TranLogs: Lossless Failure Recovery Empowered by Training Logs

Xiaoyu Liu
Zhejiang Lab
HangZhou, China
liuxy@zhejianglab.org

Lingfang Zeng
Zhejiang Lab
HangZhou, China
zenglf@zhejianglab.org

2024 International Conference on Networking, Architecture and Storage (NAS) | 979-8-3315-2050-2/24/$31.00 ©2024 IEEE | DOI: 10.1109/NAS63802.2024.10781359

***Abstract*—When running deep learning training jobs, in order to prevent training loss due to software/hardware failures, a checkpointing mechanism is usually used to periodically store snapshots of the training model in non-volatile memory to facilitate recovery from failures. However, the recovery effect of this method is closely related to the checkpointing period and the failure time, and in the worst case, even nearly one period of training accuracy will be lost. With increasing model sizes, the need to develop more fine-grained model fault recovery methods to reduce model fault loss is more urgent for deep learning training tasks that use checkpointing mechanisms as fault recovery guarantees. We present TranLogs, a checkpoint failure recovery method combined with training logs for lossless failure recovery when training deep learning models. TranLogs proposes to use logging to supplement the checkpoint file and track the iterative process of model training parameters. For the situation where the checkpoint file is not updated in time and encounters model failures, resulting in a large loss during recovery, TranLogs can handle it effectively. The best-case scenario achieves about 24% and 6% improvement in the loss and accuracy of the recovered training state, respectively, which is closer to the state already achieved by the model training at the moment of failure.**

***Index Terms*—Deep learning, Failure recovery, Checkpoint, Logging mechanism, Self-sampling.**

I. Introduction

As the ability of deep learning to solve complex problems is continuously tapped, methods based on deep learning ideas have been widely used in high energy physics[1][2][3], astronomical sciences[4][5], environmental sciences[6][7], computer vision[8], natural language processing[9], smart cities[10], automated driving[11][12],robot[13], industrialized production[14], and other scientific data processing and industrialized production environments. Existing models are difficult to meet the different processing requirements, so the scale and complexity of deep learning gradually increase with the demand of applications.

The emergence and development of large language models such as ChatGPT has pushed the deep learning training scale to a new order of magnitude, where a single machine is no longer able to accomplish the training task and distributed training must be used to do so. For example, Training GPT-4 uses up to tens of thousands of NVIDIA A100 GPUs, costing more than $100 million[15]. The addition of distributed training further increases the complexity of training, the likelihood of failures during the training process also grows, and the cost of losses caused by failures also grows. A training failure may lead to loss of hours or even days of work, seriously affecting the normal operation of training and costing huge economic losses. According to the OPT-175B training report, about 178,000 GPU hours were lost during training due to training failures caused by various faults. Since the frequency of failures increases with the training size, failures significantly slow down the training progress (up to 43%)[16]. Therefore, it is very important to build an efficient fault-tolerant system for deep learning applications.

Typically, researchers use checkpoints to periodically record the training state in order to minimize losses from failures and error tolerance. The checkpoint mechanism periodically records information about the entire training model, including the model architecture (layers of the model, connections between layers, and information about each layer), weights (current values of each trainable parameter in the model), optimizers (status and parameters of the optimizers), and some user-defined configurations. In this way, in case of a training failure, the model training job can be restarted from the nearest checkpoint to minimize the damage caused by the failure.

Checkpoint-based fault tolerance mechanism can reduce the loss caused by faults to a certain extent, and restore the model to the state of the most recent checkpoint with lower cost overhead. However, checkpoints currently faces the following three challenges: the first is to reduce the training loss from the most recent checkpoint to the time of failure to achieve lossless recovery of the training task; the second is to reduce the system I/O overhead brought by the checkpoint mechanism; and the third is to reduce the storage pressure brought by the storage demand for a large number of checkpoint files.

This paper analyzes and summarizes the current research status of checkpoints optimization methods for deep learning training tasks. In response to existing research on the need to repeat some of the training after a fault has occurred, we investigate the use of training log technology to solve the dynamic checkpoint optimization problem of machine learning training tasks and propose TranLogs, a log-based checkpoint mechanism. The time-consuming percentage of each process of acquiring data, forward propagation, loss calculation, back propagation and parameter update in a training step is analyzed in this paper. The idea of recording training logs is adopted to record the intermediate results of key steps, which avoids los-

ing all the training processes between the nearest checkpoints directly after a fault occurs, and is able to restore the training state to the state at the moment of the fault in a shorter time and with less overhead. The contributions of this paper include the following:

- Time-consuming analysis of the training process based on a self-sampling method.
- Log-based checkpoint mechanism TranLogs is designed.
- Implementing TranLogs in a typical training framework for deep learning.
- Through test experiments, it is verified that TranLogs can achieve about 24% and 6% improvement in the loss and accuracy of the recovered training state, respectively, which is closer to the state already achieved by the model training at the moment of failure.

This paper is organized as follows. We discuss the background of the paper and the research related to checkpointing mechanism in Section II. We introduce the motivation of the idea of this paper in Section III. We design and implement TranLogs in Section IV. We analyze the performance of TranLogs through experiments in Section V. We conclude the paper in Section VI.

II. Backgroung and related work

A. Checkpointing for DL Training

Due to the complex structure of deep learning models and the large amount of samples, the training tasks often occur breakdown. The main idea of deep learning fault-tolerant system is to back up the model information, which is implemented as the checkpoint mechanism in each mainstream training framework. Researchers can adopt different checkpoint strategies based on the type of training performed, which can be categorized into short-term training strategy, regular training strategy, and long-term training strategy. The main difference between different strategies lies in the tradeoff between the frequency of writing checkpoints and the number of checkpoints. The difference between each checkpoint strategy, its advantages and disadvantages are shown in Table I.

TABLE I: Comparison of different checkpoint strategies

Strategy	Frequency	Number of checkpoints	Advantage	Disadvantage
short-term	frequently	massive	recover to training status quickly	high storage and I/O overhead
regular	moderate	moderate	not noticeable by comparison	not noticeable by comparison
long-term	rarely	scant	slow recovery time, high losses	low storage and I/O overhead

The different strategies mentioned above are all belong to periodic checkpoints, although there are differences between the checkpointing frequency and the number of checkpointing files. Periodic checkpointing requires the checkpointing frequency and number to be determined before the training starts, which has the following major issues: (1) fixing the checkpointing frequency during training, which cannot be adjusted according to the system state during training; (2) significant steady-state overhead (I/O, storage) for periodic checkpointing; (3) the need to redo the work from the last checkpoint during fault recovery. The above problems become more and more obvious with the increasing scale of the model.

B. Related Work

To address the first issue above, Mohan et al.[17] proposed an online analysis algorithm, which can dynamically adjust the checkpointing frequency during the training process, optimize in time according to the state of the system, and introduce two-phase checkpointing to effectively reduce the cost of checkpointing as well as the recovery time. Zhang et al.[18] proposed a method for dynamic adjustment of checkpoint intervals through reinforcement learning, called DACM, which adaptively optimizes processing delay and fault recovery time. Zhuang et al.[19] investigated the checkpoint interval for fault recovery in distributed stream processing systems and proposed a checkpoint interval optimization model and a dynamic checkpoint interval adjustment algorithm based on the processing rate, checkpoint overhead and mean time between failures. Akber et al.[20] proposed to adjust checkpoint intervals by failure probability, and experimental results show that the proposed checkpointing strategy significantly reduces checkpointing overhead compared to periodic checkpointing. In response to the second issue above, that is, the current checkpointing mechanism has more space for optimization in terms of I/O and storage, for these two sub-problems, Wang et al.[21] proposed to utilize the local memory of the training machine to store the most recent checkpoint, which can effectively reduce the recording time of the checkpoints and the recovery time in case of training failures. Eisenman et al.[22] proposed to utilize quantization techniques and incremental iterations to reduce the checkpoint size in order to alleviate the storage bandwidth and storage space pressure. Microsoft investigated Nebula[23], which can provide an efficient distributed large-scale model training operation using PyTorch with a checkpointing solution that can reduce checkpointing time from hours to seconds by utilizing state-of-the-art distributed computing techniques. Zhang et al.[24] proposed to achieve efficient fault tolerance by combining erasure coding with unique features of deep-learning-based recommendation model training, which reduces the training time overhead by as much as 66% on large deep-learning-based recommendation models. Wang et al.[25] propose FastPersist to accelerate checkpoint creation in DL training. As well as a number of other studies related to reducing checkpointing overheads[26][27][28][29]. To address the third issue above, Gupta et al.[30] proposed just-in-time checkpointing JIT. By leveraging the fact that the GPU state is only updated during a short interval that we can track, along with the availability of multiple replicas holding the same state, JIT is employed only after a failure occurs, instead of frequent periodic checkpointing. The JIT check-

pointing solution is able to recover from common failures in few seconds by redoing at most a minibatch of work.

The above studies optimize the checkpoint mechanism for deep learning from different perspectives, and all of them achieve some performance improvement. However, relatively little research has been done on optimizing the training records between the time of failure and the most recent checkpoint, and the training during this period is wasted in existing checkpoint implementations, which does not allow for lossless recovery of the trained model. Logging technique is a method to record system operation, actions and events, such as the edit log file of HDFS (Hadoop Distributed File System) is used to record the change operations of the file system, which can be used to restore the state of the file system quickly and efficiently by replaying the edit log after system failure or restarting without the need to scan or rebuild the whole system in full volume. In this paper, we propose TranLogs, an approach that employs training logs to solve the problem of lossless recovery of deep learning training models. For the part of training from the last checkpoint to the time when the model fails, it adopt the training log to perform fast and lossless recovery of the training, which is to record the update of the weight vector of the current time in the log file after the end of each iteration. After the failure of the training, by combining the log file with the most recent checkpoint, the training can be quickly restored to the state at the moment of failure, realizing the lossless recovery of the model.

III. Motivation

A. Lossless Recovery Problem for Training Models

As the scale and complexity of the deep learning training model increases, the probability of failure during the training process also increases greatly. The periodic checkpoint mechanism can carry out the preservation of the model training progress to a certain extent, and can restore the training to the state of the most recent checkpoint record after a failure. However it cannot guarantee the restoration of the training record from the most recent checkpoint to the time of the failure, and this part of the training will be lost, as shown in Figure 1. At worst, training records close to one checkpoint record cycle will be lost. On average half a checkpoint recording cycle of training records is lost, which means that all training GPUs need to repeat the work for half the time interval of the checkpoint. Existing optimizations for traditional checkpoint do not have a relevant solution strategy for the lossless recovery problem, and the lossless recovery problem urgently needs an effective solution.

B. High Cost of Short-term Checkpoints

To address the above issue of training models for lossless recovery, the average loss of not being able to fully recover to the pre-failure state can be reduced by decreasing the period of writing checkpoints (i.e., increasing the frequency of checkpoints). Meanwhile, this approach exponentially raises the cost of model training, including the cost of additional I/O and storage space. For example, the Facebook Research team's

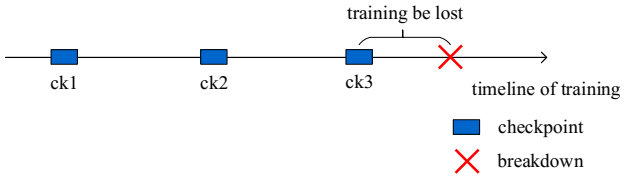


Fig. 1: Periodic checkpoint loss history training in the event of a breakdown

ResNeXt-101 (64x4d) model has a size of 638MB[31][32], and saving the checkpoint file in epochs already requires about 60GB of storage space, and even more if the epochs are further reduced. Moreover, if we want to improve the I/O rate of checkpoints in this case again, i.e., to store the checkpoint file through the local cache of the training machine, it is even more difficult to satisfy the storage space requirement of the checkpoints in the bottom cycle.

C. Time-consuming Analysis of Training Process based on Bootstrap

The model training includes multiple rounds of epochs, each epoch is divided into several iterations based on the pre-specified batch size. Then the *train step* function will be executed once in each iteration, which includes processes such as prepare data, compose gradients, update weights, compose metrics, etc. This section analyzes the time consuming situation when the above processes are executed for a specific deep model, and in order to obtain more accurate results, the model is run several times and the execution time of each process is calculated by Bootstrap method.

The Bootstrap method[33] is a statistical inference method for estimating/correcting the bias/variance information of statistical estimates. For a specific process (e.g., prepare data) when the model is executed, if the elapsed time of the execution in each of its iteration is recorded, the average elapsed time of the process can be calculated, but this method has too much overhead. Borrowing the idea of Bootstrap method, in each round of execution, record n of its execution time as x_1, x_2, ..., x_n, and then after executing B rounds, calculate the average value of the samples taken in each round $\hat{M}_j(j = 1...B)$ respectively.

$$\hat{M}_j = \frac{1}{n}\sum_{i=1}^{n} x_i, \quad j \in [1, B] \tag{1}$$

The mean and variance of these statistics are computed based on the calculated $\hat{M}_j(j = 1...B)$ as a way to estimate the overall mean of a process (e.g. prepare data).

$$\overline{m}^* = \frac{1}{B}\sum_{j=1}^{B} \hat{M}_j \tag{2}$$

$$s^2 = \frac{1}{B}\sum_{j=1}^{B} (\hat{M}_j)^2 - (\frac{1}{B}\sum_{j=1}^{B} \hat{M}_j)^2 \tag{3}$$

Based on the above theoretical, the Multilayer Perceptron(MLP) model was tested. The mean and variance of its processes were counted as shown in Table II. According to the data, the percentage of each process of the model is shown in Figure 2.

TABLE II: Mean and variance of time spent on each process of a MLP model

Train process	$\overline{m}^*$	s_2
prepare data	0.062	0.000001
compute gradients	38.652	3.117
update weights	13.532	1.536
compute metrics	3.866	0.002

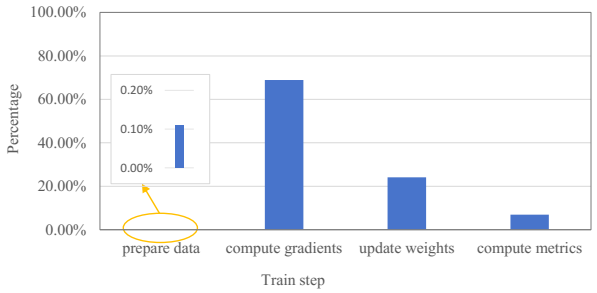


Fig. 2: Percentage of time spent on each training process for a MLP

With the above example, it can be seen that the step of calculating the gradient takes the longest percentage of time during training. If this step is avoided to be reproduced in the failure recovery phase, the failure recovery cost can be effectively reduced.

D. Tradeoffs between Time and Space

According to the conclusion of the time-consuming analysis part of the training process, if the results of gradient calculation are saved and the model recovery stage directly obtained the results. The amount of computation in the model recovery process can be reduced, and the cost of fault recovery can be reduced. This approach can save storage space to a certain extent compared with directly recording the model weights (i.e., storing the model checkpoints according to a fine-grained scale), although the training time saved is relatively short. Table III shows the comparison of a MLP models under different strategies in terms of storage space and training time savings, the actual results of this experiment are closely related to the specific model and scale, and Figure 3 gives the trend change of model scale on computation time and storage space(MLP2 has 3.19 times more neurons than MLP1).

TABLE III: Comparison of stored gradients and stored weights

Strategy	Storage space	Save training time (accounts for one update process)
store gradients	1.82GB	68.99%
store weights	1.85GB	93.11%

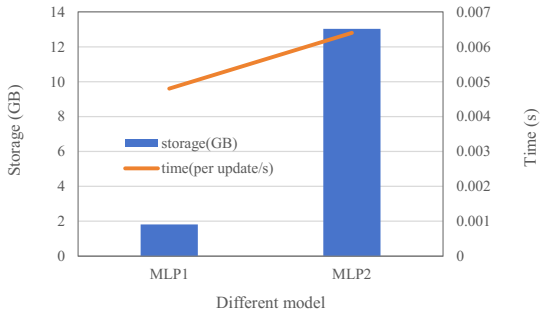


Fig. 3: Impact of model size on computation time & storage cost

As per the above experimental results, storing gradients compared to storing weights takes up much more storage space than training time. Thus weighing the storage space to be invested against the training time saved, it is relatively more cost-effective to store the intermediate results of the gradient computation step.

IV. Design and Implementation of TranLogs

A. Core Ideology

Deep learning jobs are usually terminated by the user specifying a fixed number of epochs or the training accuracy they want to achieve. However, multiple rounds of iterations processing are performed in the above methods, and in each epoch, all samples in dataset are used once if there are no special circumstances (e.g., an importance sampling method is used for training). Within each epoch, all the samples are divided according to the specified batch size and processed in multiple iterations. Then each iteration uses batch size data, and performs steps such as data preparation and gradient computation, etc., respectively. Completing one round of epoch after executing a number of iterations, and performs a checkpoint. The method in this paper records the gradient update log in each iteration processing, so that by logging each parameter update of training through fine-grained logging operations during the training process. The training results can be read after the occurrence of training faults for certain steps in the training that are not recorded in the checkpoint file, reducing the cost of repeating the training again.

Combined with the *train step* function called for actual training in the keras, the training flow is drawn as shown in Figure 4, where the step of calculating the gradient is time-consuming. After recovering from the checkpoint file to the training state at the most recent recorded checkpoint moment after a training fault occurs, it is necessary to repeat all the steps in Figure 4 until the moment of the fault to recover to the training state at the time of the fault. Through the analysis in III-C, calculating the gradient is a comparatively time-consuming step, and TranLogs saves the calculation time of this step and its predecessor steps in the iterative process by recording the updates in this step to the log, so that the results

of this step can be read directly from the file when recovering without repeating the calculations, and reduces the time cost of fault recovery. In terms of recovery accuracy, the method improves the accuracy of the training process recording, and can record the training results to one iteration before the fault occurs through the log file and the checkpoint file, realizing the lossless recovery of the training model.

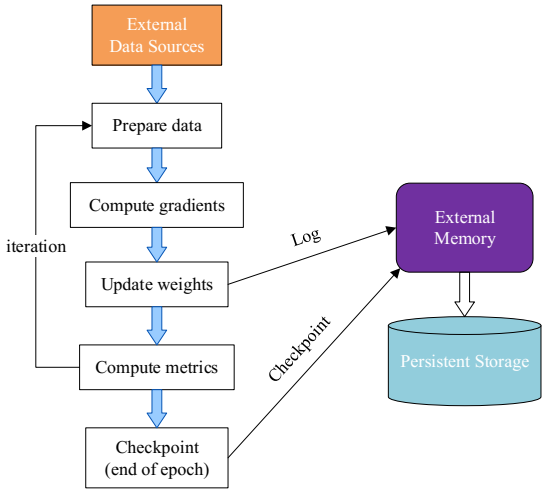


Fig. 4: Training process

B. Detail Design

This section describes the detailed design and specific implementation strategy of TranLogs, including the log efficient iteration and asynchronous persistence strategy. Log efficient iteration refers to the timely cleanup of log files that have lost their value to guarantee the neatness and efficiency of the storage environment. Asynchronous persistence strategy refers to the log file writing strategy involved in the mechanism described in the paper.

In order to provide theoretical support for the log efficient iteration strategy in numerical form, we use the value estimation of log files, which is a variable that represents the change of the value of log files over time. It is implemented based on the following assumptions:

- The value of a log file is negatively correlated with its time of existence, the longer the file has existed, the lower its value.
- The value of a log file correlates with whether or not checkpointing the file has been performed for the training epoch it is in. If checkpointing has been performed for the epoch it is in, the value of the file is reduced.
- The value of a log file is related to the availability of its associated log file. If the log file it is associated with is not available, the value of the file is relatively high.

The following clarifications are added to the above assumptions regarding the log file in relation to the checkpoint file. First, the log file exists between two checkpoint files, and its file content complements the recorded content of the checkpoint file, so the value of the log file is closely related to whether it is written to the checkpoint file before or after it. Second, since there is a certain probability of unavailability of checkpoint files, and whether checkpoint files (especially checkpoint files adjacent to the log file) are available or not is also closely related to the value of the log file. Based on the above assumptions, we arrive at the following estimate of the value of the log file.

$$V_i = (\frac{-t}{t_e}) \times u_{i-1} - \alpha_i \tag{4}$$

In the above equation, t is the time, t_e is the period length of recording checkpoints, u_i indicates whether checkpoint file i is available or not (the value can be regarded as a constant), and α_i is the existence or not of the checkpoint file. Through the above analysis, we can conclude that in the model lossless recovery mechanism based on the logging scheme, the value of the log file changes with the migration of time. Specifically, the change takes the time of recording the checkpoint file as the node, and has a gradient decreasing trend, as shown in Figure 5. From the fig it can be seen that, at the same point of time, the value of the file that has been generated for a longer period of time is lower.

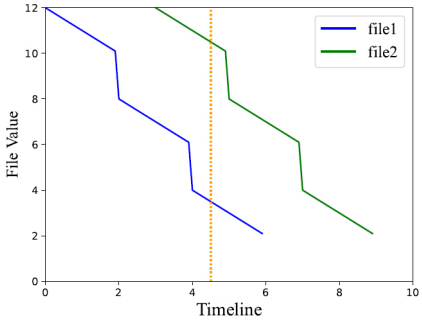


Fig. 5: Log file values over time

The log efficient iteration technique is designed and implemented based on the theory of value change of log files. From the point of view of saving storage space, log files with lower value can be deleted, but from the point of view of security, storing log files from the previous training period can serve as a backup. To balance the two, log management is performed by setting a storage space threshold. This part is specifically implemented by the log efficient iteration module introduced in Section IV-C, where the specific implementation details of this module are carried out.

The asynchronous persistence strategy comes into play when writing log files, which are first written to the memory cache layer and then asynchronously written to disk for persistence, as shown in Figure 6. This strategy ensures that log files are written at memory speed on the one hand, avoids problems such as memory failure from affecting the log files on the other hand.

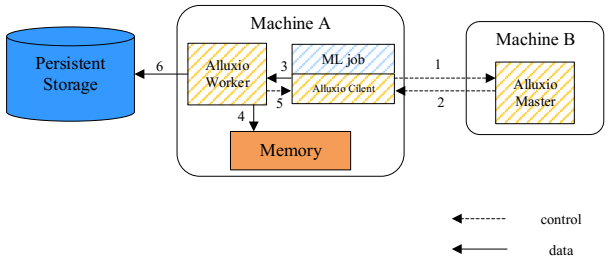


Fig. 6: Asynchronous persistence strategy

C. Implementation of TranLogs based on Tensorflow Platform

The implementation of this paper is based on the typical deep learning training frameworks Tensorflow and Keras. Specifically, it can be divided into the following three aspects.

1) Keras API Modifications: This part modifies the training function provided in the original framework, and adds the operation code needed to realize TranLogs on its basis. Take training MLP as an example, build the network structure based on the Sequential model in Keras, which is realized by class Sequential(Model) in Keras, which inherits the Model class in keras/engine/training.py. The *fit* method in the Model class is a method with batch backpropagation training capability, model training is realized by calling the *fit* method, and then the specific iteration steps are realized by the *train step* method. We have created new *fit bylog* and *train step bylog* methods in the Model class, in which logging and training recovery operations based on the log information are realized. Logging records new parameter updates generated by each iteration to a log file. The format of the log file supports hdf5, the default file format for linux. During fault recovery, the model state is first restored based on the checkpoint file. Then iterating the model parameters to the moment of failure based on messages recorded in the log file.

2) Log Efficient Iteration Module: This module is a background monitoring program that monitors the storage space occupied by log files and performs log file processing. The guiding principle is to delete logs of lower value rounds when the storage space occupied by log files is about to reach a set threshold. According to the theory of value change of log files introduced in Section IV-B, the logs of the less valuable rounds deleted here are the logs of the furthest rounds according to the current training rounds of the training task and have not been deleted before. The implementation logic is shown in Algorithm 1.

The algorithm's key parameters are the log file storage directory to be monitored, the total storage space, the threshold value, the number of rounds to be set, the number of steps in each round of the training rounds, and the pause time. The algorithm's specific steps are as follows, getting the current log file occupies the storage space size size, if size is greater than the upper limit of storage, then delete according to the current time of the longest training rounds of a log file, at the same time identify the training rounds of the parameter epoch plus 1. Then continue to carry out the above monitoring and processing until epoch reaches the total number of rounds of training or the end of the training to stop the procedure.

Algorithm 1 Efficient Iteration

Require: $dirPath, spaceTotal, threshold, epoch, step, t$
  $actualSpaceTotal \leftarrow spaceTotal \times threshold$
  $e \leftarrow 0$
  while True **do**
    $size \leftarrow getDirSize(dirPath)$
    if $size >= actualSpaceTotal$ **then**
      for $s = 0$ to $step$ **do**
        $deleteFile(e, s)$
      end for
      $e \leftarrow e + 1$
    end if
    $time.sleep(t)$
    if $e >= epoch$ **then**
      $break$
    end if
  end while

3) Persistence Policy Settings: This section mainly aim at the I/O overhead introduced by the logging mechanism. Using an asynchronous through strategy to reduce the overhead. It is implemented with the support of Alluxio distributed cache system, Alluxio provides ASYNC THROUGH write type, using this way the data will be written synchronously to the Alluxio worker first, and then persisted in the background to write to the underlying storage system. Since Alluxio 2.0, ASYNC THROUGH has become the default write type[34].

V. Experiment

This section first describes the hardware and software configuration of the experimental environment, the setup of the experiment, and the design of the test cases. Then comparing the log-based fault recovery strategy with the checkpointing mechanism that does not use the logging approach in terms of fault recovery accuracy, and verifies the effectiveness of TranLogs in optimizing the performance and usability of the training of the deep learning model in terms of fault recovery. Finally, it analyzes the test results.

A. Experimental Setup

In order to verify the effectiveness of TranLogs, this paper conducts related experiments on the deep learning training framework Tensorflow, using Alluxio as the caching system, based on the internal server nodes in the lab, with the specific hardware and software configurations shown in Table IV.

The experiments are divided into three aspects, the first is a comparison experiment, which compares TranLogs with the checkpoint method that does not use logs as an aid, analyzes and evaluates from the fault recovery accuracy. The second is a test of the log-efficient iterative strategy in TranLogs in terms of the occupation of storage space over time. The third is to test the asynchronous persistence strategy.

TABLE IV: Test environment

Component	Configuration
CPU	Intel(R) Xeon(R) Platinum 8358 CPU @ 2.60GHz
GPU	NVIDIA A40
OS	Ubuntu 22.04
Python	3.7.16
Tensorflow	2.4.0
Keras	2.4.3
Alluxio	2.9.3
Alluxio python client	0.1.4

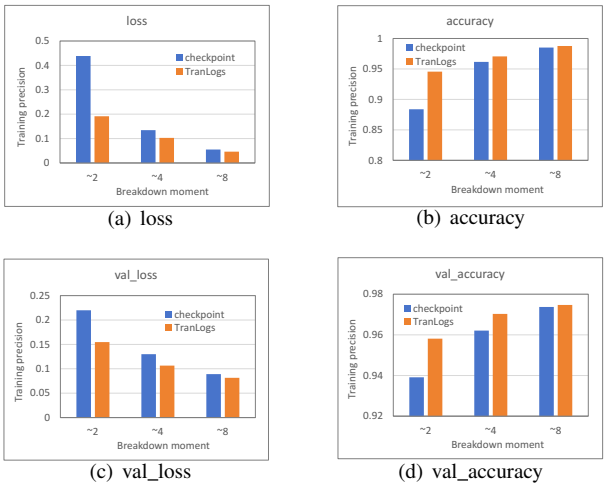
The training models used in the experiments are MLPs with different network hierarchical structures, the dataset used in the experiments is the MNIST, the checkpoint recording period is epoch, and the logging is after each parameter update. For the comparison experiments, we design three sets of test cases for comparing the recovery accuracy of TranLogs and the mainstreaming method for training without repeated computation at different moments of fault occurrence. For the logging efficient iteration strategy, we conduct experiments using different thresholds to record the changes in storage space usage. For the asynchronous persistence strategy, we conduct throughput tests under different strategies for log files of two models with different hierarchical structures. To avoid the chance of experimental results, each test case is conducted five times respectively and the average of the experimental results is taken.

B. Analysis of Experimental Result

1) Comparison of Different Checkpoint Mechanisms: Comparing TranLogs with the checkpoint method that does not use logs as an aid, simulating different moments of fault occurrence, comparing the recovery of the training state (represented by the four metrics of loss[1], accuracy[2], val_loss[3], and val_accuracy[4]) between the two methods without repeated iterative computation, the experimental results are shown in Figure 7. Without repeating the iterative computation, experimental results show that TranLogs can restore the model training to a state closer to the one already reached at the moment of failure, with an optimal improvement of about 24% and 6% for loss and accuracy, respectively.

2) Efficient Iteration Strategy for Logs: The log efficient iterative strategy in TranLogs is tested for the change in the storage space occupied by the log files under different thresholds, and the experimental results are shown in Figure 8. The results show that under different threshold conditions, as the training advances, the accumulated log files continue to increase, and the occupied space decreases when approaching the set threshold, which is exactly where our monitoring program plays a role in controlling the space occupied by the log files within the specified range.

[1]the loss value of training set
[2]the accuracy value of training set
[3]the loss value of test set
[4]the accuracy value of test set



(a) loss (b) accuracy (c) val_loss (d) val_accuracy

Fig. 7: Comparing of TranLogs and periodic checkpoint

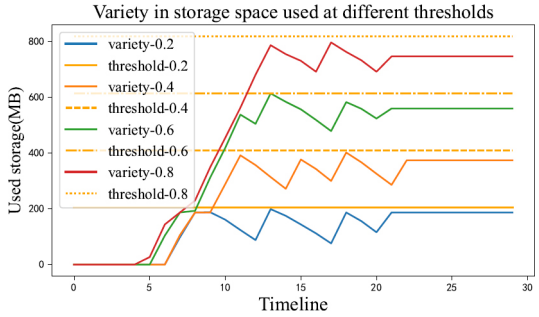


Fig. 8: Changes in storage space occupied by log files over time for different threshold settings

3) Asynchronous Persistence Strategy: This section Compares the write efficiency of log files with and without the asynchronous persistence strategy. The goal is to demonstrate that an asynchronous persistence strategy can reduce the I/O overhead introduced by the logging mechanism. The test is carried out for log files of different structural models. Experimental results are shown in Figure 9. The results show that for log files of different structural models, adopting this strategy achieves higher write throughput than not adopting this strategy.

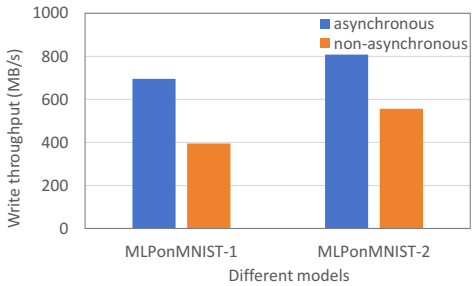


Fig. 9: Log file throughput under different policies

VI. Conclusion

In this paper, we propose TranLogs to address the difficulty of checkpoint mechanisms non-destructively recover the model training state. TranLogs explores the possibility of reducing training loss during deep learning training fault recovery by using logs and checkpoint files. We performed analyses such as time-consumption at different stages of the actual model training process, and based on these theoretical analyses, the implementation details of the log-based fault recovery strategy were determined. Experiments show that, compared to a checkpointing mechanism that does not employ logging assistance, TranLogs can recover model training to a state closer to the one already reached at the moment of failure without repeated iterative computations, with an optimal improvement of about 24% and 6% for loss and accuracy, respectively. Finally, the I/O overhead was introduced in TranLogs due to the need to write to log files, which will continue to be optimized in subsequent work.

Acknowledgment

This work is supported in part by the National Key R&D Program of China (2022YFB4500405), the National Natural Science Foundation of China (U22A6001), and the Zhejiang provincial "Ten Thousand Talents Program" (2021R52007).

References

[1] Krupa J, Lin K, Flechas M A, et al. GPU coprocessors as a service for deep learning inference in high energy physics[J]. Machine Learning, Science and Technology, 2021, 2(3): 035005.

[2] Abdughani M, Ren J, Wu L, et al. Supervised deep learning in high energy phenomenology: a mini review[J]. Communications in Theoretical Physics, 2019, 71(8): 955.

[3] Baldi P, Sadowski P, Whiteson D. Searching for exotic particles in high-energy physics with deep learning[J]. Nature communications, 2014, 5(1): 4308.

[4] Hausen R, Robertson B E. Morpheus: A deep learning framework for the pixel-level analysis of astronomical image data[J]. The Astrophysical Journal Supplement Series, 2020, 248(1): 20.

[5] Burke C J, Aleo P D, Chen Y C, et al. Deblending and classifying astronomical sources with Mask R-CNN deep learning[J]. Monthly Notices of the Royal Astronomical Society, 2019, 490(3): 3952-3965.

[6] Tahmasebi P, Kamrava S, Bai T, et al. Machine learning in geo-and environmental sciences: From small to large scale[J]. Advances in Water Resources, 2020, 142: 103619.

[7] Zhong S, Zhang K, Bagheri M, et al. Machine learning: new ideas and tools in environmental science and engineering[J]. Environmental Science & Technology, 2021, 55(19): 12741-12754.

[8] Chai J, Zeng H, Li A, et al. Deep learning in computer vision: A critical review of emerging techniques and application scenarios[J]. Machine Learning with Applications, 2021, 6: 100134.

[9] Torfi A, Shirvani R A, Keneshloo Y, et al. Natural language processing advancements by deep learning: A survey[J]. arXiv preprint arXiv: 2003.01200, 2023.

[10] Bhattacharya S, Somayaji S R K, Gadekallu T R, et al. A review on deep learning for future smart cities[J]. Internet Technology Letters, 2022, 5(1): 187.

[11] Rastgoo M N, Nakisa B, Maire F, et al. Automatic driver stress level classification using multimodal deep learning[J]. Expert Systems with Applications, 2019, 138: 112793.

[12] Ni J, Chen Y, Chen Y, et al. A survey on theories and applications for self-driving cars based on deep learning methods[J]. Applied Sciences, 2020, 10(8): 2749.

[13] Truby R L, Della Santina C, Rus D. Distributed proprioception of 3D configuration in soft, sensorized robots via deep learning[J]. IEEE Robotics and Automation Letters, 2020, 5(2): 3299-3306.

[14] Arinez J F, Chang Q, Gao R X, et al. Artificial intelligence in advanced manufacturing: Current status and future outlook[J]. Journal of Manufacturing Science and Engineering, 2020, 142(11): 110804.

[15] Dina Bass. Microsoft Strung Together Tens of Thousands of Chips in a Pricey Supercomputer for OpenAI[Online]. Bloomberg, March 13, 2023. Accessed on: July 24, 2024. Available: https://www.bloomberg.com/news/articles/2023-03-13/microsoft-built-an-expensive-supercomputer-to-power-openai-s-chatgpt.

[16] Maeng K, Bharuka S, Gao I, et al. Understanding and improving failure tolerant training for deep learning recommendation with partial recovery[J]. Proceedings of Machine Learning and Systems, 2021, 3: 637-651.

[17] Mohan J, Phanishayee A, Chidambaram V. CheckFreq: Frequent,Fine-GrainedDNN Checkpointing[C]//19th USENIX Conference on File and Storage Technologies (FAST 21). USENIX, 2021: 203-216.

[18] Zhang Z, Liu T, Shu Y, et al. Dynamic Adaptive Checkpoint Mechanism for Streaming Applications Based on Reinforcement Learning[C]//28th International Conference on Parallel and Distributed Systems (ICPADS 23). IEEE, 2023: 538-545.

[19] Zhuang Y, Wei X, Li H, et al. An optimal checkpointing model with online OCI adjustment for stream processing applications[C]//27th International Conference on Computer Communication and Networks (ICCCN 18). IEEE, 2018: 1-9.

[20] Akber S M A, Chen H, Wang Y, et al. Minimizing overheads of checkpoints in distributed stream processing systems[C]//7th International Conference on Cloud Networking (CloudNet 18). IEEE, 2018: 1-4.

[21] Wang Z, Jia Z, Zheng S, et al. Gemini: Fast failure recovery in distributed training with in-memory checkpoints[C]//29th Symposium on Operating Systems Principles (SOSP 23). ACM, 2023: 364-381.

[22] Eisenman A, Matam K K, Ingram S, et al. Check-N-Run: A checkpointing system for training deep learning recommendation models[C]//19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22). USENIX, 2022: 929-943.

[23] Microsoft. Boost Checkpoint Speed and Reduce Cost with Nebula[Online]. Azure Machine Learning, August 29, 2024. Accessed on: September 23, 2024. Available: https://learn.microsoft.com/en-us/azure/machine-learning/reference-checkpoint-performance-for-large-models?view=azureml-api-2&tabs=PYTORCH.

[24] Zhang T, Liu K, Kosaian J, et al. Efficient Fault Tolerance for Recommendation Model Training via Erasure Coding[J]. Proceedings of the VLDB Endowment, 2023, 16(11): 3137–3150.

[25] Wang G, Ruwase O, Xie B, et al. FastPersist: Accelerating Model Checkpointing in Deep Learning[J]. arXiv preprint arXiv: 2406.13768, 2024.

[26] Chen M, Hua Y, Bai R, et al. A Cost-Efficient Failure-Tolerant Scheme for Distributed DNN Training[C]//41st International Conference on Computer Design (ICCD 23). IEEE, 2023: 150-157.

[27] Nicolae B, Li J, Wozniak J M, et al. Deepfreeze: Towards scalable asynchronous checkpointing of deep learning models[C]//20th International Symposium on Cluster, Cloud and Internet Computing (CCGRID 20). IEEE, 2020: 172-181.

[28] Agrawal A, Reddy S, Bhattamishra S, et al. DynaQuant: Compressing Deep Learning Training Checkpoints via Dynamic Quantization[J]. arXiv preprint arXiv: 2306.11800, 2023.

[29] Cores I, Rodríguez G, Martín M J, et al. Improving scalability of application-level checkpoint-recovery by reducing checkpoint sizes[J]. New Generation Computing, 2013, 31: 163-185.

[30] Gupta T, Krishnan S, Kumar R, et al. Just-In-Time Checkpointing: Low Cost Error Recovery from Deep Learning Training Failures[C]//19th European Conference on Computer Systems (EuroSys 24). ACM, 2024: 1110-1125.

[31] facebookresearch. facebookresearch/ResNeXt[Online]. github, August 31, 2021. Accessed on: September 23, 2024. Available: https://github.com/facebookresearch/ResNeXt.

[32] NVIDIA. NVIDIA/DeepLearningExamples[Online]. github, December 15, 2019. Accessed on: September 23, 2024. Available: https://github.com/NVIDIA/DeepLearningExamples/tree/master/PyTorch/Classification/ConvNets/resnext101-32x4d.

[33] Efron B. Bootstrap methods: another look at the jackknife[M]//Breakthroughs in statistics: Methodology and distribution. New York, NY: Springer New York, 1992: 569-593.

[34] Alluxio. Architecture[Online]. ALLUXIO, January 1, 2024. Accessed on: July 23, 2024. Available: https://docs.alluxio.io/os/user/stable/en/overview/Architecture.html.