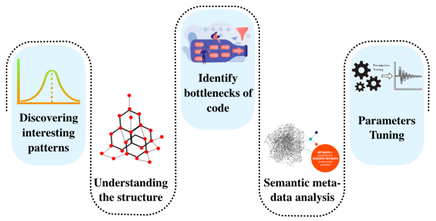
Two-phase IO Enabling Large-scale Performance Introspection

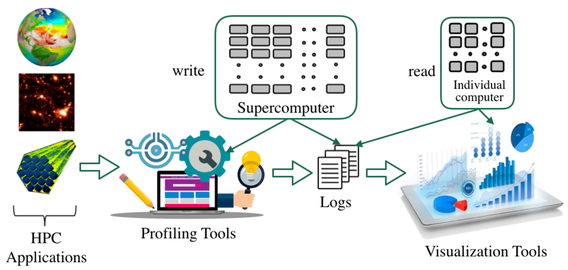
Ke Fan (kfan23@uic.edu), Sidharth Kumar (sidharth@uic.edu)

Background

Performance analysis tools are essential to help HPC experts:



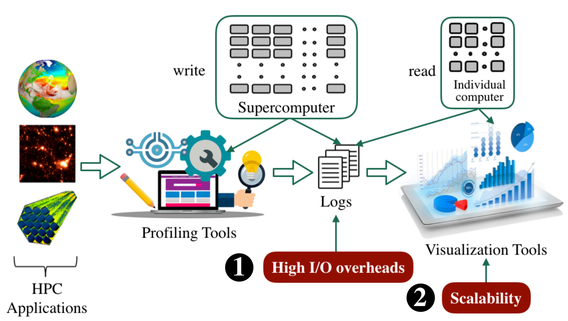
An **end-to-end Performance analysis framework** often consists of a profiling tool and a visualization tool.



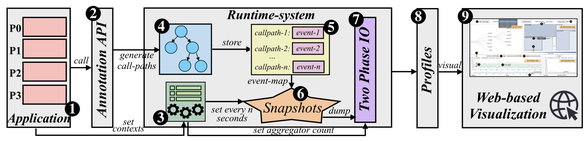
A profiling tool typically runs on supercomputers, while a visualization tool typically runs on individual computers.

Challenges

The **challenges** of profiling and visualizing large-scale parallel programs:



To address them, we proposed an end-to-end framework — **Viveka**:

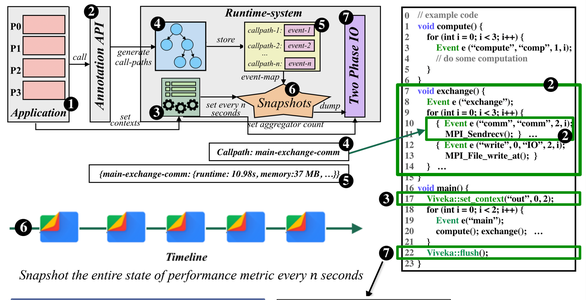


End-to-end Performance Introspection Framework

Profiling Tool:

The parallel profiling includes: 1) source-code annotation API, 2) a runtime system, and 3) parallel I/O.

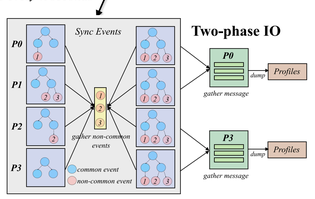
Our main innovation is in developing a **scalable low-overhead runtime**.




```
0  // example code
1  void compute() {
2    for (int i = 0; i < 3; i++) {
3      Event e ("compute", "comp", 1, i);
4      // do some computation
5    }
6  }
7  void exchange() {
8    Event e ("exchange");
9    for (int i = 0; i < 3; i++) {
10     { Event e ("comm", "comm", 2, i);
11       MPI_Sendrecv(); }  ...
12     { Event e ("write", 0, "IO", 2, i);
13       MPI_File_write_at(); }
14   } ...
15 }
16 void main() {
17   Viveka::set_context("out", 0, 2);
18   for (int i = 0; i < 2; i++) {
19     Event e("main");
20     compute(); exchange();  ...
21   }
22   Viveka::flush();
23 }
```

The profiling API with necessary parameters:

- *Viveka:: Profiler(string filename, double io_frequency = 0, int file_count);*
- *Viveka:: Event(string event_name, bool is_common = 1);*
- *Viveka::~Event();*
- *bool Viveka::flush();*



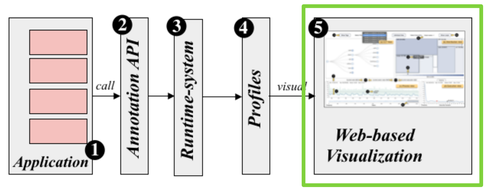
Before the two-phase I/O, we perform an event synchronization phase across all processes to synchronize the non-common events.

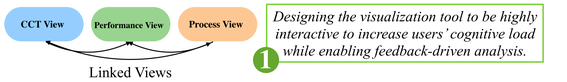
```
"main": 1, compute: 2, exchange: 3, comm: 4, write: 5
 call_path, tag,  loop, times
"1", , 0, "9.34 - 0.17 | 8.67 - 9.03"
"1→2", "comp", 1, "2.78 + 1.92 + 2.48 - 2.82 + 1.66 + 2.73 | 2.54 + 2.13 + 1.88 - 2.67 + 1.52 + 2.93"
"1→3", , 0, "5.19 - 4.95 | 4.99 - 5.23"
"1→3→4", "comm", 2, "3.24 - 3.35 | 3.86 - 3.51"
"1→3→5", "IO", 2, "1.59 - 1.67 | 1.32 - 1.52"
```

The profiles are written in a customized compact file format that reduces all redundancies to minimize the size of the profiles.

Visualization Tool:

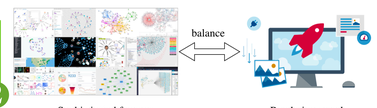
The generated profiles can then be read by the visualization tool.





Designing the visualization tool to be highly interactive to increase users' cognitive load while enabling feedback-driven analysis.

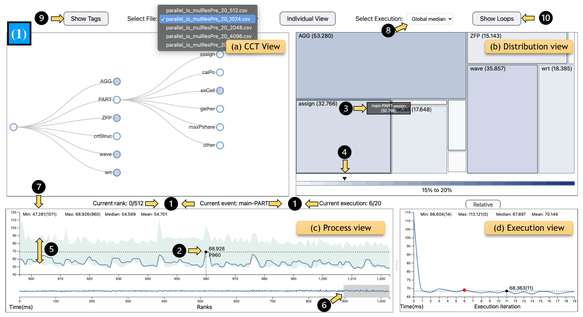
Designing it to strike a balance between providing complex features and operating efficiently at high process counts.



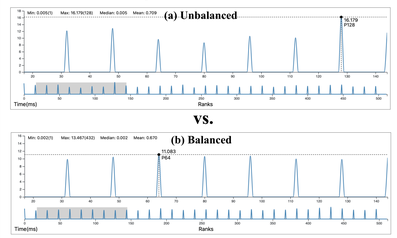
Employing lazy loading for data processing, which identifies resources as non-blocking and loads them only when needed.

Case Study

The screenshot of visualizing a parallel IO application:



Load-balance study:



Process view demonstrating the impact of balanced data aggregation phase. In unbalanced data aggregation (a), there are aggregator (strangler-) processes that must write extra data and therefore slow down the entire application.

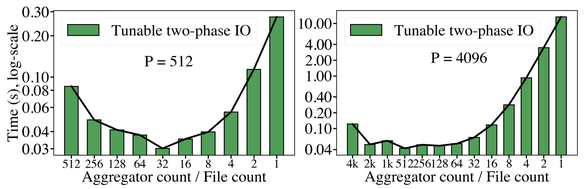
Experiments

All our experiments are performed on the Theta supercomputer at the Argonne Leadership Computing Facility (ALCF).

Argonne NATIONAL LABORATORY

Architecture: Intel-Cray XC40
Cores: 281,088
Speed: 11.7 petaflops
Memory: 843 TB
High-bandwidth Memory: 70TB

The experiments for **two-phase I/O** with varying aggregators (P is number of total processes):

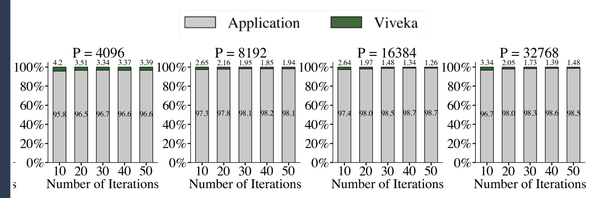


Y-axis is the time taken for the whole program, and x-axis is the number of selected aggregators (= the number of output files).

*From the results, the program with **P/16** aggregators consistently outperforms others.*

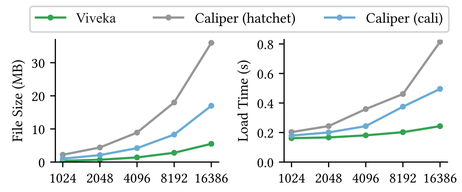
Experiments

We executed the whole program n time repeatedly (ranging from 10 to 50) to test the cost of our performance tool.



From the results, we can see our cost is lower than 5% even for long executions (about 5 mins).

We compare the output sizes of Viveka and **Caliper** with the same program running in the same environment. The output format is tool-specific.



The profiles in our compact format result in the smallest size.

Conclusion

In this paper, we presented Viveka, a lightweight end-to-end system for profiling and visualizing the performance of MPI-based applications.

- A simple data format for the generated logs that minimizes meta data, leading to a smaller storage footprint and faster load times. (on average 3x lightweight than Caliper).
- Developed a custom two-phase data aggregation system to scale parallel I/O (of performance data) to high core counts, ensuring minimal overhead.
- A lightweight web-based visualization dashboard that is capable of performing interactive analysis of performance data collected at high process counts.
- A case study on real parallel applications to demonstrate the efficacy of our profiling and visualization system.

Acknowledgements

This work was funded in part by NSF RII Track-4 award 2132013, NSF PPoSS planning award 2217036, NSF PPoSS large award 2316157 and, NSF collaborative research award 2221811. We thank the ALCF's Directors Discretionary (DD) program for offering us the compute hours to run our experiments on the Theta Supercomputer.