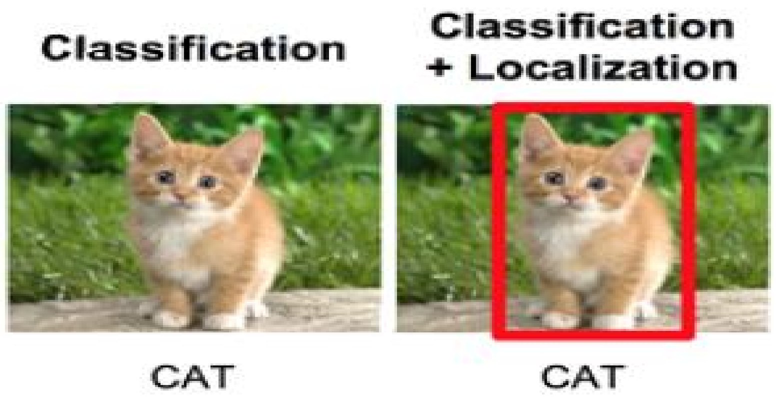
CheckFreq: Frequent, Fine-Grained DNN Checkpointing

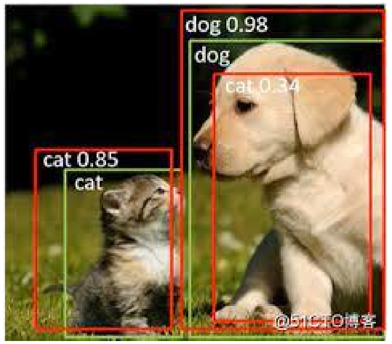
Jayashree Mohan, UT Austin; Amar Phanishayee, Microsoft Research;
Vijay Chidambaram, UT Austin and VMware research
FAST 2021

2022.05.23
Presented by Chu Xin
chuxin@dankook.ac.kr

Deep Neural Networks (DNNs)

- **DNNs are widely used (Classification , Object detection , Language Translation)**





DNN Checkpointing

- Due to the huge amount of data and model size for training DNNs, it usually takes tens of hours or even days to train a large DNN model.
- For the sake of training speed, the latest model parameter updates are stored in the GPU cache
- When there is an abnormality during the training process, or the training machine hangs up, the system has to start the training from scratch, wasting time and money

Solution: People periodically back up the intermediate state of the model in training to disk after certain batches of training are completed (checkpointing)

DNN Training

- **DNN training is compute-intensive and time-consuming**

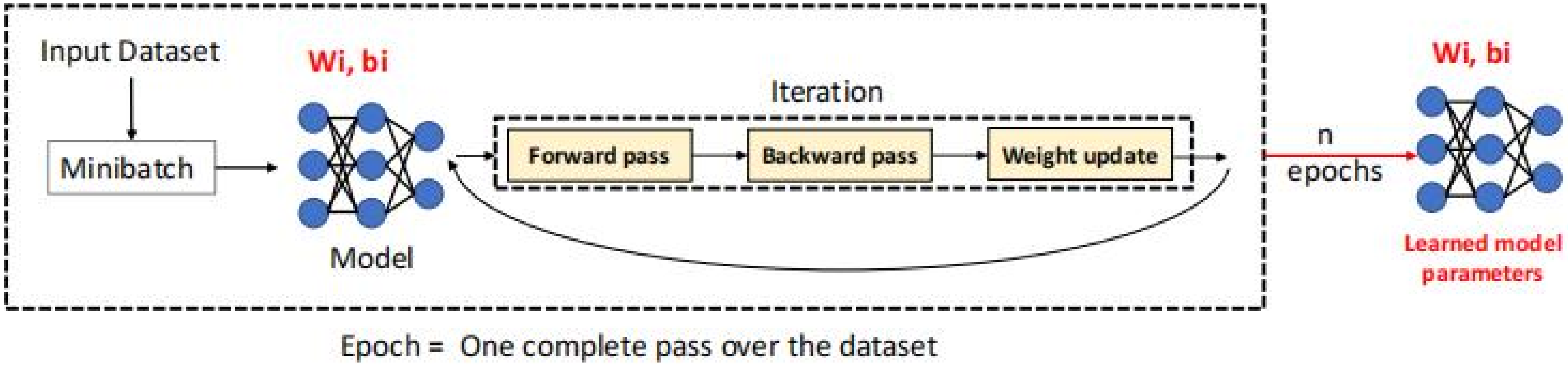
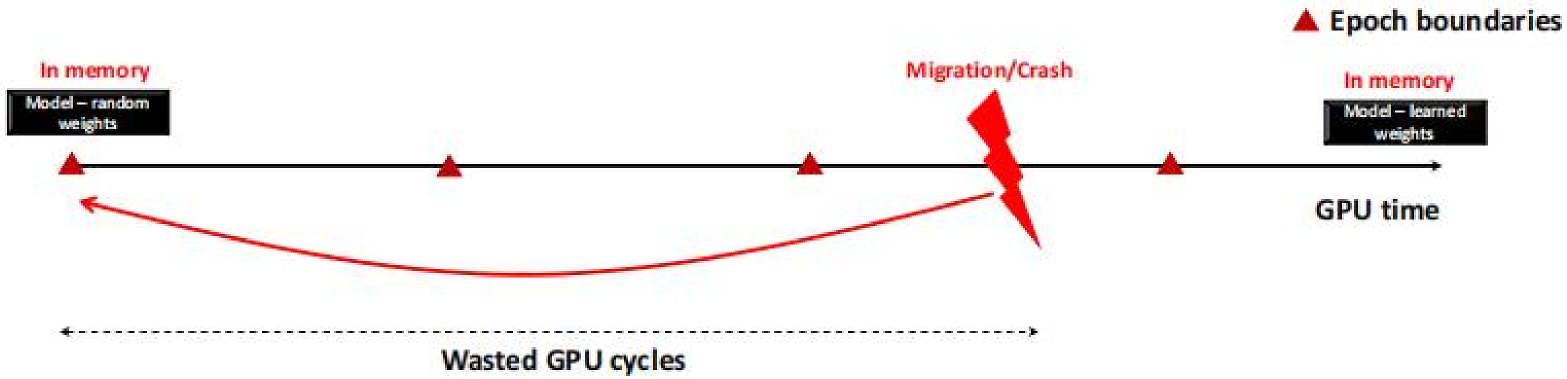
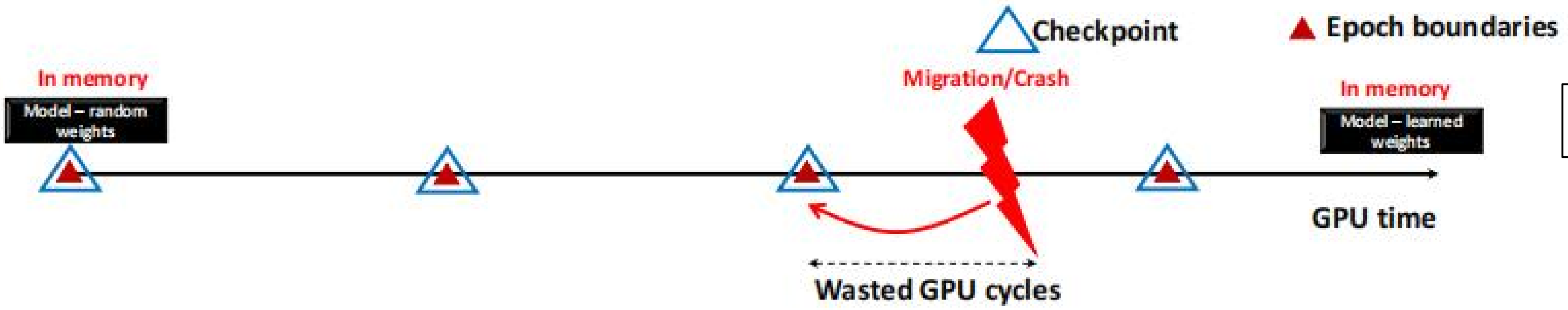


Image is from : https://www.usenix.org/conference/fast21/presentation/mohan

DNN Checkpointing



No Checkpoint



Use Checkpoint

DNN Checkpointing

➢ **The most common checkpointing mostly adopts synchronous mode**

1. When the model has finished training the nth batch,
2. The framework needs to pause the training of batch n+1
3. Then the model in the GPU's cache is synchronously flushed to the local or remote disk.
4. When the model checkpoint is completed, the training of batch n+1 can continue

DNN Checkpointing

➢ **The most common checkpointing mostly adopts synchronous mode**

1. When the model has finished training the nth batch
2. The framework needs to pause the training of batch n+1
3. Then the model in the GPU's cache is synchronously flushed to the local or remote disk.
4. When the model checkpoint is completed, the training of batch n+1 can continue

Ineffective

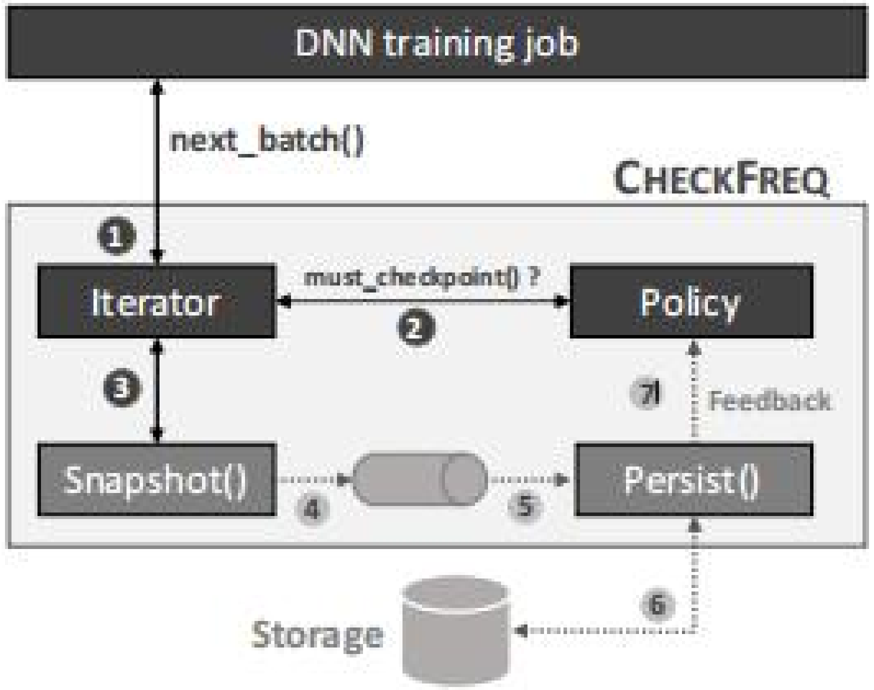
✓ Need fine-grained, iteration-level checkpointing

CheckFreq

- **Frequency: How often to checkpoint?**
- **Low-Cost: How to minimize the cost of a checkpoint?**
- **Invariant: How to resume correctly from a checkpoint?**

CheckFreq: Provide an automated, frequent checkpointing framework for DNN training

CheckFreq



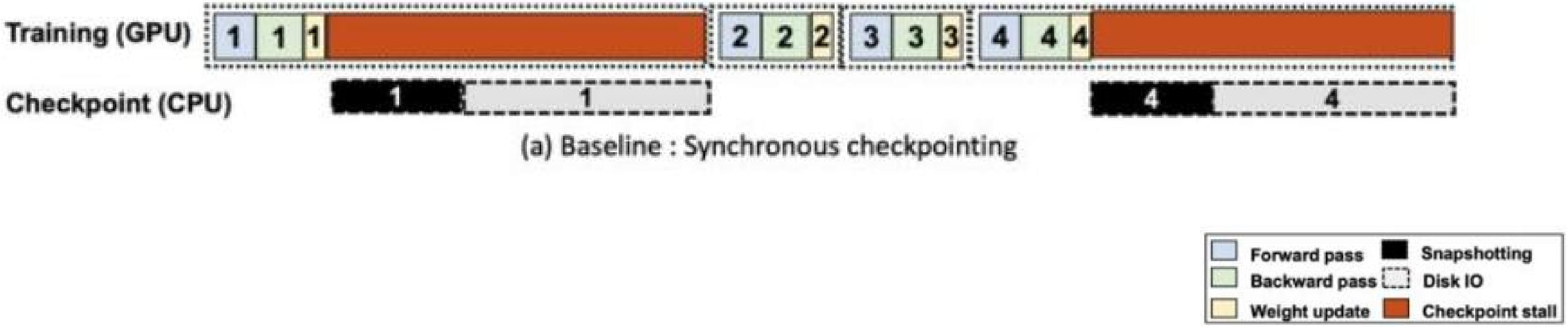
Technique	Benefits
Checkpointing mechanism (How to checkpoint?)	
2-phase checkpointing	Splits checkpointing into two phases and pipelines them carefully with compute to make checkpoints cheap
Recoverable data iterator	Maintains data invariant, allows resuming training at iteration boundaries without affecting accuracy
Checkpointing policy (When to checkpoint?)	
Systematic online profiling	Automatically determines checkpointing frequency, cognizant of model characteristics
Adaptive rate tuning	Dynamically tunes checkpointing frequency to reduce overhead due to interference

2-Phase Checkpointing

- Synchronous checkpointing introduces checkpoint stalls => Runtime overhead
- Low-cost checkpointing mechanism that is split into a pipelined snapshot() and persist() phase

Snapshot() : Serialize and copy into a user-space buffer
Persist() : Write out the serialized contents to disk

2-Phase Checkpointing



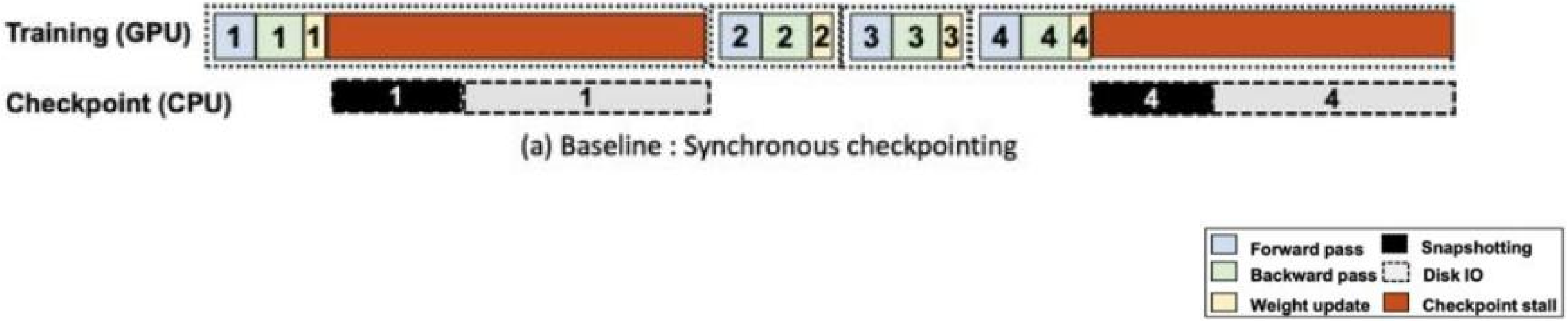
(a) Baseline : Synchronous checkpointing

- Assuming that the first batch passes through the parameters forward and backward, and finally the weights of the model parameters are updated,

- At this time, the synchronous checkpointing process will first copy the model in the GPU cache to DRAM (called snapshotting in the figure), and then fsync to disk through the file system (called Disk IO in the figure).

- The system needs to wait for both snapshotting and Disk IO to finish before starting the second batch of training.

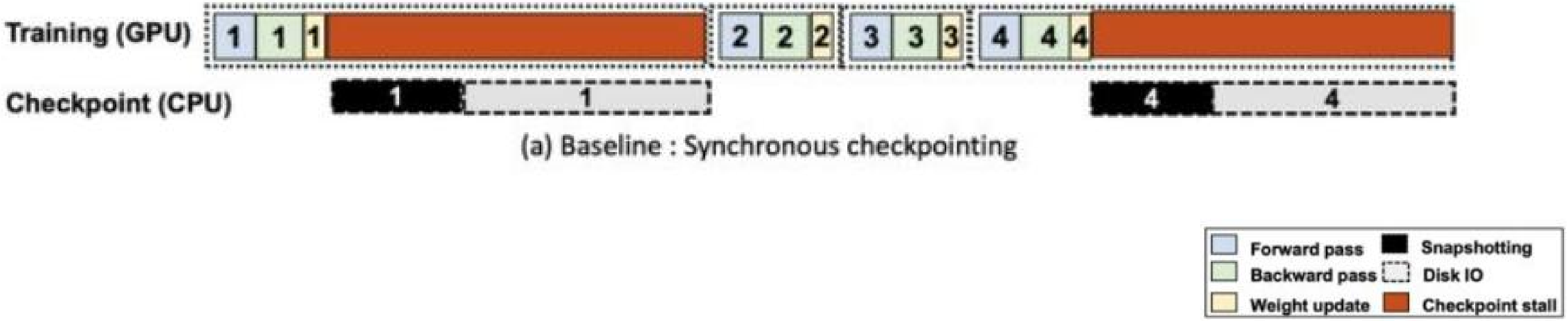
2-Phase Checkpointing



(a) Baseline : Synchronous checkpointing

- Because checkpointing needs to save the value of each parameter of the model after the first batch training, the parameters of the model cannot be changed during snapshotting.

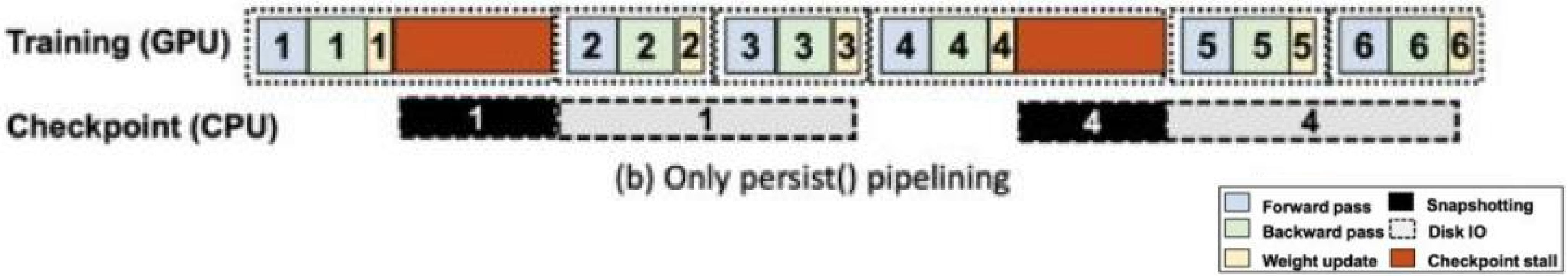
2-Phase Checkpointing



(a) Baseline : Synchronous checkpointing

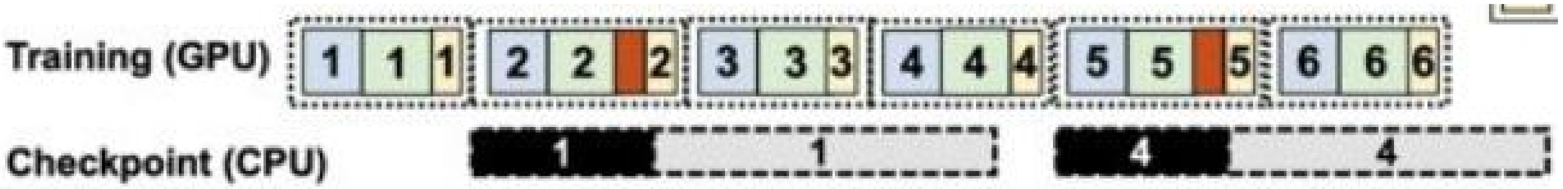
- So pause the training of the second batch, but when we copy the model data from GPU cache to DRAM(Snapshotting), this time the model has two separate copies in GPU and DRAM(CPU)
- That is to say, in the Disk IO stage, the model in the GPU can be changed and continue to be trained.

2-Phase Checkpointing

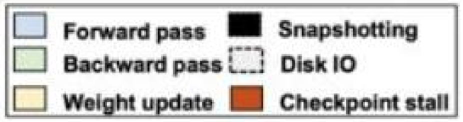


So : training is only paused during snapshotting, and the second batch can be trained in parallel while the system copies the batch 1 model in DRAM to disk.

2-Phase Checkpointing



(c) Snapshot() and persist() pipelining

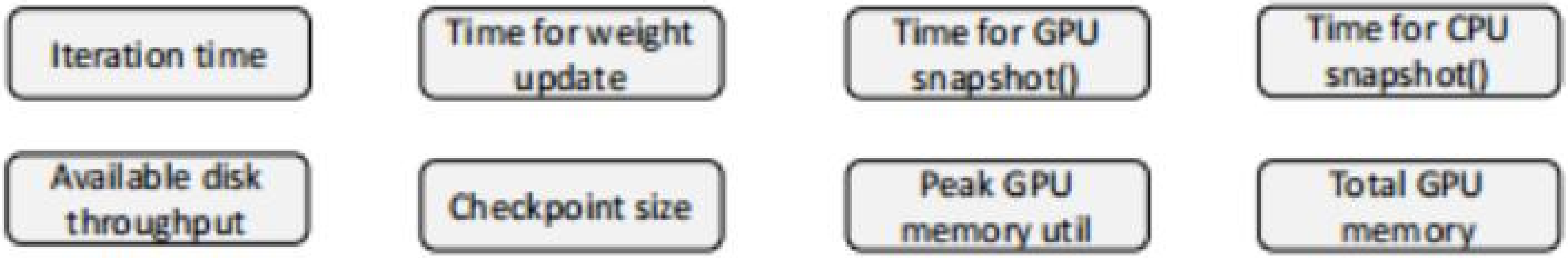


When the second batch is executed to the weight update stage, if the snapshotting has not ended, the system will suspend the training at this time

When to checkpoint?

- **Systematic Online Profiling**

  - CheckFreq’ s data iterator automatically profiles several iteration-level and checkpoint-specific metrics



Algorithmically determines the checkpointing frequency such that:
• Overhead due to checkpoint stalls is within the user-given limit

Experimental setup

- **Checkfreq is integrated with PyTorch**

- **Uses the state-of-the-art NVIDIA DALI data loading library to support resumability**

- **Experiments are performed on two different servers from an internal GPU cluster at Microsoft**

1. Conf-Volta : Server with eight V100 GPUs (32GiB), with a SSD
2. Conf-Pascal : Server with eight 1080Ti GPUs (11GiB), with a HDD

Experimental setup

Evaluate CheckFreq on 7 different DNNs :

- ResNet18, ResNet50, ResNext101, DenseNet121, VGG16,InceptionV3 on Imagenet-1k
- Bert-Large pretraining on Wikipedia & BookCorpus dataset

Evaluation

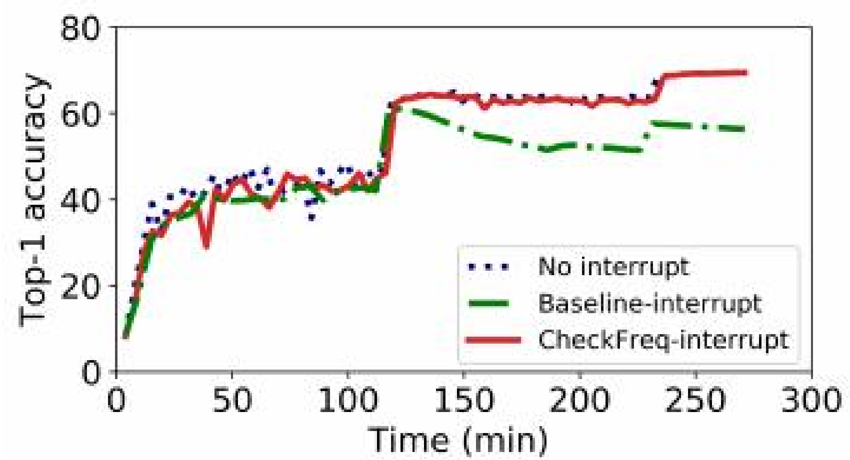


Figure 6: **Impact of resumable data iterator on accuracy.** Performing iteration-level checkpointing with baseline non-resumable data iterator violates the data invariant, results in significant loss of accuracy if job is interrupted. However, CheckFreq's iterator does not affect the final accuracy.

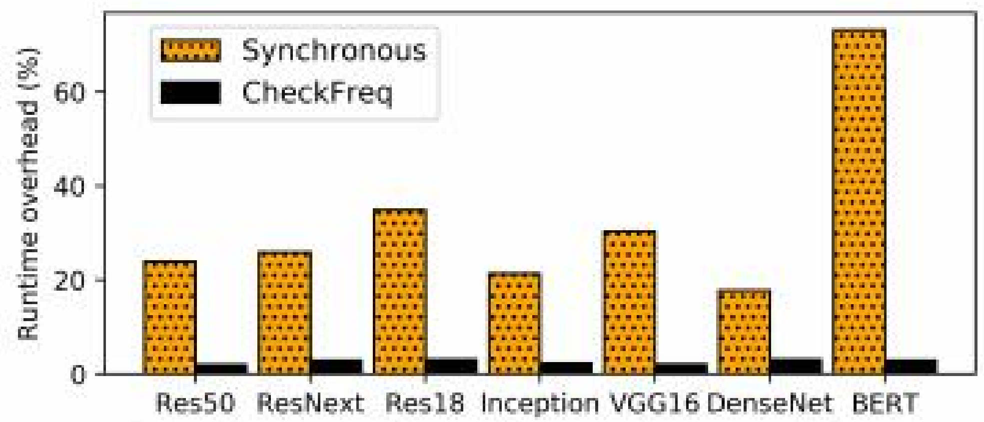


Figure 7: **Runtime overhead for various models.** At a frequency chosen by CheckFreq, synchronous checkpointing incurs upto 70% overhead while CheckFreq's pipelined checkpointing reduces runtime overhead to under 3.5%

Evaluation

Model	Recovery (seconds) Baseline	CF	Recovery (seconds) Baseline	CF
ResNet18	840	5	180	3
ResNet50	2100	24	540	8
VGG16	5700	25	1320	31
ResNext101	7080	32	1680	14
DenseNet121	2340	7	600	4
Inceptionv3	3000	27	780	42
BERT	4920	85	4500	43

(a) 1 GPU (V100) (b) 8 GPU (1080Ti)

Table 6: **Average recovery time (CF - CheckFreq)**.

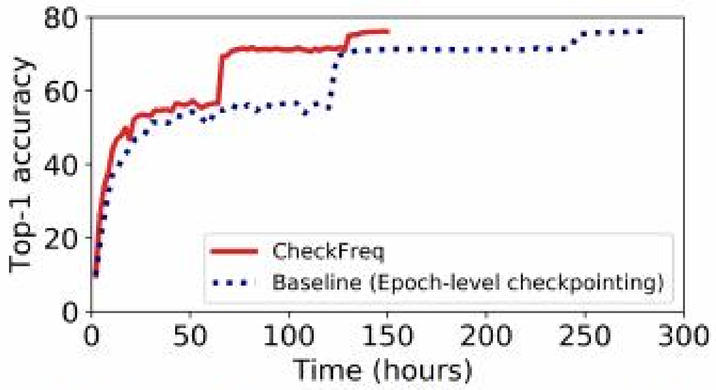


Figure 8: **End-to-end training**. We train Resnet50 using a `Conf-Pascal` GPU with interruptions every 5 hours. CheckFreq trains to state-of-the-art accuracy (76.1%) 2× faster than epoch-based checkpointing by reducing recovery time.

Conclusion

- CheckFreq provides an automatic, fine-grained checkpointing framework for DNN training

- CheckFreq allows frequent checkpointing while incurring a low cost

- When the job is interrupted, CheckFreq reduces recovery time for popular DNNs from hours to seconds