High-throughput Small File Access for Large-scale Machine Learning Applications

Hiroki Ohtsuji, Erika Hayashi, Takuya Okamoto, Eiji Yoshida
FUJITSU LIMITED
Kawasaki, Japan

Osamu Tatebe
University of Tsukuba
Tsukuba, Japan

I. Introduction

Storage systems used in high-performance computing are required to provide a high-throughput fine-grained data access method due to the increasing amount of data handled by machine learning applications. For example, according to [1], the number of data handled by modern language models is increasing at a rate of 10x per year. As the number of data increases, the number of files stored in the storage system also increases. Considering the pre-processing of datasets with various type of scripts, archiving formats optimized for learning frameworks such as TFRecord cannot fit all environments. Since access to the data stored in the shared storage system is performed via the network, the existence of a large number of files results in a large amount of communication and data processing. In this abstract, we describe a method for high throughput data access to a large number of small files using the RPC implementation, which is designed to use NVM as a storage device, and evaluation results of its access performance.

II. Overhead and Optimization Method of Remote File Access

Remote file access to shared storage has multiple bottlenecks between the application and the server. These bottlenecks are caused by system calls and multiple function calls required for network communication and data processing. The authors aim to optimize this path by speeding up the processing in the server, client-side programs, and application interfaces. The first two of them are realized by polling-based asynchronous RPC, and the last one is accelerated by asynchronous internal parallel processing while maintaining synchronous semantics of the application interface. This abstract describes the first two parts of the above items.

III. The Lightweight Remote File Access

Lightweight remote file access can reduce the overhead incurred on both the client and server sides. As shown in Fig. 1, the I/O system call issued by the application is hooked by the preload library, and the request is sent to the server through asynchronous communication regardless of the type of the I/O request. On the server, data access is also performed asynchronously as memory access.

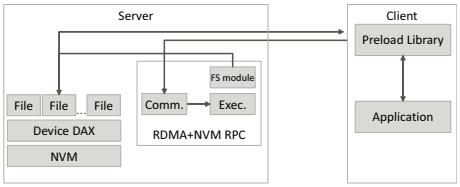

Fig. 1: A functional diagram of the lightweight remote file access mechanism.

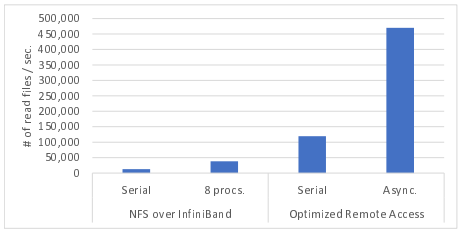

Fig. 2: Throughput comparison between conventional I/O and lightweight I/O mechanism. Serial is when the application issues requests sequentially, and the other is when the application issues requests in parallel or asynchronously.

IV. Evaluation

A comparison of the access performance between conventional remote file access and lightweight remote file access was performed. The former is achieved by accessing tmpfs on a remote node shared by NFS, while the latter is achieved by using an access library and RPC mechanism developed by the authors. The size of the target file is 1KB. Since the purpose of this evaluation is to know the processing performance of remote file access, the data to be read was placed in the consecutive space on DRAM. The performance of sequential access and asynchronous access were evaluated. In the case of NFS, asynchronous access was performed by running eight processes of the benchmark program simultaneously. In the case of lightweight access, asynchronous RPC requests are used. The results show that the lightweight implementation is 9.5 times faster for sequential access and 12.3 times faster for asynchronous access. It can be expected to provide sufficient access performance for machine learning applications over the next few years.

V. Conclusion and Future Work

This abstract described the problem of small file access, which is increasing due to new workloads such as machine learning and showed the design of the lightweight remote file access method to reduce the overhead. Our implementation has demonstrated higher performance compared to the conventional method. The authors are planning to implement the design to a full-featured parallel file system to evaluate the performance with real applications.

References

[1] Open AI. Better Language Models and Their Implications. https://openai.com/blog/better-language-models/.
[2] readfile: implement readfile syscall. https://lore.kernel.org/lkml/20200704140250.423345-2-gregkh@linuxfoundation.org/.