WEKA

Checkpointing for Resiliency and Performance in AI Pipelines

White Paper

Checkpointing for Resiliency and Performance in AI Pipelines

Introduction

Today's organizations are increasingly tasked to bring things to market quickly and predictably. For organizations to be successful, there needs to be an urgency for agility, transparency, and a recognition that as requirements change, the supporting architecture for large initiatives must be enhanced to keep pace.

As AI/ML is evolving, and significantly expanding in the enterprise space, and that means that what were once research lab level requirements have evolved into enterprise deployments that must meet Service Level Objective (SLO) commitments for time to deploy, completion of epochs, and resiliency for different failure events. The technique that has become dominant to assist with these SLOs in AI/ML is checkpointing. Checkpointing allows you to save a 'journal' of a training epoch at a variety of points and then use that saved point for a number of purposes.

Why Checkpoint?

On GPU hosts, learning rate losses can vary for a number of reasons. Simple errors such as omitting a data shuffle in a data preparation stage, or an unexpected, failed code update can set a project back hours or even days if not caught in time. Hardware failures can also force a recovery to a known good state of a model. GPUs in a cluster whether on-prem or in the cloud can have outages. In the cloud, because the GPU hardware tends to be managed as a service, you may have GPUs removed and others added into clusters ad-hoc depending on the maintenance and update schedule of the cloud provider.

You can recover from a failure using a checkpoint without having to go back to the beginning of the training cycle, and you can also clone models out to test where deviations in precision have occurred without having to wait for a full training epoch to finish. This enables developer productivity by sharing in the training that has occurred up to that point without having to restart from the beginning.

Depending on your needs, you can checkpoint as granularly as needed after every embed or relatively infrequently by queueing up a longer chain of embeds and then dump a larger set of checkpoints all at once. The tradeoff is in how far back you need to go to recover the checkpoint and restart the training. The farther back you go, the more training needs to be re-run to bring the model back to the point where the failure occurred. In an era where a small cluster of 10 NVIDIA H100 systems can cost as much as $1000 per hour to run, being able to recover your model as quickly as possible without having to re-run large amounts of training to bring the model up-to-date is paramount.

The current industry trend with LLMs is to checkpoint as often as possible to minimize any recovery time. Aggressive checkpointing however, may introduce an impact on the data infrastructure in the form of IO bottlenecks to the data storage being used in the environment. This bottlenecking directly impacts AI researchers and developers wall clock time to develop and train AI models. In particular, any delays in checkpointing commits to storage can stall model training until the checkpoint completes.

Impact on Infrastructure

In a production environment, the storage layer has an outsized impact on the ability to checkpoint often. If the storage layer is unable to efficiently process the checkpoint writes, then the model training will be stalled until the checkpoint completes. Because checkpointing is a latency sensitive operation, when a storage system struggles to run multiple jobs (Figure 1) at the same time in different phases, such as concurrent reads + writes, massively parallel reads and committing tuned and trained models to storage at the completion of a job, the time to completion of a checkpoint may be affected. Checkpointing during a pipeline adds another layer of IO to be handled by this infrastructure. Legacy storage vendors have made great claims to demonstrate how they can do data management, scale performance, and lower costs, but in many cases, a product designed for legacy file services can struggle to keep pace with the diverse high-performance read, write, and archive requirements an AI pipeline demands.

How WEKA Helps

The WEKA Data Platform is specifically designed for this use case with a high-performance file system that services AI/ML workloads with open standards. WEKA's ability to run diverse concurrent workloads distinguishes it from other legacy storage by providing industry leading metadata performance, scale out read and write performance on large and small file operations, as well as operational efficiency when storing billions of small files.

Many elements of a workflow require each step to complete before you can reach the end of the job. With checkpointing specifically, training on a model cannot complete until the checkpoint has finished writing. If a checkpoint requires performance that exceeds a legacy storage system's capability, job runtime can slow significantly and the researcher starts asking the question again: "why isn't it running as fast as I need?".

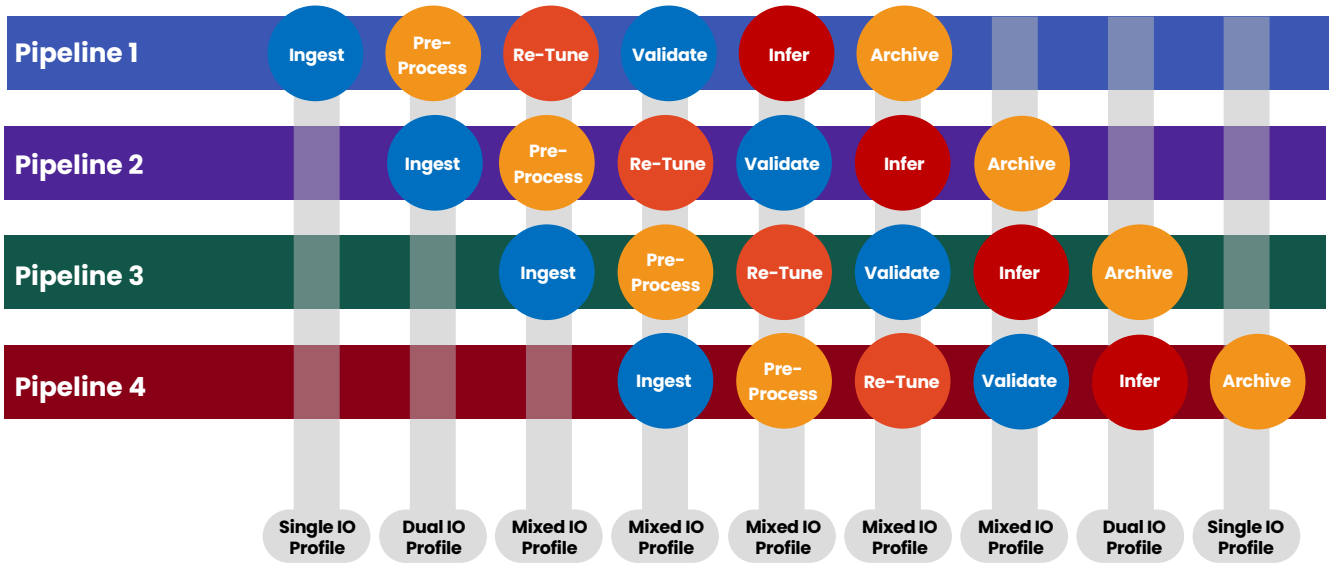

FIG. 1

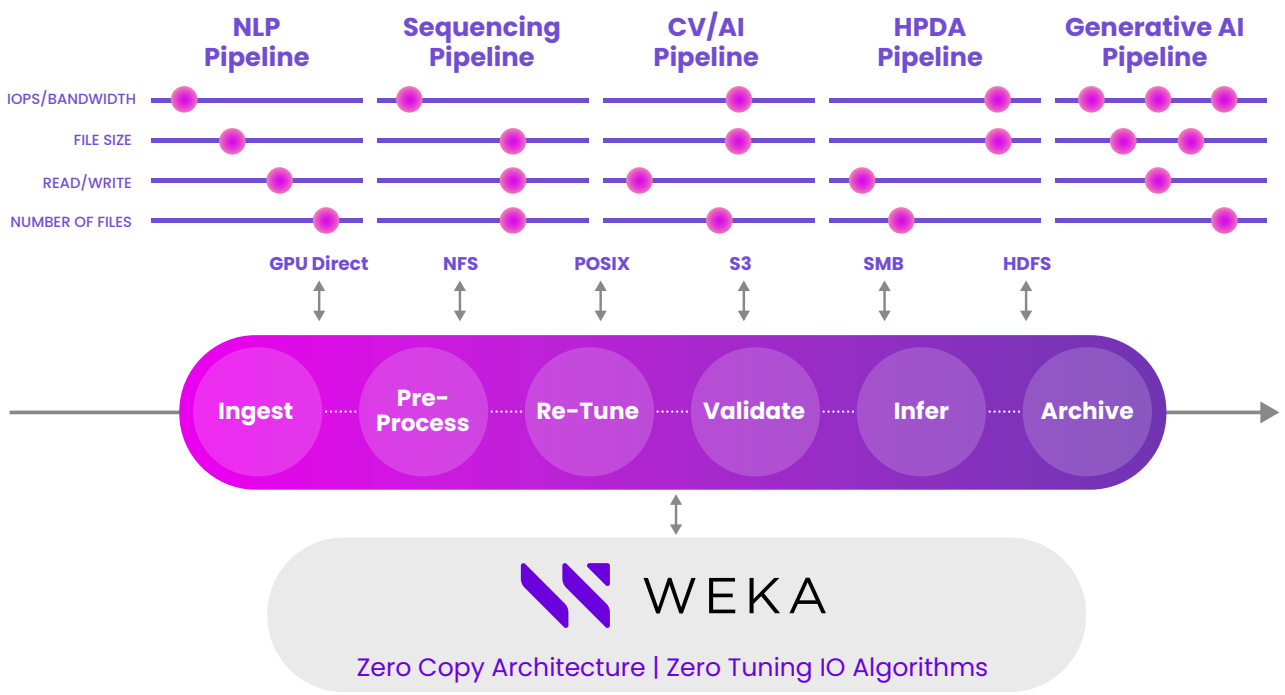

FIG. 2

The WEKA Data Platform's services are specifically designed to provide concurrent operations for all your workloads (Figure 2), whether NLP, Computational Chemistry for Genomics, Image Recognition, Financial Analysis, Generative AI, or something that has yet to be realized. With the ease of creating data experiments on as small an environment as your laptop, imagine the concurrency of tens or hundreds or thousands of experiments on enterprise level infrastructure. WEKA can handle this data blender with ease.

Checkpoint Performance Testing

To test checkpointing performance between WEKA and another Storage platform, we utilized a tool from NVIDIA, NeMo. NeMo is widely popular and is used as a foundational component in large numbers of AI projects, both large and small and is representative of a pipeline tool that would be used for checkpointing.

NVIDIA created the NeMo™ generative AI toolkit to help with the entire data pipeline and tool chain that a scientist uses. This provides a consistent and optimized infrastructure to enhance existing Deep Learning (DL) technology and foundation models based on the lineage of their NVIDIA GPU Container (NGC) efforts. NeMo provides a framework for training, retuning foundation models, GPU telemetry, TensorBoard observability and a rich checkpoint capability designed to accommodate different aspects of model development. If you have an AI stack running against NVIDIA GPUs, NeMo assists this workflow by providing features that allow taking a pipeline for model training all the way from POC to production. During this workflow, when failures occur, training can be reverted and restarted using checkpointing with model observability.

NeMo's checkpoint capability allows for incremental training with observability. On the system being monitored, when one of the errors mentioned earlier occurs, reverting back to the last point when training was progressing as expected by using best practices for NeMo's checkpoint recovery capabilities enables rapid recovery, translating to faster time to restart training, less idling of expensive GPUs while waiting to restart and a safety net for recovery of work, regardless of toolchain or model modality.

While NeMo's features are well fleshed out, it is just a layer of a stack and is dependent on a performant infrastructure underneath it to drive real world results. There are other AI frameworks as well that depend on checkpointing best practices to ensure a smooth and recoverable AI training workflow.

Process and Models Used

For this paper, we have built out a NeMo environment and will use it to train a FastPitch parallel text-to-speech model, an ASR speech-to-text model, as well as a larger MegatronBERT model (BioMegatron) for NLP. This shows checkpointing performance across relatively small, medium, and large language models.

To start, we will finetune a single speaker FastPitch (with alignment) model on 5 mins of a new speaker's data. We will finetune the model parameters only on the new speaker's text and speech pairs. We download the training data, then generate and run a training command to finetune FastPitch, and synthesize the audio from the trained checkpoint. Documentation of how to run a FastPitch training with NeMo including the specific model parameters used in this example and run logs is listed in the appendix of this paper.

WEKA Integration

Leveraging WEKA with NeMo is simple. WEKA can be deployed in all the hyperscale clouds, or on premises, or in a hybrid mode. Compute clients interact with WEKA via any supported protocol. POSIX, NFS, SMB and S3 are all supported as well as NVIDIA GDS. Since many applications are containerized, WEKA also provides a CSI driver to directly attach to Kubernetes pods. For NeMo, the user experience feels like being attached to a big, fast NVME disk that is capable of exabyte scale but adheres to all the standards you would expect from a POSIX resource. To a researcher it feels as if your data is as local as it would be on a workstation or laptop.

Once NeMo is configured with a WEKA POSIX client, steps for data preparation, training, tuning and model deployment are turnkey, and checkpoint usage is well documented and understood.

Test Environment

To test out checkpointing, we built a small WEKA cluster in AWS consisting of 6x i3en.2xlarge EC2 instances using the WEKA POSIX client to talk to the GPU. We then built out a separate c5n.18xlarge instance as a dedicated high-performance NFS server. The training was done on a single G4dn.8xl instance with a T4 GPU with 16GB of vRAM. These instances were all chosen to ensure that at the throughput needed, networking would not be a bottleneck. Latency was measured on the WEKA cluster using a WEKA stats function to measure round-trip latency from client to the WEKA cluster. For the NFS server, Tshark (terminal based Wireshark) line capture was used to measure roundtrip latency of the checkpoint completion.

Write latency measurement:

During the FastPitch training we followed the best practices for checkpointing and using a trace mechanism, measured how long it took to complete each checkpoint in the epoch. As shown in Figure 3, the average span between checkpoints was only 10 seconds. Notably, the WEKA data platform was able to write the checkpoints for the FastPitch model at an average of ~900us, while the NFS server was nearly twice the latency at ~1850us.

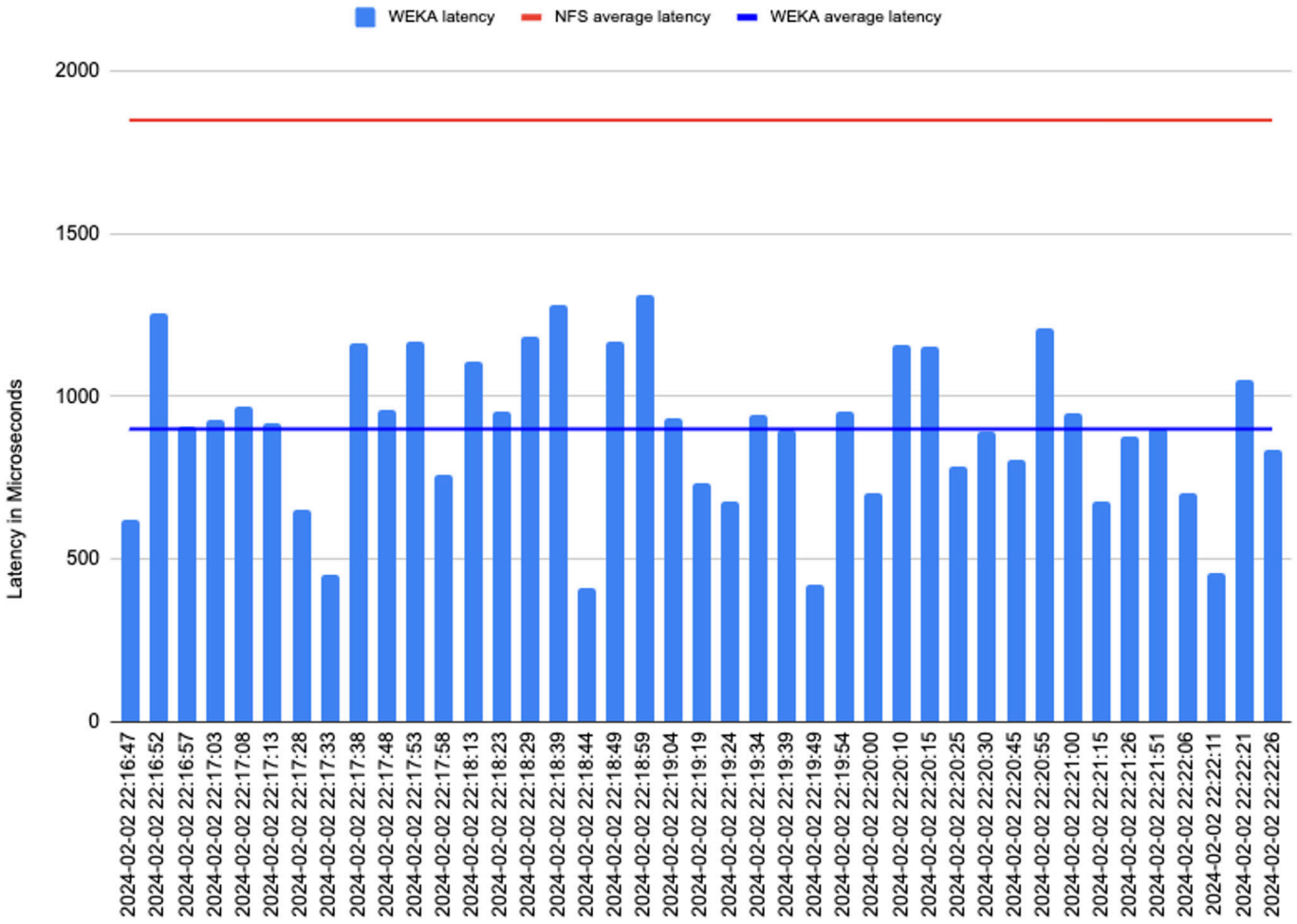

FIG. 3

For the medium-sized ASR speech-to-text model in Figure 4, similar results were seen, With WEKA again being in the ~900us average, while the NFS server was ~1825us. Note that as the model gets larger, and the number of parameters increase, the amount of time between checkpoints becomes less predictable due to new layers being added and different paths taken depending on weights and biases. This lack of predictability makes it harder to tune a filesystem to match the training workload, especially when multiple pipelines are running in parallel. In this case, some checkpoints are 30+ seconds apart, and others happen less than 5 seconds apart.

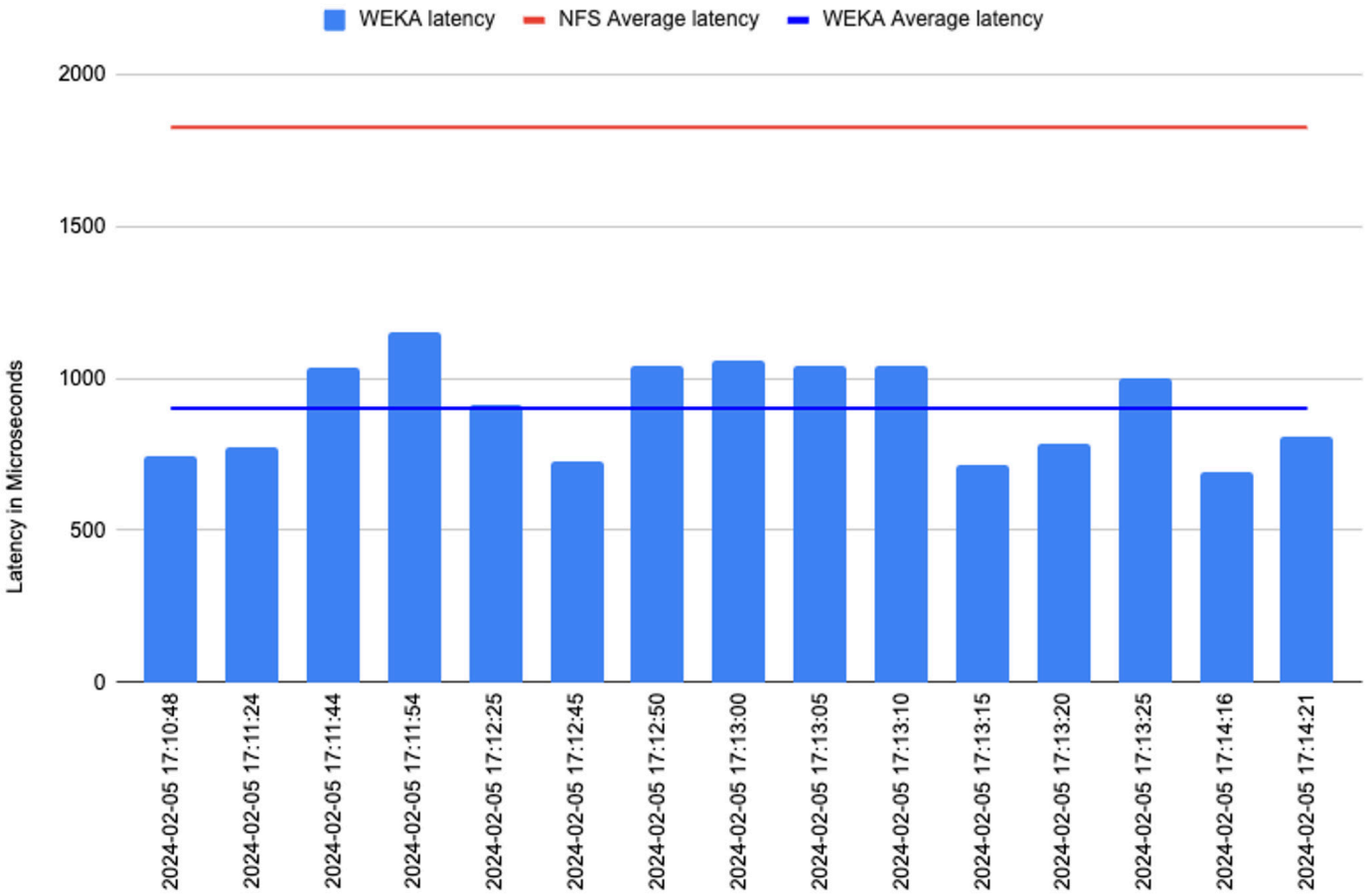

FIG. 4

And finally, when training the larger BioMegatron model over a much larger set of parameters, we see in Figure 5 that WEKA was able to service the checkpointing at an average of ~1090us and the NFS server averaged 1850us. While the resolution of the graph makes it impossible to show the timestamps, the unpredictability of the checkpointing is even more so than before, with some over 45 seconds apart, and others happening as little as 4 seconds apart. This highlights why latency is so important: Since checkpoint timing is non-predictable, the ability to handle it before the next checkpoint comes in becomes crucial, otherwise it can stall the entire model training until each checkpoint is committed.

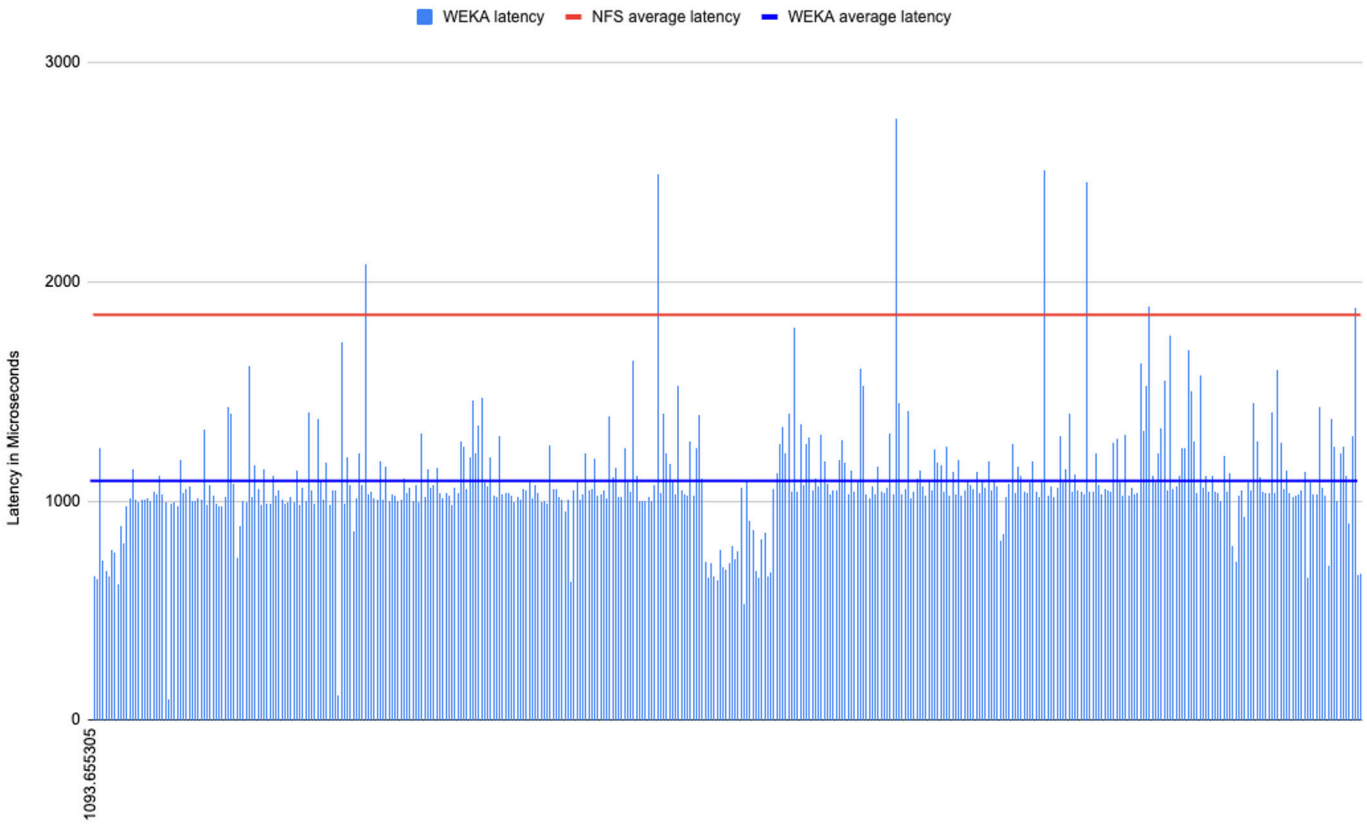

FIG. 5

Results

In this testing, we have shown how a cloud-based small WEKA infrastructure can be up to 50% faster at handling checkpointing than a high-performance NFS server. At the small scale presented here, the differences may not seem significant, but consider this: Distributed model training as models get larger and larger will also have distributed checkpointing. With the latest GPUs such as the GB200 from NVIDIA being hundreds of times faster than the single T4 used in this test, you can expect a much faster rate of training and a much faster rate of checkpointing as well. If the model is distributed over multiple GPUs, then the latency time will stall all GPUs until the checkpoint is committed.

Conclusions

Solutions for AI have until recently been based on infrastructure geared towards traditional enterprise IT. These solutions have often left customers with components in the stack that limit AI/ML development performance or and deviate from best practices in order to accommodate legacy architectures. Modern AI deployments require low-latency checkpointing to mitigate risk and enable operational predictability. If the storage layer in the stack is unable to meet an SLO (Service level Objective), it needs to be replaced to avoid costly delays to the AI data pipeline.

WEKA – A better solution for model checkpointing

As shown in the results above, the WEKA latency advantage translates to real world savings in checkpointing completion times. From a sustainability standpoint, the ability to rapidly recover and prevent GPUs from re-doing hours or days of work can't be ignored. In contrast to NFS based solutions for model training, WEKA has proven itself to have significantly lower latency, as well as being able to handle any IO profile of workload thrown at it. The table below highlights some of the differences between WEKA and most legacy enterprise NFS storage in an AI environment.

	Legacy Infrastructure (NFS)	WEKA Powered Infrastructure
Provides SLO for Checkpoints	May have to compromise on how often checkpoints can be taken to optimize for the infrastructure	No checkpoint restrictions. Checkpoint as often as needed
Multi-IO profile performance during Checkpointing	Performance highly variable. Higher latency can lead to bottlenecks across the infrastructure	Reads, writes, metadata all with consistent low latency. QOS to help with isolation of dedicated work flows
Tuning required per workload/workflow	Highly specialized per workload, may require multiple instances of storage services with multiple different mounting points for tuned parameters	Single mount with POSIX client provides all storage services, deployed with devops-level of standards and observability

WEKA features and industry integrations also helps optimize and streamline all parts of an AI-native infrastructure beyond just checkpointing:

- No filesystem or client tuning requirement to optimize for the workload. Control of training and job dynamics is left at the researcher level as hyperparameters.
- WEKA's ability to run anywhere including on-premises or in the cloud allows researchers to run jobs anywhere and take advantage of diverse compute resources.
- As performance requirements change, WEKA can elastically scale up and down to match the requirements needed in the AI workflow.
- NVIDIA BasePOD and SuperPOD certified allowing for well-known reference architectures to speed deployment.
- WEKA scaling into multi-exabytes of capacity, and the ability to address trillions of files ensure that as data sets continually get larger, researchers don't have to worry about engineering special data directory hierarchies to hold all the data.
- WEKA transparent tiering enables researchers to store large datasets and models for 'explainable AI' at scale to achieve practical cost economics and operational ease as the environment scales. Tiering assists with both infrastructure cost and the sustainable OPEX of power and cooling as well.
- WEKA Snap-to-object capabilities enable both data mobility for hybrid use cases, and file system recovery for data protection regardless of the amount of data stored.

The WEKA AI native data platform has proven these capabilities by enabling some of the largest GPU clusters in production in the world to do groundbreaking AI research in areas as diverse as computer vision, NLP, High frequency trading, Computational Chemistry and materials science, Generative AI for media and entertainment and more. WEKA continues to help customers get the most out of their AI processes and workflows.

Want to know more? Visit https://www.weka.io/data-platform/solutions/ai-machine-learning/

Appendix

How To build a NeMo toolkit:

Use the NeMo QuickStart guide at : https://docs.nvidia.com/deeplearning/nemo/user-guide/docs/en/main/starthere/intro.html to set up and deploy a basic NeMo environment.

Finetuning FastPitch with the NeMo toolkit:

https://github.com/NVIDIA/NeMo/blob/stable/tutorials/tts/FastPitch_Finetuning.ipynb

FastPitch training setup and parameters for the test run:

```
[NeMo W 2024-02-02 21:47:39 nemo_logging:349] /usr/local/lib/python3.10/dist-packages/hydra/_internal/
hydra.py:119: UserWarning: Future Hydra versions will no longer change working directory at job runtime
by default.
    See https://hydra.cc/docs/1.2/upgrades/1.1_to_1.2/changes_to_job_working_dir/ for more information.
      ret = run_job(

[NeMo W 2024-02-02 21:47:39 nemo_logging:349] /usr/local/lib/python3.10/dist-packages/lightning_fabric/
connector.py:554: UserWarning: 16 is supported for historical reasons but its usage is discouraged.
Please set your precision to 16-mixed instead!
      rank_zero_warn(

Using 16bit Automatic Mixed Precision (AMP)
GPU available: True (cuda), used: True
TPU available: False, using: 0 TPU cores
IPU available: False, using: 0 IPUs
HPU available: False, using: 0 HPUs
[NeMo I 2024-02-02 21:47:39 exp_manager:394] Experiments will be logged at
ljspeech_to_9017_no_mixing_5_mins/FastPitch/2024-02-02_21-47-39
[NeMo I 2024-02-02 21:47:39 exp_manager:835] TensorboardLogger has been set up
[NeMo W 2024-02-02 21:47:39 exp_manager:931] The checkpoint callback was told to monitor a validation
value and trainer's max_steps was set to 1000. Please ensure that max_steps will run for at least 25
epochs to ensure that checkpointing will not error out.
 NeMo-text-processing :: INFO     :: Creating ClassifyFst grammars.
Creating ClassifyFst grammars.
[NeMo W 2024-02-02 21:48:09 en_us_arpabet:66] apply_to_oov_word=None, This means that some of words
will remain unchanged if they are not handled by any of the rules in self.parse_one_word(). This may
be intended if phonemes and chars are both valid inputs, otherwise, you may see unexpected deletions in
your input.
[NeMo I 2024-02-02 21:48:09 dataset:229] Loading dataset from ./9017_manifest_train_dur_5_mins_local.
json.
76it [00:00, 2569.52it/s]
[NeMo I 2024-02-02 21:48:09 dataset:267] Loaded dataset with 76 files.
[NeMo I 2024-02-02 21:48:09 dataset:269] Dataset contains 0.08 hours.
[NeMo I 2024-02-02 21:48:09 dataset:377] Pruned 0 files. Final dataset contains 76 files
[NeMo I 2024-02-02 21:48:09 dataset:379] Pruned 0.00 hours. Final dataset contains 0.08 hours.
[NeMo I 2024-02-02 21:48:09 dataset:229] Loading dataset from ./9017_manifest_dev_ns_all_local.json.
2it [00:00, 2545.86it/s]
[NeMo I 2024-02-02 21:48:09 dataset:267] Loaded dataset with 2 files.
[NeMo I 2024-02-02 21:48:09 dataset:269] Dataset contains 0.00 hours.
[NeMo I 2024-02-02 21:48:09 dataset:377] Pruned 0 files. Final dataset contains 2 files
[NeMo I 2024-02-02 21:48:09 dataset:379] Pruned 0.00 hours. Final dataset contains 0.00 hours.
[NeMo I 2024-02-02 21:48:09 features:289] PADDING: 1
 NeMo-text-processing :: INFO     :: Creating ClassifyFst grammars.
Creating ClassifyFst grammars.
[NeMo W 2024-02-02 21:48:42 en_us_arpabet:66] apply_to_oov_word=None, This means that some of words
will remain unchanged if they are not handled by any of the rules in self.parse_one_word(). This may
be intended if phonemes and chars are both valid inputs, otherwise, you may see unexpected deletions in
your input.
```

```
[NeMo W 2024-02-02 21:48:42 modelPT:161] If you intend to do training or fine-tuning, please call the
ModelPT.setup_training_data() method and provide a valid configuration file to setup the train data
loader.
    Train config :
    dataset:
      _target_: nemo.collections.tts.torch.data.TTSDataset
      manifest_filepath: /ws/LJSpeech/nvidia_ljspeech_train_clean_ngc.json
      sample_rate: 22050
      sup_data_path: /raid/LJSpeech/supplementary
      sup_data_types:
      - align_prior_matrix
      - pitch
      n_fft: 1024
      win_length: 1024
      hop_length: 256
      window: hann
      n_mels: 80
      lowfreq: 0
      highfreq: 8000
      max_duration: null
      min_duration: 0.1
      ignore_file: null
      trim: false
      pitch_fmin: 65.40639132514966
      pitch_fmax: 2093.004522404789
      pitch_norm: true
      pitch_mean: 212.35873413085938
      pitch_std: 68.52806091308594
      use_beta_binomial_interpolator: true
    dataloader_params:
      drop_last: false
      shuffle: true
      batch_size: 24
      num_workers: 0

[NeMo W 2024-02-02 21:48:42 modelPT:168] If you intend to do validation, please call the ModelPT.setup_
validation_data() or ModelPT.setup_multiple_validation_data() method and provide a valid configuration
file to setup the validation data loader(s).
    Validation config :
    dataset:
      _target_: nemo.collections.tts.torch.data.TTSDataset
      manifest_filepath: /ws/LJSpeech/nvidia_ljspeech_val_clean_ngc.json
      sample_rate: 22050
      sup_data_path: /raid/LJSpeech/supplementary
      sup_data_types:
      - align_prior_matrix
      - pitch
      n_fft: 1024
      win_length: 1024
      hop_length: 256
      window: hann
      n_mels: 80
      lowfreq: 0
      highfreq: 8000
      max_duration: null
      min_duration: null
      ignore_file: null
      trim: false
      pitch_fmin: 65.40639132514966
      pitch_fmax: 2093.004522404789
      pitch_norm: true
      pitch_mean: 212.35873413085938
      pitch_std: 68.52806091308594
      use_beta_binomial_interpolator: true
```

```
    dataloader_params:
      drop_last: false
      shuffle: false
      batch_size: 24
      num_workers: 0

[NeMo I 2024-02-02 21:48:42 features:289] PADDING: 1
[NeMo I 2024-02-02 21:48:42 save_restore_connector:249] Model FastPitchModel was successfully restored
from /NeMo/weka-workspace/nemo/tutorials/tts/tts_en_fastpitch_align.nemo.
[NeMo I 2024-02-02 21:48:42 modelPT:1234] Model checkpoint restored from nemo file with path : `./
tts_en_fastpitch_align.nemo`
LOCAL_RANK: 0 - CUDA_VISIBLE_DEVICES: [0]
[NeMo I 2024-02-02 21:48:43 modelPT:728] Optimizer config = Adam (
    Parameter Group 0
        amsgrad: False
        betas: [0.9, 0.999]
        capturable: False
        differentiable: False
        eps: 1e-08
        foreach: None
        fused: None
        lr: 0.0002
        maximize: False
        weight_decay: 1e-06
    )
[NeMo I 2024-02-02 21:48:43 lr_scheduler:772] Scheduler not initialized as no `sched` config supplied to
setup_optimizer()

  | Name                | Type                                | Params
------------------------------------------------------------------------------
0 | mel_loss_fn         | MelLoss                             | 0
1 | pitch_loss_fn       | PitchLoss                           | 0
2 | duration_loss_fn    | DurationLoss                        | 0
3 | energy_loss_fn      | EnergyLoss                          | 0
4 | aligner             | AlignmentEncoder                    | 1.0 M
5 | forward_sum_loss_fn | ForwardSumLoss                      | 0
6 | bin_loss_fn         | BinLoss                             | 0
7 | preprocessor        | AudioToMelSpectrogramPreprocessor | 0
8 | fastpitch           | FastPitchModule                     | 45.8 M
------------------------------------------------------------------------------
45.8 M    Trainable params
0         Non-trainable params
45.8 M    Total params
183.035   Total estimated model params size (MB)
[NeMo W 2024-02-02 21:48:48 nemo_logging:349] /usr/local/lib/python3.10/dist-packages/pytorch_lightning/
loops/fit_loop.py:281: PossibleUserWarning: The number of training batches (7) is smaller than the
logging interval Trainer(log_every_n_steps=100). Set a lower value for log_every_n_steps if you want to
see logs for the training epoch.
      rank_zero_warn(

Training: 0it [00:00, ?it/s][NeMo I 2024-02-02 21:48:48 preemption:56] Preemption requires torch
distributed to be initialized, disabling preemption
Epoch 24: 100%|██| 7/7 [00:02<00:00,  3.21it/s, v_num=7-39, train_step_timing in
Validation: 0it [00:00, ?it/s]
Validation:   0%|                                          | 0/1 [00:00<?, ?it/s]
Validation DataLoader 0:   0%|                            | 0/1 [00:00<?, ?it/s]
Epoch 24: 100%|██| 7/7 [00:02<00:00,  2.71it/s, v_num=7-39, train_step_timing in
                                                                          Epoch 24, global
step 175: 'val_loss' reached 1.89732 (best 1.89732), saving model to '/NeMo/weka-workspace/nemo/
tutorials/tts/ljspeech_to_9017_no_mixing_5_mins/FastPitch/2024-02-02_21-47-39/checkpoints/FastPitch--
val_loss=1.8973-epoch=24.ckpt' as top 3
```

```
Epoch 49: 100%|█| 7/7 [00:02<00:00,  3.18it/s, v_num=7-39, train_step_timing in
Validation: 0it [00:00, ?it/s]
Validation:   0%|                                          | 0/1 [00:00<?, ?it/s]
Validation DataLoader 0:   0%|                            | 0/1 [00:00<?, ?it/s]
Epoch 49: 100%|█| 7/7 [00:02<00:00,  2.65it/s, v_num=7-39, train_step_timing in
                                                                        Epoch 49, global
step 350: 'val_loss' reached 2.04128 (best 1.89732), saving model to '/NeMo/weka-workspace/nemo/
tutorials/tts/ljspeech_to_9017_no_mixing_5_mins/FastPitch/2024-02-02_21-47-39/checkpoints/FastPitch--
val_loss=2.0413-epoch=49.ckpt' as top 3
Epoch 74: 100%|█| 7/7 [00:02<00:00,  3.28it/s, v_num=7-39, train_step_timing in
Validation: 0it [00:00, ?it/s]
Validation:   0%|                                          | 0/1 [00:00<?, ?it/s]
Validation DataLoader 0:   0%|                            | 0/1 [00:00<?, ?it/s]
Epoch 74: 100%|█| 7/7 [00:02<00:00,  2.71it/s, v_num=7-39, train_step_timing in
                                                                        Epoch 74, global
step 525: 'val_loss' reached 2.00175 (best 1.89732), saving model to '/NeMo/weka-workspace/nemo/
tutorials/tts/ljspeech_to_9017_no_mixing_5_mins/FastPitch/2024-02-02_21-47-39/checkpoints/FastPitch--
val_loss=2.0018-epoch=74.ckpt' as top 3
Epoch 99: 100%|█| 7/7 [00:02<00:00,  3.16it/s, v_num=7-39, train_step_timing in
Validation: 0it [00:00, ?it/s]
Validation:   0%|                                          | 0/1 [00:00<?, ?it/s]
Validation DataLoader 0:   0%|                            | 0/1 [00:00<?, ?it/s]
Epoch 99: 100%|█| 7/7 [00:02<00:00,  2.63it/s, v_num=7-39, train_step_timing in
                                                                        Epoch 99, global
step 700: 'val_loss' reached 1.92470 (best 1.89732), saving model to '/NeMo/weka-workspace/nemo/
tutorials/tts/ljspeech_to_9017_no_mixing_5_mins/FastPitch/2024-02-02_21-47-39/checkpoints/FastPitch--
val_loss=1.9247-epoch=99.ckpt' as top 3
Epoch 124: 100%|█| 7/7 [00:02<00:00,  3.35it/s, v_num=7-39, train_step_timing in
Validation: 0it [00:00, ?it/s]
Validation:   0%|                                          | 0/1 [00:00<?, ?it/s]
Validation DataLoader 0:   0%|                            | 0/1 [00:00<?, ?it/s]
Epoch 124: 100%|█| 7/7 [00:02<00:00,  2.75it/s, v_num=7-39, train_step_timing in
                                                                        Epoch 124, global
step 875: 'val_loss' reached 1.86749 (best 1.86749), saving model to '/NeMo/weka-workspace/nemo/
tutorials/tts/ljspeech_to_9017_no_mixing_5_mins/FastPitch/2024-02-02_21-47-39/checkpoints/FastPitch--
val_loss=1.8675-epoch=124.ckpt' as top 3
```