LPS

Lightweight Provenance Service for High-Performance Computing

Dong Dai*, Yong Chen, Philip Carns, John Jenkins, and Robert Ross

In general, provenance is documented history of an object

Little Dancer Aged Fourteen

1. Degas, Edgar (*created1878-1881*)
2. René De Gas (*heritage 1917*)
3. Adrien-Aurélien Hébrard (*a contract 5/3/1918)*
4. Nelly Hébrard (*heritage 1937)*
5. M. Knoedler & Company, Inc. *(cosigned 1955)*
6. Paul Mellon *(purchased 5/1956)*
7. National Gallery of Art (*bequest 1999*).

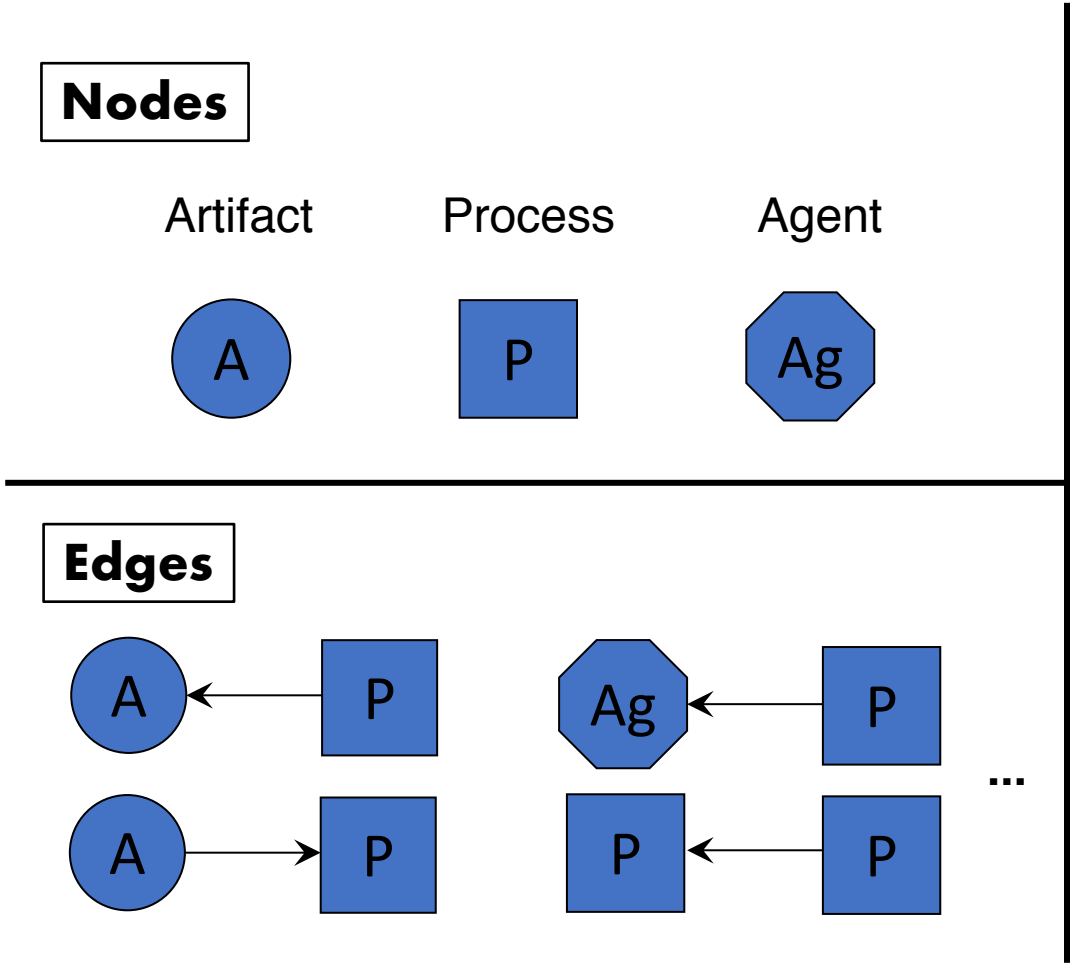
- From National Gallery of Art website

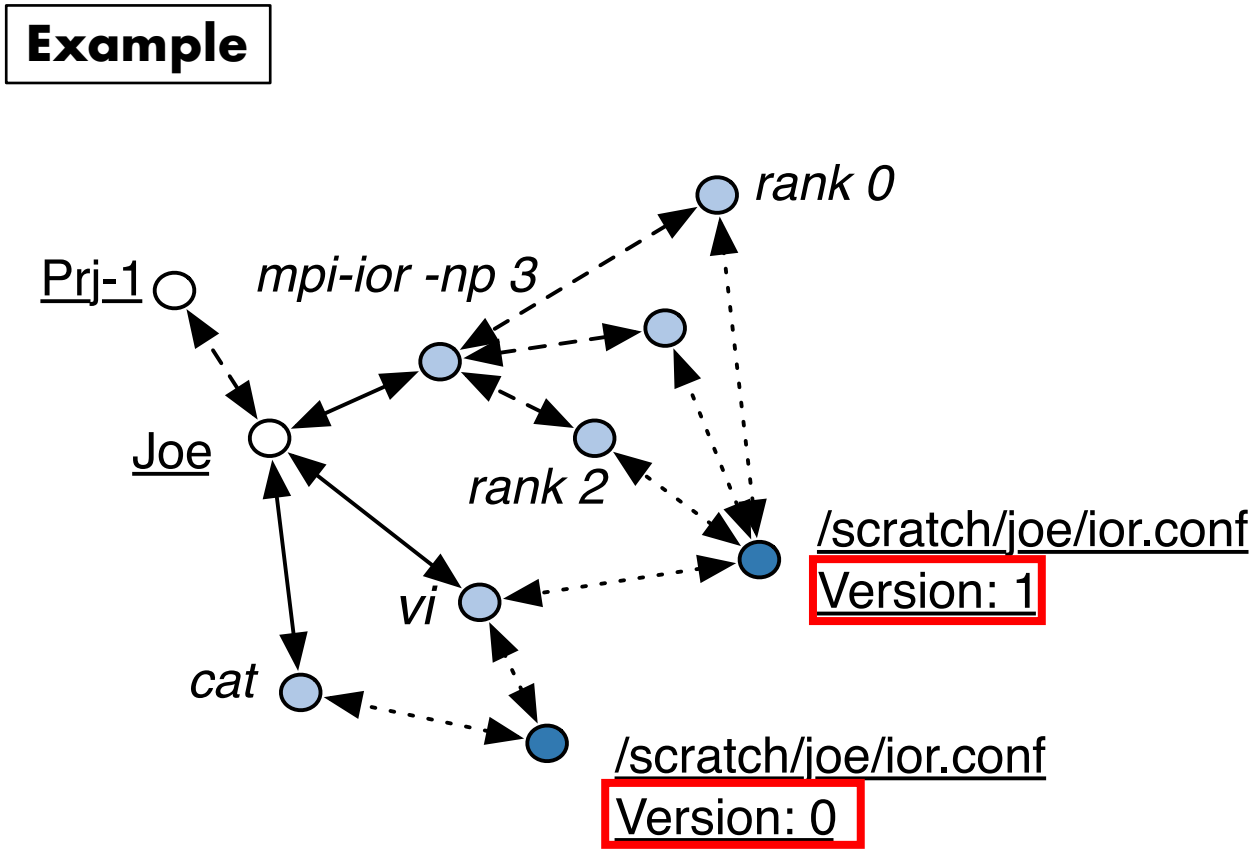
In computer science, provenance means the lineage of data, including

- processes that act on data
- agents that are responsible for those processes.

Open Provenance Model (OPM)

- a graph-based provenance representation model

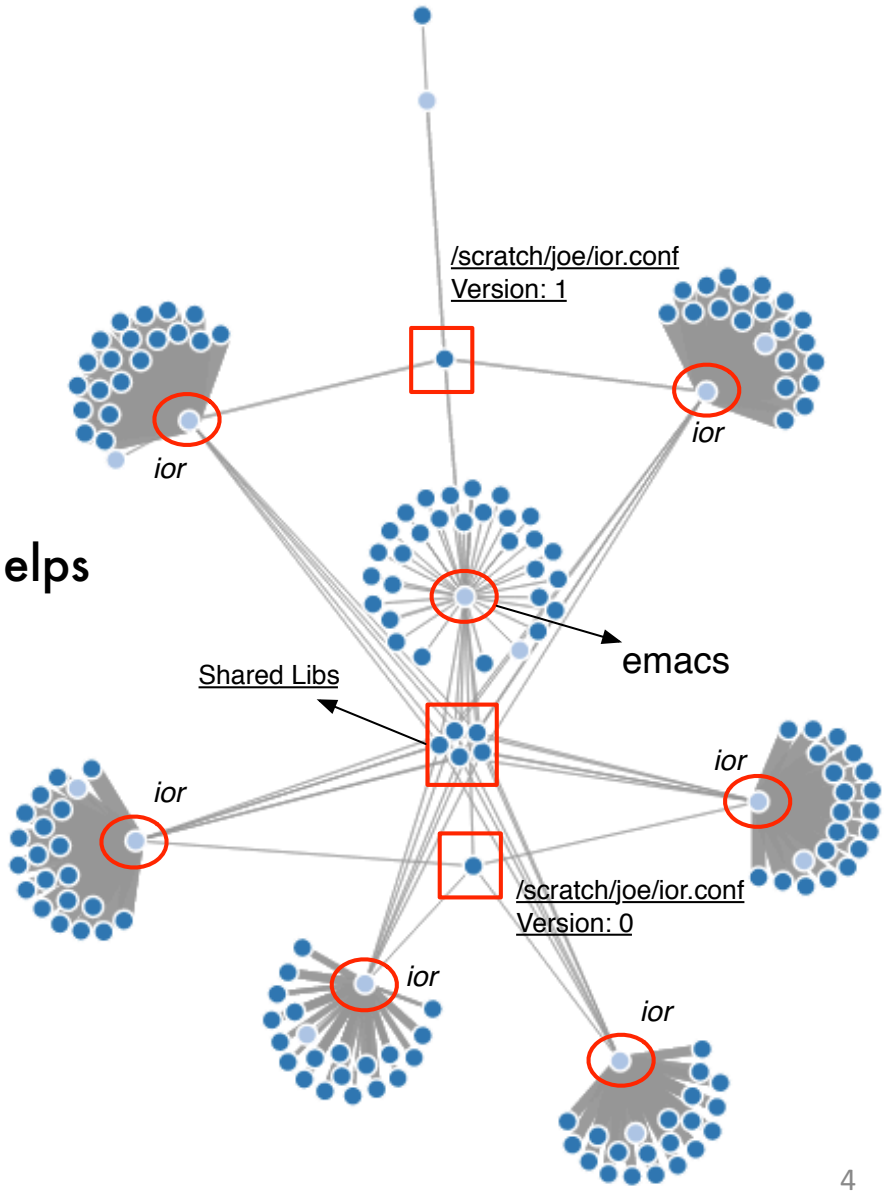




HPC provenance is useful in the simulate-analyze-publish science discovery cycle

- Evaluate a new system
  - repeatedly run the same benchmark (typically time consuming)
  - calculate avg and std for comparing
- Questions
  - If unexpected variations occur, how to ensure they are **from your system** or from **your evaluations**?
  - Can other easily repeat the same evaluations?
  - …

Provenance Helps



Requirements on managing provenance in HPC

- **Performance Requirements:**
  - HPC users are performance sensitive.
  - Managing overhead should be less than 1% slowdown and less than 1MB memory footprint per core.
- **Coverage Requirements:**
  - Provenance generated from multiple physical locations.
  - Provenance could have various granularities.
- **Transparency Requirements:**
  - Users should not change or recompile their codes for provenance.
  - More aggressively, users should not disable it when provenance is used in critical missions.

However, in many cases, these metrics are conflicting

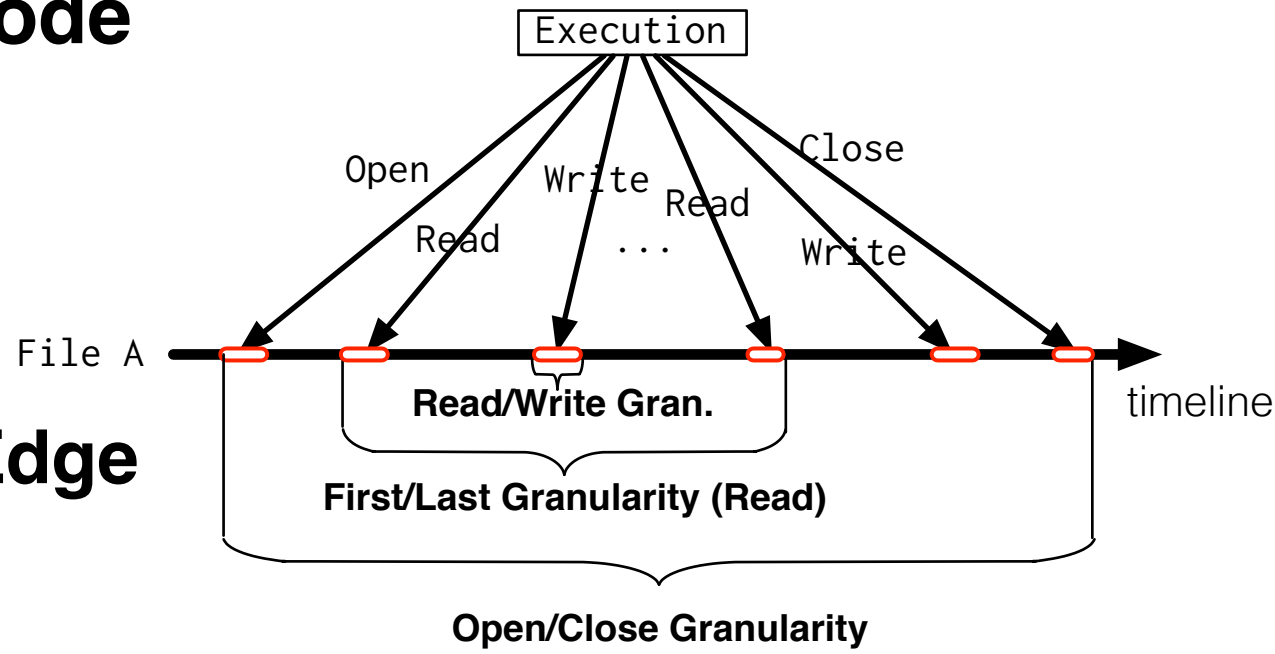
- To cover more details, one has to collect more fine-grain events, introducing higher **probing** overheads
- To be transparent, one has to rely on noisy system events, introducing higher **processing** overheads

Existing solutions make a fixed tradeoff among overhead, coverage, and transparency during design.

- PASSv2 [ATC'06], SPADEv2 [Middleware'12]: transparent and detailed, with 23% and 10% overhead respectively
- Zoom [VLDB'07], VisTrails [CC'08]: low overhead, but only covers workflows
- ...

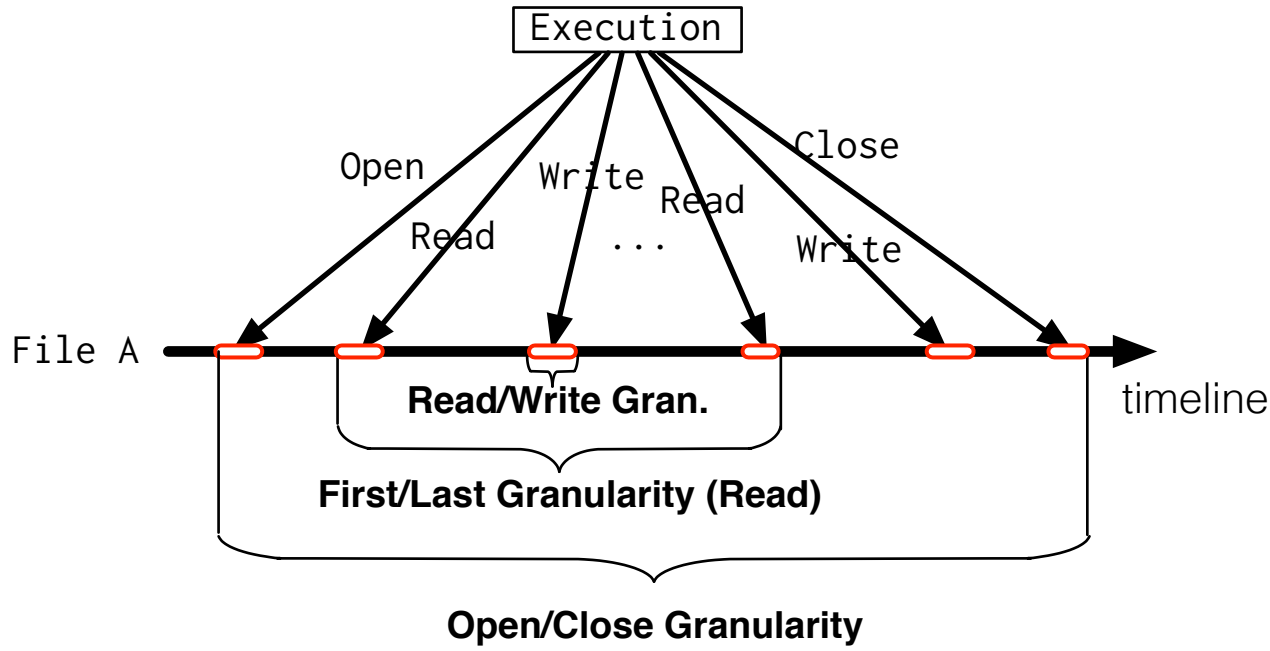
LPS – Adapt with Flexible Provenance Granularity

- Fix the primitive of provenance **Node**
  - Agent: user
  - Process: local process
  - Artifact: whole file (local or distributed)
- Adapt the primitive of provenance **Edge**
  - Mainly “Process” to “Artifact”
  - Also important for causality inference
  - Open/Close
  - First/Last
  - Read/Write



LPS – Adapt with Flexible Provenance Granularity

- Open/Close
  - 2 events per `{process, file}` pair
  - No need to know all read/write operations
  - **Low Overheads, Coarse Granularity**
- First/Last
  - 2 events per `{process, file}` pair
  - **Need** to know all read/write operations
  - **High Overheads, Fine Granularity**
- Read/Write
  - **N** events per `{process, file}` pair
  - **Need** to know all read/write operations
  - **Resource consuming, Not practical in HPC**



Flexibly **Change** the granularity according to current workloads

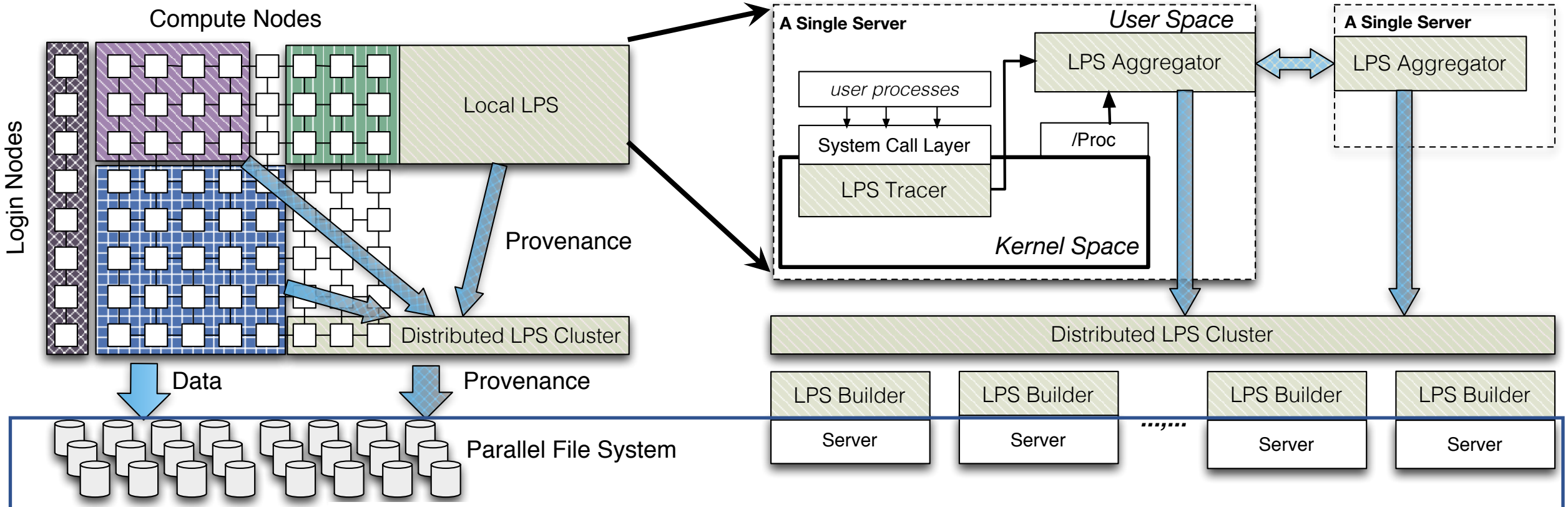
LPS – Unified Model for Flexible Granularities

- Flexibly changing the granularities means uncomplete event pairs, like `Open` and `LastAccess`;
- A unified granularity model
  - $[T_{event\_start}, T_{event\_end}]$ to represent a unified granularity
  - allows any two events to be paired to form a granularity
  - rule table determines the legal granularity

	Open	First Access	Last Access	Close
Open	×	×	◇◇	◇
First Access	×	×	◇◇◇	◇◇
Last Access	×	×	×	×
Close	×	×	×	×

LPS Design and Implementation

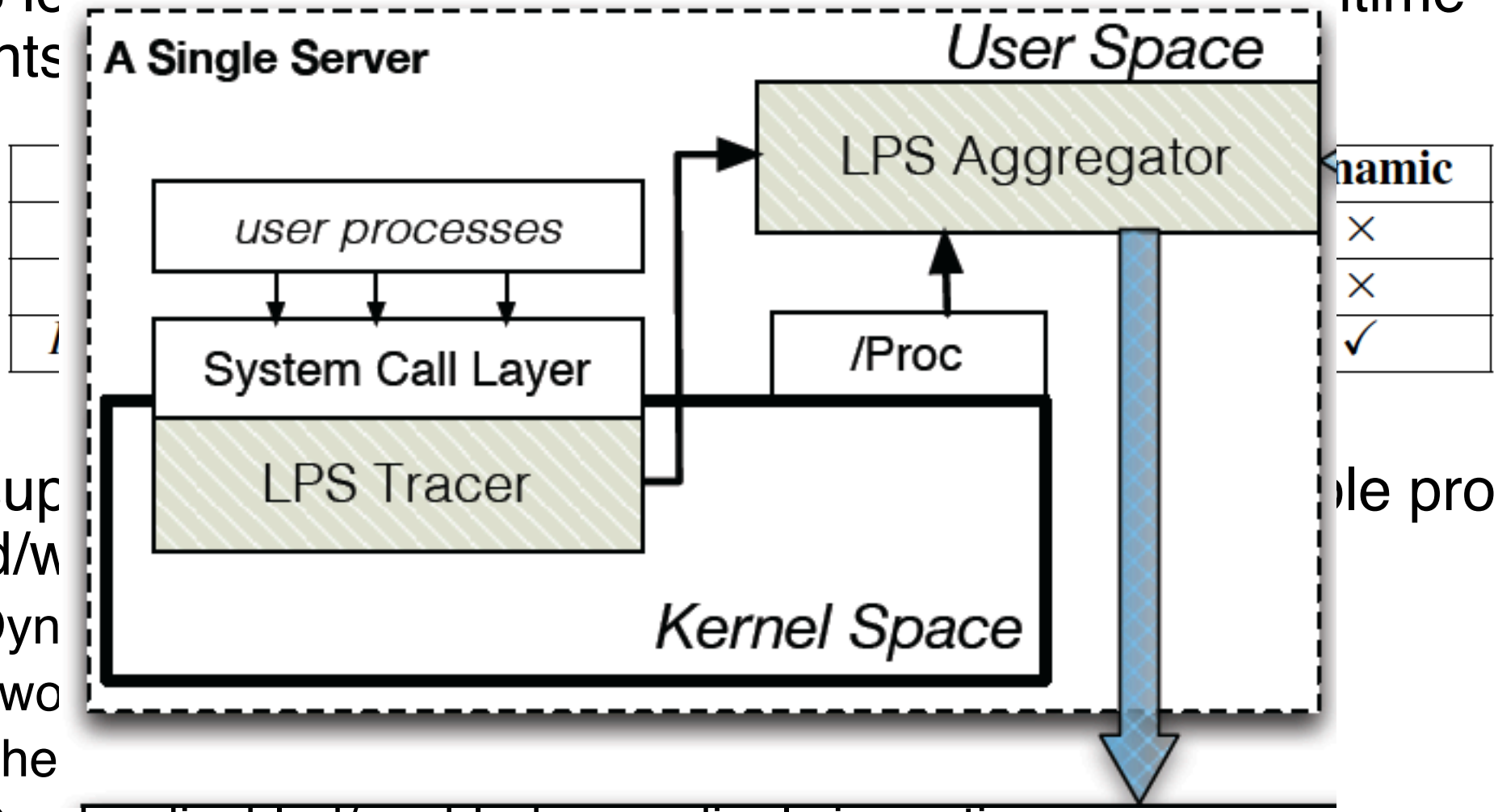
LPS Overall Architecture in HPC



covered in Dong et. al., GraphMeta: A Graph-Based Engine for Managing Large-Scale HPC Rich Metadata, IEEE Cluster, 2016

LPS Tracer

- LPS le ntime events



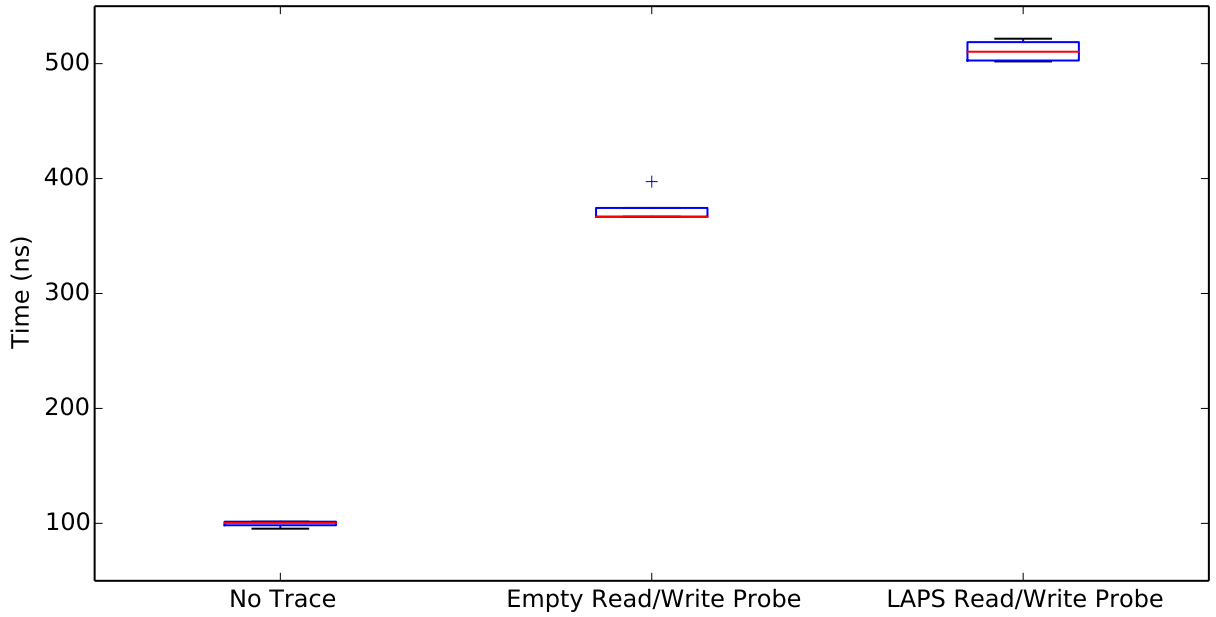
namic
×
×
✓

- To sup le probing read/w
  - Dyn
  - Two
  - The
  - Can be disabled/enabled accordingly in runtime

LPS Aggregator

1. Monitoring overhead and direct granularity change
2. Pruning noisy events to improve performance

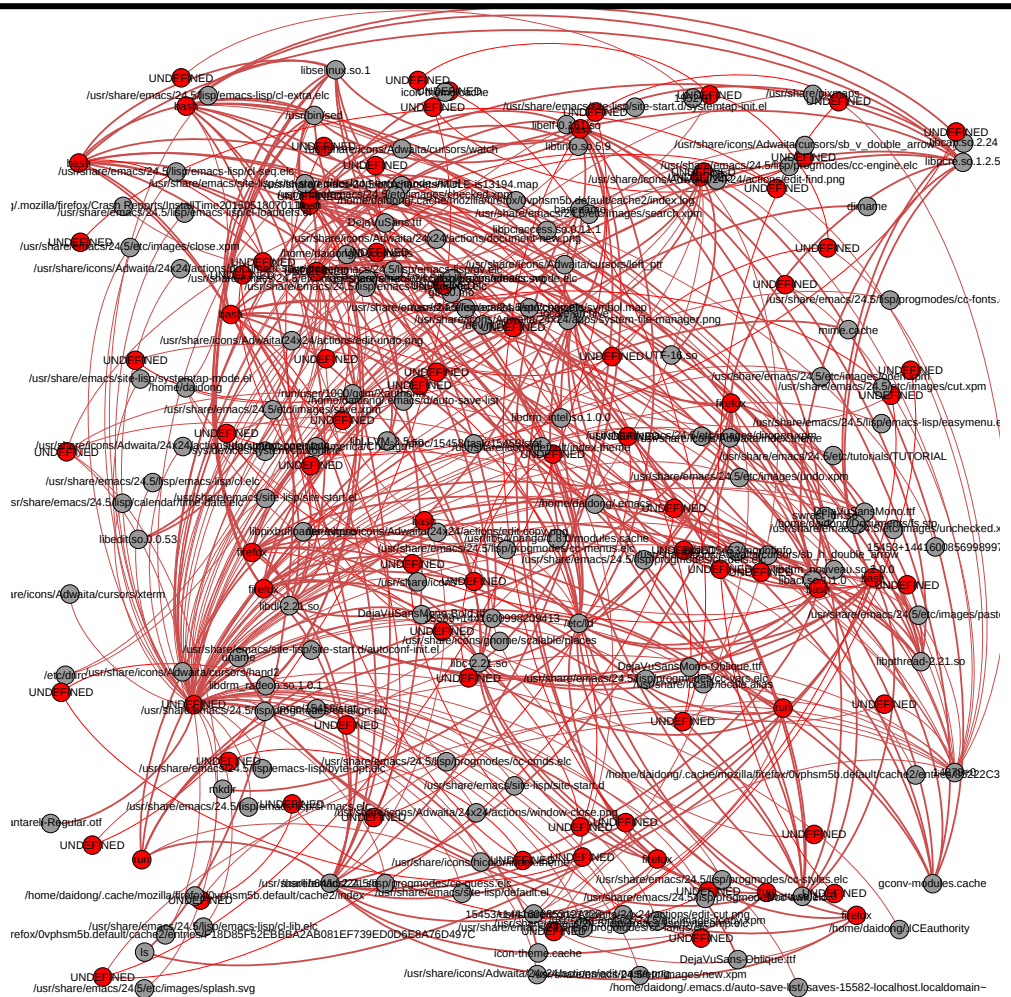
- Instrumentation introduces overheads
  - Instrument Read/Write towards an application issuing 1M 1-byte writes
- The aggregator monitors read/write frequency
  - a counter records the events
  - a timer that resets the counter
  - notify and change granularity



LPS Aggregator

1. Monitoring overhead and direct granularity change
2. Pruning noisy events to improve performance

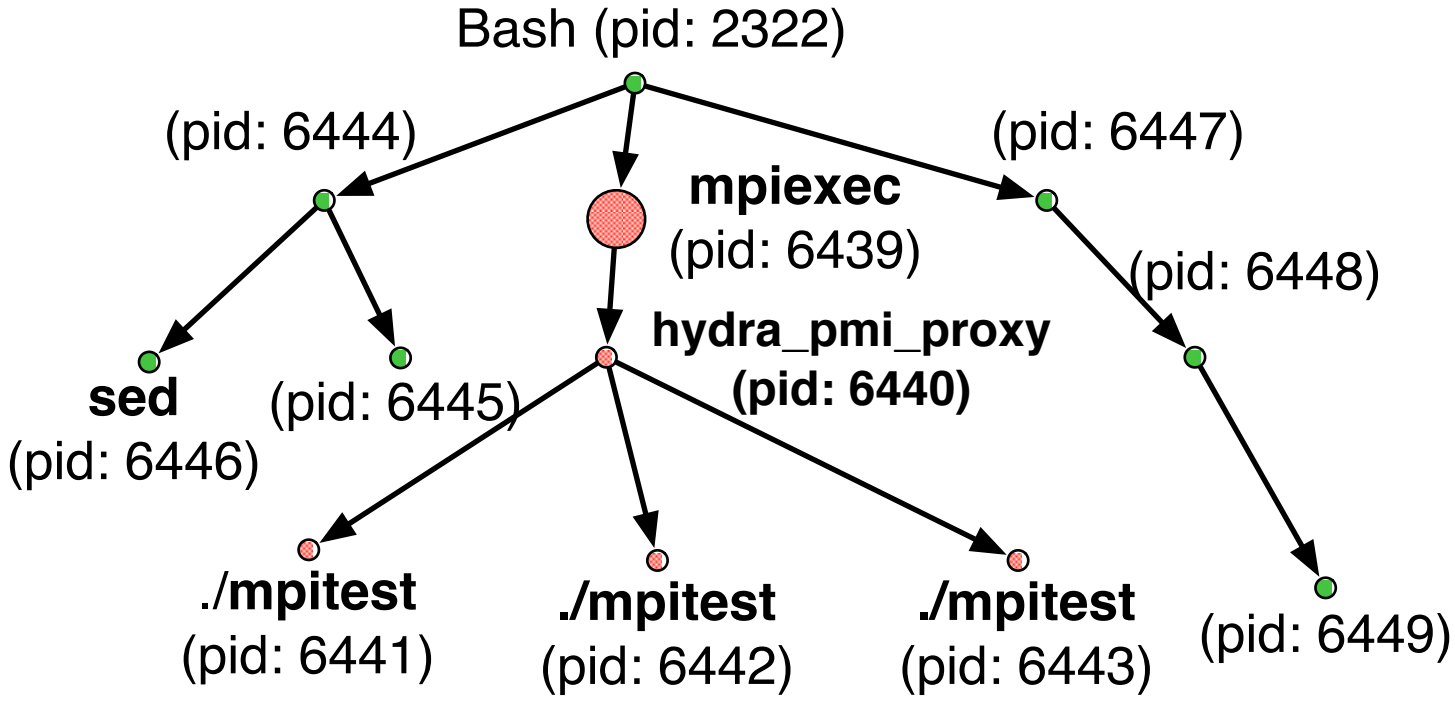
Raw system events from kernel instrumentation



LPS Aggregator

1. Monitoring overhead and direct granularity change
2. Pruning noisy events to improve performance

- Representative Executions
  - Executions that users care the most
  - Eliminate unimportant child processes
  - Eliminate helper child processes
- Events of non-R executions are counted to their ancestor R executions



LPS Builder

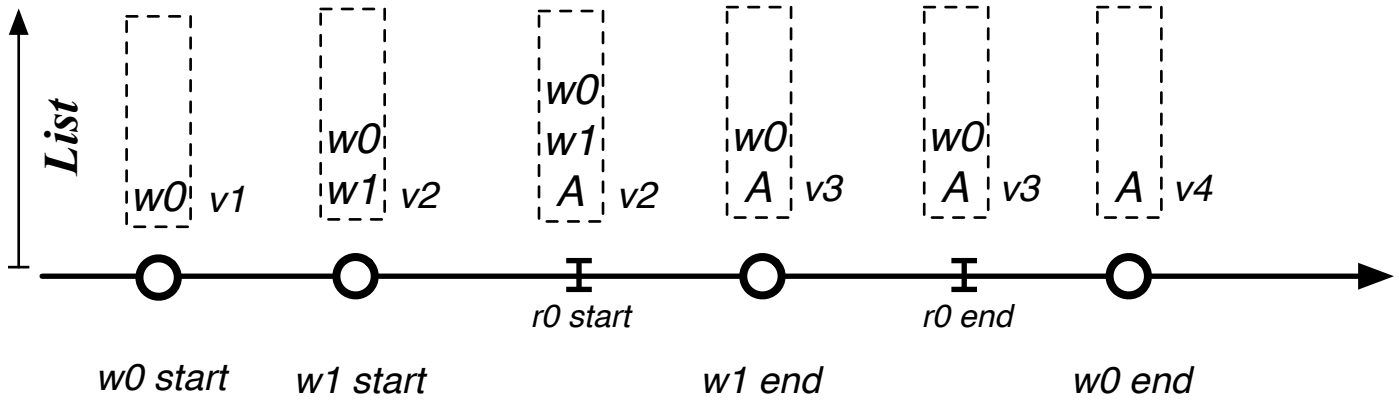
1. Fusing with environmental variables
2. Building provenance with versioning

- Local aggregators generate isolated provenance events
  - Workflows or jobs that are across multiple servers
- A fatal challenge
  - To match identities in different machines needs a unique ID
  - Unique IDs are generated by specific software, no transparency
- A compromise solution
  - LPS relies on specific environmental variables to match identities
  - LPS should be notified about the name of these env variables

LPS Builder

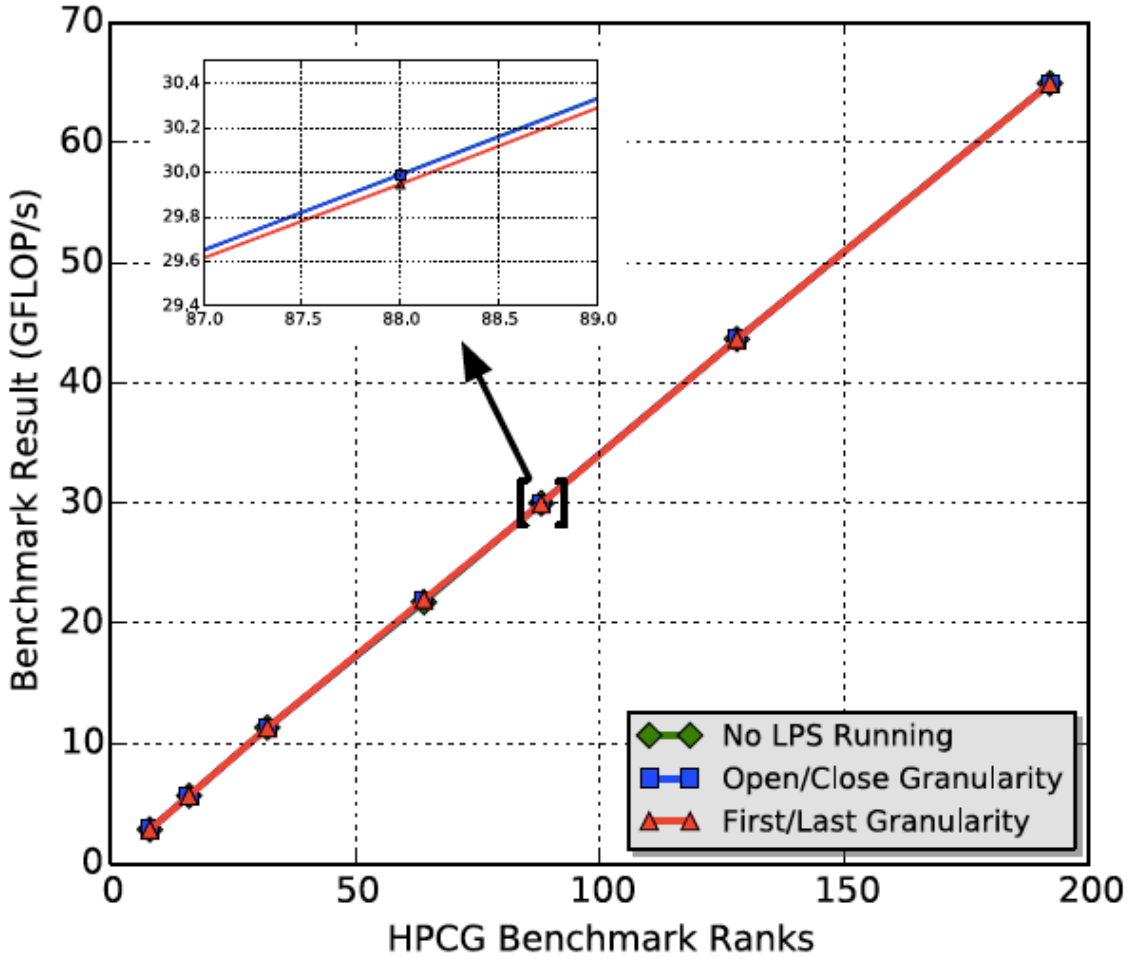
1. Fusing with environmental variables
2. Building provenance with versioning

- Events on the same file will be sent to the same **builder**
- RULE: If events are overlapped, read always depend on the newest version that the overlapped writes create



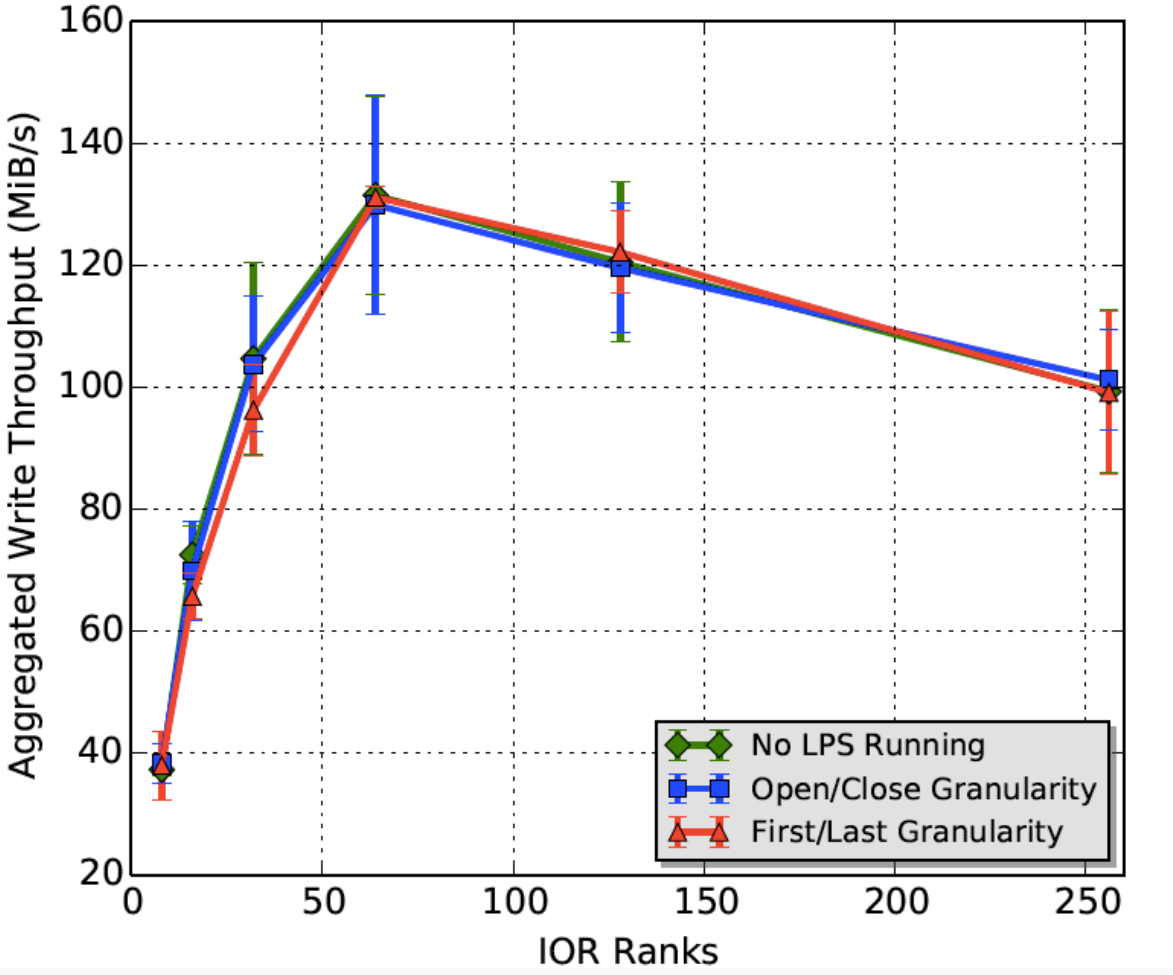
Evaluation Results – *LPS* on Benchmarks

- All evaluations done on CloudLab
  - 45 servers are used to build the HPC environment
- HPCG Benchmark
  - Network- and CPU-intensive workloads
  - Report performance in GFlops
  - Runs on 8 – 196 processes
- Results
  - Less than 0.1% performance degradation



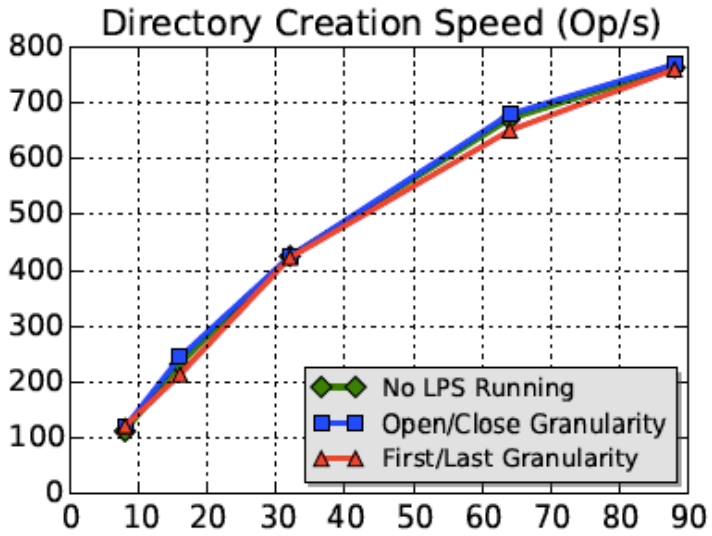
Evaluation Results – *LPS* on Benchmarks

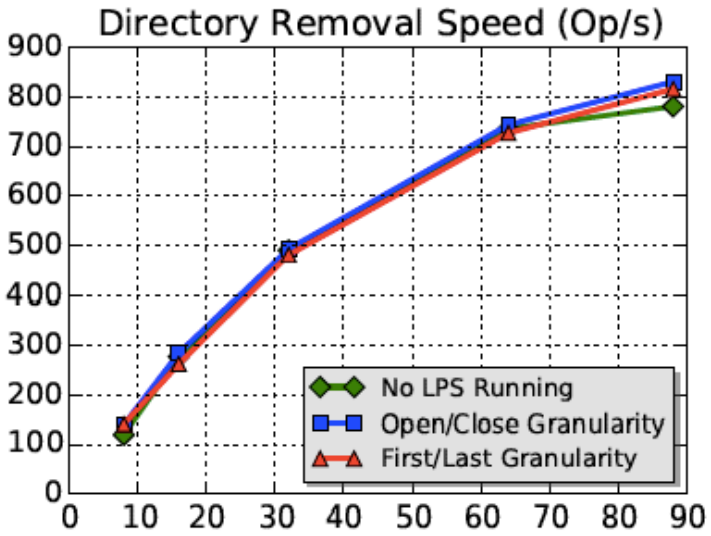
- IOR Benchmark
  - Data-intensive workloads
  - 256 processes, each running on a core
  - issues 50K rounds of random 4K writes
  - Runs on 8 – 256 processes
- Results
  - Open/Close barely introduces overhead (around 0.1%)
  - First/Last introduces noticeable overheads, but they are largely covered by the variations and still less than 1%

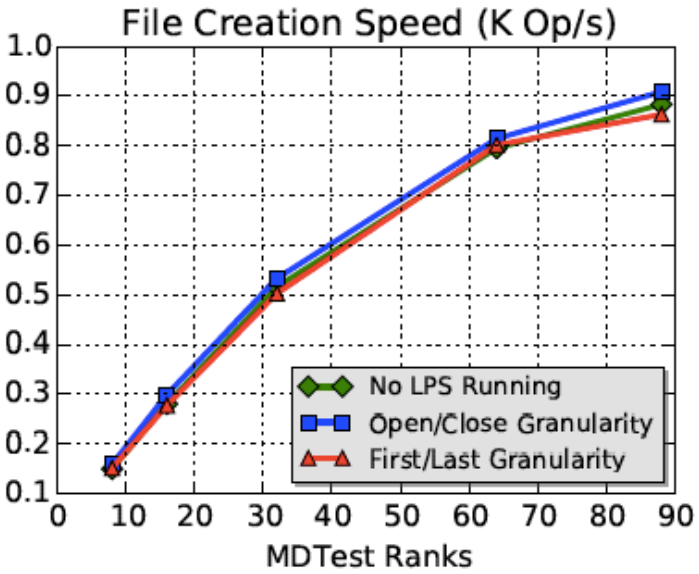


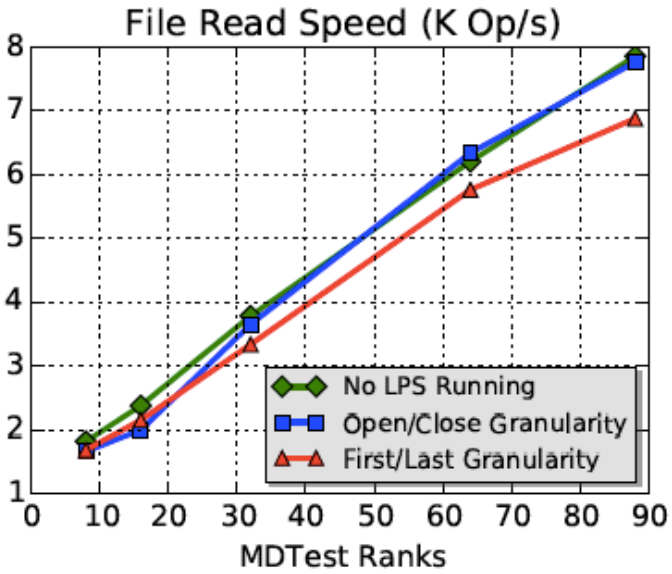
Evaluation Results – *LPS* on Benchmarks

- MDTest Benchmark
  - metadata-intensive workloads
  - report 4 representative operations
- Results
  - Less than 1% overhead in all ops except file read
  - MDTest file reads hit the I/O buffer a lot.
  - Should switch to open/close in this case

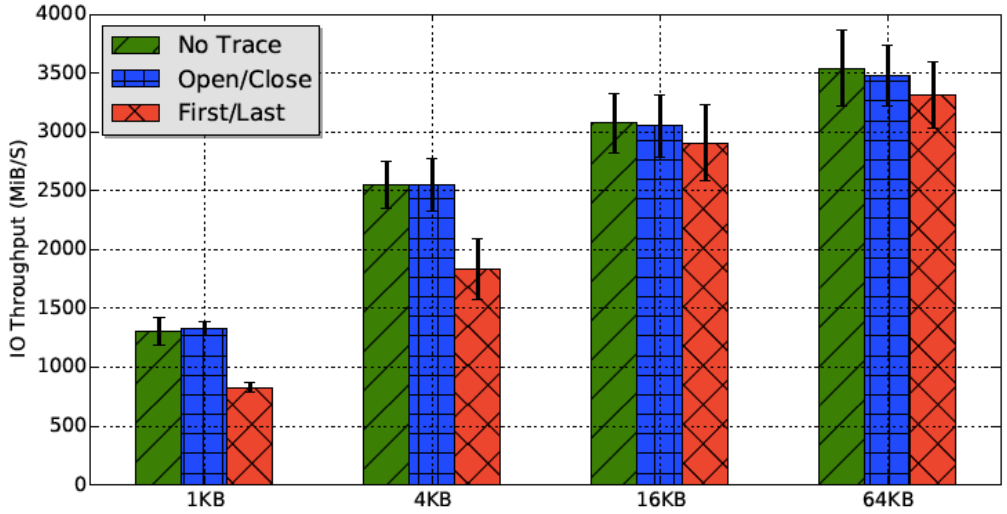




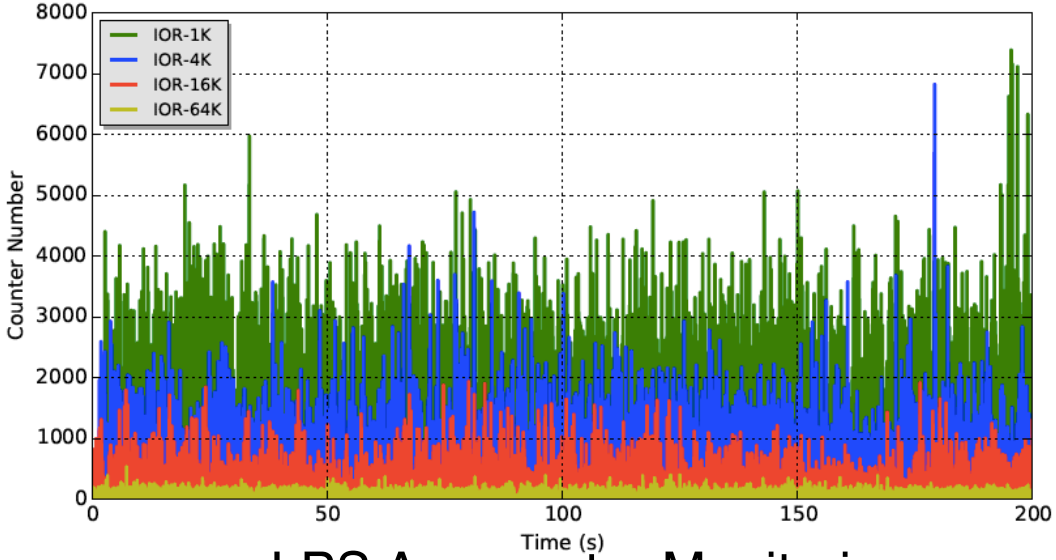




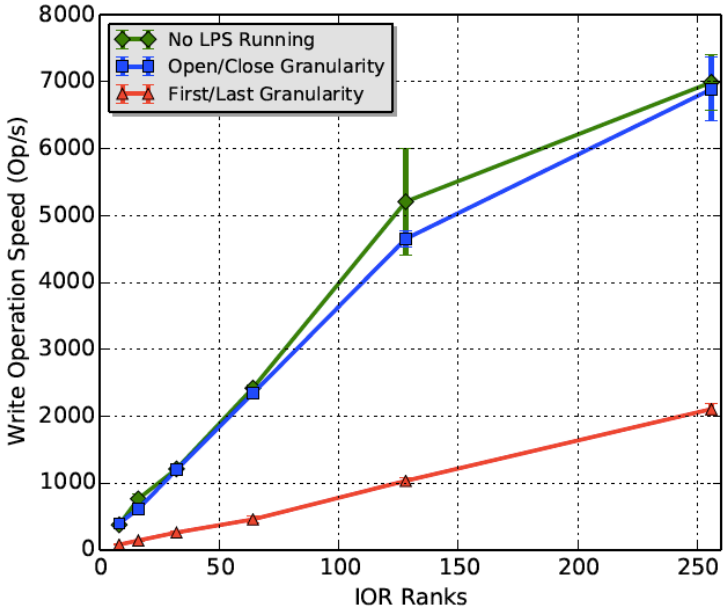
Evaluation Results – *LPS* Component by Component



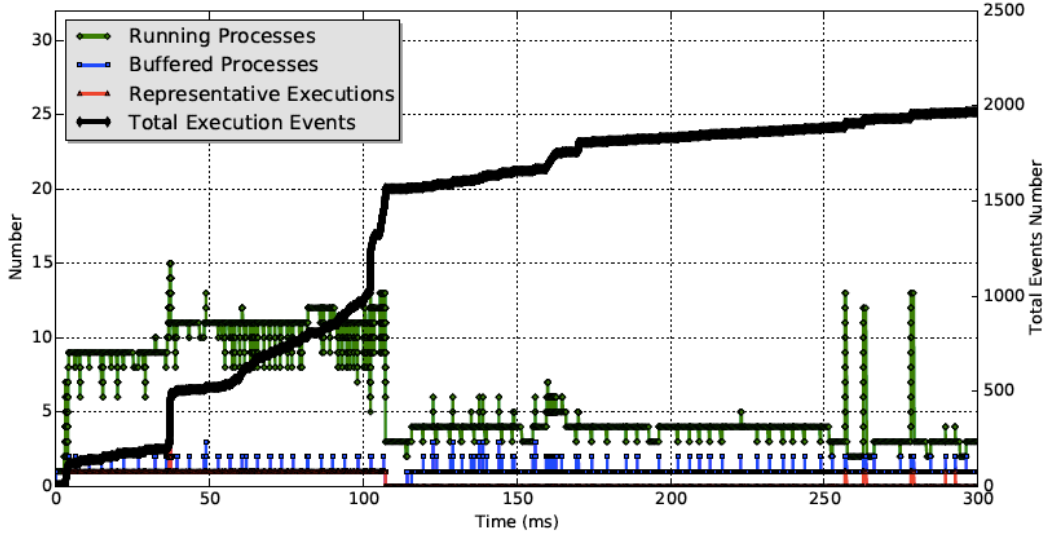
LPS Tracer Overhead



LPS Aggregator Monitoring



LPS Builder Scalability



LPS Aggregator Pruning

Summary and Future Work

- Summary
  - It is always needed to balance between performance and accuracy
  - Using flexible provenance granularity, LPS is able to capture full provenance in HPC environment with low overhead
- Future Work
  - Quantitatively analysis on the uncertainty introduced by flexible provenance granularity

Thanks & Questions

Provenance