Adaptive multi-tier intelligent data manager for Exascale

admire-eurohpc.eu

ADMIRE User Day

FTIO: Predicting I/O Phases Using Frequency Techniques

Ahmad Tarraf
Technical University Darmstadt

December 12th 2023
Barcelona Supercomputing Center

Grant Agreement number: 956748 — ADMIRE — H2020-JTI-EuroHPC-2019-1

Motivation

- HPC applications usually alternate between compute and I/O phases (e.g., Checkpointing)
- Compute resources are allocated **exclusive**, while **I/O bandwidth is a shared resource; which often suffers from:**
  - ➢ **Variability**: I/O performance depends on what others are doing
  - ➢ **Contention**: causes lower overall I/O performance
  - ➢ **Lower utilization**: compute resources are often "wasted" while waiting for I/O

SC22 after Jack Dongarra's presentation in the Dallas ballroom

Motivation (cont')

- Several **solutions** exist: I/O scheduling, I/O-aware batch scheduling, burst buffers/caches, ….
- **But they often required knowledge about the application's I/O behavior:** Number of processes doing I/O, request size, transferred bytes, files accesses, ...
- Especially the **temporal I/O behavior** can be useful if proved online, to answer questions like:

When does the application require I/O resource?

How many resources are need?

How long are the phases?

When should the buffers be flushed?

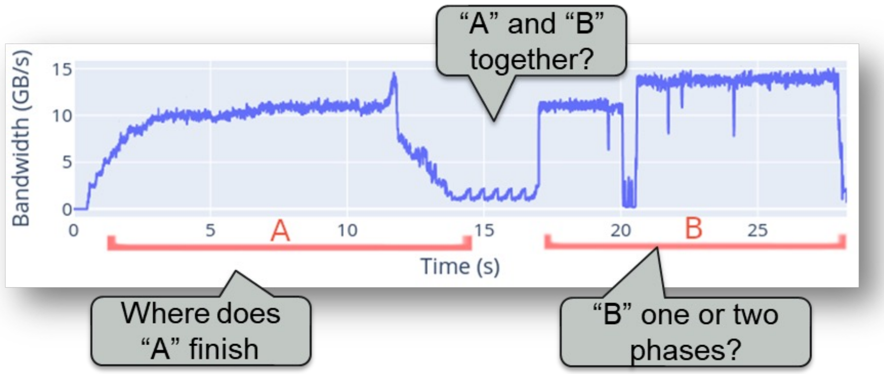
Is it compatible to share them with others?

Should we use burst buffers?

Use for I/O scheduling?

- **This information is not easy to get!**
- **Especially, if we try to describe it in terms of phases!**
  - I/O phases composed of many I/O requests
  - Not all I/O is interesting
  - Boarders of the I/O phases? Threshold?

→Application and system dependent!



FTIO: Frequency Techniques for I/O

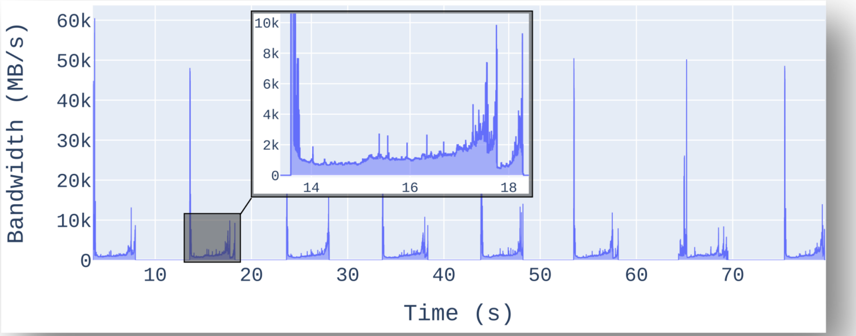
Periodic I/O is often encountered in HPC!

→ Information about applications' periodicity, even if not perfectly precise, leads to good contention-avoidance techniques [1, 2, 3]

→ Frequency Techniques for I/O:

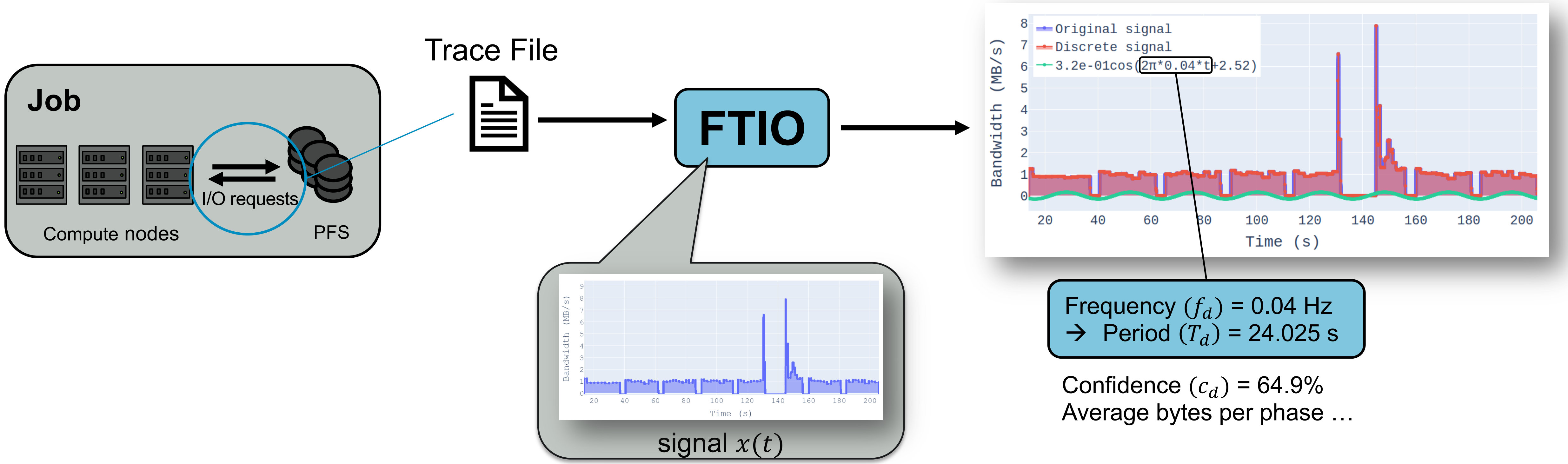
- Examine the I/O behavior in the **frequency domain** rather than the time domain
- Describes the temporal behavior of the I/O phases through a single metric, namely the **period** (T_d)
- Additional metrics quantify the **confidence** in the results and **further characterize** the I/O behavior based on the identified period
- Online (**prediction**) and offline (**detection**) realizations with a low overhead

Period (T_d) of I/O:
The time between the start of consecutive I/O phases



FTIO: Capturing Periodic Behavior

- **FTIO** treats I/O bandwidth over time as a **signal** $x(t)$



Frequency (f_d) = 0.04 Hz
→ Period (T_d) = 24.025 s

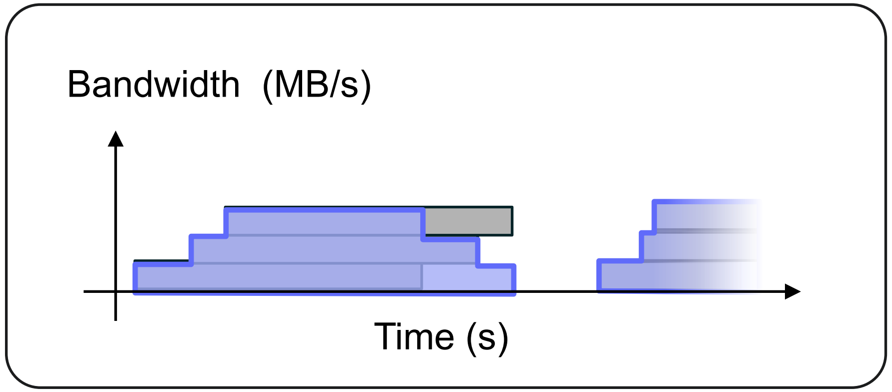
Confidence (c_d) = 64.9%
Average bytes per phase …

FTIO: Required Input

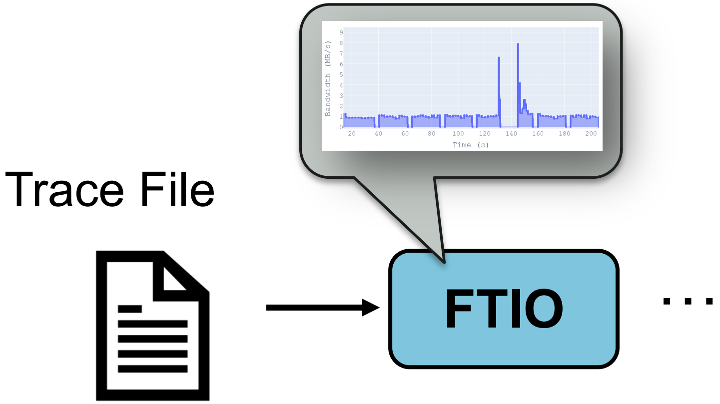
Trace file containing:

- Bandwidth per rank
- Time (start and end) when the bandwidth changed

→ FTIO calculates internally the **application-level bandwidth** by **overlapping** the rank-level metrics



- Application-level bandwidth and (start) time can also be provided directly
- Basically, any level is ok



FTIO: Required Input (cont')

Supported formats/tools for **online prediction**:

- **TMIO** (JSONL, MessagePack)
- ADMIRE Monitoring Proxy

Supported formats/tools for **offline detection**:

- Darshan
- Recorder (folder)
- TMIO (JSON, JSONL, MessagePack)
- ADMIRE Monitoring Proxy

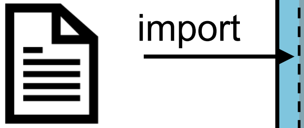
TMIO:

- Tracing **MPI-IO**
- C++ library that uses the PMPI interface
- Flushes I/O data online
- Can be easily attached to existing code
- Will be made publicly available

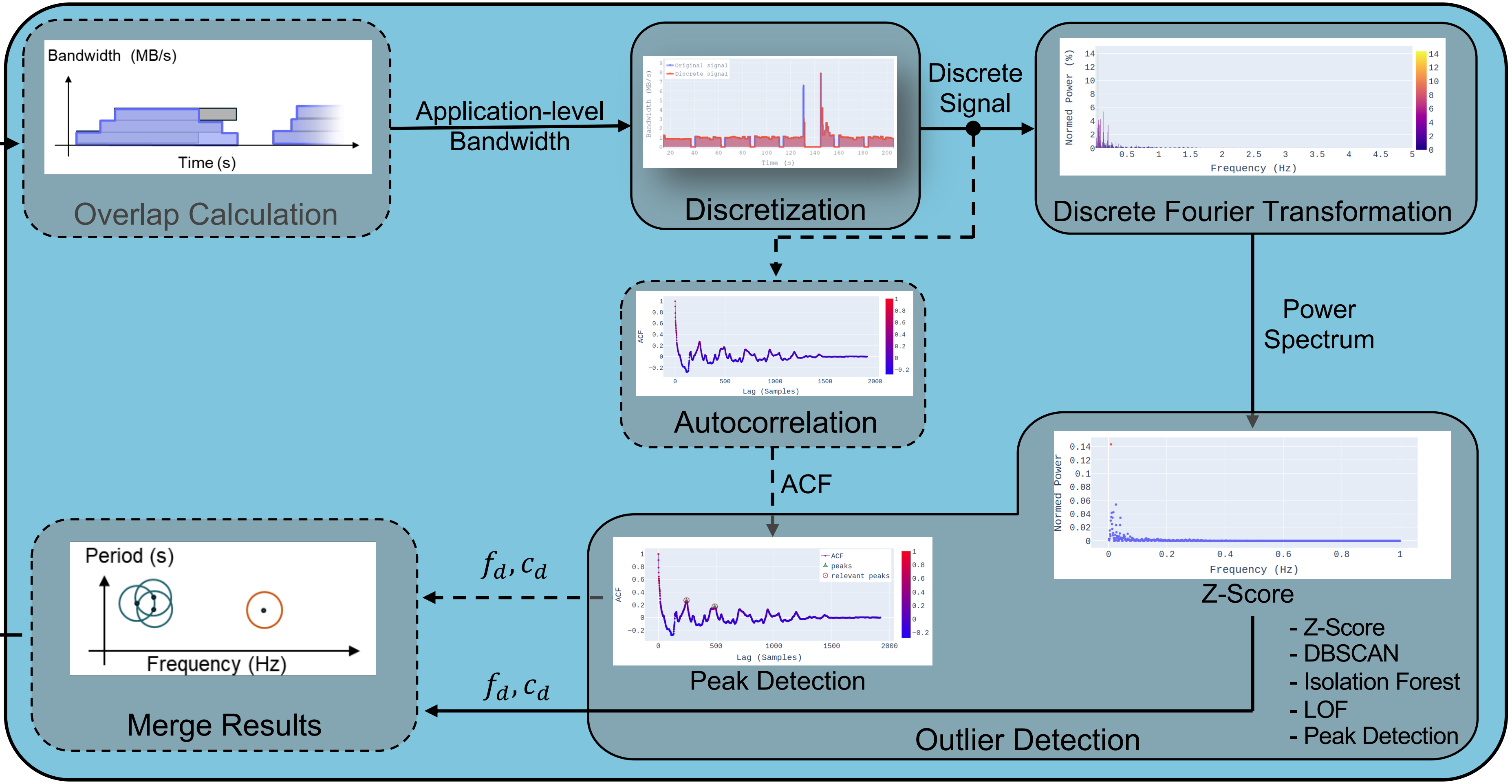
demo.json

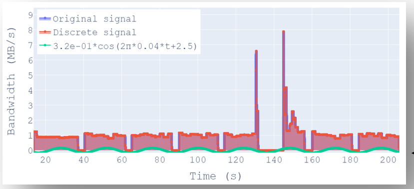
```
"bandwidth": {
    "b_rank_avr": [1.496276, 2.013454, 2.062243, ...],
    "t_rank_start": [1.950272, 1.964889, 1.975749, ...],
    "t_rank_e": [1.964871, 1.975739, 1.986342, ...]
}
```

FTIO: Approach



Trace file:
- Bandwidth per rank
- Bandwidth per node
- …



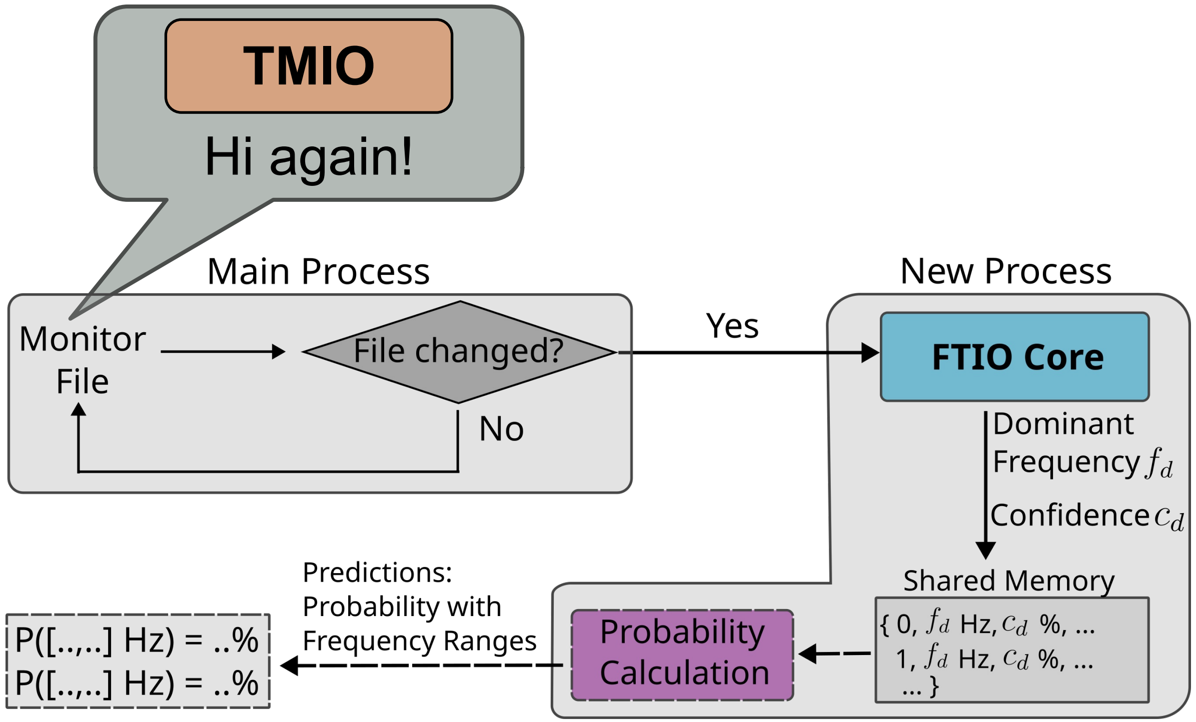


Frequency f_d
with a confidence c_d

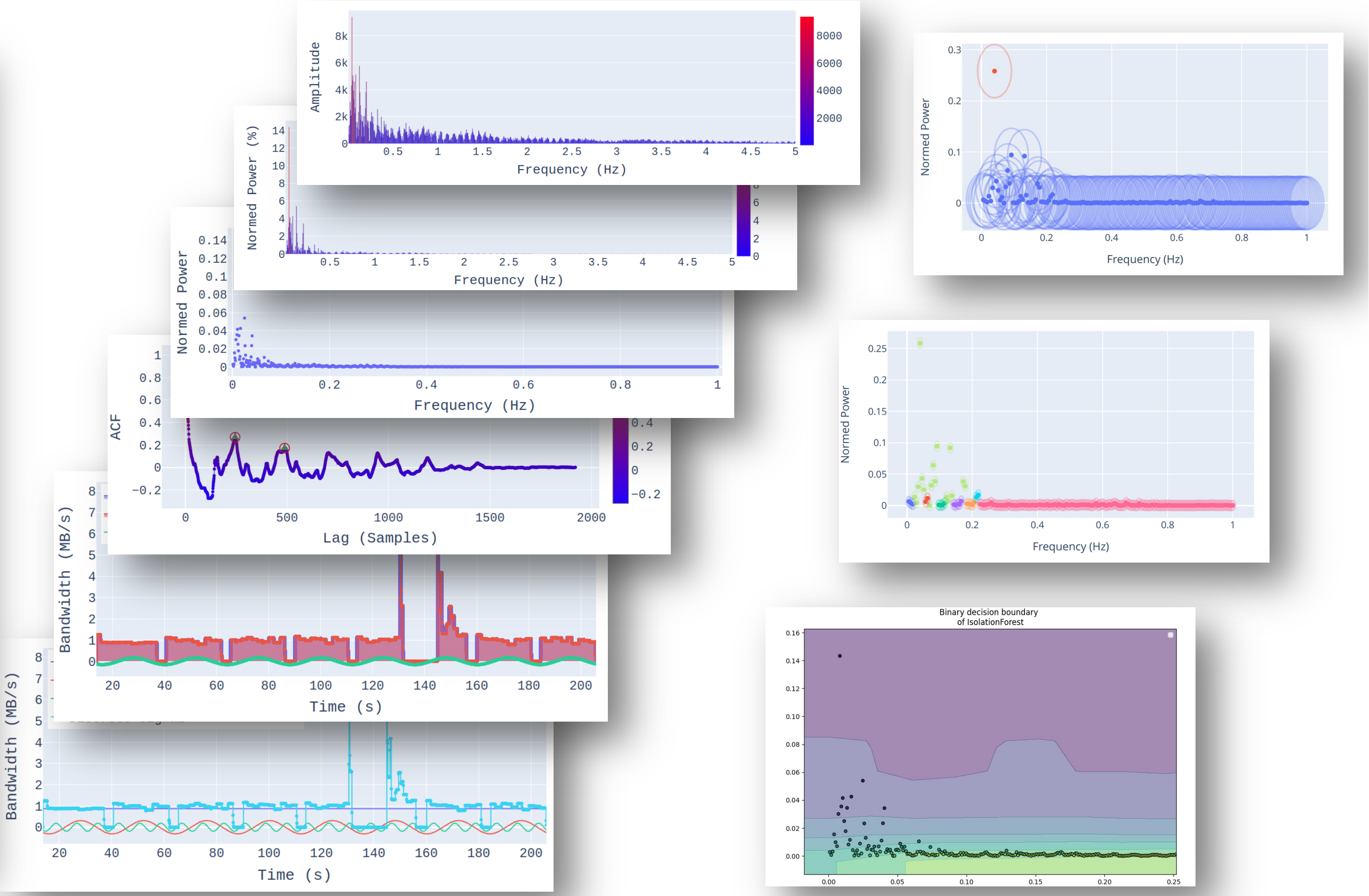
EuroHPC Joint Undertaking

FTIO: Online Version

- Predicts the period **during the execution** of an application
- Monitors a file for changes, whenever a changes is detected, a new prediction **process** is launched
- To adapt to **changing I/O behavior** we offer:
  1. Adapting time windows (discards the old data at some point); the width of the time window is determined by FTIO based on the found period
  2. Probability calculations with frequency intervals



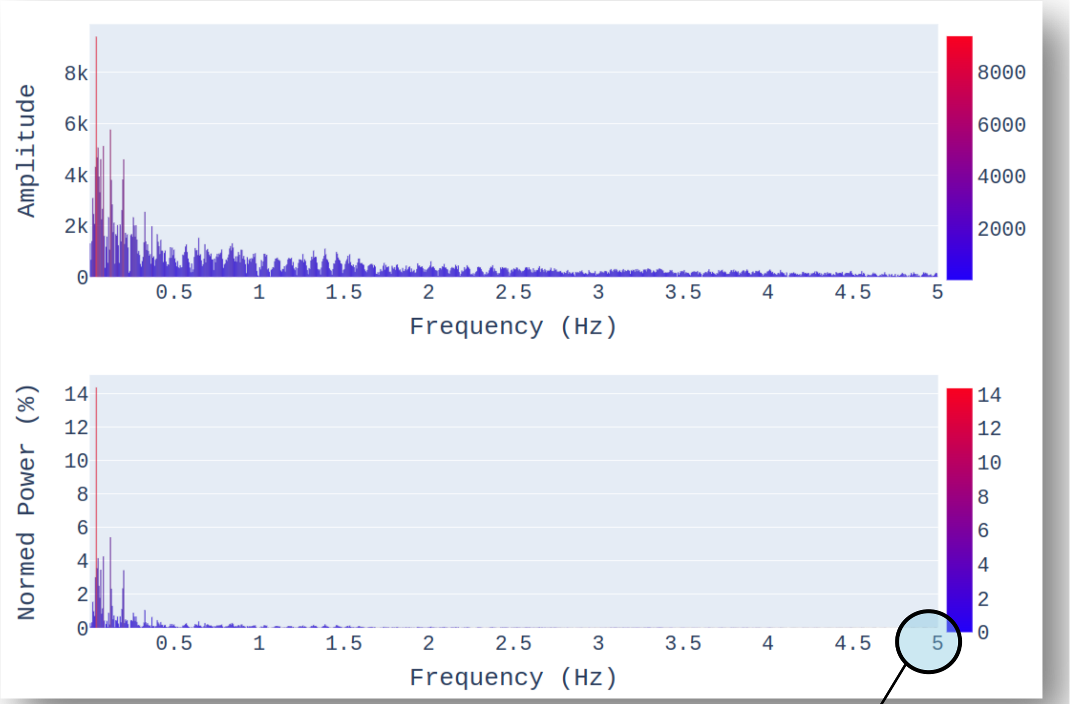
FTIO: User Interface



FTIO: In a Nutshell

Periodicity detection:

- **DFT + outlier detection** (Z-score, DB-Scan, Isolation forest, peak detection, or LOF)
- **Optionally: Autocorrelation + Peak detection**
- Merge results from both predictions (DB-Scan)

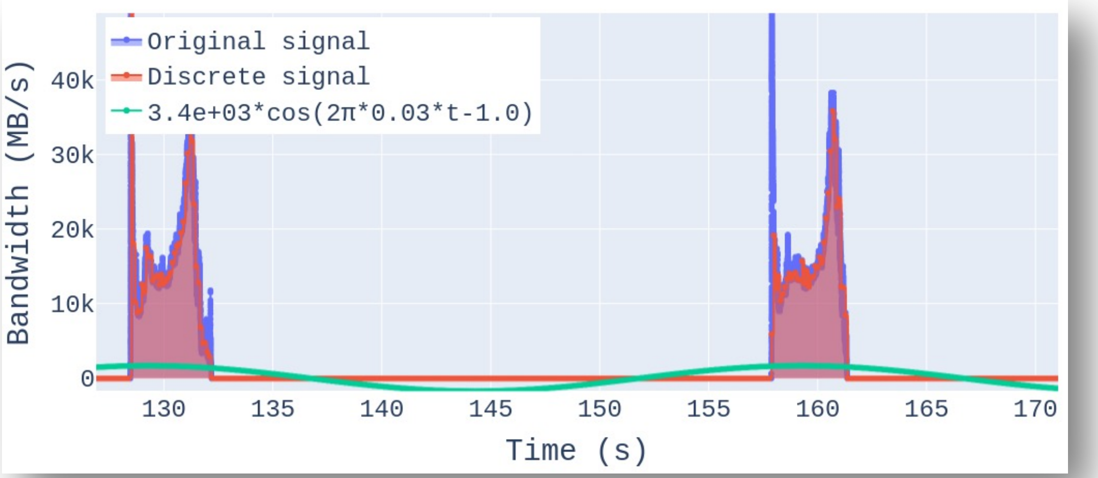


With $f_s = 10$ Hz, the limit is at $\frac{f_s}{2}$
→ DFT is symmetric, only half the spectrum is needed

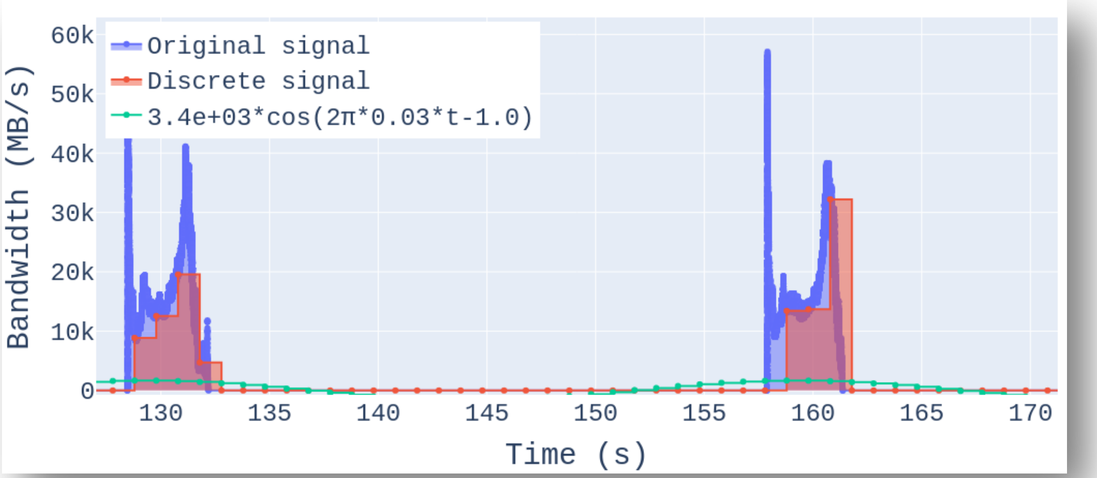
Properties

- Filters noise (e.g., using the power spectrum)
- **Several parameters to influence the accuracy** (sampling frequency f_s, time window Δt, and number of samples N)
- Two methods to adapt to changing behavior (probability calculation with frequency ranges or time window adaptation)
- Optimized Python code that uses true multiprocessing (pools or manual process creation)

Sampling Frequency



$f_s = 10\ Hz$



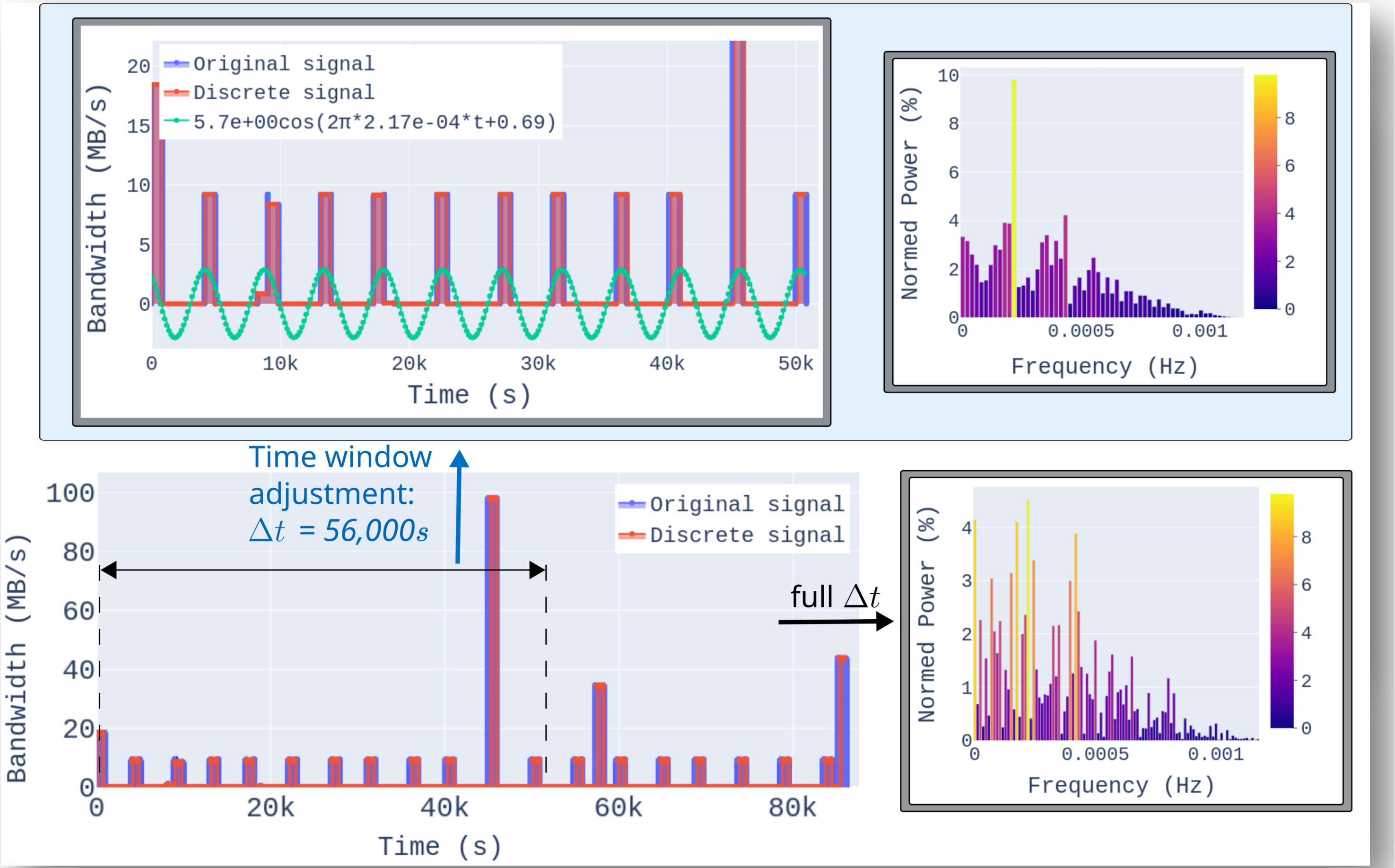
$f_s = 1\ Hz$

Sampling frequency **(f_s):**

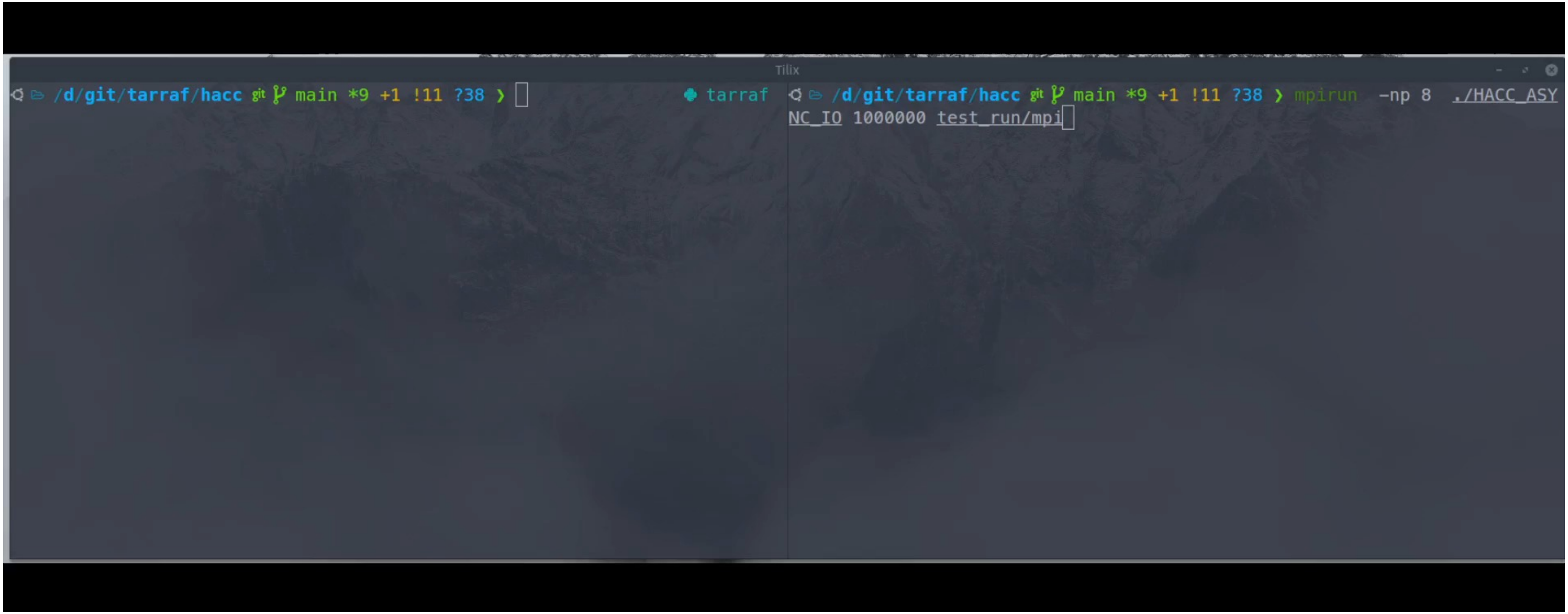
- Used to control the granularity at which the data is captured
- Specifies the range of frequencies of interest (Nyquist: $[0, \frac{f_s}{2}]$)

FTIO: Detection Example

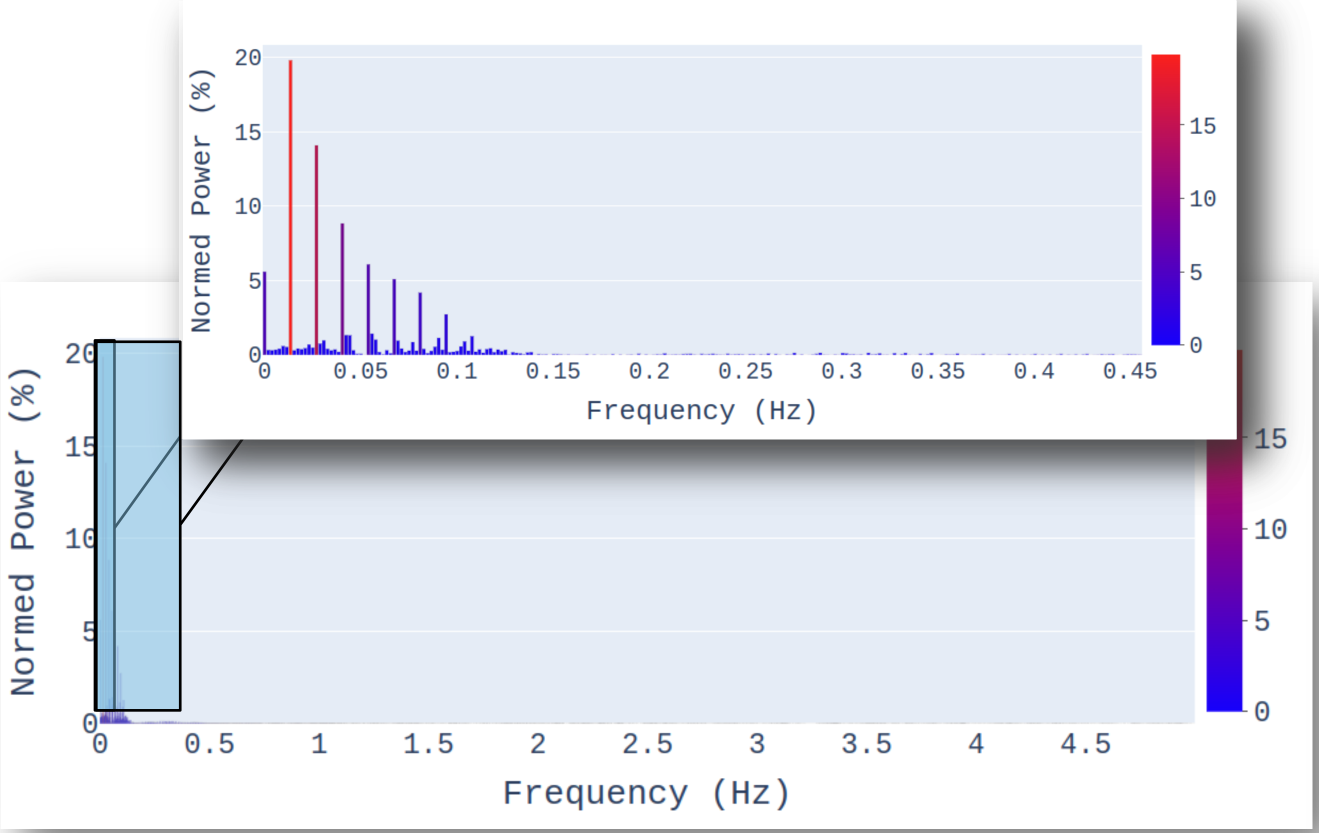
- Nek5000 with 2048 ranks on the Mogon II from the I/O trace website
- FTIO automatically $f_s = 0.006$ Hz (bin widths in seconds)
- FTIO detected I/O phases are not periodic for entire time window due to **irregular** I/O phases:
  - Phases at 0 s and 45,000 s write 13 and 75 GB, respectively
  - Phases at 57,000 s and 85,000 s write around 30 GB each
  - Other phases write 7 GB
- When time window changed to 56,000 FTIO detects a period of **4642.1 s** with a confidence of **85.4 %**



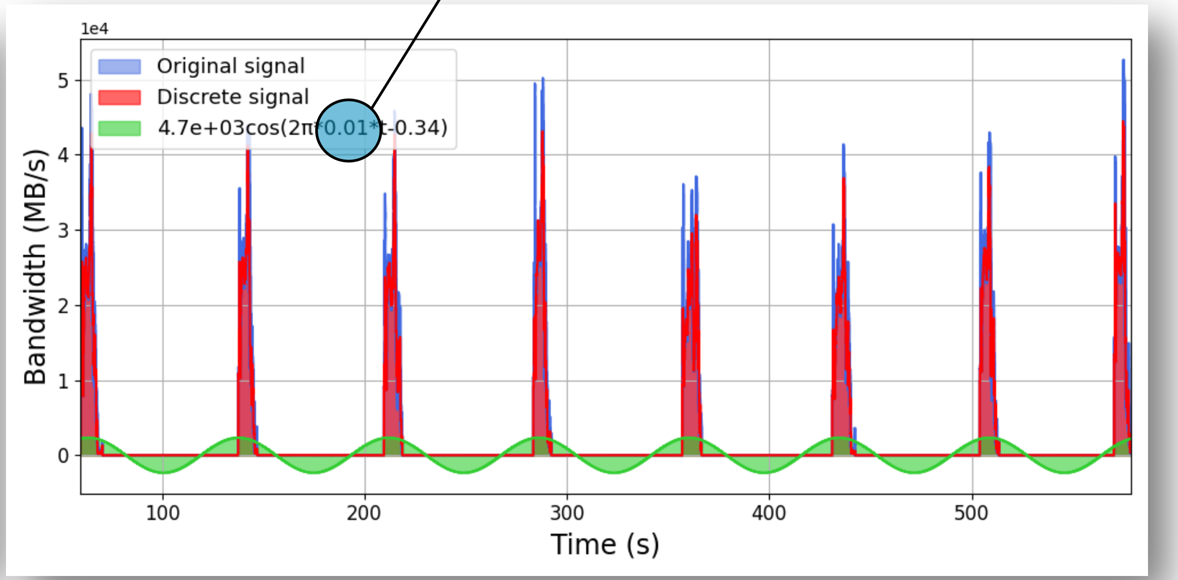
FTIO: Online Demo



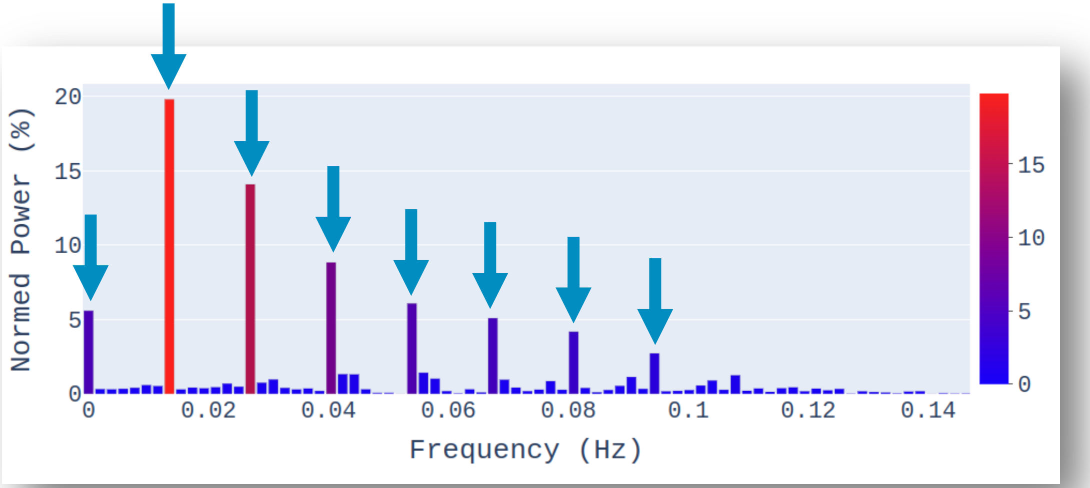
FTIO and I/O Bursts: IOR With 7680 Ranks

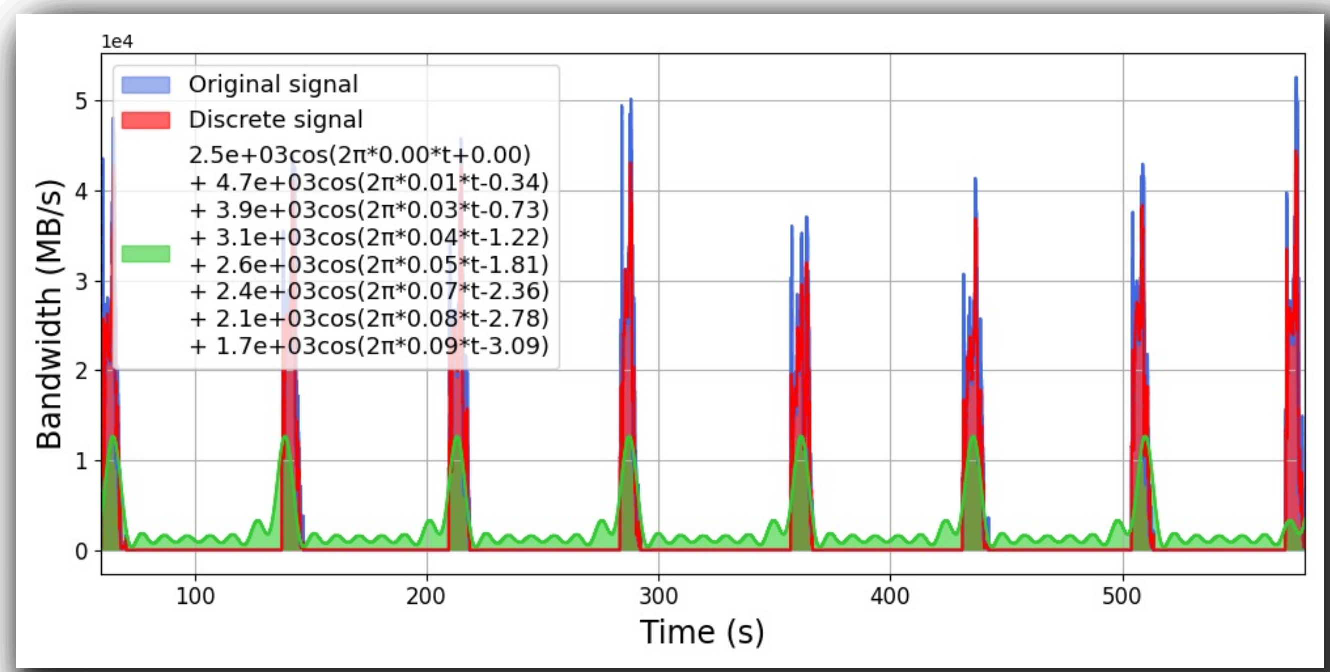


$f_d \cong 0.013\ Hz$
→Period = 74.18 s



FTIO and I/O Bursts: IOR With 7680 Ranks





FTIO Meets I/O-Sets

- **IO-Sets:** Method for I/O scheduling and the **Set-10** heuristic
  - F. Boito, G. Pallez, L. Teylo, and N. Vidal, IEEE TPDS 2023, https://inria.hal.science/hal-03648225
  - Places applications on classes according to their time between the start of consecutive I/O phases (w_iter), priority depends on class
  - Validated with simulation and a proof-of-concept implementation

- **FTIO:** Frequency techniques to characterize temporal I/O behavior
  - A. Tarraf, A. Bandet, F. Boito, G. Pallez, and F. Wolf, (under review), https://arxiv.org/abs/2306.08601
  - Finds the frequency (f_d) of I/O phases ($\frac{1}{period}$, or $\frac{1}{w_{iter}}$)

- **Set-10 implementation on BeeGFS servers**
  - C. Barthelemy, F. Boito, E. Jeannot, G. Pallez, and L. Teylo, unpublished for now
  - The BeeGFS client sends the application's priority together with each I/O request to the servers

FTIO Meets I/O-Sets: FTIO + Set-10

- The application runs: **TMIO** continuously appends to a trace
- **FTIO** in prediction mode watches over the trace of each application:
  - **Periodically outputs frequency and confidences**
  - A wrapper code, which called FTIO, recovers its output
  - Calculates priority according to Set-10 heuristic and writes it to a per-application /sys/kernel/config/ file
  - Before the first FTIO prediction, use a default value
  - **Whenever FTIO cannot answer (low confidence < 50%), keep the previously given priority**
- The BeeGFS client recovers the priority from the file when sending requests

FTIO Meets I/O-Sets: Experimental Methodology

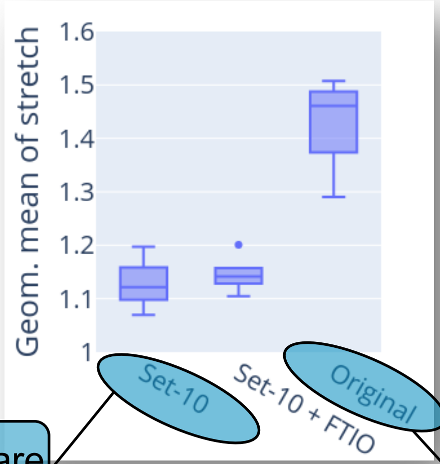
- Using the **Grid'5000 French infrastructure** (as we needed root access)
- BeeGFS with a single OSS and a single OST, writes to a local hard disk
- Applications are generated with IOR benchmarking tool
  - Used fsync option (to have stable performance without caching)
  - Used MPI-IO API with file-per-process write access
  - Modified IOR to get start and end timestamps of I/O phases (to calculate metrics) and included **TMIO**

- 16 applications, each with 8 processes, all on the same client node
  - 15 with low-frequency: 10 iterations of sleep (compute) for 360 s then write 320MB (period of ~384s)
  - 1 with high-frequency: 200 iterations of sleep (compute) for 18 s then write 16MB (period of ~19.2s)

- Basically, we recreated an experiment from the **IEEE TPDS paper where Set-10 had excellent results** (while adapting to a different platform)

FTIO Meets I/O-Sets: Results

Decreased by 20 %

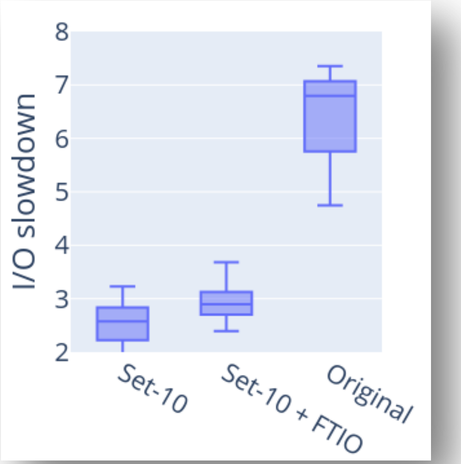


Priorities are hardcoded

BeeGFS without modifications

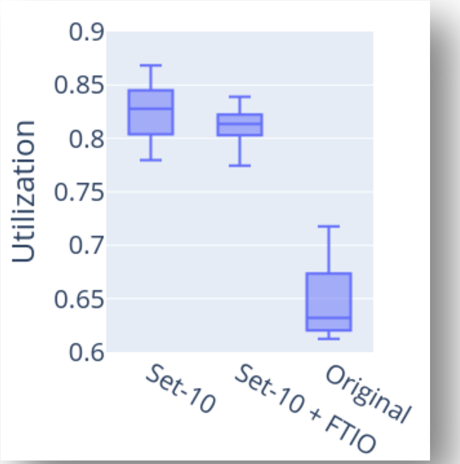
The lower, the better

Decreased by 54 %



The higher, the better

Increased by 25 %



Stretch:

For each application, **how much it was slowed-down by others** compared to running by itself (minimum of 1, meaning no slow down). We take the geometric mean of the 16 applications.

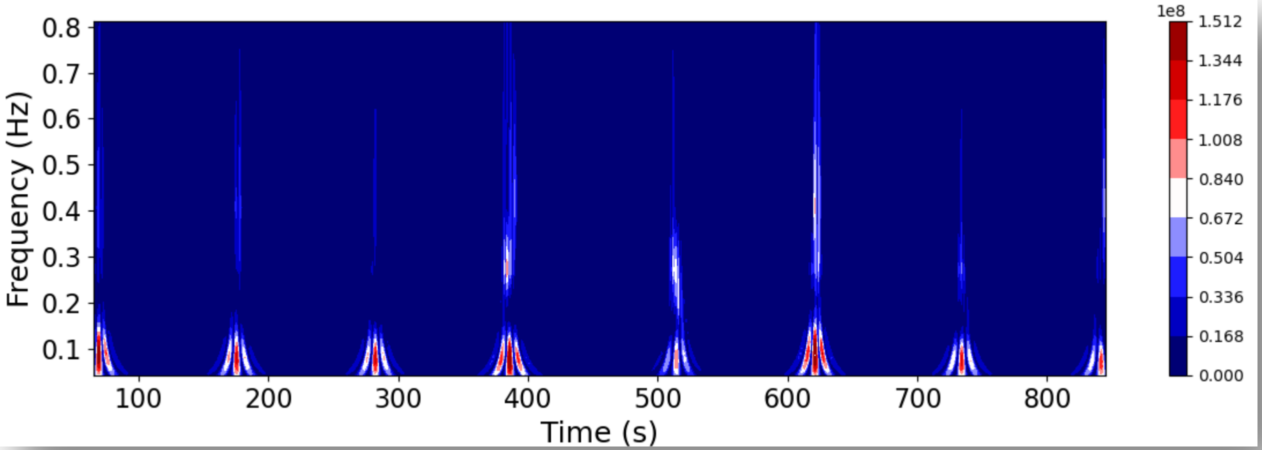
IO-Slowdown:

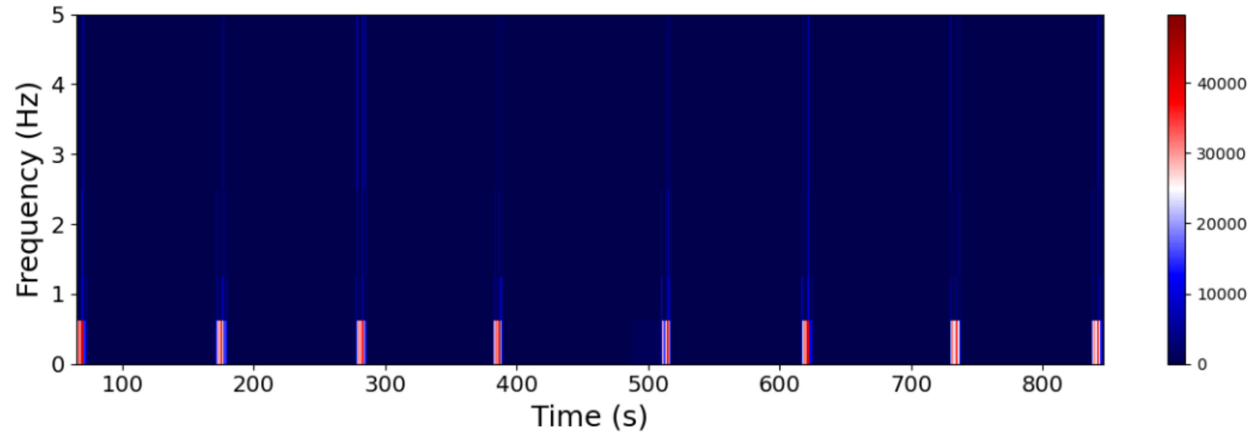
For each application, **how much slower its I/O was compared to running by itself** (minimum of 1, meaning no slow down). We take the geometric mean of the 16 applications.

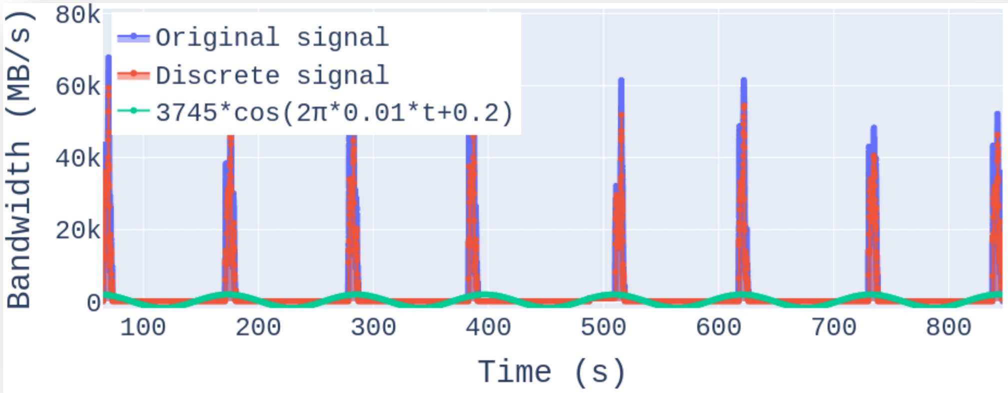
Utilization:

How much of the system **time was spent on compute** (NOT doing I/O or waiting for I/O), so between 0 and 1 (1 means no I/O at all).

Extension: Wavelet Transform



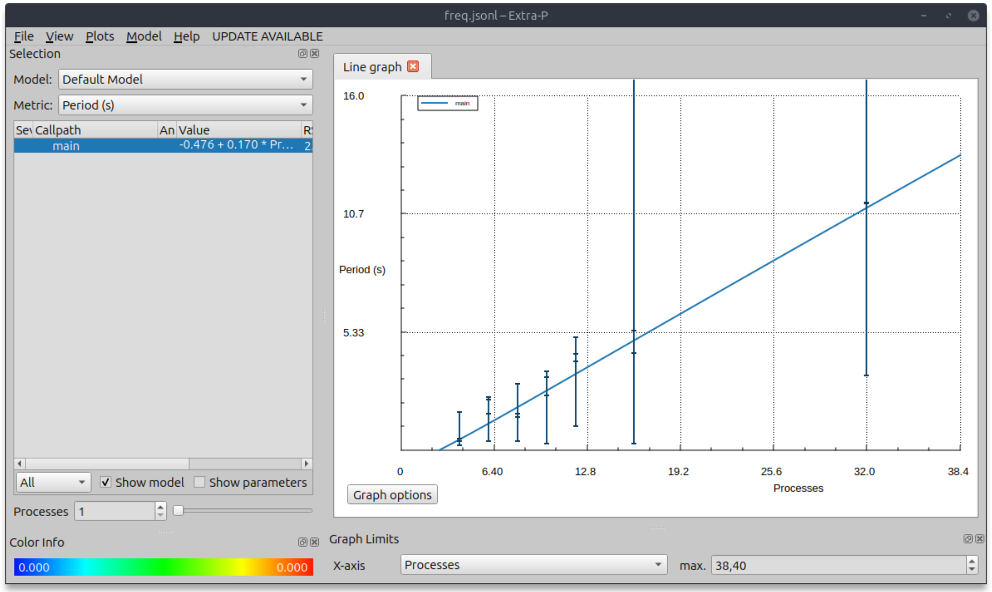




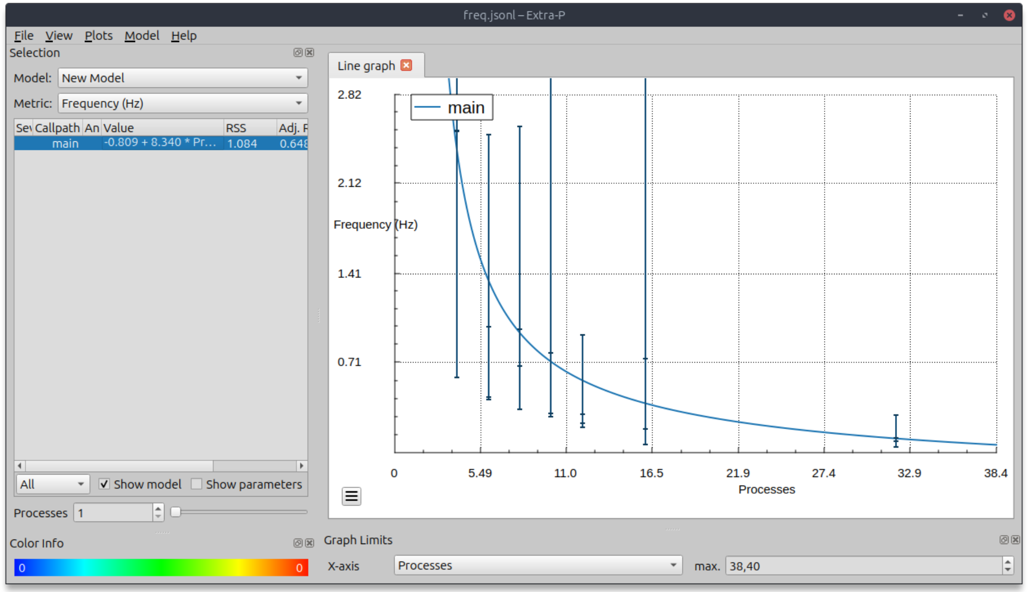
IOR with 9216 ranks executed on the Lichtenberg cluster

FTIO Meets Extra-P:

- First steps to generate performance models for the periods of the I/O phases:



Period vs number of processes



Frequency vs number of processes

Conclusion

- An approach to characterize and predict the temporal I/O behavior of an application with a simple metric: its period, obtained using DFT
- Additional metrics describe the **confidence** in the results and allow for further characterization
- **Online** and **offline** realization
- Several **parameters** can be changed to enhance the results obtained

Will be made publicly available in GitHub very soon

Contact: ahmad.tarraf@tu-darmstadt.de

References

1. Anne Benoit, Thomas Herault, Lucas Perotin, Yves Robert, and Frédéric Vivien. 2023. Revisiting I/O bandwidth-sharing strategies for HPC applications. Technical Report RR-9502. INRIA. 56 pages. https://hal.inria.fr/hal-04038011
2. Matthieu Dorier, Gabriel Antoniu, Rob Ross, Dries Kimpe, and Shadi Ibrahim. 2014. CALCioM: Mitigating I/O interference in HPC systems through crossapplication coordination. In IPDPS'14. IEEE, 155–164.
3. Emmanuel Jeannot, Guillaume Pallez, and Nicolas Vidal. 2021. Scheduling periodic I/O access with bi-colored chains: models and algorithms. J. of Scheduling 24, 5 (2021), 469–481.

ADMIRE
malleable data solutions for HPC

Acknowledgment

- We acknowledge the support of the European Commission and the German Federal Ministry of Education and Research (BMBF) under the EuroHPC Programmes DEEP-SEA (GA no. 955606, BMBF funding no. 16HPC015) and ADMIRE (GA no. 956748, BMBF funding no. 16HPC006K), which receive support from the European Union’s Horizon 2020 programme and DE, FR, ES, GR, BE, SE, GB, CH (DEEP-SEA) or DE, FR, ES, IT, PL, SE (ADMIRE). This work was also supported by the French National Research Agency (ANR) in the frame of DASH (ANR-17-CE25-0004), by the Project Région Nouvelle Aquitaine 2018-1R50119 “HPC scalable ecosystem”

- We gratefully acknowledge the computing time provided on the high-performance computer Lichtenberg at the NHR Centers NHR4CES at TU Darmstadt. This is funded by the Federal Ministry of Education and Research, and the state governments participating on the basis of the resolutions of the GWK for national high performance computing at universities (*www.nhr-verein.de/unsere-partner*). Moreover, we also gratefully acknowledge the computing time on the PlaFRIM experimental testbed supported by Inria, CNRS (LABRI and IMB), Université de Bordeaux, Bordeaux INP and Conseil Régional d’Aquitaine (https://www.plafrim.fr)

ADMIRE malleable data solutions for HPC | EuroHPC Joint Undertaking | Federal Ministry of Education and Research | Agence Nationale de la Recherche ANR | DEEP-SEA

EuroHPC Joint Undertaking

Thank you for your attention!

Questions?

Experiment Platform: Lichtenberg Cluster

Executed were executed Lichtenberg cluster:

- 8 login nodes and 643 compute nodes
- MPI section of the cluster hosts 630 nodes each with 96 CPU cores and 384 GB main memory
- Shared file system (IBM Spectrum Scale)
- Peak performance of 106 GB/s for writes and 120 GB/s for reads
- File system is shared (no exclusive access), while the compute nodes have user-exclusive access