Automatic Identification of Application I/O Signatures from Noisy Server-Side Traces

Yang Liu[⋆], Raghul Gunasekaran[†], Xiaosong Ma[⋄,∗] and Sudharshan S. Vazhkudai[†]

[⋆]*North Carolina State University, yliu43@ncsu.edu*
[⋄]*Qatar Computing Research Institute, xma@qf.org.qa*
[†]*Oak Ridge National Laboratory, {gunasekaranr, vazhkudaiss}@ornl.gov*

Abstract

Competing workloads on a shared storage system cause I/O resource contention and application performance vagaries. This problem is already evident in today's HPC storage systems and is likely to become acute at exascale. We need more interaction between application I/O requirements and system software tools to help alleviate the I/O bottleneck, moving towards I/O-aware job scheduling. However, this requires rich techniques to capture application I/O characteristics, which remain evasive in production systems.

Traditionally, I/O characteristics have been obtained using client-side tracing tools, with drawbacks such as non-trivial instrumentation/development costs, large trace traffic, and inconsistent adoption. We present a novel approach, I/O Signature Identifier (IOSI), to characterize the I/O behavior of data-intensive applications. IOSI extracts signatures from *noisy, zero-overhead* server-side I/O throughput logs that are already collected on today's supercomputers, without interfering with the compiling/execution of applications. We evaluated IOSI using the Spider storage system at Oak Ridge National Laboratory, the S3D turbulence application (running on 18,000 Titan nodes), and benchmark-based pseudo-applications. Through our experiments we confirmed that IOSI effectively extracts an application's I/O signature despite significant server-side noise. Compared to client-side tracing tools, IOSI is transparent, interface-agnostic, and incurs no overhead. Compared to alternative data alignment techniques (e.g., dynamic time warping), it offers higher signature accuracy and shorter processing time.

1 Introduction

High-performance computing (HPC) systems cater to a diverse mix of scientific applications that run concurrently. While individual compute nodes are usually dedicated to a single parallel job at a time, the interconnection network and the storage subsystem are often shared among jobs. Network topology-aware job placement attempts to allocate larger groups of contiguous compute nodes to each application, in order to provide more stable message-passing performance for inter-process communication. I/O resource contention, however, continues to cause significant performance vagaries in applications [16, 59]. For example, the indispensable task of checkpointing is becoming increasingly cumbersome. The CHIMERA [13] astrophysics application produces 160TB of data per checkpoint, taking around an hour to write [36] on Oak Ridge National Laboratory's Titan [3] (currently the world's No. 2 supercomputer [58]).

This already bottleneck-prone I/O operation is further stymied by resource contention due to concurrent applications, as there is no I/O-aware scheduling or inter-job coordination on supercomputers. As hard disks remain the dominant parallel file system storage media, I/O contention leads to excessive seeks, significantly degrading the overall I/O throughput.

This problem is expected to exacerbate on future extreme-scale machines (hundreds of petaflops). Future systems demand a sophisticated interplay between application requirements and system software tools that is lacking in today's systems. The aforementioned I/O performance variance problem makes an excellent candidate for such synergistic efforts. For example, knowledge of application-specific I/O behavior potentially allows a scheduler to stagger I/O-intensive jobs, improving *both* the stability of individual applications' I/O performance and the overall resource utilization. However, I/O-aware scheduling requires detailed information on application I/O characteristics. In this paper, we explore the techniques needed to capture such information in an automatic and non-intrusive way.

Cross-layer communication regarding I/O characteristics, requirements or system status has remained a challenge. Traditionally, these I/O characteristics have been captured using client-side tracing tools [5, 7], running on the compute nodes. Unfortunately, the information provided by client-side tracing is not enough for inter-job coordination due to the following reasons.

[∗]Part of this work was conducted at North Carolina State University.

First, client-side tracing requires the use of I/O tracing libraries and/or application code instrumentation, often requiring non-trivial development/porting effort. Second, such tracing effort is entirely elective, rendering any job coordination ineffective when only a small portion of jobs perform (and release) I/O characteristics. Third, many users who do enable I/O tracing choose to turn it on for shorter debug runs and off for production runs, due to the considerable performance overhead (typically between 2% and 8% [44]). Fourth, different jobs may use different tracing tools, generating traces with different formats and content, requiring tremendous knowledge and integration. Finally, unique to I/O performance analysis, detailed tracing often generates large trace files themselves, creating additional I/O activities that perturb the file system and distort the original application I/O behavior. Even with reduced compute overhead and minimal information collection, in a system like Titan, collecting traces for individual applications from over 18,000 compute nodes will significantly stress the interconnect and I/O subsystems. These factors limit the usage of client-side tracing tools for development purposes [26, 37], as opposed to routine adoption in production runs or for daily operations.

Similarly, very limited server-side I/O tracing can be performed on large-scale systems, where the bookkeeping overhead may bring even more visible performance degradations. Centers usually deploy only rudimentary monitoring schemes that collect *aggregate* workload information regarding combined I/O traffic from concurrently running applications.

In this paper, we present IOSI (I/O Signature Identifier), a novel approach to characterizing per-application I/O behavior from *noisy, zero-overhead server-side I/O throughput logs*, collected without interfering with the target application's execution. IOSI leverages the existing infrastructure in HPC centers for periodically logging high-level, server-side I/O throughput. E.g., the throughput on the I/O controllers of Titan's Spider file system [48] is recorded once every 2 seconds. Collecting this information has no performance impact on the compute nodes, does not require any user effort, and has minimal overhead on the storage servers. Further, the log collection traffic flows through the storage servers' Ethernet management network, without interfering with the application I/O. Hence, we refer to our log collection as zero-overhead.

Figure 1 shows sample server-side log data from a typical day on Spider. The logs are composite data, reflecting multiple applications' I/O workload. Each instance of an application's execution will be recorded in the server-side I/O throughput log (referred to as a *sample* in the rest of this paper). Often, an I/O-intensive application's samples show certain repeated I/O patterns,

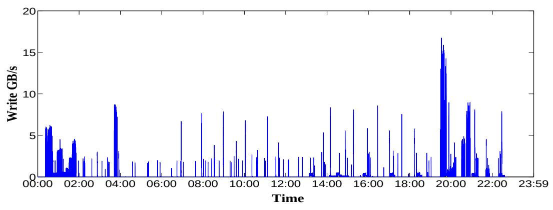


Figure 1: Average server-side, write throughput on Titan's Spider storage (a day in November 2011).

as can be seen from Figure 1. Therefore, the main idea of this work is to *collect and correlate multiple samples, filter out the "background noise", and finally identify the target application's native I/O traffic common across them.* Here, "background noise" refers to the traffic generated by other concurrent applications and system maintenance tasks. Note that IOSI is not intended to record fine-grained, per-application I/O operations. Instead, it derives an estimate of their bandwidth needs along the execution timeline to support future I/O-aware smart decision systems.

Contributions: (1) We propose to extract per-application I/O workload information from existing, zero-overhead, server-side I/O measurements and job scheduling history. Further, we obtain such knowledge of a target application without interfering with its computation/communication, or requiring developers/users' intervention. (2) We have implemented a suite of techniques to identify an application's I/O signature, from noisy server-side throughput measurements. These include i) *data preprocessing*, ii) *per-sample wavelet transform (WT) for isolating I/O bursts*, and iii) *cross-sample I/O burst identification*. (3) We evaluated IOSI with real-world server-side I/O throughput logs from the Spider storage system at the Oak Ridge Leadership Computing Facility (OLCF). Our experiments used several pseudo-applications, constructed with the expressive IOR benchmarking tool [1], and S3D [56], a large-scale turbulent combustion code. Our results show that IOSI effectively extracts an application's I/O signature despite significant server-side noise.

2 Background

We first describe the features of typical I/O-intensive parallel applications and the existing server-side monitoring infrastructure on supercomputers – two enabling trends for IOSI. Next, we define the per-application I/O signature extraction problem.

2.1 I/O Patterns of Parallel Applications

The majority of applications on today's supercomputers are parallel numerical simulations that perform iterative, timestep-based computations. These applications are write-heavy, periodically writing out intermediate results and checkpoints for analysis and resilience, respectively. For instance, applications compute for a fixed number of timesteps and then perform I/O, repeating this sequence multiple times. This process creates regular, predictable I/O patterns, as noted by many existing studies [25, 49, 61]. More specifically, parallel applications' dominant I/O behavior exhibits several distinct features that enable I/O signature extraction:

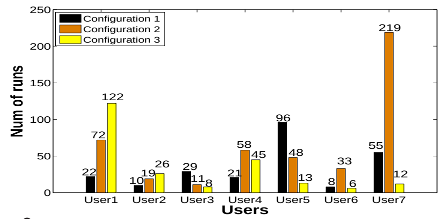


Figure 2: Example of the repeatability of runs on Titan, showing the number of runs using identical job configurations for seven users issuing the largest jobs, between July and September 2013.

Burstiness: Scientific applications have distinct compute and I/O phases. Most applications are designed to perform I/O in short bursts [61], as seen in Figure 1.

Periodicity: Most I/O-intensive applications write data periodically, often in a highly regular manner [25, 49] (both in terms of interval between bursts and the output volume per burst). Such regularity and burstiness suggests the existence of steady, wavelike I/O signatures. Note that although a number of studies have been proposed to optimize the checkpoint interval/volume [19, 20, 39], regular, content-oblivious checkpointing is still the standard practice in large-scale applications [51, 66]. IOSI does not depend on such periodic I/O patterns and handles irregular patterns, as long as the application I/O behavior stays consistent across multiple job runs.

Repeatability: Applications on extreme-scale systems typically run many times. Driven by their science needs, users run the same application with different input data sets and model parameters, which results in repetitive compute and I/O behavior. Therefore, applications tend to have a consistent, identifiable workload signature [16]. To substantiate our claim, we have studied three years worth of Spider server-side I/O throughput logs and Titan job traces for the same time period, and verified that applications have a recurring I/O pattern in terms of frequency and I/O volume. Figure 2 plots statistics of per-user jobs using identical job configurations, which is highly indicative of executions of the same application. We see that certain users, especially those issuing large-scale runs, tend to reuse the same job configuration for many executions.

Overall, the above supercomputing I/O features motivate IOSI to find commonality between multiple noisy server-side log samples. Each sample documents the server-side aggregate I/O traffic during an execution of the same target application, containing different and unknown noise signals. The intuition is that *with a reasonable number of samples, the invariant behavior is likely to belong to the target application.*

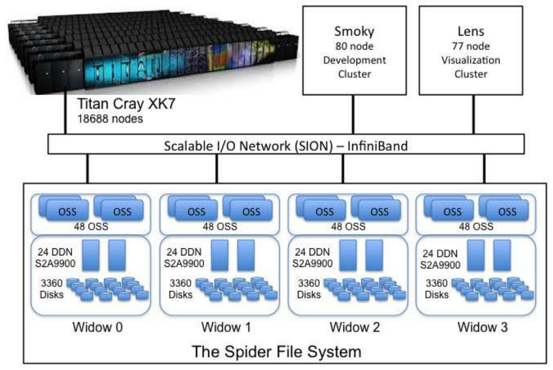


Figure 3: Spider storage system architecture at OLCF.

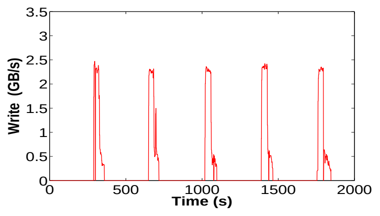
2.2 Titan's Spider Storage Infrastructure

Our prototype development and evaluation use the storage server statistics collected from the Spider center-wide storage system [55] at OLCF, a Lustre-based parallel file system. Spider currently serves the world's No. 2 machine, the 27 petaflop Titan, in addition to other smaller development and visualization clusters. Figure 3 shows the Spider architecture, which comprises of 96 Data Direct Networks (DDN) S2A9900 RAID controllers, with an aggregate bandwidth of 240 GB/s and over 10 PBs of storage from 13,440 1-TB SATA drives. Access is through the object storage servers (OSSs), connected to the RAID controllers in a fail-over configuration. The compute platforms connect to the storage infrastructure over a multistage InfiniBand network, SION (Scalable I/O Network). Spider has four partitions, widow$[0-3]$, with identical setup and capacity. Users can choose any partition(s) for their jobs.

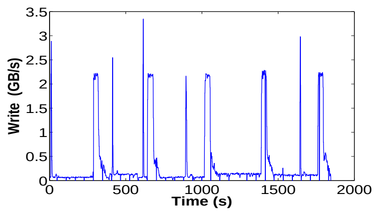
Spider has been collecting server-side I/O statistics from the DDN RAID controllers since 2009. These controllers provide a custom API for querying performance and status information over the management Ethernet network. A custom daemon utility [43] polls the controllers for bandwidth and IOPS at 2-second intervals and stores the results in a MySQL database. Bandwidth data are automatically reported from individual DDN RAID controllers and aggregated across all widow partitions to obtain the overall file system bandwidth usage.

2.3 Problem Definition: Parallel Application I/O Signature Identification

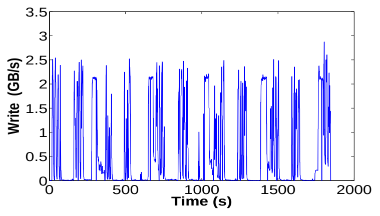
As mentioned earlier, IOSI aims to identify the I/O signature of a parallel application, from zero-overhead, aggregate, server-side I/O throughput logs that are *al-*



(a) IOR_A target signature



(b) Sample IOR_AS1



(c) Sample IOR_AS6

Figure 4: I/O signature of IOR_A and two samples

ready being collected. IOSI's input includes (1) the start and end times of the target application's multiple executions in the past, and (2) server-side logs that contain the I/O throughput generated by those runs (as well as unknown I/O loads from concurrent activities). The output is the extracted I/O signature of the target application.

We define an application's *I/O signature* as the I/O throughput it generates at the server-side storage of a given parallel platform, for the duration of its execution. In other words, if this application runs alone on the target platform without any noise from other concurrent jobs or interactive/maintenance workloads, the server-side throughput log during its execution will be its signature. It is virtually impossible to find such "quiet time" once a supercomputer enters the production phase. Therefore, IOSI needs to "mine" the true signature of the application from server-side throughput logs, collected from its multiple executions. Each execution instance, however, will likely contain different noise signals. We refer to each segment of such a noisy server-side throughput log, punctuated by the start and end times of the execution instance, a *"sample"*. Based on our experience, generally 5 to 10 samples are required for getting the expected results. Note that there are long-running applications (potentially several days for each execution). It is possible for IOSI to extract a signature from even partial samples (e.g., from one tenth of an execution time period), considering the self-repetitive I/O behavior of large-scale simulations.

Figure 4 illustrates the signature extraction problem using a pseudo-application, IOR_A, generated by IOR [1], a widely used benchmark for parallel I/O performance evaluation. IOR supports most major HPC I/O interfaces (e.g., POSIX, MPIIO, HDF5), provides a rich set of user-specified parameters for I/O operations (e.g., file size, file sharing setting, I/O request size), and allows users to configure iterative I/O cycles. IOR_A exhibits a periodic I/O pattern typical in scientific applications, with 5 distinct *I/O bursts*. Figure 4(a) shows its I/O signature, obtained from a quiet Spider storage system partition during Titan's maintenance window. Figures 4(b) and 4(c) show its two server-side I/O log samples when executed alongside other real applications and interactive I/O activities. These samples clearly demonstrate the existence of varying levels of noise. Thus, IOSI's purpose is to find the common features from multiple samples (e.g., Figures 4(b) and 4(c)), to obtain an I/O signature that approximates the original (Figure 4(a)).

3 Related Work

I/O Access Patterns and I/O Signatures: Miller and Katz observed that scientific I/O has highly sequential and regular accesses, with a period of CPU processing followed by an intense, bursty I/O phase [25]. Carns et al. noted that HPC I/O patterns tend to be repetitive across different runs, suggesting that I/O logs from prior runs can be a useful resource for predicting future I/O behavior [16]. Similar claims have been made by other studies on the I/O access patterns of scientific applications [28, 47, 53]. Such studies strongly motivate IOSI's attempt to identify common and distinct I/O bursts of an application from multiple noisy, server-side logs.

Prior work has also examined the identification and use of I/O signatures. For example, the aforementioned work by Carns et al. proposed a methodology for continuous and scalable characterization of I/O activities [16]. Byna and Chen also proposed an I/O prefetching method with runtime and post-run analysis of applications' I/O signatures [15]. A significant difference is that IOSI is designed to automatically extract I/O signatures from existing coarse-grained server-side logs, while prior approaches for HPC rely on client-side tracing (such as MPI-IO instrumentation). For more generic application workload characterization, a few studies [52, 57, 64] have successfully extracted signatures from various server-side logs.

Client-side I/O Tracing Tools: A number of tools have been developed for general-purpose client-side instrumentation, profiling, and tracing of generic MPI and CPU activity, such as mpiP [60], LANL-Trace [2], HPCT-IO [54], and TRACE [42]. The most closely related to IOSI is probably Darshan [17]. It performs low-overhead, detailed I/O tracing and provides powerful post-processing of log files. It outputs a large collection of aggregate I/O characteristics such as operation counts and request size histograms. However, existing client-side tracing approaches suffer from the limitations mentioned in Section 1, such as installation/linking require-

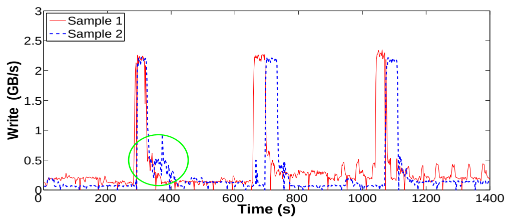


Figure 5: Drift and scaling of I/O bursts across samples

ments, voluntary participation, and producing additional client I/O traffic. IOSI's server-side approach allows it to handle applications using any I/O interface.

Time-series Data Alignment There have been many studies in this area [6, 9, 10, 27, 38, 46]. Among them, dynamic time warping (DTW) [10, 46] is a well-known approach for comparing and averaging a set of sequences. Originally, this technique was widely used in the speech recognition community for automatic speech pattern matching [23]. Recently, it has been successfully adopted in other areas, such as data mining and information retrieval, for automatically addressing time deformations and aligning time-series data [18, 30, 33, 67]. Due to its maturity and existing adoption, we choose DTW for comparison against the IOSI algorithms.

4 Approach Overview

Thematic to IOSI is the realization that the noisy, server-side samples contain common, periodic I/O bursts of the target application. It exploits this fact to extract the I/O signature, using a rich set of statistical techniques. Simply correlating the samples is not effective in extracting per-application I/O signatures, due to a set of challenges detailed below.

First, the server-side logs do not distinguish between different workloads. They contain I/O traffic generated by many parallel jobs that run concurrently, as well as interactive I/O activities (e.g., migrating data to and from remote sites using tools like FTP). Second, I/O contention not only generates "noise" that is superimposed on the true I/O throughput generated by the target application, but also *distorts* it by slowing down its I/O operations. In particular, I/O contention produces *drift* and *scaling* effects on the target application's I/O bursts. The degree of drift and scaling varies from one sample to another. Figure 5 illustrates this effect by showing two samples (solid and dashed) of a target application performing periodic writes. It shows that I/O contention can cause shifts in I/O burst timing (particularly with the last two bursts in this case), as well as changes in burst duration (first burst, marked with oval). Finally, the noise level and the runtime variance caused by background I/O further create the following dilemma in processing the I/O signals: IOSI has to rely on the application's I/O bursts to properly *align* the noisy samples as they are

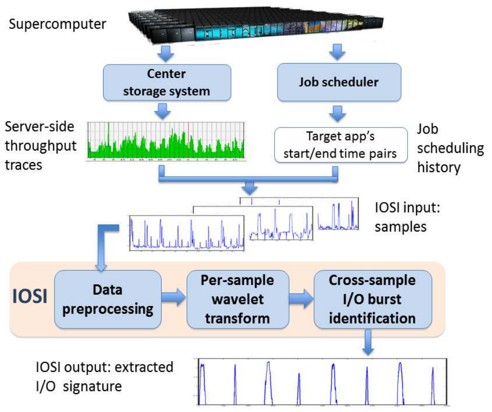

Supercomputer
Center storage system
Job scheduler
Server-side throughput traces
Target app's start/end time pairs
Job scheduling history
IOSI input: samples
IOSI
Data preprocessing
Per-sample wavelet transform
Cross-sample I/O burst identification
IOSI output: extracted I/O signature

Figure 6: IOSI overview

the only common features; at the same time, it needs the samples to be reasonably aligned to *identify* the common I/O bursts as belonging to the target application.

Recognizing these challenges, IOSI leverages an array of signal processing and data mining tools to discover the target application's I/O signature using a black-box approach, unlike prior work based on white-box models [17, 59]. Recall that IOSI's purpose is to render a *reliable estimate of user-applications' bandwidth needs*, instead of to optimize individual applications' I/O operations. Black-box analysis is better suited here for generic and non-intrusive pattern collection.

The overall context and architecture of IOSI are illustrated in Figure 6. Given a target application, multiple samples from prior runs are collected from the server-side logs. Using such a sample set as input, IOSI outputs the extracted I/O signature by *mining* the common characteristics hidden in the sample set. Our design comprises of three phases:

1. **Data preprocessing**: This phase consists of four key steps: outlier elimination, sample granularity refinement, runtime correction, and noise reduction. The purpose is to prepare the samples for alignment and I/O burst identification.
2. **Per-sample wavelet transform**: To utilize "I/O bursts" as common features, we employ wavelet transform to distinguish and isolate individual bursts from the noisy background.
3. **Cross-sample I/O burst identification**: This phase identifies the common bursts from multiple samples, using a grid-based clustering algorithm.

5 IOSI Design and Algorithms

In this section, we describe IOSI's workflow, step by step, using the aforementioned IOR_A pseudo-application (Figure 4) as a running example.

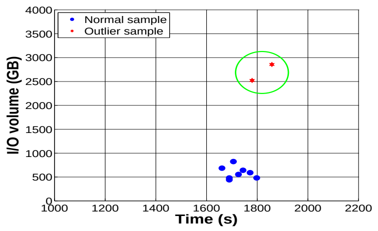


Figure 7: Example of outlier elimination

5.1 Data Preprocessing

Given a target application, we first compare the job log with the I/O throughput log, to obtain I/O samples from the application's multiple executions, particularly *by the same user* and with *the same job size* (in term of node counts). As described in Section 2, HPC users tend to run their applications repeatedly.

From this set, we then eliminate outliers – samples with significantly heavier noise signals or longer/shorter execution time.[1] Our observation from Spider is that despite unpredictable noise, the majority of the samples (from the same application) bear considerable similarity. Intuitively, including the samples that are apparently significantly skewed by heavy noise is counter-productive. We perform *outlier elimination* by examining (1) the application execution time and (2) the volume of data written within the sample (the "area" under the server-side throughput curve). Within this 2-D space, we apply the *Local Outlier Factor (LOF)* algorithm [12], which identifies observations beyond certain threshold as outliers. Here we set the threshold μ as the mean of the sample set. Figure 7 illustrates the distribution of execution times and I/O volumes among 10 IOR_A samples collected on Spider, where two of the samples (dots within the circle) are identified by LOF as outliers.

Next, we perform sample granularity refinement, by decreasing the data point interval from 2 seconds to 1 using simple linear interpolation [22]. Thus, we insert an extra data point between two adjacent ones, which turns out to be quite helpful in identifying short bursts that last for only a few seconds. The value of each extra data point is the average value of its adjacent data points. It is particularly effective in retaining the amplitude of narrow bursts during the subsequent WT stage.

In the third step, we perform *duration correction* on the remaining sample data set. This is based on the observation that noise can only prolong application execution, hence the sample with the *shortest* duration received the *least* interference, and is consequently closest in duration to the target signature. We apply a simple trimming process to correct the drift effect mentioned in Section 4, preparing the samples for subsequent correlation and alignment. This procedure discards data points

[1]Note that shorter execution time can happen with restart runs resuming from a prior checkpoint.

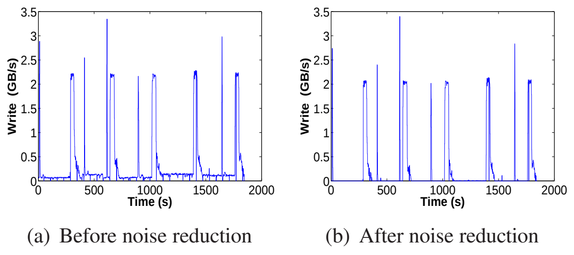


Figure 8: IOR_A samples after noise reduction

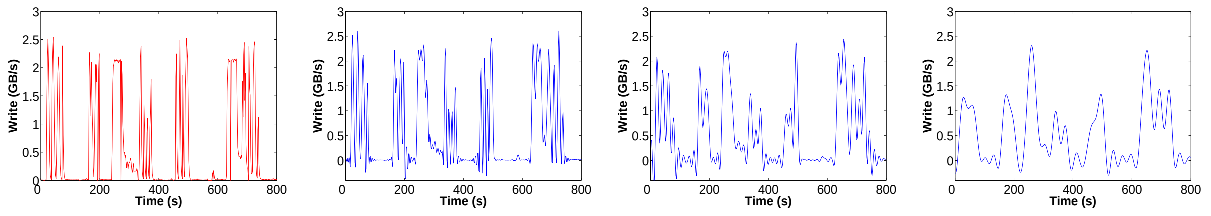
at regular intervals to shrink each longer sample to match the shortest one. For example, if a sample is 4% longer than the shortest one, then we remove from it the 1st, 26th, 51st, ..., data points. We found that after outlier elimination, the deviation in sample duration is typically less than 10%. Therefore such trimming is not expected to significantly affect the sample data quality.

Finally, we perform preliminary *noise reduction* to remove *background noise*. While I/O-intensive applications produce heavy I/O bursts, the server-side log also reports I/O traffic from interactive user activities and maintenance tasks (such as disk rebuilds or data scrubbing by the RAID controllers). Removing this type of persistent background noise significantly helps signature extraction. In addition, although such noise does not significantly distort the shape of application I/O bursts, having it embedded (and duplicated) in multiple application's I/O signatures will cause inaccuracies in I/O-aware job scheduling. To remove background noise, IOSI (1) aggregates data points from all samples, (2) collects those with a value lower than the overall average throughput, (3) calculates the *average background noise level* as the mean throughput from these selected data points, and (4) lowers each sample data point by this average background noise level, producing zero if the result is negative. Figure 8(b) shows the result of such preprocessing, and compared to the original sample in Figure 8(a), the I/O bursts are more pronounced. The I/O volume of IOR_AS1 was trimmed by 26%, while the background noise level was measured at 0.11 GB/s.

5.2 Per-Sample Wavelet Transform

As stated earlier, scientific applications tend to have a bursty I/O behavior, justifying the use of *I/O burst* as the basic unit of signature identification. An I/O burst indicates a phase of high I/O activity, distinguishable from the background noise over a certain duration.

With less noisy samples, the burst boundaries can be easily found using simple methods such as *first difference* [50] or *moving average* [62]. However, with noisy samples identifying such bursts becomes challenging, as there are too many ups and downs close to each other. In particular, it is difficult to do so without knowing the cutoff threshold for a "bump" to be considered a candidate I/O burst. Having too many or too few candidates



(a) Preprocessed IOR_AS6 segment (b) After WT (Decomposition level 1) (c) After WT (Decomposition level 2) (d) After WT (Decomposition level 3)

Figure 9: $dmey$ WT results on a segment of IOR_AS6

can severely hurt our sample alignment in the next step.

To this end, we use a WT [21, 41, 63] to smooth samples. WT has been widely applied to problems such as filter design [14], noise reduction [35], and pattern recognition [24]. With WT, a time-domain signal can be decomposed into low-frequency and high-frequency components. The approximation information remains in the low-frequency component, while the detail information remains in the high-frequency one. By carefully selecting the *wavelet function* and *decomposition level* we can observe the major bursts from the low-frequency component. They contain the most energy of the signal and are isolated from the background noise.

By retaining the temporal characteristics of the time-series data, WT brings an important feature not offered by widely-used alternative techniques such as Fourier transform [11]. We use WT to clarify individual bursts from their surrounding data, without losing the temporal characteristics of the time-series sample.

WT can use quite a few wavelet families [4, 45], such as *Haar*, *Daubechies*, and *Coiflets*. Each provides a transform highlighting different frequency and temporal characteristics. For IOSI, we choose *discrete Meyer (dmey)* [40] as the mother wavelet. Due to its smooth profile, the approximation part of the resulting signal consists of a series of smooth waves. Its output consists of a series of waves where the center of the wave troughs can be easily identified as wave boundaries.

Figures 9(a) and Figures 9(b), 9(c), 9(d) illustrate a segment of IOR_AS6 and its dmey WT results, respectively. With a WT, the high-frequency signals in the input sample are smoothed, producing low-frequency components that correlate better with the target application's periodic I/O. However, here the waves cannot be directly identified as I/O bursts, as a single I/O burst from the application's point of view may appear to have many "sub-crests", separated by minor troughs. This is due to throughput variance caused by either application behavior (such as adjacent I/O calls separated by short computation/communication) or noise, or both. To prevent creating many such "sub-bursts", we use the mean height of all wave crests for filtering – only the troughs lower than this threshold are used for separating bursts.

Another WT parameter to consider is the *decomposition level*, which determines the level of detailed information in the results. The higher the decomposition level, the fewer details are shown in the low-frequency component, as can be seen from Figures 9(b), 9(c) and 9(d). With a decomposition level of 1 (e.g. Figures 9(b)), the wavelet smoothing is not sufficient for isolating burst boundaries. With a higher decomposition level of 3 the narrow bursts fade out rapidly, potentially missing target bursts. IOSI uses a decomposition level of 2 to better retain the bursty nature of the I/O signature.

5.3 Cross-Sample I/O Burst Identification

Next, IOSI correlates all the pre-processed, and wavelet transformed samples to identify common I/O bursts. To address the circular dependency challenge mentioned earlier between alignment and common feature identification, we adapt a grid-based clustering approach called CLIQUE [8]. It performs multi-dimensional data clustering by identifying grids (called *units*) where there is higher *density* (number of data points within the unit). CLIQUE treats each such *dense unit* as a "seed cluster" and grows it by including neighboring dense units.

CLIQUE brings several advantages to IOSI. First, its model fits well with our context: an I/O burst from a given sample is mapped to a 2-D data point, based on its *time* and *shape* attributes. Therefore, data points from different samples close to each other in the 2-D space naturally indicate common I/O bursts. Second, with controllable grid width and height, IOSI can better handle burst drifts (more details given below). Third, CLIQUE performs well for scenarios with far-apart clusters, where inter-cluster distances significantly exceed those between points within a cluster. As parallel applications typically limit their "I/O budget" (fraction of runtime allowed for periodic I/O) to 5%-10%, individual I/O bursts normally last seconds to minutes, with dozens of minutes between adjacent bursts. Therefore, CLIQUE is not only effective for IOSI, but also efficient, as we do not need to examine too far around the burst-intensive areas. Our results (Section 6) show that it outperforms the widely used DTW time-series alignment algorithm [10], while incurring significantly lower overhead.

We make two adjustments to the original CLIQUE

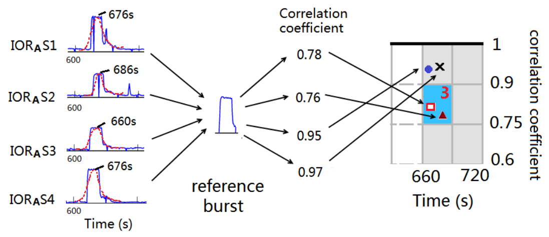


Figure 10: Mapping IOR_A I/O bursts to 2-D points

algorithm. Considering the I/O bursts are sufficiently spaced from each other in a target application's execution, we limit the growth of the cluster to the immediate *neighborhood* of a dense unit: the units that are adjacent to it. Also, we have modified the density calculation to focus not on the sheer number of data points in a unit, but on the number of *different samples* with bursts there. The intuition is that a common burst from the target application should have most (if not all) samples agree on its existence. Below, we illustrate with IOR_A the process of IOSI's common burst identification.

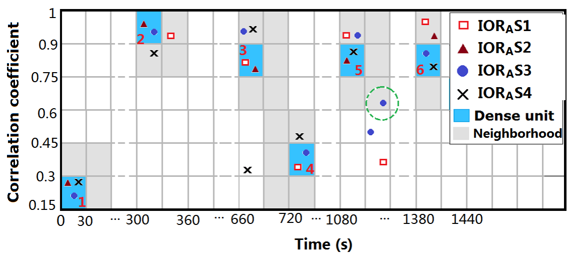


Figure 11: CLIQUE 2-D grid containing IOR_A bursts

To use our adapted CLIQUE, we need to first discretize every sample s_i into a group of 2-D points, each representing one I/O burst identified after a WT. Given its jth I/O burst $b_{i,j}$, we map it to point $\langle t_{i,j}, c_{i,j} \rangle$. Here $t_{i,j}$ is the time of the wave crest of $b_{i,j}$, obtained after a WT, while $c_{i,j}$ is the correlation coefficient between $b_{i,j}$ and a *reference burst*. To retain the details describing the shape of the I/O burst, we choose to use the pre-WT burst in calculating $c_{i,j}$, though the burst itself was identified using a WT. Note that we rely on the transitive nature of correlation ("bursts with similar correlation coefficient to the common reference burst tend to be similar to each other"), so the reference burst selection does not have a significant impact on common burst identification. In our implementation, we use the "average burst" identified by WT across all samples.

Figure 10 shows how we mapped 4 I/O bursts, each from a different IOR_A sample. Recall that WT identifies each burst's start, peak, and end points. The x coordinate for each burst shows "when it peaked," derived using the post-WT wave (dotted line). The y coordinate shows "how similar it is to a reference burst shape," calculated using the pre-WT sample (solid lines).

Therefore, our CLIQUE 2-D data space has an x range of $[0, t]$ (where t is the adjusted sample duration after preprocessing) and a y range of $[0, 1]$. It is partitioned into uniform 2-D grids (units). Defining the unit width and height is critical for CLIQUE, as overly small or large grids will obviously render the density index less useful. Moreover, even with carefully selected width and height values, there is still a chance that a cluster of nodes are separated into different grids, causing CLIQUE to miss a dense unit.

To this end, instead of using only one pair of width-height values, IOSI tries out multiple grid size configurations, each producing an extracted signature. For width, it sets the lower bound as the average I/O burst duration across all samples and upper bound as the average time distance between bursts. For a unit height, it empirically adopts the range between 0.05 and 0.25. Considering the low cost of CLIQUE processing with our sample sets, IOSI uniformly samples this 2-D parameter space (e.g., with 3-5 settings per dimension), and takes the result that identified the most data points as belonging to common I/O bursts. Due to the strict requirement of identifying common bursts, we have found in our experiments that missing target bursts is much more likely to happen than including fake bursts in the output signature. Figure 11 shows the resulting 2-D grid, containing points mapped from bursts in four IOR_A samples.

We have modified the original *dense unit* definition as follows. Given s samples, we calculate the *density* of a unit as "the number of samples that have points within this unit". If a unit meets a certain density threshold $\lceil \gamma * s \rceil$, where γ is a controllable parameter between 0 and 1, the unit is considered dense. Our experiments used a γ value of 0.5, requiring each dense unit to have points from at least 2 out of the 4 samples. All dense units are marked with a dark shade in Figure 11.

Due to the time drift and shape distortion caused by noise, nodes from different samples representing the same I/O burst could be partitioned by grid lines. As mentioned earlier, for each dense unit, we only check its immediate neighborhood (shown in a lighter shade in Figure 11) for data points potentially from a common burst. We identify *dense neighborhoods* (including the central dense unit) as those meeting a density threshold of $\lceil \kappa * s \rceil$, where κ is another configurable parameter with value larger than γ (e.g., 0.9).

Note that it is possible for the neighborhood (or even a single dense unit) to contain multiple points from the same sample. IOSI further identifies points from the common I/O burst using a *voting* scheme. It allows up to one point to be included from each sample, based on the total normalized Euclidean distance from a candidate point to peers within the neighborhood. From each sample, the data point with the lowest total distance is selected. In Figure 11, the neighborhood of dense unit 5 contains two bursts from IOR_AS3 (represented by dots). The burst in the neighborhood unit (identified by the circle) is discarded using our voting algorithm. As the only "redundant point" within the neighborhood, it is highly likely to be a "fake burst" from other concurrently running I/O-intensive applications. This can be confirmed by viewing the original sample IOR_AS3 in Figure12(b), where a tall spike not from IOR_A shows up around the 1200th second.

5.4 I/O Signature Generation

Given the common bursts from dense neighborhoods, we proceed to sample alignment. This is done by aligning all the data points in a common burst to the average of their x coordinate values. Thereafter, we generate the actual I/O signature by sweeping along the x (time) dimension of the CLIQUE 2-D grid. For each dense neighborhood identified, we generate a corresponding I/O burst at the aligned time interval, by averaging the bursts mapped to the selected data points in this neighborhood. Here we used the bursts after preprocessing, but before WT.

6 Experimental Evaluation

We have implemented the proof-of-concept IOSI prototype using Matlab and Python. To validate IOSI, we used IOR to generate multiple pseudo-applications with different I/O write patterns, emulating write-intensive scientific applications. In addition, we used S3D [31, 56], a massively parallel direct numerical simulation solver developed at Sandia National Laboratory for studying turbulent reacting flows.

Figure 13(a), 13(e) and 13(i) are the true I/O signatures of the three IOR pseudo-applications, IOR_A, IOR_B, and IOR_C. These pseudo-applications were run on the Smoky cluster using 256 processes, writing to the Spider center-wide parallel file system. Each process was configured to write sequentially to a separate file (stripe size of 1MB, stripe count of 4) using MPI-IO. We were able to obtain "clean" signatures (with little noise) for these applications by running our jobs when Titan was not in production use (under maintenance) and one of the file system partitions was idle. Among them, IOR_A represents simple periodic checkpointing, writing the same volume of data at regular intervals (128GB every 300s). IOR_B also writes periodically, but alternates between two levels of output volume (64GB and 16GB every 120s), which is typical of applications with different frequencies in checkpointing and results writing (e.g., writing intermediate results every 10 minutes but checkpointing every 30 minutes). IOR_C has more complex I/O patterns, with three different write cycles repeated periodically (one output phase every 120s, with output size cycling through 64GB, 32GB, and 16GB).

6.1 IOR Pseudo-Application Results

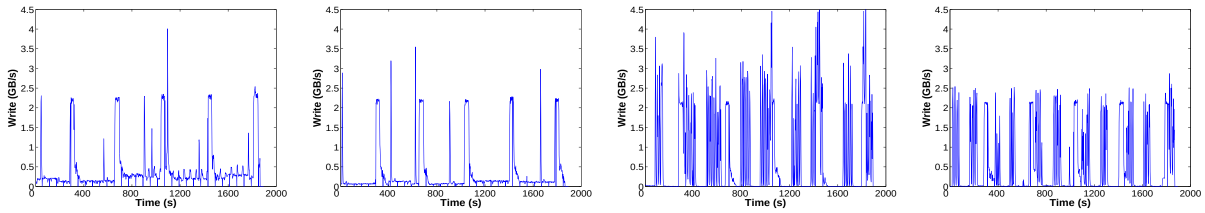
To validate IOSI, the IOR pseudo-applications were were run at different times of the day, over a two-week period. Each application was run at least 10 times. During this period, the file system was actively used by Titan and other clusters. The I/O activity captured during this time is the noisy server-side throughput logs. From the scheduler's log, we identified the execution time intervals for the IOR runs, which were then intersected with the I/O throughput log to obtain per-application samples.

It is worth noting that the I/O throughput range of all of the IOR test cases is designed to be 2-3GB/s. After analyzing several months of Spider log data, we observed that it is this low-bandwidth range that is highly impacted by background noise. If the bandwidth of the application is much higher (say 20GB/s), the problem becomes much easier, since there is less background noise that can achieve that bandwidth level to interfere.

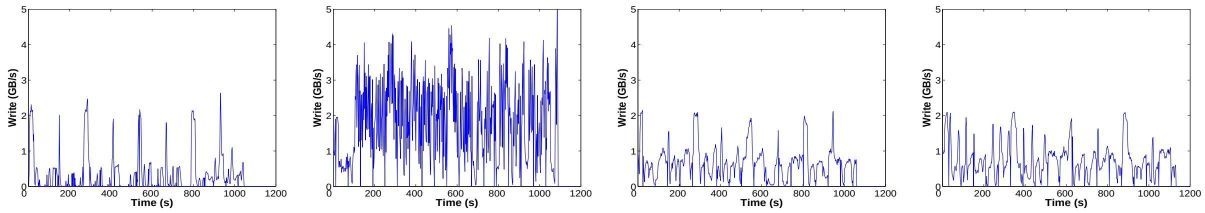
Due to the space limit, we only show four samples for each pseudo-app in Figure 12. We observe that most of them show human-visible repeated patterns that overlap with the target signatures. There is, however, significant difference between the target signature and any individual sample. The samples show a non-trivial amount of "random" noise, sometimes (e.g., IOR_AS1) with distinct "foreign" repetitive pattern, most likely from another application's periodic I/O. Finally, a small number of samples are noisy enough to make the target signature appear overwhelmed (which should be identified as outliers and discarded from signature extraction).

Figure 13 presents the original signatures and the extracted signatures using three approaches: IOSI with and w/o data preprocessing, plus DTW with data preprocessing. As introduced in Section 3, DTW is a widely used approach for finding the similarity between two data sequences. In our problem setting, similarity means a match in I/O bursts from two samples. We used sample preprocessing to make a fair comparison between DTW and IOSI. Note that IOSI without data preprocessing utilizes samples after granularity refinement, to obtain extracted I/O signatures with similar length across all three methods tested.

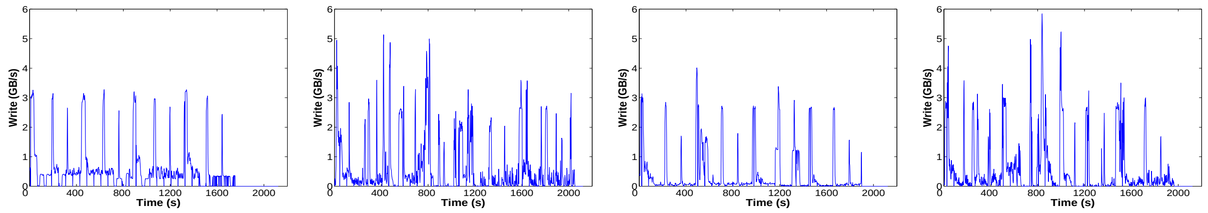
Since DTW performs pair-wise alignment, it is unable to perform effective global data alignment across multiple samples. In our evaluation, we apply DTW as follows. We initially assign a pair of samples as input to DTW, and feed the result along with another sample to DTW again. This process is repeated until all samples are exhausted. We have verified that this approach performs better (in terms of both alignment accuracy and processing overhead) than the alternative of averaging all pair-wise DTW results, since it implicitly carries



(a) $IOR_A S1$ (b) $IOR_A S2$ (c) $IOR_A S3$ (d) $IOR_A S4$

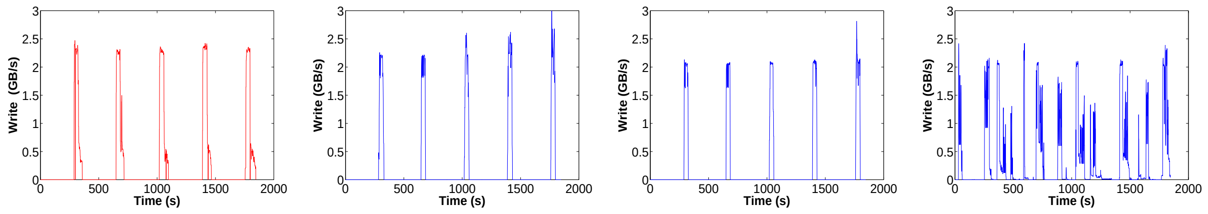


(e) $IOR_B S1$ (f) $IOR_B S2$ (g) $IOR_B S3$ (h) $IOR_B S4$

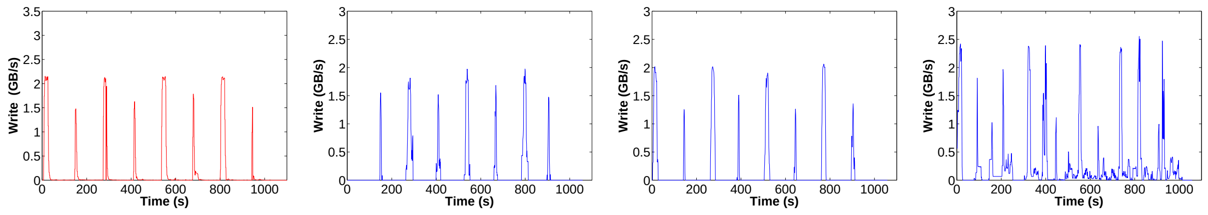


(i) $IOR_C S1$ (j) $IOR_C S2$ (k) $IOR_C S3$ (l) $IOR_C S4$

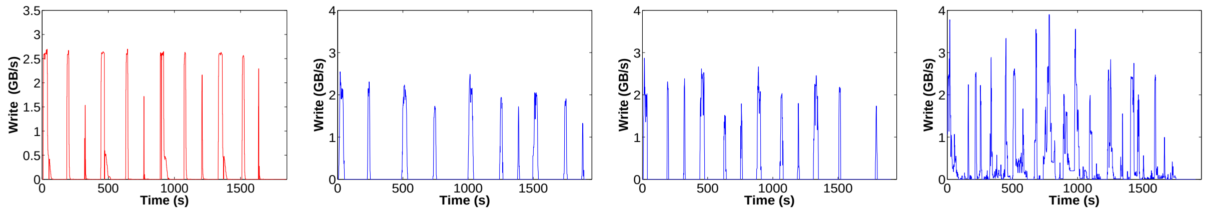
Figure 12: Samples from IOR test cases



(a) IOR_A target signature (b) IOSI w/o data preprocessing (c) IOSI with data preprocessing (d) DTW with data preprocessing

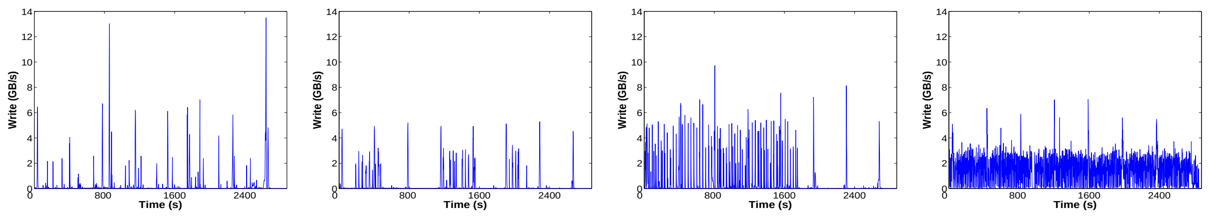


(e) IOR_B target signature (f) IOSI w/o data preprocessing (g) IOSI with data preprocessing (h) DTW with data preprocessing



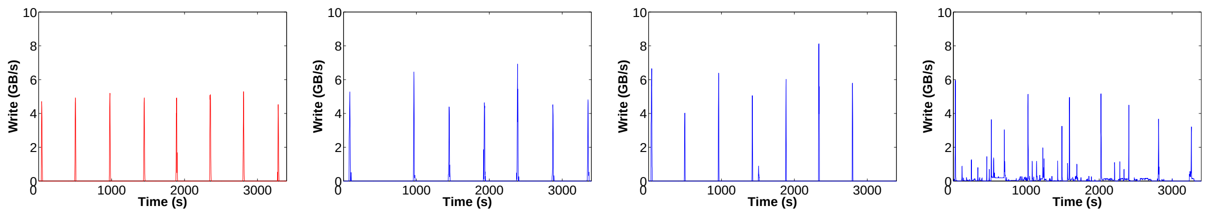
(i) IOR_C target signature (j) IOSI w/o data preprocessing (k) IOSI with data preprocessing (l) DTW with data preprocessing

Figure 13: Target and extracted I/O signatures of IOR test cases



(a) $S3D\_S1$ (b) $S3D\_S2$ (c) $S3D\_S3$ (d) $S3D\_S4$

Figure 14: S3D samples



(a) S3D target I/O signature (b) IOSI w/o data preprocessing (c) IOSI with data preprocessing (d) DTW with data preprocessing

Figure 15: S3D target I/O signature and extracted I/O signature by IOSI and DTW

out global data alignment. Still, DTW generated significantly lower-quality signatures, especially with more complex I/O patterns, due to its inability to reduce noise. For example, DTW's IOR_C (Figure 13(l)) signature appears to be dominated by noise.

In contrast, IOSI (with or w/o data preprocessing) generated output signatures with much higher fidelity. In both cases, IOSI is highly successful in capturing I/O bursts in the time dimension (with small, yet visible errors in the vertical height of the bursts). Without preprocessing, IOSI missed 3 out of the 25 I/O bursts from all pseudo-applications. With preprocessing, however, the symptom is much improved (no burst missed).

6.2 S3D Results

Next, we present results with the aforementioned large-scale scientific application, S3D. S3D was run on Titan and the Spider file system. S3D performs periodic checkpointing I/O, with each process generating 3.4 MB of data. Figure 14 shows selected samples from multiple S3D runs, where we see a lot of I/O interference since both Titan and Spider were being used in production mode. Unlike IOR, we were not able to run S3D on a quiescent Spider file system partition to obtain its "clean" signature to validate IOSI. Instead, we had to use client-side I/O tracing, to produce the target I/O signature (Figure 15(a)). The I/O signature also displays variance in peak bandwidth, common in real-world, large job runs. Again, we extracted the I/O signature from the samples using IOSI (with and without data preprocessing), plus DTW with preprocessing (Figure 15).

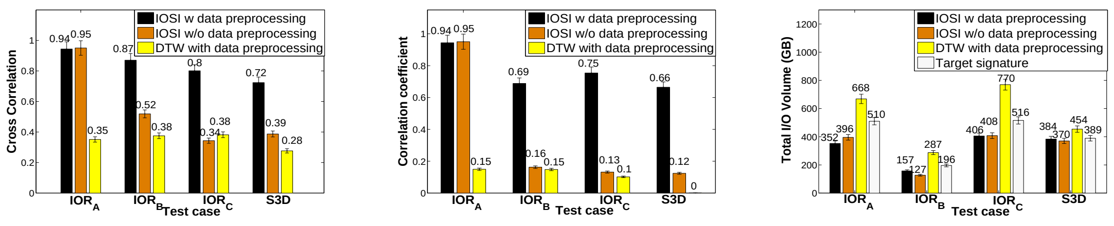
As in the case of IOR, IOSI with data preprocessing performs better than IOSI without data preprocessing and DTW. This result suggests that IOSI is able to the extract I/O signatures of real-world scientific applications from noisy throughput logs, collected from a very busy supercomputing center. While both versions of IOSI missed an I/O burst, the data preprocessing helps deliver better alignment accuracy (discussed in Figures 16(a) and 16(b)). The presence of heavy noise in the samples likely caused DTW's poor performance.

6.3 Accuracy and Efficiency Analysis

We quantitatively compare the accuracy of IOSI and DTW using two commonly used similarity metrics, cross correlation (Figure 16(a)) and correlation coefficient (Figure 16(b)). Correlation coefficient measures the strength of the linear relationship between two samples. Cross correlation [65] is a similarity measurement that factors in the drift in a time series data set. Figure 16 portraits these two metrics, as well as the total I/O volume comparison, between the extracted and the original application signature.

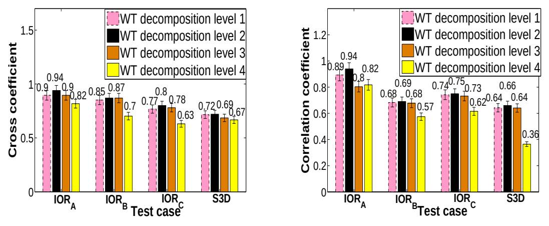
Note that correlation coefficient is inadequate to characterize the relationship between the two time series when they are not properly aligned. For example, with IOR_B, the number of bursts in the extracted signatures by IOSI with and without data preprocessing is very close. However, the one without preprocessing suffers more burst drift compared to the original signature. Cross correlation appears more tolerant to IOSI without preprocessing compared to correlation coefficient. Also, IOSI significantly outperforms DTW (both with preprocessing), by a factor of 2.1-2.6 in cross correlation, and 4.8-66.0 in correlation coefficient.

Note that the DTW correlation coefficient for S3D is too small to show. Overall, IOSI with preprocessing achieves a cross correlation between 0.72 and 0.95, and



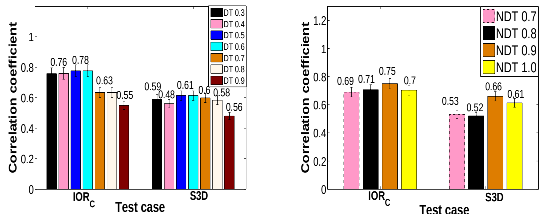
(a) Cross correlation (b) Correlation coefficient (c) Total I/O volume

Figure 16: Result I/O signature accuracy evaluation



(a) Cross correlation (b) Correlation coefficient

Figure 17: IOSI - WT sensitivity analysis



(a) Density threshold (DT) (b) Neighborhood density threshold (NDT)

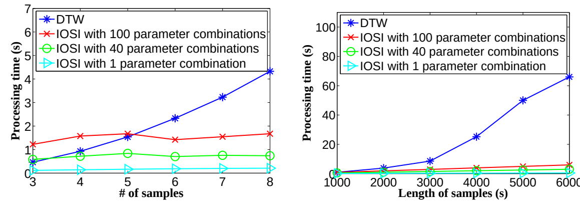
Figure 18: IOSI - Clustering sensitivity analysis

a correlation coefficient between 0.66 and 0.94.

We also compared the total volume of I/O traffic (i.e., the "total area" below the signature curve), shown in Figure 16(c). IOSI generates I/O signatures with a total I/O volume closer to the original signature than DTW does. It is interesting that without exception, IOSI and DTW err on the lower and higher side, respectively. The reason is that DTW tends to include foreign I/O bursts, while IOSI's WT process may "trim" the I/O bursts in its threshold-based burst boundary identification.

Next, we performed sensitivity analysis on the tunable parameters of IOSI, namely the *WT decomposition level*, and *density threshold/neighborhood density threshold in CLIQUE clustering*. As discussed in Section 5, we used a WT decomposition level of 2 in IOSI. In Figures 17(a) and 17(b), we compare the impact of WT decomposition levels using both cross correlation and correlation coefficient. Figure 17(a) shows that IOSI does better with a decomposition level of 2, compared to levels 1, 3 and 4. Similarly, Figure 17(b) shows that the correlation coefficient is the best at the WT decomposition level of 2.

In Figure 18(a), we tested IOSI with different density thresholds $\lceil \gamma * s \rceil$ in CLIQUE clustering, where γ is the controllable factor and s is the number of samples. In IOSI, the default γ value is 50%. From Figure 18(a) we noticed a peak correlation coefficient at γ value of around 50%. There is significant performance degradation at over 70%, as adjacent bursts may be grouped to form a single burst. In Figure 18(b), we tested IOSI with different neighborhood density thresholds $\lceil \kappa * s \rceil$, where κ is another configurable factor with value larger than γ. IOSI used 90% as the default value of κ. Figure 18(b) suggests that lower thresholds perform poorly, as more neighboring data points deteriorates the quality of identified I/O bursts. With a threshold of 100%, we expect bursts from all samples to be closely aligned, which is impractical.

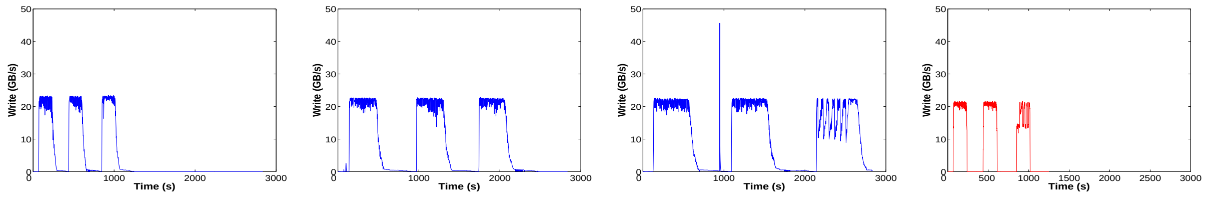


(a) Scalability in # of samples (b) Scalability in sample duration

Figure 19: Processing time analysis

Finally, we analyze the processing time overhead of these methods. IOSI involves mainly two computation tasks: wavelet transform and CLIQUE clustering. The complexity of WT (discrete) is *O(n)* [29] and CLIQUE clustering is $O(C^k + nk)$ [32], where k is the highest dimensionality, n the number of input points, and C the number of clusters. In our CLIQUE implementation, k is set to 2 and C is also a small number. Therefore we assume C^k as a constant, resulting in a complexity of *O(n)*, leading to the overall linear complexity of IOSI. DTW, on the other hand, has a complexity of $O(mn)$ [34], where m and n are the lengths of the two input arrays.

Experimental results confirm the above analysis. In Figure 19(a), we measure the processing time with different sample set sizes (each sample containing around 2000 data points). For IOSI, the processing time appears to stay flat as more samples are used. This is because the CLIQUE clustering time, which is rather independent of the number of samples and depends more on the number



(a) 160-node job (b) 320-node job (c) 640-node job (d) IOSI Result

Figure 20: Weak scaling sample and IOSI extracted I/O signature

of grids, dominates IOSI's overhead. Even with 100 2-D IOSI parameter settings (for the CLIQUE grid size), DTW's cost catches up with 5 samples and grows much faster beyond this point. Figure 19(b) shows results with 8 samples, at varied sample lengths. We see that IOSI processing time increases linearly while DTW displays a much faster growth. To give an idea of its feasibility, IOSI finishes processing three months of Spider logs (containing 80,815 job entries) in 72 minutes.

6.4 Discussion

I/O signatures and application configurations Scientific users tend to have a pattern of I/O behavior. However, they do scale applications with respect to the number of nodes, resulting in similar I/O characteristics. In Figure 20, we show the I/O pattern of a real Titan user, running jobs with three different node counts (160, 320, and 640). From Figures 20(a)-20(c), we observe that the total I/O volume increases linearly with the node count (weak scaling), but the peak bandwidth remains almost constant. As a result, the time spent on I/O also increases linearly. IOSI can discern such patterns and extract the I/O signature, as shown in Figure 20(d). As described earlier, in the data preprocessing step we perform runtime correction and the samples are normalized to the sample with the shortest runtime. In this case, IOSI normalizes the data sets to that of the shortest job (i.e., the job with the smallest node count), and provides the I/O signature of the application for the smallest job size.

Identifying different user workloads Our tests used a predominant scientific I/O pattern, where applications perform periodic I/O. However, as long as an application exhibits similar I/O behavior across multiple runs, the common I/O pattern can be captured by IOSI as its algorithms make no assumption on periodic behavior.

False-positives and missing bursts False-positives are highly unlikely as it is very difficult to have highly correlated noise behavior across multiple samples. IOSI could miscalculate small-scale I/O bursts if they happen to be dominated by noise in most samples. Increasing the number of samples can help here.

IOSI for resource allocation and scheduling The IOSI generated signature can be used to influence both resource allocation as well as scheduling. Large-scale file systems are typically made available as multiple partitions, with users choosing one or more for their runs. A simple partition allocation strategy would be to let the users choose a set of under-utilized partitions. However, when all partitions are being actively used by multiple users, the challenge is in identifying a set of partitions that will have the least interference on the target application. The IOSI extracted signature can be correlated with the I/O logs of the partitions to identify those that will have a minimal impact on the application. If we are unable to find an optimal partition for an application, the scheduler can even stagger such jobs, preferring others in the queue. The premise here is that finding a partition that better suits the job's I/O needs can help amortize the I/O costs over the entire run. These benefits could outweigh the cost of waiting longer in the queue.

7 Conclusion

We have presented IOSI, a zero-overhead scheme for automatically identifying the I/O signature of data-intensive parallel applications. IOSI utilizes *existing* throughput logs on the storage servers to identify the signature. It uses a suite of statistical techniques to extract the I/O signature from noisy throughput measurements. Our results show that an entirely *data-driven* approach, exploring existing monitoring and job scheduling history can extract substantial application behavior, potentially useful for resource management optimization. In particular, such information gathering does not require any developer effort or internal application knowledge. Such a black-box method may be even more appealing as systems/applications grow larger and more complex.

Acknowledgement

We thank the reviewers and our shepherd, Kimberly Keeton, for constructive comments that have significantly improved the paper. This work was supported in part by the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is managed by UT Battelle, LLC for the U.S. DOE (under the contract No. DE-AC05-00OR22725). This work was also supported in part by the NSF grants CCF-0937690, CCF-0937908, and a NetApp Faculty Fellowship.

References

[1] IOR HPC Benchmark, `https://asc.llnl.gov/sequoia/benchmarks/#ior`.

[2] Los Alamos National Laboratory open-source LANL-Trace, `http://institute.lanl.gov/data/tdata`.

[3] Titan, `http://www.olcf.ornl.gov/titan/`.

[4] Wavelet, `http://users.rowan.edu/~polikar/WAVELETS/WTtutorial.html`.

[5] Using Cray Performance Analysis Tools. *Document S-2474-51, Cray User Documents, `http://docs.cray.com`*, 2009.

[6] J. Aach and G. M. Church. Aligning gene expression time series with time warping algorithms. *Bioinformatics*, 17:495–508, 2001.

[7] L. Adhianto, S. Banerjee, M. Fagan, M. Krentel, G. Marin, J. Mellor-Crummey, and N. R. Tallent. Hpctoolkit: Tools for performance analysis of optimized parallel programs. *Concurrency and Computation: Practice and Experience*, 22(6):685–701, 2010.

[8] R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan. Automatic subspace clustering of high dimensional data for data mining applications. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD '98)*, 1998.

[9] L. Bergroth, H. Hakonen, and T. Raita. A survey of longest common subsequence algorithms. In *Proceedings of the 7th International Symposium on String Processing and Information Retrieval (SPIRE'00)*, 2000.

[10] D. J. Berndt and J. Clifford. Using dynamic time warping to find patterns in time series. In *Working Notes of the Knowledge Discovery in Databases Workshop*, 1994.

[11] R. N. Bracewell. *The Fourier transform and its applications*. McGraw-Hill New York, 1986.

[12] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander. LOF: identifying density-based local outliers. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD '00)*, 2000.

[13] S. Bruenn, A. Mezzacappa, W. Hix, J. Blondin, P. Marronetti, O. Messer, C. Dirk, and S. Yoshida. 2d and 3d core-collapse supernovae simulation results obtained with the chimera code. *Journal of Physics: Conference Series*, 180(2009)012018, 2009.

[14] C. S. Burrus, R. A. Gopinath, H. Guo, J. E. Odegard, and I. W. Selesnick. *Introduction to wavelets and wavelet transforms: a primer*, volume 23. Prentice Hall Upper Saddle River, 1998.

[15] S. Byna, Y. Chen, X.-H. Sun, R. Thakur, and W. Gropp. Parallel I/O prefetching using MPI file caching and I/O signatures. In *Proceedings of the ACM/IEEE Conference on Supercomputing (SC '08)*, 2008.

[16] P. Carns, K. Harms, W. Allcock, C. Bacon, S. Lang, R. Latham, and R. Ross. Understanding and improving computational science storage access through continuous characterization. In *Proceedings of the IEEE 27th Symposium on Mass Storage Systems and Technologies (MSST'11)*, 2011.

[17] P. H. Carns, R. Latham, R. B. Ross, K. Iskra, S. Lang, and K. Riley. 24/7 Characterization of petascale I/O workloads. In *Proceedings of the First Workshop on Interfaces and Abstractions for Scientic Data Storage (IASDS'09)*, 2009.

[18] S. Chu, E. J. Keogh, D. Hart, and M. J. Pazzani. Iterative Deepening Dynamic Time Warping for Time Series. In *Proceedings of the 2nd SIAM International Conference on Data Mining (SDM'02)*, 2002.

[19] J. Daly. A model for predicting the optimum checkpoint interval for restart dumps. In *Proceedings of the 1st International Conference on Computational Science (ICCS'03)*, 2003.

[20] J. T. Daly. A higher order estimate of the optimum checkpoint interval for restart dumps. *Future Generation Computer Systems*, 22(3):303–312, 2006.

[21] I. Daubechies. Orthonormal bases of compactly supported wavelets II: Variations on a theme. *SIAM Journal on Mathematical Analysis*, 24:499–519, 1993.

[22] C. de Boor. *A practical guide to splines*. Springer-Verlag New York, 1978.

[23] J. R. Deller, J. G. Proakis, and J. H. Hansen. *Discrete-time processing of speech signals*. IEEE New York, NY, USA, 2000.

[24] P. Du, W. A. Kibbe, and S. M. Lin. Improved peak detection in mass spectrum by incorporating continuous wavelet transform-based pattern matching. *Bioinformatics*, 22(17):2059–2065, 2006.

[25] E.L.Miller and R.H.Katz. Input/output behavior of supercomputing applications. In *Proceedings of the ACM/IEEE Conference on Supercomputing (SC'91)*, 1991.

[26] G. R. Ganger. Generating Representative Synthetic Workloads: An Unsolved Problem. In *Proceedings of the Computer Measurement Group (CMG'95)*, 1995.

[27] Z. Gong and X. Gu. PAC: Pattern-driven Application Consolidation for Efficient Cloud Computing. In *Proceedings of the 18th IEEE/ACM International Symposium on Modeling, Analysis, and Simulation of Computer Telecommunications Systems (MASCOTS'10)*, 2010.

[28] R. Gunasekaran, D. Dillow, G. Shipman, R. Vuduc, and E. Chow. Characterizing Application Runtime Behavior from System Logs and Metrics. In *Proceedings of the Characterizing Applications for Heterogeneous Exascale Systems (CACHES'11)*, 2011.

[29] H. Guo and C. S. Burrus. Fast approximate Fourier transform via wavelets transform. In *Proceedings of the International Symposium on Optical Science, Engineering, and Instrumentation*, 1996.

[30] M. Hauswirth, A. Diwan, P. F. Sweeney, and M. C. Mozer. Automating vertical profiling. In *Proceedings of the 20th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA'05)*, 2005.

[31] E. R. Hawkes, R. Sankaran, J. C. Sutherland, and J. H. Chen. Direct numerical simulation of turbulent combustion: fundamental insights towards predictive models. *Journal of Physics: Conference Series*, 16(1):65, 2005.

[32] M. Ilango and V. Mohan. A Survey of Grid Based Clustering Algorithms. *International Journal of Engineering Science and Technology*, 2(8):3441–3446, 2010.

[33] E. Keogh and C. A. Ratanamahatana. Exact indexing of dynamic time warping. *Knowledge and Information Systems*, 7(3):358–386, 2005.

[34] E. J. Keogh and M. J. Pazzani. Scaling up dynamic time warping for datamining applications. In *Proceedings of the 6th ACM SIGKDD International Conference on Knowledge Discovery and Data Dining*, 2000.

[35] N. G. Kingsbury. The dual-tree complex wavelet transform: a new technique for shift invariance and directional filters. In *Proceedings of the 8th IEEE Digital Signal Processing (DSP) Workshop*, 1998.

[36] D. Kothe and R. Kendall. Computational science requirements for leadership computing. *Oak Ridge National Laboratory, Technical Report*, 2007.

[37] Z. Kurmas and K. Keeton. Synthesizing Representative I/O Workloads Using Iterative Distillation. In *Proceedings of the 11th IEEE/ACM International Symposium on Modeling, Analysis, and Simulation of Computer Telecommunications Systems (MASCOTS'03)*, 2003.

[38] C. Lipowsky, E. Dranischnikow, H. Göttler, T. Gottron, M. Kemeter, and E. Schömer. Alignment of noisy and uniformly scaled time series. In *Proceedings of the Database and Expert Systems Applications (DEXA'09)*, 2009.

[39] Y. Liu, R. Nassar, C. Leangsuksun, N. Naksinehaboon, M. Paun, and S. L. Scott. An optimal checkpoint/restart model for a large scale high performance computing system. In *Proceedings of the International Parallel Distributed Processing Symposium (IPDPS'08)*, 2008.

[40] Y. Long, L. Gang, and G. Jun. Selection of the best wavelet base for speech signal. In *Proceedings of the International Symposium on Intelligent Multimedia, Video and Speech Processing*, 2004.

[41] S. Mallat. A theory for multiresolution signal decomposition: the wavelet representation. *Pattern Analysis and Machine Intelligence*, 11(7):674–693, 1989.

[42] M. P. Mesnier, M. Wachs, R. R. Simbasivan, J. Lopez, J. Hendricks, G. R. Ganger, and D. R. O'Hallaron. //trace: Parallel trace replay with approximate causal events. In *Proceedings of the 5th USENIX Conference on File and Storage Technologies (FAST'07)*, 2007.

[43] R. Miller, J. Hill, D. A. Dillow, R. Gunasekaran, S. Galen, and D. Maxwell. Monitoring Tools For Large Scale Systems. In *Proceedings of the Cray User Group (CUG'10)*, 2010.

[44] K. Mohror and K. L. Karavanic. An Investigation of Tracing Overheads on High End Systems. Technical report, PSU, 2006.

[45] W. G. Morsi and M. El-Hawary. The most suitable mother wavelet for steady-state power system distorted waveforms. In *Proceedings of the Canadian Conference on Electrical and Computer Engineering*, 2008.

[46] M. Müller. Dynamic time warping. *Information Retrieval for Music and Motion*, pages 69–84, 2007.

[47] N. Nieuwejaar, D. Kotz, A. Purakayastha, C. Sclatter Ellis, and M. Best. File-access characteristics of parallel scientific workloads. *IEEE Transactions on Parallel and Distributed Systems*, 7(10):1075–1089, 1996.

[48] S. Oral, F. Wang, D. Dillow, G. Shipman, R. Miller, and O. Drokin. Efficient object storage journaling

in a distributed parallel file system. In *Proceedings of the 8th USENIX Conference on File and Storage Technologies (FAST'10)*, 2010.

[49] B. Pasquale and G. Polyzos. A static analysis of I/O characteristics of scientific applications in a production workload. In *Proceedings of the ACM/IEEE Conference on Supercomputing (SC'93)*, 1993.

[50] T. C. Peterson, T. R. Karl, P. F. Jamason, R. Knight, and D. R. Easterling. First difference method: Maximizing station density for the calculation of long-term global temperature change. *Journal of Geophysical Research: Atmospheres (1984–2012)*, 103(D20):25967–25974, 1998.

[51] J. S. Plank and M. G. Thomason. Processor allocation and checkpoint interval selection in cluster computing systems. *Journal of Parallel and distributed Computing*, 61(11):1570–1590, 2001.

[52] A. Povzner, K. Keeton, A. Merchant, C. B. Morrey III, M. Uysal, and M. K. Aguilera. Autograph: automatically extracting workflow file signatures. *ACM SIGOPS Operating Systems Review*, 43(1):76–83, 2009.

[53] P. C. Roth. Characterizing the I/O behavior of scientific applications on the Cray XT. In *Proceedings of the 2nd International Workshop on Petascale Data Storage: held in conjunction with SC'07*, 2007.

[54] S. Seelam, I.-H. Chung, D.-Y. Hong, H.-F. Wen, and H. Yu. Early experiences in application level I/O tracing on Blue Gene systems. In *Proceedings of the International Parallel Distributed Processing Symposium (IPDPS'08)*, 2008.

[55] G. Shipman, D. Dillow, S. Oral, and F. Wang. The Spider Center Wide File System: From Concept to Reality. In *Proceedings of the Cray User Group (CUG'09)*, 2009.

[56] K. Spafford, J. Meredith, J. Vetter, J. Chen, R. Grout, and R. Sankaran. Accelerating S3D: a GPGPU case study. In *Euro-Par 2009 Parallel Processing Workshops*, 2010.

[57] V. Tarasov, S. Kumar, J. Ma, D. Hildebrand, A. Povzner, G. Kuenning, and E. Zadok. Extracting flexible, replayable models from large block traces. In *Proceedings of the 10th USENIX Conference on File and Storage Technologies (FAST'12)*, 2012.

[58] TOP500 Supercomputer Sites, `http://www.top500.org/`.

[59] A. Uselton, M. Howison, N. Wright, D. Skinner, N. Keen, J. Shalf, K. Karavanic, and L. Oliker. Parallel I/O performance: From events to ensembles. In *Proceedings of the International Parallel Distributed Processing Symposium (IPDPS'10)*, 2010.

[60] J. S. Vetter and M. O. McCracken. Statistical scalability analysis of communication operations in distributed applications. In *Proceedings of the 8th ACM SIGPLAN symposium on Principles and Practices of Parallel Programming (PPoPP'01)*, 2001.

[61] F. Wang, Q. Xin, B. Hong, S. A. Brandt, E. L. Miller, D. D. E. Long, and T. T. Mclarty. File system workload analysis for large scale scientific computing applications. In *Proceedings of the IEEE 21th Symposium on Mass Storage Systems and Technologies (MSST'04)*, 2004.

[62] W. W.-S. Wei. *Time series analysis*. Addison-Wesley Redwood City, California, 1994.

[63] Y. Xu, J. B. Weaver, D. M. Healy, and J. Lu. Wavelet transform domain filters: a spatially selective noise filtration technique. *IEEE Transactions on Image Processing*, 3(6):747–758, 1994.

[64] N. J. Yadwadkar, C. Bhattacharyya, K. Gopinath, T. Niranjan, and S. Susarla. Discovery of application workloads from network file traces. In *Proceedings of the 8th USENIX Conference on File and Storage Technologies (FAST'10)*, 2010.

[65] J.-C. Yoo and T. H. Han. Fast normalized cross-correlation. *Circuits, Systems and Signal Processing*, 28(6):819–843, 2009.

[66] J. W. Young. A first order approximation to the optimum checkpoint interval. *Communications of the ACM*, 17(9):530–531, 1974.

[67] D. Yu, X. Yu, Q. Hu, J. Liu, and A. Wu. Dynamic time warping constraint learning for large margin nearest neighbor classification. *Information Sciences*, 181(13):2787–2796, 2011.