Using lossy compression for interactive analysis over network

Rei Aoyagi
Graduate School of Information Sciences, Tohoku University
Sendai, Miyagi, Japan

Keichi Takahashi
Cyberscience Center, Tohoku University
Sendai, Miyagi, Japan

Yoichi Shimomura
Cyberscience Center, Tohoku University
Sendai, Miyagi, Japan

Hiroyuki Takizawa
Cyberscience Center, Tohoku University
Sendai, Miyagi, Japan

1 INTRODUCTION

Today's scientific simulations require a massive amount of computing resources. Thus, most of scientific simulations are executed on high performance computing (HPC) systems hosted by HPC centers. To gain scientific insights, researchers conduct post-processing on the simulation results. A common way to carry out post-processing is to run an interactive data analysis tool such as Jupyter Notebook on the HPC system and retrieve the post-processed results. In certain scenarios, transferring the simulation results directly from the center becomes essential. This is especially relevant when utilizing specialized hardware or proprietary software for data analysis. In such cases, a portion of the data can be streamed over the network. However, maintaining interactivity presents two challenges: (1) limited network bandwidth and (2) network latency.

Dong et al. proposed a data staging approach that automatically moves data from HDDs to SSDs to accelerate post-processing [2]. They assumed a simple access pattern that sequentially scans the elements of an array one by one. Since the address to be accessed next can be predicted, they employed prefetching. The prefetched elements in SSDs were replaced by first-in first-out order. However, the access pattern in practice is not as predictable as they assumed. Furthermore, they did not discuss staging over the network.

2 PROPOSAL

We propose middleware to support interactive array analysis over network. By using error-bounded lossy compression, we increase the effective network bandwidth. In addition, by using multi-level caching, we hide the network latency. The cache hit ratio is enhanced by prefetching mechanisms.

The overall design of the middleware is illustrated in Fig. 1. In our proposal, data is transferred from server to client in the following four steps: (1) Reading a block from storage, (2) compressing the block, (3) transferring the compressed block over the network, and (4) decompressing the block. Step (1) and (2) are conducted on the server, while step (3) and (4) are conducted on the client. We introduce L1 to L4 cache at each step of the transfer to take advantage of access locality. Furthermore, we introduce prefetchers at each cache to improve the cache hit ratio.

Since the access pattern of interactive analysis on a multidimensional array usually exhibits spatial locality, we propose a cache replacement and prefetching policy as follows. The prefetcher at each cache level keeps fetching the blocks around the last accessed block until the cache becomes full. Each cache evicts blocks located farther than a certain distance from the last accessed block.

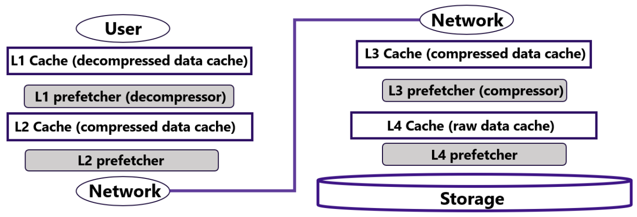
User
L1 Cache (decompressed data cache)
L1 prefetcher (decompressor)
L2 Cache (compressed data cache)
L2 prefetcher
Network
Network
L3 Cache (compressed data cache)
L3 prefetcher (compressor)
L4 Cache (raw data cache)
L4 prefetcher
Storage

Figure 1: Overview of the proposed middleware

Table 1: Average latency of 100 requests in ms

TileDB	Enabled cache levels					
	No cache	L1	L2	L3	L4	L2+L4
697	518	496	302	478	494	286

3 EVALUATION

We implement our middleware using MGARD [1] for lossy compression, HDF5 for array management in storage and Python's standard library for key-value store. The server and client communicate over HTTP. The relative error tolerance for the compression was set to 0.1. We compared our proposal with TileDB [3] by measuring the average latency for 100 requests. Each request retrieves a 64 MiB three-dimensional block. The L1, L2 and L3 cache capacities were set to 512 MiB and L4 cache capacity was set to 1024 MiB.

Table 1 shows the evaluation results. By comparing TileDB with No Cache, we can see that compression reduces the latency. By comparing No Cache with L4 cache we can see prefetching data from Storage reduces the latency. L1 cache also reduces the latency but not as much as L2 cache because L2 cache holds compressed blocks while L1 cache holds eight decompressed blocks, which is not large enough to improve the cache hit ratio.

REFERENCES

[1] Mark Ainsworth, Ozan Tugluk, Ben Whitney, and Scott Klasky. 2019. Multilevel Techniques for Compression and Reduction of Scientific Data—The Multivariate Case. *SIAM Journal on Scientific Computing* 41, 2 (2019), A1278–A1303.
[2] Bin Dong, Teng Wang, Houjun Tang, Quincey Koziol, Kesheng Wu, and Suren Byna. 2018. ARCHIE: Data Analysis Acceleration with Array Caching in Hierarchical Storage. In *2018 IEEE International Conference on Big Data*. 211–220.
[3] Stavros Papadopoulos, Kushal Datta, Samuel Madden, and Timothy Mattson. 2016. The TileDB Array Data Storage Manager. 10, 4 (Nov. 2016), 349–360.