Automatic Characterization of HPC Job Parallel Filesystem I/O Patterns

Joseph P. White
Center for Computational Research
Buffalo, NY, USA
jpwhite4@buffalo.edu

Alexander D. Kofke
University at Buffalo
Buffalo, NY, USA
adkofke@buffalo.edu

Robert L. DeLeon
Center for Computational Research
Buffalo, NY, USA
rldeleon@buffalo.edu

Martins Innus
Center for Computational Research
Buffalo, NY, USA
minnus@buffalo.edu

Matthew D. Jones
Center for Computational Research
Buffalo, NY, USA
jonesm@buffalo.edu

Thomas R. Furlani
Center for Computational Research
Buffalo, NY, USA
furlani@buffalo.edu

ABSTRACT

As part of the NSF funded XMS project, we are actively researching automatic detection of poorly performing HPC jobs. To aid the analysis we have generated a taxonomy of the temporal I/O patterns for HPC jobs. In this paper we describe the design of temporal pattern characterization algorithms for HPC job I/O. We have implemented these algorithms in the Open XDMoD job analysis framework. These I/O classifications include periodic patterns and a variety of characteristic non-periodic patterns. We present an analysis of the I/O patterns observed on the /scratch filesystem on an academic HPC cluster. This type of analysis can be extended to other HPC usage data such as memory, CPU and interconnect usage. Ultimately this analysis will be used to improve HPC throughput and efficiency by, for example, automatically identifying anomalous HPC jobs.

CCS CONCEPTS

• **General and reference** → **Metrics**; **Performance**; • **Theory of computation** → *Pattern matching*;

KEYWORDS

Performance, pattern analysis, I/O

ACM Reference Format:
Joseph P. White, Alexander D. Kofke, Robert L. DeLeon, Martins Innus, Matthew D. Jones, and Thomas R. Furlani. 2018. Automatic Characterization of HPC Job Parallel Filesystem I/O Patterns. In *PEARC '18: Practice and Experience in Advanced Research Computing, July 22–26, 2018, Pittsburgh, PA, USA.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3219104.3219121

1 INTRODUCTION

As part of the NSF funded XD Metrics Service (XMS) project, we are developing techniques for the automatic detection of poorly performing HPC jobs. We have developed several complementary detection mechanisms including simple analyses, such as comparing the actual usage of an HPC job to the original requested resources, to a more complex analysis that uses machine learning algorithms to classify jobs [7]. The source data for these analyses include the job accounting information, application information and job performance data recorded from CPU hardware counter values, I/O metrics and O/S counters. Another focus of the XMS project is facilitating the characterization of HPC system workloads. Workload analyses are an essential part of the machine lifecycle. Hardware and software designers, end users and facility operators all can benefit from details on HPC cluster usage.

One interesting avenue of research is the identification of poorly performing HPC jobs based on anomaly detection. The overall idea is that if we have a characterization of the properties of a typical HPC job in a given category then we will be able to identify atypical jobs. This categorization and anomaly detection procedure must be an automatic process. The sheer volume of the data means that it is not feasible for system support staff to manually check all HPC jobs. For example, in 2016 there were over 4 million jobs using over 100 million CPU hours on the HPC cluster at our center. Additionally, job I/O characterization is an important prerequisite for many areas of study including the development of I/O aware job scheduling algorithms and parallel filesystem design and tuning.

2 RELATED WORK

The study of HPC job I/O behavior has been an active area research for three decades from Miller and Katz [17] in the early 1990's to recent work Luu et al. [14]. HPC job I/O behavior is bursty, but with regular cycles and many compute-intensive HPC jobs use very little I/O [17, 18, 20, 27]. HPC jobs have distinct phases of I/O operation, which differs between and is characteristic of different application software [21].

Buneci and Reed [2] generated signatures containing structural and temporal features from time-series performance metrics and used these to distinguish between well- and poor-performing applications. They used a heuristic algorithm that detected several patterns in timeseries data including periodic patterns.

Data collection strategies for HPC job I/O can be broadly organized into three categories: Instrumentation and analysis of the application software, analysis of data from filesystem I/O nodes and instrumentation/analysis of data from the compute nodes.

PEARC '18, July 22–26, 2018, Pittsburgh, P A, USA
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-6446-1/18/07…$15.00
https://doi.org/10.1145/3219104.3219121

Application instrumentation provides detailed information about filesystem I/O usage and accurate timing information. However, it typically requires recompiling the code or relinking with dedicated data collection tools (such as `Score-P` [16], `MPIProf` [10] or `IOT` [9]). Tools that require recompilation are typically not enabled for all jobs on an HPC resource so I/O characterization must be done on a case by case basis. Chang et. al [5] chose 15 different cases across 13 different applications and instrumented them using the `MPIProf` and `IOT` tools. Tools such are Darshan [3] are lightweight and can be enabled by default for all jobs on an HPC resource. However, there are some codes that are incompatible with Darshan so sites that have it enabled provide a mechanism for users to disable it. Unfortunately, this results in many jobs without instrumented data. For example, Luu et al. [14] used Darshan data from a period of four years on three large scale supercomputers with an average Darshan coverage of 20% to 40%.

Log data collected from filesystem I/O nodes has the advantage that it is available regardless of the HPC jobs that are running, however it is challenging to identify the HPC job that caused a given I/O request. Liu et. al [12] developed a system for automatically identifying I/O intensive jobs from the logs of the filesystem I/O nodes. They were able to mine the correlation between I/O traffic and application executions to obtain information on application I/O patterns. This data mining relies on the assumption that a given application has a consistent I/O pattern that remains the same between different HPC jobs.

The compute nodes on an HPC resource can be instrumented using one of the many available tools, such as PCP [24], `tacc_stats` [6], Ganglia [15] or LDMS [1]. A benefit of this approach is that I/O metrics can be collected for all jobs without requiring them to be recompiled and it is straightforward to associate the I/O data collected on a compute node with the job that ran on the node. The disadvantage is that data is not as fine grained as the event data obtained by instrumenting the job software directly.

3 CCR RESOURCE CHARACTERIZATION

This study looked at data collected on the Center for Computational Research's (CCR) academic HPC resource, Rush, which is a heterogeneous system containing approximately 700 x86_64 compute nodes. All compute nodes have local hard disk storage and are interconnected with Gigabit Ethernet and InfiniBand. Rush has two global filesystems: a 3PB IBM GPFS [23] high-performance parallel file system for the global shared parallel scratch space and Isilon IQ36000x Storage Arrays for general network file system access. The system uses Slurm as the resource manager.

Every compute node has the Performance Co-Pilot (PCP) [24] software running and configured to collect a wide range of system metrics every thirty seconds. The Slurm job prolog and job epilog scripts run a command that triggers PCP data collection on the compute nodes assigned to each HPC job. This ensures that metric data is recorded immediately before and after every HPC job. The metrics collected by PCP include the O/S counters such as the load average, Linux kernel CPU statistics, information about processes from the `/proc` filesystem and information from hardware drivers reported to the O/S such as the amount of data transferred to/from GPFS.

Data are analyzed using the SUPReMM Job Summarization software [4] that processes the compute-node level PCP archives to generate job-level summary information. The SUPReMM Job Summarization software integrates with Open XDMoD [19], which provides a rich set of analysis and charting tools for HPC job accounting and performance metrics. The SUPReMM Job Summarization software is written in python and has a modular plugin-based architecture, which makes it very easy to extend to add new data analyses. The software includes plugins that generate simple statistics, such as the overall CPU usage for the job, the total amount of data transferred to/from any filesystems, and the maximum memory usage. There are also more complicated analysis plugins such as job catastrophe detection that finds step-function timeseries behavior of the CPU core L1D cache load rate to detect a catastrophic drop in job performance.

Compute node sharing is enabled on Rush. The memory and CPU usage of individual HPC jobs is restricted using Linux cgroups. The summarization software parses the cgroup information so that the memory and CPU usage metrics can be computed correctly. Unfortunately, filesystem I/O is not constrained by cgroups and there are no metrics collected that allow the correct attribution of I/O data for a given job on a shared node. Therefore, we have excluded shared jobs from this analysis as there is no accurate I/O data available.

4 JOB CHARACTERIZATION TECHNIQUES

We performed an ad-hoc analysis of jobs on our academic HPC cluster using XDMoD. We selected a few hundred jobs that used the GPFS filesystem and looked at the timeseries I/O data. We observed that there were a small number of common I/O patterns: main I/O usage near the start of the job, main I/O usage near the end of the job, I/O activity at the start and end but not during the job, low I/O at the start or end but high in the middle. We also observed jobs with approximately regular activity throughout the job and regular periodic I/O activity. We assume that these I/O patterns are signatures of common use cases for HPC job I/O.

In this section we describe the two complementary I/O pattern detection algorithms that we implemented as plugins for the SUPReMM job summarization software.

4.1 Simple job classifier

Given the observed job I/O behavior we wrote a simple heuristic classification algorithm that categorized jobs based on a very coarse measure of when the majority of the I/O occurred. This simple classifier was not intended to comprehensively categorize all job types; we expected that there would be jobs not correctly identified by a simple algorithm and there would be jobs that do not fit into the simple categories. However, if the simple algorithm has good accuracy then it has captured the principle characteristics. Since we are interested in coarse grained temporal behavior of jobs, the algorithm splits the job into four equal width time intervals and compares measurements between these four sections. The heuristic algorithm classified the read and write data separately. For jobs that run on multiple compute nodes, the mean value per compute node was used. The algorithm produces a single read and a single write classification for each job.

The algorithm is as follows: If the average amount of data transferred per node for the job was less than a threshold (1MB) then the job is categorized as having no significant filesystem usage (NO USAGE). The timeseries data for the job is then split into four equally spaced sections and the amount of data transferred during each section is calculated ($s_0 \dots s_3$). The coefficient of variation of these four measurements is computed and if it is below a threshold (0.25) then the job is categorized as having approximately uniform data transfer (~UNIFORM) otherwise the job is classified according to Algorithm 1 below. The job categories are START meaning that the majority of the I/O occurs at the beginning of the job, END for jobs that perform most of the I/O at the end, HILL for jobs that perform I/O predominately in the middle and CANYON for jobs that have I/O at the start and end but not during the middle. Jobs that have I/O that are not matched by the algorithm are marked as OTHER.

Algorithm 1 Simple job classification

```
if $s_0 > s_1 + s_2 + s_3$ then
    if $s_3 > 2(s_1 + s_2)$ then
        $a \leftarrow$ CANYON
    else
        $a \leftarrow$ START
    end if
else if $s_3 > s_0 + s_1 + s_2$ then
    if $s_0 > 2(s_1 + s_2)$ then
        $a \leftarrow$ CANYON
    else
        $a \leftarrow$ END
    end if
else if $s_1 + s_2 > 2(s_0 + s_3)$ then
    $a \leftarrow$ HILL
else if $\min(s_0, s_3) > 2\max(s_1, s_2)$ then
    $a \leftarrow$ CANYON
else
    $a \leftarrow$ OTHER
end if
```

4.2 Periodicity Detection

In addition to the simple job classifier described in the previous section, we also constructed a model that seeks periodic I/O behavior. We implemented the `AUTOPERIOD` algorithm described in [26] in python. This algorithm is designed to automatically detect periodicity in time series data. We chose this algorithm because of the relative ease of implementation in python using existing libraries and because of the anticipated computational efficiency and accuracy. Here we give a brief overview of the method. Full details of the algorithm are provided in [26].

The `AUTOPERIOD` algorithm uses a two-tier approach, by considering the information in both the periodogram and the autocorrelation function ($\mathcal{ACF}$). The algorithm to determine whether a job is periodic and to compute the period with the highest spectral power has the following individual steps:

- Compute the periodogram of the data using the Lomb-Scargle method.

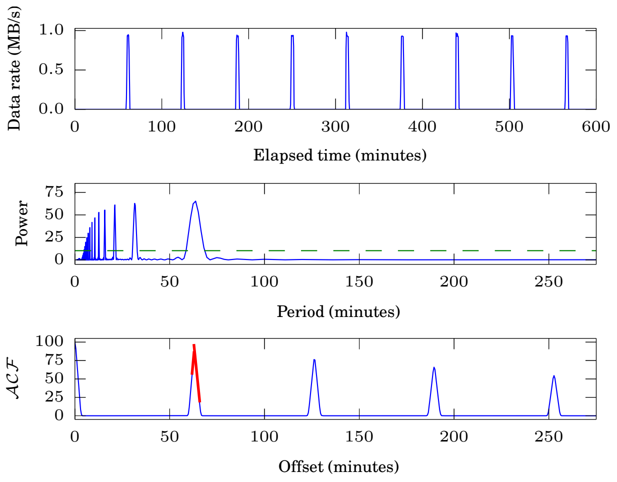

Figure 1: Illustration of the `AUTOPERIOD` algorithm steps. The top plot shows the data read from `/scratch` for an HPC job. The middle plot shows the periodogram computed from the I/O data and the bottom plot shows the autocorrelation function for the data. Note the different x-axis scales. The red line shows the detected hill using two segment linear approximation.

- Determine the statistically significant periods. If there are none, the job is non-periodic.
- Compute the circular AutoCorrelation Function ($\mathcal{ACF}$).
- From the candidate periods with significant spectral power in the periodogram, check to see if the period is on a hill or valley of the $\mathcal{ACF}$. The points are checked in order from highest to lowest power.
- Choose the candidate period with the highest power that is on a hill of the $\mathcal{ACF}$. This is the period of the job. If there are no peaks on a hill of the $\mathcal{ACF}$ the job is non-periodic.

The algorithm is illustrated in Figure 1, which shows the data from an HPC job that had periodic reads from the filesystem. The top plot shows the filesystem read rate over time. The periodogram calculated from this data is shown in the second plot (note the different x-axis scale). The dashed line in the periodogram shows the computed significance threshold. The circular AutoCorrelation Function ($\mathcal{ACF}$) is shown in the bottom plot. In this case, only one of the candidate points in the periodogram corresponds to a hill on the $\mathcal{ACF}$. It is also the point with the highest power. The result of the hill detection algorithm is shown as a bold red line.

To compute the periodogram we used an implementation of the Lomb-Scargle algorithm [13] and [22] from the Astropy python library [25]. The Lomb-Scargle algorithm was chosen because it does not require evenly sampled measurements: the job data does not have even spacing at the beginning and end of the jobs and has approximately even spacing during the job (there is some jitter in the measurement time).

Initially we used the Monte-Carlo method [26] to determine the statistical significance of the candidate peaks in the periodogram. Unfortunately, this was too computationally expensive slowing the model to an unacceptable extent so instead we used an estimate of the statistically significant periods described in [8]. The Monte Carlo computation increased the runtime of the overall process by 30% on average. With few exceptions, the faster method provides an adequate replacement for the more rigorous Monte Carlo calculation.

The hill detection on the $\mathcal{ACF}$ is performed by choosing a specific portion of the $\mathcal{ACF}$ around the candidate period and using a two segment linear approximation. The location of the split between the two linear segments is chosen as the split location that minimizes the total approximation error. The angle between the two linear segments can then be trivially calculated to determine whether the candidate period is on a hill or valley.

5 VALIDATION

We tested our implementation of the AUTOPERIOD algorithm against reference periodic data that we created for the task. We also attempted to reproduce the results from the original paper. We were not able to obtain the MSN query log data used in the paper so we used equivalent data from Google's Trends service. We were able to reproduce identical results from the paper using this data, which gave us confidence that the algorithm implementation was correct.

During initial testing with job I/O data, we noticed some cases where the algorithm identified periodic behavior but the I/O data did not appear to be periodic or was periodic but appeared to have a different period than the main period identified. Therefore, we tried to quantify the accuracy of the period detection algorithm on HPC job I/O data. We attempted to develop a strategy to automatically verify the period detection result by comparing the predicted signal to the job data. However, we were not able to develop a reliable automatic verification method. This was mainly because the AUTOPERIOD algorithm does not provide any phase or amplitude information and the simple phase detection approaches that we tried were not successful.

Since we were not able to easily automatically verify the results, we used a manual approach. We developed a simple web-based application that allowed rapid manual verification. The web interface displayed a plot of the data rate for a randomly selected job alongside a visualization of the period detection result. If the job was detected as periodic the period result was plotted as a sine wave with the phase aligned with the maximum value of the job data rate. The interface had three web form buttons "correct classification", "incorrect classification" and "not sure". When the form button was clicked the page refreshed with the plots for another job that was selected at random. This simple interface was very easy to use and we were able to manually verify thousands of jobs with minimal effort.

The simple categorization algorithm was manually verified similarly to the period detection. We created a separate web-based application that displayed the data rate plot for a job selected at random and had web form buttons for START, END, CANYON, HILL OTHER and ~UNIFORM, and PERIODIC[1].

Overall the manual classification of jobs was in satisfactory agreement with the automatic classifiers.

[1]We split the OTHER and ~UNIFORM into distinct categories after the verification software was written.

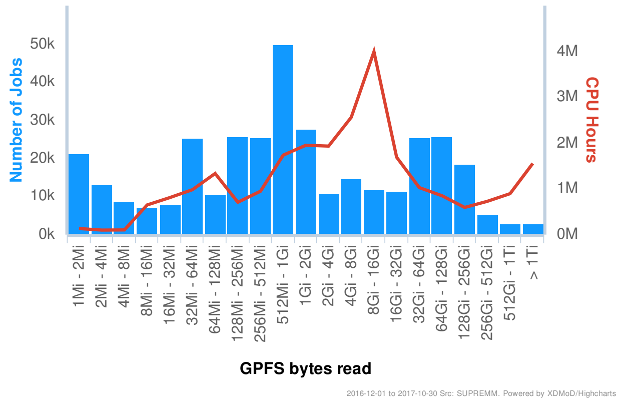

Figure 2: XDMoD plot showing the number of jobs and core hours for non-shared HPC jobs broken down by GPFS data reads from December 2016 to October 2017. There were 1.2 M jobs (~42 M core hours) with less than 1 MB GPFS reads. and 150,000 jobs (~6.6 M core hours) with no usage data (not shown).

6 RESULTS

The results presented here are from GPFS filesystem data collected on the academic HPC resource, Rush, at CCR. The data presented here cover the period from December 2016 to October 2017 inclusive. During this time the filesystem was used as high speed scratch space — the system policy was that files older than three weeks were subject to removal by a scrubber and the filesystem was not backed up.

The job analysis was performed on a subset of jobs that ran on Rush during the studied time period: we excluded shared-node jobs because we do not have reliable information about I/O usage for each job. We also excluded jobs that were shorter than ten minutes to ensure that there were sufficient measurements to be able to resolve any time series patterns (the metric collection period was 30 seconds).

There were 1.7 million jobs longer than ten minutes out of a total of 3.7 million jobs (95.6 million and 96.1 million core hours respectively). Of these 1.7 million jobs there were ~970,000 shared jobs and ~720,000 exclusive jobs (22 million and 74 million core hours respectively). The data coverage for the studied jobs is good — 92% of jobs by core hours had performance data (625,000 jobs using 68 million core hours).

The breakdown of I/O usage for the non-shared jobs by total amount of data read from `/scratch` is shown in Figure 2. The amount of data read varies significantly between different jobs. The most frequent read sizes are 0.5 − 1 GB by job count and 64 − 128 GB by CPU hours. The breakdown of application usage for jobs that read more than 1 MB data is shown in Figure 3. The "uncategorized" label corresponds to jobs that did not match a known community application and are mostly composed of user written codes. The usage breakdowns by amount of data written to `/scratch` are qualitatively the same.

A large proportion of the studied jobs did not use scratch space at all. There were 470,000 jobs (38 million core hours) with no scratch

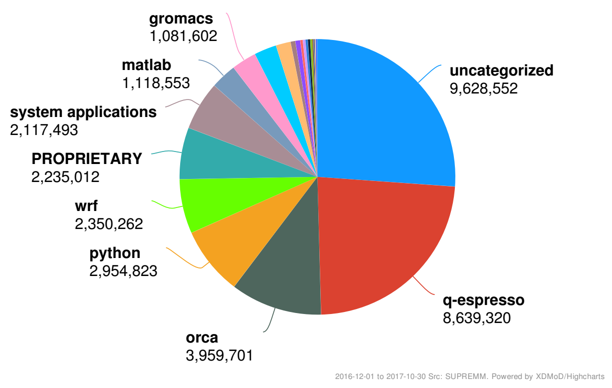

Figure 3: Application core hours for jobs that had GPFS usage data and $> 1MB$ GPFS reads from December 2016 to October 2017.

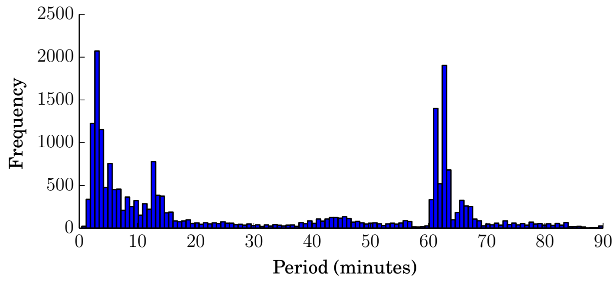

Figure 4: Histogram of write data period for jobs that were detected as periodic and had approximately uniform writes over time. The x-axis is truncated at 90 minutes. There are 1005 jobs with write period longer than 90 minutes.

usage. The top three application categories for these jobs were a commercial HPC software package (195,000 jobs), the molecular dynamics package GROMACS (140,000 jobs) and the set of uncategorized applications (96,000 jobs).

The results from the simple classifier algorithm are shown in Table 1. Aside from the NO USAGE category, OTHER and ~UNIFORM are the most common categories. It is interesting to note that with a few exceptions the diagonal entries, that is those in which the read and write classifications are the same, are generally larger than the non-diagonal entries.

There were ~22,500 jobs that had ~UNIFORM or OTHER write classification from the simple classifier and were also detected as periodic. The histogram of the detected periods is shown in Figure 4. There are distinct peaks in the histogram around 3 minutes, 13 minutes and 62 minutes there is also a bulge around 45 minutes. 95% of jobs in the 60-64 minute period were scavenger queue jobs associated with a single PI. Using Open XDMoD we checked the job launch scripts and found that the jobs were all configured to use checkpointing using the DMTCP software with a one hour checkpoint time.

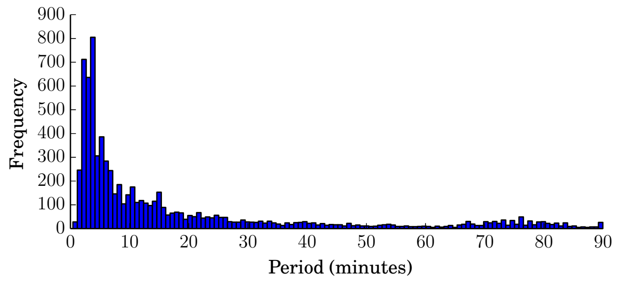

Figure 5: Histogram of read data period for jobs that were detected as periodic and had approximately uniform reads over time. The x-axis is truncated at 90 minutes. There are 788 jobs with read period longer than 90 minutes.

There were ~8,300 jobs with read periodic classification and ~UNIFORM or OTHER from the simple classifier. The longest read period was ~9 hours. The histogram of the read periods is shown in Figure 5. The read histogram is qualitatively different to the write histogram. There is a peak around 3 minutes, but there are no sharp peaks in the region of 13 and 62 minutes.

We briefly examined the job category classifications as a function of application. As would be expected if the classifications were independent of application, most applications had either OTHER or ~UNIFORM as the most common job classification. Several did not, with either START or END being the most common classification. However, the number of jobs falling into these application categories was rather small and might simply represent local usage rather than any inherent tendency in the application performance. We examined the percentage of periodic jobs as a function of application as well. A similar trend was noticed in that while some applications such as WRF, LAMMPS and Quantum ESPRESSO had a substantially higher percentage of periodic jobs it is difficult to determine if this represents an inherent property of the application performance or a variation in local usage. We also looked at the percentage of periodic jobs as a function of wall duration. We found that for longer jobs, those greater than 12 hours, that the fraction of periodic jobs is approximately double the numbers shown in Table 2. This could be due to a greater usage of check-pointing in the longer jobs.

6.1 Classification failures

During the manual verification of the results, we observed some cases where the periodic detection gave incorrect or unexpected results. As expected, we also observed many complex I/O patterns that were not well captured by the simple heuristic classifier. In this section we list a few examples.

Figure 6 shows an example job that had periodic write behavior with two main frequencies. The AUTOPERIOD algorithm detected 13 minutes as the main period, which corresponds to the distance between each pair of peaks in the plot. The other period is approximately one hour, which was also a statistically significant period in the periodogram, but the AUTOPERIOD algorithm selected the 13 minute period since it has a higher power in the periodogram.

Table 1: Number of jobs broken down by classification type

read \ write	HILL	START	END	~UNIFORM	OTHER	CANYON	NO USAGE
HILL	1166	652	170	30	131	116	138
START	218	4829	2053	7694	2010	2642	2356
END	3366	4395	4263	518	8899	1840	397
~UNIFORM	106	621	521	9554	1336	241	545
OTHER	240	837	713	2874	9731	833	112
CANYON	1389	730	2208	1130	2793	11397	847
NO USAGE	1468	775	32885	7077	2234	2824	492980

Table 2: Percentage of jobs broken down by classification type. The right-hand column is the overall percentage of jobs that are periodic and had approximately uniform I/O over time.

	Percent of all jobs by Category							
Direction	HILL	START	END	~UNIFORM	OTHER	CANYON	NO USAGE	Periodic (% overall)
Read	0.38	3.42	3.72	2.03	2.41	3.22	84.8	1.31
Write	1.25	2.02	6.72	4.53	4.26	3.12	78.1	3.55

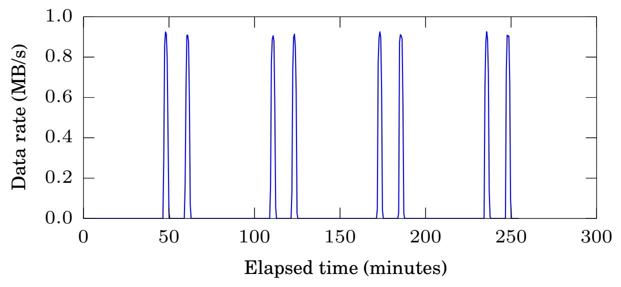

Figure 6: Example job that had periodic writes with two periods. This job ran for 24 hours and the pattern repeated for the whole job. Only the first few cycles are shown in the plot for clarity.

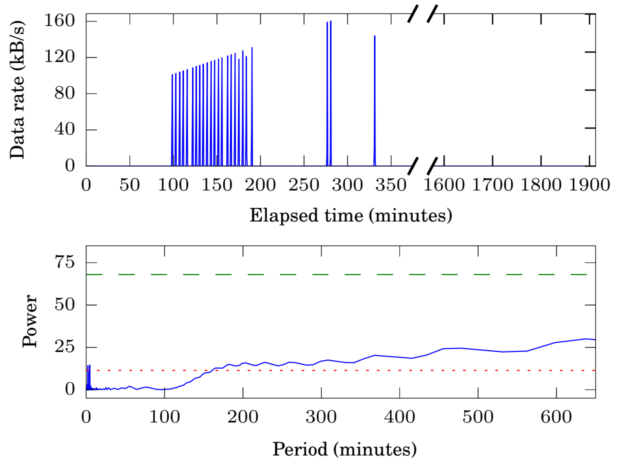

Figure 7: The read data rate over time for a job that was incorrectly identified as periodic and the associated periodogram. The green dashed line indicates the significance level $p < 0.01$ computed using a Monte Carlo method. The red dotted line indicates the significance level estimated using the exponential assumption.

Figure 7 shows an example job that was incorrectly marked as periodic because the estimate of the significance threshold for the periodogram was much less than the actual significance threshold (as determined by the Monte Carlo method). In this case there the $\mathcal{ACF}$ had enough noise that one of the (incorrect) candidate periods on the periodogram looked enough like a hill to the hill detection algorithm.

Figure 8 shows an example where the heuristic classifier gave the correct result as per the design, but which shows the limitations of the simple model. The job does have the majority of the I/O during the first quarter, however there appears to be multiple distinct phases of operation over the coarse of the job.

Figure 9 shows an example job where the `AUTOPERIOD` algorithm identified a period that was not immediately obvious to the manual classifiers. This job has a repeating pattern that stops after ~8 hours and then there is no write activity for the rest of the job (~34 hours). The periodogram identifies a short period around 13 minutes as well as a repeating pattern with period 196 minutes. The longer period has the higher power and is the one that the algorithm selects. Note that the fact that there was no activity for the majority of the job does not affect the periodic detection.

7 CONCLUSIONS

We have implemented two complementary algorithms to classify I/O data of HPC jobs: a simple heuristic classifier that is based on

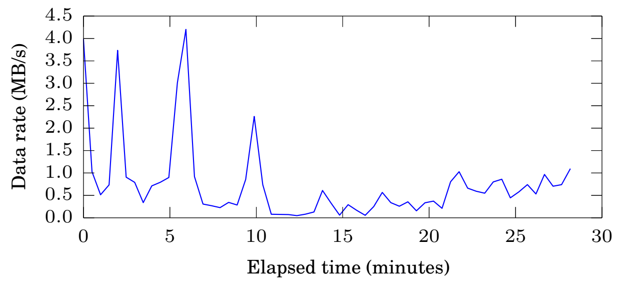

Figure 8: Example job that was classified as a START job by the heuristic classifier.

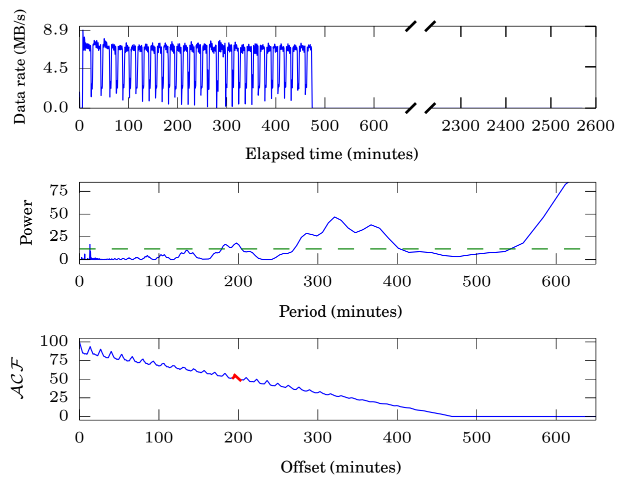

Figure 9: The write rate to `/scratch` for an example job and the corresponding periodogram and $\mathcal{ACF}$. No data was written by this job after the first ~8 hours.

simple assumptions about typical I/O patterns in HPC software and a classifier that detects periodic patterns and determines their period. These were implemented in the SUPReMM job summarization software that forms part of the Job performance data workflow for Open XDMoD.

The categorization software was run on the I/O metrics for the GPFS-based `/scratch` filesystem for all HPC jobs on an academic HPC cluster. We were able to detect the signature of periodic writes consistent with check-pointing. The simple heuristic model worked well but there were a sizable number of jobs that did not fit into the simple categories.

One of the original motivations for this work was to categorize jobs to help aid in job anomaly detection. The categorization data on its own is not expected to be sufficient to detect job anomalies. However, an example of a direct application of this work is for job checkpointing identification. Users of the "scavenger" partition on Rush are expected to be using checkpointing because the jobs may be preempted. Users whose jobs are categorized as no significant usage or do not have periodic writes can easily be identified. Support staff can then contact the users to see if they need help setting up checkpointing for their jobs.

8 FUTURE WORK

Currently the periodicity detection summarization plugin stores all I/O data in memory before computing the periodicity. The memory usage requirements are still fairly modest: ~100 MB data required to process a job that had a one month wall time. However, the plugin runs at the same time as the other analyses (there are currently thirty plugins, most of which compute simple statistics about job metrics). The current summarization software uses ~1 GB on average when processing a typical HPC job. Ideally the individual plugins would try to minimize their memory footprint. One way to reduce the memory requirements of periodicity detection summarization plugin would be to change to a streaming algorithm by using an incremental calculation of the Fourier Transform.

The results presented here use the combined I/O data from all compute nodes for each HPC job. However, different HPC applications have different I/O characteristics on different nodes and combining the measurements from all nodes obscures these differences. Luu et. al. [14] identified three common spacial modes: local file I/O, where each compute node writes to independent files, subset file I/O, where a subset of compute nodes are responsible for the file I/O and serial I/O where only one compute node performs file I/O. Our analysis works fine for serial I/O and will work fine for the subset and local modes assuming the compute nodes are lock-step synchronized. During testing we did observe the existence of HPC jobs that used the local I/O paradigm and where the compute node had periodic I/O but there was a phase difference between individual compute nodes. We have not done any analysis to determine how common this case is, but we note that this class of job would still be marked as periodic using our algorithm. In future, we want to investigate including node analysis with the temporal analysis so see how much this improves job characterization.

This study focused on the scratch filesystem I/O for a GPFS filesystem. The main reason for initially looking at the data for the scratch filesystem is that the scratch I/O is by design (by either the developer of the application, or by the configuration setting of the user). The periodic detection library and time segment analysis codes are independent of the metric source data, so it is trivial to run these algorithms on any available metrics. In addition to the GPFS parallel filesystem, the HPC resources at CCR have local disks on the compute nodes and use Isilon network attached storage for the `/projects` and `/home` directories. We collect usage metrics for all of these filesystems, so these data could easily be added to the analysis. The two obvious ways to include the extra filesystems would be to combine all of the filesystem I/O into a single metric and run the categorization algorithm on that or treat them separately. The disadvantage of combining all data is that it hides the difference between read and write operations if the job copies a file from one filesystem to another. The disadvantage of handling them separately is the increased amount of data will make analysis more complex.

The I/O data for shared jobs were excluded from this analysis but shared jobs comprise a reasonable fraction of the resource usage. For example, in 2017 shared jobs on Rush comprised ~19% of jobs by core-hour (~41% of jobs by job count). It may be possible to use a similar technique as [11] to extract the I/O signatures for a subset of the shared jobs.

Splitting the job into four sections is equivalent to a very coarse smoothing of the data. Increasing the number of split points (using finer grained smoothing) should be able to distinguish between different types of job within the broader categories. You could also try using a hill detection algorithm to try to find where the steep slope is. For example in the case of a START job you might want to distinguish between a job that performs max rate I/O for a short period of time verses a job that performs lower rate I/O for a longer period of time.

ACKNOWLEDGMENTS

The authors would like to thank staff members of the Center for Computational Research University at Buffalo for their assistance with the manual classification of the HPC job I/O patterns.

This analysis would not have been possible without the support and expertise of the XDMoD development team, including Cynthia Cornelius, Rudra Chakraborty, Steven M. Gallo, Jeffrey T. Palmer, Benjamin Plessinger, Ryan Rathsam, Nikolay Simakov, Jeanette Sperhac, and former members Thomas Yearke, Amin Ghadersohi and Ryan Gentner.

This work was sponsored by the National Science Foundation (NSF) under award ACI 1445806 for the XD Metrics Service (XMS).

REFERENCES

[1] Anthony Agelastos, Benjamin Allan, Jim Brandt, Paul Cassella, Jeremy Enos, Joshi Fullop, Ann Gentile, Steve Monk, Nichamon Naksinehaboon, Jeff Ogden, Mahesh Rajan, Michael Showerman, Joel Stevenson, Narate Taerat, and Tom Tucker. 2014. The Lightweight Distributed Metric Service: A Scalable Infrastructure for Continuous Monitoring of Large Scale Computing Systems and Applications. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC '14)*. IEEE Press, Piscataway, NJ, USA, 154–165. https://doi.org/10.1109/SC.2014.18

[2] Emma S. Buneci and Daniel A. Reed. 2008. Analysis of Application Heartbeats: Learning Structural and Temporal Features in Time Series Data for Identification of Performance Problems. In *Proceedings of the 2008 ACM/IEEE Conference on Supercomputing (SC '08)*. IEEE Press, Piscataway, NJ, USA, Article 52, 12 pages.

[3] Phillip Carns, Robert Latham, Robert Ross, Kamil Iskra, Samuel Lang, and Katherine Riley. 2009. 24/7 Characterization of petascale I/O workloads. In *2009 IEEE International Conference on Cluster Computing and Workshops*. IEEE, Piscataway, NJ, USA, 1–10. https://doi.org/10.1109/CLUSTR.2009.5289150

[4] University at Buffalo Center for Computational Research. 2016. SUPReMM Job Summarization Software. https://github.com/ubccr/supremm.

[5] Yan-Tyng Sherry Chang, Henry Jin, and John Bauer. 2016. Methodology and Application of HPC I/O Characterization with MPIProf and IOT. In *Proceedings of the 5th Workshop on Extreme-Scale Programming Tools (ESPT '16)*. IEEE Press, Piscataway, NJ, USA, 1–8. https://doi.org/10.1109/ESPT.2016.005

[6] Todd Evans, William L. Barth, James C. Browne, Robert L. DeLeon, Thomas R. Furlani, Steven M. Gallo, Matthew D. Jones, and Abani K. Patra. 2014. Comprehensive Resource Use Monitoring for HPC Systems with TACC_Stats. In *Proceedings of the First International Workshop on HPC User Support Tools (HUST '14)*. IEEE Press, Piscataway, NJ, USA, 13–21. https://doi.org/10.1109/HUST.2014.7

[7] Steven M. Gallo, Joseph P. White, Robert L. DeLeon, Thomas R. Furlani, Helen Ngo, Abani K. Patra, Matthew D. Jones, Jeffrey T. Palmer, Nikolay Simakov, Jeanette M. Sperhac, Martins Innus, Thomas Yearke, and Ryan Rathsam. 2015. Analysis of XDMoD/SUPReMM Data Using Machine Learning Techniques. In *2015 IEEE International Conference on Cluster Computing*. IEEE, 642–649. https://doi.org/10.1109/CLUSTER.2015.114

[8] James H Horne and Sallie L Baliunas. 1986. A prescription for period analysis of unevenly sampled time series. *The Astrophysical Journal* 302 (March 1986), 757–763. https://doi.org/10.1086/164037

[9] iot 2015. IOT FAQ. http://iodoctors.com/faq.html.

[10] H. Jin. 2017. Using MPIProf for performance analysis. http://www.nas.nasa.gov/hecc/support/kb/using-mpiprof-for-performance-analysis_525.html.

[11] Yang Liu, Raghul Gunasekaran, Xiaosong Ma, and Sudharshan S. Vazhkudai. 2014. Automatic Identification of Application I/O Signatures from Noisy Server-side Traces. In *Proceedings of the 12th USENIX Conference on File and Storage Technologies (FAST'14)*. USENIX Association, Berkeley, CA, USA, 213–228. http://dl.acm.org/citation.cfm?id=2591305.2591326

[12] Y. Liu, R. Gunasekaran, X. Ma, and S. S. Vazhkudai. 2016. Server-Side Log Data Analytics for I/O Workload Characterization and Coordination on Large Shared Storage Systems. In *SC16: International Conference for High Performance Computing, Networking, Storage and Analysis*. 819–829. https://doi.org/10.1109/SC.2016.69

[13] N. R. Lomb. 1976. Least-squares frequency analysis of unequally spaced data. *Astrophysics and Space Science* 39, 2 (01 Feb. 1976), 447–462. https://doi.org/10.1007/BF00648343

[14] Huong Luu, Marianne Winslett, William Gropp, Robert Ross, Philip Carns, Kevin Harms, Mr Prabhat, Suren Byna, and Yushu Yao. 2015. A Multiplatform Study of I/O Behavior on Petascale Supercomputers. In *Proceedings of the 24th International Symposium on High-Performance Parallel and Distributed Computing (HPDC '15)*. ACM, New York, NY, USA, 33–44. https://doi.org/10.1145/2749246.2749269

[15] Matthew L Massie, Brent N Chun, and David E Culler. 2004. The ganglia distributed monitoring system: design, implementation, and experience. *Parallel Comput.* 30, 7 (2004), 817 – 840. https://doi.org/10.1016/j.parco.2004.04.001

[16] Dieter an Mey, Scott Biersdorf, Christian Bischof, Kai Diethelm, Dominic Eschweiler, Michael Gerndt, Andreas Knüpfer, Daniel Lorenz, Allen Malony, Wolfgang E. Nagel, Yury Oleynik, Christian Rössel, Pavel Saviankou, Dirk Schmidl, Sameer Shende, Michael Wagner, Bert Wesarg, and Felix Wolf. 2012. Score-P: A Unified Performance Measurement System for Petascale Applications. In *Competence in High Performance Computing 2010*, Christian Bischof, Heinz-Gerd Hegering, Wolfgang E. Nagel, and Gabriel Wittum (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 85–97.

[17] Ethan L. Miller and Randy H. Katz. 1991. Input/Output Behavior of Supercomputing Applications. In *Proceedings of the 1991 ACM/IEEE Conference on Supercomputing (Supercomputing '91)*. ACM, New York, NY, USA, 567–576. https://doi.org/10.1145/125826.126133

[18] N. Nieuwejaar, D. Kotz, A. Purakayastha, C. Sclatter Ellis, and M. L. Best. 1996. File-access characteristics of parallel scientific workloads. *IEEE Transactions on Parallel and Distributed Systems* 7, 10 (Oct. 1996), 1075–1089. https://doi.org/10.1109/71.539739

[19] Jeffrey T. Palmer, Steven M. Gallo, Thomas R. Furlani, Matthew D. Jones, Robert L. DeLeon, Joseph P. White, Nikolay Simakov, Abani K. Patra, Jeanette M. Sperhac, Thomas Yearke, Ryan Rathsam, Martins Innus, Cynthia D. Cornelius, James C. Browne, William L. Barth, and Richard T. Evans. 2015. Open XDMoD: A tool for the comprehensive management of high-performance computing resources. *Computing in Science and Engineering* 17, 4 (2015), 52–62. https://doi.org/10.1109/MCSE.2015.68

[20] B. K. Pasquale and G. C. Polyzos. 1994. Dynamic I/O characterization of I/O intensive scientific applications. In *Proceedings of Supercomputing '94*. 660–669. https://doi.org/10.1109/SUPERC.1994.344330

[21] Emilia Rosti, Giuseppe Serazzi, Evgenia Smirni, and Mark S. Squillante. 2002. Models of Parallel Applications with Large Computation and I/O Requirements. *IEEE Trans. Softw. Eng.* 28, 3 (March 2002), 286–307. https://doi.org/10.1109/32.991321

[22] J. D. Scargle. 1982. Studies in astronomical time series analysis. II - Statistical aspects of spectral analysis of unevenly spaced data. *The Astrophysical Journal* 263 (Dec. 1982), 835–853. https://doi.org/10.1086/160554

[23] Frank Schmuck and Roger Haskin. 2002. GPFS: A Shared-disk File System for Large Computing Clusters. In *Proceedings of the 1st USENIX Conference on File and Storage Technologies (FAST'02)*. USENIX Association, Berkeley, CA, USA, 16–16. http://dl.acm.org/citation.cfm?id=1973333.1973349

[24] Silicon Graphics Inc, Aconex, and Red Hat. 2000. Performance Co-Pilot (PCP). https://pcp.io.

[25] The Astropy Collaboration. 2018. The Astropy Project: Building an inclusive, open-science project and status of the v2.0 core package. *ArXiv e-prints* (Jan. 2018). arXiv:astro-ph.IM/1801.02634

[26] Michail Vlachos, Philip Yu, and Vittorio Castelli. 2005. On Periodicity Detection and Structural Periodic Similarity. In *Proceedings of the 2005 SIAM International Conference on Data Mining*. SIAM, 449–460. https://doi.org/10.1137/1.9781611972757.40 arXiv:http://epubs.siam.org/doi/pdf/10.1137/1.9781611972757.40

[27] Feng Wang, Qin Xin, Bo Hong, Scott A. Brandt, Ethan L. Miller, Darrell D. E. Long, and Tyce T. Mclarty. 2004. File System Workload Analysis For Large Scientific Computing Applications. In *NASA/IEEE Conference on Mass Storage Systems and Technologies (MSST 2004)*. 139–152.