Node-Level Deep Learning Input Pipeline Optimization on GPGPU-Accelerated HPC Systems

28 Mar 2018

Captain Justin Fletcher
Air Force Research Laboratory

Integrity ★ Service ★ Excellence

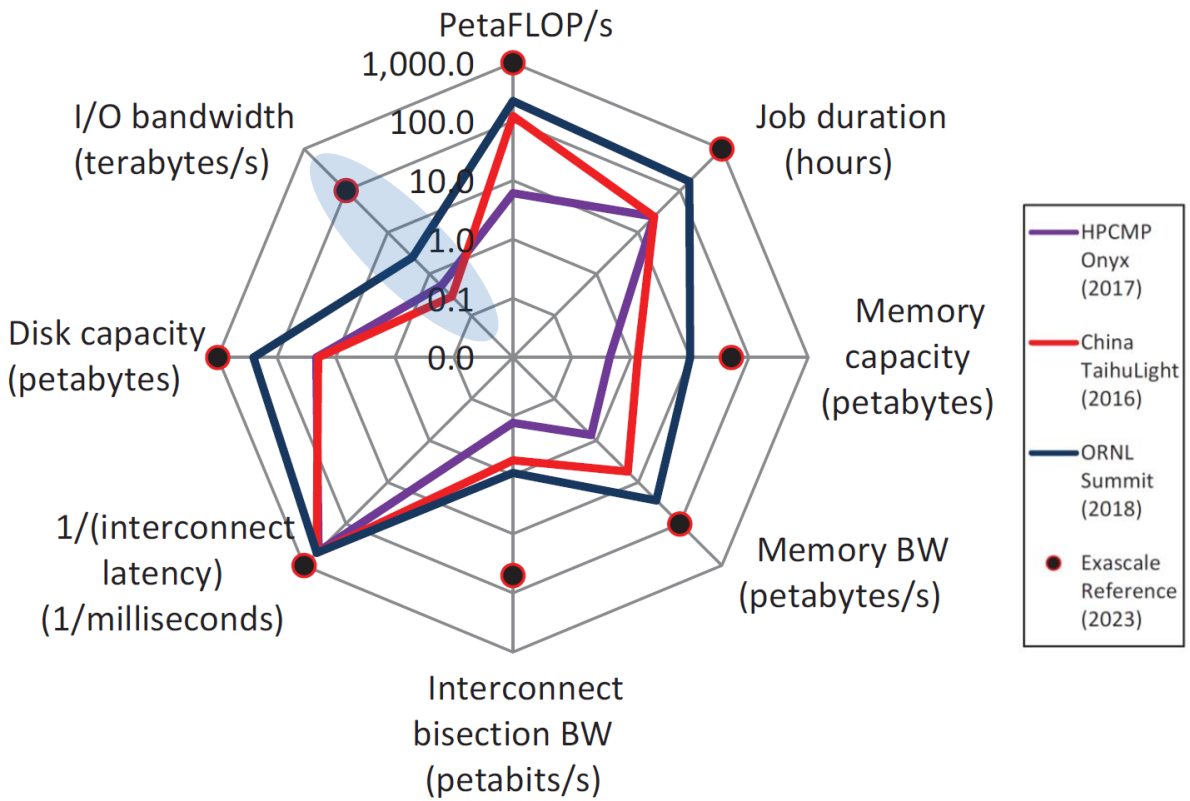
Introduction

- **Deep Learning on high performance computing (HPC) systems has unique challenges**
  - Shared (contested) I/O bandwidth
  - Distributed file systems
  - High compute density

- **From this talk you should get:**
  - The relationship between DL concurrency and I/O
  - A simple, effective method for hiding I/O on existing systems
  - An appreciation for the importance of specialized I/O systems for DL on HPC

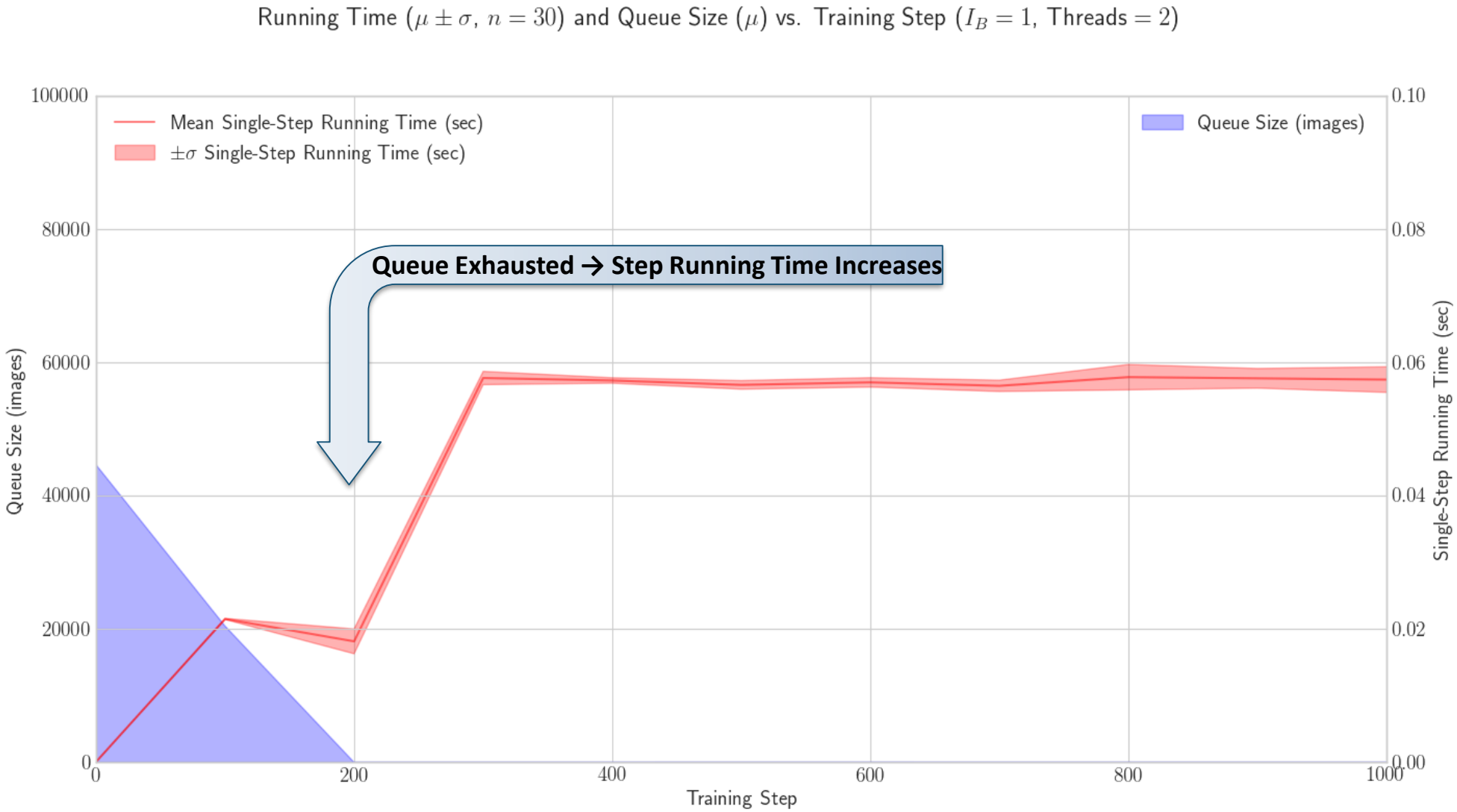
Motivation

- **Deep learning on modern HPC systems is bound by system-wide I/O**
- **TensorFlow optimization step running time on 4x P100, 20-core Power8 node is ~50ms/step, when:**
  - System under full, mostly DL, load
  - LeNet-5 training on MNIST (batch size 256)
  - Standard asynchronous data queue
  - Typically ~16 jobs running on separate nodes
- **Loading the dataset into memory yields ~17x speedup (~3ms)**
- **Instrumenting the training shows exhausted input queues**

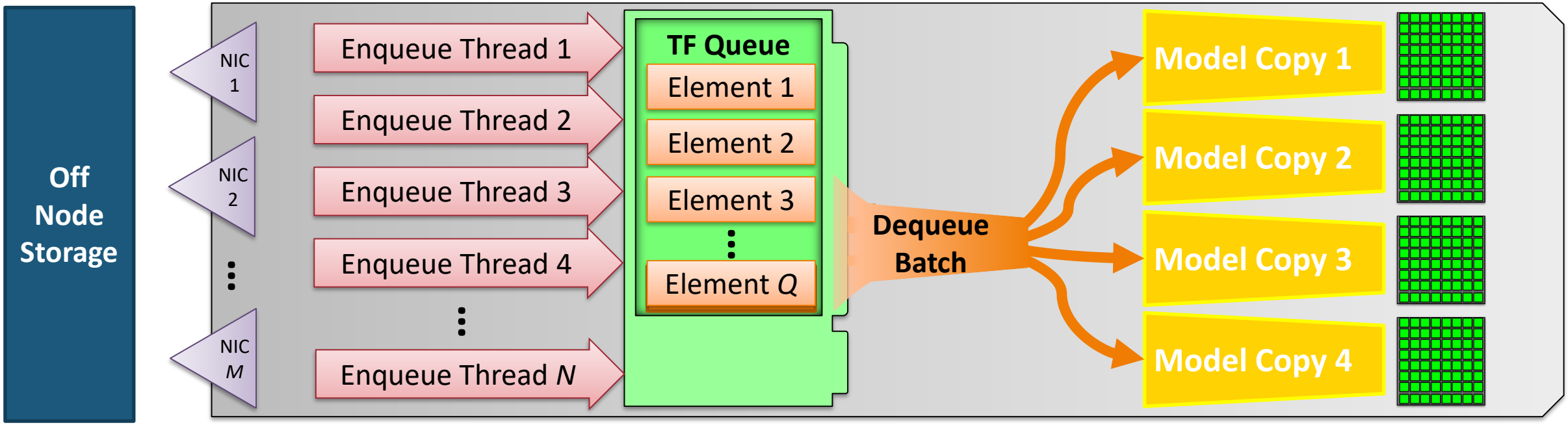


*HPC is likely to remain **I/O bound** for the foreseeable future [1]*

Impact of Queue Exhaustion

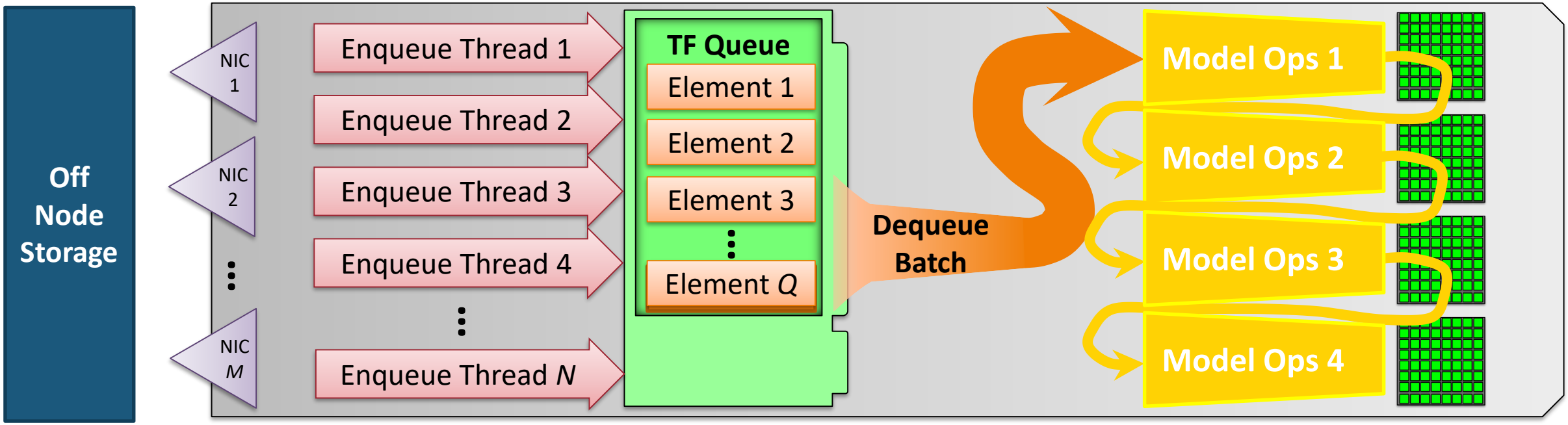


Causes of Queue Exhaustion: Data Parallelism



- **Dequeue rate exceeds enqueue rate**
- **Data parallel concurrency scheme increases dequeue rate**
  - Enqueue threads share storage I/O bandwidth
  - More model copies = more data throughput
  - Exacerbated by large data elements

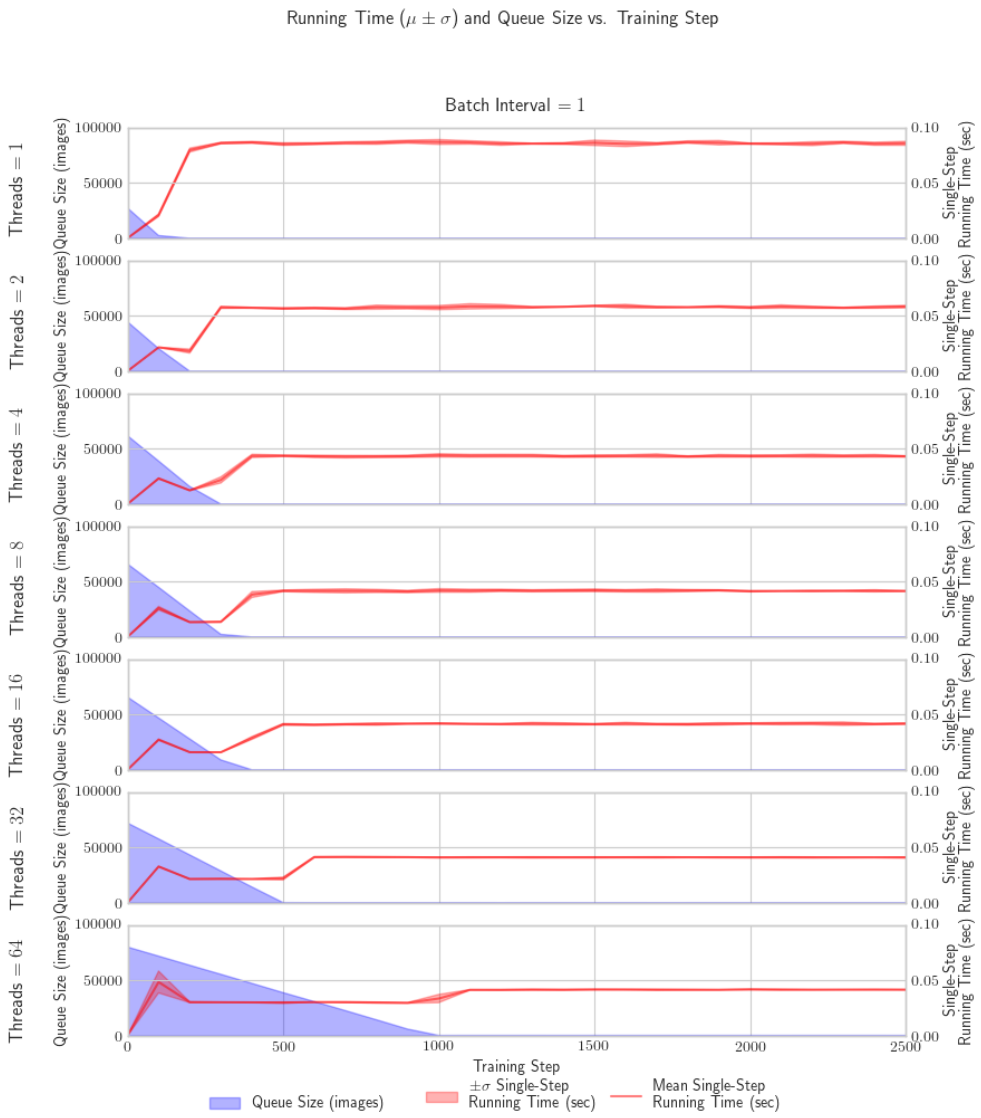
Causes of Queue Exhaustion: Pipeline Parallelism



- **Dequeue rate exceeds enqueue rate**
- **Model or pipeline parallel schemes increase dequeue rate**
  - Typically used when model won't fit on one device
  - Pipelines operations, increasing throughput

Standard Approach: Increase Thread Count

- **Enqueue threads asynchronously enqueue data element**
- **Adding more enqueue threads:**
  - Delays queue exhaustion
  - Decreases slowdown cause by exhaustion
- **We need to increase the net enqueue rate further**
  - We can't increase enqueue rate…
  - So, we must decease the dequeue rate.



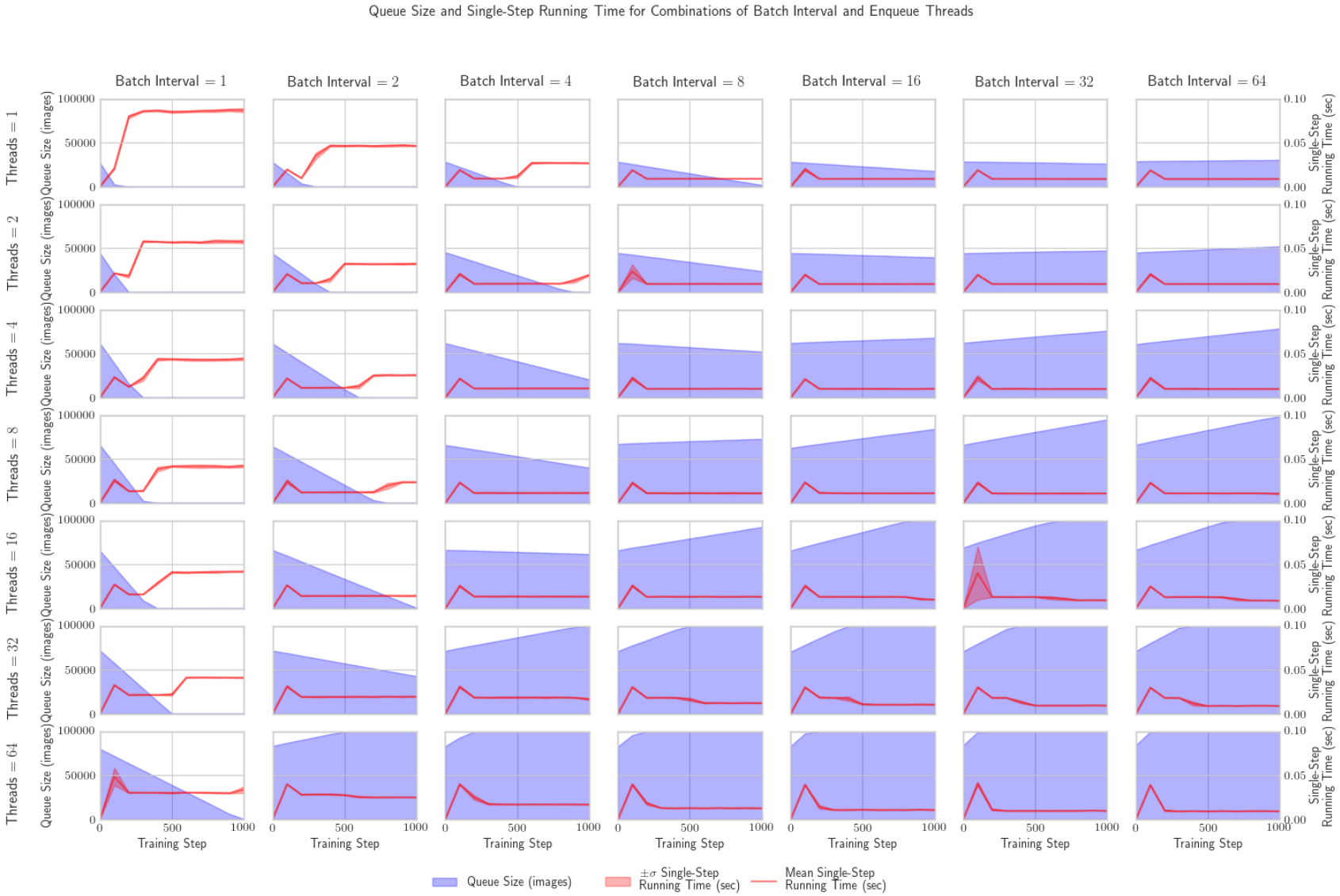
Batch Repetition

- **Artificially slow the dequeue rate by dequeuing batches less than once per step**
  - Allows the queue to fill up
  - Trivial to implement

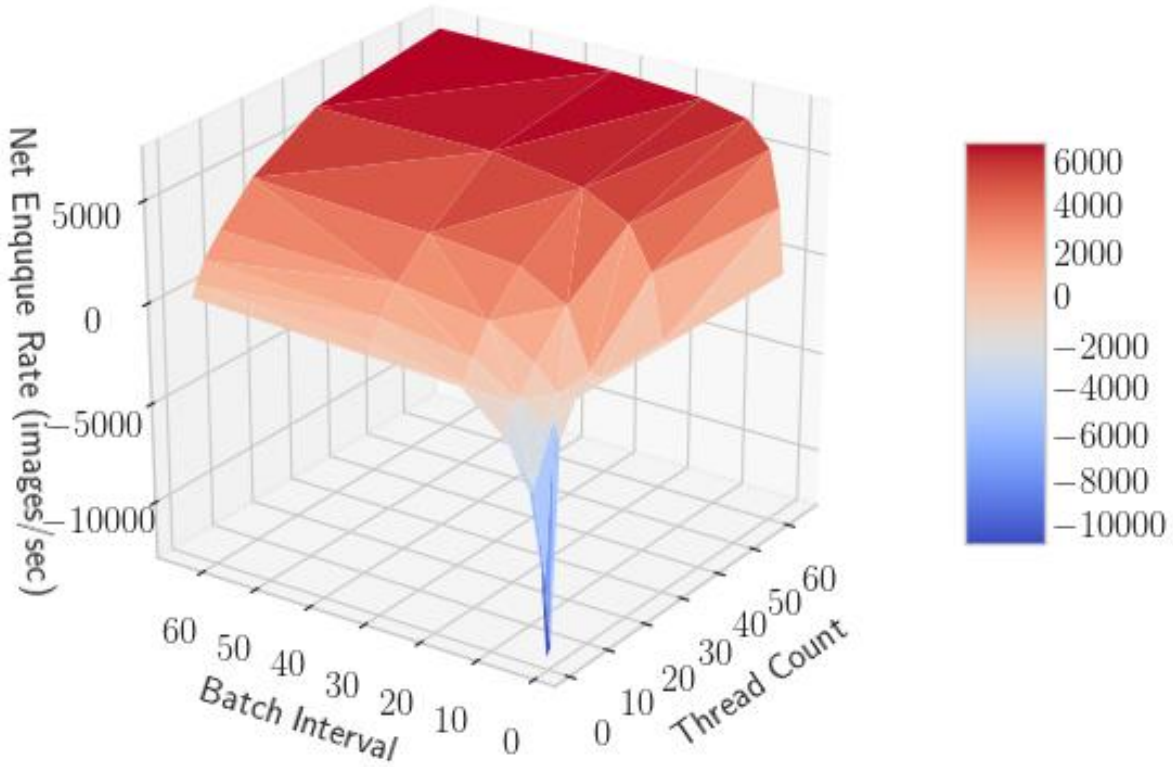
```
# If it is a batch refresh interval, refresh the batch.
if((i % batch_interval == 0) or (i == 0)):

    # Update the batch.
    train_images, train_labels = sess.run([image_batch,
                                           label_batch])

# Make a dict to load the batch onto the placeholders.
train_dict = {model.stimulus_placeholder: train_images,
              model.target_placeholder: train_labels,
              model.keep_prob: FLAGS.keep_prob}

# Run a single step of the model.
sess.run(model.optimize, feed_dict=train_dict)
```

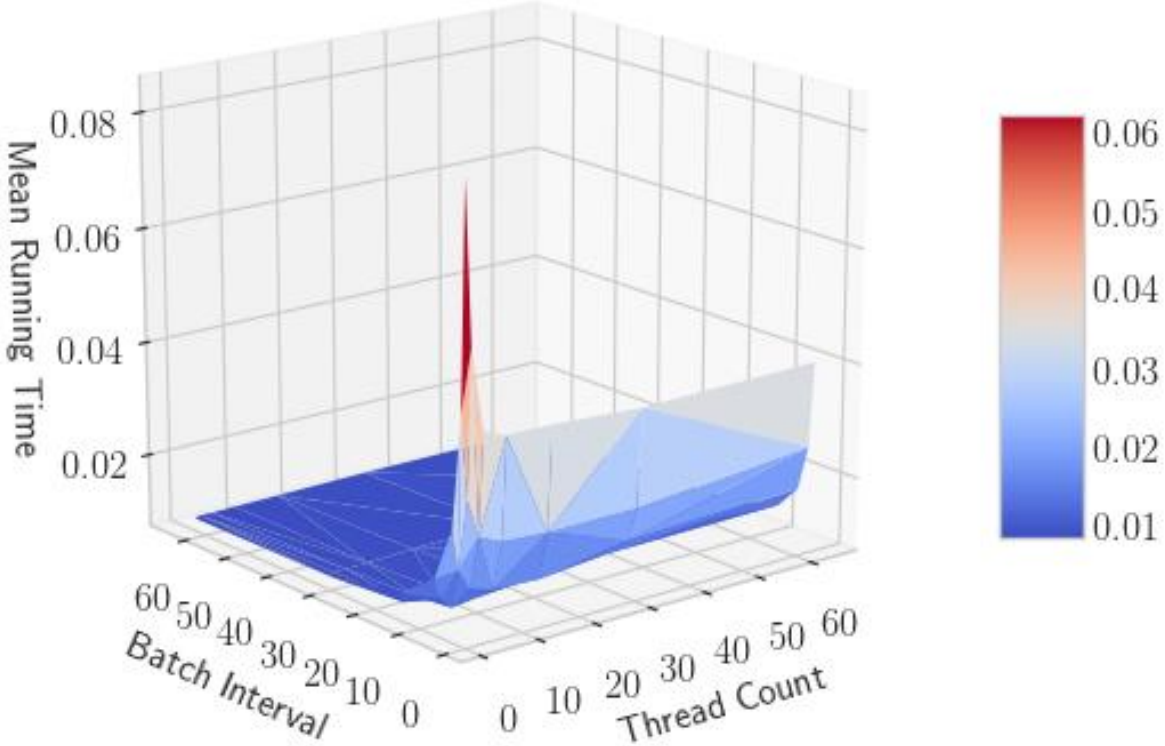
- **Repeating batches introduces new problems**
  - The model is optimized with less new data/step
  - Your epochs per second will decrease
  - Generalization is more impacted by how representative any individual batch is of the true data distribution

Batch Repetition Prevents Queue Exhaustion



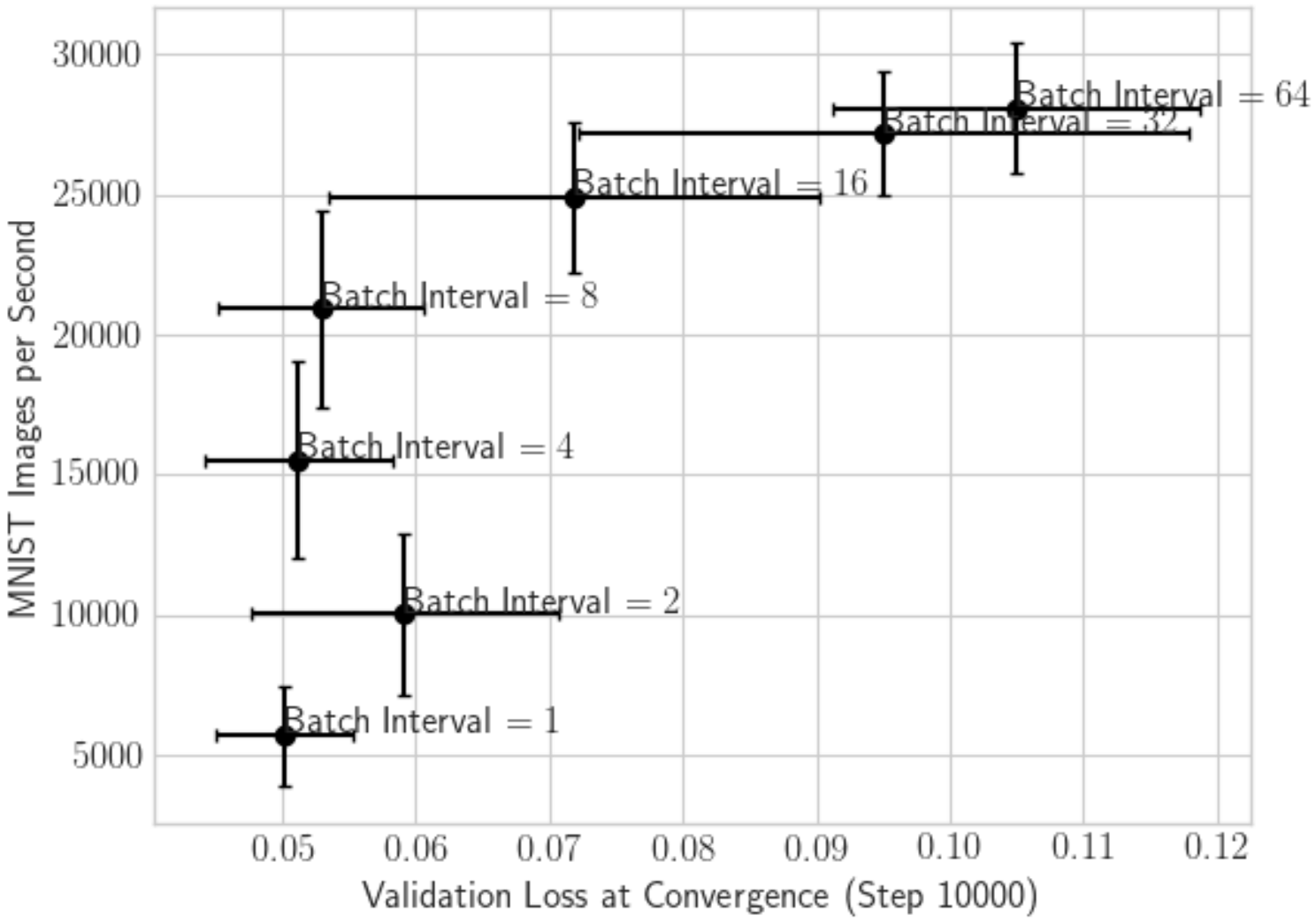
Batch Interval Impact on Net Enqueue Rate





- **Running time is inversely proportional to net enqueue rate**
  - Validates the hypothesis that training was I/O bound
  - We get diminishing returns for batch intervals >16
- **Batch intervals allow for better throughput with less threads**

Batch Interval & Model Performance



Summary

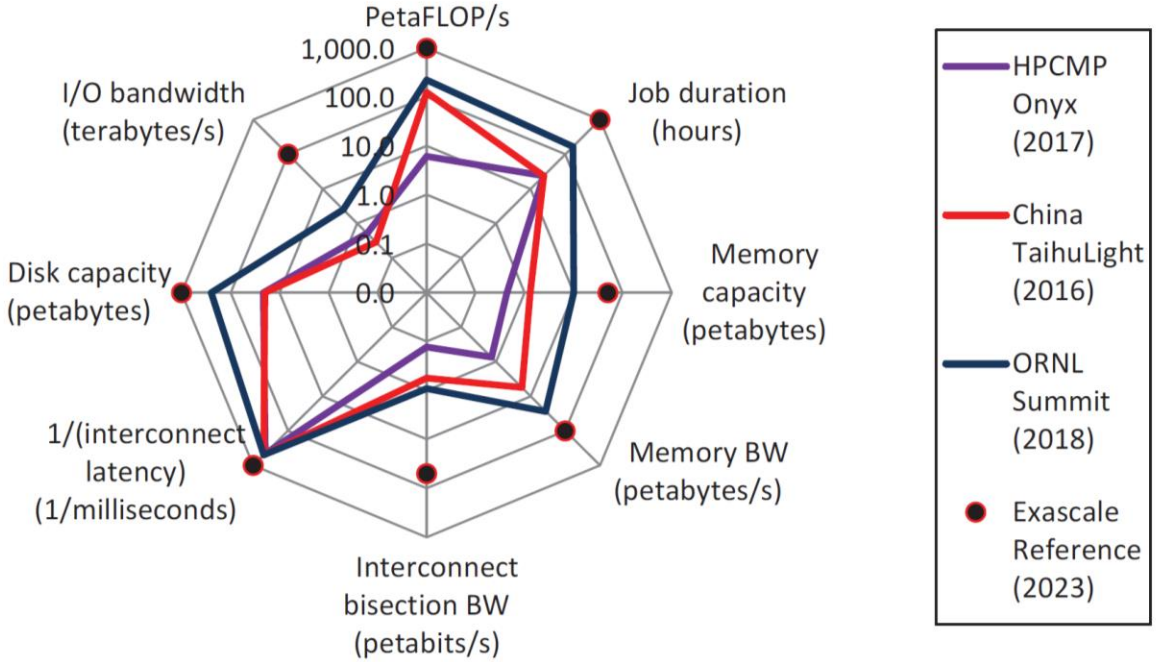
- HPC systems are structurally likely to be I/O bound for DL workloads
- Repeating batches for an interval of steps can hide I/O latency and keep the GPUs fed
- Small refresh intervals don’t impact converged optimization, but decrease runtime
- If you want to talk more, ask me about my circular data queues

HPCMP Distro A Source

Unclassified

Global Exemplar Architectures

Relative to Fastest HPCMP System



Distribution A: Approved for public release: distribution unlimited.

HPC Technology Outlook
Page-7