CARAT: Client-Side Adaptive RPC and Cache Co-Tuning for Parallel File Systems

Md Hasanur Rashid[1], Nathan R. Tallent[2], Forrest Sheng Bao[3], Dong Dai[1]

[1]University of Delaware, [2]Pacific Northwest National Laboratory, [3]Iowa State University

DIRLAB

HPC apps are increasingly data-intensive

- I/O performance directly gates application throughput

Proper PFS tuning yields large gains

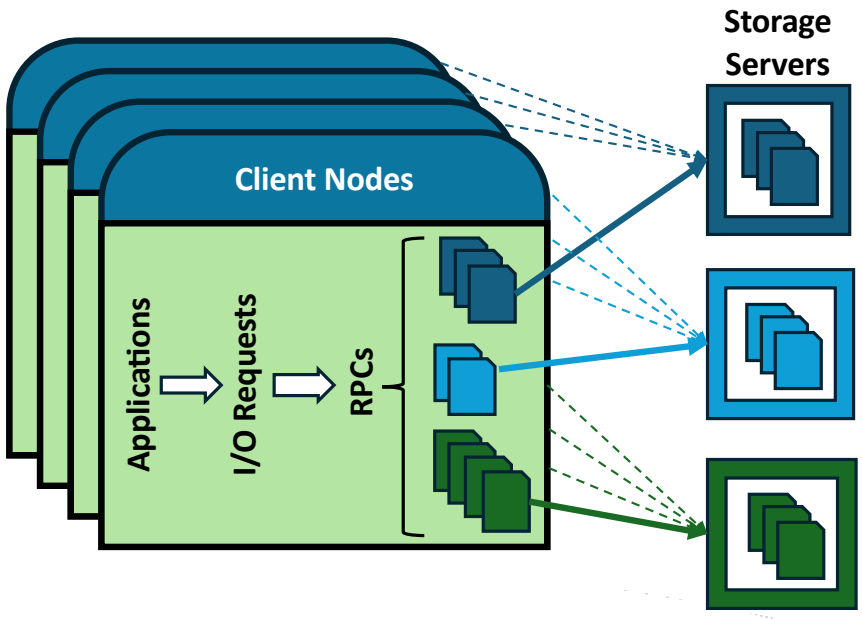
- **6.8×** speedup on OpenPMD · **7.9×** on FLASH-IO *[Bez et al., PDSW '21]*

Static fails → need adaptive

- Workloads shift across phases
- Clients on different nodes see different conditions
- Server contention varies with time

Global is heavy → need client-side & local.

- Cluster-wide instrumentation
- Pattern detection is noisy
- One config applied uniformly to all clients



Different clients → different I/O patterns → different optimal configs

Can each **client** tune itself **online** — using only **local** observations?

DIRLAB

Lustre client I/O path

Writes → dirty-page cache → fixed-size **RPC extents**

Parameter	Role
max_pages_per_rpc	RPC size cap
max_rpcs_in_flight	Concurrency cap
max_dirty_mb	Dirty cache budget

Three coupled failure modes:

❌ **Under-filled extents**

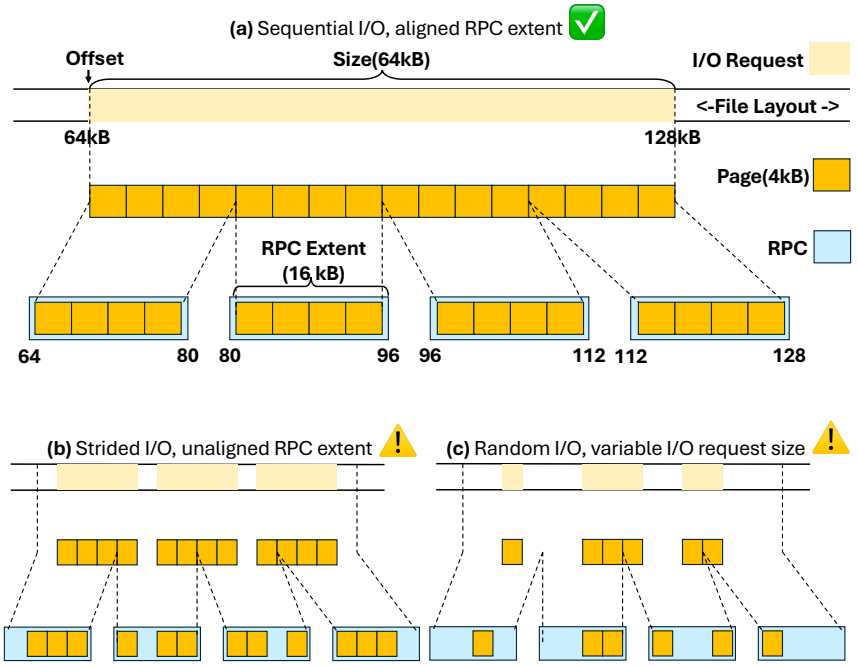
↳ delayed dispatch, cache fragmentation

❌ **Server-side congestion**

↳ queueing delays from RPC bursts

❌ **Cache limit too low / high**

↳ premature flush *or* flush storms



RPC size, concurrency, and cache budget are coupled — they must be tuned jointly

Five system-level challenges

[C1] Coupled control knobs

↳ RPC + cache interact; tuning independently is counterproductive

[C2] Different adaptation timescales

↳ RPC effects appear quickly; cache effects emerge gradually

[C3] Dynamic workload phases

↳ I/O behavior changes over time → static settings become suboptimal

[C4] Client heterogeneity

↳ Different clients on the same application see different conditions

[C5] Shared-system interference

↳ Contention and concurrent applications shift the best setting

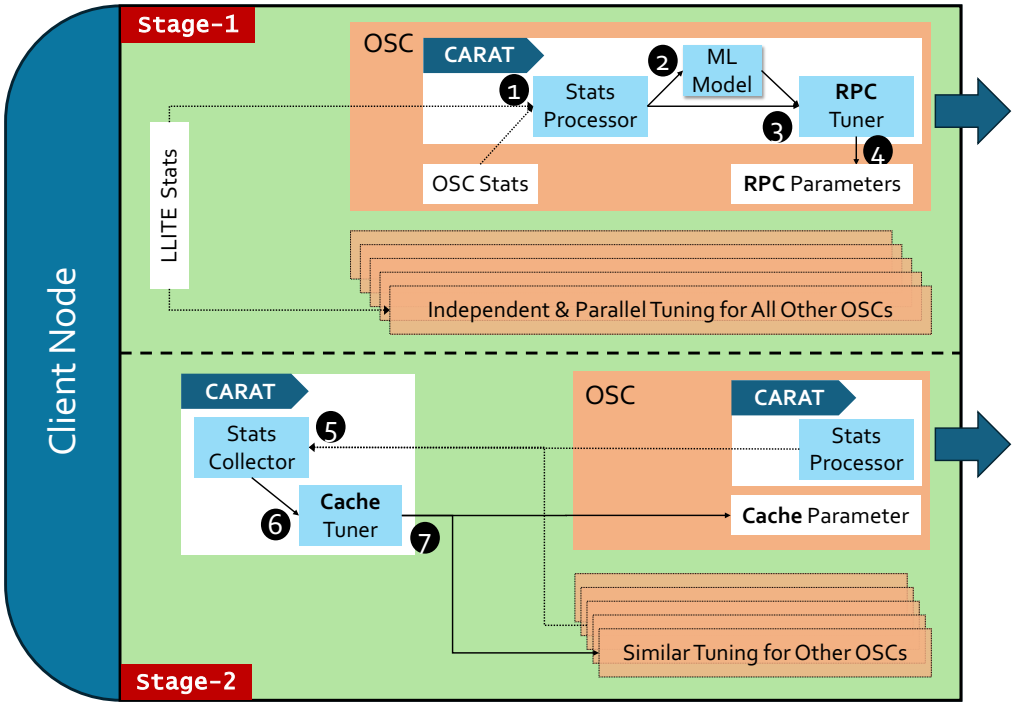
Each challenge demands a client-side, adaptive, local-only response

DIRLAB

Our approach: **client-local observability + ML, per-client, online**

Three contributions *(addressing C1-C5):*

1. **Client-local metric design** *[C3,C4,C5]*
   Local signals implicitly encode global state
2. **Two-stage RPC + cache co-tuning** *[C1,C2]*
   Separate cadences for the coupled knobs
3. **Lightweight ML-driven decisions** *[C3,C5]*
   Per-client, no central coordination



Client-side · Adaptive · Local-only

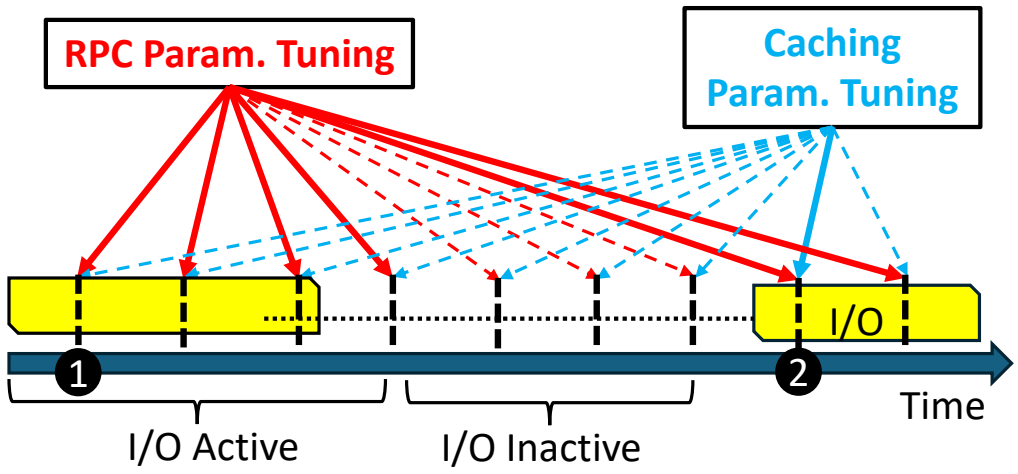
Two-stage tuning *(addressing the timescale challenge)*

Stage 1 — RPC parameters

- Tuned every interval (0.5 s in our experiments)
- Effects appear immediately
- Active only during **I/O-active** phases

Stage 2 — Cache parameter

- Tuned only at the **I/O active boundary** (after inactive period)
- Effects emerge gradually
- Avoids confounding with RPC changes



Six client-local metrics — collected with negligible overhead

Metric	Captures locally	Reflects globally
RPC Page Utilization	Extent fill quality	—
RPC Channel Utilization	Concurrency vs. cap	Server grants, contention
Unit Page RPC Latency	—	Network + server load
Data Transfer Volume	Throughput signal	Total RPC traffic
Dirty Cache Utilization	Buffer pressure	Server drain rate
Estimated Cache Update	In-place updates	—

Plus: short-term deltas between intervals → catches phase transitions

Local metrics implicitly encode global system state — no cluster-wide instrumentation

Classification, not regression

"Will this configuration improve performance by >15%?"

Two models: one for read, one for write

- Read and write exercise **different Lustre paths**
  - reads stress request parallelism
  - writes stress dirty-page cache + writeback + lock pressure
- At runtime: select model by **dominant I/O volume** in the interval

Model comparison

(error rate, lower is better)

Model	Read	Write
SVM	0.23	0.25
FC-NN	0.11	0.21
RNN	0.17	0.23
TCN	0.11	0.28
GBDT ✅	**0.04**	**0.12**

Conditional Score Greedy policy

- **Filter**: keep configs with predicted probability > $\tau = 0.8$
- **Rank**: by score combining probability + preference for larger values (stability)
- Avoids both over-conservative ("safe but small gain") and unstable random exploration

GBDT + filtered greedy → high-confidence, stable updates only

Cluster (CloudLab c6525-25g)

- 5 client nodes + 4 OSS (2 OSTs each) + 1 MGS/MDS
- 16-core EPYC, 128 GB RAM, 25 Gb NIC
- Lustre 2.15.5

Baselines

- **Default:** Lustre stock configuration
- **Optimal:** best config per workload (offline search)

Training data — Filebench Patterns Only

Naming: `[stream]_[op]_[access]_[size]` →
24 patterns
(stream: s/f, op: rd/wr, access: sq/rn, size: 8KB / 1MB / 16MB)

Training process

- Repeated pattern execution with **random parameter exploration** for sample collection
- Split 80 / 20 train / validation, hyperparameter search per model

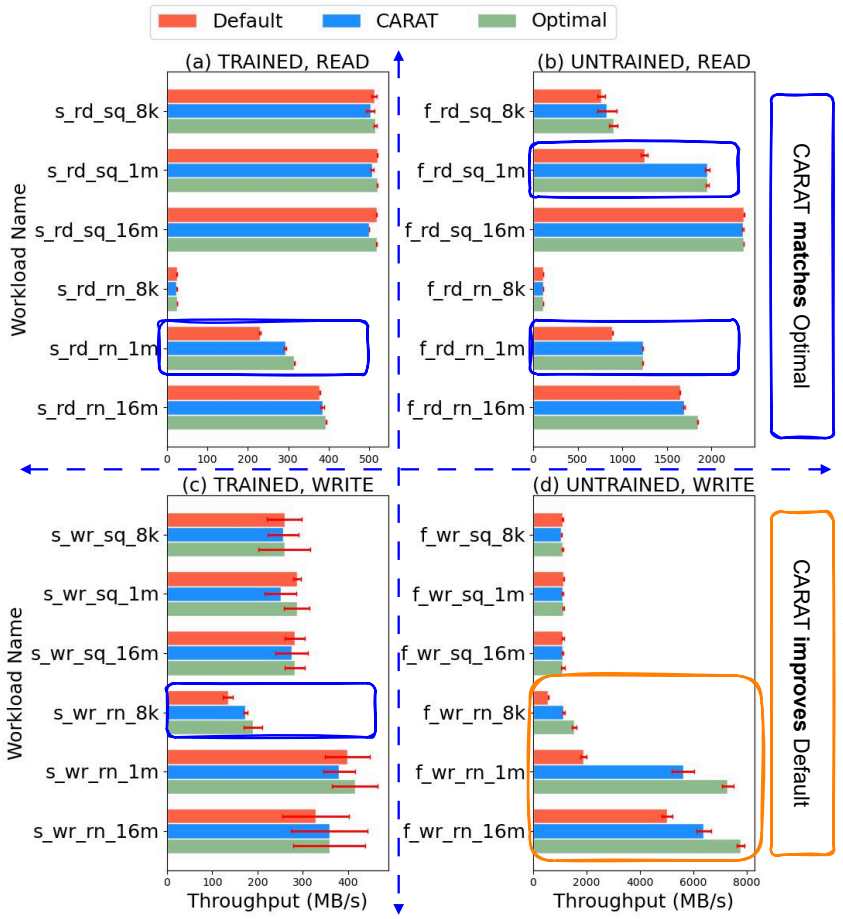
Dataset size

100,730 read-only samples · **98,078** write-only samples

DIRLAB

24 Filebench workloads × {default, CARAT, optimal}

Across all workloads, CARAT either matches default within 10% or significantly improves it — **up to 3× over default**

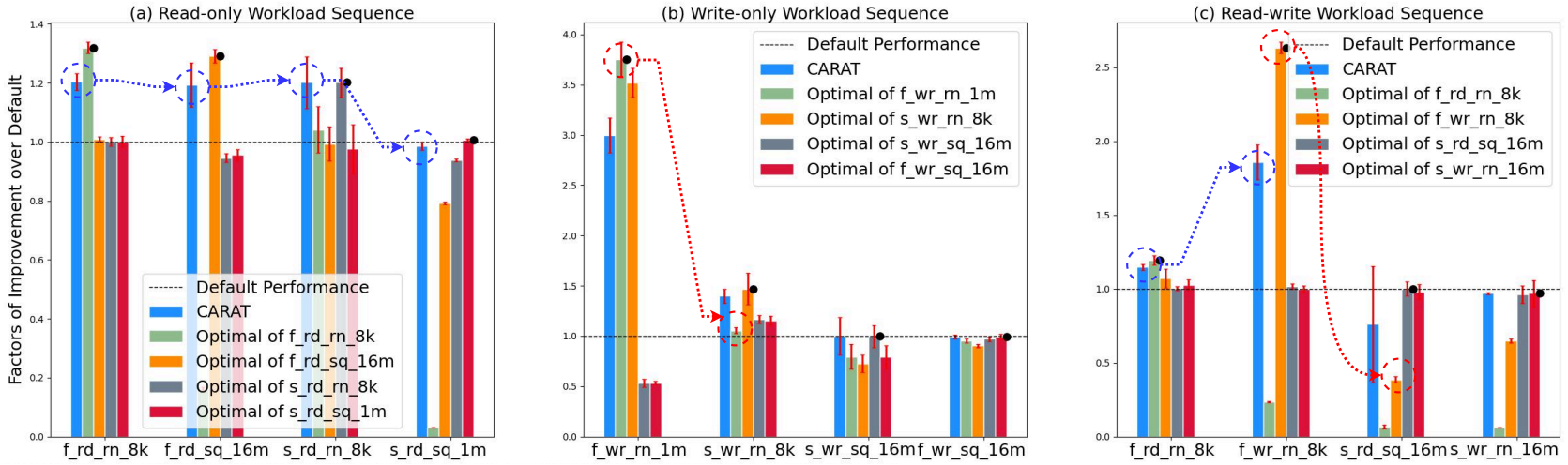


DIRLAB

IMPROVES to Near-optimal Performance

STATIC Optimal Results in **Sub-optimal** Performances

ADAPTIVE to **Dynamic Changes** in Workload



Adapts to phase shifts

Multi-client interference

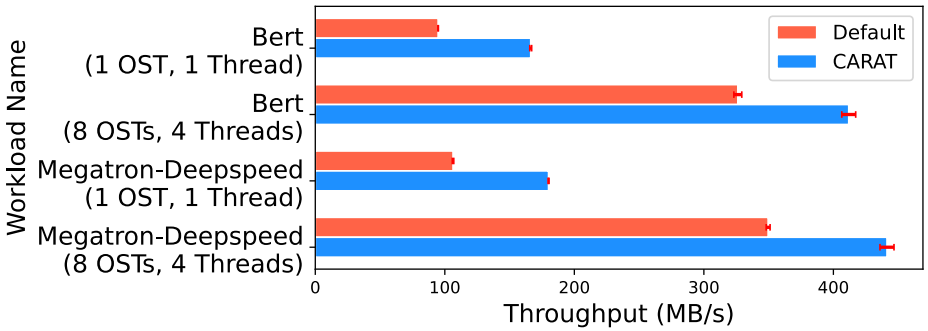
Scenario	CARAT vs Default
All read	1.15×
All write	**1.47×**
Mixed read–write	**3.0×**

Each client decides independently · local signals only · scales without coordination

Traditional HPC — H5bench

- **VPIC-IO** (3D write): 459.7 → 514.6 MB/s
- **BDCATS-IO** (3D read): 398.2 → 405.3 MB/s

Deep learning kernels — DLIO



Generalizes from synthetic patterns to real DL + HPC apps
Adaptive across workloads it was never trained on

1. Local-only metrics

implicitly encode global state — no cluster-wide instrumentation

2. Adaptive

two-stage tuning — RPC + cache, online, as workloads shift

3. Client-side

decisions scale — no coordination, generalizes to unseen apps

Up to 3× over default — across static, dynamic, multi-client, and real-world workloads

Paper **Artifact**

Md Hasanur Rashid · mrashid@udel.edu

Thank you · Questions?