ORION: Observability Framework for Hybrid Workflow-Centric I/O Monitoring in HPC Systems

Hariharan Devarajan[1], Ray A. O. Sinurat[2], Izzet Yildirim[3], Olga Kogiou[4],
Aniruddha Marathe[1], and Nikoli Dryden[1]
[1]Lawrence Livermore National Laboratory, [2]University of Chicago, [3]Illinois Tech, and [4]Florida State University
hariharandev1@llnl.gov, rayandrew@uchicago.edu, iyildirim@hawk.illinoistech.edu, ok22b@fsu.edu,
marathe1@llnl.gov, and dryden1@llnl.gov

***Abstract*—Scientific workflows depend on efficient data exchange behavior, so practical, low-overhead I/O performance visibility is essential for HPC optimization. Existing I/O monitoring tools often force extreme choices, such as tracing all I/O events or using only coarse profiling, which either inflates data volume or hides attribution context. We present ORION, a workflow-centric I/O monitoring framework that enables users to dynamically choose temporal and spatial monitoring granularity for different workflow regions. First, we show that multi-faceted I/O profiling across dimensions such as time, processes, and files is critical for understanding workflow behavior while adding low overhead, with top-k bottleneck fidelity stable within the recommended $\alpha \in [2, 8]$ operating range. Second, we show that ORION captures critical I/O events in high-interest regions while preserving aggregated views in low-interest regions, improving both fidelity and efficiency. Third, we show that ORION's reconciliation algorithm combines I/O traces and I/O profiles at workload-dependent rates from 0.35M (DLIO) to 1.15M events/s (IOR), enabling scalable post hoc analysis. Finally, we show on large-scale workloads, including Montage and Llama fine-tuning, that ORION Selective Aggregation identifies critical bottlenecks with $6.8\times$–$36\times$ fewer trace events (100% ordered top-k severe-bottleneck accuracy on all workloads) and remains portable across two heterogeneous LLNL systems (Corona and Tuolumne), supporting practical I/O monitoring at HPC scale.**

***Index Terms*—HPC, I/O monitoring, observability, performance analysis, selective aggregation, tracing, workflow-centric analysis**

I. Introduction

Modern scientific workflows are fundamentally data-exchange driven, with simulation, analytics, and AI stages continuously moving data across memory, network, and storage layers on heterogeneous hardware (CPUs, GPUs, interconnects, storage subsystems). Workflows on modern HPC systems remain difficult to tune [1]–[4] because this exchange behavior changes with phase, scale, and workload composition, creating interacting effects across processes, nodes, files, and devices, and shifting dynamically over time and space in ways that change where data is produced, transferred, and consumed [5], [6]. Effective optimization therefore depends on high-fidelity visibility into I/O and storage behavior across workflow regions.

The I/O monitoring landscape can be categorized into three groups: profiling tools [7], [8], tracing tools [9], [10], and hybrid tools [11], [12] that support both modes. Profiling tools such as Darshan [7] and Extrae [8] provide efficient aggregated views of I/O behavior and workload trends. Tracing tools such as Pilgrim [13], ScalaIOTrace [14], Recorder [9], and DFTracer [10] capture detailed event sequences that expose ordering, burstiness, and cross-layer interactions in data movement. General performance frameworks such as Score-P [12], Scalasca [15], and HPCToolkit [16] provide broader profiling/tracing context across fidelity levels.

Despite the impact of existing I/O tools, production workflows still cannot obtain workload-tailored, high-fidelity I/O performance insight. This is due to the following three challenges. First, fixed-granularity profiling often lacks sufficient spatial and temporal detail, because coarse aggregation can hide phase transitions and cross-process or cross-file imbalance in I/O behavior. Second, full tracing often over-collects data and raises analysis and runtime costs; in Montage [17], aggregated monitoring yields $\sim$1K events while full tracing yields $\sim$2B events; ORION selective aggregation recovers all bottlenecks with only $\sim$219M events ($6.8\times$ reduction), showing both extremes leave analysts poorly served. Third, post hoc reconciliation of profiles and traces remains fragile, because imperfect time alignment across data sources weakens bottleneck attribution. Together, these empirical gaps motivate a tunable I/O monitoring approach that adapts collection and aggregation to each workload, so workflows capture the right evidence at the right time and location without data bloat or prohibitive overhead.

We design ORION to adapt monitoring fidelity to workflow context, enabling workload-tailored, high-fidelity performance insight. We achieve this with three complementary components: First, a Structural Aggregator enables users to define workload-specific temporal and spatial aggregation dimensions that allow workflows to retain critical structure while avoiding unnecessary event explosion. Second, a rule-based selection model enables context-aware switching between coarse and fine monitoring across different temporal and spatial regions, so workflows can highlight the most interesting regions while collecting detailed evidence only where and when it is most valuable. Third, a Hybrid Analysis Engine unifies profile- and trace-level evidence into a single analytical view so workflows can support cross-granularity bottleneck attribution and optimization. Together, these components convert monitoring from an all-or-nothing choice into a workload-centric capability that enables practical, high-fidelity

optimization at HPC scale. Our contributions are as follows:

1) We design ORION as a workflow-centric framework that enables workload-tailored, high-fidelity performance data insight without missing critical data or incurring prohibitive overhead.
2) We design a rule-based event classifier that routes events between aggregation and tracing at $\sim$50 M events/s in the synthetic query benchmark, enabling context-aware granularity switching with negligible capture overhead.
3) We develop the event-reconciliation algorithm that unifies profile and trace evidence for mixed-granularity bottleneck detection and optimization.
4) We demonstrate strong impact and portability across two heterogeneous LLNL systems (Corona and Tuolumne) on large-scale workflows (Montage and Llama fine-tuning), achieving actionable diagnosis with $6.8\times$–$36\times$ event reduction versus full tracing, including a $9.6\times$ reduction for Montage versus $\sim$2B DFTracer events.

II. Background

A. Complexity of Scientific Workflows in HPC

Scientific workflows in HPC exhibit distinct I/O and storage behavior across three workload classes: simulation, analytics, and AI. In simulation-centric workloads, CM1 [18] and VPIC [19] produce periodic checkpoint bursts, large restart files, and synchronized output phases that stress shared file systems and metadata services. In analytics-centric workloads, KMeans [20] clustering pipelines and BDCats [21] data analytics are dominated by staged read-transform-write patterns with many intermediate files, where throughput depends on partitioning quality and metadata locality. In AI-centric workloads, CosmoFlow [22] training involves sustained high-throughput reads of multidimensional scientific tensors and periodic checkpoint writes, UNet3D [23] training adds patch-based volumetric data loading with frequent minibatch staging and checkpointing, and LLM pipelines such as Llama [24] combine large sequential dataset reads with iterative model-state and optimizer-state writes, collectively creating bursty storage demand and tight coupling between input pipelines and accelerator utilization.

In coupled workflows, these classes exchange data in ways that create complex cross-stage I/O behavior. In MuMMI [25], simulation outputs are continuously filtered and analyzed, selected regions are passed to ML models, and model outputs feed back into subsequent simulation decisions. In 1000 Genomes [26] pipelines, multi-stage preprocessing and variant-analysis tasks exchange large intermediate datasets across storage-backed stages with repeated read/write reuse. In Montage workflows [17], image tiles and metadata products flow across mosaicking stages with a mix of many small file operations and larger aggregate products. In iPIC3D [27] campaigns, simulation data is exchanged with downstream analysis and storage checkpoints across repeated phases, creating alternating burst and steady-state I/O regions. These exchanges involve mixed granularities (small metadata-heavy files plus large contiguous arrays), mixed access semantics (append, random read, burst write), and mixed spatial paths (intra-node, inter-node, and storage-backed handoff), making workflow performance highly sensitive to when, where, and how data moves.

B. Diverse Tools in HPC

The HPC monitoring/tooling landscape is diverse along two axes: *component focus* and *collection mode*. By component focus, one class of tools targets application and runtime behavior. Score-P [12] is designed to generate unified performance measurements across programming models (MPI, OpenMP, CUDA, and others), with the goal of producing consistent profiles and traces that can be compared across runs and systems. Scalasca [15] builds on such measurements to identify and quantify inefficiency patterns, particularly communication and synchronization bottlenecks, so users can localize scaling limits and load imbalance at large process counts. HPCToolkit [16] emphasizes statistically sampled call-path attribution to pinpoint where execution time and hardware stalls accumulate, helping developers prioritize optimization effort on the most expensive code regions.

A second class targets storage and I/O behavior. Darshan [7] provides lightweight job-level summaries to characterize I/O usage at scale and quickly answer what files were accessed, which interfaces were used, and where performance anomalies likely occurred. Recorder [9] captures fine-grained I/O traces with temporal ordering so users can reconstruct access sequences, understand inter-process interactions, and diagnose contention that aggregate counters can hide. DFTracer [10] focuses on end-to-end dataflow and I/O timeline analysis across workflow components, aiming to correlate computation phases with storage behavior and reveal cross-layer bottlenecks. DataCrumbs [28] targets provenance-aware I/O observability, attaching contextual metadata to I/O events so analysts can trace data movement decisions and reason about bottlenecks across POSIX, MPI-IO, and higher-level libraries.

By collection mode, profilers provide compact aggregate summaries with low overhead and are effective for broad trend analysis, tracers provide event-level fidelity that explains ordering and dependency effects in detail, and hybrid or switchable systems combine both modes for flexible analysis workflows. Together, these approaches provide a strong, complementary foundation for HPC performance engineering by enabling users to study behavior at multiple fidelities and across multiple software and hardware layers.

III. Motivation

Modern HPC workflows require three capabilities: richer aggregation dimensions to expose bottlenecks coarse views miss, selective profiling/tracing to preserve fidelity while controlling data volume, and reconciliation to preserve attribution across workflow regions. Table I demonstrates these needs by comparing bottleneck coverage against trace-size cost across representative monitoring modes.

TABLE I
BOTTLENECK COVERAGE VS. TRACE-SIZE/OVERHEAD COST ACROSS MONITORING MODES (UNET3D DLIO, 8 NODES, 1024 FILES). TRACE SIZE IS ON-DISK COMPRESSED JSON LINES (GZIP), MATCHING DFTRACER'S STORAGE PATH.

Mode	Bottlenecks	Size (MB)	OH (%)
Darshan Profile (File)	0/96	0.009	3.41
Score-P Profile (File)	0/96	0.010	3.58
Darshan Profile (File+10 bins)	2/96	0.011	4.98
DFTracer Trace	96/96	0.128	5.84
Score-P Trace	96/96	0.205	7.20
ORION Profile	94/96	0.051	3.41
ORION Selective	96/96	0.081	3.35

We run a UNet3D DLIO benchmark on 8 nodes with 1024 files (mean sample size 140 MB, standard deviation 60 MB) and compare seven monitoring modes. Trace size is reported as on-disk compressed JSON Lines output using gzip, matching the storage path used by DFTracer; therefore, size should be interpreted as compressed monitoring volume rather than raw in-memory event-record size. Darshan Profile and Score-P Profile (file-only) both find 0 bottlenecks at similar cost (1K events; 0.009/0.010 MB; 3.41%/3.58% overhead), while Darshan with 10 fixed time bins finds only 2 bottlenecks (10K events, 0.011 MB, 4.98% overhead). DFTracer full tracing finds 96 bottlenecks with 1.38M events and 0.128 MB compressed trace size (5.84% overhead). Score-P Trace finds the same 96 bottlenecks but produces a 1.6× larger compressed trace (2.21M events, 0.205 MB, 7.2% overhead).

ORION profile-style aggregation retains 94 bottlenecks (97.9% of DFTracer) at much lower cost (279K events, 0.051 MB, 3.41% overhead), and ORION selective mode recovers all 96 bottlenecks with 479K events and 0.081 MB (3.35% overhead). This motivates DFTracer full tracing as our reference baseline. ORION is implemented with DFTracer and Darshan and designed for the I/O-facing Score-P adapter; HPCToolkit [16] is included only as broader performance-tool context. We use DFTracer because it is the strongest available user-space I/O tracing baseline for our target workflows: it has lower tracing overhead than the alternatives in this comparison, exposes workflow context needed by the WisIO/DFAnalyzer bottleneck stack, and provides the most direct component-level integration for evaluating selective aggregation without changing the downstream diagnosis method.

IV. ORION

ORION is a plugin interface that extends both existing I/O monitoring tools and downstream analysis engines. On the monitoring side, it enhances existing performance tools with workload-centric, tunable aggregation and tracing; on the analysis side, it provides functions to efficiently canonicalize and merge profile and trace records into a unified representation for mixed-granularity analysis.

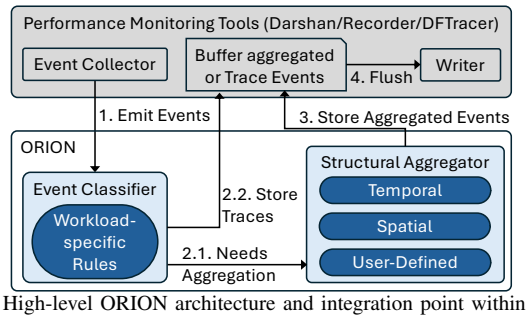


Fig. 1. High-level ORION architecture and integration point within existing I/O monitoring pipelines.

A. Architecture for Capturing with ORION

For the capture path, ORION focuses on three high-level objectives:

1) **Plugin-design:** insert ORION logic between event collection and event buffering so existing tools (e.g., Darshan, Recorder, DFTracer) can reuse their current backends.
2) **Workload-centric flexibility:** allow users to specify what to aggregate versus trace and define aggregation dimensions so monitoring behavior matches workload intent.
3) **Low-overhead operation:** keep selection and aggregation lightweight, with low single-digit application overhead in our evaluated workloads, so monitoring can remain continuously enabled in production.

Figure 1 shows how ORION is integrated into an existing I/O monitoring pipeline and maps each handoff to the numbered arrows in the diagram. Instead of buffering collected events directly, the tool's Event Collector emits them to the ORION *Event Classifier* (Figure 1, arrow 1). Traced events go straight to the tool buffer (arrow 2.2); the rest go to the *Structural Aggregator* (arrow 2.1), which stores aggregated records in the same buffer (arrow 3). The tool's writer flushes the buffer as usual (arrow 4), preserving current output formats and analysis workflows.

The **integration point** is intentionally narrow within the performance tools. ORION attaches after raw events are captured and before backend buffering/serialization. The monitoring tool remains responsible for instrumentation, timestamping, and final persistence, while ORION is responsible for workload-centric event decision logic. This split keeps deployment practical: tools retain stable collectors and storage code paths, and ORION contributes adaptive selection/aggregation semantics.

1) Event Classifier: The Event Classifier is the first ORION component in the capture pipeline (Figure 1, arrow 1) and provides a user-facing interface that decides, for each event, whether it should be aggregated or traced before downstream buffering (Figure 1, arrows 2.1 and 2.2). The key challenges are to (i) support a wide range of workload policies, (ii) stay efficient at high event rates without adding significant overhead, and (iii) remain compatible with event formats and feature sets emitted by different monitoring tools. ORION addresses this by defining classification policies in YAML, where rules can include events as aggregated events or exclude them from aggregation (thereby keeping them in trace form),

TABLE II
WORKLOAD-DERIVED SELECTIVE POLICIES.

Workload	Prior knowledge	Trace	Aggregate	Dims.
Montage	many small files	stage/file	repetitive POSIX/stdio	time,file,stage
UNet3D	loader/ckpt. phases	loader, ckpt.	low-level metadata	epoch,step,cat.
Llama-1B	step/ckpt. state	ckpt., optimizer	noncritical access	epoch,step,func.

while preserving a consistent policy model across tools such as Darshan, Recorder, and DFTracer. Event categories are fully user-defined at any granularity (e.g., via DFTracer decorators); the same classification/aggregation logic applies uniformly to any event type—POSIX, ROCm, CUDA—not just I/O calls.

Not all events have equal diagnostic value; tracing all inflates overhead while aggregating all hides critical signals. ORION therefore balances cost and fidelity through three practical classification criteria motivated by real-workload evaluation cases:

1) **Impact on overall performance:** prioritize tracing of events that materially affect end-to-end runtime, while aggregating lower-impact events. In Llama-1B fine-tuning, this criterion is used to reduce monitoring overhead by aggregating costly-to-capture, lower-value activity while preserving high-value diagnostic signal.
2) **Focus on specific workflow regions:** preserve detail in workflow phases that are analytically important and aggregate less critical phases. In UNet3D, we keep finer visibility on data-loading behavior while more aggressively aggregating less critical checkpoint-related activity.
3) **Granularity of performance data:** control event explosion by aggressively aggregating very fine-grained, repetitive operations while retaining enough structure for diagnosis. In Montage, numerous small accesses are strong candidates for aggregation to reduce performance-data bloat without losing bottleneck interpretability.

ORION policies are workload-specific but scale-invariant across the reported runs: developers choose rules from workflow names, file classes, phases, checkpoint/data-loading boundaries, epoch/step context, and event categories known before full tracing. DFTracer full traces are used only as the evaluation reference. **Writing a policy in practice.** Authoring is guided refinement, not blind search: users start from an all-aggregate policy at α=4, which needs no workload knowledge yet preserves 10/10 top-k on Montage and UNet3D and 8/10 on Llama-1B (Section V-C), then promote the phases, files, or categories flagged by the analyzer to tracing. Since rules only choose trace vs. aggregate, incomplete rules never discard events and degrade toward this baseline; workload-agnostic random/uniform selection reaches only 30–75%.

At initialization, the YAML rules are parsed into a vector of AST representations so they can be evaluated efficiently at runtime. The rule engine supports logical operators (AND, OR, NOT), comparison operators ($==, !=, <, >$), and wildcard-style regular expression matching (LIKE with $*$), allowing users to express complex policies over event features—for example, {`field: category, op: LIKE, value: ''checkpoint*'', action: aggregate`} aggregates all checkpoint-category events. During load time, rule tags are validated against features actually emitted by the underlying tool to prevent misconfiguration, and referenced aggregation dimensions are checked for availability in event data. At runtime, each incoming event provides concrete values for rule variables, the AST expression is evaluated with short-circuit AND/OR nodes, and the classifier marks the event for tracing or aggregation according to the policy result. Let N_r be the number of rules and V the number of AST vertices; the worst-case per-event classification cost is $O(N_r \cdot V)$, and in practice remains low because short-circuit evaluation and small N_r (typically $\sim$10 rules) keep evaluation fast.

2) Structural Aggregator: The Structural Aggregator is the second component in the same capture path (Figure 1, arrow 2.1) and transforms events tagged for aggregation into compact, multi-dimensional summaries that are emitted alongside traced records to the existing tool buffers (Figure 1, arrow 3). The main objective is to enable users to define workload-specific temporal and spatial aggregation dimensions so workflows retain critical structure while avoiding unnecessary event explosion. The key challenges are to (i) support flexible aggregation dimensions that users can define based on workload and monitoring goals, (ii) maintain high throughput for aggregation at event rates produced by modern I/O tools, and (iii) ensure aggregated records preserve critical structural context for bottleneck diagnosis without becoming too coarse-grained.

ORION addresses these challenges through a structured aggregation pipeline. First, users define aggregation dimensions from available event features emitted by the underlying performance tool. For example, Darshan events can be aggregated by file, rank, or operation type (read/write), while DFTracer events expose richer features such as category, function name, arguments, return values, timestamps, process and thread IDs, and workload context (e.g., epoch and step), all of which can be used as dimensions. Second, dimensions can be specified in two ways: (a) directly in the same YAML policy used by the classifier, or (b) through event-level contextual tags (key/value), as supported by DFTracer. Third, the selected dimension set is materialized into a C++ key array, which is mapped through a hash table (C++ `unordered_map`) to per-key aggregation state storing statistics such as duration, count, size, offsets, and related summaries. Fourth, ORION uses streaming-friendly updates (sum, count, min, max) so statistics are incrementally maintained per event at high throughput. Finally, each aggregated output is emitted as a single record containing both key dimensions and value statistics, which allows efficient downstream buffering/analysis without additional transformation. Let K be the number of dimensions in the key and S the number of maintained statistics; because hash insert/lookup is average-case $O(1)$, the practical per-event aggregation cost is $O(K + S)$.

B. Tuning Time Granularity

The goal of time-granularity tuning is to identify an aggregation interval that preserves meaningful temporal behavior while still reducing trace volume. If the interval is too small, ORION behaves close to full tracing because events are split across many narrow bins and aggregation becomes ineffective. If the interval is too large, temporally distinct phases are merged and important dynamics for bottleneck diagnosis are lost. ORION therefore uses a target-event heuristic to choose a practical interval efficiently.

Let d_e denote the characteristic duration (or maximum expected cost) of a target event of interest e. We select the aggregation interval Δt using

$$\Delta t = \alpha \cdot d_e, \quad \alpha \in [2, 8].$$

Target events depend on workload semantics. For simulation workloads, e can be one iteration. For deep-learning training, e can be one training step. For LLM training, e can be a small collection of consecutive steps that together represent a meaningful compute unit. Once this target event is identified, choosing Δt between $2\times$ and $8\times$ its duration provides a robust operating range for selective aggregation.

The lower bound ($\alpha < 2$) is usually ineffective because events are not aggregated enough and behavior approaches raw tracing. The upper bound ($\alpha > 8$) is usually too coarse because multiple temporal behaviors collapse into one window, reducing fidelity for downstream analysis. This bounded rule gives users a simple, workload-aware method to tune temporal aggregation without exhaustive search.

Estimating d_e in practice. In practice, d_e is estimated from a short representative run by measuring the duration of the target event class (e.g., iteration, step, or dominant recurring I/O phase). We use $\alpha = 4$ as a default start point and tune within $[2, 8]$. In the structural-aggregation microbenchmark (Section V-B1), $d_e \approx 12.5\,\text{ms}$, so the recommended range maps to $\Delta t \in [25\,\text{ms}, 100\,\text{ms}]$; tested windows from 10 ms to 30 s stress-test under- and over-aggregation. This heuristic is workload-dependent (re-estimate d_e when phase structure changes); a real-workload snapshot is consistent with this: in Llama-1B, coarse 5 s-only aggregation drops to 80% top-k, while selective finer-grain retention restores 100%, showing over-coarsening harms burst-heavy phases. Sweeping α from 2 to 32 across IOR, DLIO, and the Montage/UNet3D/Llama-1B production workloads confirms this bound generalizes: top-k accuracy is stable within $\alpha \in [2, 8]$ but degrades by 25% at α=16 and 29% at α=32 as critical events merge into coarser buckets.

C. Architecture for Analyzing with ORION

For the analysis path, ORION focuses on three high-level objectives:

1) **Diverse resolution:** merge records produced at different granularities (e.g., node/system utilization profiles with coarser time scales and application-level aggregates with workflow keys such as epoch and step) into a common timeline for joint reasoning.

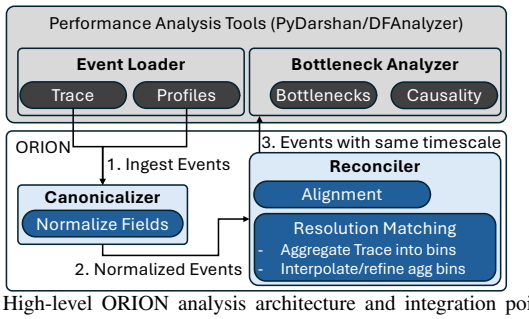


Fig. 2. High-level ORION analysis architecture and integration point over existing analysis tools.

Algorithm 1 ORION mixed-granularity reconciliation.

Require: Traces T, profiles P, target Δ_a, profile width Δ_p, key K
Ensure: Reconciled events R

```
Canonicalize T to key K, target buckets, and statistics.
Canonicalize P to the same schema using aggregate counters.
Coalesce duplicate profile rows by key and bucket.
if Δ_a = Δ_p then
    Keep profile buckets unchanged.
else if Δ_a > Δ_p then
    Map profile buckets to enclosing target buckets.
else
    Split profile buckets into sub-buckets of width Δ_a.
    Distribute statistics using bounded interpolation.
end if
Aggregate traces into A_T by K and target bucket.
Aggregate profiles into A_P by K and target bucket.
Drop rows from A_P with keys present in A_T.
R ← aggregate(A_T ∪ A_P) by K and bucket.
return R
```

2) **Efficiency:** execute normalization and merging as embarrassingly parallel operations so analysis can leverage multi-core and multi-node HPC resources.
3) **Extensible outputs:** emit normalized merged events in a schema that can be directly consumed by downstream bottleneck detection, attribution, and visualization workflows.

Figure 2 shows the high-level analysis flow and how ORION integrates with existing monitoring outputs. The analysis tool's Event Loader passes its native traces and profiles to the ORION *Canonicalizer* (Figure 2, arrow 1), which normalizes fields; the *Reconciler* (arrow 2) aligns and resolution-matches them and returns same-timescale events in the tool's schema to its Bottleneck Analyzer (arrow 3), preserving downstream detection, attribution, and visualization.

The **integration point** for analysis is similarly narrow: ORION attaches inside existing performance analysis tools after they load profile/trace artifacts and before they execute downstream analytics. The analysis tool remains responsible for data loading and downstream consumption, while ORION performs canonicalization and mixed-granularity reconciliation and returns events in the tool's expected schema. This keeps adoption practical and enables analysis improvements without replacing existing bottleneck-attribution or visualization pipelines.

1) Reconciling Mixed Granularity Events at Scale: The Reconciler is the core ORION component in the analysis path (Figure 2, between arrows 2 and 3). Its primary role

is to transform heterogeneous profile and trace records into a temporally aligned, resolution-aware representation suitable for bottleneck detection and visualization. The design assumes that analysts require a target resolution that captures macroscopic behavior in workflow context, even when raw event rates are high. For example, although individual I/O events may occur at $\sim$ 10–100 μs, long-running jobs (e.g., 10-hour workflows) still require controlled downsampling and reconciliation to keep the number of analysis points tractable.

Algorithm 1 summarizes the reconciliation operator and realizes the Reconciler shown in Figure 2 for combining profile and trace evidence at a target analysis resolution. The reconciliation key K groups records by dimensions such as target time interval, process, file, category, I/O category, access pattern, and function name; the output R is the reconciled events returned to downstream analysis with aggregate statistics such as count, time, and size. After canonicalization places trace and profile records into a common key and target-bucket schema, the Reconciler handles resolution matching by reusing equal-width buckets, rolling up finer profile buckets, or refining coarser profile buckets through bounded interpolation. Trace records have precedence for overlapping keys and target buckets to avoid double counting; profile evidence is merged only where trace evidence is absent.

Beyond this operator, ORION positions reconciliation as part of a hierarchical analysis workflow. Coarse profile signals can identify candidate bottleneck windows and reduce the search space; within those windows, ORION can attribute contributions from finer-grained application aggregates and available traces. When source granularity is coarser than required analysis granularity (e.g., 1-second profiles needed at 1-millisecond analysis bins), the Reconciler performs constrained interpolation rather than naive expansion.

ORION supports multiple interpolation strategies depending on signal structure: (i) distribution-guided interpolation using summary statistics such as min, max, sum, and count to reconstruct plausible intra-bin behavior, and (ii) trend-guided interpolation using local time-series patterns to preserve directional dynamics across adjacent windows. Following HPC I/O performance-prediction practice [29], ORION treats interpolation as bounded prediction: for coarse bucket value S_p, sub-bucket estimates use distribution/trend weights w_i with $\hat{s}_i = S_p w_i / \sum_j w_j$, preserve $\sum_i \hat{s}_i = S_p$, and stay within profile min/max bounds. Trace ordering, phase boundaries, and known activity intervals further constrain feasible placement; without such constraints, ORION keeps the record coarse rather than inferring unsupported fine-grain structure.

D. Implementation Details

We implement ORION [1] as two complementary plugins: a capture-side plugin to enable workload-centric performance data capture and an analysis-side plugin for post-processing reconciliation, together with a concrete end-to-end integration in an existing toolchain.

[1]Code is merged upstream in dftracer v2.0.2 and dfanalyzer v0.0.9 and the darshan code will be sent upstream soon.

1) Capture Plugin Implementation: The ORION capture plugin is implemented in C++17 using standard data structures, primarily `unordered_map` and fixed/variable-length arrays, keeping per-event operations lightweight and compatible with low-level monitoring runtimes. This follows the implementation ecosystem of widely used HPC I/O capture tools (Darshan, Recorder, DFTracer), which are predominantly C/C++; aligning with this stack keeps integration overhead low and avoids cross-language boundaries in high-frequency event paths.

2) Analysis Plugin Implementation: The ORION analysis plugin is implemented in Python atop a Dask-based analysis engine, matching the software ecosystem of downstream analysis tools such as PyDFTracer, WisIO, Drishti, and DFAnalyzer, which are Python-centric and often dataframe-oriented. Dask parallelizes canonicalization/reconciliation and scales across multi-core and multi-node HPC environments, letting ORION's mixed-granularity pipeline run on distributed compute resources.

3) Integration with Existing Toolchains: As an exemplar integration, we connect ORION to DFTracer on the capture side and DFAnalyzer on the analysis side. We selected DFTracer because it exposes a clean, unified interface for both aggregation and tracing while maintaining low monitoring overhead relative to alternatives such as Score-P, Recorder, and Darshan in our target workloads. On the analysis side, DFAnalyzer already supports Dask-backed distributed dataframes, which provides a natural insertion point for ORION canonicalization/reconciliation and allows us to demonstrate end-to-end impact without restructuring existing downstream analysis stages. Table III summarizes the concrete adapter points for each tool design: for DFTracer/DFAnalyzer and Darshan the integration is implemented and evaluated (Darshan filtering is limited to rank and file name, which disables cross-rank aggregation), while Score-P remains an identified adapter target where the same insertion pattern applies to its substrate-plugin hooks. As a runtime granularity-selection layer rather than a standalone collector, ORION inherits its scaling behavior and failure modes from the underlying tool; its own per-event cost is microsecond-range and memory-bound (Figure 3), adding no failure modes beyond the host tool's collection/storage path.

V. Evaluation

We evaluate ORION in three stages: methodology (hardware, workloads, tools, metrics), internal component benchmarks (aggregation overhead, classifier performance, reconciliation throughput), and real-workload validation on Montage, UNet3D, and Llama-1B.

A. Methodology

Our methodology keeps comparisons reproducible and fair across tools, configurations, and workloads. Claim-to-evidence map: Section V-B1 validates capture overhead and time-granularity tuning, Section V-B2 validates classifier cost, Section V-B3 validates reconciliation throughput and interpolation

TABLE III
CONCRETE ORION ADAPTER POINTS IN I/O-FACING TOOL DESIGNS.

Tool	Tool design hook	ORION insertion/status
DFTracer/-DFAnalyzer [10]	Unified event API; buffered JSONL; Dask analysis load.	Pre-buffer classifier/aggregator; post-load reconciler before WisIO/DFAnalyzer ranking. Implemented/evaluated.
Darshan [7], [30]	Modules update file/rank records; core serializes logs at shutdown.	Wrap module-record updates; reconcile after PyDarshan/log parsing. Implemented/evaluated; filtering limited to rank/file name, no cross-rank aggregation.
Score-P [12]	Substrate plugins receive measurement events beside profile/trace backends.	ORION substrate routes events to trace/aggregate sinks; reconcile after CUBE/OTF2 load. Adapter target.

consistency, and Section V-C validates end-to-end bottleneck fidelity via ordered top-k preservation.

1) Hardware Platform: Our experiments use two LLNL systems to evaluate portability. On Tuolumne [31], the platform comprises 1152 nodes, and we utilize up to 128 nodes for evaluation. Each compute node is equipped with 4 AMD MI300A APUs [32]. Each MI300A APU integrates 24 AMD Zen 4 x86 CPU cores, 228 AMD CDNA 3 GPU compute units, and 128 GB of unified HBM3 memory that presents a single shared address space to both CPU and GPU. Each node has a 4×200 Gb/s HPE Slingshot 11 NIC. Tuolumne provides a global Lustre file system for persistent storage. On Corona [33], the cluster comprises 121 nodes, each with a 48-core AMD Rome processor, 256 GB of memory, 1.5 TB NVMe SSD, and eight AMD MI50 GPUs. Each Corona node has access to the Lustre parallel file system for persistent storage and NFS for user workspace.

2) Software Configuration: Software configurations differ by system. On Tuolumne (TOSS 4), we use AMD compiler `6.2.4`, GCC `12.2`, Python `3.11`, Cray MPICH `8.1.25`, CMake `3.29.2`, and ROCm `6.2.4` for MI300A-enabled workloads. On Corona, we use GCC `11.2.1`, MVAPICH2 `2.3.7`, and Python `3.11`.

3) Workloads: We use a three-tier workload suite. For internal evaluations, we use microbenchmarks to isolate and measure the overhead and performance of Structural Aggregation and Event Classifier logic. For reconciliation evaluation, we use community benchmarks such as DLIO [34], which represents deep learning I/O pipelines with checkpointing and sample loading, H5Bench [35], which represents HDF5-centric simulation I/O with structured array and tensor access, and IOR [36], which represents analytics-style parallel file-system stress under sequential and random phases. For real production-scale validation, we use Montage [17], which represents astronomy workflows with many small and medium files across multi-stage transformations, UNet3D [23] training, which represents 3D scientific AI training with mixed compute and I/O phases, and Llama-1B [37] fine-tuning, which represents LLM adaptation workloads with iterative checkpoint, dataset streaming, and optimizer-state I/O.

4) Performance Tools: We evaluate the demonstrated DFTracer/DFAnalyzer path using DFTracer `v2.0.2`, DFAnalyzer `v0.0.8`, Dask Distributed `2024.1.0`, pandas `2.2.0`, and pyarrow `13`. The concrete adapter points for DFTracer/DFAnalyzer and other I/O-facing tool designs are described in Section IV-D3.

5) Metrics: We report runtime overhead, classifier benchmark runtime, compressed trace size on disk (MB), number of events, and throughput measured as events per second (events/s). Runtime overhead uses $(T_{\text{inst}} - T_{\text{base}})/T_{\text{base}}$; trace size is gzip-compressed JSON Lines output, not raw in-memory event footprint. We additionally report ordered severe-bottleneck preservation as follows.

We define DFTracer *full tracing* as the canonical full-trace reference term used throughout this section. Bottlenecks are detected and severity-ranked using WisIO [38]. `topk_accuracy_pct` is strict ordered preservation of the top-k most severe DFTracer bottlenecks: let $\mathcal{T}_k = \langle t_1, \ldots, t_k \rangle$ be the DFTracer order and $\mathcal{O}_k = \langle o_1, \ldots, o_k \rangle$ the ORION order over aligned candidates. A bottleneck is preserved only if the same candidate appears at the same rank, penalizing both misses and rank changes: I_i=1 if o_i=t_i and 0 otherwise. The preserved count is $\texttt{topk_preserved} = \sum_{i=1}^{k} I_i$ out of $\texttt{topk_k} = |\mathcal{T}_k| = k$, giving the reported percentage $\texttt{topk_accuracy_pct} = 100 \times \texttt{topk_preserved}/\texttt{topk_k}$, the strict ordered preservation rate over the top-k most severe bottlenecks. Throughout our evaluations, k=10.

6) Baseline rationale: DFTracer full tracing is our reference because it captures all user-space I/O events without aggregation and, per our motivation study, provides the highest-fidelity bottleneck baseline at lower overhead than Score-P full tracing. ORION's goal is to reduce collected data while preserving the DFTracer/WisIO severe-bottleneck order; this validity is limited to user-space I/O bottlenecks and excludes kernel, network-fabric, or storage-controller contention diagnostics.

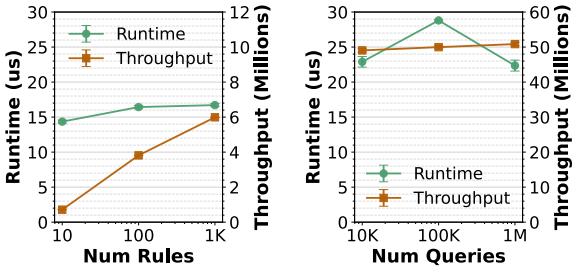
B. Internal Evaluations

1) Overhead of Structural Aggregation (Time and Space): The goal of this experiment is to quantify Structural Aggregation time/space overhead and test the $\alpha \in [2, 8]$ tuning heuristic (Section IV-B). We use a microbenchmark with three aggregation dimensions (time, process, file), 32 requests of 1 MB per process on 4 nodes, and mean per-request duration $d_e \approx 12.5$ ms, so the recommended range is $\Delta t \in [25\,\text{ms}, 100\,\text{ms}]$. We evaluate 10 ms ($\alpha \approx 0.8$), 100 ms ($\alpha \approx 8$), 1 s ($\alpha \approx 80$), and 30 s ($\alpha \approx 2400$), covering the upper endpoint and under/over-aggregation cases.

Table IV reports conventional runtime overhead using the mean baseline runtime. Baseline runtime is 3.154 s. The 10 ms window is under-aggregated and increases runtime to 3.551 s (12.60% overhead). At 100 ms, runtime is 3.218 s (2.05% overhead), while coarser 1 s and 30 s windows are 3.181 s (0.87%) and 3.171 s (0.56%), respectively.

TABLE IV
TIME-SPACE OVERHEAD TRADE-OFF FOR 32×1 MB REQUESTS PER PROCESS ON 4 NODES ($d_e \approx 12.5$ MS, $\alpha \in [2, 8]$; SECTION V-B1).

Tool	Configuration	Events	Size (MB)	OH (%)
DFTracer	10 ms ($\alpha \approx 0.8$)	145K	6.754	12.60
ORION	100 ms ($\alpha \approx 8$)	53K	0.230	2.05
ORION	1 s ($\alpha \approx 80$)	36K	0.242	0.87
ORION	30 s ($\alpha \approx 2400$)	34K	0.233	0.56
Darshan DXT	all events	15.4K	7.56	15.00
Darshan Profile	profile agg. 96×4	220	0.15	3.20
ORION w/ Darshan	selective	2.31K	2.25	5.20



(a) Rule Adding Cost: runtime and throughput vs number of rules.

(b) Rule Querying Cost: runtime and throughput vs # queries (1000 rules).

Fig. 3. ORION's Event Classifier benchmark shows stable batched query runtime and sustains ∼50 M events/s throughput across query volumes with 1000 rules.

The results support the $\alpha \in [2, 8]$ heuristic by showing the expected boundary behavior. At 10 ms ($\alpha \approx 0.8$), ORION shows highest overhead and largest output (145K events, 6.754 MB), indicating under-aggregation. At 100 ms ($\alpha \approx 8$), ORION reaches 53K events and 0.230 MB. At 1 s and 30 s ($\alpha \approx 80$ and 2400), events drop further, but size gains are marginal (0.242 MB and 0.233 MB), indicating over-coarsening. Results represent the mean across 5 repeated runs; d_e should be re-estimated when workload phase structure or scale changes. Table IV (bottom three rows) extends this comparison to Darshan: full-fidelity DXT tracing and file-only profiling bracket ORION selective tracing of the critical access window, which lands between Darshan's two fixed operating points while retaining fine-grained attribution where it matters. This table quantifies cost; fidelity across α comes from the sweep in Section IV-B (stable top-k for $\alpha \in [2, 8]$, $-25\%/-29\%$ at α=16/32).

2) Performance and Cost of Event Classifier: The primary objective of this experiment is to verify that Event Classifier execution does not become a runtime bottleneck. We use a synthetic benchmark with two phases. In the first phase (rule-adding), we measure the one-time initialization cost of inserting rules while increasing the number of rules (10, 100, 1000). In the second phase (rule-querying), we fix the classifier at 1000 rules and vary the number of classification queries (10K, 100K, 1M) to evaluate steady-state throughput when each event is evaluated against the rule set. Figure 3 reports average runtime and throughput on dual y-axes, with standard-deviation whiskers.

The rule-adding trend shows that initialization runtime remains nearly flat as rule count increases (14.35–16.71 μs for 10–1000 rules). The rule-querying benchmark fixes 1000 rules and executes batches of 10K, 100K, and 1M queries. The 22–29 μs values are batch runtimes for these query loops, not per-event latency; measured throughput is stable at 49.0M–50.9M events/s, or roughly 20 ns amortized per query in this synthetic benchmark. This isolates rule-evaluation overhead and excludes tool callback, timestamping, serialization, and file-system costs.

From a system perspective, these results indicate that Event Classification initialization and evaluation are memory-bound: increasing rule or query volume does not introduce runaway compute cost, and throughput remains high and stable at scale. Consequently, the impact on application performance is expected to be minimal, and ORION can keep classification in the online path without significantly perturbing application progress.

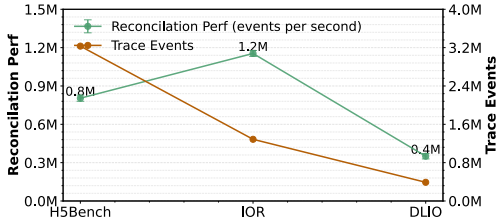


Fig. 4. ORION reconciliation scales across workloads: higher-event workloads (H5Bench and IOR) sustain substantially higher reconciliation throughput than lower-event DLIO, indicating efficient profile+trace integration at larger event volumes.

3) Reconciliation Performance: The goal of this experiment is to quantify reconciliation throughput across diverse workload classes using the benchmarks from Section V-A (H5Bench, IOR, DLIO) to evaluate efficiency over different profile/trace mixtures.

Figure 4 summarizes reconciliation performance across the three workloads. The x-axis lists the workload, the left y-axis reports reconciliation throughput in events per second, and the right y-axis shows trace-event volume to provide input-size context. For all benchmarks, data is collected at 5 s profile granularity and reconciled at a 1 s target analysis resolution. This setting exercises the coarser-profile/finer-target case in Algorithm 1, where profile buckets are refined into target-width sub-buckets using bounded interpolation before being merged with trace evidence.

For this evaluation, ORION reconciles profile/counter records (C) and trace records (X). H5Bench reconciles C=59K and X=3.13M, with throughput around ∼0.80M events/s. IOR reconciles C=348K and X=896K, with the highest throughput around ∼1.15M events/s. DLIO-UNet3D reconciles C=28K and X=360K, with reconciliation throughput around ∼0.35M events/s.

These results show ORION sustains high reconciliation throughput while integrating profile/trace inputs across workload classes and granularities. Relative to DLIO, IOR and H5Bench process larger reconciliation workloads at higher throughput, indicating the Dask-backed path amortizes over-

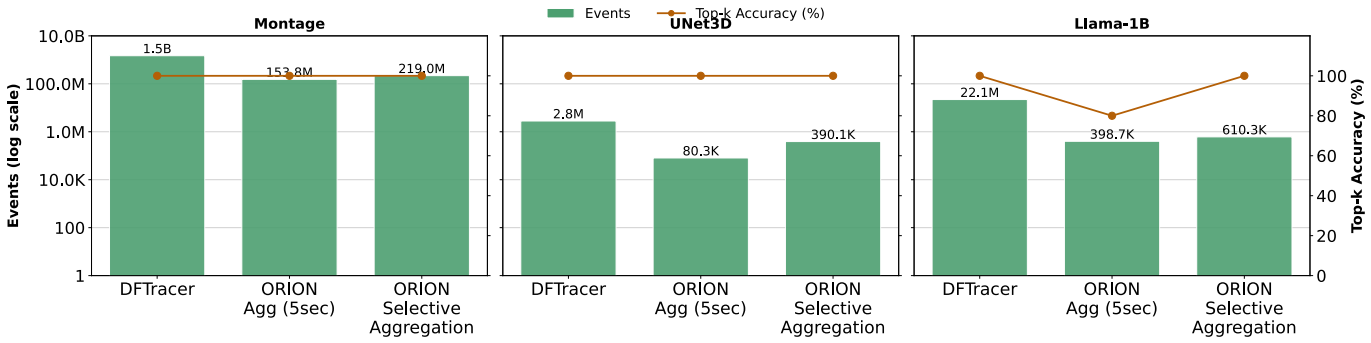


Fig. 5. Combined real-workload comparison for Montage, UNet3D, and Llama-1B. Bars show event volume; points show ordered top-k severe-bottleneck accuracy. Event reduction is computed as $\texttt{events}_{\texttt{DFTracer}}/\texttt{events}_{\texttt{mode}}$.

head at larger event volumes while remaining efficient for smaller deep-learning traces.

As a direct interpolation-quality signal, we report interpolation consistency at top-k (IC@k): agreement between rankings from native finer-grain data and coarser-to-finer reconciliation. At k=10, selective-mode IC@10 is 100% (DLIO), 87.6% (IOR), and 100% (H5Bench); IOR's lower consistency reflects its bursty, highly parallel MPI-IO pattern, where short high-intensity bursts create sharp phase boundaries harder to reconstruct from coarser 5 s buckets. The remaining 12.4% gap does not affect real-workload top-k results, which use trace-level evidence directly.

This subsection measures throughput; reconciler accuracy is validated on real workloads (Section V-C) via ordered top-k preservation, an end-to-end fidelity check rather than a direct interpolation-error metric.

C. Real Scientific Workloads

We evaluate ORION on Montage, UNet3D, and Llama-1B. Table V reports overhead from 5 Montage makespan runs and 20 AI-training epoch samples; ORION stays low single-digit (max 3.0%), with small negative values indicating statistically indistinguishable runs. This section is also the Reconciler accuracy evaluation: ordered top-k preservation (Section V-A5) tests severe-bottleneck rank fidelity.

1) Montage: Montage processes 2MASS [39] FITS data through reprojection and mosaicking, generating many short file-level operations. We run Montage on 32 Corona nodes with 1024 processes over 5 iterations for a 2.5 GB dataset. ORION targets short stdio/POSIX-style calls that inflate trace size but add limited individual bottleneck value. Events drop from ∼1.48B (DFTracer) to ∼153.8M under Agg (5 s) (∼9.6×) and ∼219.0M under Selective (∼6.8×); both preserve 10/10 ordered top-k bottlenecks. Overhead is 3.0% (Agg) and 1.5% (Selective); Agg costs more, hashing many small-file events into per-key summaries.

2) UNet3D AI Training: UNet3D represents 3D medical AI training with mixed compute and I/O behavior, including data-loader activity and checkpoint operations over NPZ-based inputs. We run an 8×-augmented UNet3D pipeline on 32 Tuolumne nodes (128 MI300A APUs), with one worker per GPU process, for 20 epochs. ORION aggregates low-level POSIX calls while preserving framework-level data-loader and checkpoint context. Events drop from ∼2.82M (DFTracer) to ∼80.3K under ORION Aggregation (5 s) (∼35× reduction) and to ∼390.1K under ORION Selective Aggregation (∼7.2× reduction), while both preserve 10/10 ordered top-k bottlenecks. The larger reduction under ORION Aggregation reflects many small metadata events, primarily NumPy stat calls.

3) Llama-1B LLM Fine-Tuning: Llama-1B fine-tuning is a computation-dominated large-scale LLM workload where tracing overhead can become expensive in both time and storage volume. We run 20 epochs on 128 Tuolumne nodes (512 MI300A APUs) using DeepSpeed (ZeRO-0) [40], one worker per process, with the 1.1 GB pretokenized OpenHermes-2.5 [41] dataset. Events drop from ∼22.1M (DFTracer) to ∼398.7K under ORION Aggregation (5 s) (∼55× reduction, 8/10 ordered top-k preserved) and to ∼610.3K under ORION Selective Aggregation (∼36× reduction, 10/10 preserved). ORION Aggregation misses 2 of the top-10 bottlenecks because 5 s bins merge short checkpoint and optimizer-state bursts near epoch boundaries; ORION Selective preserves these classes, keeping the actionable diagnosis: checkpoint/optimizer-state I/O is the dominant storage-side bottleneck, not dataset streaming.

4) Workload Analysis: Table V shows that Agg gives highest compression, while Selective gives the best fidelity–cost balance. Selective reaches 10/10 ordered top-k accuracy on all workloads with ∼6.8×–∼36× event reduction; for preserved top-k cases, ORION retains the same file/function/category attribution used by WisIO/DFAnalyzer. Montage compressed traces drop from 13.11 GB (DFTracer) to 2.18 GB (Agg) and 2.74 GB (Selective). Top-k accuracy has zero variance. Beyond ordered top-k, ORION covers WisIO's full bottleneck sets (25 Montage, 12 UNet3D, 17 Llama-1B) at 98%/100%/95% accuracy and preserves severity, not just rank: Montage metadata-access latency stays within 3% of the full DFTracer trace, and Llama-1B checkpoint-write throughput within 5%. At matched 10% profiled-event budgets, workload-agnostic baselines underperform: random sampling reaches 72%/75%/35% top-k accuracy, uniform 61%/62%/30% (Montage/UNet3D/Llama-1B), both degrading because they reduce granularity for high-impact regions (small metadata files in Montage, large checkpoints in Llama) ORION's workload-aware policy preserves.

TABLE V
REAL-WORKLOAD EVENT REDUCTION, RUNTIME OVERHEAD (OH), AND ORDERED TOP-k SEVERE-BOTTLENECK PRESERVATION. EACH MODE CELL IS EVENTS/OH/PRESERVED-TOP-k; OH USES MEAN RUNTIME VERSUS NO-DFTRACER.

Workload	DFT OH	Agg: events/OH	Selective: events/OH
Montage	1.1%	153.8M/3.0%	219.0M/1.5%
UNet3D	-0.2%	80.3K/0.4%	390.1K/-0.2%/10/10
Llama-1B	0.0%	398.7K/0.9%	610.3K/-0.1%

VI. RELATED WORK

Prior HPC I/O observability work spans summary-oriented profiling, event-oriented tracing, and hybrid tool-chains operating in both modes. Lightweight profilers such as Darshan and TAU are widely used for trend analysis and production deployment at scale [42], [43]. Tracers such as Recorder, Pilgrim, and ScalaIOTrace retain fine-grained ordering and timing information needed to diagnose burstiness, contention and cross-layer interactions [9], [13], [14]. Frameworks such as Score-P and Scalasca support both summary-based and trace-based analysis within a common measurement ecosystem [12], [15]. However, these systems are still typically operated in one specific collection mode per run, leaving users to choose between coarse-grained summaries and high-volume traces rather than adapt collection online to workload needs.

A second line of work expands observability across layers and operational contexts. TOKIO integrates signals from existing monitoring tools to provide a more holistic view of end-to-end I/O behavior across the stack [44]. EZIOTracer unifies kernel-space and user-space tracing for data-intensive applications [45]. DFTracer targets workflow-aware, multi-level data-flow tracing for AI-driven workflows, while PROV-IO emphasizes provenance-centric views of scientific data and I/O activity [10], [46]. XIO further highlights the need for explainable I/O analysis by explicitly exposing causal relationships across user-level and kernel-level paths [28]. These systems improve visibility across different software layers and operational contexts, but generally focus on a particular analysis objective rather than providing a unified online mechanism to switch between summary and detailed views during collection.

Recent work has also explored data-driven diagnosis and workload-aware optimization. SmartIO demonstrates that runtime I/O optimization can be driven online without exhaustive offline training or expensive online search [2]. ML-oriented performance workflows automate model building from collected performance data [1]. At the workflow level, the community has increasingly recognized that modern HPC applications are often coupled, data-intensive workflows rather than isolated monolithic executables [3]. Even in these settings, multidimensional aggregation, event selection, and mixed-granularity profiling are still commonly treated as post-hoc analysis tasks rather than as primary controls during data capture.

In this landscape, ORION targets three gaps. First, existing approaches that exploit dimensions such as time, process, file, and workflow context largely use them offline, not as online workload-centric controls during event capture. Second, most tools emphasize either profile-style summaries or trace-style detail, but not both simultaneously with explicit user-controlled switching and shared buffering semantics. Third, current tooling provides limited support for rigorous trace-profile reconciliation, especially when mixed-granularity data must be integrated in a resolution-aware manner rather than inspected side by side.ORION is a plugin framework, demonstrated and implemented with DFTracer/DFAnalyzer and Darshan, and adapter-ready for the I/O-facing Score-P path, rather than a competing tool. HPCToolkit is broader performance-analysis context, not an ORION I/O adapter target, so a direct top-k comparison against Score-P or Darshan is not meaningful. Section III shows their profiling modes yield similar event-volume/overhead profiles, while DFTracer full tracing is the highest-fidelity user-space baseline among them and thus a conservative reference for bottleneck-preservation evaluation. ORION's novelty over Score-P is in *when* and *how* granularity is chosen: Score-P selects instrumented functions at compile time and profiles/traces them uniformly, while ORION performs per-region (time, process, file) granularity control at runtime, changing trace/aggregate assignments dynamically without recompilation.

VII. CONCLUSIONS

ORION, a plugin framework demonstrated with DFTracer/DFAnalyzer and Darshan and designed for the I/O-facing Score-P adapter path, shows that monitoring granularity need not be fixed per run. We draw four lessons. (1) Aggregation windows should be a small multiple of the target event: at $\alpha \approx 8$, overhead falls from 12.6% to 2.05% and trace size $29\times$, while $\alpha \geq 16$ loses 25–29% top-k accuracy. (2) Rule-based trace/aggregate routing ($\sim$50 M events/s at 1000 rules) is cheap enough for the online path. (3) Trace-precedence reconciliation (0.35–1.15 M events/s) lets existing analyzers run unchanged. (4) Workload-aware selection preserves bottleneck ranking where coarsening and sampling fail: Selective keeps 10/10 top-k on all workloads with $6.8\times$–$36\times$ fewer events and $\leq$1.5% overhead, while 5 s aggregation drops to 8/10 and random sampling to 35% on Llama-1B, whose checkpoint bursts need finer granularity. ORION suits workflows where full tracing is too costly and users know phase, file, or category structure. Future work: policy adaptation, GPU/network-telemetry reconciliation, and broader portability.

ACKNOWLEDGMENT

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-CONF-2018940). This material is based upon work supported by LLNL LDRD 25-ERD-042.

References

[1] Z. Zhu, C. Wang, and S. Neuwirth, "Advancing HPC Performance Modeling with an Interactive, Automated and Tool-Agnostic ML-Driven Workflow," in *Proceedings of the 37th International Conference on Scalable Scientific Data Management*, ser. SSDBM '25. New York, NY, USA: Association for Computing Machinery, Jun. 2025, pp. 1–6. [Online]. Available: https://dl.acm.org/doi/10.1145/3733723.3733740

[2] H. B. Ather, C. Wang, H. Devarajan, H. Childs, and K. Mohror, "SmartIO: A Lightweight End-to-End Workflow for Runtime I/O Optimization of HPC Systems," in *Proceedings of the SC '25 Workshops of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC Workshops '25. New York, NY, USA: Association for Computing Machinery, Nov. 2025, pp. 1396–1405. [Online]. Available: https://dl.acm.org/doi/10.1145/3731599.3767509

[3] T. Ben-Nun, T. Gamblin, D. S. Hollman, H. Krishnan, and C. J. Newburn, "Workflows are the New Applications: Challenges in Performance, Portability, and Productivity," in *2020 IEEE/ACM International Workshop on Performance, Portability and Productivity in HPC (P3HPC)*, Nov. 2020, pp. 57–69. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9309042

[4] H. Devarajan and K. Mohror, "Extracting and characterizing I/O behavior of HPC workloads." Heidelberg, Germany: IEEE, Sep. 2022.

[5] V. Meza and A. Antonio, "Scheduling strategies for efficient data movement in data-intensive scientific workflow execution," Jan. 2026. [Online]. Available: https://hdl.handle.net/1992/77981

[6] K. Loi, C. Song, H. Chao, L. Chen, and A. Guo, "High temporal-resolution HVAC control in GPU-centric data centers via reinforcement learning and a knowledge-embedded physics-informed neural network," *Journal of Building Engineering*, vol. 123, p. 115499, Apr. 2026. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2352710226003207

[7] H. Shan, K. Antypas, and J. Shalf, "Characterizing and predicting the I/O performance of HPC applications using a parameterized synthetic benchmark," in *SC '08: Proceedings of the 2008 ACM/IEEE Conference on Supercomputing*, Nov. 2008, pp. 1–12.

[8] A. Munera, S. Royuela, G. Llort, E. Mercadal, F. Wartel, and E. Quiñones, "Experiences on the characterization of parallel applications in embedded systems with Extrae/Paraver," in *Proceedings of the 49th International Conference on Parallel Processing*, ser. ICPP '20. New York, NY, USA: Association for Computing Machinery, Aug. 2020, pp. 1–11. [Online]. Available: https://dl.acm.org/doi/10.1145/3404397.3404440

[9] C. Wang, J. Sun, M. Snir, K. Mohror, and E. Gonsiorowski, "Recorder 2.0: Efficient Parallel I/O Tracing and Analysis," in *2020 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, May 2020, pp. 1–8.

[10] H. Devarajan, L. Pottier, K. Velusamy, H. Zheng, I. Yildirim, O. Kogiou, W. Yu, A. Kougkas, X.-H. Sun, J. S. Yeom, and K. Mohror, "DFTracer: An Analysis-Friendly Data Flow Tracer for AI-Driven Workflows," in *SC24: International Conference for High Performance Computing, Networking, Storage and Analysis*. Atlanta, GA: IEEE, Jun. 2024.

[11] NERSC, "Darshan I/O Profiler - NERSC Documentation," Jul. 2024. [Online]. Available: https://docs.nersc.gov/tools/performance/darshan/

[12] A. Knüpfer, C. Rössel, D. a. Mey, S. Biersdorff, K. Diethelm, D. Eschweiler, M. Geimer, M. Gerndt, D. Lorenz, A. Malony, W. E. Nagel, Y. Oleynik, P. Philippen, P. Saviankou, D. Schmidl, S. Shende, R. Tschüter, M. Wagner, B. Wesarg, and F. Wolf, "Score-P: A Joint Performance Measurement Run-Time Infrastructure for Periscope,Scalasca, TAU, and Vampir," in *Tools for High Performance Computing 2011*, H. Brunst, M. S. Müller, W. E. Nagel, and M. M. Resch, Eds. Berlin, Heidelberg: Springer, 2012, pp. 79–91.

[13] C. Wang, P. Balaji, and M. Snir, "Pilgrim: scalable and (near) lossless MPI tracing," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '21. New York, NY, USA: Association for Computing Machinery, Nov. 2021, pp. 1–14. [Online]. Available: https://dl.acm.org/doi/10.1145/3458817.3476151

[14] K. Vijayakumar, F. Mueller, X. Ma, and P. C. Roth, "Scalable I/O tracing and analysis," in *Proceedings of the 4th Annual Workshop on Petascale Data Storage*, ser. PDSW '09. New York, NY, USA: Association for Computing Machinery, Nov. 2009, pp. 26–31. [Online]. Available: https://dl.acm.org/doi/10.1145/1713072.1713080

[15] M. Geimer, F. Wolf, B. J. N. Wylie, E. Ábrahám, D. Becker, and B. Mohr, "The Scalasca performance toolset architecture," *Concurrency and Computation: Practice and Experience*, vol. 22, no. 6, pp. 702–719, 2010, _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/cpe.1556. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.1556

[16] L. Adhianto, J. Anderson, R. M. Barnett, D. Grbic, V. Indic, M. Krentel, Y. Liu, S. Milaković, W. Phan, and J. Mellor-Crummey, "Refining HPCToolkit for application performance analysis at exascale," *The International Journal of High Performance Computing Applications*, vol. 38, no. 6, pp. 612–632, Nov. 2024. [Online]. Available: https://doi.org/10.1177/10943420241277839

[17] M. Rynge, G. Juve, J. Kinney, J. Good, B. Berriman, A. Merrihew, and E. Deelman, "Producing an Infrared Multiwavelength Galactic Plane Atlas Using Montage, Pegasus, and Amazon Web Services," *Annual Astronomical Data Analysis Software and Systems (ADASS)*, vol. 485, p. 211, May 2014, conference Name: Astronomical Data Analysis Software and Systems XXIII ADS Bibcode: 2014ASPC..485..211R. [Online]. Available: https://ui.adsabs.harvard.edu/abs/2014ASPC..485..211R

[18] L. Schielicke, Y. Li, J. Schyns, A. Sperschneider, J. P. Solano Marchini, and C. P. Gatzen, "Meeting summary: Exploring cloud dynamics with Cloud Model 1 and 3D visualization – insights from a university modeling workshop," *Weather and Climate Dynamics*, vol. 5, no. 2, pp. 703–710, May 2024. [Online]. Available: https://wcd.copernicus.org/articles/5/703/2024/

[19] N. P. Tan, R. B. F. Bird, G. Chen, S. V. Luedtke, B. J. Albright, and M. Taufer, "Analysis of Vector Particle-In-Cell (VPIC) memory usage optimizations on cutting-edge computer architectures," *Journal of Computational Science*, vol. 60, Jan. 2022, institution: Los Alamos National Lab. (LANL), Los Alamos, NM (United States) Number: LA-UR-21-31487. [Online]. Available: https://www.osti.gov/pages/biblio/1844166

[20] M. C. Dani, H. Doreau, and S. Alt, "K-means Application for Anomaly Detection and Log Classification in HPC," in *Advances in Artificial Intelligence: From Theory to Practice*, S. Benferhat, K. Tabia, and M. Ali, Eds. Cham: Springer International Publishing, 2017, pp. 201–210.

[21] M. M. A. Patwary, S. Byna, N. R. Satish, N. Sundaram, Z. Lukić, V. Roytershteyn, M. J. Anderson, Y. Yao, Prabhat, and P. Dubey, "BD-CATS: big data clustering at trillion particle scale," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '15. New York, NY, USA: Association for Computing Machinery, Nov. 2015, pp. 1–12. [Online]. Available: https://dl.acm.org/doi/10.1145/2807591.2807616

[22] A. Mathuriya, D. Bard, P. Mendygral, L. Meadows, J. Arnemann, L. Shao, S. He, T. Kärnä, D. Moise, S. J. Pennycook, K. Maschhoff, J. Sewall, N. Kumar, S. Ho, M. F. Ringenburg, P. Prabhat, and V. Lee, "CosmoFlow: Using Deep Learning to Learn the Universe at Scale," in *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis*, Nov. 2018, pp. 819–829. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8665771

[23] F. Turk, M. Luy, and N. Barisci, "Comparison of Unet3D Models for Kidney Tumor Segmentation," Jan. 2020. [Online]. Available: https://www.preprints.org/manuscript/202001.0314/v1

[24] Z. Li, S. He, P. Chaturvedi, V. Kindratenko, E. A. Huerta, K. Kim, and R. Madduri, "Secure Federated Learning Across Heterogeneous Cloud and High-Performance Computing Resources: A Case Study on Federated Fine-Tuning of LLaMA 2," *Computing in Science & Engineering*, vol. 26, no. 3, pp. 52–58, Jul. 2024. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/10485269

[25] J. Y. Moon, F. Di Natale, H. I. Ingolfsson, H. Bhatia, and J. R. Chavez, "Multiscale Machine-Learned Modeling Infrastructure RAS," Lawrence Livermore National Lab. (LLNL), Livermore, CA (United States), Tech. Rep. MuMMI RAS, Sep. 2021.

[26] N. Siva, "1000 genomes project," *Nature Biotechnology*, vol. 26, no. 3, pp. 256–257, Mar. 2008. [Online]. Available: https://github.com/pegasus-isi/1000genome-workflow

[27] J. J. Williams, D. Medeiros, I. B. Peng, and S. Markidis, "Characterizing the Performance of the Implicit Massively Parallel Particle-in-Cell iPIC3D Code," Aug. 2024, arXiv:2408.01983 [physics]. [Online]. Available: http://arxiv.org/abs/2408.01983

[28] S. Neuwirth, H. Devarajan, C. Wang, and J. Lofstead, "XIO: Toward eXplainable I/O for HPC Systems," in *Proceedings of the 37th International Conference on Scalable Scientific Data Management*, ser. SSDBM '25. New York, NY, USA: Association

for Computing Machinery, Jun. 2025, pp. 1–6. [Online]. Available: https://dl.acm.org/doi/10.1145/3733723.3733741

[29] M. Agarwal, D. Singhvi, P. Malakar, and S. Byna, "Active Learning-based Automatic Tuning and Prediction of Parallel I/O Performance," in *2019 IEEE/ACM Fourth International Parallel Data Systems Workshop (PDSW)*, Nov. 2019, pp. 20–29.

[30] "darshan-hpc/darshan," Sep. 2024, original-date: 2021-05-12T21:21:37Z. [Online]. Available: https://github.com/darshan-hpc/darshan

[31] "Tuolumne | HPC @ LLNL." [Online]. Available: https://hpc.llnl.gov/hardware/compute-platforms/tuolumne

[32] A. Smith, G. H. Loh, M. J. Schulte, M. Ignatowski, S. Naffziger, M. Mantor, M. F. N. Kalyanasundharam, V. Alla, N. Malaya, J. L. Greathouse, E. Chapman, and R. Swaminathan, "Realizing the AMD Exascale Heterogeneous Processor Vision : Industry Product," in *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*, Jun. 2024, pp. 876–889. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/10609636

[33] "Corona | HPC @ LLNL." [Online]. Available: https://hpc.llnl.gov/hardware/compute-platforms/corona

[34] H. Devarajan, H. Zheng, A. Kougkas, X.-H. Sun, and V. Vishwanath, "DLIO: A Data-Centric Benchmark for Scientific Deep Learning Applications," in *2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*, May 2021, pp. 81–91.

[35] T. Li, S. Byna, Q. Koziol, H. Tang, J. L. Bez, and Q. Kang, "h5bench: a Parallel I/O Benchmark Suite for HDF5," Apr. 2023, original-date: 2020-10-28T18:31:02Z. [Online]. Available: https://github.com/hpc-io/h5bench

[36] "hpc/ior," Jul. 2025, original-date: 2015-06-23T15:56:39Z. [Online]. Available: https://github.com/hpc/ior

[37] A. Grattafiori, A. Dubey, A. Jauhri, A. Pandey *et al.*, "The Llama 3 Herd of Models," arXiv preprint arXiv:2407.21783, 2024.

[38] I. Yildirim, H. Devarajan, A. Kougkas, X.-H. Sun, and K. Mohror, "WisIO: Automated I/O Bottleneck Detection with Multi-Perspective Views for HPC Workflows," in *ICS '25: 2025 International Conference on Supercomputing*, Salt Lake City, UT, USA, Jun. 2025.

[39] M. F. Skrutskie, R. M. Cutri, R. Stiening, M. D. Weinberg, S. Schneider, J. M. Carpenter, C. Beichman, R. Capps, T. Chester, J. Elias *et al.*, "The Two Micron All Sky Survey (2MASS)," *The Astronomical Journal*, vol. 131, no. 2, pp. 1163–1183, 2006.

[40] S. Rajbhandari, J. Rasley, O. Ruwase, and Y. He, "ZeRO: Memory Optimizations Toward Training Trillion Parameter Models," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC '20)*, 2020, pp. 1–16.

[41] Teknium, "OpenHermes 2.5: An Open Dataset of Synthetic Data for Generalist LLM Assistants," Hugging Face, 2023. [Online]. Available: https://huggingface.co/datasets/teknium/OpenHermes-2.5

[42] S. Snyder, P. Carns, K. Harms, R. Ross, G. K. Lockwood, and N. J. Wright, "Modular HPC I/O Characterization with Darshan," in *2016 5th Workshop on Extreme-Scale Programming Tools (ESPT)*, Nov. 2016, pp. 9–17.

[43] S. Shende, A. D. Malony, W. Spear, and K. Schuchardt, "Characterizing I/O Performance Using the TAU Performance System," *Applications, Tools and Techniques on the Road to Exascale Computing*, pp. 647–655, 2012. [Online]. Available: https://ebooks.iospress.nl/doi/10.3233/978-1-61499-041-3-647

[44] G. K. Lockwood, N. J. Wright, S. Snyder, P. Carns, G. Brown, and K. Harms, "TOKIO on ClusterStor: Connecting Standard Tools to Enable Holistic I/O Performance Analysis," 2018. [Online]. Available: https://escholarship.org/uc/item/8j14j182

[45] M. I. Naas, F. Trahay, A. Colin, P. Olivier, S. Rubini, F. Singhoff, and J. Boukhobza, "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*, 2021, pp. 1–11.

[46] R. Han, S. Byna, H. Tang, B. Dong, and M. Zheng, "PROV-IO: An I/O-Centric Provenance Framework for Scientific Data on HPC Systems," in *Proceedings of the 31st International Symposium on High-Performance Parallel and Distributed Computing*, ser. HPDC '22. New York, NY, USA: Association for Computing Machinery, Jun. 2022, pp. 213–226. [Online]. Available: https://dl.acm.org/doi/10.1145/3502181.3531477

[47] "XIO: Toward eXplainable I/O for HPC Systems | Proceedings of the 37th International Conference on Scalable Scientific Data Management." [Online]. Available: https://dl.acm.org/doi/10.1145/3733723.3733741