Fast checkpointing of Large Language Models with TensorStore CHFS

Sohei Koyama Kohei Hiraga (advisor) Osamu Tatebe (advisor)
University of Tsukuba

Introduction

Large language models have attracted the attention of industry and research community because of their utility [9]. Training large language models requires enormous computational resources, and state preservation during model training, which is alson known as checkpointing, is time- and resource-intensive. Although creating checkpoints at a high frequency is desirable, the frequency of checkpoints is limited by the write bandwidth to the parallel file system [2]. In this study, we propose the TensorStore consistent hashing file system (CHFS) to accelerate checkpoint creation in training large language models. CHFS [8], an ad hoc parallel file system, is constructed by bundling the persistent memory of compute nodes. Furthermore, checkpoints are created on the CHFS to accelerate checkpoint creation. In this study, the CHFS was integrated into TensorStore [3] so that it could be used seamlessly from T5X [6], Orbax [5], and JAX [1]. The time required for checkpoints creation in training a large language model was also measured to demonstrate the effectiveness of using the CHFS. The evaluation results revealed that the bandwidth of checkpoint creation can be improved by up to ~4.5 times using the CHFS.

Related Research

Check-N-Run [2] is a scalable checkpoint system used to train large-scale recommendation models. It uses incremental checkpointing and quantization techniques to reduce data size without compromising accuracy. Our study did not reduce the data size, but improved the storage performance, accelerating checkpoint creation.

Background

TensorStore

TensorStore [3] is a library designed for efficient reading and writing of large-scale multi-dimensional arrays. One use case for TensorStore is checkpointing of the language model. TensorStore has already been used for checkpoint management in JAX [4]. The C++ implementation of the same achieves high performance by automatically using multiple cores for task encoding/decoding and performing I/O operations in parallel to maximize throughput. TensorStore provides a Python API that uses the same indexing and operation syntax as standard NumPy operations. The Listing 1 illustrates the method of using TensorStore.

```
import tensorstore as ts
dataset = ts.open({
  'kvstore': { 'driver': 'file', 'path': '/path/to/dataset/', },
  'driver': 'n5', 'metadata': { ... }, 'create': True, 'delete_existing': True, }).result()
dataset[80:82, 99:102].write([[1, 2, 3], [4, 5, 6]]).result()
```

Listing 1: The method of using TensorStore

CHFS

CHFS [8] is an ad hoc parallel file system that uses Intel Optane Persistent Memory installed on compute nodes. Because CHFS distributes metadata, it has high scalability for metadata access performance. CHFS provides a CHFS Client written in the C language, which offers improved performance than that of the access via FUSE. CHFS constructs a parallel file system by executing a daemon process, chfsd, on each compute node.

Design and Implementation of TensorStore CHFS

Figure 1 depicts the checkpoint creation while training large language models using T5X. In HPC clusters, checkpoints were created on the parallel file system Lustre. TensorStore uses the "file" key-value store driver to create checkpoints on Lustre. The compute nodes and Lustre were connected using high-speed interconnects. However, the frequency of checkpoint creation is limited by the lack of write bandwidth of Lustre.

Figure 2 depicts the checkpoint creation process when TensorStore CHFS is used. TensorStore connects to the chfsd daemon via the CHFS client and sends the checkpoint data to chfsd. Then, chfsd writes the checkpoint data to the Intel Optane Persistent Memory installed on the compute node.

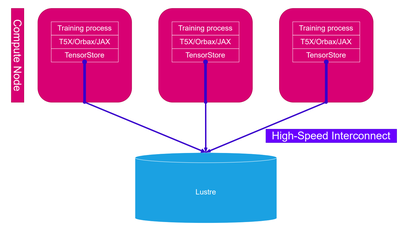

Figure 1. Checkpoint process using Lustre filesystem.

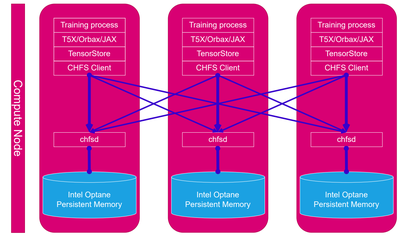

Figure 2. Checkpoint process using TensorStore CHFS.

TensorStore natively supports multiple storage drivers, including Google Cloud Storage, local and network file systems, and in-memory storage systems. In this study, the CHFS was added to TensorStore as a key-value storage driver. The Listing 2 outlines the method to use TensorStore CHFS.

```
dataset = ts.open({
  'driver': 'zarr', 'kvstore': { 'driver': 'chfs', 'path': '/path/to/dataset/', },
}, dtype=ts.uint32, shape=[1000, 20000], create=True).result()
```

Listing 2: The method to use TensorStore CHFS

TensorStore CHFS is based on the "file" key-value storage driver and uses the CHFS API instead of the Posix API. It majorly differs from the "file" key-value storage in that the "file" key-value storage has ACID guarantees; however, the TensorStore CHFS does not have ACID guarantees because it focuses on performance. ACID guarantees are not required for checkpoints creation.

Evaluation

The bandwidth of checkpoint creation was measured with and without TensorStore CHFS using T5X. To measure the bandwidth, the parameter size of the T5 1.1 model was changed, and the model was divided into 8, 16, and 32 nodes. The corresponding checkpoint sizes were 115, 179, and 257GiB. The evaluation was performed using the Big Memory Supercomputer, Pegasus, at the University of Tsukuba. The Lustre file system was used for comparison with the TensorStore CHFS. Lustre is a 7.1 PB DDN EXAScaler with an effective 40GB/s. Each compute node of the Pegasus is equipped with a 2TiB Intel Optane Persistent Memory 300 series and an Nvidia H100 Tensor Core GPU. The limit of the TensorStore parameter, file_io_concurrency, was set to 128. The evaluation results are depicted in Figure 3. In the case of 32 nodes, the bandwidth of checkpoint creation with TensorStore CHFS was 78 GiB/s, which is 4.5 times that with Lustre.

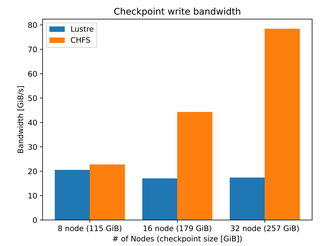

Figure 3. Checkpoint write bandwidth.

Conclusion

The frequency of checkpoint creation in large language models is limited by the write bandwidth to a parallel file system. In this study, we aim to reduce the checkpoint creation time by writing to the Intel Optane Persistent Memory installed on the compute nodes. We propose TensorStore CHFS, a storage driver that adds an ad hoc parallel file system CHFS to the TensorStore. The proposed method succeeded in increasing the checkpoint creation bandwidth of the T5 1.1 model by 4.5 times on 32 nodes.

References

[1] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018.

[2] Assaf Eisenman, Kiran Kumar Matam, Steven Ingram, Dheevatsa Mudigere, Raghuraman Krishnamoorthi, Krishnakumar Nair, Misha Smelyanskiy, and Murali Annavaram. Check-N-Run: A checkpointing system for training deep learning recommendation models. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, pages 929–943, 2022.

[3] Google. Tensorstore. `https://google.github.io/tensorstore/`. Accessed: July 31, 2023.

[4] Google. Tensorstore for high-performance, scalable array storage. `https://ai.googleblog.com/2022/09/tensorstore-for-high-performance.html`. Accessed: July 31, 2023.

[5] Orbax. Orbax documentation. `https://orbax.readthedocs.io/en/latest/`. Accessed: July 31, 2023.

[6] Adam Roberts, Hyung Won Chung, Anselm Levskaya, Gaurav Mishra, James Bradbury, Daniel Andor, Sharan Narang, Brian Lester, Colin Gaffney, Afroz Mohiuddin, et al. Scaling up models and data with `t5x` and `seqio`. *arXiv preprint arXiv:2203.17189*, 2022.

[7] Claude E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27(3):379–423, 1948.

[8] Osamu Tatebe, Kazuki Obata, Kohei Hiraga, and Hiroki Ohtsuji. Chfs: Parallel consistent hashing file system for node-local persistent memory. In *International Conference on High Performance Computing in Asia-Pacific Region*, pages 115–124, 2022.

[9] Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, et al. A survey of large language models. *arXiv preprint arXiv:2303.18223*, 2023.

skoyama@hpcs.cs.tsukuba.ac.jp