IOPathTune: Adaptive Online Parameter Tuning for Parallel File System I/O Path

Md. Hasanur Rashid
mrashid2@uncc.edu
University of North Carolina at Charlotte
USA

Dong Dai
ddai@uncc.edu
University of North Carolina at Charlotte
USA

ABSTRACT

Motivation. In High-Performance Computing (HPC) clusters, parallel file systems play a vital role in providing timely access to extreme-scale data. Hence its performance is critical. The key to delivering extreme I/O performance in parallel file systems relies on the efficient I/O paths connecting clients with the storage servers. To achieve an efficient I/O path, the parallel file systems must have proper settings on multiple PFS parameters that control how data flows in the I/O path. Although carefully picked, the default settings often fail to deliver optimal performance for diverse and changing workloads.

To effectively tune I/O path-relevant parameters for parallel file systems, we look for several key properties: adaptive, timely, and flexible. The tuning framework should adjust parameters adaptively when workloads change and respond to runtime variables such as I/O contentions. The tuning decision delivery needs to be fast due to the short burst nature of the workloads. The tunable parameters should also be modifiable dynamically. The framework must accommodate the flexibility to tune different clients differently. Due to the limited I/O, computation, and communication resources for performance tuning, the framework should also avoid the expensive probing or profiling of the HPC environment.

Design of IOPathTune. In this study, we propose IOPathTune, a new tuning framework designed to tune Lustre I/O Path online from the client side adaptively. We focus on two parameters after closely investigating the Lustre I/O path: `max_pages_per_rpc` (maximum number of pages in a single RPC) and `max_rpcs_in_flight` (maximum number of RPCs in flight). Both of them belong to the Lustre Portal RPC (PtlRPC) subsystem. They are dynamically tunable, take immediate effect, and control the I/O flow for each Lustre Client. These important features align well with our goals.

One critical design choice of IOPathTune is to avoid probing storage servers or other compute nodes. We design it to solely depend on the statistics collected by the PFS client library. Our tuning algorithm proceeds with tuning every ten seconds, and in each turn, it chooses to tune either of the two parameters alternately. The tuning action consists of either multiplying or dividing the parameter value by two each time, much like the primary approach of TCP congestion control.

Specifically, during tuning, we calculate different metrics along the I/O path. The metrics we derive are: how much data is in the dirty page cache, how fast the pages are getting cached, how quickly the client can generate RPCs, and what speed the client is achieving while transferring RPCs for the last observation period. By comparing these metrics, if we observe that our last tuning action improves the bandwidth, we reciprocate the previous tuning action. Otherwise, we do the opposite of the last tuning action. We also observe whether I/O contentions are developing or not based on these metrics. In the case of I/O contentions, we adopt a conservative approach: assign blame on our previous tuning action and revert it.

Evaluation. To test the efficacy of our framework IOPathTune, we leverage CloudLab [1] to conduct the evaluations (system setup, software details, and representative results are displayed on the poster). We tested 20 different Filebench [3] workloads with varying I/O patterns (random or sequential), I/O operations (read or write), I/O request sizes, number of processes and threads, and others. We ran three different evaluations and checked how IOPathTune handles different scenarios.

The first test was standalone workload executions from a single client. We observed either on par with slight degradation or better performance than the default configuration across all workloads. Some of the most considerable improvement includes 231%, 113%, and 96% for Filebench fivestreamwriternd, whole file read-write, and sequential read-write workloads. Our second kind of test was dynamic testing, where we continuously changed the workloads after running for 300 seconds. We changed the workload six times at a single run and experimented with five runs, each with six different combinations of workloads. We observed consistent improvements from IOPathTune similar to the standalone performance evaluation, indicating our algorithm can quickly catch up and converge to better configurations.

Our final type of test was executing different workloads from different clients at the same time. We compared IOPathTunes' performance with the default configuration and CAPES [2] execution. IOPathTune improved the total bandwidth of the cluster by 129.30% compared to the default execution and by 89.57% compared to the CAPES execution. The findings prove that our proposed IOPathTune framework can independently tune parameters from different clients to achieve better performance.

CCS CONCEPTS

• **Computing methodologies** → *Shared memory algorithms.*

REFERENCES

[1] Dmitry Duplyakin, Robert Ricci, Aleksander Maricq, Gary Wong, Jonathon Duerig, Eric Eide, Leigh Stoller, Mike Hibler, David Johnson, Kirk Webb, et al. 2019. The Design and Operation of {CloudLab}. In *2019 USENIX annual technical conference (USENIX ATC 19)*. 1–14.

[2] Yan Li, Kenneth Chang, Oceane Bel, Ethan L Miller, and Darrell DE Long. 2017. CAPES: Unsupervised storage performance tuning using neural network-based deep reinforcement learning. In *Proceedings of the international conference for high performance computing, networking, storage and analysis*. 1–14.

[3] Vasily Tarasov, Erez Zadok, and Spencer Shepler. 2016. Filebench: A flexible framework for file system benchmarking. *USENIX; login* 41, 1 (2016), 6–12.