Toward Heterogeneity-Aware Striping in Lustre

Sooyoung Lim, Jaegi Son, and Dongmin Kim
Intelligent IDC Group, Korea Electronics Technology Institute (KETI)
Seongnam-si, Gyeonggi-do, Republic of Korea
{slim, jgson, dmkim}@keti.re.kr

***Abstract*—Lustre is a widely adopted parallel file system in high-performance computing (HPC) environments. However, as clusters increasingly incorporate heterogeneous storage devices with varying performance characteristics, Lustre's default striping mechanism fails to fully leverage the capabilities of faster devices and can even degrade I/O performance. We propose an adaptive, heterogeneity-aware striping mechanism that automatically distributes I/O workload based on the resource capability of each storage device.**

***Index Terms*—Distributed systems, File systems, Heterogeneous cluster, High-performance Computing (HPC), Lustre, Striping**

I. Introduction

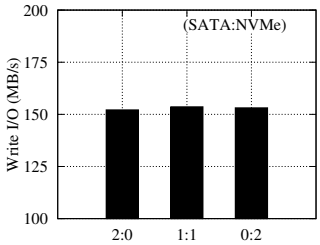
Lustre has become the de facto standard of scalable parallel file systems in HPC and AI workloads, where massive I/O performance is achieved through data striping across multiple Object Storage Targets (OSTs) [1]. Modern Lustre deployments increasingly adopt heterogeneous storage configurations, combining SATA SSDs, SAS drives, and NVMe SSDs [2]. In such heterogeneous environments, uniform striping may lead to suboptimal performance due to unbalanced I/O workloads and uncoordinated lock contention.

Several enhancements to Lustre striping have been proposed to improve I/O performance and adaptability. Overstriping allows multiple stripes per OST to mitigate contention and improve parallelism [3]. Dynamic striping adapts stripe parameters based on file size or access patterns [4], [5]. However, these approaches do not account for the performance disparity inherent in heterogeneous storage environments. This limitation becomes critical when workloads require consistent high throughput.

Our experiments reveal that Lustre's performance remains largely unaffected by the type of underlying storage devices. This results from a critical limitation of Lustre's default striping policy, which statically assigns I/O without considering discrepant storage performance. We propose a heterogeneity-aware striping mechanism that incorporates computing resource-based weighting and adaptive OST selection for more balanced and efficient use of heterogeneous storage environments.

II. Motivation

To evaluate the impact of storage configurations on Lustre performance, we conducted a series of IOR write benchmarks. To investigate striping performance, each Object Storage Server (OSS) is configured with two OSTs, employing direct I/O to bypass the cache and accurately capture the native device performance. Figure 1 shows write performance on heterogeneous storage configurations with SATA and NVMe SSDs. It displays three stripe distribution scenarios: all stripes on two SATA SSDs, uniform striping across a SATA SSD and an NVMe SSD, and all stripes on two NVMe SSDs. In this dual-OSS setup, all three striping distributions perform nearly identical write bandwidth (i.e., 153MB/s). Given that RPC and transfer limits were fully relaxed in our experiments, this counterintuitive result suggests that factors beyond network or RPC constraints dominate the observed performance. This study demonstrates that naïve stripe allocation to high-performance devices does not guarantee optimal performance. As a result, storage performance becomes constrained not by hardware potential but by policy limitations.



Heterogeneous Storage Write Performance

Fig. 1. Write I/O bandwidth of IOR benchmarks, where each process wrote 1GB of data, for a total of 24GB.

However, the existing Lustre architecture does not provide mechanisms to handle computing resource heterogeneity. By default, clients receive a static list of OSTs (`lov_config`) from the Management Service (MGS) and apply a round-robin or user-defined striping policy across them. However, no performance feedback loop exists between the OSS nodes and the client, meaning high-load or underperforming OSTs are treated equally with idle or high-performance ones. Moreover, no component in the management server centrally monitors resources and incorporates them into striping decisions. While such data is available through tools like */proc*, */sys*, *vmstat*, or *iostat* on the OSS nodes, clients cannot query this information in real-time during stripe allocation. Given the absence of resource-aware striping at the client side and centralized monitoring in the management service, integrating real-time OSS performance metrics into striping decisions is essential for efficient resource utilization.

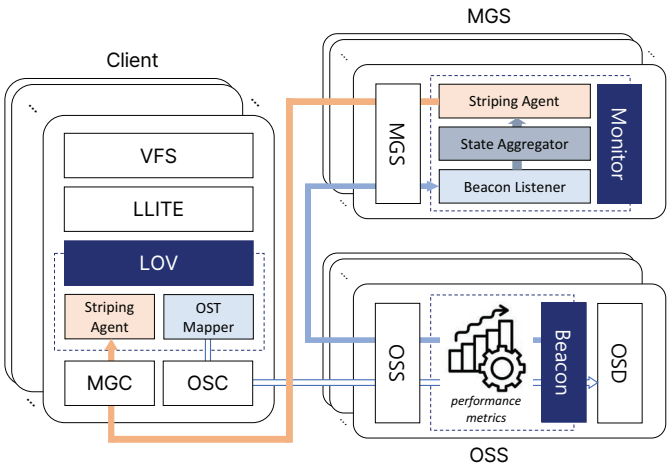


Fig. 2. The Redesigned Lustre architecture and the overall striping process. Colored squares mark the redesigned Lustre components. The hollow arrows indicate the heterogeneity-aware striping mechanism, which adopts the periodic mechanism represented by the plain ones.

III. Proposed Architecture

Figure 2 presents the redesigned Lustre architecture and describes the overall process of the heterogeneity-aware striping mechanism ($Beacon \rightarrow Monitor \rightarrow LOV \rightarrow OSD$). We propose a modular extension to the legacy architecture that integrates system resource awareness into the MGS and client-side striping logic. The design is structured in three modules: (1) the deployment of a lightweight resource beaconing system on each OSS, (2) the deployment of a resource monitoring layer in the MGS to fetch, aggregate and transmit resource utilization logs, and (3) the modification of the client's Logical Object Volume (LOV) to utilize these resource insights during file creation.

A. Resource Beaconing on OSS

In the first stage, each OSS is equipped with a lightweight daemon that periodically collects system-level metrics. These include CPU utilization, memory usage, and storage performance collected via standard system monitoring tools, as well as OST-level utilization statistics obtained from the Object Storage Device (OSD) layer (i.e., osd-ldiskfs) through Lustre's `procfs` interface. The collected data is encapsulated in a structured *ResourceBeacon* and periodically transmitted to the MGS via RPC at configurable intervals.

B. Resource Monitoring on MGS

The core of the proposed architecture is the *ResourceMonitor*, a component integrated into the MGS that fetches and aggregates resource metrics from all OSS nodes. By organizing this novel component into three functional modules, clients no longer need to parse raw system metrics while maintaining the legacy metadata flow through the MDS.

- ***Beacon Listener*** receives incoming *ResourceBeacons* from each OSS and extracts relevant resource metrics.
- ***State Aggregator*** retains an in-memory database of per-OST resource states. It optionally exports this information via `procfs` or `sysfs` for integration with system-level tools. It also marks certain OSTs as degraded or overloaded based on customizable thresholds to dynamically blacklist them.
- ***Striping Agent*** responds to client queries with preprocessed resource summaries such as ranked OST lists or filtering lists.

C. Striping on Client LOV Layer

Clients retrieve metadata—including stripe count, size, and target OSTs—from the MDS at file creation time and rely on these static instructions during the file I/O. To enable resource-aware striping without disrupting this legacy workflow, we extend the LOV layer to interact with the *ResourceMonitor* with two additional modules.

- ***Striping Agent*** initiates a query to the MGS-side *Striping Agent* to obtain up-to-date resource information for available OSTs. This communication occurs before stripe selection, allowing the client to receive a filtered or ranked list of OST candidates based on the current resource utilization.
- ***OST Mapper*** selects the target OSTs for data striping based on the given policy from the client-side *Striping Agent*. This information is then passed to the standard `lov_object_alloc()` function, which creates the LOV object accordingly.

IV. Conclusion and Future Work

This paper proposed the redesigned striping mechanism in Lustre filesystem to improve resource utilization in heterogeneous storage systems. Our approach introduces a modular extension that integrates OSS-side resource beaconing, centralized resource monitoring on the MGS, and a modified client LOV layer, enabling informed striping decisions based on real-time system resource usage. Future work will involve implementing the proposed striping policies and evaluating the effectiveness of real-time re-striping mechanisms under workload prediction.

Acknowledgment

This work was supported by Institute for Information & communications Technology Promotion(IITP) grant funded by the Korea government(MSIT) (No. RS-2025-02215256), Development of High-Speed Parallel File System Technology for AI Semiconductor Cloud

References

[1] A. George, A. Dilger, M. J. Brim, R. Mohr, A. Shehata, J. Y. Choi, A. M. Karimi, J. Hanley, J. Simmons, D. Manno, V. M. Vergara, S. Oral, and C. Zimmer, "Lustre unveiled: Evolution, design, advancements, and current trends," *ACM Trans. Storage*, vol. 21, no. 3, pp. 1–109, Jun. 2025. [Online]. Available: https://doi.org/10.1145/3736583

[2] N. S. Islam, X. Lu, M. Wasi-ur Rahman, D. Shankar, and D. K. Panda, "Triple-h: A hybrid approach to accelerate hdfs on hpc clusters with heterogeneous storage architecture," in *2015 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, 2015, pp. 101–110.

[3] M. Moore and P. Farrell, "Exploring lustre overstriping for shared file preformance on disk and flash," in *2019 Lustre User Group (LUG) Conference*, 2019.

[4] J. Reed, J. Archuleta, M. J. Brim, and J. Lothian, "Evaluating dynamic file striping for lustre," 2015. [Online]. Available: https://arxiv.org/abs/1504.06833

[5] G. Xian, W. Yang, Y. Tan, J. Feng, Y. Li, J. Zhang, and J. Yu, "Mobilizing underutilized storage nodes via job path: A job-aware file striping approach," *Parallel Computing*, vol. 121, p. 103095, 2024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167819124000334