Server Bulk Inference

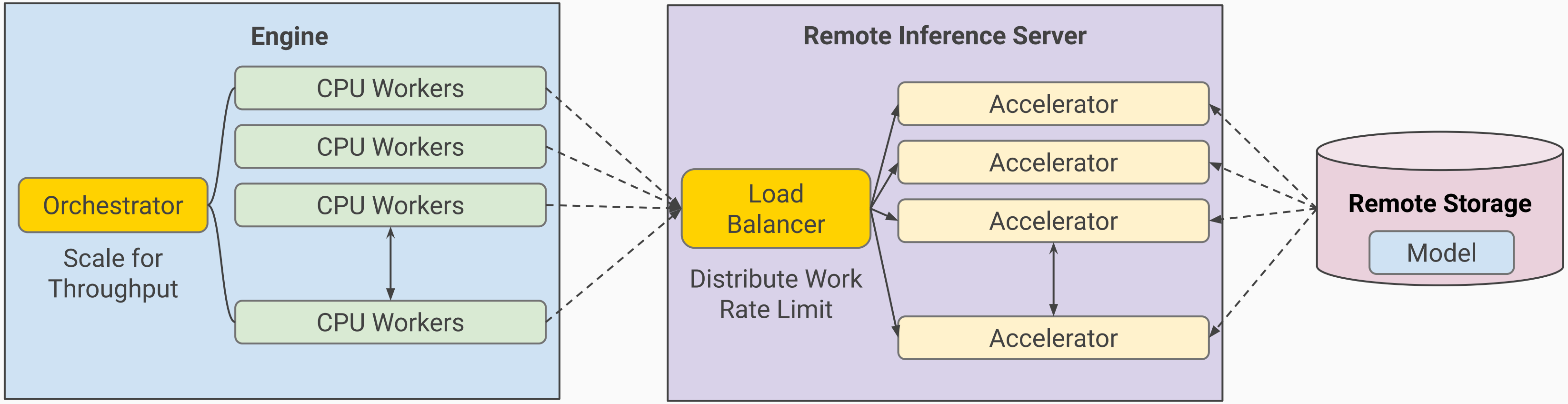


Beam API

```
class RemoteInferenceDoFn(beam.DoFn):
  def __init__(self, server_addr):
    self.server_addr = server_addr

  def setup():
      # Occurs once as worker initialization.
      self.model_client = ModelClient(self.server_addr)

  def process(self, element):
      # Occurs for each PCollection element.
      prediction_request = BuildRequest(element)
      prediction_result = self.model_client(prediction_request)
      generation = prediction_result.output()
      Return [generation]
```

- Model inputs are serialized to send to the remote server.
- Model server parses request and runs JAX model inference function.
- Serialized result is returned to the DoFn worker.

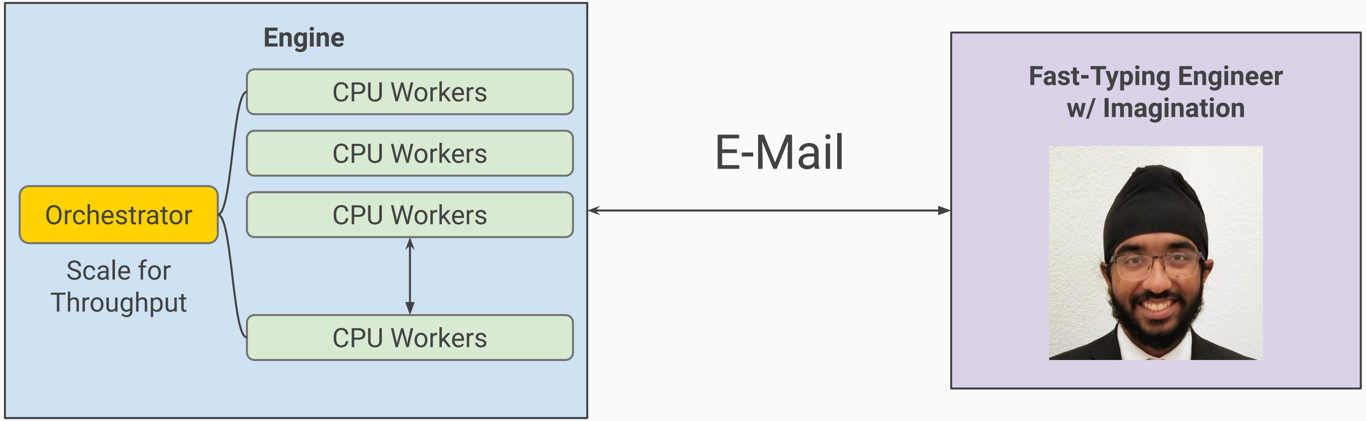
Remote Server Bulk Inference



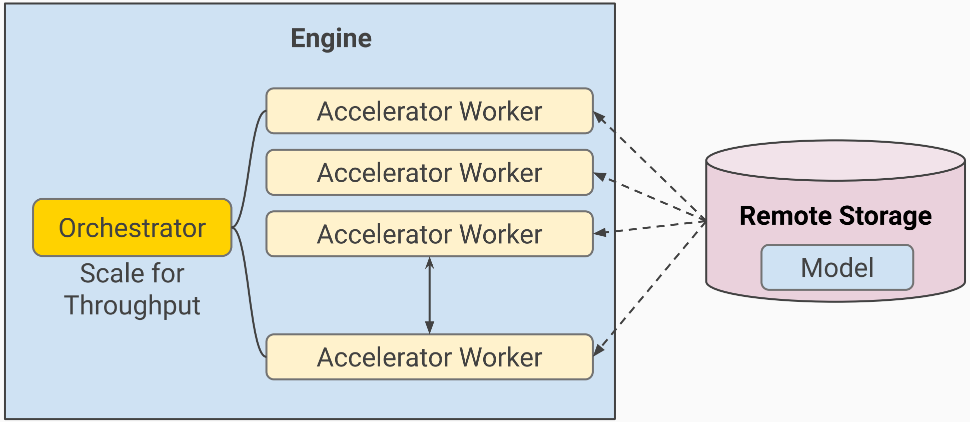
- Load-scaling systems are opponents.
- Number of accelerators can be difficult to determine.
- Server spin-up/down needs to be handled externally.
- Developer has to maintain model server.

BEAM SUMMIT

Remote Engineer Bulk Inference



Local Server(less) Bulk Inference

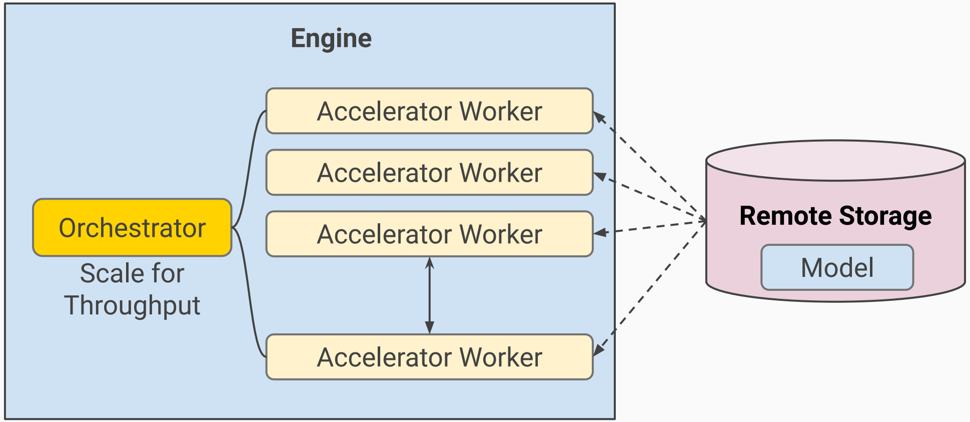


Beam API

```
import jax
from sample_model import MyModel

class LocalInferenceDoFn(beam.DoFn):
  def __init__(self, model_checkpoint):
    self.model_checkpoint = model_checkpoint

  def setup():
      # Occurs once as worker initialization.
      jax.config.update('jax_platforms', 'tpu')
      self.MyModel = MyModel.load(model_checkpoint)

  def process(self, element):
      # Occurs for each PCollection element.
     generation = self.MyModel.inference(element)
     return [generation]
```

- Beam worker configures JAX backend to utilize a TPU accelerator.
- Model is loaded directly on the Beam Worker.
- Model's inference function is executed locally.

Local Server(less) Bulk Inference



- Accelerator workers are directly saturated with work.
- Number of accelerators can be scaled for throughput.
- All in a single pipeline run.
- No more model server.

Beam API

```
class PredictionPTransform(beam.PTransform):
  def expand(self, pcoll):
    return pcoll | beam.ParDo(
      LocalInferenceDoFn(
        model_checkpoint="/path/to/checkpoint")
      ).with_resource_hints(
        min_ram="4GB",
        accelerator="type:sample-tpu;count:1"
    )
generations = (
  read_remote_dataset('/path/to/data')
  | 'Preprocess' >> beam.ParDo(PreprocessData())
  | 'RunInference' >> PredictionPTransform()
  | 'Postprocess' >> beam.ParDo(PostprocessData())
)
```

- User can specify accelerator information to the engine via resource hint API.
- Pipelines can have several inference stages all managed by the engine.
- Accelerator inference stages and CPU stages can be intermixed freely.

Serverless Challenges

- Potential Debugging Complexity
  - Issues may require joint experience with JAX and Beam
- Accelerator Workers are not CPU workers
  - Engine may require adjustment for novel latency
- Worker scaling can be excessive for predictable workloads
  - Bulk Inference at core is [Data In -> Predictions Out].

Takeaways

- Bulk Inference systems are a vast space.
  - Design choices in any dimension will rarely fit all use cases.
  - One can optimize for cost, dev velocity, latency, etc.
- Beam is a powerful, flexible tool for applying operations to data at scale.
  - Inference → One such transformation.
  - With novel engine design, the API naturally extends to GenAI use-cases.

Thank you!

Questions?

Uday Kalra

linkedin.com/in/udaykalra

Special Thanks:

Zhao Fu
Alex Salcianu
Xiaopan Zhang
Anton Bobkov
Tanvir Hassan
Mark Omernick
Hao Zhou
Aleksander Zaks
Zsolt Márton
Josh Newlan
Zachary Westrick
Ruoyu Liu
Evan Rosen
Gene Huang