IOMax: Maximizing Out-of-Core I/O Analysis Performance on HPC Systems

Izzet Yildirim[1], Hariharan Devarajan[2], Anthony Kougkas[1], Xian-He Sun[1], Kathryn Mohror[2]

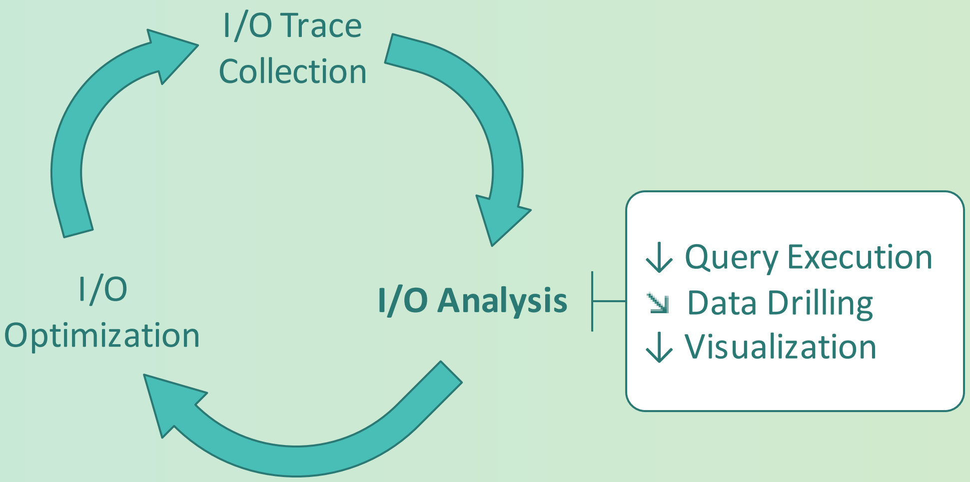
iyildirim@hawk.iit.edu, hariharandev1@llnl.gov, akougkas@iit.edu, sun@iit.edu, kathryn@llnl.gov

[1]Illinois Institute of Technology, [2]Lawrence Livermore National Laboratory

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344. This material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research under the DOE Early Career Research Program (LLNL-PRES-857387). Also, the material is based upon work supported by the National Science Foundation under Grant no. NSF OAC-2104013, OCI-1835764, and CSR-1814872.

I/O Analysis Cycle

- Optimizing I/O efficiency has become essential for maximizing productivity as scientific workloads on HPC systems become increasingly data-intensive
- The go-to solution: I/O Analysis
  - Involves **gathering I/O traces and examining patterns** to detect anomalies
  - Existing I/O analysis tools **utilize data drilling** to identify I/O bottlenecks within trace data that can fit in memory
- Recently, traces from scientific workloads have reached terabytes in size, **necessitating out-of-core analysis**



I/O Analysis Cycle

- Optimizing I/O efficiency has become essential for maximizing productivity as scientific workloads on HPC systems become increasingly data-intensive
- The go-to solution: I/O Analysis
  - Involves **gathering I/O traces and examining patterns** to detect anomalies
  - Existing I/O analysis tools **utilize data drilling** to identify I/O bottlenecks within trace data that can fit in memory
- Recently, traces from scientific workloads have reached terabytes in size, **necessitating out-of-core analysis**

"**data drilling**": Iterative exploration of data for deeper insights

"**out-of-core analysis**": Analysis of data too large to fit in memory by means of distributed computing

State-of-the-art

- Currently, I/O analysis is conducted using various tools, including
  - **Profiling/Tracing**: Darshan, Recorder, DLIO Profiler
  - **Analysis**: PyDarshan, Drishti, DXT Explorer, Recorder-viz
- Studies mostly rely on these tools to detect I/O problems, for instance
  - **UMAMI** [1], **TOKIO** [2], **IOMiner** [3]: System-wide, Darshan, Pandas, In-memory
  - **Extracting and characterizing I/O behavior of HPC workloads** [4]: Workflow-level, Recorder, Parquet, Out-of-core

Challenges

- I/O analysis on large-scale I/O traces, involving data drilling, is a complex task that faces three major challenges:

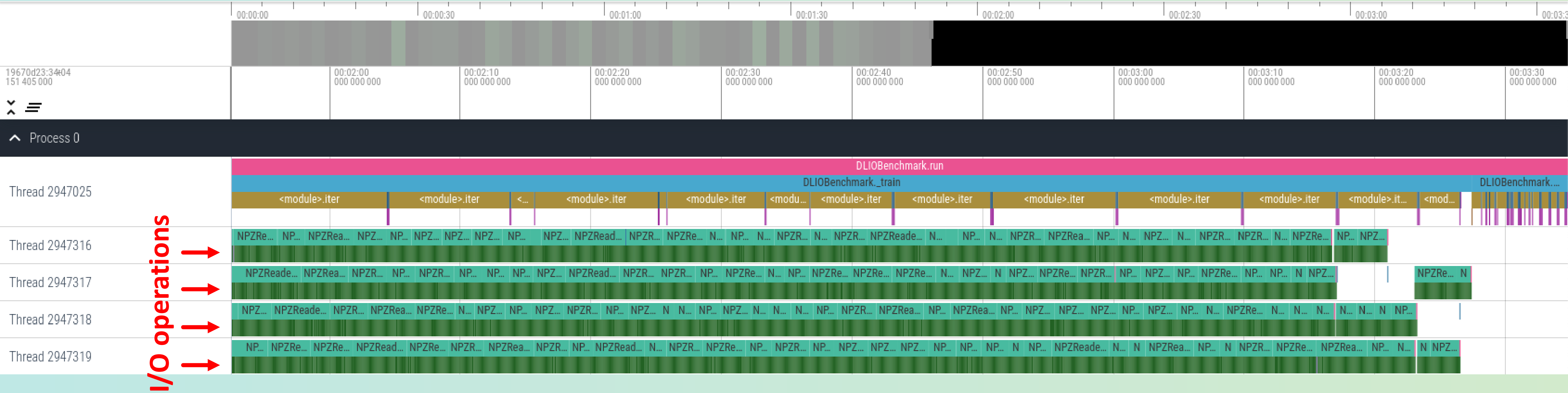
Analyzing terabyte-scale I/O traces

Inefficient slicing and grouping

Lack of global query optimization

Challenges

- As the volume of traces generated by scientific workloads has grown to terabytes in size, there is an increasing need for out-of-core I/O analysis
- For instance, traces from DLIO Benchmark [5] for a TensorFlow workload reaches 5TB in size
  - Accesses 5376 files of 132MB in size, 4 read threads, 1000 epoch
- Calculating the unoverlapped I/O requires efficient slicing and grouping in async I/O scenarios

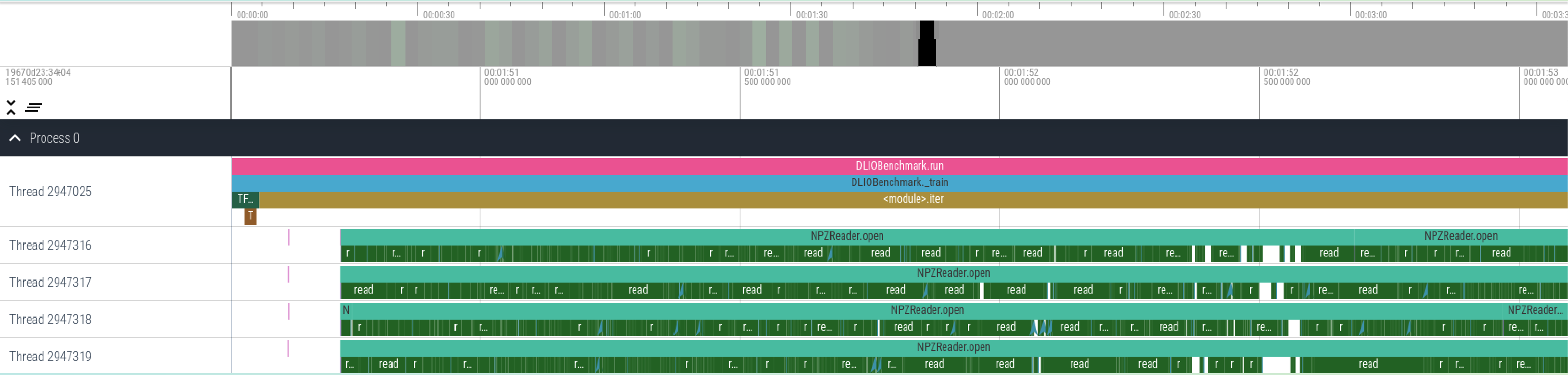


Challenges

- Typical queries to analyze this trace would look like the following in SQL

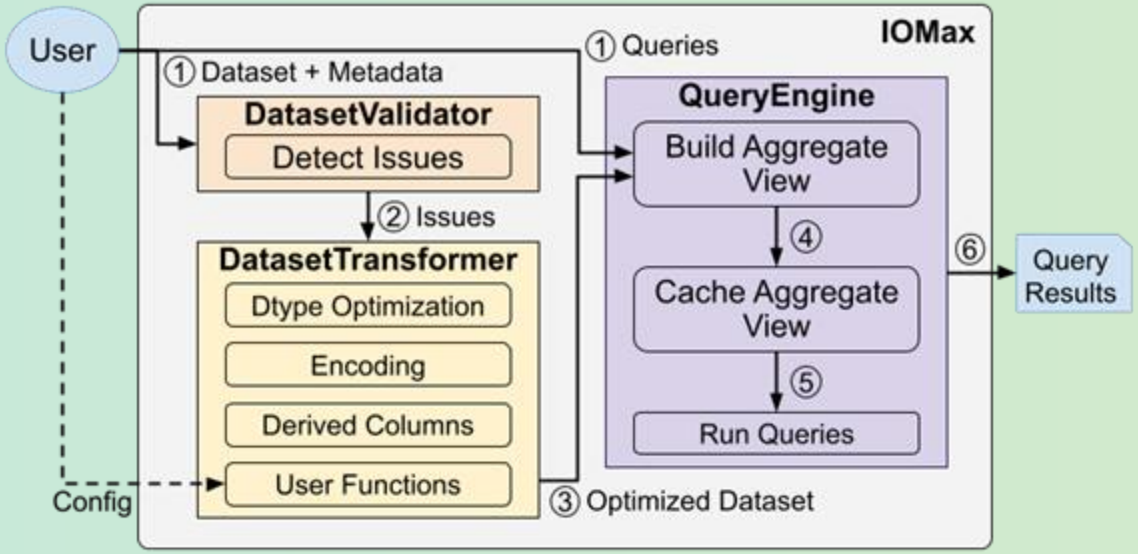
```
# Total I/O size by each thread within a specific time
SELECT SUM(size) FROM traces WHERE time > ‘01:50’ and time < ‘01:53’ GROUP BY thread_id

# Aggregated I/O BW per process within a specific time
SELECT SUM(size)/SUM(duration) FROM traces WHERE time > ‘01:50’ and time < ‘01:53’ GROUP BY process_id
```



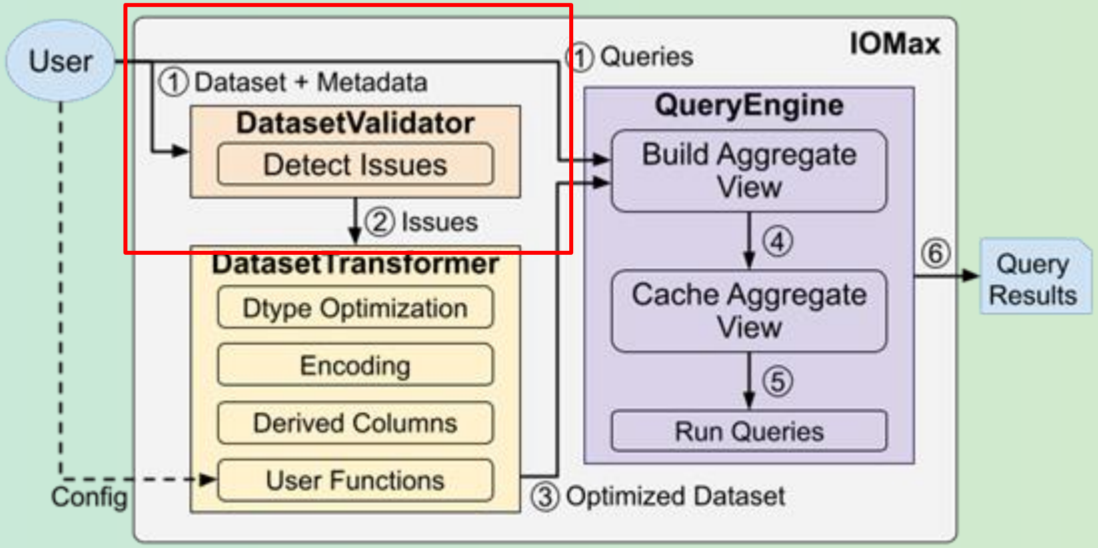
Solution: IOMax

- The IOMax tool is aimed at enhancing the efficiency of data drilling analysis on large-scale I/O traces
- It has three main components
  - `DatasetValidator`
  - `DatasetTransformer`
  - `QueryEngine`



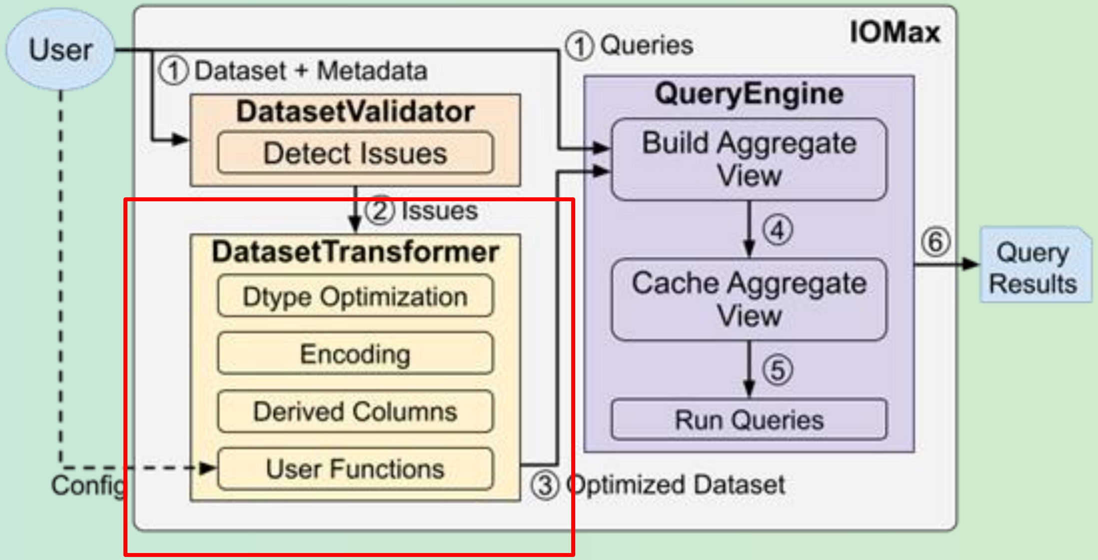
IOMax → DatasetValidator

- Detects issues within the I/O traces
  - Misinferred datatypes
  - Inefficient datatypes
  - Encoding issues (binary/string)
- For instance
  - I/O Category and Access Pattern columns are among commonly misinferred datatypes
  - String columns are highly inefficient for analysis, both when storing and querying them



IOMax → DatasetTransformer

- Corrects misinferred datatypes by converting them into the expected datatypes
- For instance
    - I/O Category and Access Pattern columns usually have limited set of values, hence can be transformed into more efficient datatypes
    - Strings columns such as File and Process Names can be encoded as integers values through hashing



Categorical data columns in I/O trace

Boolean encoding when needed

Column	Misinferred	Transformed	Reduction
I/O Category	32-bit Integer	8-bit Integer	~4x
Access Pattern	32-bit Integer	Boolean	~4x

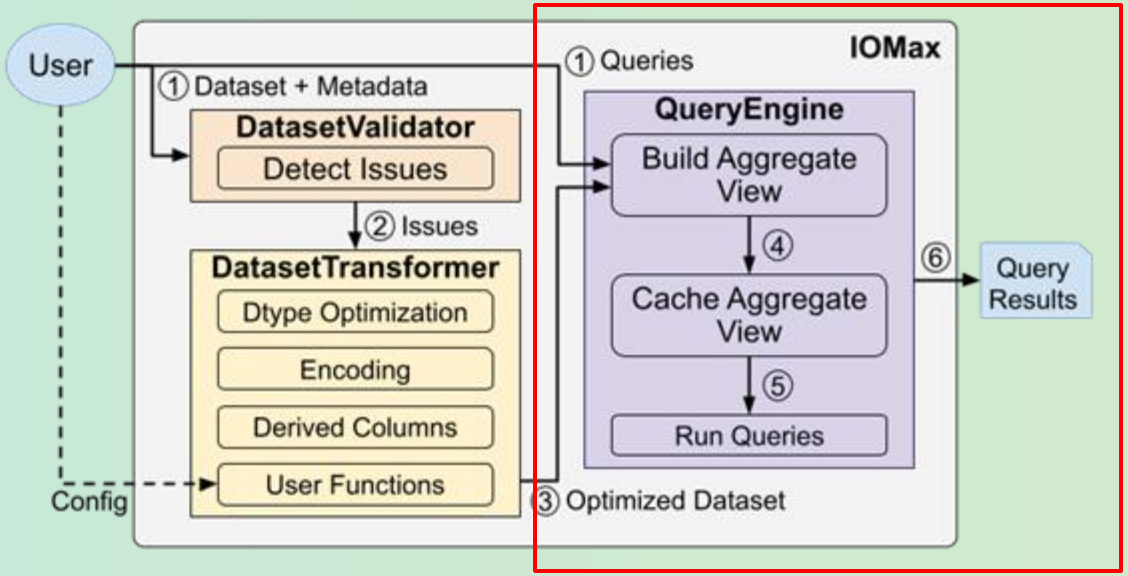
Categorical encoding when needed

Reduction varies due to variable length nature of string columns

Column	Original	Transformed	Reduction
File Name	String	64-bit Integer	10-20x
Process Name	String	64-bit Integer	10-20x
I/O Function	String	Category	60-80x

IOMax → QueryEngine

- Constructs an execution plan that leverages query reduction and in-memory caching techniques to minimize redundant I/O costs




```
# Total I/O size by each thread within a specific time
SELECT SUM(size) FROM traces WHERE time > '01:50' and time < '01:53' GROUP BY thread_id

# Aggregated I/O BW per process within a specific time
SELECT SUM(size)/SUM(duration) FROM traces WHERE time > '01:50' and time < '01:53' GROUP BY process_id
```

Determine appropriate aggregation methods for the metrics accessed

Group "time ranges" with the expected time resolution out of time-based slicing

Merge grouping queries via the aggregate view to avoid redundant access to the same data

Evaluations

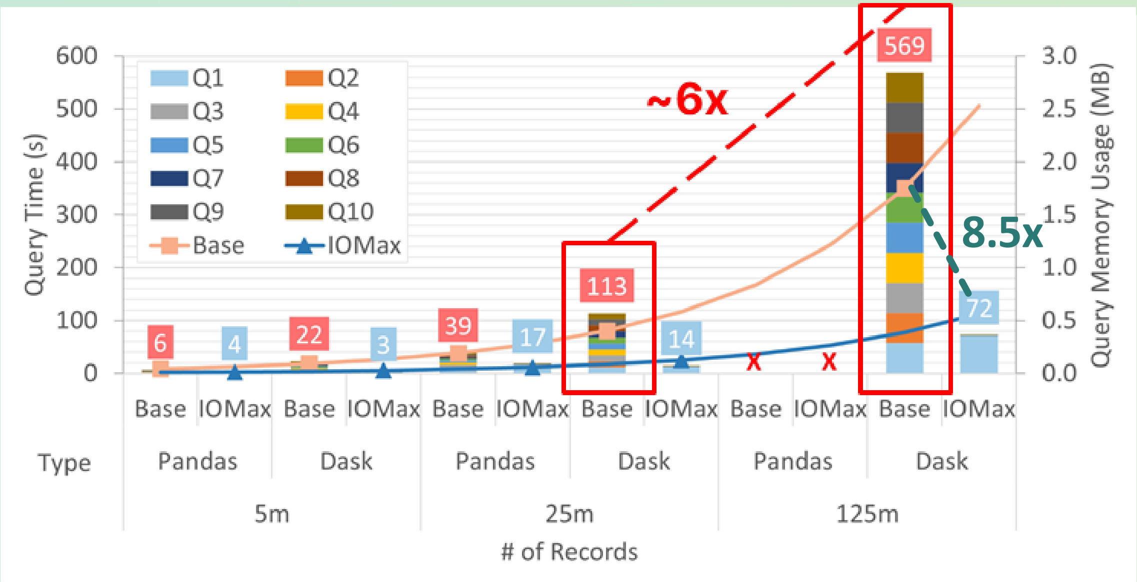
- **Workloads**
  - Microbenchmarks utilizing varied I/O trace records
    - 5 million, 25 million, 125 million
    - Derived from real-world I/O traces
  - 4 scientific HPC workflows
    - 1000 Genomes, Montage (in this presentation)
    - HACC, CM1 (in the paper)

Workload	# of Records	# of Files	# of Processes	Size
1000 Genomes	715,248,240	21,268,291	2,712	440GB
Montage	12,346,353	19,680	11,488	30GB

- **Hardware**
  - Lassen supercomputer at LLNL
    - IBM Power9 CPU, 256GB RAM
    - IBM Spectrum Scale FS (GPFS)
- **Software**
  - Pandas (Data analysis library)
    - Utilized for in-memory analysis
  - Dask (Distributed computing library)
    - Utilized for out-of-core analysis
    - Pandas compatible APIs
  - Recorder (I/O tracing library)
    - Provides fine-grained I/O events

Evaluations → Data Reduction

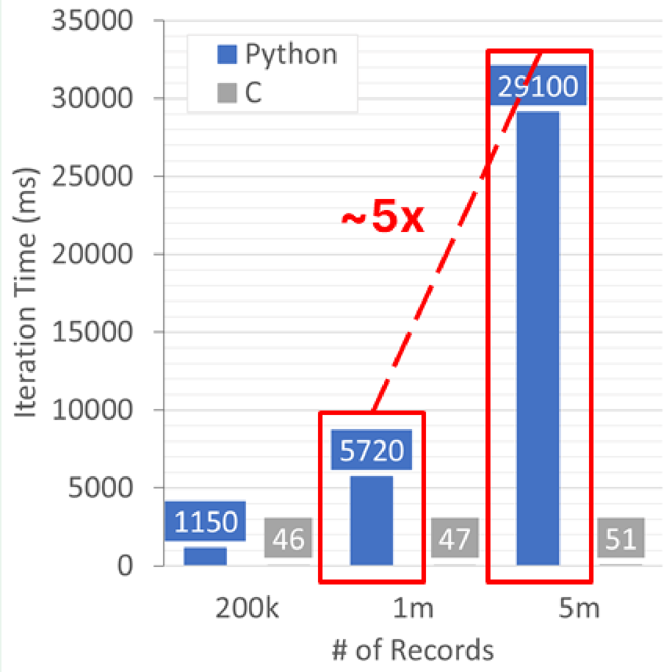
- 10 real-world I/O analysis queries are executed against datasets to showcase the effects of independent query treatment and the increasing memory footprint
- The execution time of the unoptimized queries increases linearly, **~6x per scale**, while optimized version scales well
- The optimized version has a **memory footprint that is 8.5x smaller** than the unoptimized version



Pandas fails to load traces in memory

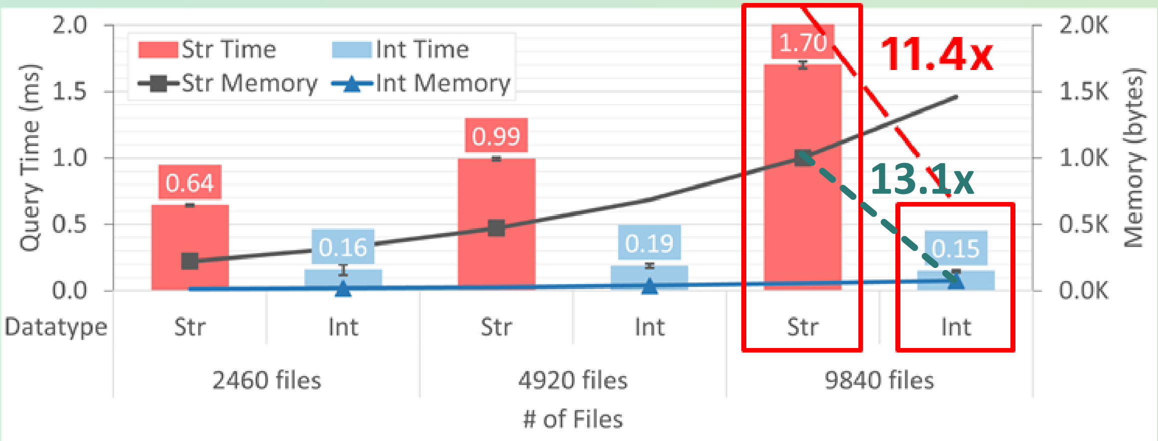
Evaluations → Iterative Queries

- Performing I/O analysis and data drilling involves iterative operations. Typically, this is part of the analysis process. In our case, we offloaded these iterative operations to our preprocessor (`DatasetTransformer`), which is written in C
- The findings reveal a linear increase in Python's iteration time, **~5x per scale**
- In contrast, C exhibits a constant iteration time, averaging around 50 ms



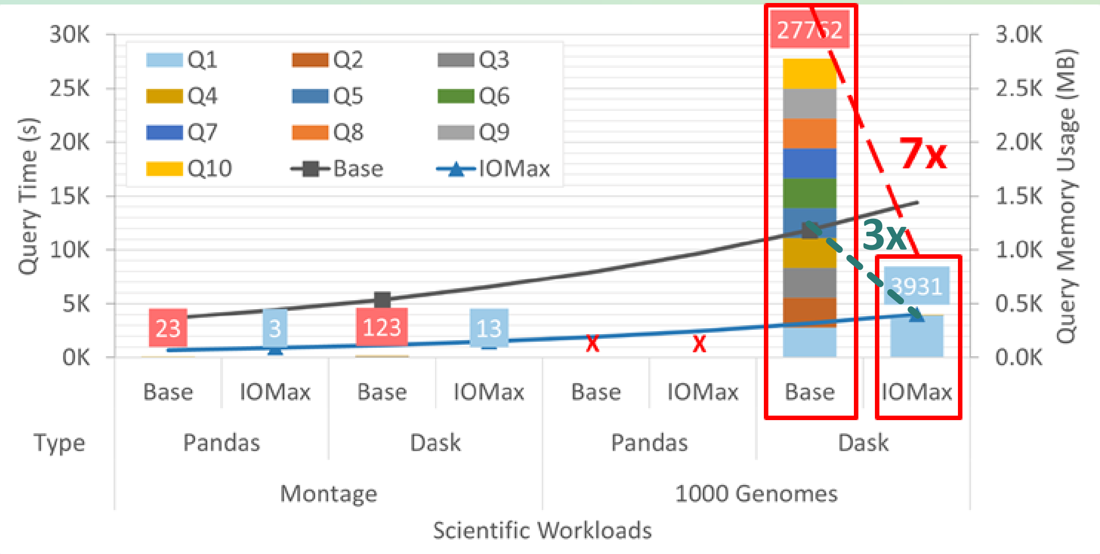
Evaluations → Datatype Performance

- Range queries were executed on both string and integer indices to identify the top ⅛, ¼, and ½ of the most accessed files
  - Our `DataTransformer` transforms string columns like filename into integer through hashing
- Accessing subsets with string indices exhibits a linear increase in access time relative to the number of files, **resulting in an 11.4x slower performance**
  - In contrast, integer indices shows a nearly constant access time, averaging around 15µs
- String indices incur a **memory footprint 13.1x larger** than that of integer indices



Evaluations → Scientific Workflows

- 10 real-world I/O analysis queries were executed against the I/O traces of Montage and 1000 Genomes to illustrate the overall benefits derived from our methodology
- The finding show that IOMax **improves the analysis performance up to 7x** for large-scale I/O traces
- Additionally, it **reduces the memory footprint** of queries **by 3x**



Conclusion

Analyzing terabyte-scale I/O traces

Our methodology improves real-world **large-scale I/O analysis performance by 7x**

Inefficient slicing and grouping

Our data transformer **improves data-slicing performance by 11.4x**

Lack of global query optimization

Our query optimizations achieve **up to 8.6x performance improvement** and an **11x reduction in memory usage**

Thank you!

Any questions?

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344. This material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research under the DOE Early Career Research Program (LLNL-PRES-857387). Also, the material is based upon work supported by the National Science Foundation under Grant no. NSF OAC-2104013, OCI-1835764, and CSR-1814872.

References

1. G. K. Lockwood et al., “UMAMI: a recipe for generating meaningful metrics through holistic I/O performance analysis,” in Proceedings of the 2nd Joint International Workshop on Parallel Data Storage & Data Intensive Scalable Computing Systems - PDSW-DISCS’17, Denver, Colorado: ACM Press, 2017, pp. 55–60. doi: 10.1145/3149393.3149395.
2. 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,” Proceedings of the 2018 Cray User Group, 2018.
3. [1] T. Wang, S. Snyder, G. Lockwood, P. Carns, N. Wright, and S. Byna, “IOMiner: Large-Scale Analytics Framework for Gaining Knowledge from I/O Logs,” in 2018 IEEE International Conference on Cluster Computing (CLUSTER), Belfast: IEEE, Sep. 2018, pp. 466–476. doi: 10.1109/CLUSTER.2018.00062.
4. H. Devarajan and K. Mohror, “Extracting and characterizing I/O behavior of HPC workloads,” in 2022 IEEE International Conference on Cluster Computing (CLUSTER), Heidelberg, Germany: IEEE, Sep. 2022, pp. 243–255. doi: 10.1109/CLUSTER51413.2022.00037.
5. 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), Melbourne, Australia: IEEE, May 2021, pp. 81–91. doi: 10.1109/CCGrid51090.2021.00018.