HadaFS: A File System Bridging the Local and Shared Burst Buffer for Exascale Supercomputers

21st USENIX Conference on File and Storage Technologies

Xiaobin He; Bin Yang; Jie Gao; Wei Xiao; Qi Chen; Shupeng Shi; Dexun Chen; Weiguo Liu; Wei Xue; Zuo-ning Chen

Paper Notes

By JeongHa Lee

Abstract

Current supercomputers introduce SSDs to form a Burst Buffer (BB) layer to meet the HPC application's growing I/O requirements. BBs can be divided into two types by deployment location. One is the local BB, which is known for its scalability and performance. The other is the shared BB, which has the advantage of data sharing and deployment costs. How to unify the advantages of the local BB and the shared BB is a key issue in the HPC community. We propose a novel BB file system named HadaFS that provides the advantages of local BB deployments to shared BB deployments. First, HadaFS offers a new Localized Triage Architecture (LTA) to solve the problem of ultra-scale expansion and data sharing. Then, HadaFS proposes a full-path indexing approach with three metadata synchronization strategies to solve the problem of complex metadata management of traditional file systems and mismatch with the application I/O behaviors. Moreover, HadaFS integrates a data management tool named Hadash, which supports efficient data query in the BB and accelerates data migration between the BB and traditional HPC storage. HadaFS has been deployed on the Sunway New-generation Supercomputer (SNS), serving hundreds of applications and supporting a maximum of 600,000-client scaling.

Problem Statement and Research Objectives

- The contradiction between BBs' scalability and application behaviors
  - With the barrier to exascale computing being broken, the **I/O concurrency of cutting-edge supercomputers can reach hundreds of thousands**, which **stresses the scalability of BBs**.
- Complex metadata management mismatches application behaviors
  - Traditional file systems are designed for generality, so their file management is implemented in **the directory tree structure** and strictly follows the POSIX protocol.
  - However, in **HPC**, **computing nodes are generally responsible for reading and writing data** and **rarely perform directory tree access**. ➔ how to relax POSIX remains a huge challenge.
- Inefficiencies in data management
  - A recent study found that although **most applications** on Summit and Cori **can use the BB to speed up I/O performance, the BB utilization is low**, and it is necessary to develop flexible data management tools for users.
  - So, the BB system needs to consider efficiently and conveniently migrating data between the BB and the GFS. Data migration between the BB and the GFS can be divided into two types.
    - In transparent data migration, software automatically migrates the BB data to the GFS in blocks or files, which may cause a large amount of unnecessary data migration.
    - In non-transparent migration, data migration often needs computing nodes to participate, leading to the computing resource being idle during the data migration process and wasting resources.

Proposed Method

Localized Triage Architecture (LTA)

- Both existing methods have certain limitations.
  - The traditional kernel file system handles the application's I/O requests by mounting the file system through the operating system.
  - An alternative method is to mount the file system through the application and bypass the kernel.
- Combines both advantages in a new approach : **HadaFS follows the idea of bypassing the kernel and uses it by directly mounting the client into the application.**
- In HadaFS, the server connected to a client is called the **bridge server**.
  - The bridge server is **responsible for handling all I/O requests generated by the client and writes data to the underlying file**. Servers are a fully connected structure.
- HadaFS adopts the method of connecting only one server per client.
  - In order to control the number of clients served by a single server at the same time and avoid the performance bottleneck caused by too many clients connected to a single server.
  - Note that if the storage space of one bridge server is filled up, all the clients connected to it will automatically switch to another HadaFS server.

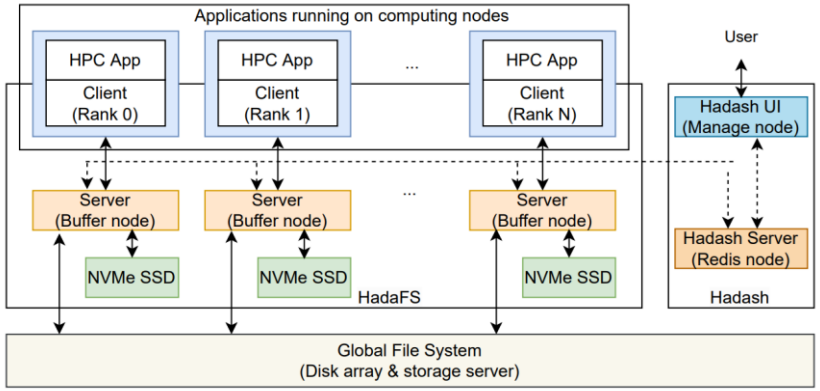


Figure 2: Architecture of HadaFS

Proposed Method

Namespace and metadata handling

- In order to improve the scalability and performance, HadaFS **abandons the idea of directory trees.**
  - For a file in HadaFS, its data is stored on the bridge server of the HadaFS client that generated the file, and its metadata storage location is determined by the path hash.
  - Files' metadata are stored by **key-value**, and **the file path is a globally unique ID (key)**.
- Two kinds of metadata databases are maintained on each HadaFS server.
  - **Local metadata database (LMDB)**: stores the first and fourth category metadata information of the file locally, and the file's local identification (LID) is the local path corresponding to the file.
  - **Global metadata database (GMDB)**: stores the first two and the fourth categories of metadata information.

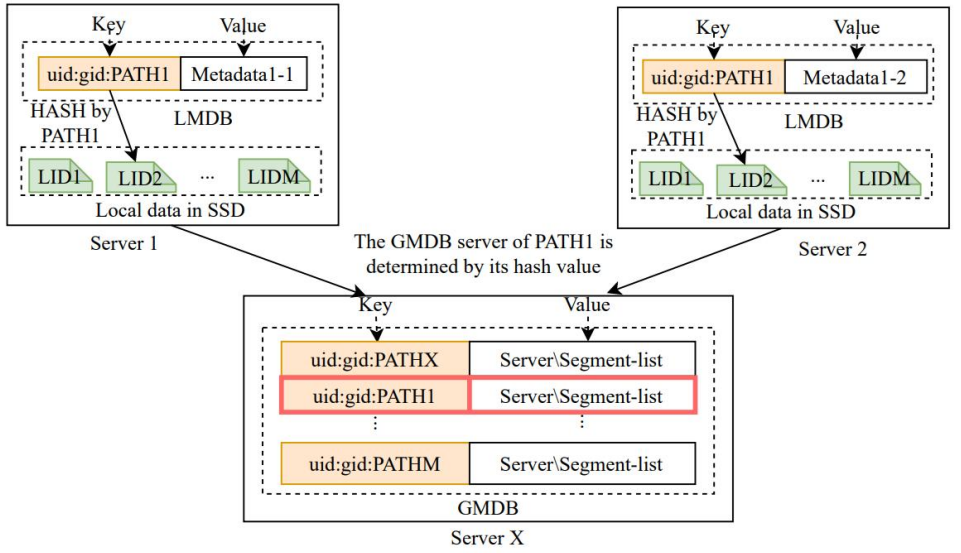


Figure 3: Two K-V tables on the HadaFS server

Proposed Method

HadaFS I/O control and data flow

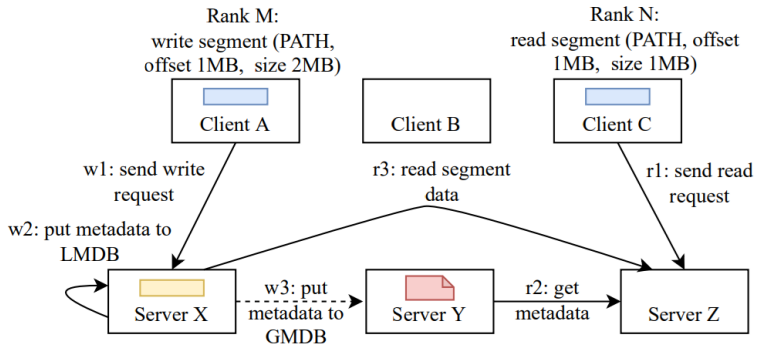


Figure 4: An example of HadaFS I/O control and data flow

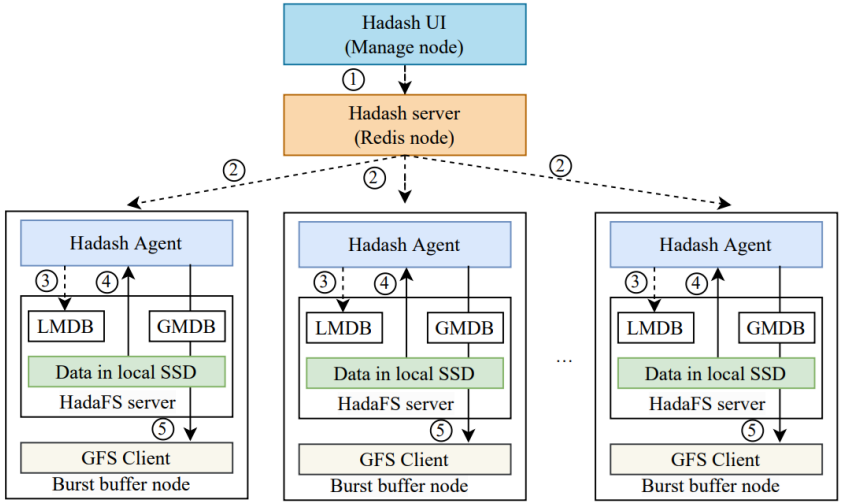


Figure 5: The stage-out flow of Hadash

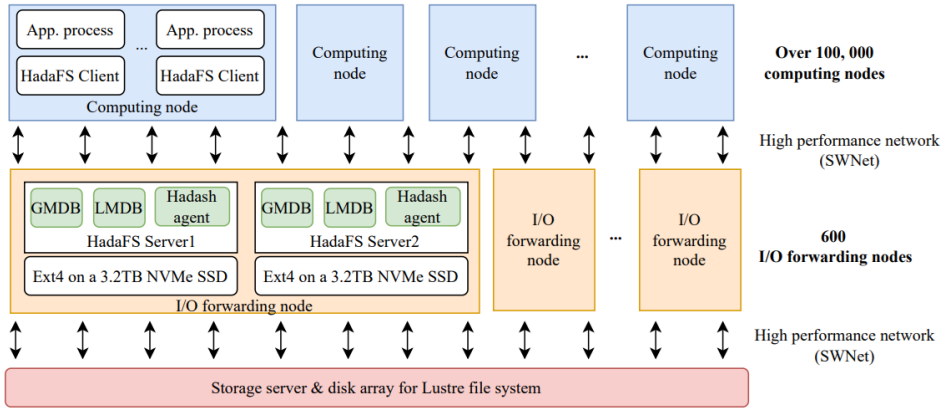
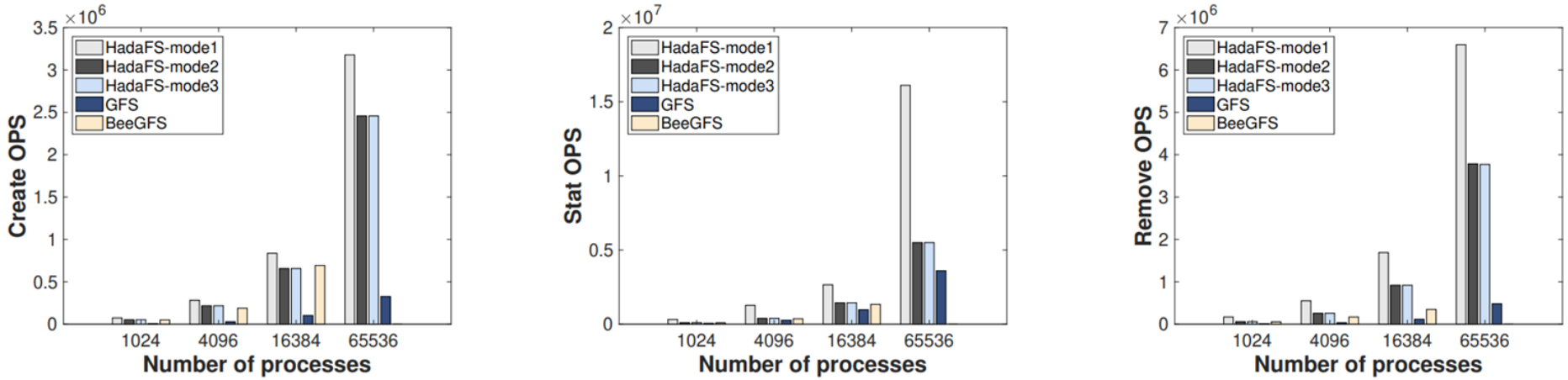


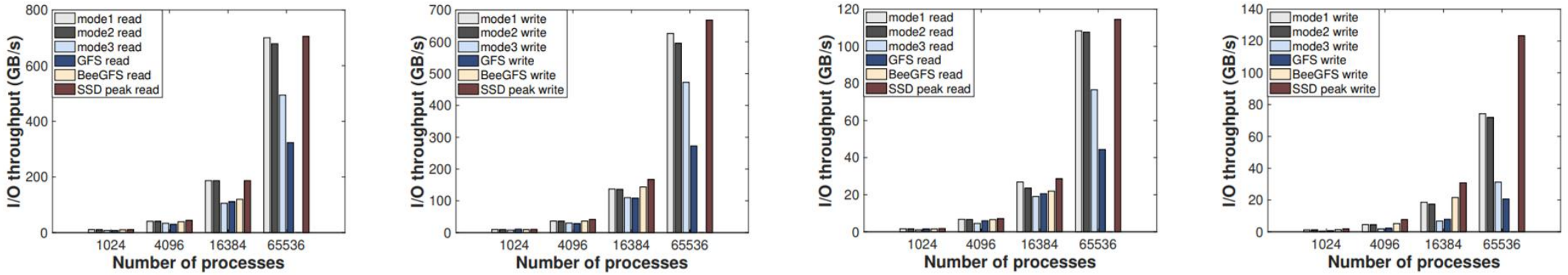
Figure 6: The deployment of HadaFS

Evaluation and Results



(a) create OPS (b) stat OPS (c) remove OPS

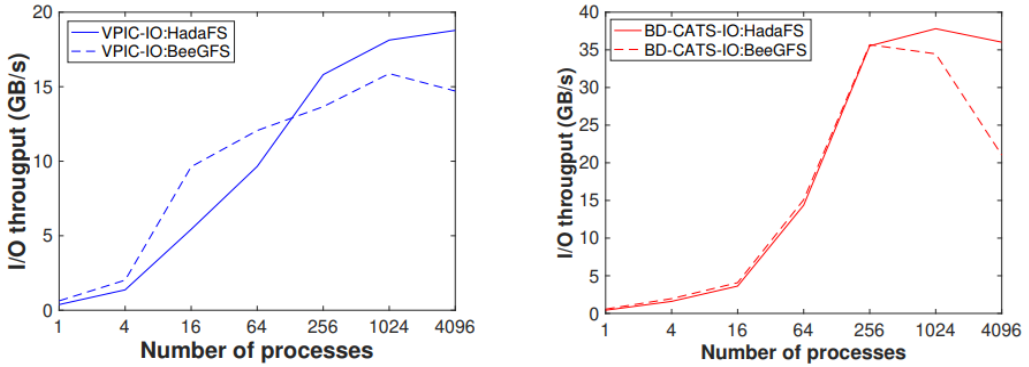
Figure 7: Metadata performance comparison



(a) Sequential read performance (b) Sequential write performance (c) Random read performance (d) Random write performance

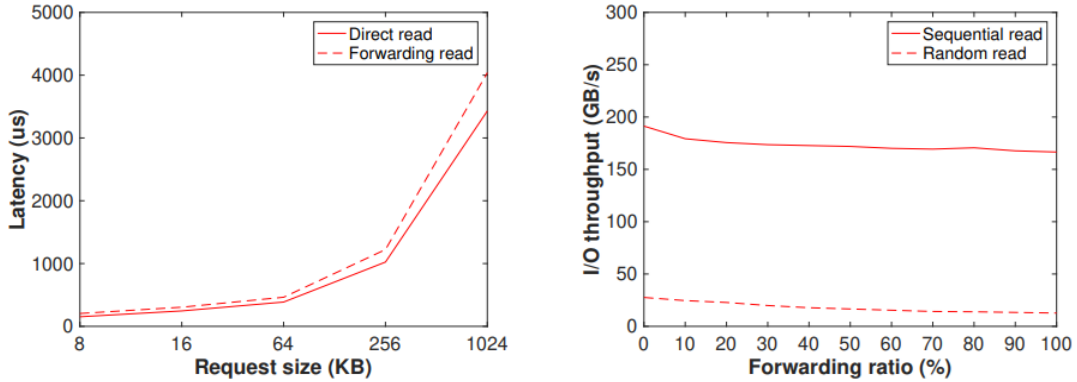
Figure 8: I/O throughput comparison

Evaluation and Results



(a) Write performance (b) Read performance

Figure 10: Performance evaluation on the shared file



(a) Forwarding latency (b) Forwarding throughput

Figure 13: Forwarding evaluation of HadaFS

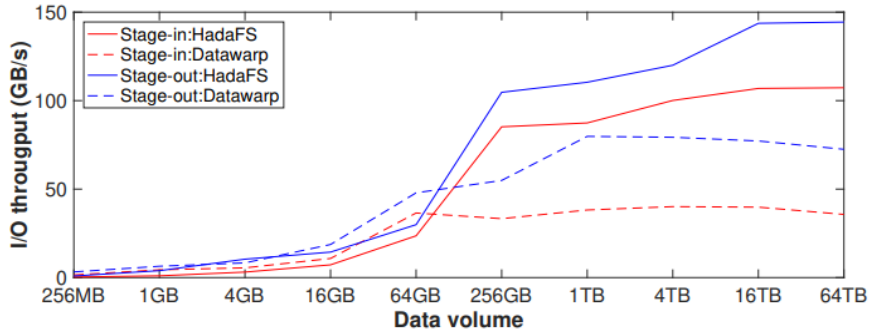


Figure 11: Data migration throughput comparison

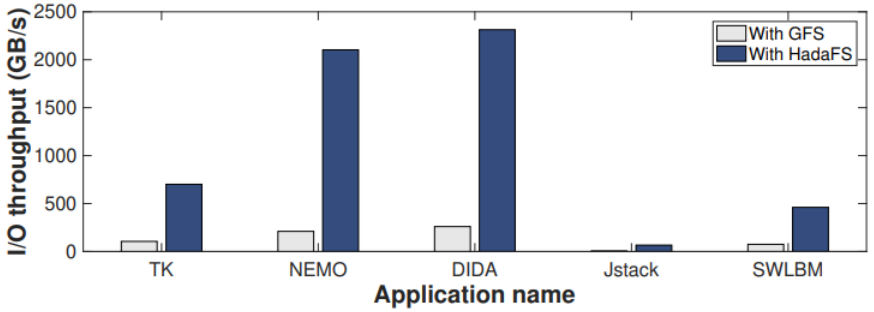


Figure 16: Performance improvement for large-scale real-world applications

Notes

* Figure: https://doi.org/10.1145/3317550.3321422

- Depending on the deployment location of SSDs, BBs can be classified into two types
  1) local BB: SSDs are deployed on each computing node as local disks;
     - Not suitable for scenarios such as N-1 I/O mode (all processes share one file) and workflow due to the difficulty of data sharing.
     - The architecture results in significant resource waste due to the large variance in I/O load between HPC applications and the relatively low percentage of data-intensive applications.
     - The deployment cost will rise sharply in the future as supercomputers scale up rapidly.
  2) shared BB: SSDs are deployed on dedicated nodes that can be accessed by computing nodes, such as I/O forwarding nodes to support shared data access.
     - The advantage of data sharing and deployment costs compared to the local BB.
     - But it is challenging to support ultra-scale supercomputers with hundreds of thousands of clients. ➔ e.g. LPCC is inefficient for data sharing and metadata-intensive access because data stored on the Lustre clients' SSDs must be flushed to the Lustre server before being shared.
- Traditional file system v.s. Kernel bypass I/O