Context Matters: Constant-Time File Lifecycle Prediction from File Creation Call Stacks

Hocine Mahni[*]
Louis-Marie Nicolas[*]
hocine.mahni@ensta.fr
nicolas.louismarie@gmail.com
Lab-STICC, CNRS UMR 6285, ENSTA, Institut Polytechnique de Paris
Brest, France

Jalil Boukhobza
jalil.boukhobza@ensta.fr
Lab-STICC, CNRS UMR 6285, ENSTA, Institut Polytechnique de Paris
Brest, France

Abstract

Many I/O optimizations, such as data placement on hierarchical storage systems, depend on knowledge of file lifetimes and access patterns. However, such information is typically unavailable at file creation time or must be inferred through costly runtime monitoring. In this paper, we demonstrate that file creation context alone is a strong predictor of file lifetime and lifecycle behavior. We introduce CLiP, a Constant-time Lifecycle Prediction approach that enables the prediction of either a scalar file lifetime or a full file lifecycle distribution at the moment of file creation. Our method leverages the file creation function call stack hash as contextual information and performs lightweight prediction using a very small pre-trained per-application lookup table, incurring negligible runtime overhead compared to existing NN-based approaches. Moreover, in contrast to tracing-based techniques that always intercept all I/Os, once trained, CLiP only requires the instrumentation of file creation functions. Our evaluation on three representative HPC workloads (NAMD, Incompact3d and LAMMPS) shows that CLiP predicts $\sim$ 83.2% of lifetimes within ±20% of the observed values versus $\sim$ 82.0% for a pathname-based CNN baseline, while reducing prediction overhead by $\sim$ 99.7%, i.e., $\approx$ 292× faster. These results highlight that context at file creation time matters and that exploiting this context can open the door to practical and low-overhead I/O optimizations in large-scale systems.

CCS Concepts

• **Information systems** → Information storage systems; *Storage management*; **Hierarchical storage management**; **Information lifecycle management**;

Keywords

Data placement, Multi-Tier Storage, File lifetime, High Performance Computing, Simulation, File lifecycle, LD_PRELOAD Tracer

ACM Reference Format:
Hocine Mahni, Louis-Marie Nicolas, and Jalil Boukhobza. 2026. Context Matters: Constant-Time File Lifecycle Prediction from File Creation Call Stacks. In *6th Workshop on Challenges and Opportunities of Efficient and Performant Storage Systems (CHEOPS '26), April 27–30, 2026, Edinburgh, Scotland Uk*. ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3805687.3806258

[*]Both authors contributed equally to this research.

This work is licensed under a Creative Commons Attribution 4.0 International License.
CHEOPS '26, Edinburgh, Scotland Uk
© 2026 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-2604-0/2026/04
https://doi.org/10.1145/3805687.3806258

1 Introduction

The emergence of exascale supercomputers has intensified the long-standing imbalance between computational performance and storage system capabilities [6]. While compute performance continues to scale rapidly, input/output (I/O) subsystems lag behind, making data movement a dominant performance bottleneck for many high-performance computing (HPC) applications [17, 33]. Concurrent I/Os issued by thousands of nodes can saturate shared storage systems, increasing application wait times and reducing overall system throughput. These challenges are particularly pronounced for applications and workflows that generate checkpoints, intermediate results, or temporary data, such as modern machine learning and deep learning workloads, leading to bursty, diverse, and highly variable I/O behavior [4, 16, 23].

Addressing these bottlenecks requires a wide range of I/O optimizations across the software stack. File systems and storage services must decide where data should be placed [9], how it should be striped [11, 15, 19], cached [5], or evicted [1, 5, 20]. Runtimes and schedulers increasingly seek to incorporate I/O-awareness when orchestrating tasks or workflows, while I/O libraries aim to adapt access strategies such as striping to expected file behavior. Across all of these layers, a common challenge emerges: many optimization decisions critically depend on knowing how long a file will live and how it will be accessed over its lifetime.

File lifetime and lifecycle information can inform a variety of decisions. Scalar lifetime estimates are valuable for data placement and capacity planning. More detailed lifecycle descriptions—such as histograms capturing read and write activity over time—can enable scheduling optimizations (including data movement scheduling) and I/O pattern-aware tuning (including stripe size tuning). Knowledge of whether a file is short-lived or persistent, write-once or read-many, or accessed intensively only during specific phases can further guide stripe configuration, buffering strategies, and initial data placement. Although prior work has proposed tracing- and learning-based methods to characterize or predict file behavior, obtaining accurate and low-overhead guidance *at file creation time* remains challenging in practice.

Existing work addresses this need through two main classes of techniques. First, tracing and monitoring tools collect detailed

I/O events to characterize file behavior [7, 8, 18, 22, 30, 31, 37], but continuously intercepting I/O operations can introduce non-negligible overhead at scale. Second, learning-based predictors build models from past executions (thus still requiring traces for training), then provide predictions online. However, many predictors either rely on features that become informative only after observing part of the I/O stream, or incur non-trivial runtime cost due to model inference [21, 27, 34]. As a result, obtaining useful information *at file creation time* remains challenging without paying either continuous monitoring overhead or runtime inference cost.

In this paper, we argue that file creation context alone provides strong predictive power over file lifetime and lifecycle behavior. We observe that a file's creation context, captured by the call stack, encodes rich semantic information about the file's role within the application. Crucially, this context is available immediately at `create/open` call, making it suitable for proactive decisions such as initial placement or caching, before any I/O activity is observed. We argue that files created by different code paths often serve distinct purposes, such as checkpoints, intermediate scratch data, visualizations, or long-lived outputs, and these roles are reflected in their subsequent access patterns and lifetimes.

Building on this insight, we introduce CLiP, a lightweight approach that predicts either a scalar file lifetime or a full file lifecycle distribution at the moment of file creation. Our method leverages the file creation call stack as contextual input and performs inference in constant time using a compact per-application lookup table learned offline from training runs. This design incurs negligible runtime overhead and a negligible memory footprint and requires no application modifications.

We evaluate CLiP on representative HPC workloads and demonstrate that call-stack-based context enables accurate predictions of file lifetimes and lifecycle histograms across diverse applications. These predictions can be readily consumed by file systems, runtimes, schedulers, or I/O libraries to drive early decisions about data placement, eviction, scheduling, and access optimization. Our results show that context at file creation time matters, and we hope that exploiting this context enables practical, proactive, and low-overhead I/O optimizations across the HPC software stack.

2 Related Work

Research on HPC I/O has produced a wide range of approaches that enables modeling and predicting I/Os.

2.1 I/O characterization and tracing

Profiling and tracing tools are commonly used to characterize I/O behavior in large-scale HPC systems. Darshan is a representative low-overhead profiler that collects per-file statistics and enables system-wide analyses of I/O usage at production scale [6, 22, 31, 36]. Such data has supported retrospective studies of access patterns, reuse, and sharing behavior [4, 26].

While these tools are essential for understanding I/O workloads, they primarily describe behavior *after* it has occurred. Even periodicity-based techniques e.g.FTIO [32] require observing enough I/O activity to identify the period before they can predict future phases. As a result, they are not designed to support proactive decisions at file creation time, which is the focus of this work.

2.2 Predicting file lifetime and I/O behavior

Several studies explicitly target prediction of file lifetime or future I/O behavior to guide optimizations such as placement or caching. Machine-learning-based approaches have been proposed to infer file lifetimes from metadata features, including directory paths or naming conventions, under the assumption that such features encode semantic intent [21, 34]. These studies demonstrate that lifetime prediction can improve storage decisions, but they typically rely on features that are application-specific and require frequent model retraining, which could be costly.

2.3 Leveraging program context for I/O prediction

Recent work has leveraged program context—especially call stacks—as a semantic signal to predict I/O behavior. Omnisc'IO [12, 13] and GrIOt [24, 25] learn spatial and temporal patterns from observed I/O sequences (grammar-based and graph-based learning, respectively), enabling the prediction of recurring behaviors. However, these methods require intercepting I/O activity throughout execution, so the overhead grows linearly with the number of I/O operations to predict.

Our approach is complementary but distinct. Rather than modeling I/O sequences during execution, we predict file lifetime and lifecycle properties *at file creation time*, using only the creation call stack. This enables proactive, constant-time inference without continuous tracing or online learning, and supports early optimization decisions, such as initial data layout [11, 15, 19] and data placement [2, 20, 38], before the I/O behavior unfolds.

3 Contribution

CLiP predicts, at file creation/open time, either (i) a scalar file lifetime, or (ii) a coarse file lifecycle described as I/O counts over time bins. The key idea is to use the *creation context*—captured by the call stack at the `create/open` site—as a compact semantic signature of the file's role in the application.

CLiP is a **two-phase approach**:

- **Offline learning :** During training runs, a tracer (LD_PRELOAD) intercepts the application's POSIX I/O calls ①. For each created file, it extracts the creation context by linking the filename to the creation call stack ②, then derives the file behavior (lifetime and binned I/O counts) and aggregates it per context to build the lookup table (call-stack hash → behavior) ③ (Fig. 1a).
- **Online prediction :** At runtime, CLiP intercepts only `create/open` ①, computes the context identifier $c = \text{hash}(\text{callstack}_{\text{create/open}})$, then performs a constant-time *lookup* [10, 14] in the trained table to return $\widehat{L}$ or $\widehat{h}$ ②.

3.1 Offline learning

3.1.1 Creation context extraction. CLiP is trained *offline* from a set of training runs. During these runs, a tracer (via `LD_PRELOAD`) intercepts POSIX I/O calls. For each file f, it records the **creation context** as the call stack observed at the first `open()` (or `create`) and maps it to a context identifier:

$$c(f) \;=\; \text{hash}\big(\text{callstack}_{\text{create/open}}(f)\big).$$

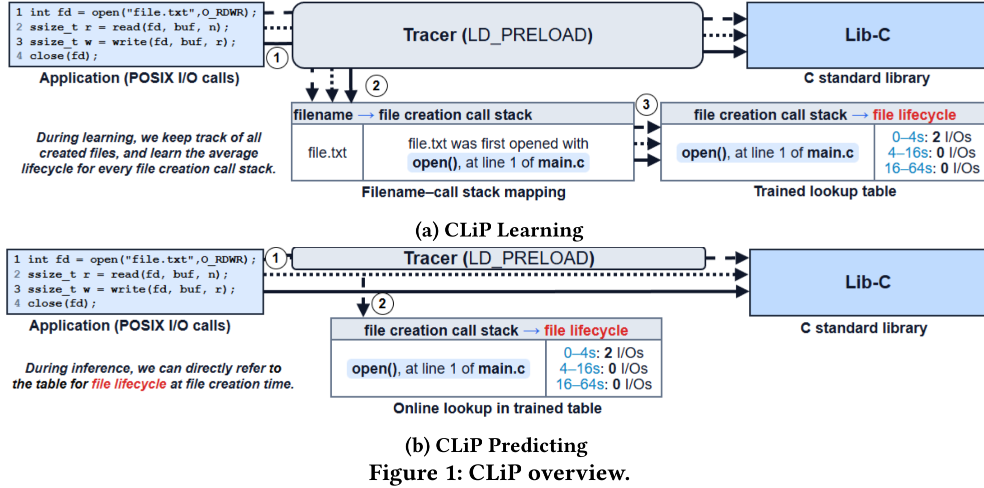

Figure 1: CLiP overview.

Figure 1a (②) shows how each filename is linked to its creation call stack during offline learning.

3.1.2 Behavior extraction. During offline learning, the tracer observes the I/O sequence after the first `create/open` of each file f (`read/write/close`) and computes the target behavior (lifetime and/or lifecycle histogram). These values are aggregated per creation context and stored in a *lookup table* (③ in Fig. 1a), implemented as a hash map that associates the context identifier c = $\text{hash}(\text{callstack}_{\text{create/open}})$ to its representative behavior (median lifetime or mean histogram).

Lifetime. We define the observed lifetime $L(f)$ as the time elapsed between the first `create/open` event of f and the end of its activity window (i.e., the last access attributed to f by the trace processing pipeline).

Lifecycle histogram. Given time bins $B = \{b_1, \ldots, b_k\}$ after the first open, we define:

$$h(f) \;=\; \text{io_histogram}(f, B), h_b(f) \;=\; \#\text{I/O events in bin } b.$$

Computing $h(f)$ requires observing I/O events *during offline training* to build the lookup tables. In the online phase, CLiP does not need to trace these I/Os, it only hashes the `create/open` call stack and performs a table lookup to return $\widehat{L}$ or $\widehat{h}$.

3.1.3 Per-application lookup tables. The training phase builds one lookup table per application. Each entry corresponds to one creation context c and stores a representative behavior derived from the training runs. We build two tables depending on the prediction task: a scalar lifetime table and a lifecycle histogram table.

3.1.4 Scalar lifetime table. Algorithm 1 builds a table T_L that maps a context c to a lifetime estimate $\widehat{L}(c)$. For each training file, we insert its observed lifetime into the multiset associated with its context. After processing the training runs, we compute $\widehat{L}(c)$ as the median of the multiset for c. We also compute a global fallback $\widehat{L}_0$ as the median lifetime over all training files.

3.1.5 Lifecycle histogram table. Algorithm 2 builds a table T_H that maps a context c to a histogram estimate $\widehat{h}(c)$. For each training file, we compute its histogram $h(f)$ over bins B and append it to the list associated with c. After processing the training runs, we compute $\widehat{h}(c)$ as an element-wise mean across the histograms stored for c. We also compute a global fallback $\widehat{h}_0$ as the element-wise mean across all training histograms. Once trained offline, CLiP performs online prediction at create/open with a single lookup (Fig. 1b).

Algorithm 1: Scalar lifetime model (Context $\rightarrow \widehat{L}$)

Require: Training runs R_{train}
Ensure: Table T_L and global fallback $\widehat{L}_0$
1: Initialize empty map $T_L : c \mapsto$ multiset of lifetimes , multiset $S \leftarrow \emptyset$
2: **for all** run $r \in R_{\text{train}}$ **do**
3: **for all** file f created/opened in r **do**
4: $c \leftarrow \text{hash}(\text{callstack}_{\text{create/open}}(f))$
5: $L \leftarrow \text{lifetime}(f)$
6: insert L into $T_L[c]$; insert L into S
7: **end for**
8: **end for**
9: **for all** context c in T_L **do**
10: $\widehat{L}(c) \leftarrow \text{median}(T_L[c])$
11: **end for**
12: $\widehat{L}_0 \leftarrow \text{median}(S)$
13: **return** T_L and $\widehat{L}_0$

Algorithm 2: Lifecycle histogram (Context $\rightarrow \widehat{h}$)

Require: Training runs R_{train}, time bins B
Ensure: Table T_H and global fallback $\widehat{h}_0$
1: Initialize empty map $T_H : c \mapsto$ list of histograms, $A \leftarrow [\ ]$
2: **for all** run $r \in R_{\text{train}}$ **do**
3: **for all** file f created/opened in r **do**
4: $c \leftarrow \text{hash}(\text{callstack}_{\text{create/open}}(f))$
5: $h \leftarrow \text{io_histogram}(f, B)$ $\{h_b = \#\text{I/Os in bin } b\}$
6: append h to $T_H[c]$; append h to A
7: **end for**
8: **end for**
9: **for all** context c in T_H **do**
10: $\widehat{h}(c) \leftarrow \text{mean_vector}(T_H[c])$ {element-wise mean}
11: **end for**
12: $\widehat{h}_0 \leftarrow \text{mean_vector}(A)$
13: **return** T_H and $\widehat{h}_0$

3.2 Online inference at create/open time

At runtime, when a file is created or opened, CLiP computes the context identifier c from the current create/open call stack and looks it up in the trained table. Figure 1b summarizes online inference: the tracer intercepts `create/open` ①, hashes the creation call stack, and retrieves the stored prediction via a table lookup ②.

3.2.1 Lifetime prediction. Given c, the predictor returns:

$$\widehat{L} = \begin{cases} \widehat{L}(c), & \text{if } c \in T_L, \\ \widehat{L}_0, & \text{otherwise.} \end{cases}$$

This value is available at creation/open time and can be used by HPC storage components that need a lifetime estimate before I/O proceeds.

3.2.2 Lifecycle prediction. Given c, the predictor returns:

$$\widehat{h} = \begin{cases} \widehat{h}(c), & \text{if } c \in T_H, \\ \widehat{h}_0, & \text{otherwise.} \end{cases}$$

$\widehat{h}$ provides the predicted number of I/O events per time bin after first open, which matches the lifecycle definition used in the evaluation.

The bin width mainly affects the discretization quality. Finer bins increase temporal resolution but are more sensitive to small time shifts across bin boundaries, whereas a small number of coarse, log-spaced bins (0–4s up to 1024s+) provides a robust multi-scale summary, as commonly done in large-scale lifecycle analyses [40]. Importantly, the bin width has negligible impact on CLiP's overhead, as the cost of a trained hash table returning a scalar value or a pointer to an array of scalars is the same. As such, online inference is a constant-time hash+lookup [10, 14] independent of bin size.

4 Evaluation Methodology

This section describes the evaluation protocol used to assess (i) scalar file lifetime prediction and (ii) file lifecycle prediction (I/O activity distribution over time) at file creation/open time, using the creation context captured by the call-stack hash.

4.1 Experimental Protocol

We evaluate our predictor offline using I/O traces collected from repeated executions (runs) of the same workload. For each application, we considered a total of four chronological runs. We used run 1 for training and runs 2–4 for evaluation.

Training. During the first execution of an application, for each file creation context (i.e., the file creation call-stack hash), we store the behavior observed in the traces, including the file lifetime and a I/O activity profile.

Inference. At inference time, when a file is created/opened, CLiP computes the context identifier c (the hash of the create/open call stack) and performs a constant-time lookup in the per-application table. If c is present, CLiP returns the context-specific prediction learned during training. If c was never observed during training, CLiP falls back to an application-level default learned from all training files. For lifecycle prediction, this default is the element-wise mean histogram computed over all training files of the application, ensuring that inference always returns a valid prediction, even for unseen contexts.

4.2 Evaluated Workloads

We use three workloads from real HPC applications. As introduced in Table 1, traces were collected from three representative codes. LAMMPS [35] (Large-scale Atomic/Molecular Massively Parallel Simulator) is a molecular dynamics tool. We use a publicly available input file[1] to simulate a 1WDN Glutamine-Binding Protein with 1024 processes on 14 compute nodes. NAMD [28] is also a molecular dynamics tool. We use another publicly available input file[2] to run NAMD on an STMV virus. Incompact3d [39] is a Navier–Stokes solver. We use a publicly available X3D benchmark input file[3]. The table also contains information about the applications I/Os and I/O call stacks. Notably, the *callstacks_open* column of Table 1 shows that our lookup table will contain 4, 19, and 23 entries for LAMMPS, Incompact3d, and NAMD respectively.

4.3 Evaluation metrics

We report three families of metrics:

- **Lifetime prediction quality :** We report the fraction of files whose predicted lifetime is within a given relative error threshold of the *real* lifetime, and we summarize errors with statistics (median error rather than mean) to avoid being dominated by a few extreme files.
- **Lifetime prediction overhead :** We measure overhead as the wall-clock inference time added by the predictor, reported as average and standard deviation per I/O (in µs). This captures the runtime cost of using the model online and enables a direct comparison between CLiP (table lookup) and CNN-based baselines [34]. For the CNN baseline, inference was executed on an NVIDIA GeForce RTX 3070 Laptop GPU.
- **I/O activity profiling quality :** We split time after creation into a small number of coarse bins and compare predicted vs. observed I/O counts in each bin. Because many bins can be zero, we use a symmetric percentage error (stable when the true count is zero) and report the average accuracy per bin.

4.4 Experiments Performed

We perform two experiments for our two prediction targets.

***Experiment 1** : Lifetime prediction.* For each file, we predict a single scalar value representing its lifetime. In the evaluation, we compare the predicted lifetime against the lifetime observed in the trace for each file in the test runs. We also include as a baseline the pathname-based lifetime predictor that uses the full file path available at create/open time to infer lifetime via a CNN (convolutional neural network) model [34]. Both CLiP and the CNN baseline are trained and evaluated per application, with a separate model built for each application using run 1 as training data and runs 2–4 as test data.

***Experiment 2**: Lifecycle prediction (I/O activity over time).* For each file, we predict a coarse I/O activity profile after creation. We discretize time since creation into a small number of bins and predict the number of I/O operations expected in each bin. In evaluation, we compare predicted and observed per-bin I/O counts.

[1]https://www.hecbiosim.ac.uk/access-hpc/benchmarks
[2]https://www.ks.uiuc.edu/Research/namd/utilities/stmv/
[3]https://github.com/xcompact3d/X3D-benchmarking

Table 1: List of the applications used in the evaluation

	app	events	files	callstacks_open	callstacks_io	io_count	io_volume	span_s	lifetime_p50_s	lifetime_p95_s	io_per_file_p50	io_per_file_p95
0	incompact3d	2.67M	1.65K	19	67	2.67M	38.63GB	51.80m	0.73s	4.21s	1.29K	5.17K
1	lammps	36.90K	39	4	36	36.90K	372.31MB	40.87m	0.45s	1.36m	899	3.29K
2	namd	31.77K	39	23	138	31.77K	227.81GB	49.43m	1.59s	7.19m	203	7.00K

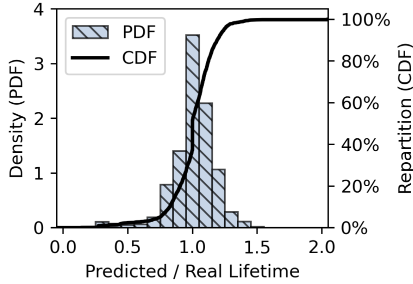

Figure 2: Lifetime prediction accuracy for Incompact3d using CLiP

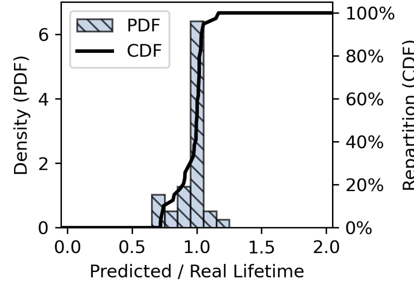

Figure 3: Lifetime prediction accuracy for LAMMPS using CLiP

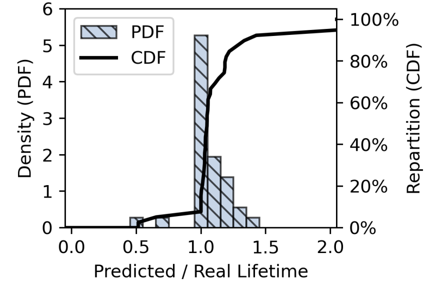

Figure 4: Lifetime prediction accuracy for NAMD using CLiP

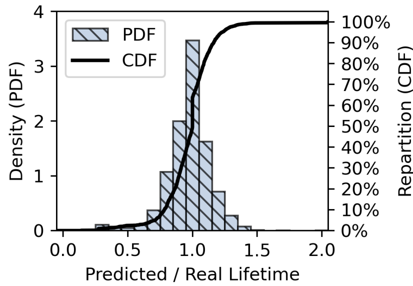

Figure 5: Lifetime prediction accuracy for Incompact3d using a CNN

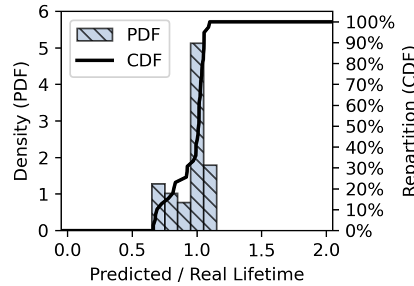

Figure 6: Lifetime prediction accuracy for LAMMPS using a CNN

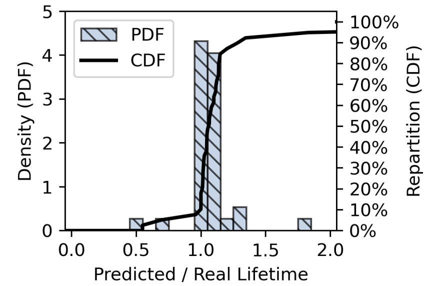

Figure 7: Lifetime prediction accuracy for NAMD using a CNN

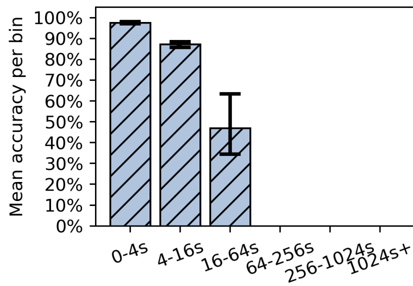

Figure 8: I/O count prediction accuracy per time interval for Incompact3d

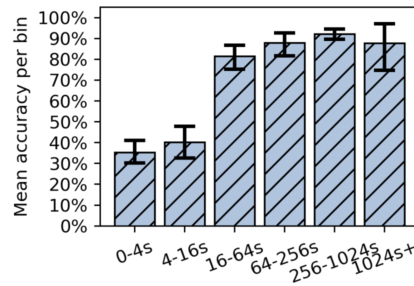

Figure 9: I/O count prediction accuracy per time interval for LAMMPS

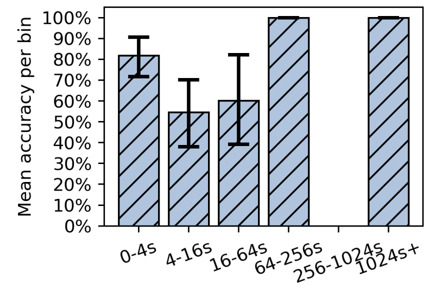

Figure 10: I/O count prediction accuracy per time interval for NAMD

Application	CLiP Scalar (μs/I/O)		CLiP Lifecycle (μs/I/O)		CNN (μs/I/O)	
	Avg	Std	Avg	Std	Avg	Std
LAMMPS	1.6139	0.5751	2.1274	0.7048	473.1587	98.5465
NAMD	1.5589	0.4447	4.4505	2.7046	500.7131	107.3837
Incompact3d	1.7588	0.6957	4.2520	6.1046	467.3402	125.3750

Table 2: Scalar lifetime and lifecycle prediction overhead

5 Results and Discussion

5.1 Experiment 1

Figures (2, 3, 4) report lifetime prediction accuracy of CLiP using the ratio Predicted / Real Lifetime on the x-axis (range 0–2). The left y-axis shows the PDF (density of files), while the right y-axis shows the CDF (cumulative repartition, in %). A ratio of 1 corresponds to a perfect prediction; values below 1 indicate under-estimation and values above 1 indicate over-estimation. Overall, the three figures show a strong concentration around 1, meaning that the creation call-stack context is a reliable signal to predict file lifetime at creation time.

In **LAMMPS** (Fig. 3), the distribution is the most concentrated, with 64.10% of predictions within ±5%, 74.36% within ±10%, and 89.74% within ±20% of the real lifetime. In **Incompact3d** (Fig. 2), the peak remains centred near 1 but with a wider spread at tight tolerances; 35.16% of predictions fall within ±5%, 55.74% within ±10%, and 82.82% within ±20%, reaching 90.47% within ±25%. In **NAMD** (Fig. 4), the distribution is also centred near 1 but exhibits a more visible tail; 48.72% of predictions are within ±5%, 61.54% within ±10%, and 76.92% within ±20% (and 79.49% within ±25%),

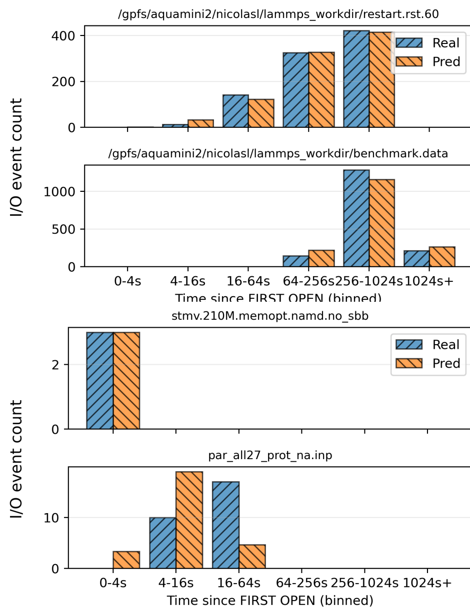

Figure 11: Example file lifecycles for LAMMPS and NAMD.

indicating that most files are well predicted while a smaller fraction incurs larger errors.

Figures (5, 6, 7) report the lifetime prediction accuracy of a state-of-the-art pathname-based CNN baseline[34] using the ratio Predicted / Real Lifetime. Across the three workloads, the distributions are centred around 1 but are generally less concentrated than CLiP.

In **Incompact3d** (Fig. 5), the CNN achieves 34.55% within ±5%, 54.83% within ±10%, and 81.66% within ±20% (and 88.46% within ±25%), closely tracking CLiP but with slightly lower accuracy at wider tolerances. In **LAMMPS** (Fig. 6), the CNN reaches 51.28% within ±5%, 76.92% within ±10%, and 84.62% within ±20% (and 87.18% within ±25%), remaining noticeably less concentrated than CLiP at ±5% and ±20%. In **NAMD** (Fig. 7), the CNN obtains 41.03% within ±5%, 58.97% within ±10%, and 79.49% within ±20% (and 82.05% within ±25%), with occasional larger over-estimations visible in the right tail.

5.2 Experiment 2

Figures (8,9,10) evaluate lifecycle prediction by comparing predicted vs. observed I/O counts over time since `first open`, using logarithmic bins (from 0–4s up to 1024s+). The y-axis reports mean per-bin accuracy (%) with error bars showing variability across files; empty bins indicate intervals with no I/O.

In **Incompact3d** (Fig. 8), accuracy peaks in the early bins (≈ 97% then ≈ 88%) and decreases to ≈ 45–50% in the next one. The absence of I/O in later bins indicates that the workload is dominated by short-lived files whose accesses occur soon after opening (burst-style I/O), leaving the long-time intervals mostly empty.

In **LAMMPS** (Fig. 9), early bins are harder because they often correspond to low-count regimes (many files perform only a handful of I/Os soon after the first open call). In that case, a deviation of only 1–2 I/Os—or a small timing shift across a bin boundary—translates into a large *relative* error, increasing variability explains the lower mean accuracy (≈ 35–40%), while later phases are predicted well (about 82–92% and still ≈ 88% in the last bin). In **NAMD** (Fig. 10), accuracy is moderate early (about 55–82%) and reaches ≈ 100% in late bins; one mid/late bin is empty (no I/O). This means that, in our traces, no file exhibits I/O activity during that time interval, reflecting phase-separated execution where I/O bursts occur in specific phases (initialization and/or late output/checkpoint) separated by compute-dominated periods; with logarithmic bins, such gaps naturally appear as empty bins.

Fig. 11 provides concrete examples of file lifecycles and illustrates both successes and limitations of our binned lifecycle predictor. Importantly, apparent errors can stem from *bin-boundary sensitivity* rather than from a fundamentally wrong prediction. Small temporal shifts can move I/O events across adjacent bins and yield large per-bin discrepancies even when the overall activity is comparable. We therefore complement per-bin accuracy with qualitative inspection (Fig. 11) and plan to use shift-tolerant metrics, such as time-series distances like DTW [3, 29], in future work.

For **LAMMPS** (left column), the `restart.rst.60` example (top) shows that the predicted distribution closely matches the real one across the active bins; both concentrate most I/O in the late phase, with similar magnitudes, which is consistent with the high per-bin accuracies observed for **LAMMPS** in Fig. 9. In contrast, the `benchmark.data` example (bottom) remains qualitatively correct (dominant activity in late bins), but exhibits a noticeable amplitude mismatch in the busiest interval, highlighting that even when the temporal location is captured, predicting exact counts can remain difficult for high-volume files. For **NAMD** (right column), `memopt.namd.no_sbb` (top) is a near-perfect case where both real and predicted activity are concentrated in a single early bin, yielding an almost identical histogram. The `par_all27_prot_na.inp` example (bottom) illustrates the main limitation of histogram-based evaluation: the real activity spans two adjacent bins, while the prediction shifts a significant fraction of I/O mass into one bin. Such small temporal shifts across bin boundaries produce large per-bin discrepancies even when the overall activity is comparable, which partly explains why lifecycle prediction appears less accurate (and more variable) than scalar lifetime prediction.

Table 2 reports the per-I/O inference overhead for CLiP (scalar and lifecycle) and the CNN baseline. CLiP remains in the few-microseconds range for both scalar and lifecycle prediction (∼1.6–4.5 μs/I/O), while the CNN incurs two orders of magnitude higher cost (∼467–501 µs/I/O), confirming that CLiP delivers lifecycle profiling with negligible runtime overhead. Memory-wise, assuming a perfect minimal hashing function, CLiP's per-application lookup tables are tiny, amounting to only a few KB at most ≈0.3,KB for scalar and ≈1.0,KB for lifecycle with 8 bins, raw payload, whereas the CNN baseline requires storing ≈100 M parameters[34] (≈400 MB in FP32) per application.

6 Conclusion

This paper presents *CLiP*, a creation-time predictor of file lifetime and file lifecycle for HPC applications. The method uses only the hash of the create/open call stack as context and performs inference by constant-time lookup in a per-application table. We evaluate on three representative HPC workloads (NAMD, Incompact3D, and LAMMPS). Overall, CLiP predicts ∼ 83.2% of file lifetimes

within ±20% relative error, comparable to a pathname-based CNN baseline ∼ 82.0%, while reducing prediction overhead by ∼ 99.7% (i.e., ≈ 292× faster). For lifecycle prediction (binned I/O counts after first open), accuracy reaches ≈ 97% in the `0-4s` bin on Incompact3D and ≈ 92% in the `256-1024s` bin on LAMMPS. These results show that create/open context enables proactive placement and caching decisions at file creation time with constant-time inference and negligible overhead.

A key limitation of lifecycle evaluation with coarse histograms is bin-boundary sensitivity: small temporal shifts can move I/O events between adjacent bins, and empty bins simply mean there is no I/O in that interval. As future work, we will move from binned histograms to time-series representations of file activity and evaluate predictions using Dynamic Time Warping (DTW), which better tolerates small temporal misalignments while still penalizing true shape differences. We also plan to integrate our predictions into concrete system policies, including (i) *striping decisions* (e.g., selecting stripe count/size based on predicted lifetime/lifecycle) and (ii) predictive prefetching (e.g., prefetch only when a read phase is expected soon), enabling end-to-end improvements driven by creation-time hints. The implementation of CLiP and the scripts to reproduce the experiments are available open source at the following link https://github.com/hocinemahni/CLip.

References

[1] Lydia Ait-Oucheggou, Stéphane Rubini, Abdella Battou, and Jalil Boukhobza. 2025. QM-ARC: QoS-aware Multi-tier Adaptive Cache Replacement Strategy. *Future Generation Computer Systems* 163 (2025), 107548.

[2] Benjamin Berg, Daniel S. Berger, and Sara et al. McAllister. 2020. The CacheLib Caching Engine: Design and Experiences at Scale. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI '20)*. 753–768.

[3] Donald J Berndt and James Clifford. 1994. Using dynamic time warping to find patterns in time series. In *Proceedings of the 3rd international conference on knowledge discovery and data mining*. 359–370.

[4] Jean Luca Bez, Suren Byna, and Shadi Ibrahim. 2023. I/O Access Patterns in HPC Applications: A 360-Degree Survey. *ACM Comput. Surv.* 56, 2 (Sept. 2023), 1–41. doi:10.1145/3611007

[5] Jalil Boukhobza, Pierre Olivier, Wen Sheng Lim, Liang-Chi Chen, Yun-Shan Hsieh, Shin-Ting Wu, Chien-Chung Ho, Po-Chun Huang, and Yuan-Hao Chang. 2025. A Survey on Flash-Memory Storage Systems: A Host-Side Perspective. *ACM Transactions on Storage* 21, 3 (2025), 1–59.

[6] Philip Carns, Kevin Harms, William Allcock, Charles Bacon, Samuel Lang, Robert Latham, and Robert Ross. 2011. Understanding and improving computational science storage access through continuous characterization. *ACM Transactions on Storage (TOS)* 7, 3 (2011), 1–26.

[7] Philip Carns, Kevin Harms, William Allcock, Charles Bacon, Samuel Lang, Robert Latham, and Robert Ross. 2011. Understanding and Improving Computational Science Storage Access through Continuous Characterization. *ACM Trans. Storage* 7, 3 (Oct. 2011), 1–26. doi:10.1145/2027066.2027068

[8] Philip Carns, Robert Latham, Robert Ross, Kamil Iskra, and Katherine Riley. 2009. 24/7 Characterization of petascale I/O workloads. *Proceedings - IEEE International Conference on Cluster Computing, ICCC* (Oct. 2009), 1–10. doi:10.1109/CLUSTR.2009.5289150

[9] Amina Chikhaoui, Laurent Lemarchand, Kamel Boukhalfa, and Jalil Boukhobza. 2021. Multi-objective optimization of data placement in a storage-as-a-service federated cloud. *ACM Transactions on Storage (TOS)* 17, 3 (2021), 1–32.

[10] Zbigniew J Czech, George Havas, and Bohdan S Majewski. 1997. Perfect hashing. *Theoretical Computer Science* 182, 1-2 (1997), 1–143.

[11] Bin Dong, Xiuqiao Li, Limin Xiao, and Li Ruan. 2012. A new file-specific stripe size selection method for highly concurrent data access. In *2012 ACM/IEEE 13th International Conference on Grid Computing*. IEEE, 22–30.

[12] Matthieu Dorier, Shadi Ibrahim, Gabriel Antoniu, and Rob Ross. 2014. Omnisc'IO: A grammar-based approach to spatial and temporal I/O patterns prediction. In *SC'14: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 623–634.

[13] Matthieu Dorier, Shadi Ibrahim, Gabriel Antoniu, and Rob Ross. 2015. Using formal grammars to predict I/O behaviors in HPC: The omnisc'IO approach. *IEEE Transactions on Parallel and Distributed Systems* 27, 8 (2015), 2435–2449.

[14] Edward A Fox, Lenwood S Heath, Qi Fan Chen, and Amjad M Daoud. 1992. Practical minimal perfect hash functions for large databases. *Commun. ACM* 35, 1 (1992), 105–121.

[15] Anjus George, Andreas Dilger, Michael J Brim, Richard Mohr, Amir Shehata, Jong Youl Choi, Ahmad Maroof Karimi, Jesse Hanley, James Simmons, Dominic Manno, et al. 2025. Lustre Unveiled: Evolution, Design, Advancements, and Current Trends. *ACM Transactions on Storage* 21, 3 (2025), 1–109.

[16] Wei Hu, Guang-ming Liu, Qiong Li, Yan-huang Jiang, and Gui-lin Cai. 2016. Storage wall for exascale supercomputing. *Frontiers of Information Technology & Electronic Engineering* 17, 11 (2016), 1154–1175.

[17] Michael Jones, Jeremy Kepner, William Arcand, David Bestor, Bill Bergeron, Vijay Gadepally, Michael Houle, Matthew Hubbell, Peter Michaleas, Andrew Prout, et al. 2017. Performance measurements of supercomputing and cloud storage solutions. In *2017 IEEE High Performance Extreme Computing Conference (HPEC)*. IEEE, 1–5.

[18] Andreas Knüpfer, Christian Rössel, Dieter an Mey, Scott Biersdorff, Kai Diethelm, Dominic Eschweiler, Markus Geimer, Michael Gerndt, Daniel Lorenz, Allen Malony, et al. 2012. Score-p: A joint performance measurement run-time infrastructure for periscope, scalasca, tau, and vampir. In *Tools for High Performance Computing 2011: Proceedings of the 5th International Workshop on Parallel Tools for High Performance Computing, September 2011, ZIH, Dresden*. Springer, 79–91.

[19] Jianwei Liao, Guoqiang Xiao, Xiaoyan Liu, and Lingyu Zhu. 2014. Dynamic Stripe Management Mechanism in Distributed File Systems. In *IFIP International Conference on Network and Parallel Computing*. Springer, 497–509.

[20] Hocine Mahni, Stéphane Rubini, Sébastien Gougeaud, Philippe Deniel, and Jalil Boukhobza. 2025. Multicriteria File-Level Placement Policy for HPC Storage. In *Proceedings of the 40th ACM/SIGAPP Symposium on Applied Computing*. 1399–1406.

[21] Florent Monjalet and Thomas Leibovici. 2019. Predicting file lifetimes with machine learning. In *International Conference on High Performance Computing*. Springer, 288–299.

[22] Mohammed Islam Naas, François Trahay, Alexis Colin, Pierre Olivier, Stéphane Rubini, Frank Singhoff, and Jalil Boukhobza. 2021. EZIOTracer: unifying kernel and user space I/O tracing for data-intensive applications. In *Proceedings of the Workshop on Challenges and Opportunities of Efficient and Performant Storage Systems*. 1–11.

[23] Sarah Neuwirth and Arnab K Paul. 2021. Parallel i/o evaluation techniques and emerging hpc workloads: A perspective. In *2021 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, 671–679.

[24] Louis-Marie Nicolas, Salim Mimouni, Philippe Couvée, and Jalil Boukhobza. 2023. GrIOt: Graph-based Modeling of HPC Application I/O Call Stacks for Predictive Prefetch. In *Proceedings of the SC'23 Workshops of the International Conference on High Performance Computing, Network, Storage, and Analysis*. 1195–1201.

[25] Louis-Marie Nicolas, Salim Mimouni, Philippe Couvée, and Jalil Boukhobza. 2026. I/O patterns modeling of HPC applications with call stacks for predictive prefetch. *Future Generation Computer Systems* 175 (2026), 108034.

[26] Tirthak Patel, Suren Byna, Glenn K Lockwood, Nicholas J Wright, Philip Carns, Robert Ross, and Devesh Tiwari. 2020. Uncovering access, reuse, and sharing characteristics of {I/O-Intensive} files on {Large-Scale} production {HPC} systems. In *18th USENIX Conference on File and Storage Technologies (FAST 20)*. 91–101.

[27] Jingxian Peng, Lihua Yang, Huijun Wu, Wenzhe Zhang, Zhenwei Wu, Wei Zhang, Jiaxin Li, Yiqin Dai, and Yong Dong. 2025. A Survey on Machine Learning-based HPC I/O Analysis and Optimization. *IEEE Transactions on Parallel and Distributed Systems* (2025).

[28] James C Phillips, David J Hardy, Julio DC Maia, John E Stone, João V Ribeiro, Rafael C Bernardi, Ronak Buch, Giacomo Fiorin, Jérôme Hénin, Wei Jiang, et al. 2020. Scalable molecular dynamics on CPU and GPU architectures with NAMD. *The Journal of chemical physics* 153, 4 (2020).

[29] Hiroaki Sakoe and Seibi Chiba. 2003. Dynamic programming algorithm optimization for spoken word recognition. *IEEE transactions on acoustics, speech, and signal processing* 26, 1 (2003), 43–49.

[30] Sameer S Shende and Allen D Malony. 2006. The TAU parallel performance system. *The International Journal of High Performance Computing Applications* 20, 2 (2006), 287–311.

[31] Shane Snyder, Philip Carns, Kevin Harms, Robert Ross, Glenn K Lockwood, and Nicholas J Wright. 2016. Modular hpc i/o characterization with darshan. In *2016 5th workshop on extreme-scale programming tools (ESPT)*. IEEE, 9–17.

[32] Ahmad Tarraf, Alexis Bandet, Francieli Boito, Guillaume Pallez, and Felix Wolf. 2024. Capturing periodic I/O using frequency techniques. In *2024 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 465–478.

[33] Ahmad Tarraf and Felix Wolf. [n. d.]. Improving I/O Phase Predictions in FTIO Using Hybrid Wavelet-Fourier Analysis. *Frontiers in High Performance Computing* 3 ([n. d.]), 1638924.

[34] Luis Thomas, Sebastien Gougeaud, Stéphane Rubini, Philippe Deniel, and Jalil Boukhobza. 2021. Predicting file lifetimes for data placement in multi-tiered storage systems for HPC. In *Proceedings of the Workshop on Challenges and Opportunities of Efficient and Performant Storage Systems*. 1–9.

[35] Aidan P Thompson, H Metin Aktulga, Richard Berger, Dan S Bolintineanu, W Michael Brown, Paul S Crozier, Pieter J in't Veld, Axel Kohlmeyer, Stan G Moore, Trung Dac Nguyen, et al. 2022. LAMMPS-a flexible simulation tool for particle-based materials modeling at the atomic, meso, and continuum scales. *Computer Physics Communications* 271 (2022), 108171.
[36] Chen Wang, Jinghan Sun, Marc Snir, Kathryn Mohror, and Elsa Gonsiorowski. 2020. Recorder 2.0: Efficient parallel I/O tracing and analysis. In *2020 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. IEEE, 1–8.
[37] Chen Wang, Izzet Yildirim, Hariharan Devarajan, Kathryn Mohror, and Marc Snir. 2025. Recorder: Comprehensive Parallel I/O Tracing and Analysis. *arXiv preprint arXiv:2501.04654* (2025).
[38] Daniel Lin-Kit Wong, Hao Wu, Carson Molder, Sathya Gunasekar, Jimmy Lu, Snehal Khandkar, Abhinav Sharma, Daniel S Berger, Nathan Beckmann, and Gregory R Ganger. 2024. Baleen:{ML} Admission & Prefetching for Flash Caches. In *22nd USENIX Conference on File and Storage Technologies (FAST 24)*. 347–371.
[39] xcompact3d/Incompact3d. 2024. xcompact3d/Incompact3d. doi:10.5281/zenodo.5870206 [Online; Accessed 5 January 2024].
[40] Bin Yang, Hao Wei, Wenhao Zhu, Yuhao Zhang, Weiguo Liu, and Wei Xue. 2024. Full lifecycle data analysis on a large-scale and leadership supercomputer: What can we learn from it?. In *2024 USENIX Annual Technical Conference (USENIX ATC 24)*. 917–933.