A GPU Inference System Scheduling Algorithm with Asynchronous Data Transfer

Qin Zhang, Li Zha, Xiaohua Wan, Boqun Cheng

University of Chinese Academy of Sciences

中国科学院计算技术研究所
INSTITUTE OF COMPUTING TECHNOLOGY, CHINESE ACADEMY OF SCIENCES

Contents

- Background
- Related Works
- Motivation
- Model
- Scheduling Algorithm
- Experiments
- Conclusion & Future Work

Background

- Deep Learning Inference
  - Small jobs, High concurrence, Low latency
  - Strong correlation with actual applications
  - Low attention compare with training
- General Purpose GPU computing
  - Widely used in Deep Learning
  - Suitable for computing intensive jobs
  - Somewhat different from CPU when it comes to scheduling jobs

Related Works

- Clipper[crankshaw2017clipper]
  - Dynamic match the batch size of inference jobs.
- LASER[agarwal2014laser]
  - Presented by LinkedIn for online advertising using cache system.
- FLEP[wu2017flep]
  - Accelerate GPU utilization ratio through kernel preemption and kernels scheduling with interruption skills.

Related Works

- [tanasic2014enabling]
  - Use s set of hardware extensions to support multi-programmed GPU workloads.
- Anakin[Baidu org]
  - Automatic graph fusion, memory reuse, assembly level optimization.

Motivation

- No model can describe inference jobs' processing mode quantitatively.
- Computation and data transmission execute serially, leading to low GPU utilization ratio.
- Question:
  - How to quantitatively analysis the relationship between concurrency and latency?
  - How to make full use of GPUs with their characteristics?

Model

- Latency: batch filling time + GPU processing time

$$L_i(n) = \max\left(F_i(n), P_{i-1}(m)\right) + P_i(n)$$

- batch filling time: batch size / concurrency

$$F(n) = n/C_t$$

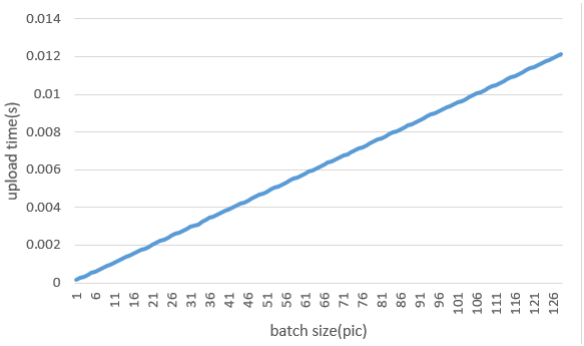
- GPU processing time: upload time + calculation time + download time, linear with data size(batch size).

$$P(n) = T_{up}(n) + T_{calc}(n) + T_{down}(n)$$

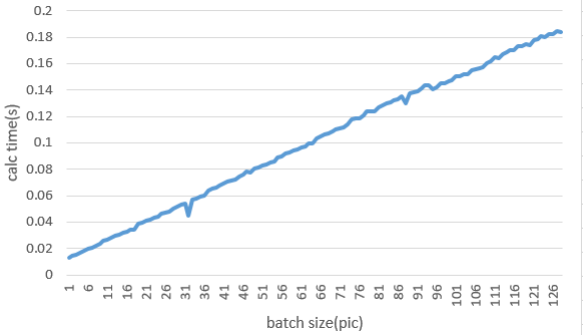
Model

- The relationship between batch size and upload time, calculation time and download time.

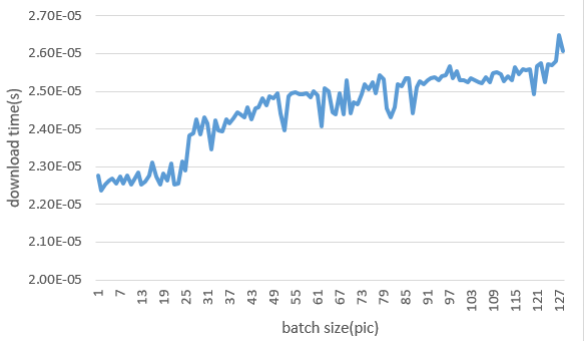
$$P(n) = kn + b$$



Upload time



Calculation time



Download time

Model

- Batch size selection:

$$n = \frac{bC_t}{1 - kC_t}$$

- Upper bound of concurrency

$$C_t = 1/k$$

- Limits of concurrency with given latency upper bound and batch size selection.

$$C_t = \frac{1}{k} - \frac{2b}{k} * \frac{1}{L}, \qquad n = \frac{L - 2b}{2k}$$

Model

- If job scheduling system supports GPU data transfer and computation asynchronous execution.

$$L_i'(n) = \max\left(F_i(n) + T_{up,i}(n), P_{i-1}(m)\right) + T_{calc,i}(n)$$

- Batch size selection:

$$n = \frac{(b_{calc} - b_{up})C_t}{1 - (k_{calc} - k_{up})C_t}$$

- Upper bound of concurrency

$$C_t = 1/(k_{calc} - k_{up})$$

Model

- Limits of concurrency with given latency upper bound and batch size selection

$$C_t = \frac{1}{k_{calc} - k_{up}} - \frac{2b_{calc}}{k_{calc} - k_{up}} * \frac{1}{L}$$

$$n = \frac{L - 2b_{calc}}{2k_{calc}}$$

- The time to upload new batch to GPU memory.

$$Up_{i+1} = Down_{i-1} - T_{down,i-1} + T_{calc,\,i} - T_{up,\,i+1}$$

Scheduling Algorithm

- Start two CUDA stream to use up the asynchronous capability of GPU.
- Select two different batch sizes to initial the model's parameters.
- Scheduling algorithm records recent several batches' GPU time, and updates the parameters.
- Calculate batch size according to real time concurrency and latency requested, and upload it to GPU memory ahead of last batch returning to host memory.

Scheduling Algorithm

- When concurrency increases
  - Batch will be filled ahead of schedule
  - Upload current filled batch
  - Update concurrency
  - Select new batch size with new concurrency, the remaining time of current processing batch and the GPU processing time of newly uploaded batch
  - Scheduling System adapts to new concurrency dynamically.

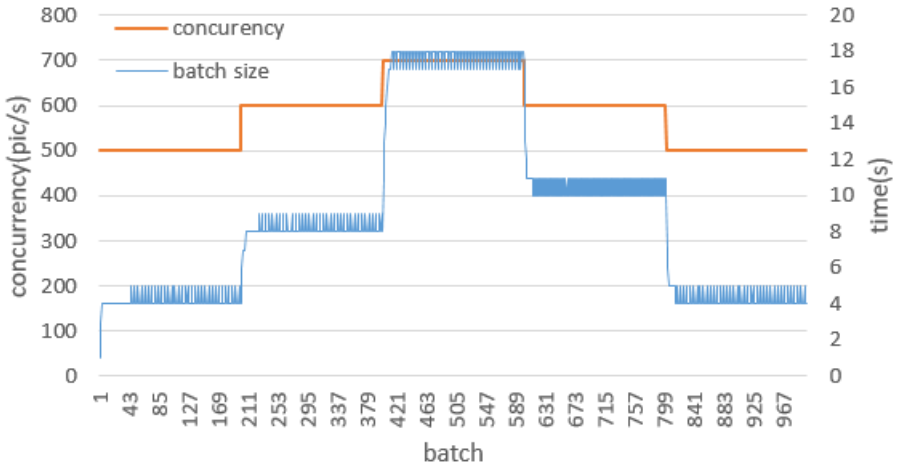
Scheduling Algorithm

- When concurrency decreases
  - Batch will not be filled when it's time to upload it.
  - Force the completion of the batch filling phase and upload the smaller batch.
  - Update concurrency
  - Select new batch size with new concurrency and the calculation time of newly uploaded batch
  - Scheduling System adapts to new concurrency dynamically.

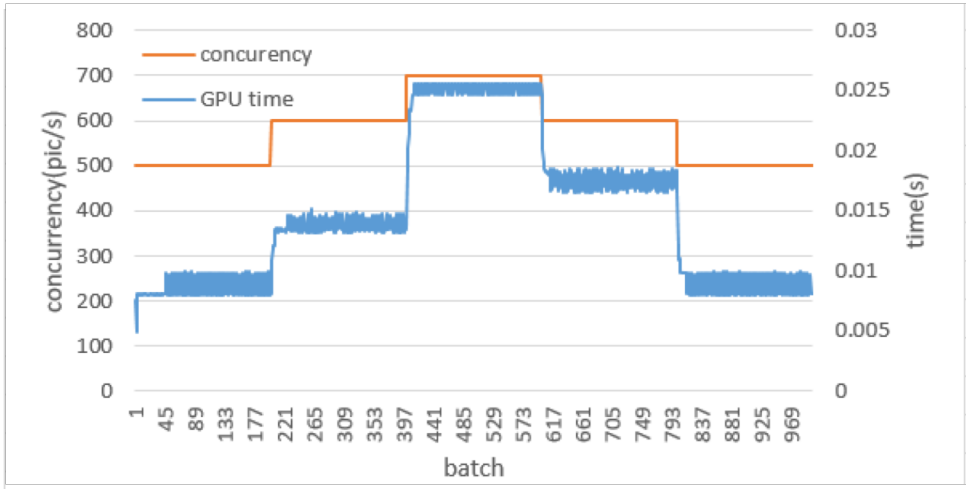
Experiments(environment)

CPU	Intel(R) Core(TM) i5-8600 CPU @ 3.10GHz
Memory	32GB
GPU	GTX 1080Ti
GPU memory	11GB
Operation System	Ubuntu 16.04
Platform	PyTorch 1.0.0
Model	ResNet-50
Dataset	CIFAR10

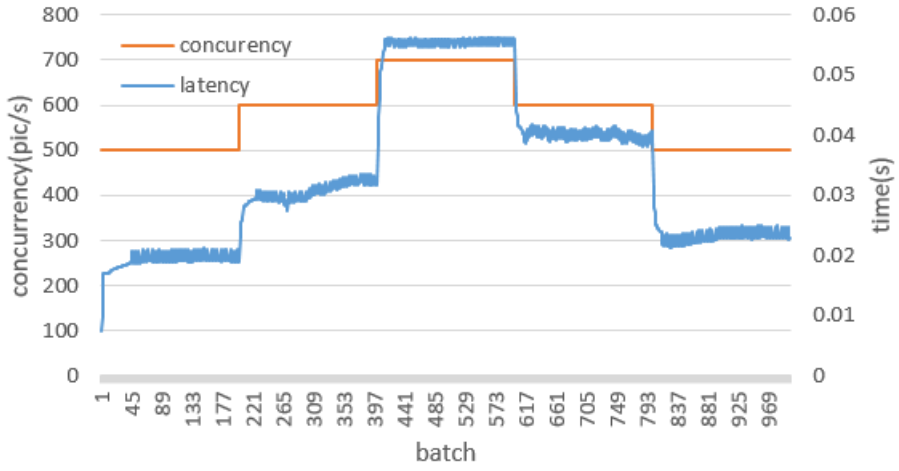
Experiments(concurrency changing)



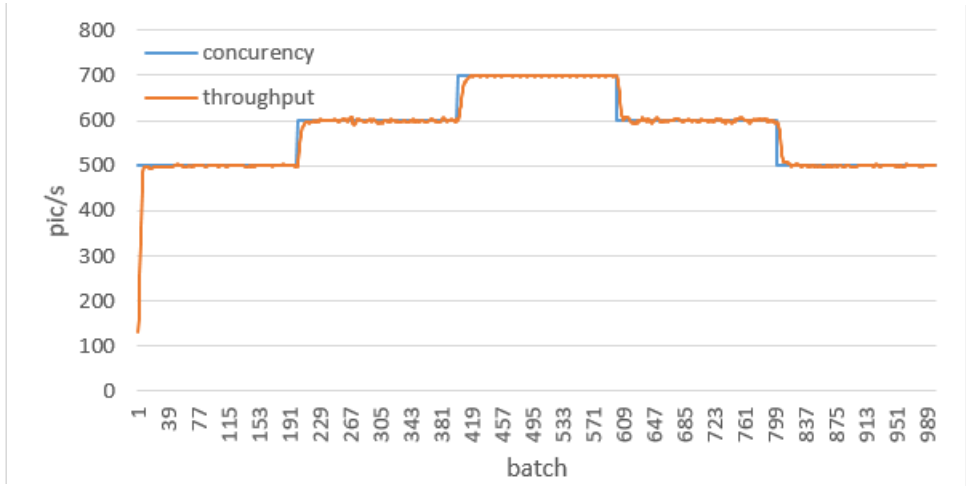
batch size



GPU processing time

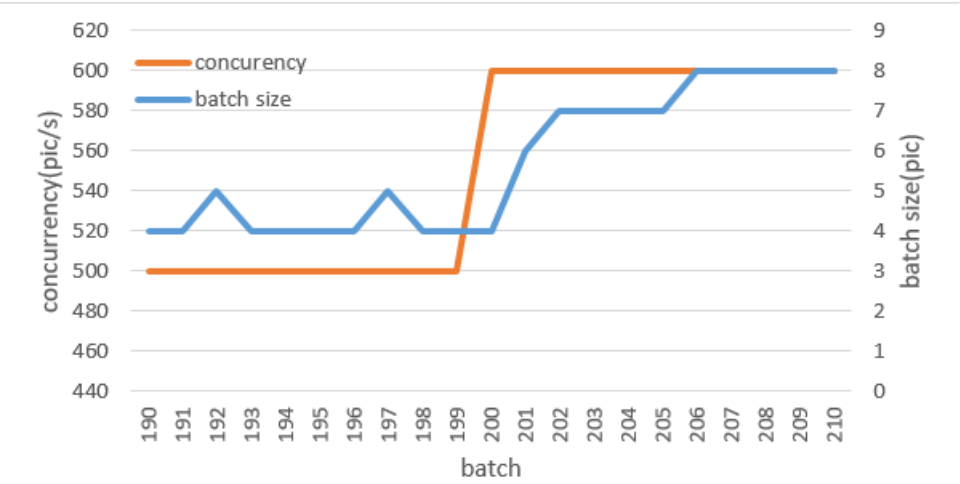


latency

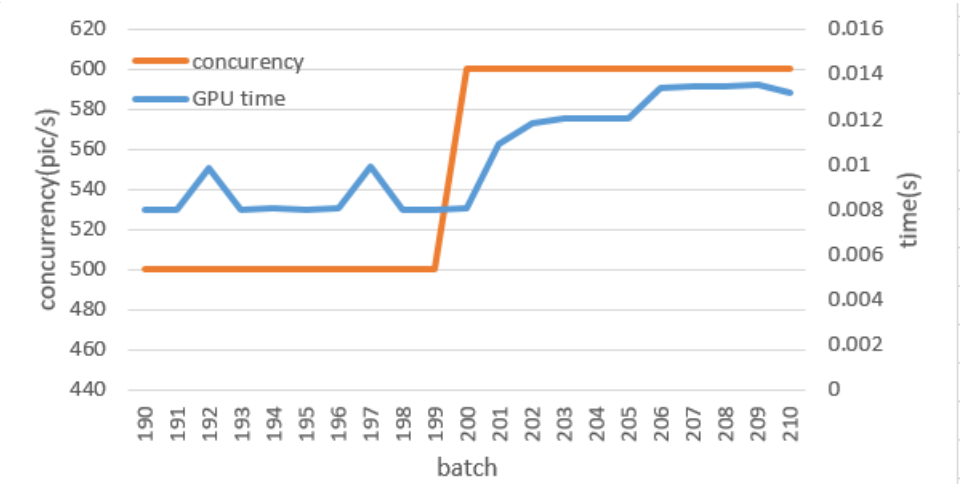


throughput

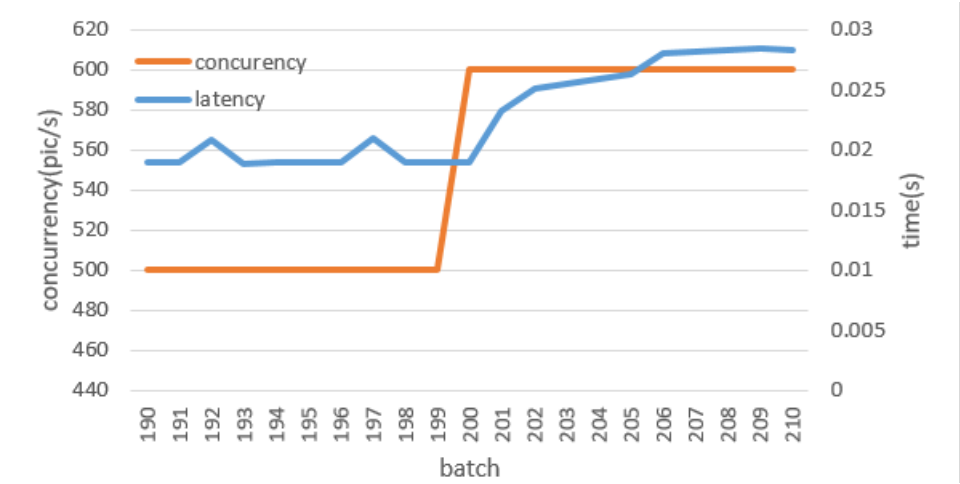
Experiments(concurrency increasing)



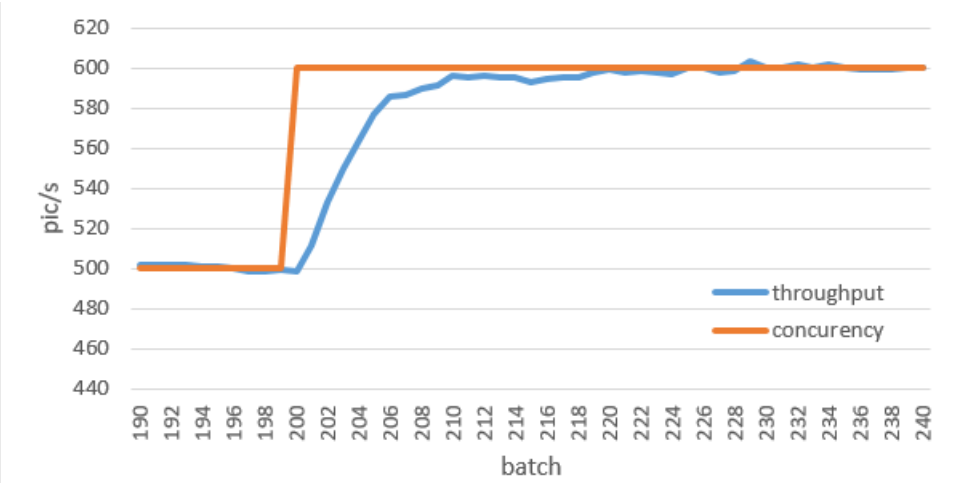
batch size



GPU processing time



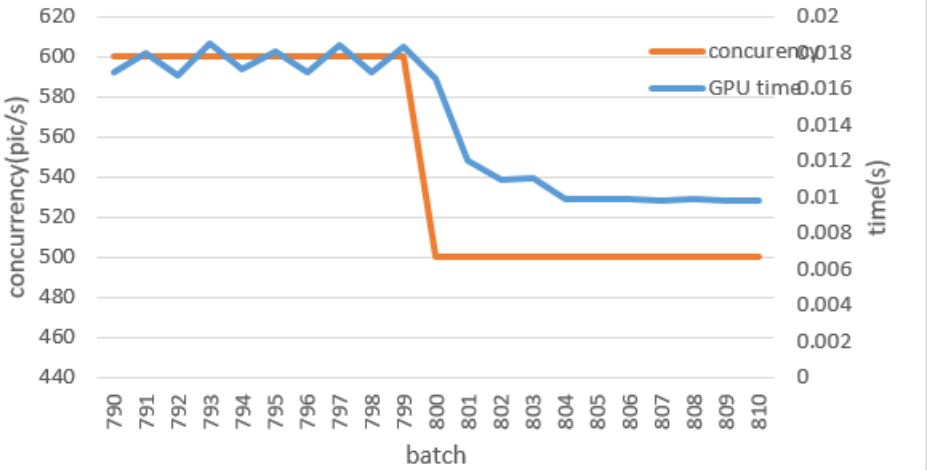
latency



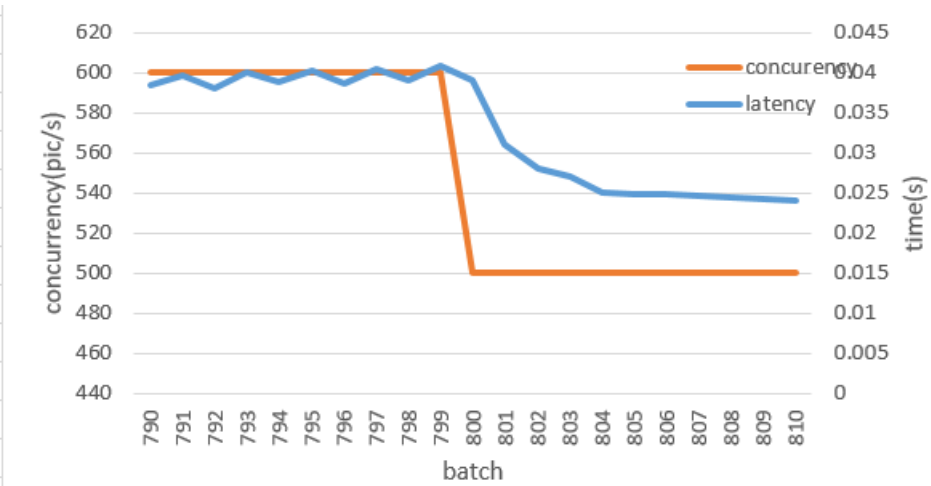
throughput

Experiments(concurrency decreasing)

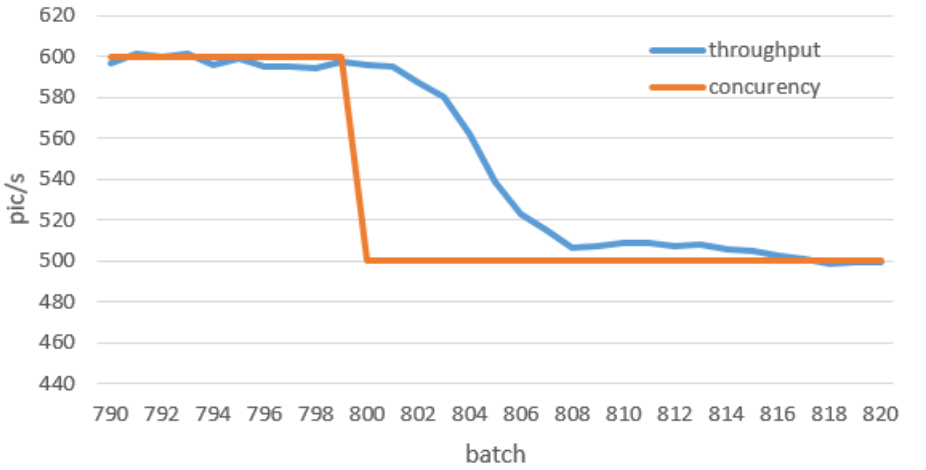
batch size



GPU processing time

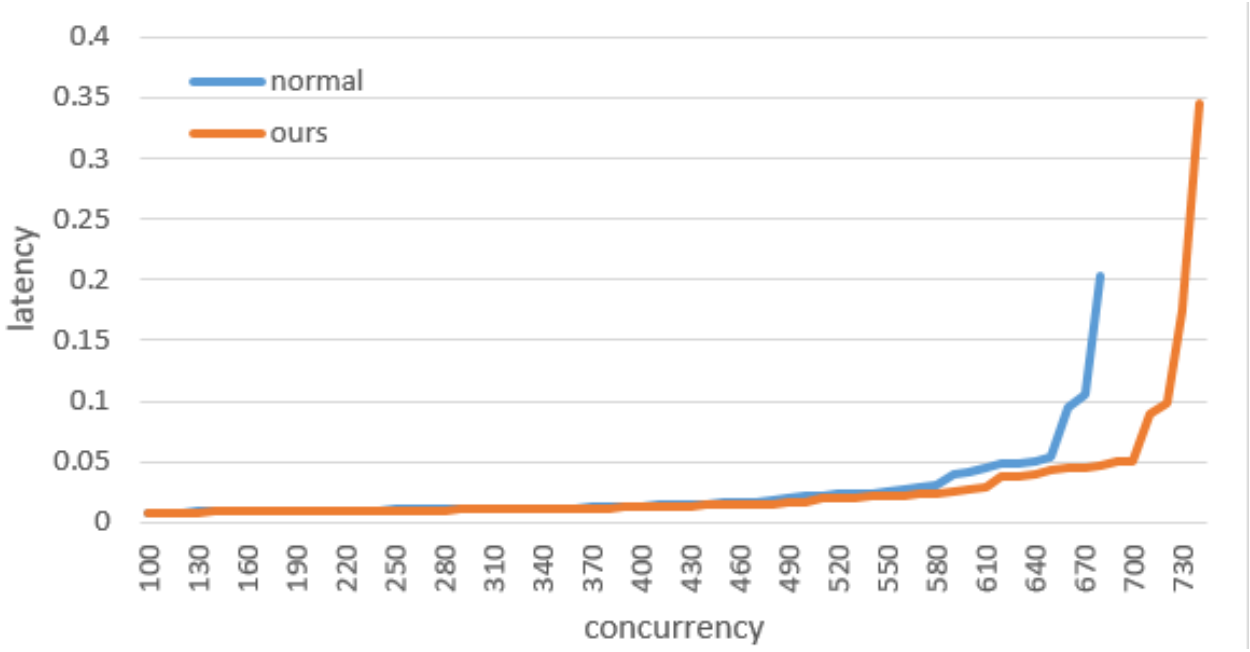


latency



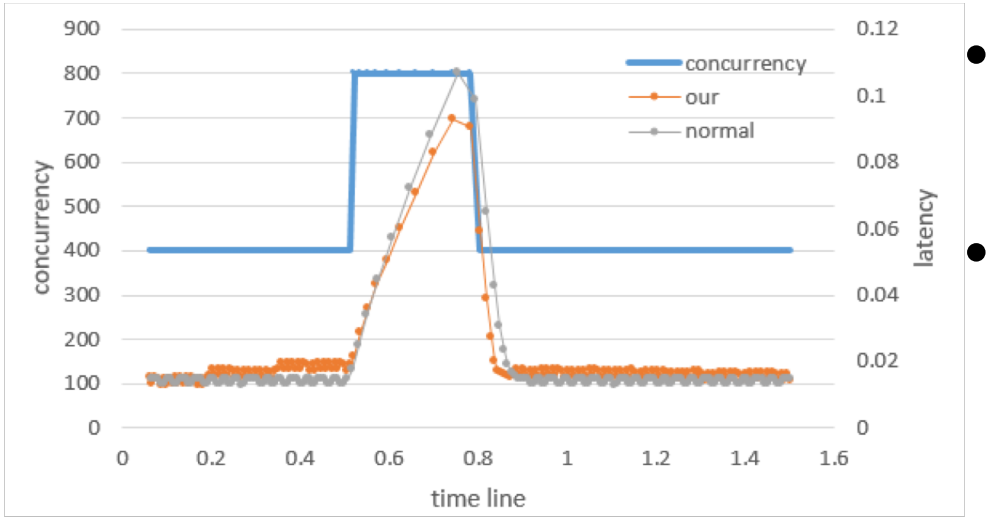
throughput

Experiments(concurrency-latency)

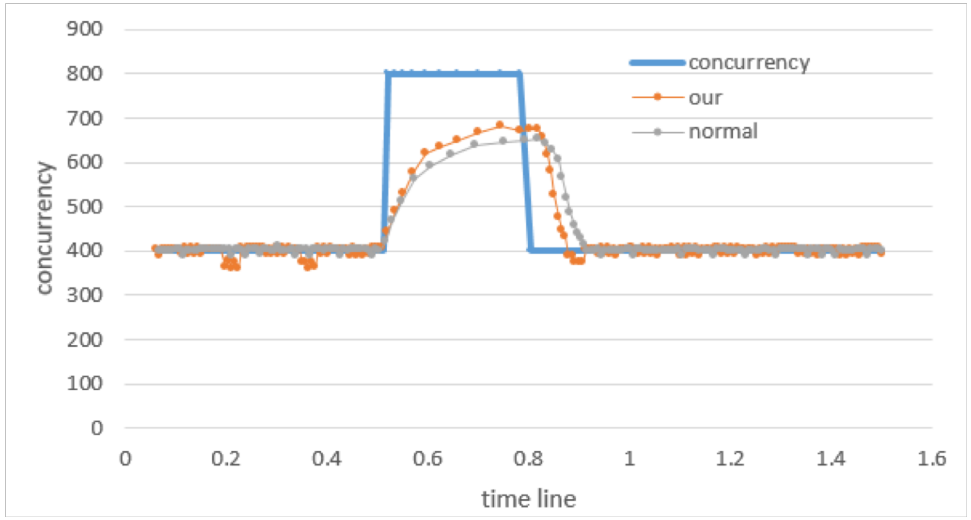


- Latency surges when concurrency increases under baseline and our scheduling algorithm.
- Our algorithm can slow down the increase of latency evidently(The larger the concurrency, the more obvious the effect).

Experiments(peak clipping)



latency



throughput

- Double the concurrency and keep 0.3s.
- Our algorithm can reduce the peak of batch size and job latency.
- Our algorithm can clip the peak of concurrency and smooth the throughput.

Conclusion

- Improve the processing capacity of the system by 9%.
- Reduces the latency by 3%-76% under different concurrency.
- Reduce the peak of latency by 16% when concurrency bursts from 400pic/s to 800pic/s for 0.3 second.
- Clip the peak of concurrency and smooth the throughput.

Future Work

- Is this method feasible in distributed system which have network communication delay?

- In deep learning training, we can also increase GPU utilization rate by making data transfer and computing asynchronously.

Q & A

Thanks!