Frequency Techniques for I/O

Ahmad Tarraf[1], Alexis Bandet[2], Francieli Boito[2], Guillaume Pallez[2], and Felix Wolf[1]

[1]Technical University of Darmstadt
[2]University of Bordeaux, INRIA

TECHNISCHE UNIVERSITÄT DARMSTADT
Parallel Programming

ADMIRE
malleable data solutions for HPC
ADAPTIVE MULTI-TIER INTELLIGENT DATA MANAGER FOR EXASCALE

DEEP-SEA

EuroHPC Joint Undertaking

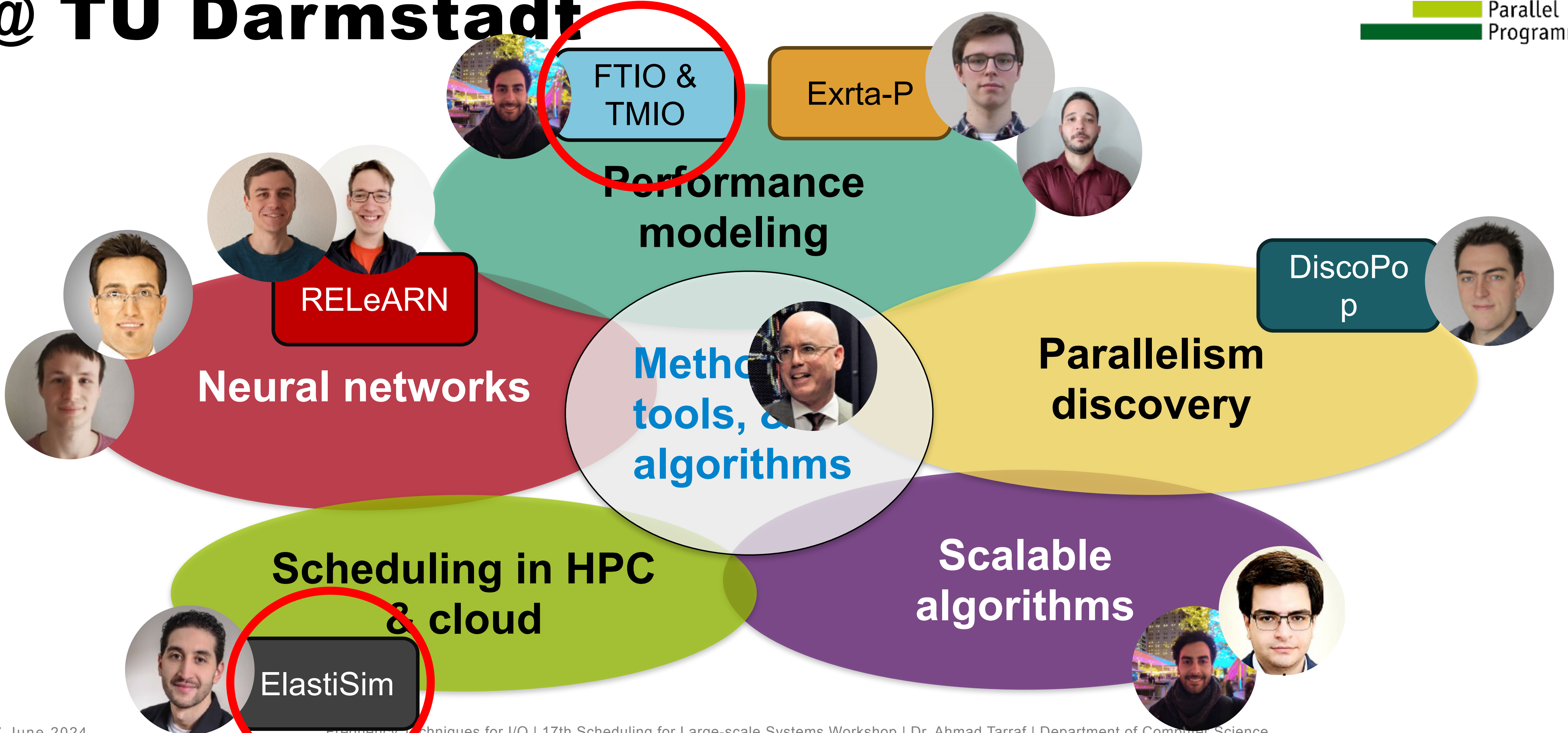
17th Scheduling for large-scale systems workshop

Image source: https://fr.ski-france.com/media/cache/gallery_default/221049-Aussois-Vue-nocturne.jpg

Parallel Programming @ TU Darmstadt

TECHNISCHE UNIVERSITÄT DARMSTADT

Parallel Programming



Survey on Malleability in HPC

TECHNISCHE UNIVERSITÄT DARMSTADT

Parallel Programming

Open Access

Collaboration between 4 EuroHPC projects over 2 years

Preprint available on IEEE

IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, VOL. 35, NO. 6, JUN 2024 1

Malleability in Modern HPC Systems: Current Experiences, Challenges, and Future Opportunities

Ahmad Tarraf[1], Martin Schreiber[2,3], Alberto Cascajo[4], Jean-Baptiste Besnard[5], Marc-André Vef[6], Dominik Huber[3], Sonja Happ[7], André Brinkmann[6], David E. Singh[4], Hans-Christian Hoppe[8], Alberto Miranda[9], Antonio J. Peña[9], Rui Machado[10], Marta Garcia-Gasulla[9], Martin Schulz[3], Paul Carpenter[9], Simon Pickartz[7], Tiberiu Rotaru[10], Sergio Iserte[9], Victor Lopez[9], Jorge Ejarque[9], Heena Sirwani[10], Jesus Carretero[4], Felix Wolf[1]

Abstract—**With the increase of complex scientific simulations driven by workflows and heterogeneous workload profiles, man-**

increase or decrease during its execution and resources added or released accordingly. Malleable jobs can also increase en-

ElastiSim: A Batch-System Simulator for Malleable and Evolving Workloads

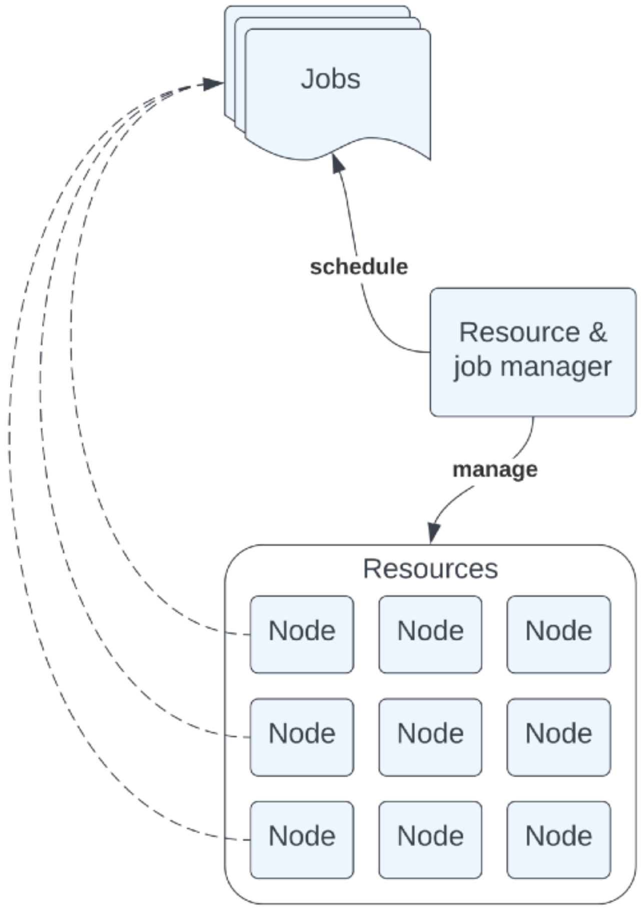
TECHNISCHE UNIVERSITÄT DARMSTADT
Parallel Programming

Resource & job management

TECHNISCHE UNIVERSITÄT DARMSTADT

Parallel Programming

- Resource and job management systems (also often called batch systems) schedule jobs and provide resources in large-scale computing environments
- Depending on the objective, batch systems aim to maximize system efficiency and decrease job completion times
- Scheduling algorithms are key components to improve system performance

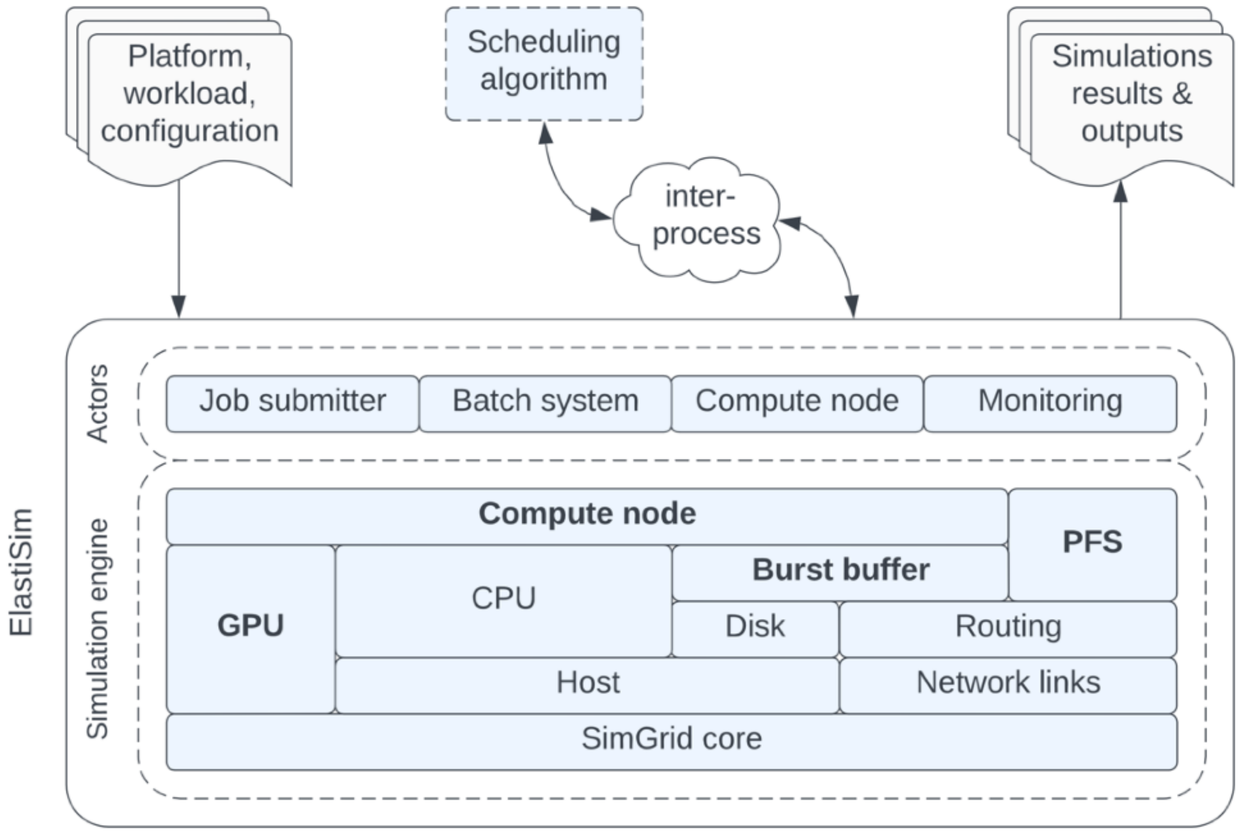


What is ElastiSim?

- ElastiSim is a simulator that simulates
  - jobs and applications,
  - the batch system supporting rigid, moldable, malleable, and evolving workloads,
  - the scheduling algorithm (as part of the batch system),
  - the platform (powered by SimGrid).

- Typical use case: evaluating algorithms for the combined scheduling of rigid, moldable, malleable, and evolving jobs

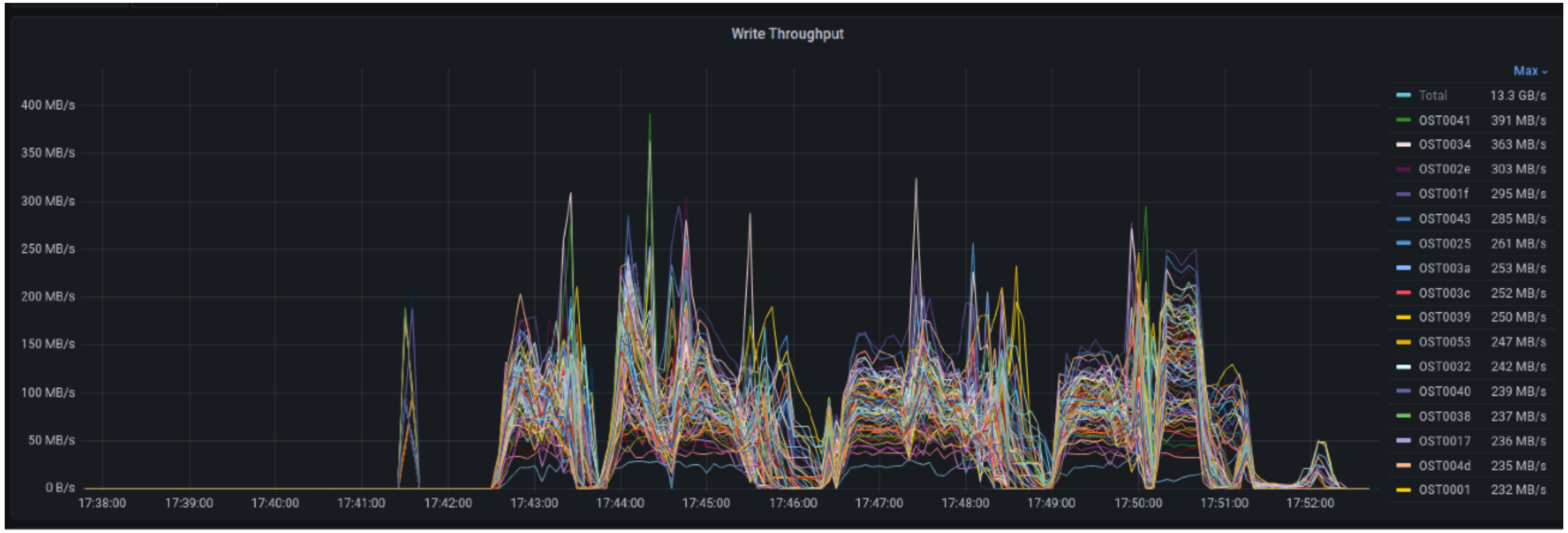
ElastiSim architecture



Summary

- Website: https://elastisim.github.io (includes Slack invitation)
- GitHub: https://github.com/elastisim
- Contact me (Taylan Özden): taylan.oezden@tu-darmstadt.de

Frequency Techniques for I/O



Motivation

- HPC applications usually alternate between compute and I/O phases (e.g., Checkpointing)
- Compute resources are allocated **exclusive**
- **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')

TECHNISCHE UNIVERSITÄT DARMSTADT

Parallel Programming

- 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**
- 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?

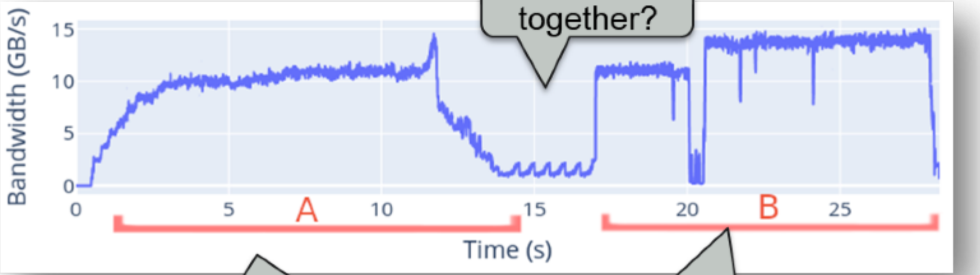
Is it compatible to share them with others?

Should we use burst buffers?

Use for I/O scheduling?

- **Difficult to get this information online with a low overhead!**
- **Especially, if we think in terms of phases!**
  - I/O phases composed of many I/O requests
  - Not all I/O is interesting
  - Boarders of I/O phases? Threshold?

→Application and system



"A" and "B" together?

Where does "A" finish

"B" one or two phases?

FTIO: Frequency Techniques for I/O

TECHNISCHE UNIVERSITÄT DARMSTADT

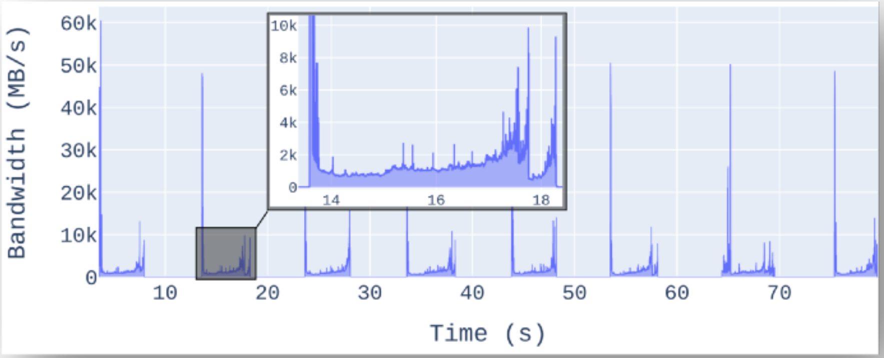
Parallel Programming

→ 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]

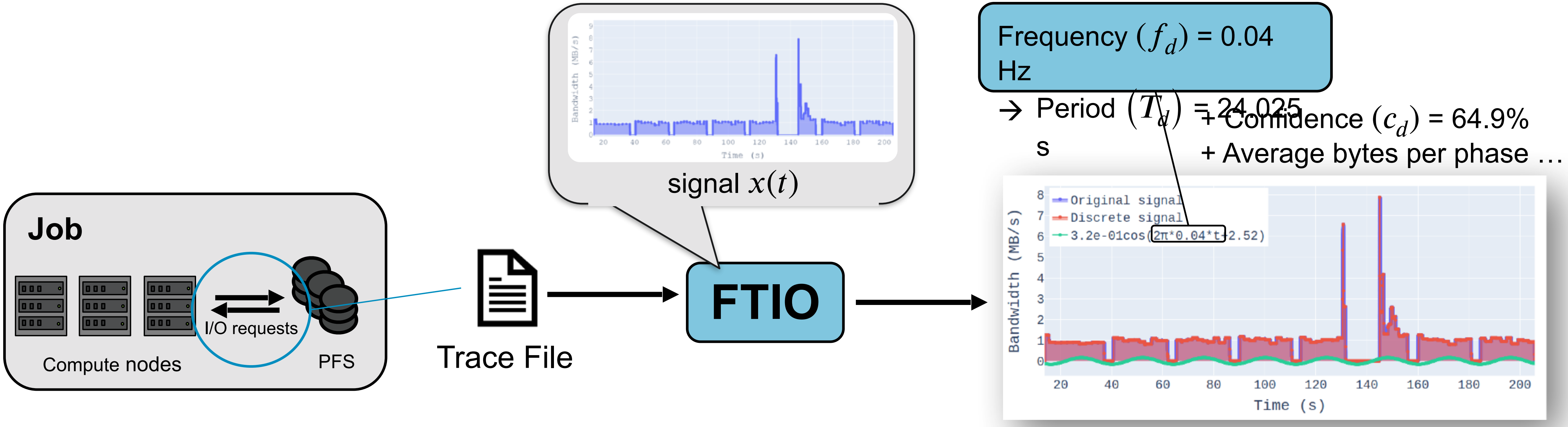
→Frequency Techniques for I/O:

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



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

FTIO: In a Nutshell



FTIO: Required Input

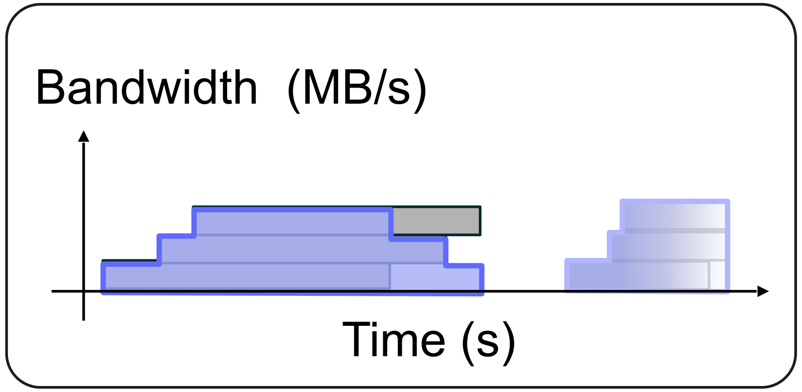
TECHNISCHE UNIVERSITÄT DARMSTADT

Parallel Programming

Trace file containing:

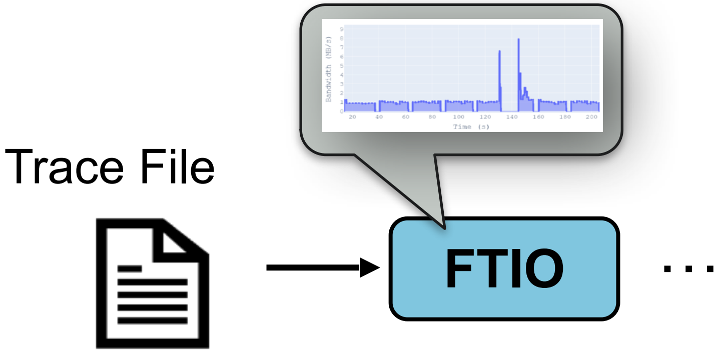
- Bandwidth per rank
- Time (start and end) when the bandwidth changed

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



Any metric can be predicted, not only I/O!
E.g., network consumption, scheduling points, energy, etc.

Application-level bandwidth and (start) time can also be provided directly → Any level is ok!



FTIO: Required Input (cont')

Supported Formats/Tools for **online prediction**:

- **TMIO** (JSONL, MessagePack, ZeroMQ)
- ADMIRE Monitoring Proxy (ISC24) new

Supported Formats/Tools for **offline detection**:

- Darshan
- Recorder (folder)
- TMIO (JSON, JSONL, MessagePack, …Q) new
- ADMIRE Monitoring Proxy (ISC24)
- Custom file format

new

TMIO:

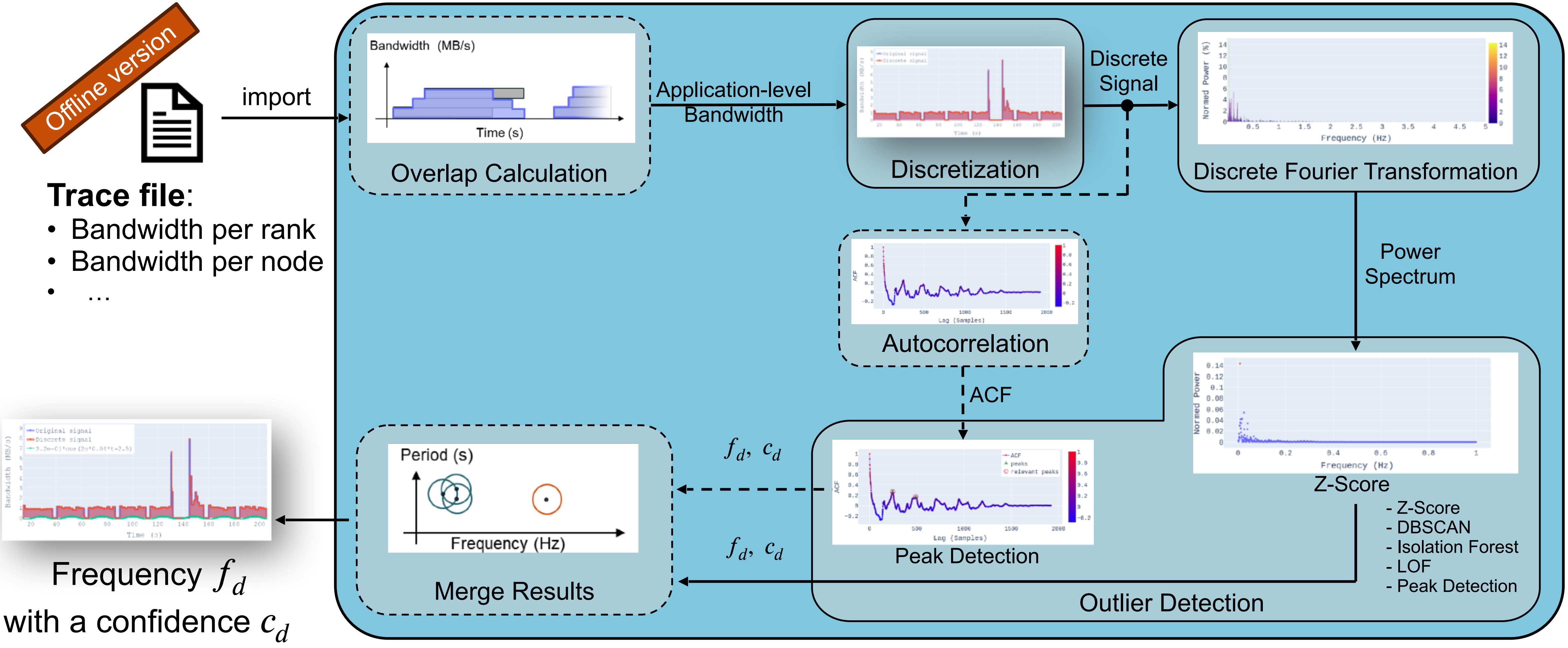
- **T**racing **MPI-IO**
- C++ library that uses the PMPI interface
- Flushes I/O data online
- Can be easily attached to existing code
- Will be made publicly available

demo.json

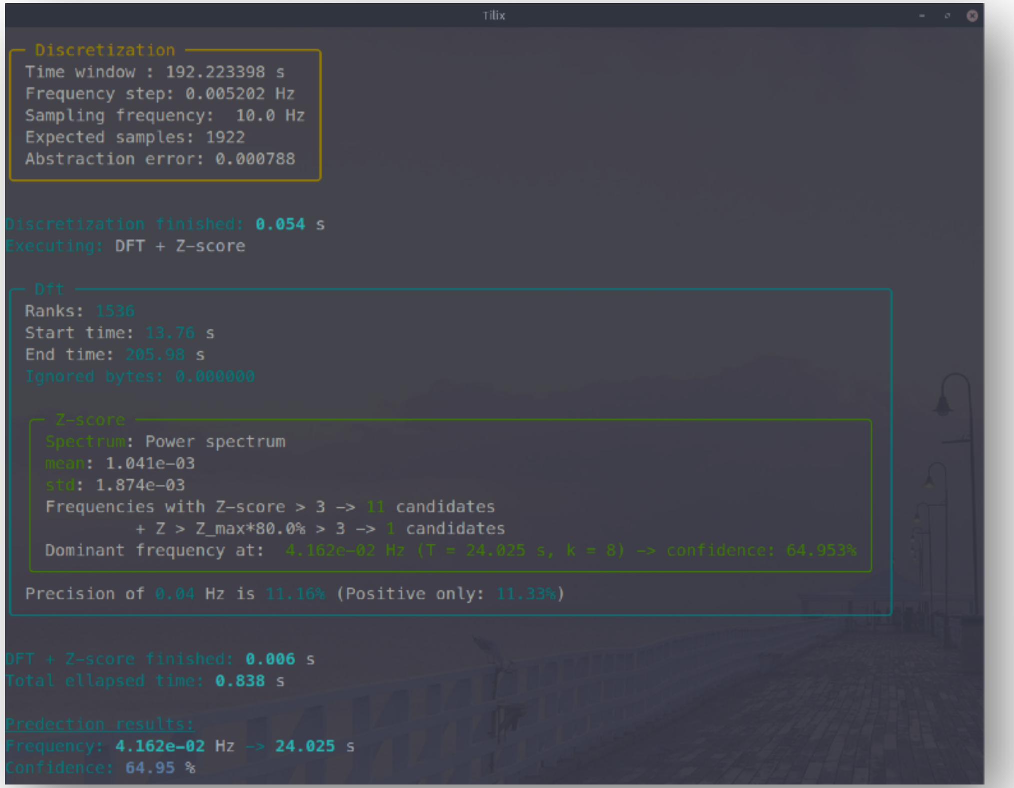
```
"bandwidth": {
    "b_rank_avr": [1.496276, 2.013454, 2.062243, ...],
    "t_rank_start": [1.950272, 1.964889, 1.975749, ...],
    "t_rank_e": [1.964871, 1.975739, 1.986342, ...]
}
```

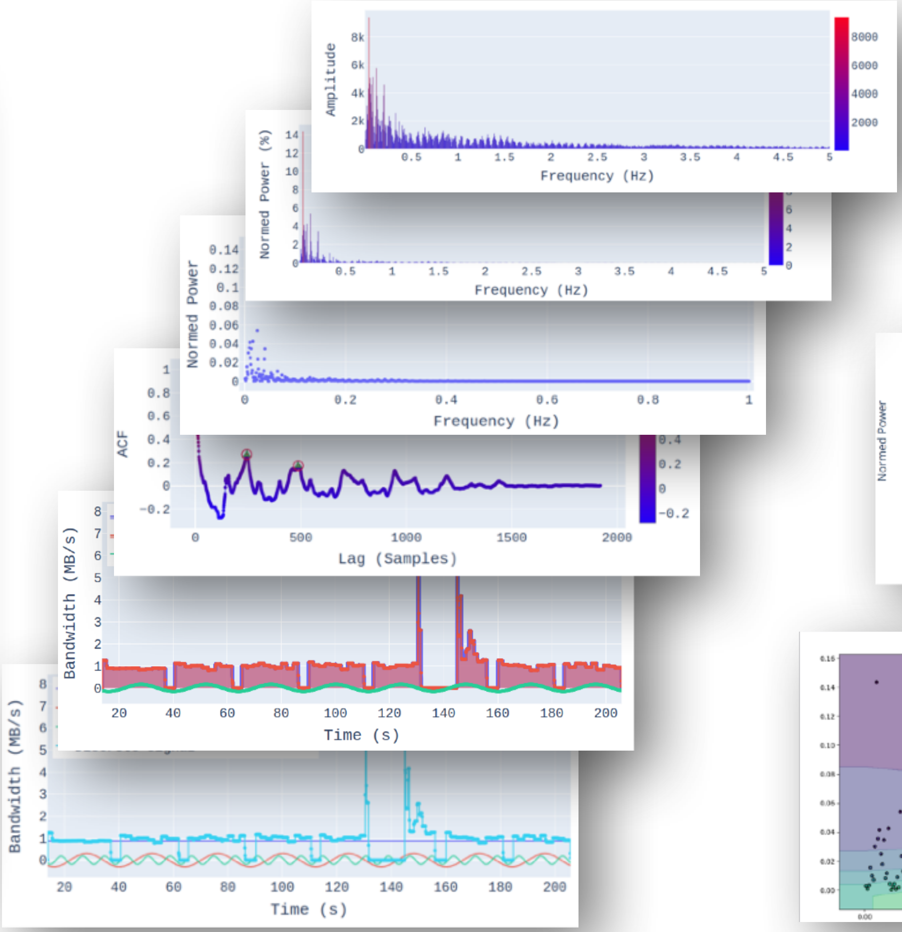
https://github.com/tuda-parallel/FTIO/blob/main/docs/file_formats.md

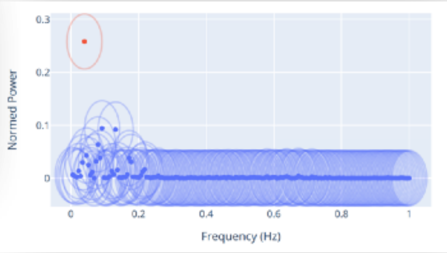
FTIO: The Core

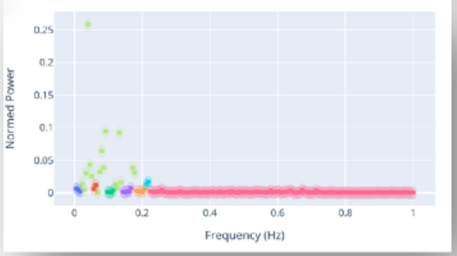


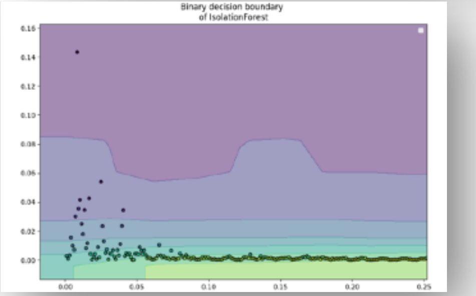
FTIO: User Interface











FTIO: Summary

TECHNISCHE UNIVERSITÄT DARMSTADT

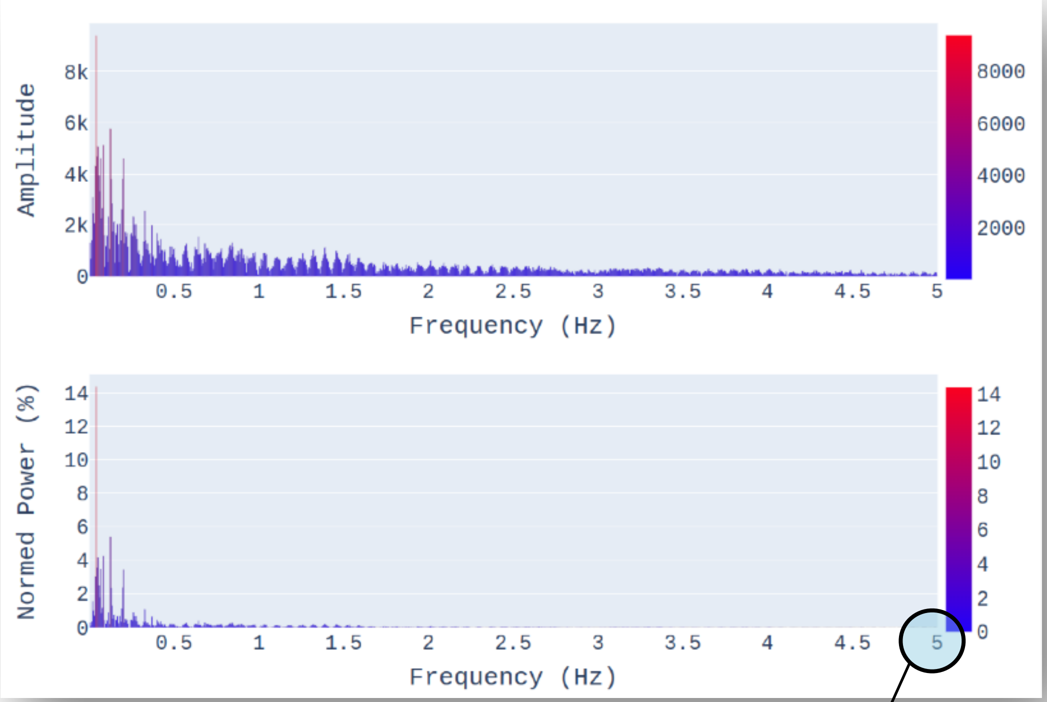
Parallel Programming

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)

Properties

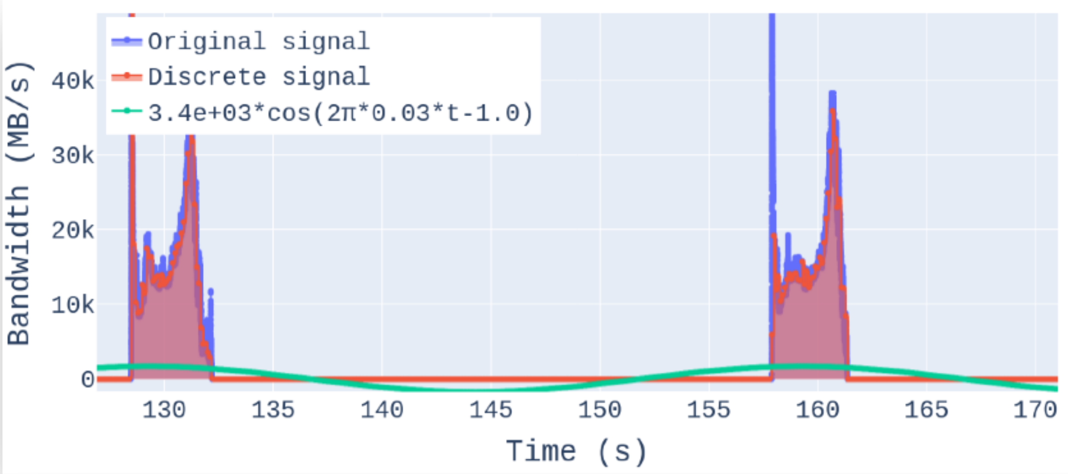
- Filters noise (e.g., using the power spectrum)
- **Several parameters control the accuracy** (sampling frequency f_s, time window Δt, and number of samples N)
- **Online version** offers two methods to adapt to changing behavior
- Optimized Python code that uses true multiprocessing (pools or manual process creation)



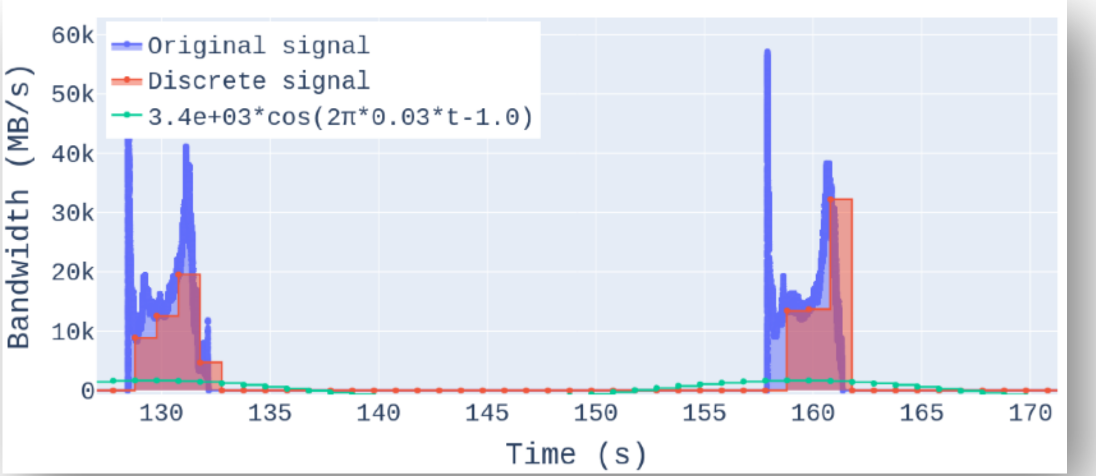
With $f_s = 10$ Hz, the limit is at $\frac{f_s}{2}$

→ DFT is symmetric, only half the spectrum is needed

FTIO: 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

TECHNISCHE UNIVERSITÄT DARMSTADT

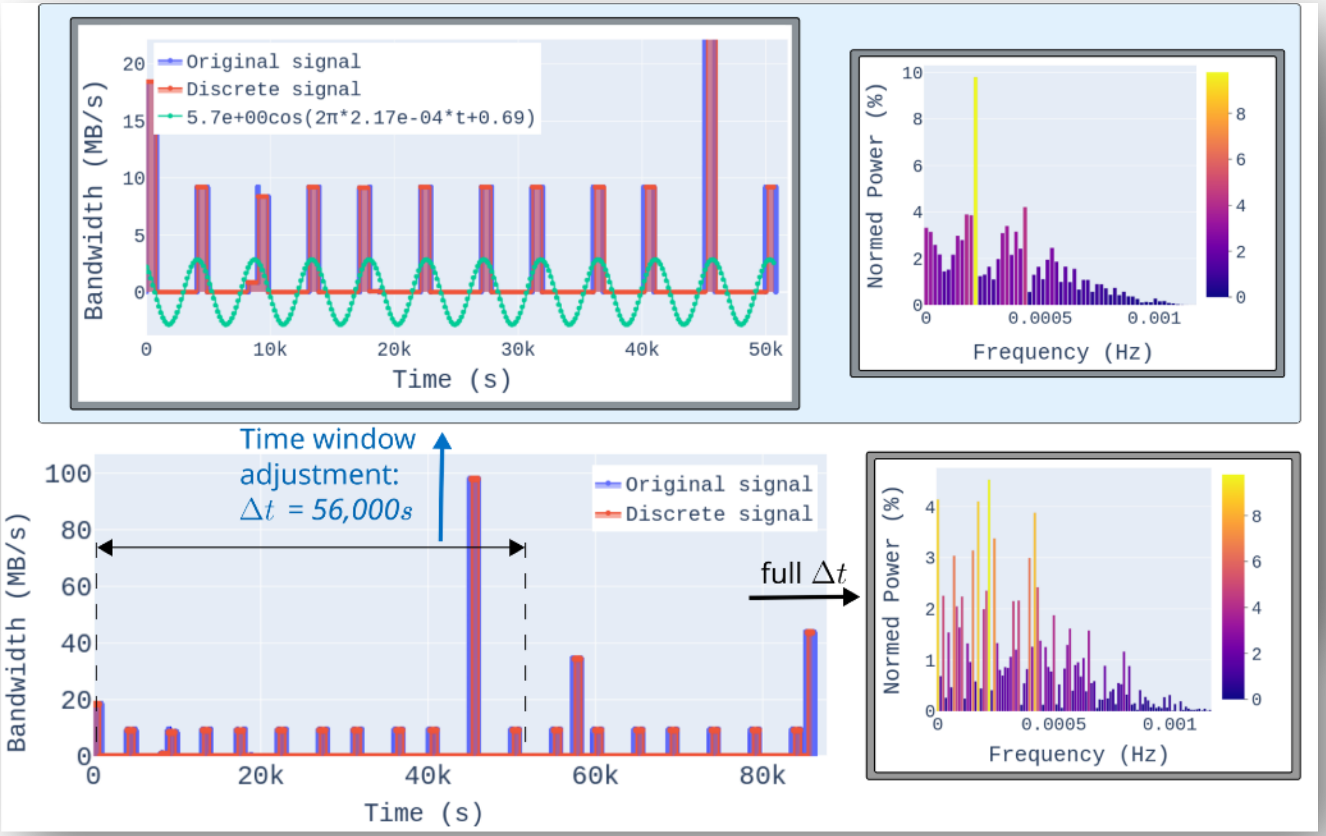
Parallel Programming

- Nek5000 with 2048 ranks on the Mogon II (downloaded from the [HPCIO analysis website]())

- FTIO automatically sets f_s to 0.0006 Hz (bin widths in seconds)

- FTIO detected I/O phases are **not periodic** in the full-time window due to **irregular** I/O phases:
  - Phases at 0 s and 45,000 s write 13 and 75 GB
  - Phases at 57,000 and 85,000 s write ~30 GB each
  - Other phases write 7 GB

→ When Δt is set to 56,000 s, FTIO detects a dominant period of **4642.1 s** with **85.4 %** confidence



FTIO: Online Version

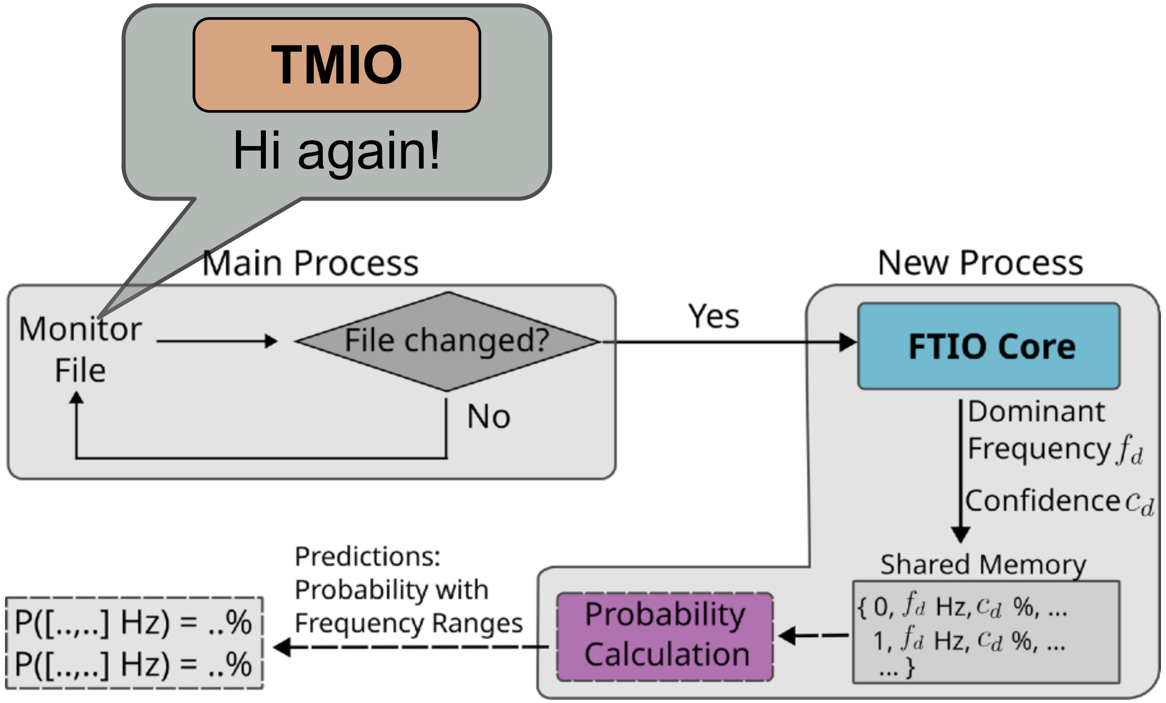
TECHNISCHE UNIVERSITÄT DARMSTADT

Parallel Programming

- 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** FTIO offers:

1. Adapting time windows (discards the old data at some point); the width of the time window is found by FTIO based on the found period
2. Probability calculations with frequency intervals

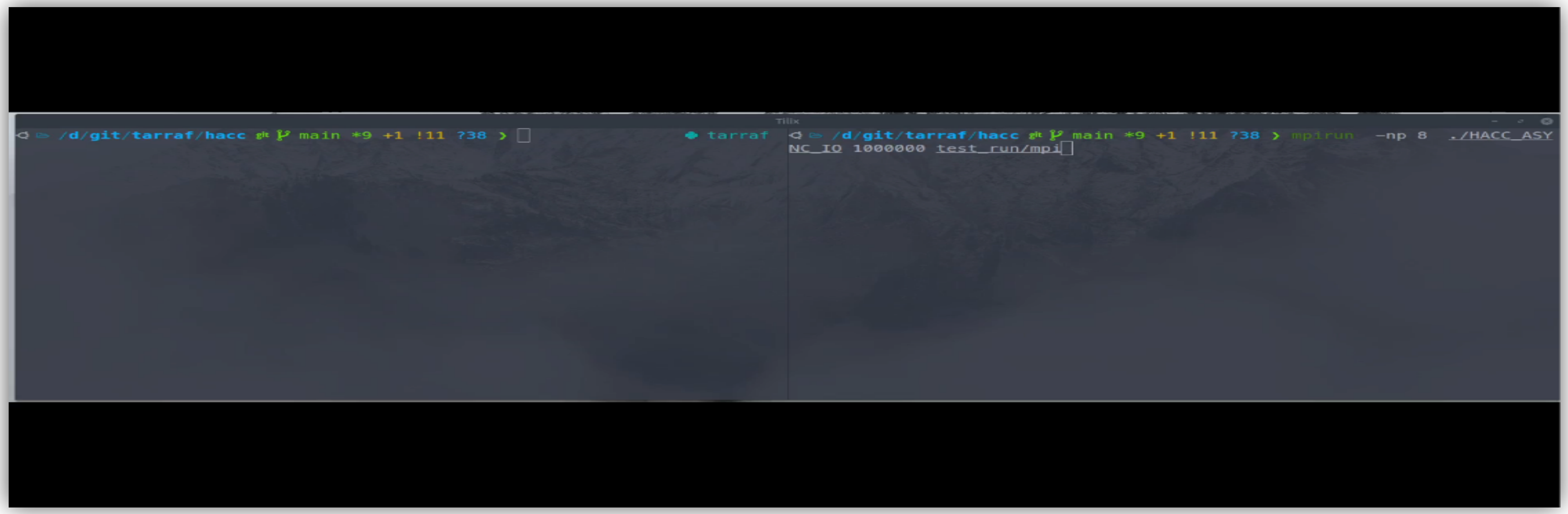


NEW: Support for TCP messages using ZeroMQ

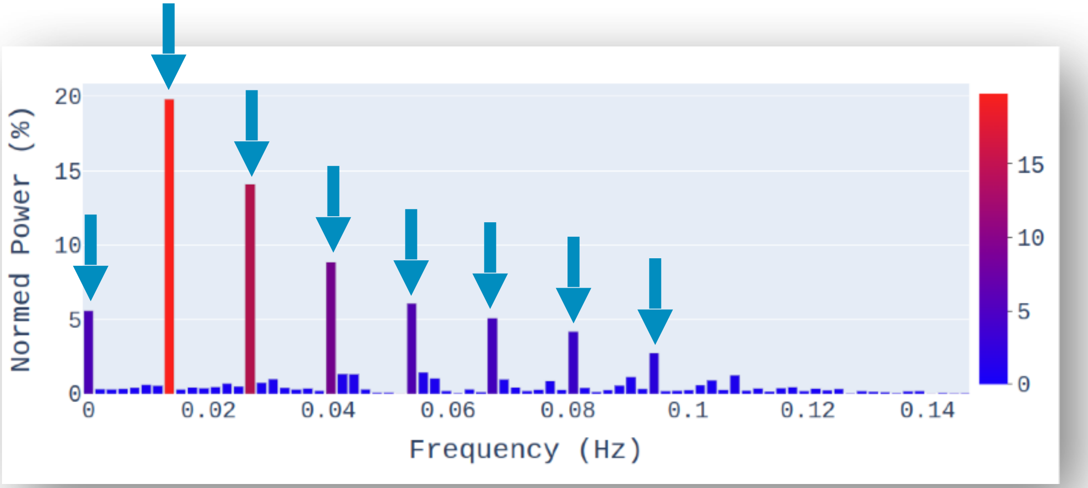
FTIO: Online Demo

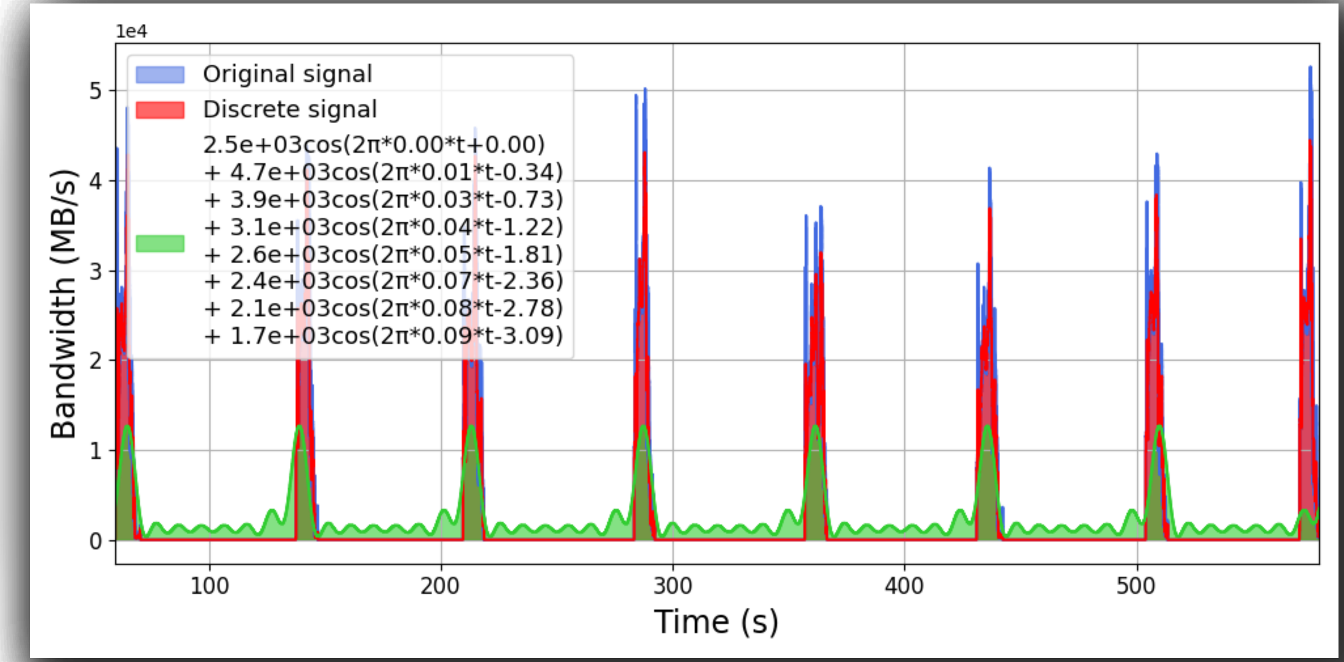
TECHNISCHE UNIVERSITÄT DARMSTADT

Parallel Programming



FTIO and I/O Bursts: IOR with 7680 Ranks

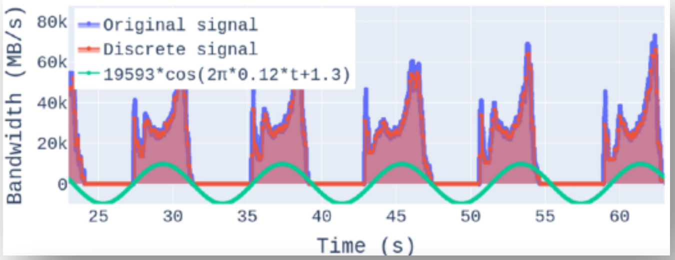




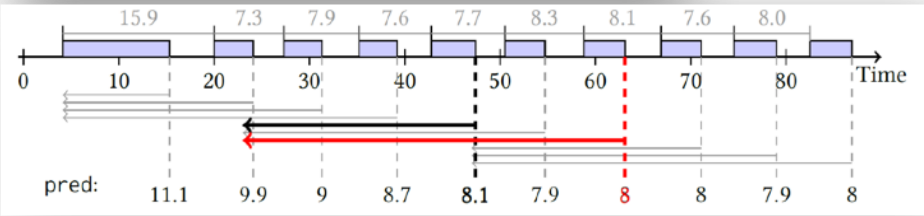
More Examples and Extensions

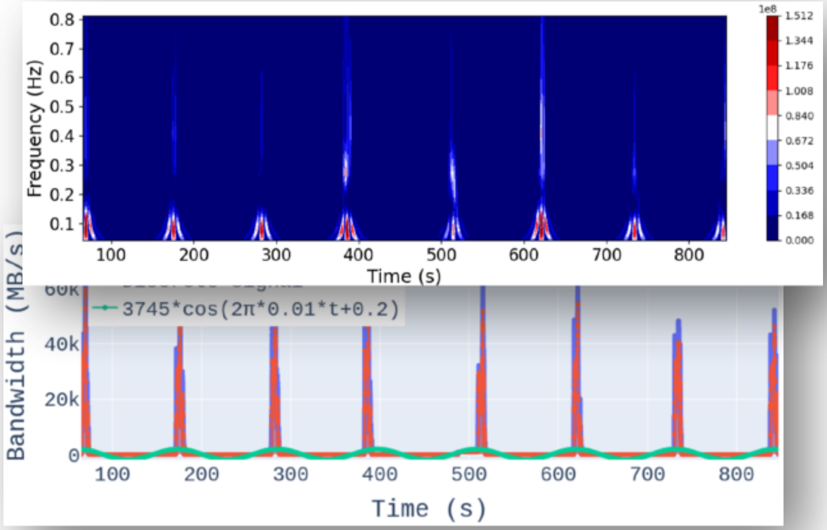
TECHNISCHE UNIVERSITÄT DARMSTADT

Parallel Programming

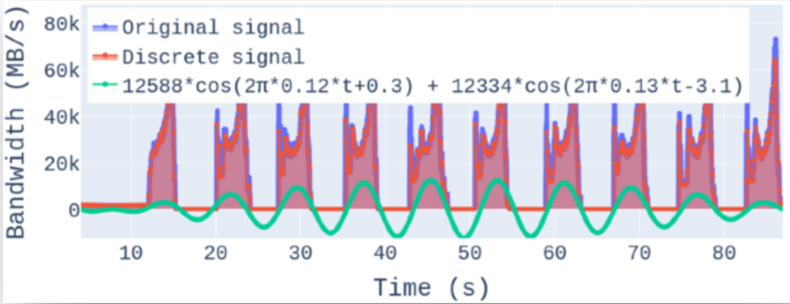


HACC-IO with 3072 ranks: Online Time window adaptation

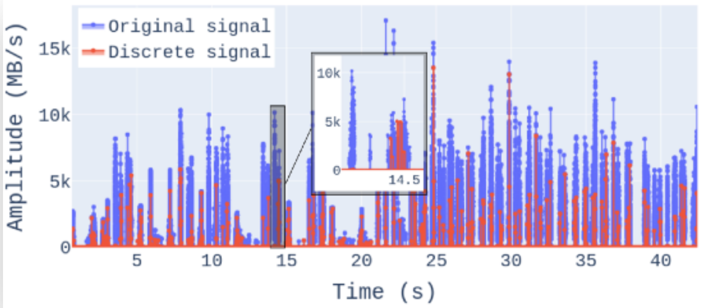




IOR with 9216 ranks: Continuous Wavelet transformation



HACC-IO with 3072 ranks: Merge Frequencies for higher accuracy



Mini-IO with 144 ranks: Non-periodic signal ($f_s = 100$ Hz)

FTIO Meets I/O-Sets

TECHNISCHE UNIVERSITÄT DARMSTADT

Parallel Programming

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

- **FTIO:** Frequency techniques to characterize temporal I/O behavior
  - Finds the frequency (f_d) of I/O phases ($\frac{1}{period}$, or $\frac{1}{w_{iter}}$)
  - A. Tarraf, A. Bandet, F. Boito, G. Pallez, and F. Wolf, IPDPS 2024 (to appear)

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

FTIO Meets I/O-Sets (cont')

TECHNISCHE UNIVERSITÄT DARMSTADT

Parallel Programming

- **FTIO** in prediction mode watches over the trace of each application:
  - **TMIO** monitors an application and continuously appends to a trace
  - **FTIO outputs frequency and confidences** whenever new traces are available
  - A wrapper code, which called FTIO, recovers the output, and calculates the 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**

- Used 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
- BeeGFS client recovers the priority from the file when sending requests

Experimental Methodology

TECHNISCHE UNIVERSITÄT DARMSTADT

Parallel Programming

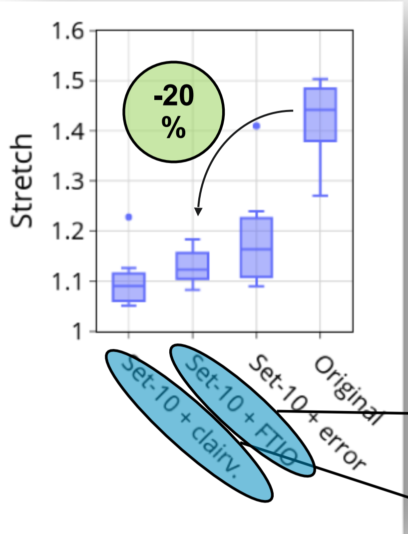
- Applications are generated using the IOR benchmarking:
  - Using fsync option(to have stable performance without caching)
  - Using MPI-IO API with file-per-process write access
  - Modified IOR:
    - To get start and end timestamps of I/O phases (to calculate metrics)
    - Included **TMIO**

- 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)

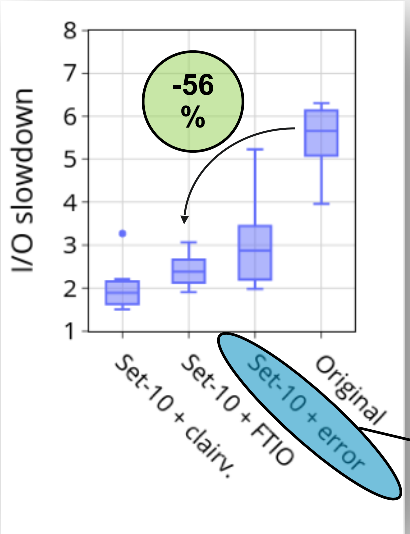
→ Basically, we recreated the experiment from the **IEEE TPDS paper where Set-10 had excellent results** (while adapting to a different platform)

I/O Scheduling Results

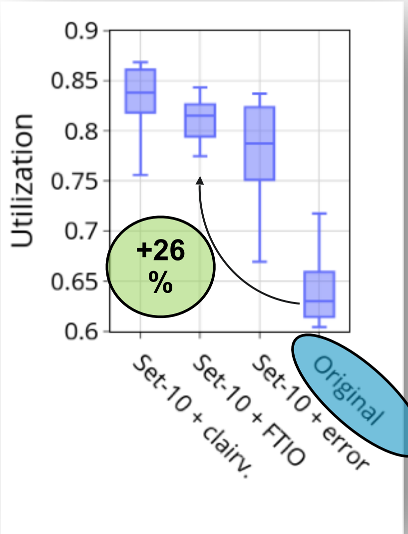
TECHNISCHE UNIVERSITÄT DARMSTADT
Parallel Programming



The lower, the better



The higher, the better



FTIO provides ***actual*** periods to Set-10

Ideal periods are hardcoded

Error (50%) is injected in the result from FTIO

BeeGFS without any modifications

Stretch:
For each application, how much **it was slowed-down by others compared to running in isolation** itself (min=1)

I/O-Slowdown:
For each application, how much **its I/O was slower-down compared to running in isolation** (min=1)

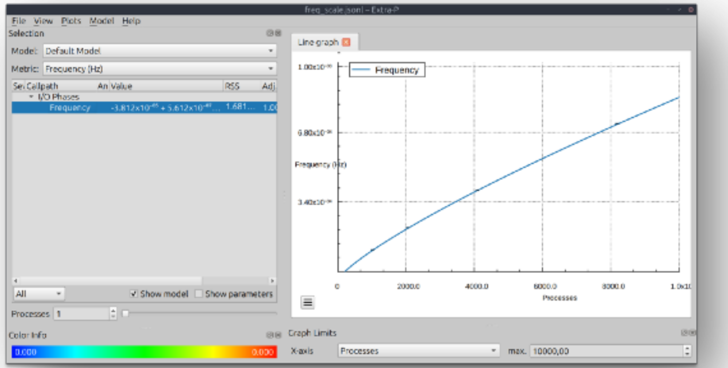
Utilization:
How much of the **system time was spent on compute,** i.e., not doing or waiting for I/O (∈ [0,1])

→FTIO makes Set-10 possible in practice, where the period is not known in advance!

Future Work

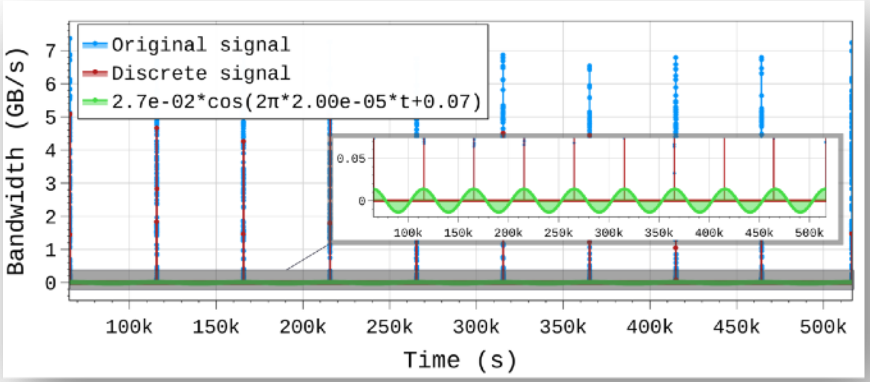
TECHNISCHE UNIVERSITÄT DARMSTADT

Parallel Programming

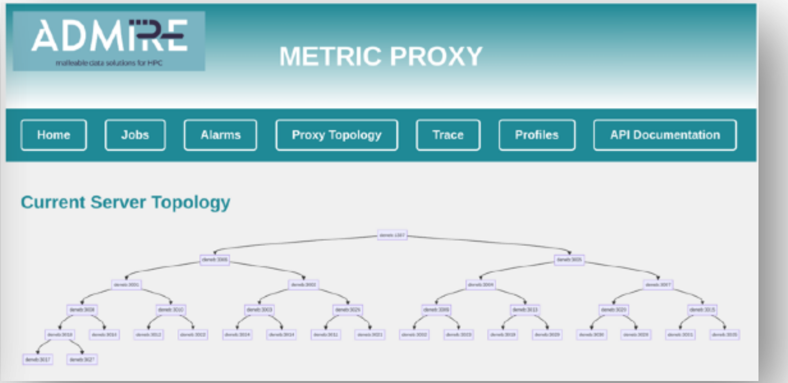


Phase models with Extra-P + FTIO

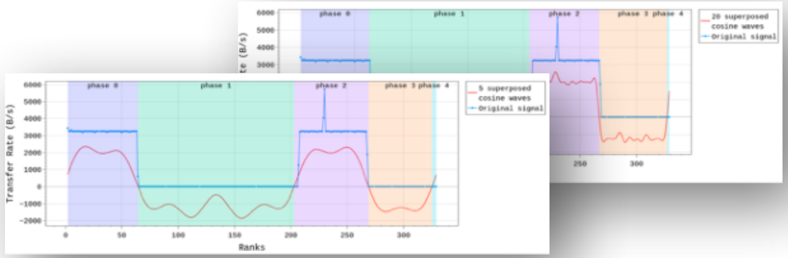
See: https://admire-eurohpc.eu/wp-content/uploads/2024/02/D5_5_admire.pdf



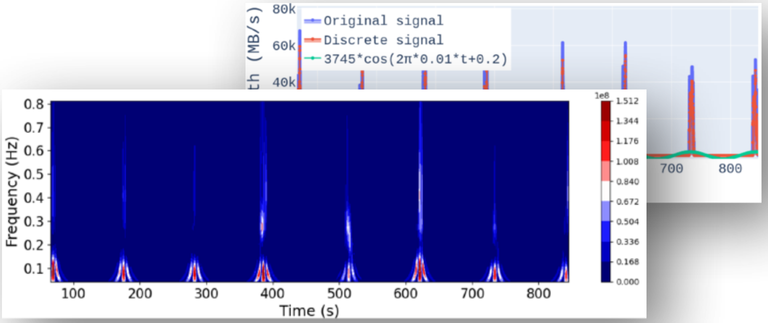
GekkoFS and FTIO



Integration with the monitoring framework Metric Proxy (ISC24)



Classifying Phase with FTIO



Extension: Enhancing predictions

… and many more!

Contact us for collaborations
ahmad.tarraf@tu-Darmstadt.de

Open-Source Tool

TECHNISCHE UNIVERSITÄT DARMSTADT

Parallel Programming

- Available on GitHub: https://github.com/tuda-parallel/FTIO
- Python code, can be easily installed: **pip install ftio-hpc**

Users, contributions, and collaborations are welcomed!

https://github.com/tuda-parallel/FTIO/blob/main/examples/API/test_api.py

```python
import numpy as np
from ftio.cli.ftio_core import core
from ftio.parse.args import parse_args
from ftio.freq._dft import display_prediction
from ftio.freq.freq_plot_core import convert_and_plot
from ftio.parse.bandwidth import overlap

# Set up data
# 1) overlap for rank level metrics
b_rank   = [0.0,0.0,1000.0,1000.0,0.0,0.0,1000.0,1000.0,0.0,0.0,1000.0,1000.0,0.0,0.0]
t_rank_s = [0.5,0.0,10.5,10.0,20.5,20.0,30.5,30.0,40.5,40.0,50.5,50.0,60.5,60]
t_rank_e = [5.0,4.5,15.0,14.5,25.0,24.5,35.0,34.5,45.0,44.5,55.0,54.5,65.0,64.5]
b,t = overlap(b_rank,t_rank_s, t_rank_e)
# or 2) Directly specify the app level metrics
# t = [10.0, 20.1, 30.0, 40.2, 50.3, 60, 70, 80.0,]
# b = [10, 0, 10, 0, 10, 0, 10, 0]

# pass arguments
argv = ["-e", "no"] #no plot
# set up data
data = {
    "time": np.array(t),
        "bandwidth": np.array(b),
        "total_bytes": 0,
        "ranks": 10
        }
#parse args
args = parse_args(argv,"ftio")
# perform prediction
prediction, dfs = core([data], args)
# plot and print info
convert_and_plot(data, dfs, args)
display_prediction("ftio", prediction)
```

Conclusion

TECHNISCHE UNIVERSITÄT DARMSTADT

Parallel Programming

- Presented an approach to characterize and predict the I/O phases 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
- Demonstrated it usefulness for I/O Scheduling
- Several **parameters** can be changed to enhance the results obtained
- ***Is available on GitHub!***

- Contact: ahmad.tarraf@tu-darmstadt.de

References

TECHNISCHE UNIVERSITÄT DARMSTADT

Parallel Programming

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.

Acknowledgment

TECHNISCHE UNIVERSITÄT DARMSTADT

Parallel Programming

- 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

TECHNISCHE UNIVERSITÄT DARMSTADT
Parallel Programming

Reproduce all our experiments from the paper!

https://github.com/tuda-parallel/FTIO/tree/main/artifacts/ipdps24