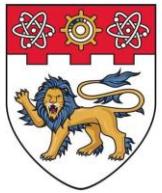
iCR
Intent-Driven Adaptable Checkpoint/Restart

Olga Kogiou, Hariharan Devarajan, Chen Wang, Weikuan Yu, Kathryn Mohror
Florida State University
Nanyang Technological University
Lawrence Livermore National Lab+







CLUSTER 2026
IEEE International Conference on Cluster Computing
September 22nd-25th, Alexandria, Virginia, United States of America

LLNL-CFPRES-2024811

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344 and Project No. 23-ERD-053.

Talk Outline

1 **Background & Motivation**

Heterogeneity, configurability, I/O intent

2 **iCR Design**

Collector, Mapper, Resolver, Applicator

3 **Implementation**

Trace-driven pipeline on DFTracer + Dask

4 **Evaluation**

Montage, Megatron-DeepSpeed

5 **Conclusion**

Contributions

Heterogeneity in workflows and data

Workflows form a Directed Acyclic Graph (DAG) of inter-dependent simulation, data analytics and AI tasks.

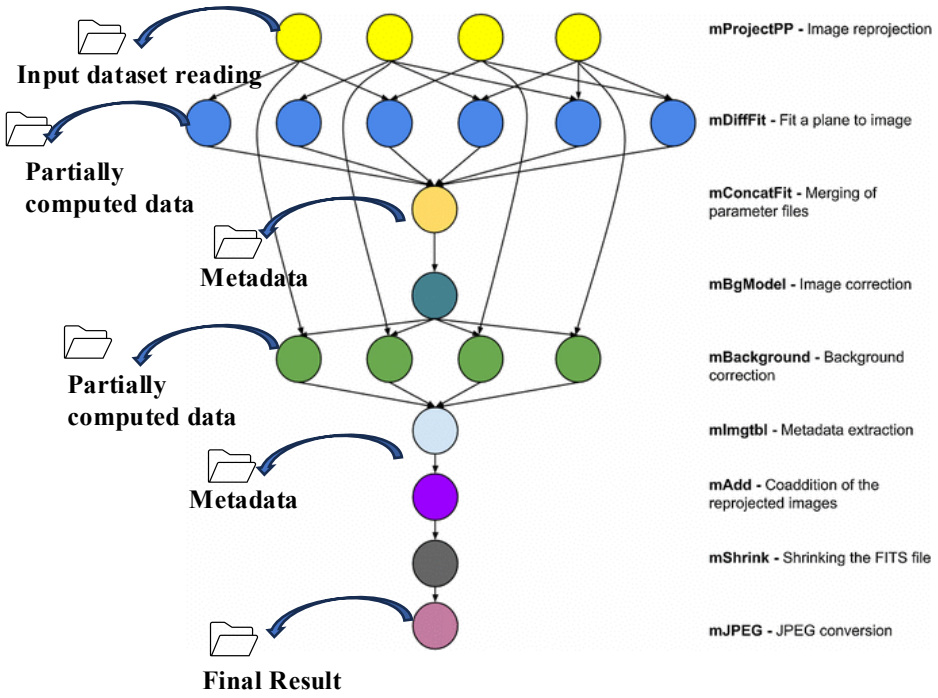
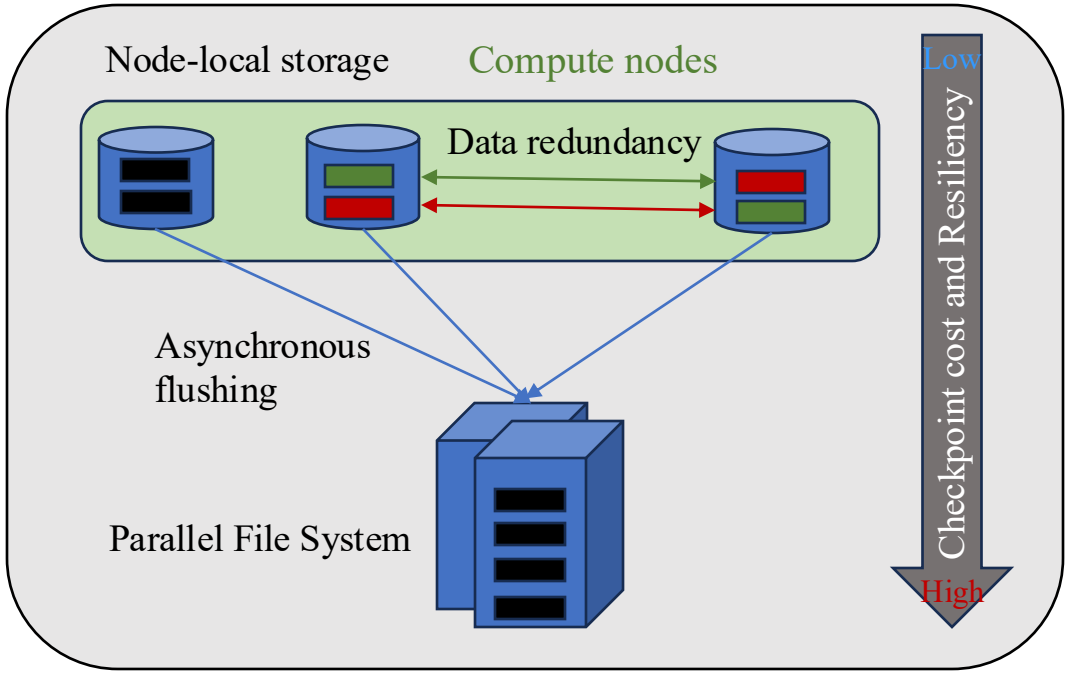


Fig. 1 — Montage: an example of a heterogenous workflow.

Heterogeneous execution times — Different Workflow stages have *different requirements* from the underlying system.

Data of unequal importance — Non persisted intermediate data can cause *rollback errors* in dependent tasks.

C/R is configurable — but used statically



~30 tunable parameters in SCR[1] / VeloC[2] /ADIOS[3]

- Checkpoint interval
- Cache base (DRAM, NVMe, shared memory, burst buffer)
- Persistent storage
- Cache size
- Redundancy scheme (SINGLE, PARTNER, XOR, R-S)
- Flush API (POSIX threads, Cray DataWarp, IBM BB)
- Failure set size

Even flexible systems such as ADIOS require manual adaptation per I/O phase.

[1] https://github.com/llnl/scr
[2] https://github.com/ECP-VeloC/VELOC
[3] https://github.com/ornladios/ADIOS2

Motivating case study: Montage on 64 nodes

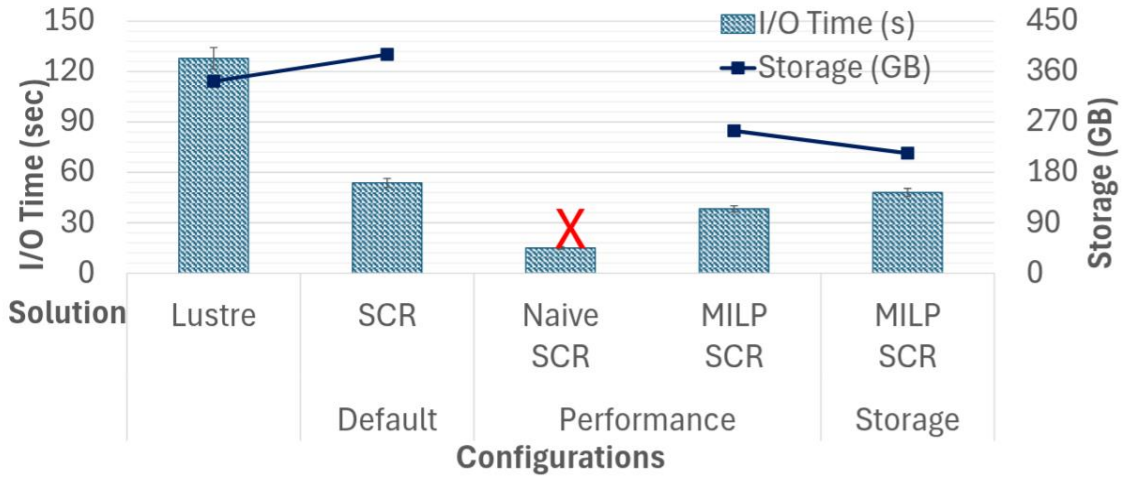


Fig. 2 — Static VS Manual SCR performance on Montage (Corona, 64 nodes, 32 ppn).

Table I— SCR configs on Montage.

Conf.	Cache	Redundancy Scheme	Flush Interval	Persist
SCR Default	NVMe	XOR	1	Lustre
Naive SCR	RAM	SINGLE	1000	Lustre

Optimization Problem!

- MILP performance: 1.62× faster, but 1.18× more storage.
- MILP storage: 1.83× smaller footprint using R-S instead of PARTNER above 1 MB.

Why not simply run an exact optimizer?

The "ideal" baseline we built

- Collected I/O traces: 30+ Pandas-like queries and 32 minutes of analysis to recover the data behaviors.
- A PuLP MILP model: maps those behaviors with SCR configurations in 2.7–3.3 minutes.

>35 min

For the entire optimization pipeline.

37 K

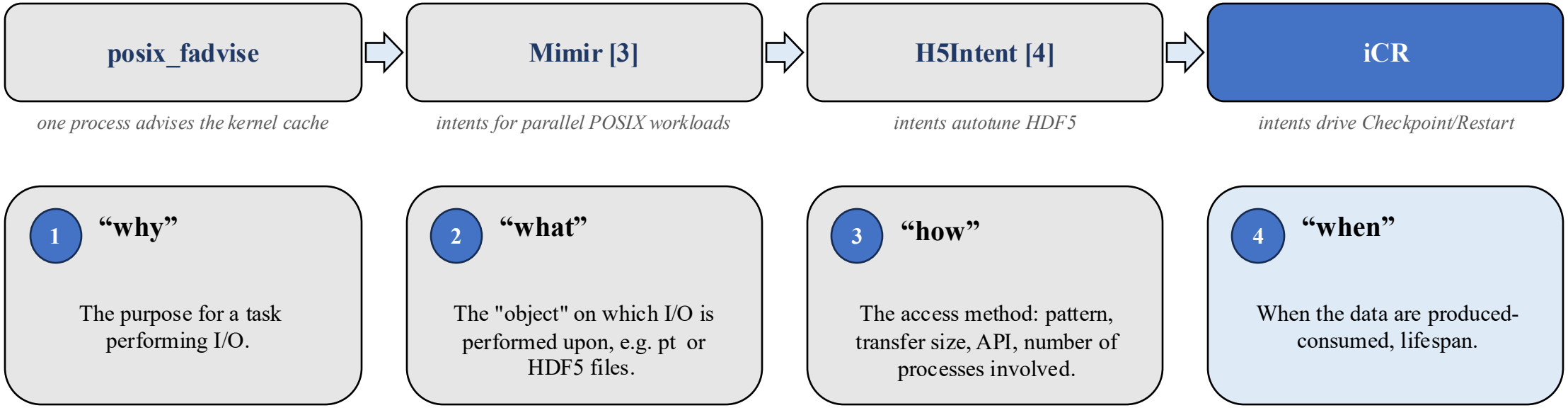
Montage

hours

Expected convergence for M-file workflows
i.e. 1000 Genome – 21 M files

This motivates a principled semantic abstraction that reduces both manual tuning and exact optimization complexity.

User I/O intents: related work



Prior work covers (1)–(3). iCR adds (4).

[3] H. Devarajan and K. Mohror, "Mimir: Extending i/o interfaces to express user intent for complex workloads in hpc," in 2023 IEEE International Parallel and Distributed Processing Symposium (IPDPS), pp. 178–188, IEEE, 2023.
[4] H. Devarajan, G. Heber, and K. Mohror, "H5intent: Autotuning hdf5 with user intent," IEEE Transactions on Parallel and Distributed Sys tems, 2024.

iCR: an end-to-end, semi-automated pipeline

Design philosophy

- Flexibility — operates atop any C/R library, workflow and system.
- Extensibility — extends Mimir/H5Intent intents with C/R semantics.
- Automation— removes manual tuning and user intervention.

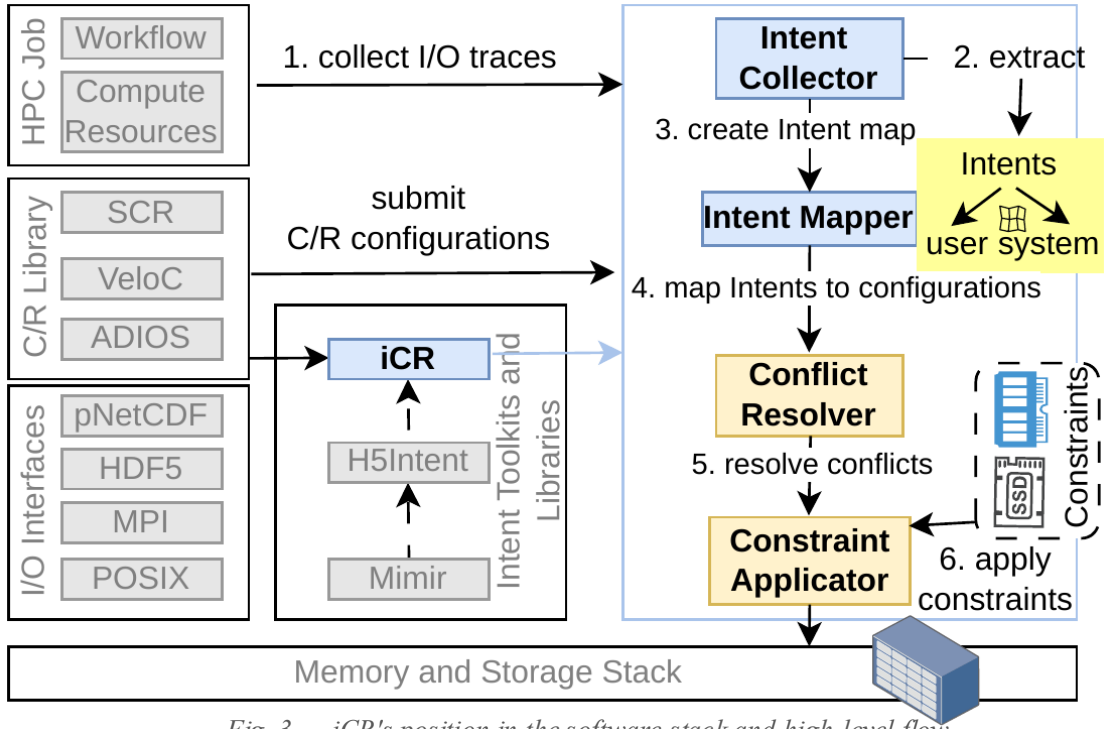
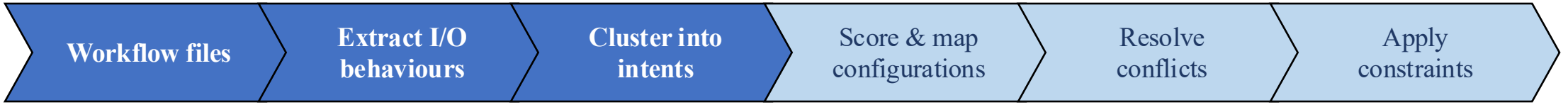
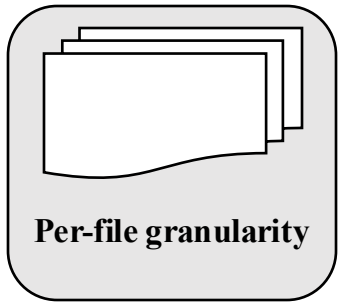
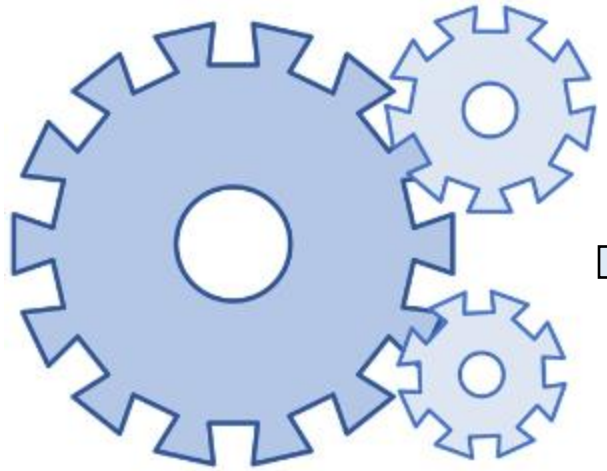


Fig. 3 — iCR's position in the software stack and high-level flow.

Intent Collector: grouping into logical intents

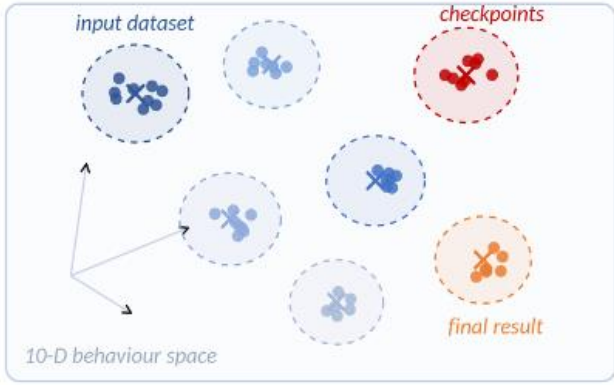






Parallel Query Engine

10 system-centric behaviors
Data type, Lifespan, Access pattern, Transfer size etc.



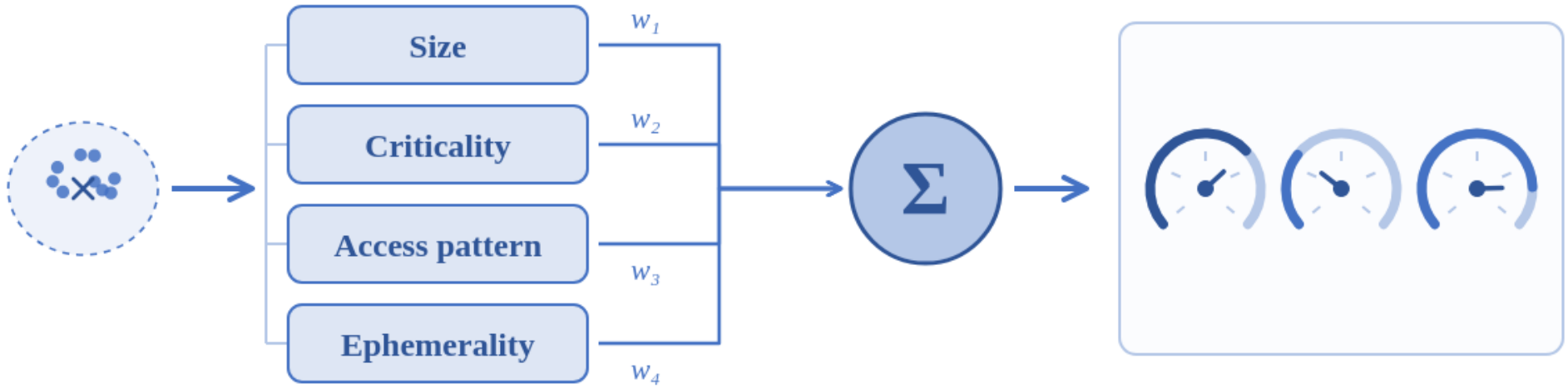
k-means clustering

Input dataset, Partially computed data, metadata etc.

Dimensionality Reduction!!

Intent Mapper: intents to C/R configurations

Workflow files | Extract I/O behaviours | Cluster into intents | **Score & map configurations** | Resolve conflicts | Apply constraints



One intent
not per file

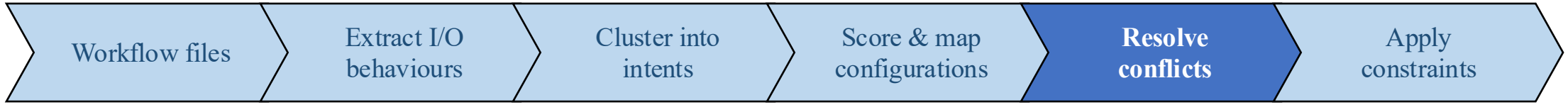
Four descriptors
four numbers per intent

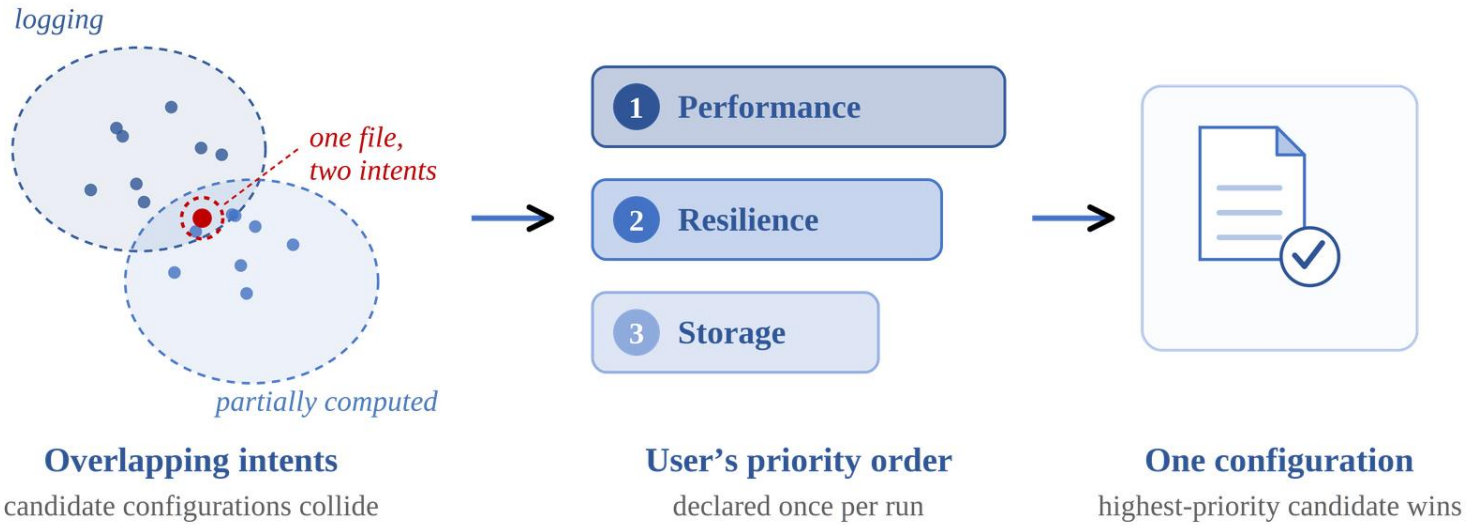
Weighted sum
one score per candidate

Adapted configuration
16 C/R knobs

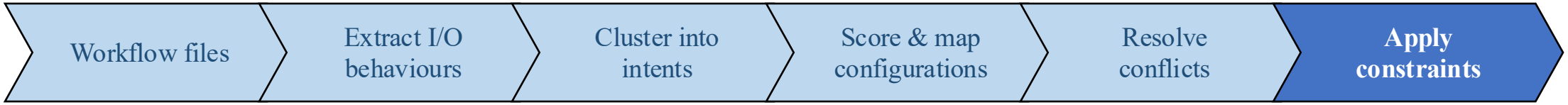
Complexity Reduction!!

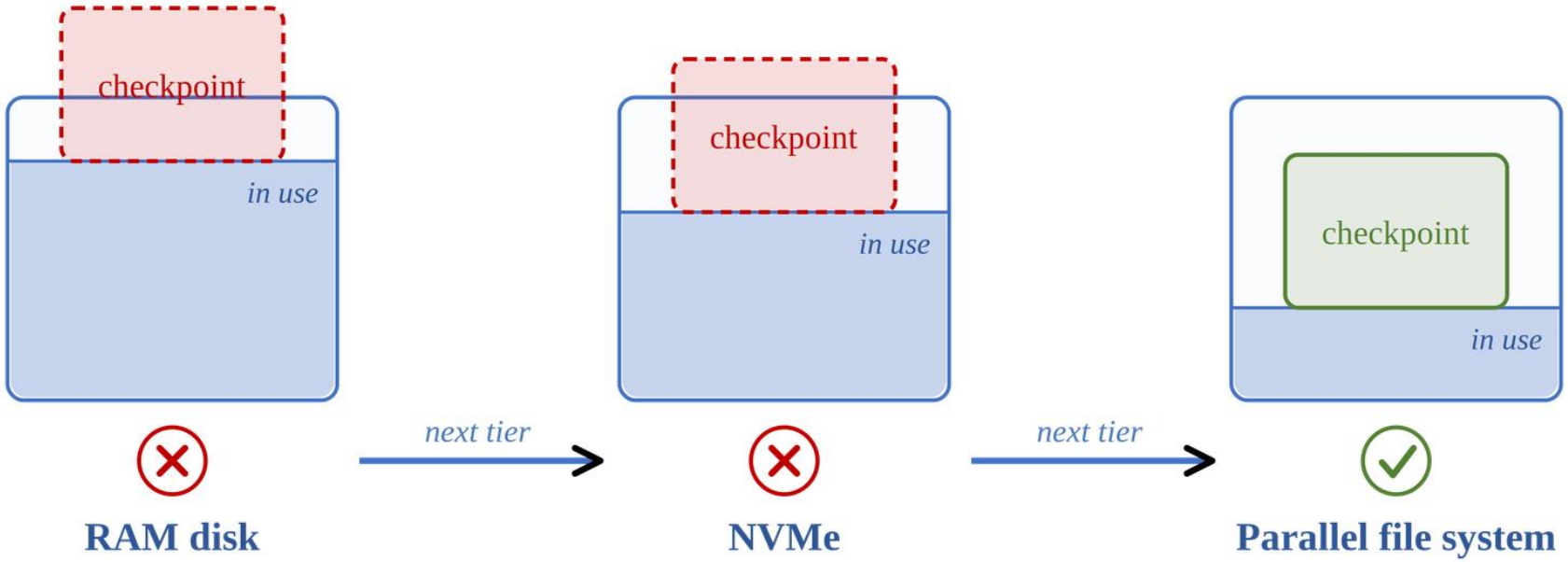
Conflict Resolver: non-exclusive intent assignment





Constraint Applicator: apply physical constraints





Implementation: the trace-driven pipeline

1	**DFTracer [5]**	Workflow I/O traces.
2	**Parallel query engine Dask**	Distributed cluster lazily loads traces into a distributed DataFrame.
3	**K-means**	Scatter-gather: workers synchronize on centroid distances In memory shared hash-map of user I/O intents

Evaluation methodology

Corona cluster (LLNL)

- 121 compute nodes
- AMD Rome, 48 CPU cores/node
- 8 AMD MI50 GPUs/node
- 256 GB memory @ 153 GiB/s
- 1.5 TB local NVMe
- InfiniBand HDR
- Lustre: 16 MDS, 36 OSS, 20 TB

- 64 nodes per workflow, five repeats

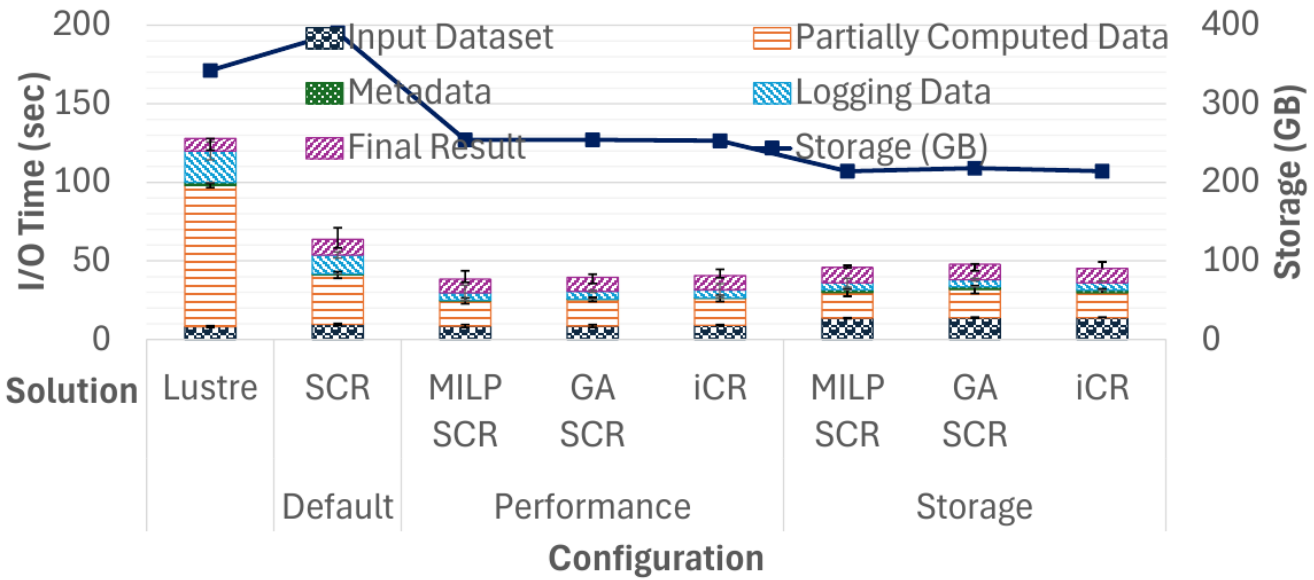
Baselines

- Lustre
- default SCR (NVMe cache, XOR, interval 1, PTHREAD)
- MILP and GA tunings of SCR ·

Workflows

- Montage — heterogeneous astronomy workflow with many intents
- Megatron-DeepSpeed — AI framework

Montage: heterogeneous workflow, 37 K files



Where the gains come from

- Partially computed data: Cache base: shared memory instead of NVMe.
- Logging/scratch: checkpoint interval to three (lifespans under 15 minutes)

Fig. 5 — I/O time by intent (bars) and storage footprint (line), 64 nodes, 100 GB 2MASS input.

1.6×
Lower I/O time vs. default SCR

1.82×
Storage saved in storage mode

6.2 min
Full pipeline — 0.08% of makespan

< 5%
Error vs. MILP/GA, 7× cheaper

Megatron-DeepSpeed: AI framework

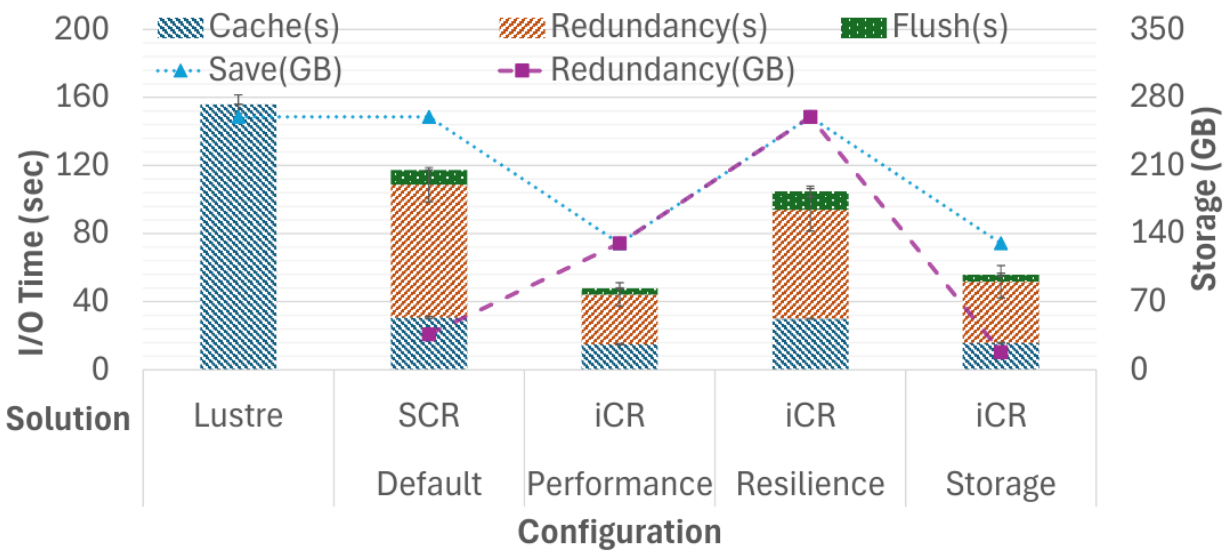


Fig. 6 — I/O time split into cache, redundancy and flush (bars); storage (lines).

2.6×

Lower I/O time — interval 2 with PARTNER instead of XOR

1.45 min

Full pipeline

Workload

- 13-billion-parameter GPT model.
- 1,000 steps, checkpoint every 200: ~130 GB per checkpoint

Cache space for redundancy data on iCR

- 18.2 GB storage mode
- 130 GB performance mode
- 260 GB resilience mode

A clear, user-selectable trade-off across the three modes.

Conclusion

1. **Workflow tasks and data are heterogenous**

Workflows tasks have different execution times. Data have different importance for workflow restart.

2. **User I/O intents can be used as a semantic abstraction**

Seven representative user I/O intents transcend workflows.

3. **Per-task or per-file granularity**

Logical user I/O intents reduce the complexity of optimization at large-scales.

2.6×

I/O time reduction vs. static C/R on Megatron-DeepSpeed.

1.82×

Storage footprint reduction vs. static C/R on Montage.

7×

Cheaper configuration search, within 5% of exact solvers.

Thank you

Questions?

github.com/castl22/iCR

ok22b@fsu.edu · hariharandev1@llnl.gov · chen.wang@ntu.edu.sg · wyu3@fsu.edu · mohror1@llnl.gov

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344 and Project No. 23-ERD-053. LLNL-CONF-2014335-DRAFT. Supported in part by the DOE Office of Science, Office of Advanced Scientific Computing Research (Early Career Research Program) and NSF awards 1763547 and 2403089.

Discussion

Other C/R libraries

iCR is backend-agnostic and integrated with VeloC and ADIOS. On a 64-node iPIC-3D run, per-checkpoint I/O time falls from 1.4 s to 0.9 s on VeloC and 1.5 s to 0.8 s on ADIOS.

Cluster assignment robustness

Are intents stable across scale? The Adjusted Rand Index between intent assignments on Wemul, from 1 M to 256 M files, averages 0.85–0.75.

Mapper weight sensitivity

Under Monte Carlo and one-at-a-time perturbations, 91 SCR mapping decisions show mean stability 0.991 and median 1.0 — no precise weight tuning needed.

Intent coverage

Beyond the three evaluated workflows: ICECap exhibits all seven intents, 1000 Genome four, MuMMI two.

Seven representative intents across six workflows

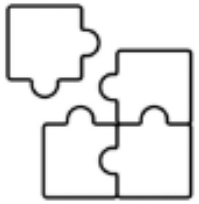
User-centric "why"	"What" — data types	"How" — access	"When" — timing
Input dataset	YAML, JSON, FITS, NC, FSTQ, INP	Many procs/nodes, random, pread/read/fread	job start < access < first computation
Partially computed	FITS, VCF, PDB, FASTQ	Many procs/nodes, sequential	write → read, in bursts
Metadata	HDR, TBL, VCF, BAM, CRAM	File-per-proc, write/read, fwrite/fgets	frequently updated, long lifespan
Scratch / logging	TXT, TMP	File-per-proc, sequential write/read	short lifespan
Anchor files	HTML, NC, VTK	File-per-proc, sequential write/read	job middle < access < job end
Checkpoints	NC, PY, PT	File-per-proc, write-only	stable intervals, in bursts, short lifespan
Final result	PNG, VTU, VTR, CSV, TAR, GZ	Many procs/nodes, distributed, write-only	at job end

Identified across Montage, MuMMI, 1000 Genome, ICECap, iPIC-3D and Megatron-DeepSpeed — clusters validated against the literature as ground truth.

How can we handle workflow, data heterogeneity and C/R configurability?

Workflow and data heterogeneity:

1. Data type i.e. FITS, HDF5
2. Data size
3. Lifespan
4. Processes that touch the data
5. Threads per process
6. Access pattern i.e. random/sequential/strided
7. Transfer size
8. System API i.e. POSIX, MPI-IO
9. I/O operations i.e. read / write
10. I/O intensity i.e. steady vs. bursty

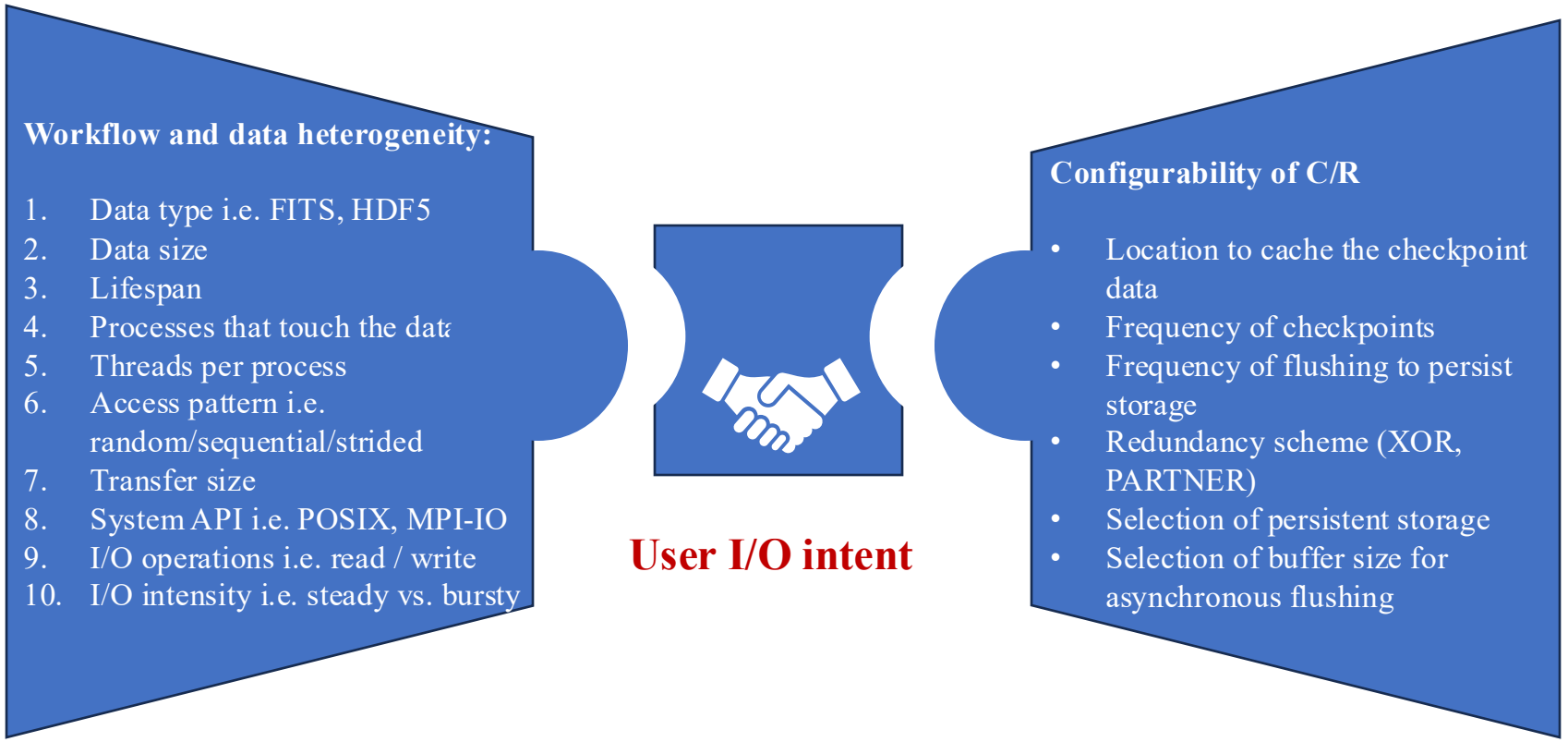


MISMATCH

Configurability of C/R

- Location to cache the checkpoint data
- Frequency of checkpoints
- Frequency of flushing to persist storage
- Redundancy scheme (XOR, PARTNER)
- Selection of persistent storage
- Selection of buffer size for asynchronous flushing

How can we handle workflow, data heterogeneity and C/R configurability?



Our approach: I/O intents as a semantic abstraction layer for C/R.

Mapper internals and C/R integration

Mapper, Resolver, Applicator

Python modules built on greedy algorithms.

Each intent is encoded as a 4-dimensional normalized vector (Size, Crit, Access, Eph) for constant-time access.

Configurations load from CSV into nested dictionaries; the Mapper computes a weighted inner product between intent and parameter weight vectors.

The Resolver memoizes with an open-addressing hash table using SipHash-2-4 — average O(1) lookup per signature.

The Applicator reads live capacity with psutil and os.cpu_count() and assigns first-fit against a mutable capacity dictionary.

Respecting data dependencies

The C/R library is integrated with DYAD for asynchronous node-to-node transfers, triggered by a CONSUMER=1 flag added to the restart API.

DYAD is part of the Flux scheduler, so it already knows where producer and consumer tasks live.

Backends supported

SCR · VeloC · ADIOS — iCR submits configurations; the workflow keeps the C/R library it is already linked against.

Assumptions and limitations

Data dependencies

iCR adds node-to-node transfers for producer/consumer data cached locally.

Task granularity

Each group of tightly coupled tasks gets its own independently configured C/R instance (e.g. via SCR_Config with a per-group config file).

Reliance on traces

A one-time cost, outweighed by savings across repeated runs.

Greedy by design

Optimality is traded for computational efficiency — bounded in the evaluation against MILP and GA solvers.

Cluster profiling

Every new cluster needs storage and interconnect bandwidth profiling — 600 test combinations over 4–128 nodes. Costly, but a one-time cost per system, reusable beyond iCR.

What the Mapper produces

User I/O intent	Representative C/R configuration
Input dataset	Fetch to cache = 1 · NVMe · cache size 256
Partially computed	Shared memory · cache size 48 · R-S redundancy
Metadata	DRAM · cache size 256 · PARTNER \ XOR \ R-S · persist to NVMe
Scratch / logging	Checkpoint interval 3 · DRAM · XOR redundancy
Anchor files	DRAM · cache size 48 · SINGLE \ PARTNER \ XOR · persist to NVMe
Checkpoints	Checkpoint interval 1 \ 2 · NVMe · PARTNER \ XOR
Final result	NVMe · R-S redundancy

Table I — Collected User I/O intents and the mapped C/R Configurations.

Reading the table

- Metadata: high criticality, small size → fast storage with redundancy.
- Scratch / logging: highly ephemeral → cheap XOR redundancy and a longer interval.

Several values score equally and the Conflict Resolver decides between them using the user's priorities.

Scalability of the Collector

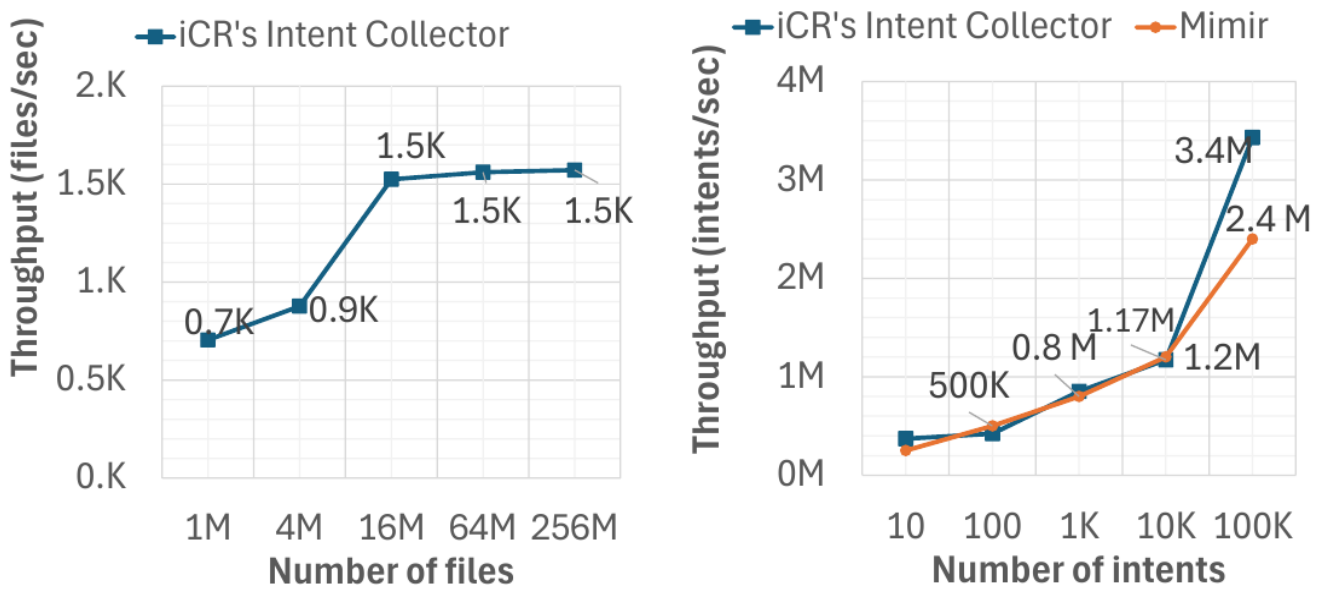


Fig. 4 — Classifying files to intents (left); storing new intents vs. Mimir (right).

1.5 K files/s

Collector saturation — near-linear up to 16 M files

3.4 M intents/s

Intent storing throughput, vs. 2.4 M for Mimir

iPIC-3D: makespan under a worst-case failure rate

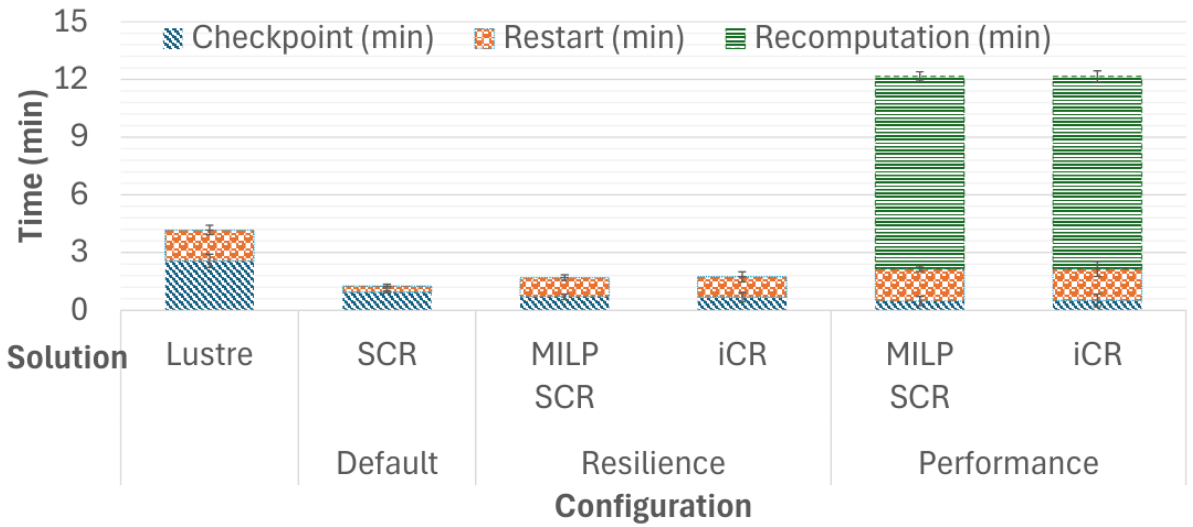


Fig. 7 — Checkpoint, restart and recomputation time under injected failures.

Optimizing for resilience preserves makespan.
Optimizing for performance costs ~10 minutes of re-computation when unflushed data are lost.

Setup

Magnetic Reconnection data, 800 cycles on 64 nodes, 70-minute makespan, checkpoint every 25 cycles.

One node fails every 200 cycles; with MTBF = 18 min the probability of a system-wide task failure is 97.95%.

Configuration	Makespan (min)
SCR default	74.2 ± 0.4
iCR resilience	73.4 ± 0.3
MILP SCR resilience	72.1 ± 0.2
iCR performance	83.3 ± 0.7
MILP SCR performance	83.2 ± 0.6

Table III — Makespan.