TYPE Brief Research Report
PUBLISHED 04 February 2026
DOI 10.3389/fhpcp.2025.1638924

OPEN ACCESS

EDITED BY
Geoffrey Fox,
University of Virginia, United States

REVIEWED BY
Norbert Pataki,
Eötvös Loránd University, Hungary
Raul Tapia,
Sevilla University, Spain

*CORRESPONDENCE
Ahmad Tarraf
ahmad.tarraf@tu-darmstadt.de

RECEIVED 31 May 2025
REVISED 26 November 2025
ACCEPTED 18 December 2025
PUBLISHED 04 February 2026

CITATION
Tarraf A and Wolf F (2026) Improving I/O phase predictions in FTIO using hybrid wavelet-Fourier analysis.
Front. High Perform. Comput. 3:1638924.
doi: 10.3389/fhpcp.2025.1638924

Improving I/O phase predictions in FTIO using hybrid wavelet-Fourier analysis

Ahmad Tarraf* and Felix Wolf

Department of Computer Science, Technical University of Darmstadt, Darmstadt, Germany

With the growing complexity of I/O software stacks and the rise of data-intensive workloads, optimizing I/O performance is essential for enhancing overall system performance on HPC clusters. While many sophisticated I/O management approaches exist that try to alleviate I/O contention, they often rely on models that predict the future I/O behavior of applications. Yet, these models are often created from past execution runs and can be error-prone due to I/O variability. In this work, we propose an enhancement to an existing tool that leverages frequency-based techniques to characterize I/O phase. We explore methods to improve prediction accuracy by incorporating multiple frequency components. Furthermore, by coupling the wavelet transformation with the Fourier transformation, we enhance the precision of our predictions while maintaining a compact and efficient behavioral characterization. We demonstrate our approach using a deep learning benchmark executed on a production cluster.

KEYWORDS
behavioral characterization, HPC, I/O, performance modeling, wavelet transform

1 Introduction

Modern HPC applications are increasingly becoming more I/O intensive with the emergence of machine learning applications and data-intensive workflows. In light of the emergence of such workloads (e.g., AI, big data analytics, and complex multi-step workflows) running alongside future exascale applications, workloads are expected to become increasingly data-intensive, leading to even less predictable I/O behavior and access patterns in HPC, as depicted by Neuwirth and Paul (2021). Many applications spend 15%–40% of their execution time performing I/O, with a good probability that this value even further increases in the future (Macedo et al., 2023; Patel et al., 2019, 2020). Depending on the application, access patterns, data volumes, and parallelism of I/O operations can vary significantly (Bez et al., 2024). In contrast, I/O profiling and characterization tools still lack the ability to capture and analyze emerging HPC workloads (Neuwirth and Paul, 2021). Moreover, the landscape of scientific applications is rapidly changing in the era of exascale computing, with trends toward more dynamic systems and applications (Tarraf et al., 2024c) and concepts like modular supercomputing, which utilizes heterogeneous resources to improve power efficiency, system utilization, and cluster scalability (Neuwirth, 2023). Indeed, characterizing the I/O behavior of an application is already challenging due to well-known problems such as I/O variability, I/O bursts, and I/O contention (Wang et al., 2004; Oral et al., 2014; Liu et al., 2016; Yu et al., 2020) and will likely become even more difficult in the future.

Performance degradation and I/O contention are often encountered when different jobs compete for shared resources, such as I/O (Tarraf et al., 2024b; Liu et al., 2021; Patel et al., 2020). HPC applications often exhibit alternating I/O and computational phases. The I/O phases tend to be periodic, with dominating write I/O operations (e.g., checkpointing) occurring in bursts synchronously across several processes (Hu et al., 2016). Jobs that periodically write can account for a quarter to a third of the node-hours on large systems and for over a third of the total amount of written data for all systems examined by Zanon Boito et al. (2025). However, since modern applications can undergo several phases, strictly periodic behavior is often only valid in specific time segments.

Recent studies like Dorier et al. (2014); Jeannot et al. (2021); Boito et al. (2023) revealed that knowledge of periodic I/O phases, even when not *perfectly precise*, leads to good contention avoidance. This motivated us to develop FTIO (Frequency Techniques for I/O) (Tarraf et al., 2024a), which allows predicting periodic I/O phases both online and offline. The tool combines signal processing techniques with outlier detection methods to determine the frequency of the I/O phases, along with metrics that gauge the confidence in the results. This paper presents an approach to improve the characterizations from FTIO. In particular, this paper contributes by:

- Provides a hybrid approach that combines the discrete Wavelet transform with the discrete Fourier transform to improve the temporal characterization of I/O in HPC.
- Explores two signal processing techniques (filtering and Fourier fitting) to further improve characterizations.
- Evaluating our solutions on a productive cluster and demonstrating its applicability.

In what follows, we discuss related work (Section 2) and present FTIO (Section 3). We then describe our new extensions for improved characterization in Sections 4 and 5. Afterwards, we evaluate the approach (Section 6) and provide a conclusion (Section 7).

2 Releated work

Many researchers have focused on characterizing and modeling I/O in HPC including work by Liu et al. (2014); Bez et al. (2019); Tang et al. (2014); McKenna et al. (2016); Wang et al. (2018); Tseng et al. (2019); He et al. (2019); Li et al. (2019); Pavan et al. (2019); Xie et al. (2019); Isakov et al. (2020); Xie et al. (2021); Kim et al. (2023); Dorier et al. (2016). A large group of these relies on past application or system logs like the works of Liu et al. (2014); McKenna et al. (2016); Wang et al. (2018); He et al. (2019); Li et al. (2019); Pavan et al. (2019); Xie et al. (2019); Isakov et al. (2020); Xie et al. (2021); Kim et al. (2023). Yet, such information is often not available. Several popular machine learning approaches have been deployed, including neural networks (Bez et al., 2019; Tseng et al., 2019), LSTMs (Li et al., 2019), decision trees (McKenna et al., 2016), and pattern matching (Tang et al., 2014), among others. While such approaches can provide good predictions, models with high predictive accuracy are often *black boxes* and cannot be interpreted directly to explain I/O performance, as described by Isakov et al. (2020). Moreover, such approaches require learning phases that, aside from being expensive, rely on past information that can be tainted by I/O variability.

Other approaches, like using context-free grammars (Dorier et al., 2016), are particularly useful for cache management and I/O prefetching (Dryden et al., 2021). On a higher level of abstraction, some approaches characterize I/O access patterns using the notion of *I/O phases*. For instance, Pavan et al. (2019) clusters jobs with similar I/O behavior, while Liu et al. (2014) extracts an I/O signature using grid clustering. Moving away from using system- or application-dependent thresholds, FTIO Tarraf et al. (2024a) allows predicting the *period* of the I/O phases at runtime using frequency techniques coupled with outlier detection methods. In previous work (Tarraf et al., 2024a), we demonstrated that, combined with the I/O scheduler Set-10, FTIO enabled a 26% boost in system utilization and a 56% reduction of I/O slowdown.

3 Background

3.1 Acquiring I/O information

FTIO supports a variety of file formats[1] from different tools, include Darshan (Snyder et al., 2016), recorder (Wang et al., 2020), Metric Proxy (Besnard et al., 2024), TMIO (Tarraf et al., 2024b), and others. Essentially, FTIO requires the bandwidth over time (i.e., the signal). Based on these two arrays, FTIO detects the period of the I/O phases either at runtime (online) or after the job's execution (offline). For instance, TMIO, which is a C++ library that is compiled with the application or simply preloaded (using LD_PRELOAD), can provide traces online or offline. Consequently, FTIO can characterize the temporal behavior based on these traces, both online and offline. Note that TMIO also supports sending Msgpack (MessagePack Developers, 2025) messages using ZMQ (ZeroMQ Developers, 2025) over sockets, which avoids creating an intermediate file. FTIO can characterize any metric at any level. Internally, these metrics are overlapped to obtain two arrays that describe the signal values and their corresponding timestamps.

3.2 Extracting the period of the I/O phases

Using DFT, FTIO examines the behavior of a signal in the frequency domain. However, this requires the signal to be sampled at equal steps (i.e., fixed sampling rate T_s). Hence, FTIO samples the signal $x(t)$ with a sampling frequency f_s to obtain $N = \Delta t \cdot f_s$ samples:

$$\{x_n = x(n/f_s) \mid n \in [0, N)\}$$

Afterwards, FTIO applies the DFT on the equally spaced sequence x_n to transform it into a sequence X_k of the same size (i.e.,

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

$k \in [0, N)$) in the frequency domain:

$$X_k = \sum_{n=0}^{N-1} x_n e^{\frac{-2\pi kn}{N} i},$$

for the frequencies $f_k = \frac{k}{N} f_s$. Considering that the time window Δt is usually constant in most cases (i.e., when working offline with an already collected trace), increasing f_s results in more components (i.e., N), but still with the same frequency resolution (distance between two consecutive frequencies in the frequency domain). As x_n consists of purely real values for our purposes (I/O signal), DFT is symmetric and only half the spectrum is needed to reconstruct the original signal with the inverse DFT (IDFT):

$$x_n = \frac{1}{N}\left(X_0 + \sum_{k=1}^{\frac{N}{2}} 2|X_k| \cos\left(\frac{2\pi kn}{N} + arg(X_k)\right)\right) \tag{1}$$

with the amplitude $|X_k|$ and the phase $arg(X_k)$. Hence, reconstructing the signal only requires half of the spectrum (*single-sided spectrum*), such that the amplitude of the cosine waves is multiplied by 2 and shifted upwards by the DC offset X_0. FTIO utilizes the Fast Fourier Transform (FFT) algorithm, which has a complexity of $O(NlogN)$. Furthermore, it uses the *power spectrum* ($p_k = \frac{1}{N} X_k^2$) to suppress noise from small amplitude high frequency components. The spectrum is normalized over the total power of the signal to ease the next step of the analysis. Afterwards, FTIO extracts the dominant frequency using one of the supported outlier detection methods, including Z-score (default), DBSCAN, isolation forest, among others. The Z-score, for instance, reveals how many standard deviations σ a power p_k is from the mean $\bar{p}$ of all powers:

$$z_k = \frac{|p_k| - |\bar{p}|}{\sigma} \tag{2}$$

To identify the dominant frequency f_d, FTIO finds the frequency candidates $\mathcal{D}_f$ that satisfy the following equation:

$$\mathcal{D}_f = \{f_k \mid z_k \geq 3 \text{ and } z_k / z_{\max} \geq 0.8\} \tag{3}$$

A Z-score beyond 3 usually indicates an outlier (Kannan et al., 2015). Since several outliers might exist, we normalize the Z-score to the largest Z-score $z_{\max} = \max_{k \geq 1}(z_k)$ and compare this to an 80% tolerance. Depending on the number of candidates, we distinguish threecases:

- *Single candidate* ($\mathcal{D}_f = \{f_k\}$): The signal is periodic with $f_d = f_k$.
- *Two candidates* ($\mathcal{D}_f = \{f_{k_1}, f_{k_2}\}$): f_d is the one with the highest power contribution. However, we have lower confidence in the results.
- *More than two candidates*: The signal is not periodic, and there is no dominant frequency.

FTIO ignores higher harmonic frequencies when candidates are multiples of two, which indicates periodic I/O bursts. The tool provides a confidence c_k and offers both online and offline characterization with enhancements like time window adaptation, among other options, as described in Tarraf et al. (2024a).

4 Improving the temporal characterization of I/O signals

As described in Section 3, FTIO provides the dominant frequency to characterize the periodic behavior of I/O phases. While this is sufficient in most cases, users might be interested in characterizing the signal more accurately. For that, the tool offers the option to further provide frequencies to enhance characterizations as described Sections 4.1 and 4.2.

4.1 Signal reconstruction with multiple frequencies

FTIO usually provides a single dominant frequency f_d to characterize the period of the I/O phases as described in Section 3. As Equation 1 indicates, the more frequency components are used, the more accurate the description becomes. At the same time, the complexity of the characterization is kept simple, as only cosine waves are added. To demonstrate this, we executed HACC-IO with 384 processes (4 nodes) on the Lichtenberg cluster (see Section 6). We varied the number of extracted frequencies from the analysis up to 10 (using the "-n" flag). As Figure 1 shows, the more frequency components are used, the more accurate the representation becomes.

FTIO prints the sorted cosine waves up to the specified limit according to their amplitude. Furthermore, as shown in Figure 1, it also provides the temporal characterizations as a sum of cosine waves, according to Equation 1 up to the specified number of components. To refine this description further, we extended FTIO to use Fourier fitting as described next.

4.2 Fourier fitting

Fourier fitting approximates a function through a sum of sinusoidal waves. Same as DFT, we can use Equation 1 for this purpose up to a specified number of components. Unlike DFT, Fourier fitting approximates the coefficients (amplitudes, phases, and frequencies) of a signal. Since DFT returns the exact decomposition of a signal as a sum of its harmonics for the fundamental frequency $\frac{f_s}{N}$ (Tarraf et al., 2024a), when only a few selected components are used, the representation is not exact. In this context, frequencies between the discrete values f_k (i.e., for a frequency resolution of $\frac{f_s}{N}$) might give a better fit for the signal. Consequently, Fourier fitting might improve the result.

Fourier fitting in FTIO uses the `curve_fit` function from SciPy to fit the sampled signal x_n using non-linear least square fitting. As initial parameters for the amplitudes, frequencies, and phases, the fitting uses the top cosine waves from DFT with up to the specified number of components. Next, the sampled signal is fitted and the optimized parameters of these components are

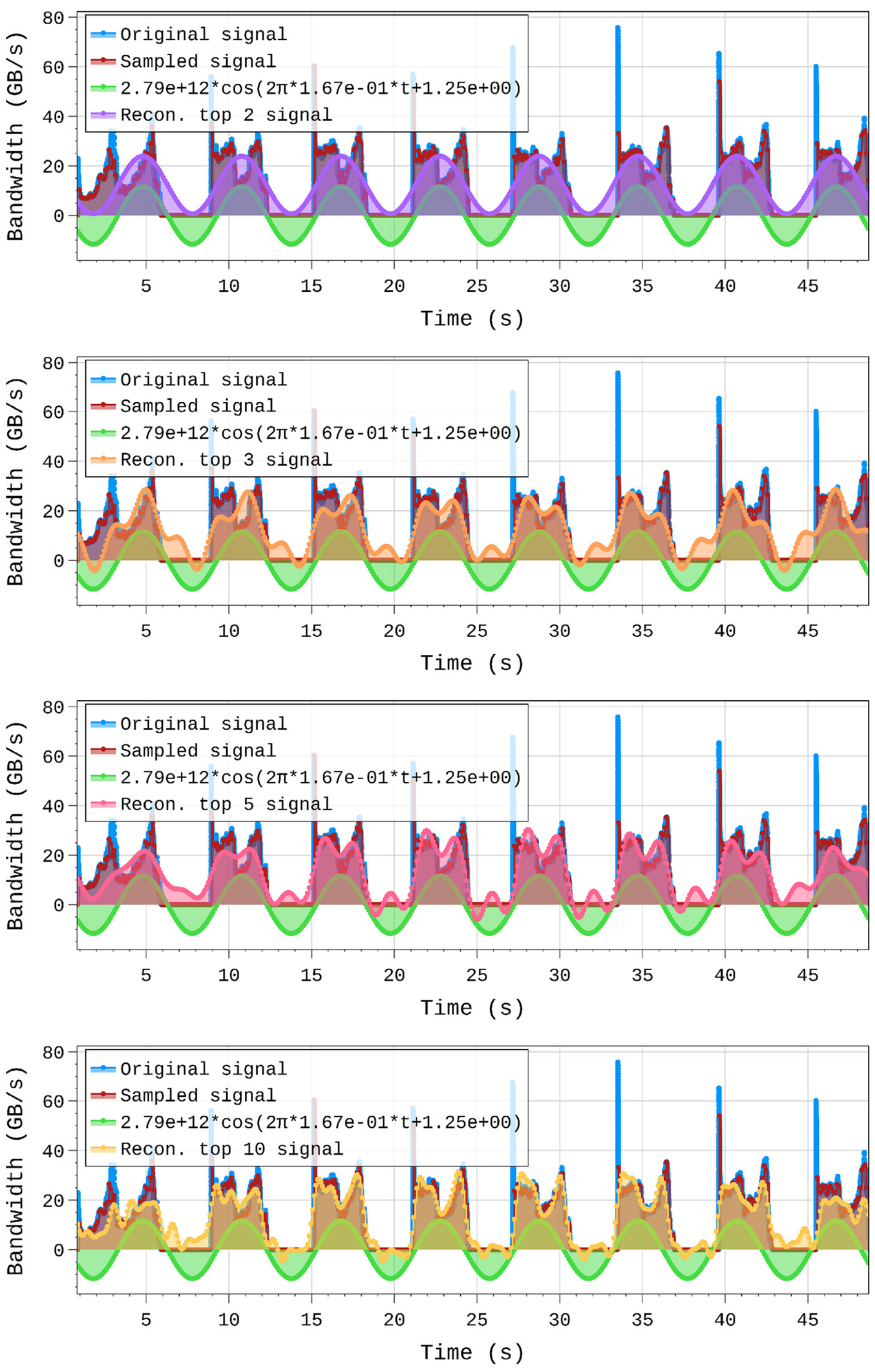

FIGURE 1
Improving signal characterization by specifying the number of frequency components to use. The top part illustrates the default behavior of FTIO. The next three plots show the results obtained with 2, 3, 5, and 10 components, respectively.

returned. We observed that Fourier fitting can improve the results if the signal contains noise (e.g., I/O variation), the signal is not strongly periodic, or the sampling frequency f_s is too low. Note that for the last case, FTIO provides an option to automatically specify the sampling frequency according to the timestamps in the signal. Section 6 provides an example of how Fourier fitting can be used to improve the results.

5 Extracting relevant I/O patterns

As mentioned in Section 1, it is well known that I/O in HPC suffers from performance variability Contrary, not all I/O activities are equally important, and hence some abstraction in the models and characterizations can be tolerated, as works on I/O scheduling showed (Boito et al., 2023). Consequently, as we are often not concerned with all I/O activity, we extended FTIO with another basic building block from the signal processing domain, namely filtering as described in Section 5.1. Furthermore, we incorporated the wavelet analysis in FTIO to extend the tool's characterization capabilities as explained in Section 5.2.

5.1 Low- and high-pass filters

Low- and high-pass filters are key components in signal processing. The filters allow reducing the effect of particular frequencies in the signal. For instance, as its name implies, a low-pass filter allows low-frequency components within a range to pass while suppressing high-frequency components. The cutoff frequency ω_c (in rad/s) is used to indicate which frequencies are filtered. For low frequencies ($\omega \ll \omega_c$), the magnitude response stays close to 1, while for larger frequencies ($\omega \gg \omega_c$), it approaches zero. At the cutoff frequency ω_c, the response is $1/\sqrt{2}$ (–3 dB).

One popular type of low-pass filter is the Butterworth filter (Blackledge, 2006). For simplicity, we skip the explanation. Interested readers can find more details about this in Oppenheim and Schafer (2010). FTIO implements the filters using the `butter` function from `SciPy`. Aside from low and high-pass filters, FTIO can provide band-pass filters. To use these filters, FTIO provides the user with the option of selecting the filter type (low-, high-, or band-pass filter), the cutoff frequencies in Hz (i.e., $w_c = 2\pi f_c$), and the order of the filter. Afterwards, the discrete sequence x_n is filtered, and the approach continues as usual.

Filtering allows FTIO to focus on a particular frequency range. If a higher frequency resolution is desired, f_s can be increased, which, however, results in a greater number of frequency components. If a tool, for instance, can only react after a specific amount of time, the filter can be used to remove the unnecessary components. Yet this option should be used with caution, as I/O variations are artifacts of the signal, and not noise. Consequently, deviation might be presented from the original behavior.

5.2 Discrete wavelet transform

Often, we are interested not only in the period of a signal but also in how long this period is or was valid. To improve FTIO and adapt to such a scenario, a multi-resolution scheme like the wavelet transformation (Graps, 1995) is used. In contrast to DFT, the wavelet transform sacrifices frequency resolution for time resolution. The idea is to convolute an input signal $x(t)$ with a set of small finite waves (i.e., the wavelets), which are scaled (i.e., by the scale parameter $a \in \mathbb{R}^+$) and translated (by the translation parameter $b \in \mathbb{R}$) versions of the mother wavelet:

$$\psi_{a,b}(t) = \frac{1}{\sqrt{|a|}}\,\psi\left(\frac{t-b}{a}\right) \tag{4}$$

Hence, the Continuous Wavelet Transform (CWT) of a signal $x(t)$ is (Mallat, 2008; Chaovalit et al., 2011):

$$W_x(a,b) = \int_{-\infty}^{\infty} x(t)\,\frac{1}{\sqrt{|a|}}\,\psi^*\left(\frac{t-b}{a}\right) dt \tag{5}$$

Such that "*" denotes the complex conjugate of the mother wavelet ψ. However, due to the continuous nature of a and b, the CWT is highly redundant (Akansu et al., 2010) and computationally expensive. By restricting the scales and translations to discrete values (i.e., usually dyadic scales and translations $a = 2^j$ and $b = k2^j$), the DWT can be derived. From the implementation perspective, the DWT is computed by successively passing a signal through high-pass and low-pass filters, producing detail (from high-pass) and approximation (from low-pass) coefficients (Chaovalit et al., 2011). In particular, at each decomposition level j, the signal is convolved with a low and high-pass filter, and the result is decimated (downsampled) by 2. Downsampling is necessary to avoid doubling the data at every level. Furthermore, since half the frequencies have been removed after filtering, half the samples can be discarded, as they do not provide any additional information (Nyquist). At each decomposition level j, the frequency range can be approximated (as filters are not ideal) to $[\frac{f_s}{2^{j+1}}, \frac{f_s}{2^j}]$. For further information, see Mertins (1999); Mallat (2008).

FTIO uses `PyWavelets` to execute DWT. The sampled signal is provided to this function. Consequently, the sampling frequency f_s specifies the highest captured frequency in the signal ($\frac{f_s}{2}$). As a result from DWT, arrays of detail and approximation coefficients are returned. FTIO allows specifying the mother wavelet and uses by default Haar (db1).

5.3 A hybrid approach: combining DWT with DFT

While the results from DWT could be used to extract the period of the I/O phases, the approach is not straightforward. When the coefficients from DWT are upsampled, they essentially show filtered versions of the signal at different frequency ranges. Yet, our approach tries to avoid using thresholds as far as possible. Hence, an approach to tackle the problem is to combine DFT with DWT.

After sampling the signal with f_s, the DWT is applied. Next, we apply DFT on the *upsampled approximation coefficients*. Consequently, DFT is applied to a smoother signal with less variation compared to the existing approach (Section 3). In case a dominant frequency is found, it's used to determine the width of a sliding window w, and ultimately the number of samples m inside it. Next, the Pearson correlation between the dominant cosine wave from DFT (if FTIO found a prediction) and the upsampled approximation coefficients is calculated inside w (i.e., for m samples). We iterate over the length of $n - m$ samples and repeat this calculation, to obtain the sliding window correlation up till $\Delta t - w$ seconds. Note that this calculation shares similarities with the concept of the CWT. Afterwards, we examine how long the sliding window correlation was positive. The start and end times of these segments are compared to the dominant cosine wave: Segments shorter than their own period are discarded, while adjacent segments whose boundaries lie within one period are merged. Finally, FTIO returns an array with start and end ranges when the input signal can be characterized by the dominant frequency f_d.

6 Evaluation

6.1 Experimental setup

All experiments in this paper were performed on the Lichtenberg cluster (Stage 1 partition), which features 643 compute nodes, each with 96 CPU cores and 384 GB main memory. The access mode is user-exclusive, and the shared file system (IBM Spectrum Scale) achieves a peak performance of 150 GB/s.

6.2 Fourier fitting

To demonstrate how Fourier fitting can improve the characterizations of a signal, we executed HACC-IO (LLNL, 2025) with 9,216 processes on the Lichtenberg cluster. HACC-IO mimics one I/O phase of HACC [Hybrid/Hardware Accelerated Cosmology Code (Habib et al., 2012)]. HACC-IO has four steps: compute, write, read, and verify. We added a loop to execute these steps repeatedly. We configured HACC-IO to write to a single shared file using MPI I/O. To capture the I/O trace, TMIO was preloaded (LD_PRELOAD). Afterwards, we used FTIO with a sampling frequency of 1 Hz to characterize the writing phases with up to 10 frequencies and refine the results using Fourier fitting. As Figure 2 shows, while selecting 10 frequency components already result in a good characterization, Fourier fitting can further improve this. In particular, by comparing the sampled signal to the output of DFT and Fourier fitting, we observed that Fourier fitting reduces the mean square error by 31.11%.

6.3 Combining DWT with DFT

To demonstrate the results of combining the DFT with DWT, we again used HACC-IO from Section 6.2. We set the decomposition level to three and executed FTIO on the trace. The results are illustrated in Figure 3. From the DFT, FTIO found the dominant frequency of the I/O phases to be 0.055 Hz (i.e., 18.01 s). As the top of Figure 3 shows, this representation is not valid for the entire time, since the application contains some variations. Using DWT, the relevant frequencies can be inspected by examining the approximation coefficients. As described in Section 5.3, by combining the results from the two transforms and examining the sliding window correlation, FTIO returns that the dominant frequency from DFT is valid inside $[9.88, 72.08]$ and $[106.38, 168.28]$ s. The entire process (computing DWT, applying sliding window correlation, and extracting the results) consumed 4.2 s. Note that the trace is 180.1 s long, but due to the sliding window (18 s), the correlation is calculated only up to 168 s.

Next, we demonstrate our approach using DLIO (Devarajan et al., 2021), an I/O benchmark for Deep Learning. We configured the benchmark to use a modified ResNet50 configuration, which utilizes PyTorch to generate training data, train the model, and perform checkpointing (35.4 GB) every epoch (a total of 10 epochs). Furthermore, DLIO trains on 1,024 files (in the npz format, each containing 100 samples) using 8 readers and computational threads, with a batch size of 400. We executed the benchmark with 96 MPI ranks on 2 nodes. On each node, a metric proxy client was running that forwarded the collected metrics to the root proxy running on the login node. After the benchmarks finished, the trace is examined directly by querying it to FTIO and using the hybrid approach that combines DFT (with f_s=10 Hz) and DWT with a decomposition level of 2. Figure 4 shows the results. In around 4 s, FTIO found that the dominant frequency 0.0361 Hz (i.e., 27.65 s) is valid inside $[84.54, 370.04]$ s. While the results are promising, further evaluation is needed to identify the limitations, optimize the critical parameters (decomposition level), and enhance the sliding window correlation at its boundaries.

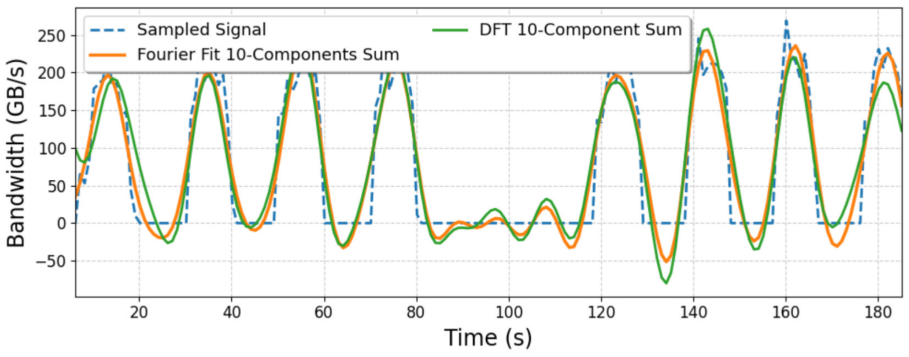

FIGURE 2
Improving the characterizations from FTIO with 10 frequencies and Fourier fitting.

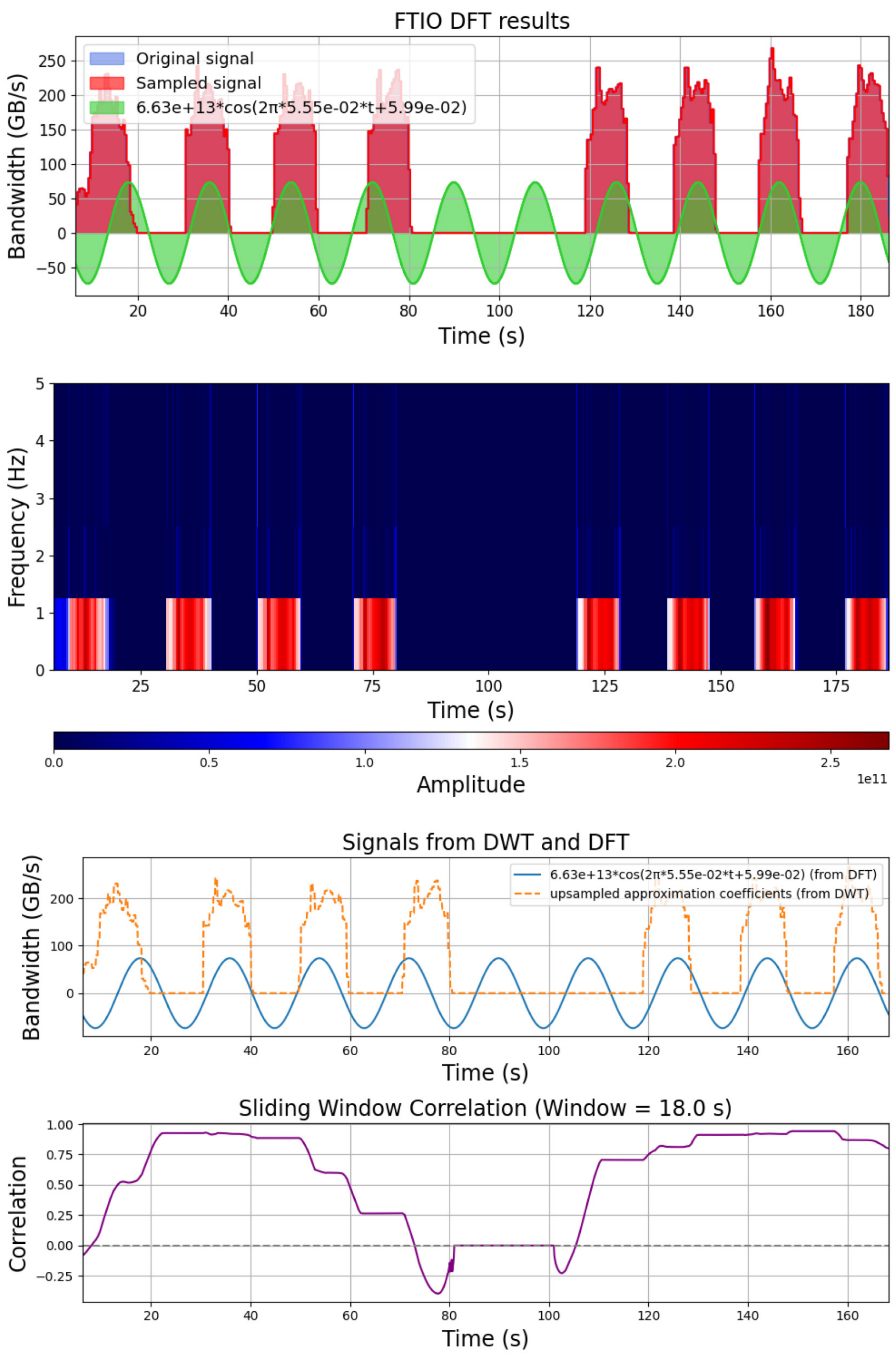

FIGURE 3
Combining the results from DFT with DWT. The **top** part shows the result from DFT on the write behavior of HACC-IO with 9216 ranks. The second figure from the **top** shows the scalogram from the DWT with a decomposition level of 2. The next figure shows the dominant cosine wave from the DFT and the upsampled approximation coefficients from the DWT. The **bottom** part shows the sliding time window correlation of the two signals for a window length of 18 s.

7 Conclusion

This paper presented an extension to FTIO that allows improving charterization using Fourier fitting and extracting valid ranges for the detected dominant frequency. The later aspect was realized by combining DFT and DWT and interpreting their common results. Consequently, FTIO's characterizations are improved in the presence of I/O variations. While the results show that this approach is promising, further evaluation and examination are needed to improve the calculation at the boundaries and parameter selections.

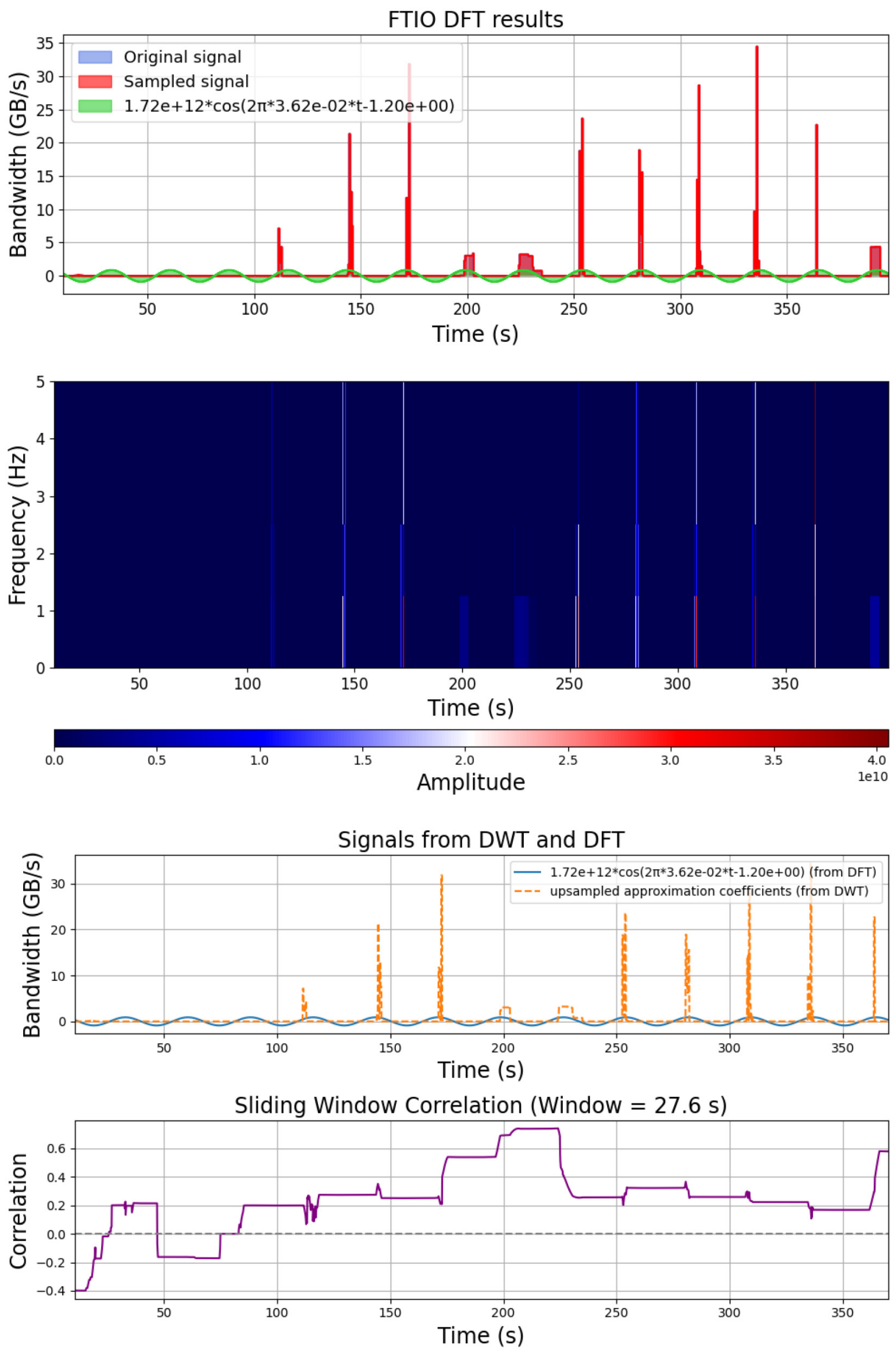

FIGURE 4
Combining the results from DFT with DWT for DLIO. After data generation, DLIO writes 10 checkpoints (35.4 GB) at the end of each epoch. By combining DWT with DFT and examining the sliding window correlation, FTIO yields a range where the dominant frequency is valid.

Data availability statement

The datasets for this study can be found in Tarraf and Wolf (2025). The instructions to reproduce the results are included in the FTIO repository[2].

2 https://github.com/tuda-parallel/FTIO/tree/main/artifacts/frontiers25

Author contributions

AT: Conceptualization, Data curation, Formal analysis, Funding acquisition, Investigation, Methodology, Project administration, Resources, Software, Supervision, Validation, Visualization, Writing – original draft, Writing – review & editing. FW: Funding acquisition, Supervision, Writing – review & editing.

Funding

The author(s) declared that financial support was received for this work and/or its publication. This work was partially funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – Project No. 449683531 (ExtraNoise).

Acknowledgments

The authors gratefully acknowledge the computing time provided to them on the high-performance computer Lichtenberg II at TU Darmstadt, funded by the German Federal Ministry of Research, Technology and Space (BMFTR) and the State of Hesse.

Conflict of interest

The author(s) declared that this work was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Generative AI statement

The author(s) declared that generative AI was used in the creation of this manuscript. Improve the language of the writing and fit the space constraints.

Any alternative text (alt text) provided alongside figures in this article has been generated by Frontiers with the support of artificial intelligence and reasonable efforts have been made to ensure accuracy, including review by the authors wherever possible. If you identify any issues, please contact us.

References

Akansu, A. N., Serdijn, W. A., and Selesnick, I. W. (2010). Emerging applications of wavelets: a review. *Phys. Commun.* 3, 1–18. doi: 10.1016/j.phycom.2009.07.001

Besnard, J.-B., Tarraf, A., Cascajo, A., and Shende, S. (2024). "Introducing the metric proxy for holistic I/O measurements," in *High Performance Computing. ISC High Performance 2024 International Workshops*, eds. M. Weiland, S. Neuwirth, C. Kruse, and T. Weinzierl (Cham: Springer Nature Switzerland), 213–226. doi: 10.1007/978-3-031-73716-9_15

Bez, J., Boito, F., Nou, R., Miranda, A., Cortes, T., and Navaux, P. O. (2019). "Detecting I/O access patterns of HPC workloads at runtime," in *SBAC-PAD 2019 - International Symposium on Computer Architecture and High Performance Computing, Campo Grande, Brazil.* doi: 10.1109/SBAC-PAD.2019.00025

Bez, J. L., Byna, S., and Ibrahim, S. (2024). I/O access patterns in HPC applications: a 360-degree survey. *ACM Comput. Surv.* 56, 1–41. doi: 10.1145/3611007

Blackledge, J. M. (2006). *Digital Signal Processing: Mathematical and Computational Methods, Software Development, and Applications*. Chichester, West Sussex, England: Horwood, 2nd edition. doi: 10.1533/9780857099457

Boito, F., Pallez, G., Teylo, L., and Vidal, N. (2023). IO-SETS: Simple and efficient approaches for I/O bandwidth management. *IEEE Trans. Paral. Distr. Syst.* 34, 2783–2796. doi: 10.1109/TPDS.2023.3305028

Chaovalit, P., Gangopadhyay, A., Karabatis, G., and Chen, Z. (2011). Discrete wavelet transform-based time series analysis and mining. *ACM Comput. Surv.* 43, 1–37. doi: 10.1145/1883612.1883613

Devarajan, H., Zheng, H., Kougkas, A., Sun, X.-H., and Vishwanath, V. (2021). "DLIO: a data-centric benchmark for scientific deep learning applications," in *IEEE/ACM International Symposium in Cluster, Cloud, and Internet Computing (CCGRID'21)*, 81–91. doi: 10.1109/CCGrid51090.2021.00018

Dorier, M., Antoniu, G., Ross, R., Kimpe, D., and Ibrahim, S. (2014). "CALCioM: mitigating I/O interference in HPC systems through cross-application coordination," in *IPDPS'14* (IEEE), 155–164. doi: 10.1109/IPDPS.2014.27

Dorier, M., Ibrahim, S., Antoniu, G., and Ross, R. (2016). Using formal grammars to predict I/O behaviors in HPC: The Omnisc'IO approach. *IEEE Trans. Paral. Distr. Syst.* 27, 2435–2449. doi: 10.1109/TPDS.2015.2485980

Dryden, N., Böhringer, R., Ben-Nun, T., and Hoefler, T. (2021). "Clairvoyant prefetching for distributed machine learning I/O," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC'21* (New York, NY, USA: ACM). doi: 10.1145/3458817.3476181

Graps, A. (1995). An introduction to wavelets. *IEEE Comput. Sci. Eng.* 2, 50–61. doi: 10.1109/99.388960

Habib, S., Morozov, V., Finkel, H., Pope, A., Heitmann, K., Kumaran, K., et al. (2012). "The Universe at extreme scale: Multi-petaflop sky simulation on the BG/Q," in *2012 International Conference for High Performance Computing, Networking, Storage and Analysis* (Salt Lake City, UT: IEEE), 1–11. doi: 10.1109/SC.2012.106

He, Y., Dai, D., and Bao, F. S. (2019). "Modeling HPC storage performance using long short-term memory networks," in *2019 IEEE 21st International Conference on High Performance Computing and Communications; IEEE 17th International Conference on Smart City; IEEE 5th International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*, 1107–1114. doi: 10.1109/HPCC/SmartCity/DSS.2019.00157

Hu, W., Liu, G.-M., Li, Q., Jiang, Y.-H., and Cai, G.-L. (2016). Storage wall for exascale supercomputing. *Front. Inf. Technol. Electr. Eng.* 17, 1154–1175. doi: 10.1631/FITEE.1601336

Isakov, M., del Rosario, E., Madireddy, S., Balaprakash, P., Carns, P., Ross, R. B., et al. (2020). "HPC I/O throughput bottleneck analysis with explainable local models," in *SC'20*, 1–13. doi: 10.1109/SC41405.2020.00037

Jeannot, E., Pallez, G., and Vidal, N. (2021). Scheduling periodic I/O access with BI-colored chains: models and algorithms. *J. Schedul.* 24, 469–481. doi: 10.1007/s10951-021-00685-8

Kannan, K. S., Manoj, K., and Arumugam, S. (2015). Labeling methods for identifying outliers. *Int. J. Stat. Syst.* 10, 231–238.

Kim, S., Sim, A., Wu, K., Byna, S., and Son, Y. (2023). Design and implementation of I/O performance prediction scheme on HPC systems through large-scale log analysis. *J. Big Data* 10:65. doi: 10.1186/s40537-023-00741-4

Li, D., Wang, Y., Xu, B., Li, W., Li, W., Yu, L., et al. (2019). "PIPULS: predicting I/O patterns using LSTM in storage systems," in *2019 International Conference on High Performance Big Data and Intelligent Systems (HPBDIS)*, 14–21. doi: 10.1109/HPBDIS.2019.8735467

Liu, S., Huang, L., Liu, H., Ruhela, A., Trueheart, V., Lindsey, S., et al. (2021). "Practice guideline for heavy I/O workloads with lustre file systems on TACC supercomputers," in *Practice and Experience in Advanced Research Computing, PEARC '21* (New York, NY, USA: ACM). doi: 10.1145/3437359.3465570

Liu, Y., Gunasekaran, R., Ma, X., and Vazhkudai, S. S. (2014). "Automatic identification of application I/O signatures from noisy server-side traces," in *12th USENIX Conference on File and Storage Technologies (FAST 14)* (Santa Clara, CA: USENIX Association), 213–228.

Liu, Y., Gunasekaran, R., Ma, X., and Vazhkudai, S. S. (2016). "Server-side log data analytics for I/O workload characterization and coordination on large shared storage systems," in *SC '16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 819–829. doi: 10.1109/SC.2016.69

LLNL (2025). *CORAL Benchmark Codes - HACC IO*. Available online at: https://asc.llnl.gov/coral-benchmarks#hacc (Accessed January 10, 2025).

Macedo, R., Miranda, M., Tanimura, Y., Haga, J., Ruhela, A., Harrell, S. L., et al. (2023). "Taming metadata-intensive HPC jobs through dynamic, application-agnostic QoS control," in *2023 IEEE/ACM 23rd International Symposium on Cluster, Cloud and Internet Computing (CCGrid)* (Bangalore, India: IEEE), 47–61. doi: 10.1109/CCGrid57682.2023.00015

Mallat, S. (2008). *A Wavelet Tour of Signal Processing, Third Edition: The Sparse Way*. New York, USA: Academic Press, Inc. 3rd edition.

McKenna, R., Herbein, S., Moody, A., Gamblin, T., and Taufer, M. (2016). "Machine learning predictions of runtime and IO traffic on high-end clusters," in *2016 IEEE International Conference on Cluster Computing (CLUSTER)*, 255–258. doi: 10.1109/CLUSTER.2016.58

Mertins, A. (1999). *Signal Analysis: Wavelets, Filter Banks, Time-Frequency Transforms, and Applications*. New York: John Wiley.

MessagePack Developers (2025). *MessagePack: It's like JSON*. But fast and small. Available online at: https://msgpack.org/index.html (Accessed January 10, 2025).

Neuwirth, S. (2023). "Modular supercomputing and its role in Europe's exascale computing strategy," in *Proceedings of The 39th International Symposium on Lattice Field Theory – PoS(LATTICE2022)* (Bonn, Germany: Sissa Medialab), 245. doi: 10.22323/1.430.0245

Neuwirth, S., and Paul, A. K. (2021). "Parallel I/O evaluation techniques and emerging HPC workloads: a perspective," in *2021 IEEE International Conference on Cluster Computing (CLUSTER)* (Portland, OR, USA: IEEE), 671–679. doi: 10.1109/Cluster48925.2021.00100

Oppenheim, A. V., and Schafer, R. W. (2010). *Discrete-Time Signal Processing. Prentice Hall Signal Processing Series*. Upper Saddle River Munich: Pearson 3, edition.

Oral, S., Simmons, J., Hill, J., Leverman, D., Wang, F., Ezell, M., et al. (2014). "Best practices and lessons learned from deploying and operating large-scale data-centric parallel file systems," in *SC'14: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 217–228. doi: 10.1109/SC.2014.23

Patel, T., Byna, S., Lockwood, G. K., and Tiwari, D. (2019). "Revisiting I/O behavior in large-scale storage systems: the expected and the unexpected," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC '19* (New York, NY, USA: ACM). doi: 10.1145/3295500.3356183

Patel, T., Garg, R., and Tiwari, D. (2020). "GIFT: a coupon based throttle-and-reward mechanism for fair and efficient I/O bandwidth management on parallel storage systems," in *Proceedings of the 18th USENIX Conference on File and Storage Technologies, FAST'20* (USA: USENIX Association), 103–120.

Pavan, P. J., Bez, J. L., Serpa, M. S., Boito, F. Z., and Navaux, P. O. A. (2019). "An unsupervised learning approach for I/O behavior characterization," in *2019 31st International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, 33–40. doi: 10.1109/SBAC-PAD.2019.00019

Snyder, S., Carns, P., Harms, K., Ross, R., Lockwood, G. K., and Wright, N. J. (2016). "Modular HPC I/O characterization with darshan," in *2016 5th Workshop on Extreme-Scale Programming Tools (ESPT)*, 9–17. doi: 10.1109/ESPT.2016.006

Tang, H., Zou, X., Jenkins, J., Boyuka, D. A., Ranshous, S., Kimpe, D., et al. (2014). "Improving read performance with online access pattern analysis and prefetching," in *Euro-Par 2014 Parallel Processing*, eds. F. Silva, I. Dutra, and V. Santos Costa (Cham: Springer International Publishing), 246–257. doi: 10.1007/978-3-319-09873-9_21

Tarraf, A., Bandet, A., Boito, F., Pallez, G., and Wolf, F. (2024a). "Capturing periodic I/O using frequency techniques," in *2024 IEEE International Parallel and Distributed Processing Symposium (IPDPS)* (San Francisco, CA, USA), 465–478. doi: 10.1109/IPDPS57955.2024.00048

Tarraf, A., Munoz, J. F., Singh, D. E., Özden, T., Carretero, J., and Wolf, F. (2024b). "I/O behind the scenes: bandwidth requirements of HPC applications with asynchronous I/O," in *2024 IEEE International Conference on Cluster Computing (CLUSTER)* (Kobe, Japan), 426–439. doi: 10.1109/CLUSTER59578.2024.00044

Tarraf, A., Schreiber, M., Cascajo, A., Besnard, J.-B., Vef, M.-A., Huber, D., et al. (2024c). Malleability in modern HPC systems: current experiences, challenges, and future opportunities. *IEEE Trans. Paral. Distr. Syst.* 35, 1551–1564. doi: 10.1109/TPDS.2024.3406764

Tarraf, A., and Wolf, F. (2025). Improving I/O phase predictions in FTIO using hybrid wavelet-Fourier analysis [data set]. *Front. High Perfor. Comput*. Zenodo. doi: 10.5281/zenodo.17713783

Tseng, S.-M., Nicolae, B., Bosilca, G., Jeannot, E., Chandramowlishwaran, A., and Cappello, F. (2019). "Towards portable online prediction of network utilization using MPI-level monitoring," in *Euro-Par 2019: Parallel Processing*, eds. R. Yahyapour (Cham: Springer International Publishing), 47–60. doi: 10.1007/978-3-030-29400-7_4

Wang, C., Sun, J., Snir, M., Mohror, K., and Gonsiorowski, E. (2020). "Recorder 2.0: efficient parallel I/O tracing and analysis," in *2020 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, 1–8. doi: 10.1109/IPDPSW50202.2020.00176

Wang, F., Xin, Q., Hong, B., Brandt, S. A., Miller, E., Long, D., et al. (2004). "File system workload analysis for large scale scientific computing applications," in *21st IEEE/12th NASA Goddard Conference on Mass Storage Systems and Technologies, April 2004* (College Park, MA, USA).

Wang, T., Snyder, S., Lockwood, G., Carns, P., Wright, N., and Byna, S. (2018). "IOMiner: large-scale analytics framework for gaining knowledge from I/O logs," in *2018 IEEE International Conference on Cluster Computing (CLUSTER)*, 466–476. doi: 10.1109/CLUSTER.2018.00062

Xie, B., Tan, Z., Carns, P., Chase, J., Harms, K., Lofstead, J., et al. (2019). "Applying machine learning to understand write performance of large-scale parallel filesystems," in *2019 IEEE/ACM Fourth International Parallel Data Systems Workshop (PDSW)*, 30–39. doi: 10.1109/PDSW49588.2019.00008

Xie, B., Tan, Z., Carns, P., Chase, J., Harms, K., Lofstead, J., et al. (2021). "Interpreting write performance of supercomputer I/O systems with regression models," in *2021 IEEE International Parallel and Distributed Processing Symposium (IPDPS)* (Portland, OR, USA: IEEE), 557–566. doi: 10.1109/IPDPS49936.2021.00064

Yu, J., Yang, W., Wang, F., Dong, D., Feng, J., and Li, Y. (2020). Spatially bursty I/O on supercomputers: causes, impacts and solutions. *IEEE Trans. Paral. Distr. Syst.* 31, 2908–2922. doi: 10.1109/TPDS.2020.3005572

Zanon Boito, F., Teylo, L., Popov, M., Jolivel, T., Tessier, F., Luettgau, J., et al. (2025). *A deep look into the temporal I/O behavior of HPC applications -extended version*. Technical Report RR-9577, Inria Labri, Univ. Bordeaux. doi: 10.1109/IPDPS64566.2025.00072

ZeroMQ Developers (2025). ZeroMQ. Available online at: https://zeromq.org/ (Accessed January 10, 2025).