FBench

A Flexible Benchmark for CFG-Based What-If Exploration of HPC I/O Patterns

Zhaobin Zhu[1], **Chen Wang**[2], **Kathryn Mohror**[3], **and Sarah Neuwirth**[1]

[1]Johannes Gutenberg University Mainz, Germany, {zhu,neuwirth}@uni-mainz.de · [2]Nanyang Technological University, Singapore · [3]Lawrence Livermore National Laboratory, USA

1 Motivation

In the exascale era, I/O can become a limiting bottleneck. Performance depends on multiple factors such as the *access pattern*, *middleware*, and *file system*.

- **Profilers:** Identify bottlenecks but provide no controlled evaluation.
- **Benchmarks:** Either too specific (FLASH-IO, HACC-IO) or too generic (IOR).
- **Trace replayers:** Reproduce behavior but cannot explore modified access patterns.

⇒ **Missing:** a systematic framework for what-if analysis and optimization.

2 The FBench Workflow

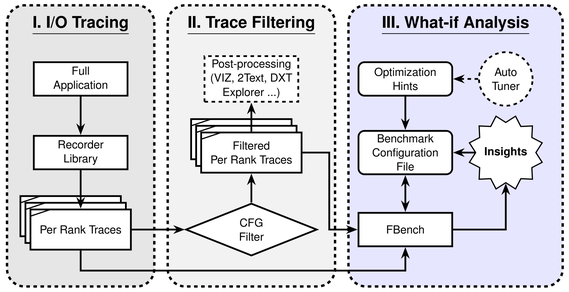


Fig. 1: Three phases: I/O tracing, trace filtering, and what-if analysis.

- **Tracing**: Recorder intercepts all I/O calls across the relevant layers and emits a **per-rank CFG**, preserving the structure between operations at ∼3 % overhead.
- **Filtering** (optional): grouping or removing symbols shrinks the CFG; the result stays valid for Recorder-Viz, Drishti, and DXT-Explorer.
- **What-if analysis**: replay *on-the-fly* from the traces or from a generated *global configuration*, and feed optimization hints back in.

3 On-the-Fly Replay

- Each rank reads only its own CFG and turns every recorded event directly into an I/O operation, using the original size, offset, and flags: $\mathcal{O}(R \cdot N)$ with **zero preprocessing**.
- Recorder encodes MPI file handles as **globally unique integers**, so collective calls match across ranks without sub-communicators.

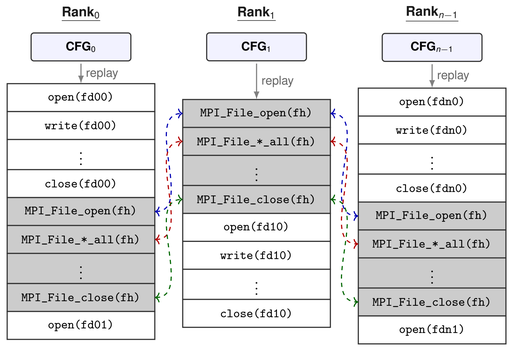


Fig. 2: Global synchronization for on-the-fly replay using MPI-IO collective operations.

- Seeks, pointer updates, and offset-based accesses are replayed as recorded; optional *inter-event delays* preserve the original timeline.

⇒ Full-fidelity replay of the recorded timeline, without preprocessing.

4 Global View: From Traces to a Tunable Configuration

Four merge steps yield a compact, repetition-based JSON configuration: group by transfer size, operation type, file, and rank; merge within a session; group by access type; merge by session count.

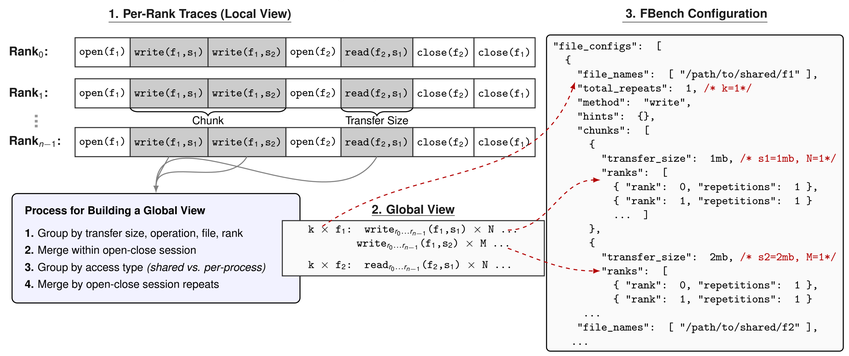


Fig. 3: Transformation of per-rank traces into an FBench configuration and the hint slot the tunables below are injected into.

- Transferred volume $D = R_{\text{total}} \sum_c \sum_{r \in \text{ranks}(c)} n_{r,c}\, s_c$: $n_{r,c}$ transfers of size s_c, over R_{total} open-close repeats. Preprocessing $\mathcal{O}(R \cdot N + S)$, execution $\mathcal{O}(S)$.
- The global view discards temporal structure by construction, which suffices for *aggregate per-file throughput* in steady checkpoint or dump phases.

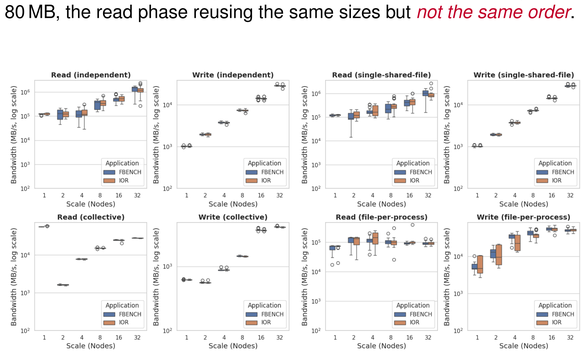
Tunables are injected into that same configuration through a human-readable JSON file, so a what-if question needs *no code change and no re-tracing*:

- **POSIX**: `use_direct` and `disable_readahead`, plus the Lustre stripe size and count.
- **MPI-IO**: `cb_buffer_size`, `cb_nodes`, `romio_*`, `striping_*`.

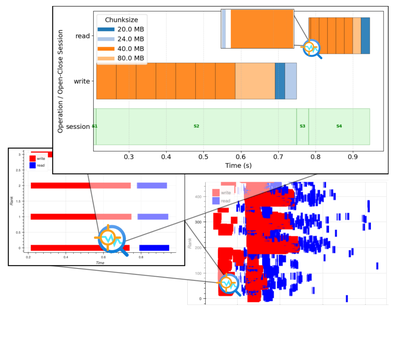
⇒ One configuration artifact replaces **rebuilding** and **re-running** the application.

5 Validation on Synthetic Workloads

IOR with up to **1,024 tasks on 32 nodes** of Corona (LLNL), POSIX and MPI-IO, file-per-process (FPP) and single-shared-file (SSF), collective and independent and **HACC-IO** on **480 ranks** (10 nodes), file-per-process, writing blocks of 20, 24, 40, and 80 MB, the read phase reusing the same sizes but *not the same order*.



(a) IOR, MPI-IO



(b) IOR, POSIX

(c) HACC-IO access pattern

Fig. 4: (a, b) FBench and IOR follow nearly identical scaling trends; (c) chunk sequence, open-close sessions, and the 480-rank view of HACC-IO.

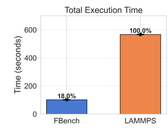
- Deviations stay between **0.6 % and 6.9 %**: they are largest in caching-dominated reads and smallest in collective and FPP writes.
- OTF is within ∼5 % on three of four configurations; MPI-IO shared-independent is hardest, at up to 15 %.
- An irregular pattern like HACC-IO cannot be expressed easily with IOR, yet FBench reproduces it from the *simplified configuration file* alone.
- Over Lustre striping, reads are page-cache dominated and the dip is tied to the **stripe count**, not the stripe size.

⇒ Synthetic scaling reproduced within **6.9%**, irregular patterns without a hand-written kernel.

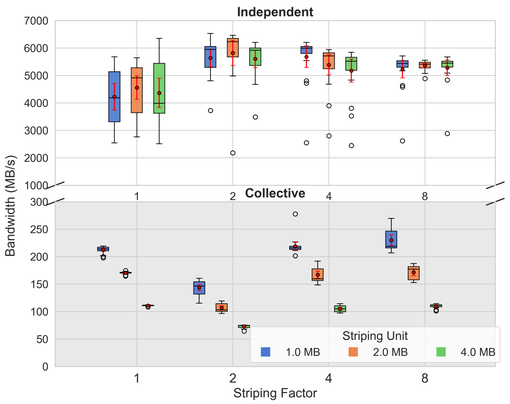
6 What-If Analysis on Real Applications

FLASH Sedov (20 nodes × 40 cores, parallel HDF5 over MPI-IO) and **LAMMPS** *melt* (864 M atoms, 124 GB shared MPI-IO dump), replayed on Lustre and UnifyFS.

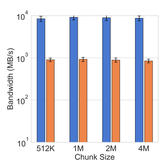
App	What-if question	Result
FLASH	collective vs. independent	0.18 vs. 5.2 GB/s ⇒ **30**×
FLASH	Lustre → UnifyFS, untuned	∼8 GB/s ⇒ **1.5**×
LAMMPS	replay instead of full run	101.8 vs. 566.5 s ⇒ **18 %**
LAMMPS	output time, Lustre → UnifyFS	66 → 8.5 s ⇒ **8**×
LAMMPS	UnifyFS chunk @ 256 MB shmem	1 vs. 4 MB ⇒ **1.25**×



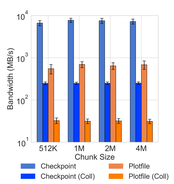
(a) LAMMPS runtime



(b) FLASH: Lustre striping



(c) UnifyFS: FBench



(d) UnifyFS: FLASH

Fig. 5: (a) FBench replays 300 timesteps in 18 % of the LAMMPS runtime; (b) independent I/O beats collective I/O on Lustre (grey background); (c, d) on UnifyFS, FBench matches the original FLASH run.

- FLASH: the *access mode*, not the striping configuration, dominates; collective buffering offers no headroom (340-400 MB/s).
- LAMMPS: the aggregator count is coupled to the stripe count, capping aggregation at **8 processes** on 10 nodes, so a mechanism meant to improve I/O becomes a bottleneck; **doubling the aggregators** alone cuts ∼20 s off the output time.
- With inter-I/O delays enabled, on-the-fly replay reproduces all five bursts and matches every inter-burst gap within **1-4 %**.

⇒ Two applications, one artifact, and a **30×** gap no striping can close.

Key Findings

- **CFG-based I/O pattern modeling** captures complex large-scale access behavior.
- **Two replay paths**: full-fidelity on-the-fly replay without preprocessing, and a global configuration for the aggregate view.
- **Application-agnostic what-if analysis**: a 30× collective/independent gap, a 1.5× untuned gain from UnifyFS, and up to 8× from simple tuning, with no code changes.

References

[1] C. Wang et al., *Recorder: A Comprehensive Multi-Level I/O Tracing Framework*, 2025.
[2] J. Bez et al., *Drishti: Guiding End-Users in the I/O Optimization Journey*, PDSW 2022.
[3] B. Behzad et al., *Improving Parallel I/O Autotuning with Performance Modeling*, HPDC 2014.
[4] J. Logan et al., *Skel: Generative Software for Producing Skeletal I/O Applications*, e-Science 2011.
[5] M. J. Brim et al., *UnifyFS: A User-level Shared File System*, IPDPS 2023.
[6] Z. Zhu et al., *Grammar-Based Workload Synthesis for What-If Exploration of HPC I/O*, HiPC 2026 (*accepted for publication*).

Gefördert durch: Bundesministerium für Forschung, Technologie und Raumfahrt · NHR Nationales Hochleistungs Rechnen

Acknowledgment: *The authors gratefully acknowledge funding from the Federal Ministry of Research, Technology and Space (BMFTR) and the state governments participating in the NHR (www.nhr-verein.de/unsere-partner). The authors gratefully acknowledge the computing time provided on the HPC systems at Lawrence Livermore National Laboratory. This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344 (LLNL-POST-2024820).*

Further information? Visit https://www.hpca-group.de/research or scan the QR code