Log-Based Identification, Classification, and Behavior Prediction of HPC Applications

Ryan D. Lewis
Department of Computer Science
Northern Illinois University
rlewis5@niu.edu

Zhengchun Liu
Data Science and Learning Division
Argonne National Laboratory
zhengchun.liu@anl.gov

Rajkumar Kettimuthu
Data Science and Learning Division
Argonne National Laboratory
kettimut@anl.gov

Michael E. Papka
Leadership Computing Facility
Argonne National Laboratory
Department of Computer Science
Northern Illinois University
papka@anl.gov

ABSTRACT

Leadership supercomputers, such as those operated by the Argonne Leadership Computing Facility (ALCF), provide an important avenue for scientific exploration and discovery, enabling simulation, data analysis & visualization, and artificial intelligence at massive scale. As we move into the exascale supercomputing era in 2021 with the advent of Aurora, Frontier, and other exascale machines, it's important that we are able to understand the interactions between the applications being run, and the hardware they run on, to optimize the use of these expensive and high-demand resources.

In previous work, we analyzed a collection of production machine scheduling and performance logs to better understand application behaviors and characteristics. This work further refines our understanding of how scientific users leverage leadership computing resources; we show that system-level hardware performance counters can work as a lightweight, low-overhead alternative to more performance-intensive benchmarking and logging instrumentation for certain data analysis tasks. We also demonstrate a method for predicting application runtimes on leadership computing resources using data gathered from logging sources at submission.

CCS CONCEPTS

• **Computing methodologies** → *Cross-validation*; *Supervised learning by classification*; *Supervised learning by regression*; • **Software and its engineering** → *Operational analysis.*

KEYWORDS

High Performance Computing; Logs data mining; Application Identification; Characterization

HPCSYSPROS'20, November 13, 2020, Atlanta, GA

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/10.1145/1122445.1122456

ACM Reference Format:
Ryan D. Lewis, Zhengchun Liu, Rajkumar Kettimuthu, and Michael E. Papka. 2020. Log-Based Identification, Classification, and Behavior Prediction of HPC Applications. In *HPCSYSPROS'20: HPCSYSPROS SC20 Workshop Proceedings, November 11–13, 2020, Atlanta, GA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

In modern high-performance computing (HPC), especially at leadership computing facilities, a great deal of log creation and collection occurs essentially all the time. Whether it is administrative, scheduling, performance monitoring, debugging, or diagnostic in nature, a single supercomputer at a user facility like the ALCF generates vast quantities of log data every day.

Argonne's recently decommissioned Mira supercomputer, a 10-petaflops Blue Gene/Q machine that went into production in 2013 and ceased operation at the end of 2019, was no exception to that rule. The combination of Mira's approach to isolating jobs to prevent cross-job interference, and a tendency towards repetitious, patterned behavior by users and applications (explored in section 2), mean that Mira's logs provide a perfect opportunity to perform correlative data analysis and visualization, allowing numerous insights into HPC applications, user demands, and resource usage characteristics on leadership machines. We also believe that this characterization will be useful for optimizing facility management, improving energy efficiency, and optimizing scheduling policy on current and future Leadership computing resources.

Our analysis efforts primarily targeted 3 different areas: application identification and identity verification, application runtime prediction, and application resource-intensiveness classification. Insights in all these areas are of potential interest to Leadership Facility staff or users.

The primary contributions of this paper are the following:

- We evaluated whether hardware performance counters could act as a viable alternative representation for higher-overhead instrumentation in HPC application identification and characterization.
- We evaluated the use of log-based analysis for prediction of application runtimes with the information available at

submission, and the potential effect of runtime prediction for scheduling.
- We evaluated the effectiveness of using hardware performance counters for minimal-overhead application classification of resource-intensiveness.

The rest of this paper is organized as follows: we introduce necessary *Background* knowledge in section 2, including a discussion of the supercomputer we analyzed and the data logging sources we included; we then explore the analysis *Methods and Results* in section 3; finally, we include a *Discussion* of our findings in §3.4. In §4 we review related work, and in §5 we summarize our conclusions and discuss future work.

2 BACKGROUND

2.1 The Mira Supercomputer

Our analysis primarily used logs collected from jobs run on Argonne's Mira supercomputer, a 10 petaflops—i.e., 10^{16} or ten thousand trillion floating point operations per second—IBM Blue Gene/Q system. This machine was in production from its launch in 2013 until it was decommissioned at the end of 2019 to make way for Aurora, its exascale successor, set to launch in 2021.

Mira consisted of 48 racks containing a total of 49, 152 computing nodes, each equipped with a 16-core, 1.6-GHz IBM PowerPC A2 processor for a total of 786, 432 cores, interconnected with a proprietary 5D torus network. Each node was equipped with 16 GB of RAM, with the machine totaling an impressive 768 TB of RAM. For storage, Mira was equipped with two General Parallel File Systems (GPFS) with 20 and 7 petabytes of capacity, respectively.

2.2 Datasets and Logs

Our analysis relied on the 5 datasets described below, drawn from Mira logs collected at the ALCF. Anonymized and cleaned, publicly-available versions of all but the Tracklib dataset are available online[1].

2.2.1 Autoperf. Autoperf [4] is a library for automatically collecting hardware performance counter and MPI usage statistics. For every MPI process spawned by a job, Autoperf records the number of MPI routine calls made, the time spent in each routine, and the total bytes communicated by each routine. This information, along with other performance data such as basic hardware performance counters, is collected over the course of the job and then saved to log files on job completion. In order to minimize overhead and save storage space, Autoperf records only the maximum, minimum, and average values—as well as the value from MPI rank 0—for each counter across all MPI processes of a given execution. It is enabled by default on Mira, but can be disabled by the user. Autoperf logs may be absent for a number of other reasons, including applications which do not use MPI or which fail to call `MPI_Finalize`, or which are built or linked with a conflicting/unsupported compiler or profiling library. As a result, the Autoperf dataset used in this paper contains records for 377, 969 tasks run on Mira from 2016–2019: about 22% of all tasks run on Mira during that period.

[1] https://reports.alcf.anl.gov/data/

2.2.2 Darshan. Darshan [2] is another performance logging tool used extensively at the ALCF. Where Autoperf collects information on MPI, Darshan provides a granular look at the input/output (I/O) instrumentation, behavior, and performance of production applications. This includes collecting information on the number of files opened, how much time is spent performing various types of file operations, the number of bytes accessed, and so on. Many of these counters are broken into subcategories: MPI versus POSIX operations; read, write, and metadata operations, etc. Like Autoperf, Darshan was enabled by default on all production applications run on Mira, but could be disabled by the user. This paper uses a dataset containing records for 385.023 tasks executed on Mira from 2016–2019: about 23% of all tasks run on Mira during that period.

2.2.3 Control System Logs. Whenever a job is scheduled and run on an ALCF supercomputer such as Mira, it has the ability to spawn multiple applications—working in series or in parallel—to perform work on the machine. Each such application execution spawned is managed by the control system as a separate task and given its own unique control system ID, and the control system records its executable ID, start time, end time, and other characteristics. This information is dutifully logged to the control system logs, and for this work we used a dataset of all 2, 592, 361 tasks executed on Mira from 2016–2019. Since both Darshan and Autoperf record information on a task-level, analysis for both can easily integrate information from the control system dataset.

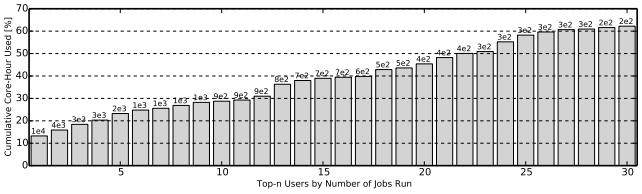
2.2.4 Cobalt. The Mira supercomputer used the component-based lightweight toolkit (Cobalt) as its scheduling and queuing system. This tool allowed users to send job submissions requesting time on Mira, and for each job, Cobalt recorded various metadata. This included a unique job ID; submission, execution, and termination timestamps; the number of compute nodes requested and actually used; the location of allocated nodes; the amount of time requested by the user for their job to run; and more. Notably, this information is logged at a job-level, rather than at a task-level. Thus, some additional work must be done to use Cobalt log data with the control system, Autoperf, and Darshan logs presented above. We used a dataset containing records for all 300, 023 jobs run on Mira from 2015–2019.

2.2.5 TrackLib. Many of the applications that ran on Mira relied on linking to external libraries to provide necessary functionality, profiling, debugging, or other benefits. Tracklib [5] is a tool linked by default with all executables run on Mira which logs the set of libraries linked to the applications run. This information is collected at the job level, and includes the library name and version, the start timestamp of the job, Cobalt job ID, and user/project information. The dataset used in this work includes information on jobs submitted from 2015–2019, about 158, 294 jobs in total, for a coverage of about 52.76%.

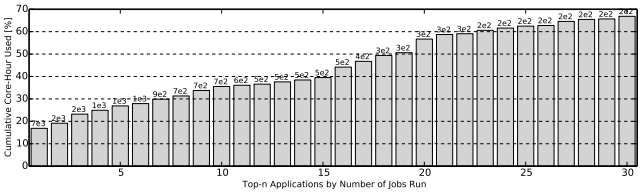
2.3 Homogeneity of User and Application Behavior

Early user-level(Figure 1a) and job-level(Figure 1b) analysis for this work revealed that both users and applications utilizing Mira exhibit highly repetitive patterns. For example, the top 30 applications consumed nearly 70% of the machine time (e.g., core-hours), and

the 30 most-active users consumed about 65% of the machine time. This patterned behavior allows for the categorization of jobs in a small number of groups at the time of submission or when the job is completed. As part of this work, we set out to find fingerprint representations for jobs using information at different stages, which may be usable for application verification to detect unapproved applications, and even predicting runtimes when a job is submitted, for smarter scheduling.



(a) Cumulative machine time used by top 30 users.



(b) Cumulative machine time used by top 30 applications.

Figure 1: Machine time consumption of top applications and users. The number on the top of each bar denotes the number of jobs in the statistic.

3 METHODS AND RESULTS

In this work, we explored the use of log-based analysis for three different cases, broken into §3.1 on *Post-Runtime Identification and Verification*, §3.2 for *Pre-Runtime Elapsed Time Prediction*, and §3.3 regarding *Classification*.

3.1 Application Identification

We developed a task representation approach for Mira using 7 engineered features in [12], designed to capture "file I/O, computation, communication, and main memory access." These 7 features were able to achieve a testing accuracy of about 99.5% when identifying applications; however, all 7 features also rely on logs, and thus have limitations such as: (1) logging introduces overhead to the application, (2) logging can be disabled by users, and (3) applications must be compiled and linked using a specifically designed compiler. Here, we attempt to replicate these results using 12 lightweight hardware performance counters available on Blue Gene Q systems [15], which require much less overhead to collect. Since these hardware performance counters are collected by Autoperf, we can conduct analysis on the same set of jobs used in [12] to directly compare the results.

We used the following hardware performance counters in our analysis, collected on the "average" routine in Autoperf:

- `PEVT_LSU_COMMIT_LD_MISSES`: Counts the number of load commands which completed but missed the L1 cache.
- `PEVT_LSU_COMMIT_CACHEABLE_LDS`: Counts the number of load commands which hit the L1 cache.
- `PEVT_L1P_BAS_MISS`: Counts the number of times the L1P cache was missed.
- `PEVT_INST_XU_ALL`: Counts the total number of instructions run on the execution unit.
- `PEVT_INST_QFPU_ALL`: Counts the total number of instructions run on the auxiliary execution unit.
- `PEVT_INST_QFPU_FPGRP1`: Counts the floating point operations.
- `PEVT_L2_HITS`: Counts the total number of L2 cache hits.
- `PEVT_L2_MISSES`: Counts the total number of L2 cache misses.
- `PEVT_L2_FETCH_LINE`: Counts the number of L2 fetches.
- `PEVT_L2_STORE_LINE`: Counts the number of L2 stores.
- `PEVT_NW_USER_PP_SENT`: Counts the number of 32-byte user point-to-point chunks sent, including packets originating or passing through the current node.
- `PEVT_NW_USER_PP_RECV`: Counts the number of 32-byte user point-to-point chunks received on the current node.

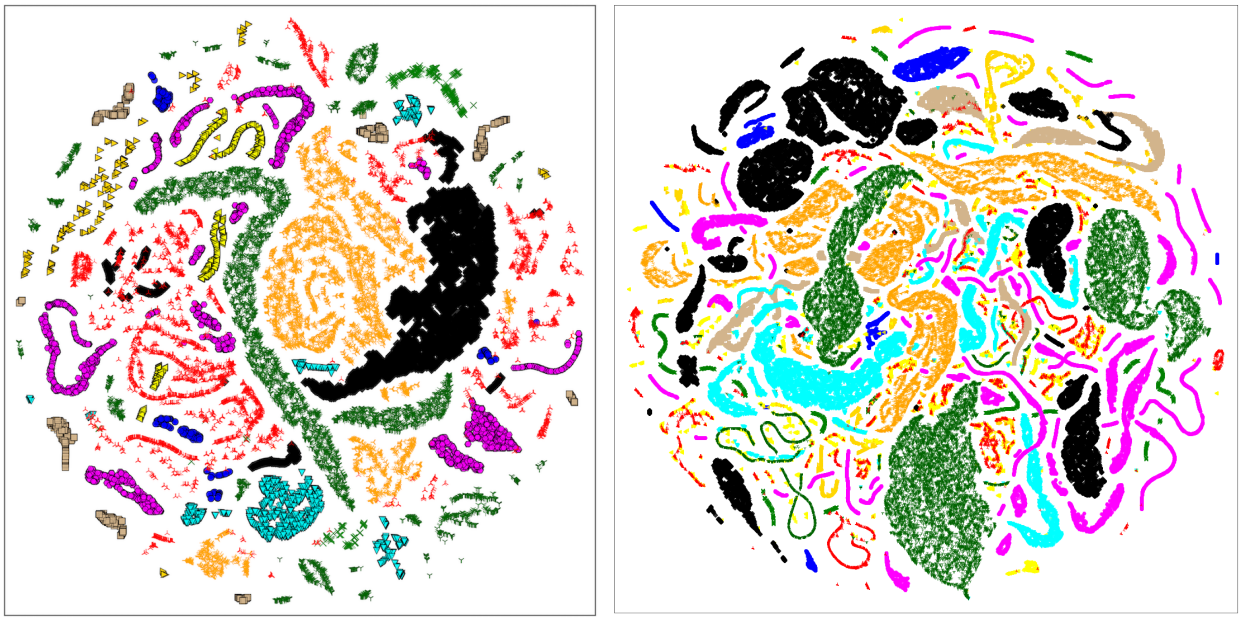
While there are numerous (200+) other hardware performance counters collected by the system, we choose to use these for our analysis as Autoperf provides a large extant dataset suitable for our purposes. In order to directly compare our results to Liu et al.'s [12], we limit our analysis to jobs run in 2018.

We first create a T-distributed stochastic neighbor embedding (t-SNE) [14] in order to visualize this 12-dimensional data in a two-dimensional format. We use a perplexity of 30 and 1000 iterations, color-encoding the top 20 core-hour consuming executables and grouping the rest as an "other" category, resulting in the figure shown in Figure 2b on the right, with t-SNE representation from [12] included in Figure 2a on the left for comparison. While the clustering is clearly different in the two figures, this is to be expected because t-SNE has a non-convex objective function. For our purposes, it is encouraging that the t-SNE exhibits clustering behavior which aligns with the color-encoded executable names.

Next, we built and trained an extreme gradient boosting machine-learning model, attempting to accurately determine an application's executable name based solely on the hardware performance counters collected during a job. XGBoost (XGB) [3] is a powerful high-performance gradient boosting framework, which we used both for its own merits and to compare 1-to-1 with Liu et al.'s 7-feature model.

We again used data from 2018 (127, 585 jobs in total) labeling them with either their executable name for the top 20 core-hour consuming executables, or 'other'. We normalized the values by dividing each hardware performance counter by the elapsed time of the application, so that all variables would have the same scale across different applications. This mitigates the impact of a job's runtime on the scale of the hardware performance counter values. We also dropped logs that had invalid or missing data, but none were included in this particular subset of the dataset. Finally, we performed a 70/30 train-test split on our dataset.

After performing a 5-fold cross-validation on the training dataset with a max-depth of 10, we found a testing accuracy of 99.1%, with a precision score of 96.7% and a recall score of 96.3%. While very slightly (~ 0.4%) less accurate than the engineered features from



(a) t-SNE generated using engineered features [12]

(b) t-SNE generated using hardware performance counters.

Figure 2: Two-dimensional t-SNE embedding of task representation. Dots (tasks) with the same color share the same executable name. Note that colors do not match across the 2 subfigures.

Autoperf used by Liu et al. [12], this suggests that hardware performance counters can be used to accurately distinguish between different executables.

3.2 Run time Prediction

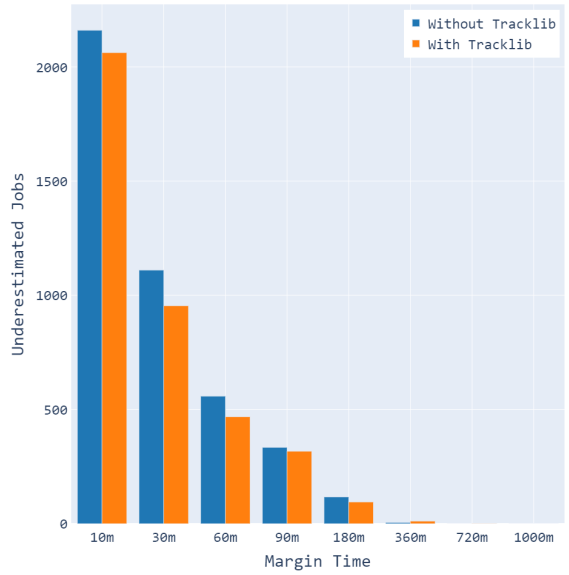
An estimated runtime is provided by the user when the job is submitted to the scheduler. This estimation is one of the key decision factors used for resource allocation. More accurate estimates can enable more efficient resource allocation and less job slowdown. However, this user estimation is known to be unreliable and largely overestimated [1]. There is much research into more accurate runtime prediction using various information available at the time of submission [7, 16, 18]. However, the fact that the information available at the time of submission is very limited makes the prediction task very difficult. Resource allocation based on an underestimated prediction risks causing a job to be killed by the resource manager before it is completed. As presented previously in §2.3, applications run at leadership computing facilities are highly repetitive, making it easier to predict resource requirements at the time the job is run.

We evaluated the effectiveness of runtime prediction, i.e., using information available only before a queued job is run to predict how long the application will run. This has potential utility for job scheduling applications and application designers. To accomplish this task, we prepared our features and trained an XGBoost regression model as follows. We began by encoding all of the categorical features (e.g., username, project, executable name, hour of the day, etc.) available to us at the time of job submission in the Cobalt dataset using a one-of-K scheme (sometimes referred to as one-hot encoding). Using data from 2016, we performed an 80/20 split, using the first 9.6 months of data for model training and the remaining 2.4 months to evaluate. We filtered out jobs that exited with non-zero exit codes, i.e., failed jobs, giving a total dataset size of 39,222 jobs.

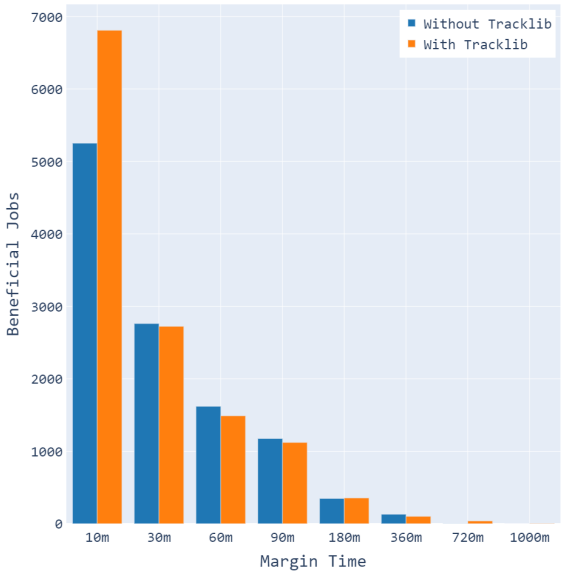
Because only a limited amount of information is available when a job is queued, accurate prediction is challenging. One of the key problems with using predicted runtime (versus a user's wall time estimation) for scheduling is that if the prediction is underestimated, the job will be killed by the scheduler, wasting core-hours and the user's machine time. This is obviously highly undesirable.

One way to mitigate this problem is to add a fixed grace period to the prediction, which reduces the possibility of underestimation, but can increase the amount of time that is overestimated, leading to inefficient scheduling. In Table 1, we evaluate the trade-off of adding a fixed time and the number of jobs that may be potentially started earlier using our predicted runtime to replace the user provided wall time. For each margin time (adding a certain number of minutes to our predicted runtime), we show how many jobs were underestimated (i.e. predicted time + margin time > actual runtime) and how many jobs benefited from the prediction (i.e., predicted time + margin time < wall time).

Given the limited amount of information provided by the job scheduler, we next incorporated the Tracklib [5] dataset to determine whether its inclusion could measurably improve our model's accuracy, as it can serve as a more reliable identifier for applications than the executable name. We used a similar methodology to the one used above, this time adding a one-hot encoding of each library included in the Tracklib dataset, joined with the Cobalt dataset. Results of our model with the Tracklib dataset used are shown in Table 2.



(a) Margin Times vs. Underestimated Jobs.



(b) Margin Times vs. Beneficial Jobs.

Figure 3: Margin Times vs. Underestimated Jobs (left) and Beneficial Jobs (right) for Runtime Prediction, with and without Tracklib.

Table 1: Practical Runtime Prediction error analysis, testing sample size is 7,844 jobs.

Margin Time (min)	Underestimated Jobs	Beneficial Jobs
10	2,160	5,252
30	1,110	2,762
60	558	1,620
90	334	1,175
180	117	348
360	5	130
720	0	3

Table 2: Practical Runtime Prediction error analysis with Tracklib, testing sample size is 7,844 jobs.

Margin time (min)	Underestimated Jobs	Beneficial Jobs
10	2,062	6,810
30	954	2,724
60	468	1,489
90	317	1,121
180	95	355
360	11	101
720	1	38
1000	0	6

Our results show that adding features from Tracklib does measurably reduce the number of jobs underestimated for most margin times, but at the cost of slight reductions in the number of jobs benefiting from the predicted runtime. For a margin time of 30 minutes, for instance, we see that without Tracklib 1,110 jobs are killed prematurely (14.1%), while only 954 were killed prematurely (12.2%) with Tracklib used. With the same margin time, the model trained without Tracklib benefited 2,762 jobs (35.2%), while with Tracklib it only benefited 2,724 (34.7%).

3.3 Resource-intensiveness Classification

Finally, we investigated how effective hardware performance counters are in classifying application runs based on different resource consumption patterns. To accomplish this, we labeled our samples using fractions of runtime spent on file I/O, computation, and MPI communication; calculated using counters from Darshan and Autoperf logs based on our prior work [12].

We defined an application as either file I/O-, computation-, or MPI communication-intensive by looking at the ratio of time spent by an application on each category relative to that application's total runtime. For our investigation, we used the same hardware performance counters described in subsection 3.1, taken from the Autoperf dataset, as the training features for an XGBoost classification model. We synthesized a dataset of jobs from 2018 for which both Autoperf and Darshan records were available, some 9,891 jobs in total. Of these, 7,683 are computation intensive, 1,625 are MPI intensive, and 190 are file I/O intensive, with 393 being not intensive in any of the 3 categories.

Using an 80/20 train-test split, we are able to achieve a testing accuracy of 96.5% in correctly identifying the resource-intensiveness of a given job using hardware performance counter information, with a recall score of 88.7% and a precision score of 92.0%. A graph of feature importance for our model is shown in Figure 4.

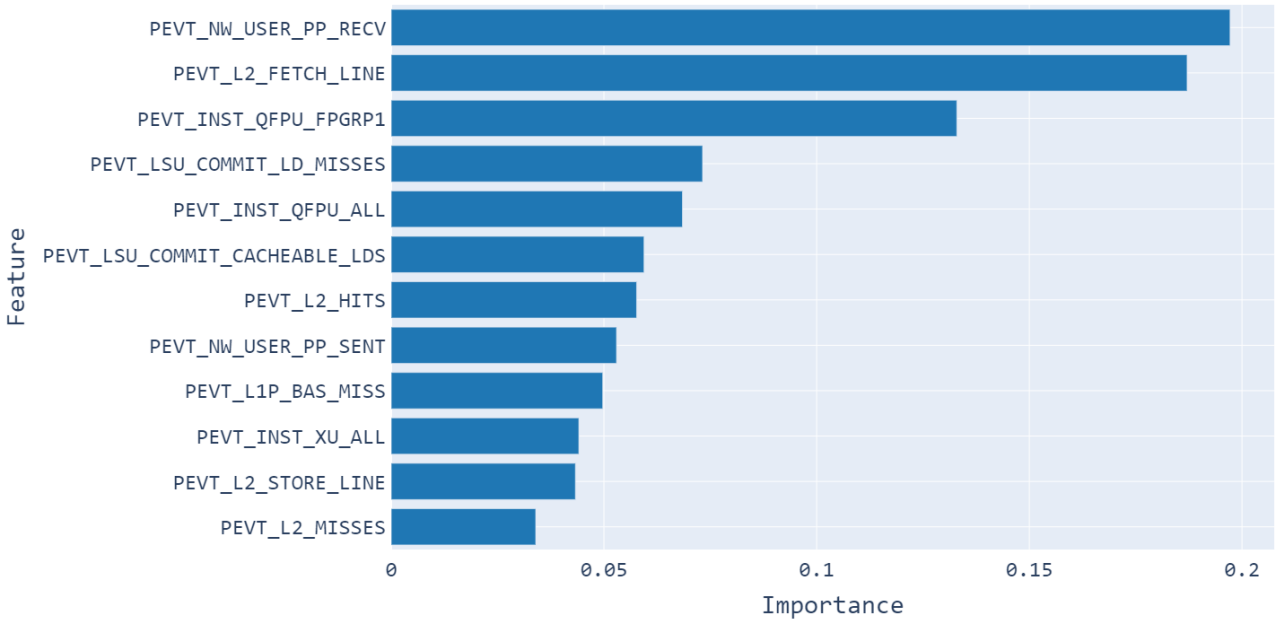


Figure 4: Importance of features used in resource-intensiveness classification.

3.4 Results Discussion

In considering our results, we came away with two important takeaways. First, that hardware performance counters may be a suitable substitute for heavier performance and instrumentation logging, without the measurable overhead instrumentation logging tools require. Hardware performance counters effectively require no additional time or resource usage, and, as we have shown, can serve as a substitute for full-on logging in applications such as verification and resource-intensiveness classification, with only slight losses in accuracy.

Second, that application run time is difficult to predict. This is largely due to the lack of identifying features available before the job has run. While incorporating additional features, such as those provided by Tracklib, can improve our model's ability to predict, and pulling from more datasets may further improve this performance, perfect (or even acceptable) predictive accuracy seems untenable with this current approach.

4 RELATED WORK

Leadership supercomputers rely on a number of advanced computing technologies, from high-performance storage systems to high-speed networks and dedicated data transfer nodes. Many of these technologies produce logs, and so quite a bit of log analysis and research has been done already in leadership computing contexts to characterize performance and user behavior [4, 8–11, 13, 17]. An adjacent example to this work is Chunduri et al's. [4] characterizing of MPI usage on Mira, using Autoperf to gain insights for the benefit of MPI users, network hardware developers, and the facilities' teams. Other examples include Lim et al.'s [8] comprehensive insights on user behavior from multiple science domains through metadata analysis of a petascale file system at a leadership computing facility. Gainaru et al.'s [6] analysis of the effects of interference on application file I/O bandwidth found that a significant percentage of the computing capacity of large-scale platforms is wasted because of interference incurred by multiple applications that access a shared parallel file system concurrently [6].

Previous work exploring application runtime prediction includes Tsafrir et al.'s [18] efforts to use system-generated runtime predictions for backfilling, Guo et al.'s [7] efforts to predict and warn users of runtime underestimation, and Naghshnejad and Singhal's [16] work on predicting HPC application runtime in cloud contexts.

5 CONCLUSIONS AND FUTURE WORK

In this work, we built on previous efforts to characterize HPC applications based on performance log mining and analysis; evaluated the utility of integrating the Tracklib log source for run-time prediction; and determined the effectiveness of using hardware performance counters for minimal-overhead application characterization, classification, and identification.

Future work includes investigating further improvements or alternative approaches to runtime estimation, as well as exploring additional applications of log-based and hardware performance counter-based analysis and insights. One application of particular interest to the authors is task representation/classification of cloud computing pipelines, where a pipeline might be optimized to take advantage of different cloud hardware based on analysis of previous runs.

ACKNOWLEDGMENTS

This material is based upon work supported by the U.S. Department of Energy, Office of Science, under contract number DE-AC02-06CH11357. The datasets used in this research were generated from resources of the Argonne Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC02-06CH11357. We thank Doug Waldron and Sudheer Chunduri, both from the Argonne Leadership Computing Facility, for providing dataset descriptions and information. Finally, we thank Laura Wolf of the Argonne Leadership Computing Facility for reviewing the work prior to submission.

REFERENCES

[1] Cynthia Bailey Lee, Yael Schwartzman, Jennifer Hardy, and Allan Snavely. 2005. Are User Runtime Estimates Inherently Inaccurate?. In *Job Scheduling Strategies for Parallel Processing*, Dror G. Feitelson, Larry Rudolph, and Uwe Schwiegelshohn (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 253–263.

[2] Philip Carns. 2014. Darshan. In *High Performance Parallel I/O*. Chapman and Hall/CRC, 351–358.

[3] Tianqi Chen and Carlos Guestrin. 2016. Xgboost: A scalable tree boosting system. *arXiv preprint arXiv:1603.02754* (2016).

[4] Sudheer Chunduri, Scott Parker, Pavan Balaji, Kevin Harms, and Kalyan Kumaran. 2018. Characterization of MPI Usage on a Production Supercomputer. In *International Conference for High Performance Computing, Networking, Storage, and Analysis*. IEEE Press, Piscataway, NJ, USA, Article 30, 15 pages. http://dl.acm.org/citation.cfm?id=3291656.3291696

[5] Sudheer Chunduri, Scott Parker, Pavan Balaji, Kevin Harms, and Kalyan Kumaran. 2018. Characterization of MPI usage on a production supercomputer. In *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 386–400.

[6] A. Gainaru, G. Aupy, A. Benoit, F. Cappello, Y. Robert, and M. Snir. 2015. Scheduling the I/O of HPC Applications Under Congestion. In *2015 IEEE International Parallel and Distributed Processing Symposium*. 1013–1022. https://doi.org/10.1109/IPDPS.2015.116

[7] Jian Guo, Akihiro Nomura, Ryan Barton, Haoyu Zhang, and Satoshi Matsuoka. 2018. Machine learning predictions for underestimation of job runtime on HPC system. In *Asian Conference on Supercomputing Frontiers*. Springer, Cham, 179–198.

[8] Seung-Hwan Lim, Hyogi Sim, Raghul Gunasekaran, and Sudharshan S. Vazhkudai. 2017. Scientific User Behavior and Data-sharing Trends in a Petascale File System. In *International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM, New York, NY, USA, Article 46, 12 pages. https://doi.org/10.1145/3126908.3126924

[9] Yuanlai Liu, Zhengchun Liu, Rajkumar Kettimuthu, Nageswara Rao, Zizhong Chen, and Ian Foster. 2019. Data Transfer between Scientific Facilities – Bottleneck Analysis, Insights and Optimizations. In *2019 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*. 122–131. https://doi.org/10.1109/CCGRID.2019.00023

[10] Zhengchun Liu, Rajkumar Kettimuthu, Ian Foster, and Yuanlai Liu. 2018. A comprehensive study of wide area data movement at a scientific computing facility. In *38th IEEE International Conference on Distributed Computing Systems*. IEEE, 8.

[11] Zhengchun Liu, Rajkumar Kettimuthu, Ian Foster, and Nageswara S. V. Rao. 2018. Cross-geography Scientific Data Transferring Trends and Behavior. In *27th International Symposium on High-Performance Parallel and Distributed Computing*. ACM, New York, NY, USA, 267–278. https://doi.org/10.1145/3208040.3208053

[12] Zhengchun Liu, Ryan Lewis, Rajkumar Kettimuthu, Kevin Harms, Philip Carns, Nageswara Rao, Ian Foster, and Michael E. Papka. 2020. Characterization and Identification of HPC Applications at Leadership Computing Facility. In *Proceedings of the 34th ACM International Conference on Supercomputing (ICS '20)*. Association for Computing Machinery, New York, NY, USA, Article 29, 12 pages. https://doi.org/10.1145/3392717.3392774

[13] Glenn K. Lockwood, Shane Snyder, Teng Wang, Suren Byna, Philip Carns, and Nicholas J. Wright. 2018. A Year in the Life of a Parallel File System. In *International Conference for High Performance Computing, Networking, Storage, and Analysis*. IEEE Press, Piscataway, NJ, USA, Article 74, 13 pages. http://dl.acm.org/citation.cfm?id=3291656.3291755

[14] Laurens van der Maaten and Geoffrey Hinton. 2008. Visualizing data using t-SNE. *Journal of machine learning research* 9, Nov (2008), 2579–2605.

[15] Heike McCraw, Dan Terpstra, Jack Dongarra, Kris Davis, and Roy Musselman. 2013. Beyond the CPU: hardware performance counter monitoring on blue gene/q. In *International Supercomputing Conference*. Springer, 213–225.

[16] M. Naghshnejad and M. Singhal. 2018. Adaptive Online Runtime Prediction to Improve HPC Applications Latency in Cloud. In *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*. 762–769.

[17] Tirthak Patel, Suren Byna, Glenn K. Lockwood, and Devesh Tiwari. 2019. Revisiting I/O Behavior in Large-scale Storage Systems: The Expected and the Unexpected. In *International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM, New York, NY, USA, Article 65, 13 pages. https://doi.org/10.1145/3295500.3356183

[18] Dan Tsafrir, Yoav Etsion, and Dror G Feitelson. 2007. Backfilling using system-generated predictions rather than user runtime estimates. *IEEE Transactions on Parallel and Distributed Systems* 18, 6 (2007), 789–803.