GrIOt: Graph-based Modeling of HPC Application I/O Call Stacks for Predictive Prefetch

Louis-Marie Nicolas, PhD student, ENSTA Bretagne Lab-STICC and Eviden BDS R&D Data Management.

Salim Mimouni, Eviden BDS R&D Data Management, Grenoble, France

Philippe Couvée, Eviden BDS R&D Data Management , Grenoble, France

Jalil Boukhobza, PhD Advisor, ENSTA Bretagne, Lab-STICC, CNRS, UMR 6285, Brest, France

Outline

Context

- There is a gap in performance between volatile memory and storage:
  - Parallel File Systems are used to increase throughput, but they can't help with the latency
  - Heterogenous storage systems are used to combine price and performance, but they require adapted data placement policies
  - In data placement policies, "prefetching" is moving data that will be used in the near future from slower to faster storage, increasing the I/O performance

Technology/ Form Factor	Latency	Throughput Read/Write	IOPS	Capacity Unit
DRAM	$\sim$ 80 ns	17/17 GB/s	-	$<$ 64 GiB
NVRAM	$\sim$ 5 µs	2.5/2.5 GB/s	4.6M	$<$ 480 GB
SSD (NVMe)	$\sim$ 20 µs	8.0/5.0 GB/s	1.2M	$<$ 32 TB
SSD	$\sim$ 100 µs	2.1/2.0 GB/s	0.8M	$<$ 8 TB
HDD	$\sim$ 10 ms	250/240 MB/s	$<$ 500	$<$ 14 TB
Tape	$>$ 20 s	315/315 MB/s	-	$<$ 15 TB

Comparison of memory technologies used in HPC[1]

- We focus on the predictive prefetching of data directly into compute nodes.

(1) Lüttgau, Jakob, et al. "Survey of storage systems for high-performance computing." *Supercomputing Frontiers and Innovations* 5.1 (2018).

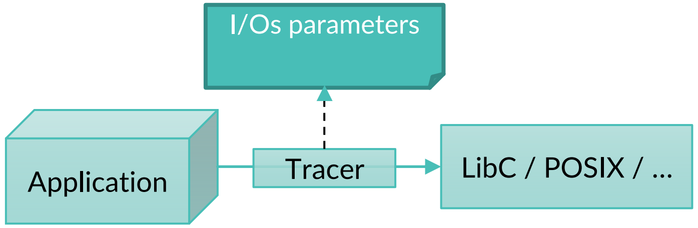
Context

- One prerequisite to predictive prefetch is the knowledge on an application future I/O behavior.

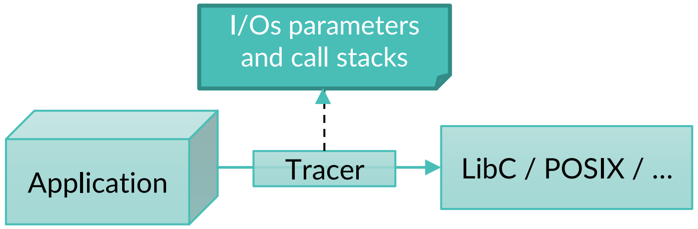
- Three main approaches:

  - ➢ White-box
    - Access and/or modification of an application source code.
    - Adding hints or prefetching primitives to the application code

  - ➢ Black-box
    - Intercepting I/Os.
    - Pattern matching, probabilistic models

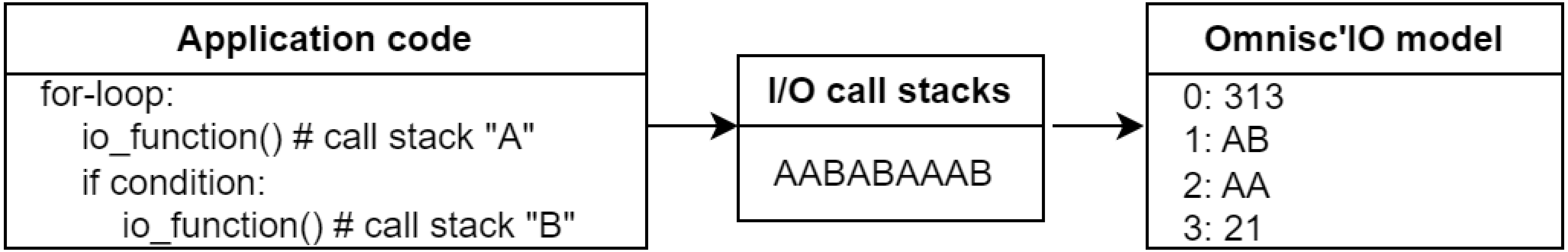


  - ➢ Gray-Box
    - Intercepting I/Os call stacks. Extracting knowledge about an application I/O structure using I/O call stacks.



State of the art

- State of the art Omnisc'IO[1] encodes the I/O call stack sequence through the compression algorithm StarSequitur.

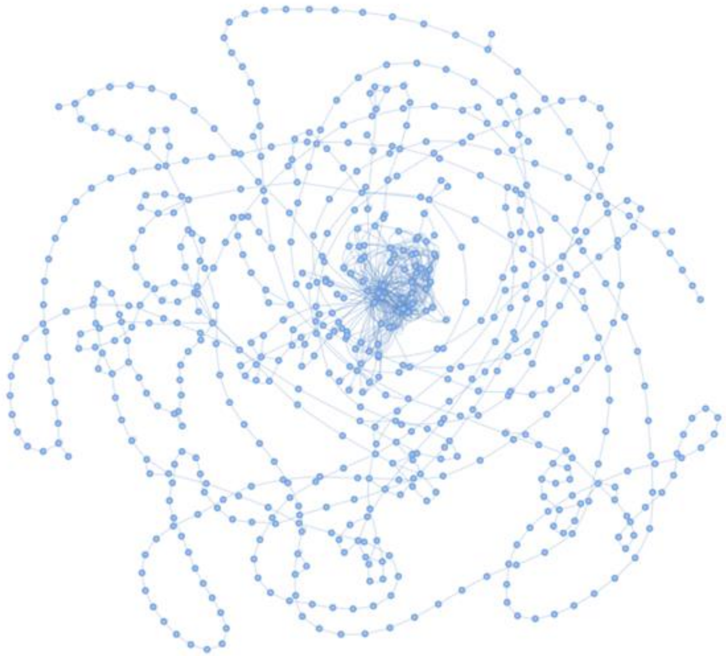


+ The model encodes losslessly the original sequence of I/O.
- It grows in size and complexity when I/Os are not fully deterministic
- It does not include probabilities or heuristics

(1) Dorier, Matthieu, et al. "Omnisc'IO: a grammar-based approach to spatial and temporal I/O patterns prediction." *SC'14*. IEEE, 2014.

Problem Statement

How to model the I/O structure of applications with both deterministic and non-deterministic I/O from I/O call stacks (grey-box) and use this model to make I/O predictions for efficient prefetch ?



A directed graph of call stacks:

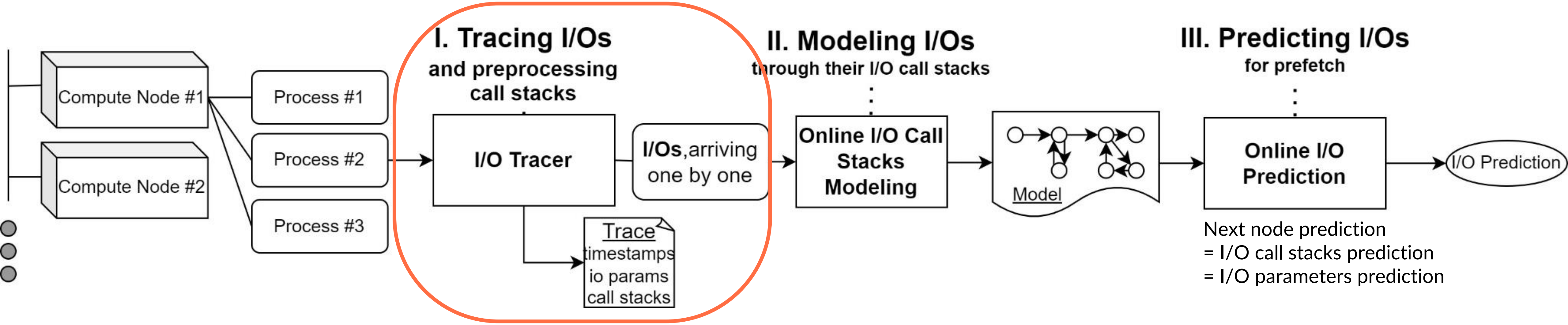
- Has a bounded size, because an application source code has a limited number of different I/O call stacks

- Can support probabilistic behavior by adding metadata to the edges

We present **GrIOt**, a **Gr**aph-based Modeling of **I/O** call s**t**acks for Predictive Prefetch.

Contribution

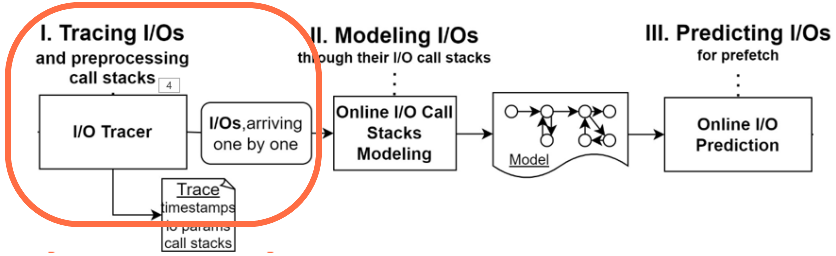
GrIOt, a Graph-based Modeling of I/O call stacks for Predictive Prefetch

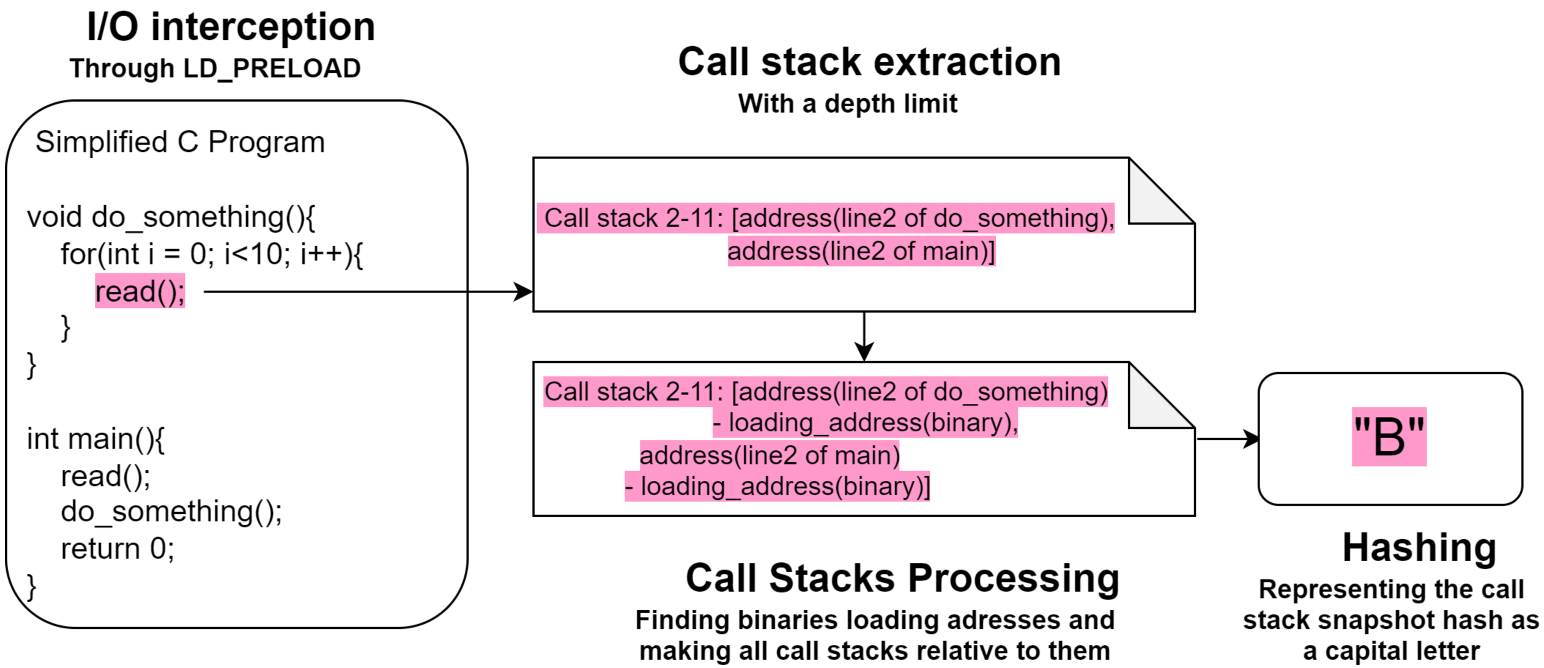


Next node prediction
= I/O call stacks prediction
= I/O parameters prediction

Contribution

Tracing I/O and call stacks

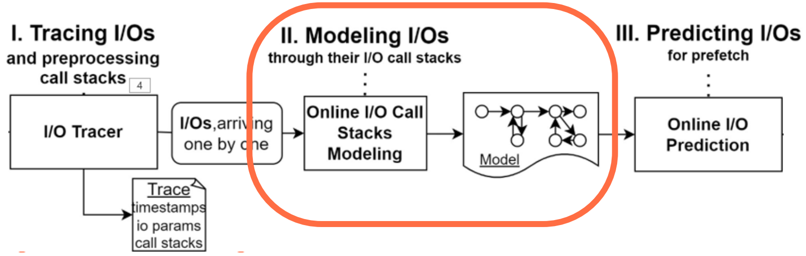




- Created of a simple tracer using LD_PRELOAD. Support for standard POSIX / libC I/O functions that are dynamically linked.
- Obtain the (relative) call stack and I/O parameters of every I/O

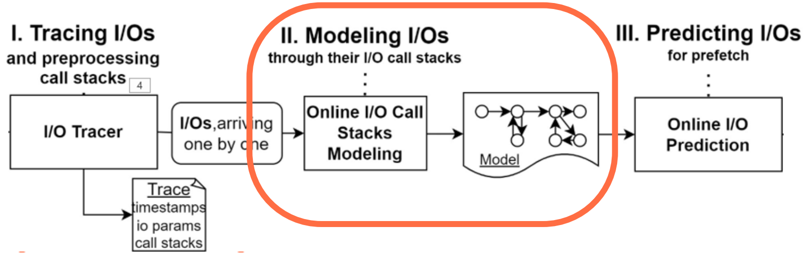
Contribution

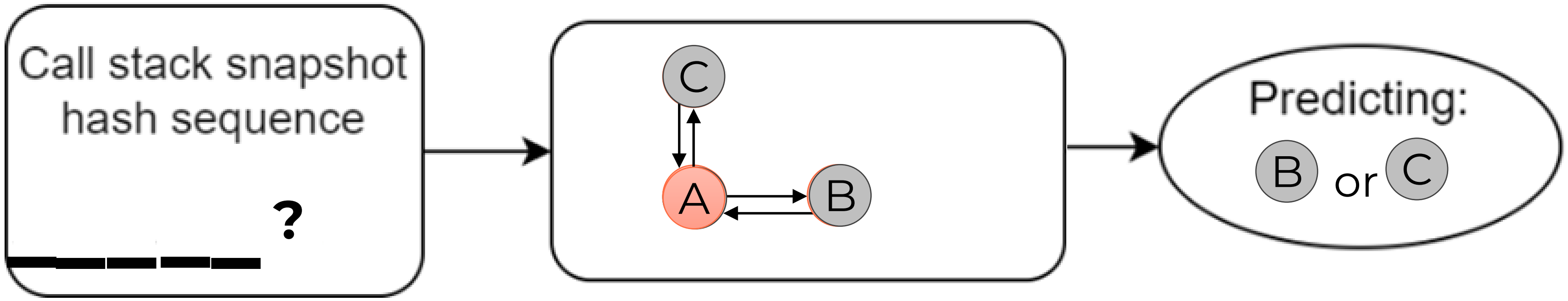
Modeling with GrIOt



- GrIOt creates a directed graph
- One node contains one call stack and possibly a fixed number of previous call stacks.
- An edge from node A to node B is created if an I/O with call stack B was made right after an I/O with call stack A.

Contribution
Modeling with GrIOt

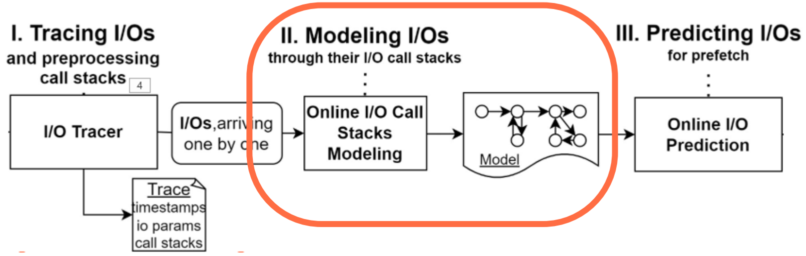


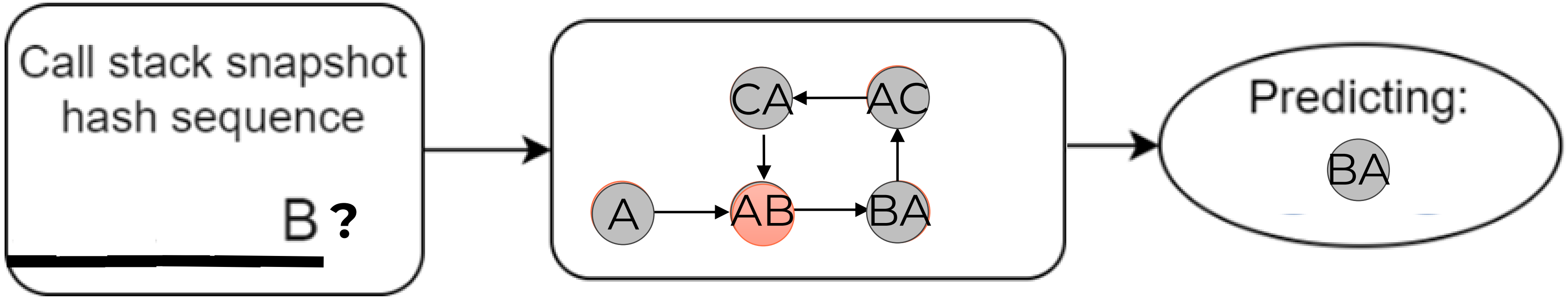


Creation of a GrIOt graph from an I/O call stack sequence
1 node = 1 call stack

- In this example, the next call stack after A depends on the call stack right before A. The graph fails to capture that.

Contribution
Modeling with GrIOt

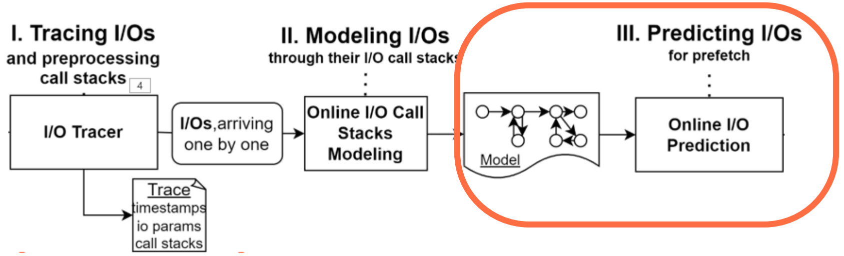




Creation of a GrIOt graph from an I/O call stack sequence
1 node = 1 call stack + 1 previous call stack

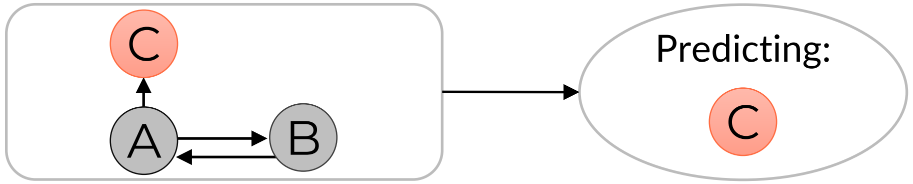
- Adding more call stacks to every node makes the graphs bigger and enables potentially better predictions

Contribution
Predicting with GrIOt

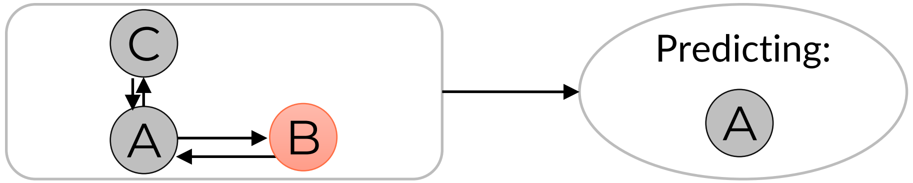


In order to make a prediction, 3 possibilities:

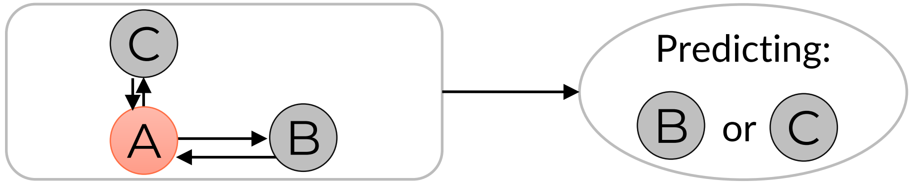
- If the node has no outgoing edge (sequential heuristic)



- If the node has a single outgoing edge



- If there is more than one edge



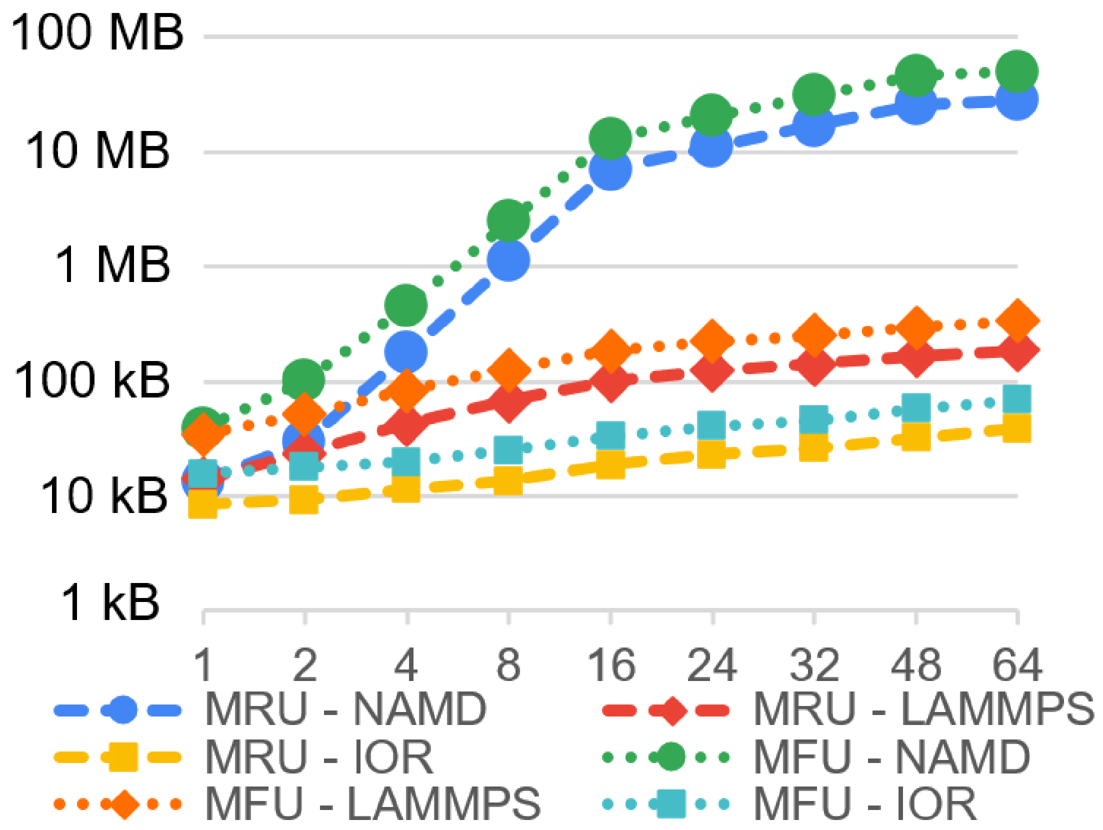
Evaluation

Methodology

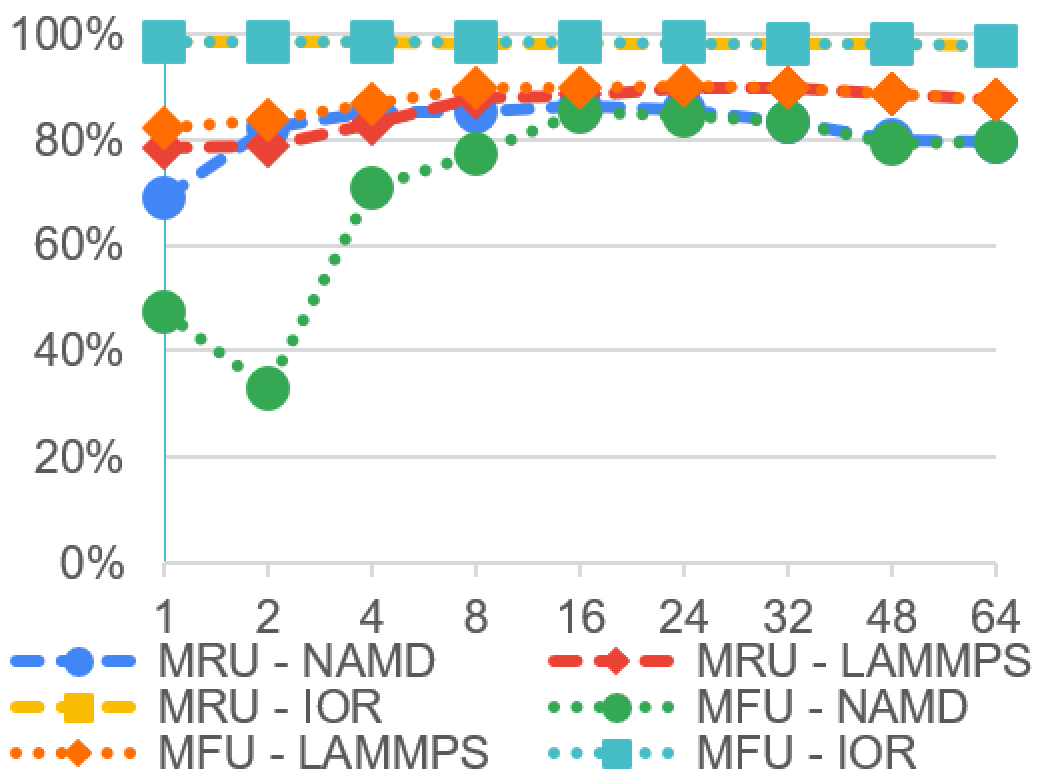
- 2 real world HPC applications, NAMD and LAMMPS. 1 synthetic benchmark, IOR.
- GrIOt was implemented in C (tracer) and Python (modeling and prediction)
- Code will be available on github for reproducibility
- 2 sets of experiments in order to:
    - Evaluate different configurations of GrIOt
    - Compare GrIOt to state-of-the-art: prediction accuracy, overhead, memory footprint

Evaluation

Results



Model size as a function of context size

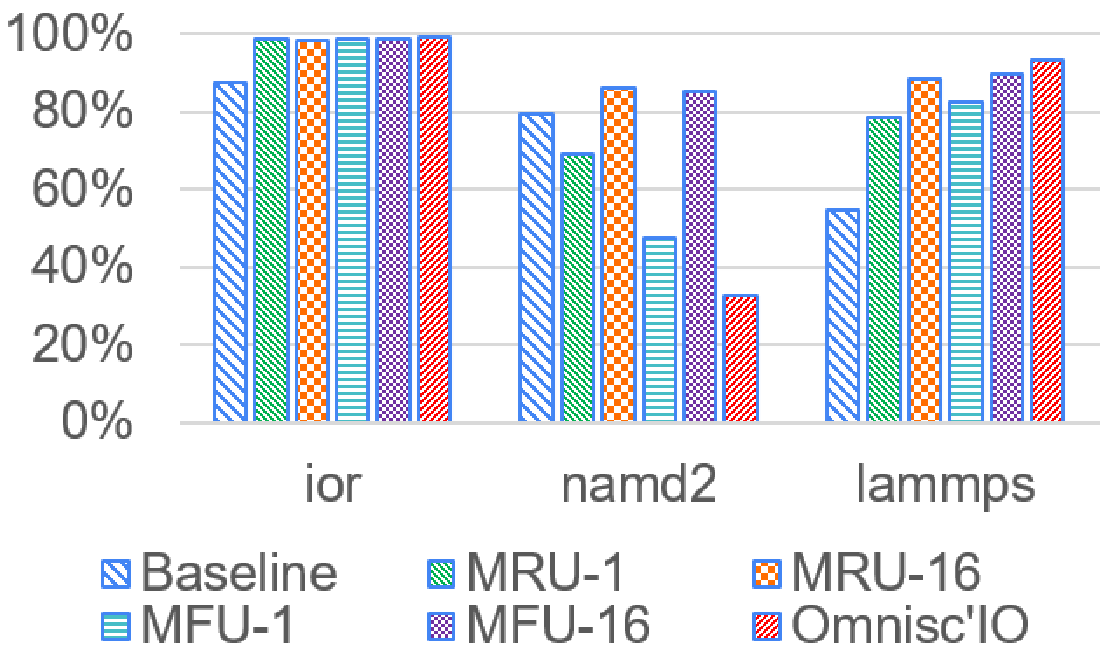


Next I/O call stack prediction accuracy as a function of context size

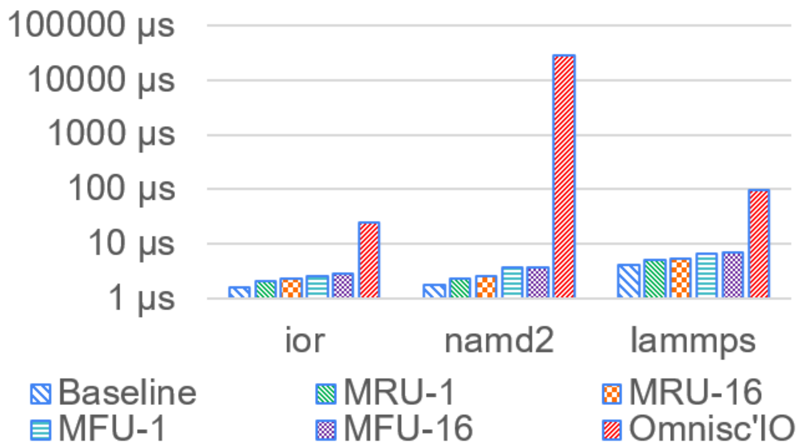
→ When context size goes up, precision and model size are going up too

Evaluation

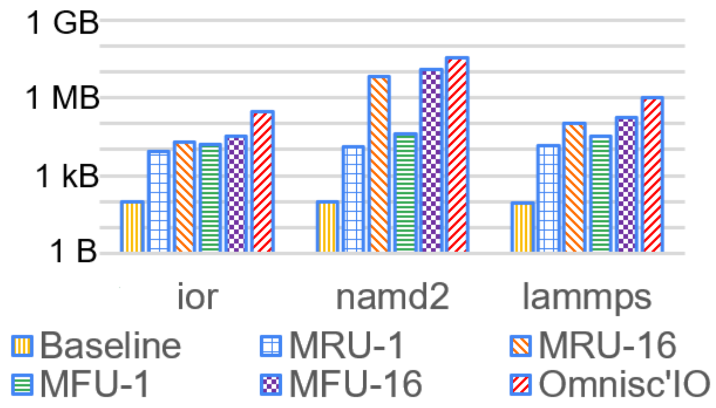
Results



Next I/O call stack prediction accuracy



Modelisation and prediction overhead



Model size

→GrIOt offers similar or better predicting capabilities than state of the art with lower overhead and model size.

Conclusion and Future Work

- **Context**: Multiple prefetching approaches, amongst which the grey-box approach using I/O call stacks.
- **Problem statement**: How to predict I/O for prefetch for both deterministic and non-deterministic I/Os.
- **Contribution**: We presented **GrIOt**, a **Gr**aph-based Modeling of **I/O** call s**t**acks for Predictive Prefetch
    - I/O and call stack interception
    - Modeling with variable-size context
    - Predicting using heuristics such as MRU and MFU
- **Results**: GrIOt offers similar or better results than the state of the art, while keeping a low overhead and memory footprint.

Conclusion and Future Work

Future work will focus on:

- Creating a prefetcher based on GrIOt
- MPI-IO compatibility
- Federating GrIOt models
- Experimenting with per-file, per-open, and per-open-callstack models rather than per-process models
- Reducing the overall overhead

Conclusion and Future Work

- **Context**: Multiple prefetching approaches, amongst which the grey-box approach using I/O call stacks.
- **Problem statement**: How to predict I/O for prefetch for both deterministic and non-deterministic I/Os.
- **Contribution**: We presented **GrIOt**, a **Gr**aph-based Modeling of **I/O** call s**t**acks for Predictive Prefetch
    - I/O and call stack interception
    - Modeling with variable-size context
    - Predicting using heuristics such as MRU and MFU
- **Results**: GrIOt offers similar or better results than the state of the art, while keeping a low overhead and memory footprint.

Thank you! Contact me at louis-marie.nicolas@eviden.com