Taxonomist: Application Detection through Rich Monitoring Data

Emre Ates[1] (✉), Ozan Tuncer[1], Ata Turk[1], Vitus J. Leung[2], Jim Brandt[2], Manuel Egele[1], and Ayse K. Coskun[1]

[1] Boston University, Boston MA 02215, USA
`{ates,otuncer,ataturk,megele,acoskun}@bu.edu`
[2] Sandia National Laboratories, Albuquerque NM 87185, USA
`{vjleung,brandt}@sandia.gov`

Abstract. Modern supercomputers are shared among thousands of users running a variety of applications. Knowing which applications are running in the system can bring substantial benefits: knowledge of applications that intensively use shared resources can aid scheduling; unwanted applications such as cryptocurrency mining or password cracking can be blocked; system architects can make design decisions based on system usage. However, identifying applications on supercomputers is challenging because applications are executed using esoteric scripts along with binaries that are compiled and named by users.
This paper introduces a novel technique to identify applications running on supercomputers. Our technique, Taxonomist, is based on the empirical evidence that applications have different and characteristic resource utilization patterns. Taxonomist uses machine learning to classify known applications and also detect unknown applications. We test our technique with a variety of benchmarks and cryptocurrency miners, and also with applications that users of a production supercomputer ran during a 6 month period. We show that our technique achieves nearly perfect classification for this challenging data set.

Keywords: Supercomputing · HPC · Application Detection · Monitoring · Security · Cryptocurrency

1 Introduction

Resource utilization and efficiency of supercomputers are top concerns for both system operators and users. It is typical to use figures of merit such as occupation of compute nodes or total CPU usage to assess utilization and efficiency; however, these metrics do not measure if the compute capacity is used meaningfully.

In fact, fraud, waste, and abuse of resources have been major concerns in high performance computing (HPC) [1]. Wasted resources in supercomputing stem from a variety of sources such as application hangs due to software and hardware faults, contention in shared resources (such as high speed networks, shared parallel file systems or memory), and fraudulent use (e.g., bitcoin mining, password cracking). Bitcoin mining in supercomputing environments has recently been gaining media attention [20,23]. Knowing which applications are running on the system is a strong aid in addressing fraud, waste, and abuse problems.

Knowledge of applications running on the system can also be used for various system-level optimizations. Bhatele et al. have shown that network-intensive applications can slow down other applications significantly [7]. Similarly, Auweter et al. presented a scheduling method that leverages application-specific energy consumption models to reduce overall power consumption [5]. Knowing the most common applications and their characteristics is also useful to system architects who make design decisions, or to the supercomputer procurers who can make better funding and procurement decisions based on knowledge of typical application requirements.

Typically, supercomputer operators and system management software running on these large computers have no knowledge of which applications are executing in the supercomputer at a given time. A supercomputer is shared by many users and runs hundreds to thousands of applications concurrently per day [19]. These applications are compiled by users using different compiler settings, which result in vastly different executables even if compiled from the same source. It has been shown that static analysis of the binaries is not enough to detect the same application compiled with different compilers or flags [13]. Furthermore, users tend to use non-descriptive names for the binaries and scripts used in their job submission (e.g., `submit128.sh`, `a.out`, `app_runner.sh`). Therefore, naive methods for detecting applications such as looking at the names of the processes and scripts are not useful.

To address these challenges, we present *Taxonomist*, an automated technique for identifying applications running in supercomputers. To identify applications, Taxonomist leverages *monitoring data* that is periodically collected at runtime from a supercomputer's compute nodes. Monitoring data includes detailed resource usage information (e.g., CPU utilization, network events, etc.), and is typically used for application tuning [2], gaining information on system usage to aid procurement [12], or for anomaly detection [26]. Each application has (often non-obvious) resource utilization patterns that can be observed in the monitored data. Taxonomist uses machine learning techniques to learn these patterns in the data. Taxonomist can then identify known applications, even when they are running with new input configurations, and also new (unknown) applications. Specifically, our contributions in this paper are as follows:

– We present Taxonomist: a novel technique that uses machine learning to identify known and unknown applications running in a supercomputer based on readily available system monitoring data (§ 4). Taxonomist is able to detect applications that are new to the system, as well as previously unseen input configurations of known applications.
– We demonstrate the effectiveness of Taxonomist on a production supercomputer using over 50,000 production HPC application runs collected over 6 months of cluster usage, a wide selection of benchmarks, and cryptocurrency miners (§ 5). We report greater than 95% *F-score* with this data set (§ 6).

2 Related Work

Several prior approaches have explored identifying applications. Peisert has identified application detection as a problem in supercomputers [21]. He focused on using MPI calls through Integrated Performance Monitoring (IPM) [24] to identify application communication patterns. Further work by Whalen et al. refined the method to classify applications based on their communication graphs [28], and DeMasi et al. used system utilization data collected by IPM to identify applications [11]. These works are based on IPM, which is a tool that monitors the MPI calls in HPC applications. IPM needs to be linked with the applications and introduces up to 5% performance overhead [11].

Combs et al. have studied the applicability of using power signatures to identify applications [8]. As Combs et al. observed, power traces from different servers are not consistently comparable, so such a method is not scalable for large-scale systems. Our evaluation confirms that using only power signatures is insufficient to identify a diverse set of applications in large-scale systems.

Monitoring data has traditionally been used for analyses other than application detection. One of the earlier examples of data analysis in supercomputers was presented by Florez et al., who monitored system calls and library function calls for anomaly detection in applications [14]. Similarly, Tuncer et al. used monitoring data to detect node-level anomalies [26]. Agelastos et al. leveraged monitoring data for troubleshooting and application optimization in a 1200-node supercomputer [3].

In contrast to related work, Taxonomist uses a monitoring system with negligible overhead [2] that is capable of monitoring every application regardless of MPI use, and does not need to be linked with the applications. Taxonomist can be trained with a selection of applications of interest, and can reliably distinguish these applications from the remaining applications. Our method can also detect unknown applications it has not been trained with, which is very important for practical real-world scenarios.

Another line of work aims at blocking unwanted applications. One way to block cryptocurrency mining in supercomputers is to prevent miners from getting the most recent blockchain additions using firewalls [22]. However, many unwanted applications such as password crackers do not need to be connected to the Internet. Furthermore, firewalls may result in packet losses, and it has been shown that even very small packet loss is unacceptable for scientific computing because of the high bandwidth requirements [10]. Another approach to prevent waste might be to whitelist only applications compiled by the system administrators. However, availability is considered to be an important aspect of HPC systems, and limiting the users to use only specific applications would harm the user experience and limit the flexibility and usability of the systems. Therefore, knowledge of the applications running on the system can be a very important aid in blocking unwanted applications.

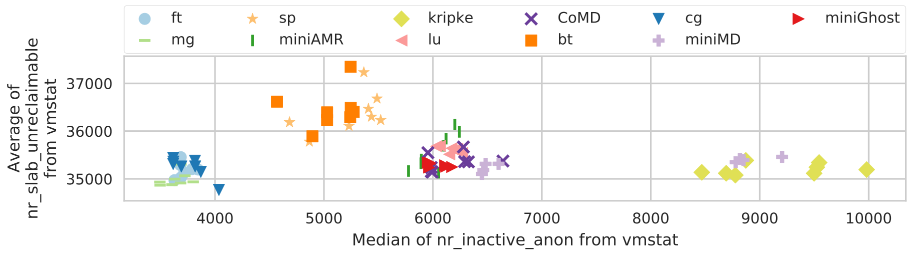


Fig. 1: Two example metrics from `/proc/vmstat` for 11 applications with two different input configurations, where each application is running on 4 nodes. These two metrics can be used to distinguish among some applications, but cannot be used to reliably detect each of the 11 applications.

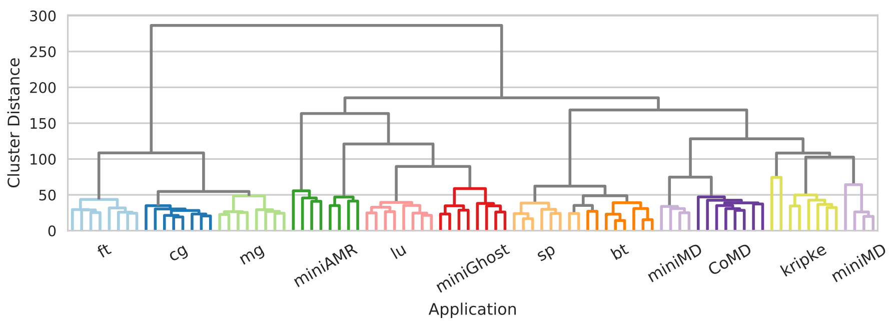


Fig. 2: Clustering of 11 different applications, where each application is running on 4 nodes with two different input configurations. We manually assign different colors to represent different applications.

3 Motivation

Taxonomist uses monitoring data to identify applications. Modern monitoring systems are able to continuously collect hundreds of metrics per second from every compute node in an HPC system [2]. It is infeasible to manually inspect this data and identify applications relying on rules of thumb and expert knowledge; therefore, we design an automated approach to systematically discover the differences between the applications.

Figure 1 shows two example metrics for a set of 11 applications we run on a supercomputer (see § 5 for details on experimental setup). The x-axis shows the median of `nr_inactive_anon`, which represents the number of anonymous memory pages that are inactive, and the y-axis shows the mean of `nr_slab_unreclaimable`, which is the number of pages in the slab memory that cannot be reclaimed. As seen in the figure, applications have different resource usage characteristics. However, these two metrics are not sufficient to distinguish between all applications.

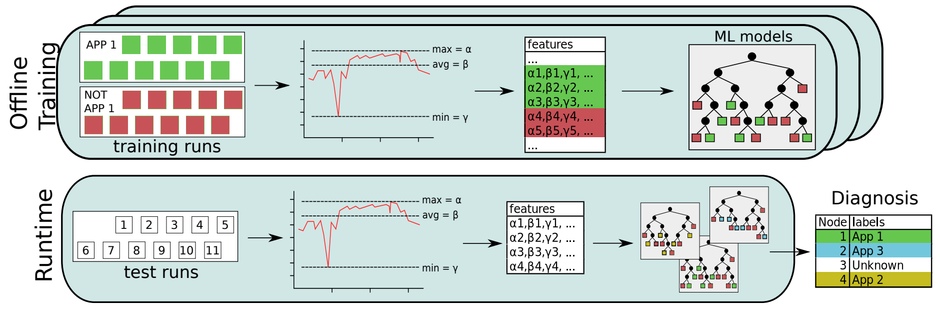


Fig. 3: Overview of Taxonomist.

It is rather challenging to determine the best metrics to distinguish among a large set of applications using intuition or simple methods.

Figure 2 demonstrates clustering of the same 11 applications using all 721 metrics we collect (see § 4.1 for details of the metrics). To construct this figure, we extract statistical features such as percentiles and standard deviation from the collected data (see § 4.2), and cluster the statistics corresponding to the compute nodes. For clustering, we use Ward's method and standardized Euclidean distance (our implementation uses Python `scipy.cluster.hierarchy.linkage`). The results indicate that nodes running the same application are close to each other in the feature space, but the clustering is not perfect (e.g., miniMD is clustered incorrectly).

Manually finding which metrics are important to distinguish each application among hundreds of monitored metrics requires extensive knowledge on the metrics and applications. With supervised learning, the most relevant features can be automatically selected, and applications can be reliably identified. Thus, Taxonomist uses supervised learning techniques.

4 Taxonomist: A Technique for Identifying Applications

Taxonomist, outlined in Fig. 3, is a technique for identifying applications in large-scale systems using monitoring data collected from the machine. The monitoring data is collected from every compute node in a timeseries format. We then generate statistical features that reduce our storage and computation overhead, while enabling us to retain meaningful information in the timeseries. Finally, we train a classifier for each application to separate that application from the rest of the applications using labeled historical data. At runtime, Taxonomist analyzes monitoring data and labels each node's application according to the predictions from the classifiers. We also mark applications as *unknown*, based on the confidence of each classifier.

4.1 Monitoring

The first step of our technique is data collection. Typically some form of monitoring is in place in supercomputers. These systems collect numeric information about the usage of the network, memory, CPUs and other subsystems.

We monitor individual nodes and consider data from all nodes that are running a specific application separately. This enables us to recognize a known application that possibly runs on a different number of nodes than the number of nodes in that application's training runs.

4.2 Statistical Feature Extraction

After collecting monitoring data, Taxonomist removes a segment (40 seconds in our implementation) from each end of the timeseries to account for the transient initialization and finalization phases from the applications. We have observed 40 seconds to be sufficient for all applications in this study; however, this duration is application dependent. We also remove any constant metrics and convert metrics that represent counter values to their deltas.

We generate statistics from the timeseries data gathered from the compute nodes. The statistics used are the minimum, maximum, mean, standard deviation, skew, kurtosis and the 5^{th}, 25^{th}, 50^{th}, 75^{th} and 95^{th} percentiles. Each metric's timeseries is distilled into these 11 features. These statistics have been shown to be useful in analyzing timeseries from supercomputers [26,27]. They are also easy to calculate, reduce storage requirements, and enable us to compare applications that have different durations. We scale each feature to the $[0, 1]$ range according to the values observed in the training set. The same scaling factors are used at runtime.

4.3 Classification

To distinguish a set of given applications, we train a machine learning model using a training set of these labeled applications. Taxonomist labels each run with the corresponding application or it can also label new runs as *unknown*.

For each classifier, we use a *one-versus-rest* version of that classifier: i.e., for each application in the training set, we train a separate classifier that differentiates the application. This approach makes it easy to add a new application to the ensemble of classifiers and to get information about the nature of each application. This approach also enables us to train for only applications of interest, and we do not have to re-train every classifier when a new application is added.

For evaluation purposes, we compare the following classification algorithms: random forests, forests of extremely randomized trees (ExtraTrees), decision trees and the support vector machine classifier (SVC) with linear and radial basis function kernels. In practice, the best performing one for our data is the random forest (§ 6).

From every classifier, we obtain confidence values on whether a new observation belongs to one of the existing training classes. For example, the confidence threshold for the random forest is the percentage of trees in the forest that agree with the final classification. If none of the confidence values are above a predetermined *confidence threshold*, we mark this new observation as *unknown*.

Confidence Threshold Selection. A very high threshold would result in conservatively labeling new inputs of known applications as unknown, while too low values would result in unknown applications being labeled as a similar known application. To select the confidence threshold we first remove each application from the training set and perform testing with examples of that application in the training set while changing the confidence threshold. Then, we remove one input of each application and perform the same test. We select the threshold that results in the highest average F-score for both scenarios.

Hyperparameter Selection. Most classifiers have hyperparameters that describe the configuration of the algorithm. We find the best hyperparameters by splitting the training set into 5 cross validation folds. With 4/5 of the training data we train classifiers with different hyperparameters, and pick the best performing one using 1/5 of the training set. We choose the important hyperparameters for each classifier and over a certain range we train all combinations of hyperparameters, i.e., grid search. We find the best hyperparameter separately for each application's classifier. Note that we never use any test data during training or hyperparameter selection.

4.4 Operation of Taxonomist

During normal operation, Taxonomist uses the monitoring data to label each node of each application after a job finishes. These labels can be used to raise alarms in the case of cryptocurrency mining and to generate system usage reports or other summaries. They can also be used in further research and development on application-specific system optimizations. Furthermore, identifying fraud, waste, and abuse after application completion is still valuable.

As Taxonomist relies on machine learning, it requires a labeled training data set as input. This data set can be collected by a collaboration of users, operations staff, and analysts. After the applications of interest are determined, data can be collected by running them with different input configurations. This training is a one-time effort unless the applications of interest change.

In our current implementation, the application needs to finish before we identify it; however, Taxonomist can be modified to work with only the first few minutes of application data. The strategy proposed by Thebe et al. [25], which executes applications for a short time before the main run is scheduled, can be used with Taxonomist.

5 Experimental Methodology

We run our experiments on a production supercomputer, using the Lightweight Distributed Metric System (LDMS) [2] already in place. We evaluate our system

Table 1: Applications used.

	Application	# of Inputs	# of Ranks	Description
Representative Applications	BT [6]	3	169	Block tri-diagonal solver
	CG [6]	3	128	Conjugate gradient
	FT [6]	3	128	Fourier transform
	LU [6]	3	192	Gauss-Seidel solver
	MG [6]	3	128	Multi-grid on meshes
	SP [6]	3	169	Scalar penta-diagonal solver
	miniAMR [15]	4	192/1536	Adaptive mesh refinement
	miniMD [15]	4	192/1536	Molecular dynamics
	CoMD [15]	3	192	Molecular dynamics
	miniGhost [15]	4	192/1536	Structured PDE solver
	Kripke [17]	4	192/1536	S_N transport
Unwanted Applications[3]	minerd	10	2/4	CPU cryptocurrency miner
	BFGminer	2	2/4	Cryptocurrency miner
	xenon	2	96/192	Zcash competition [29] winner
	davidjaenson	1	2/4	Zcash competitor
	tromp	1	2/4	Zcash competitor
	John the Ripper	194	96/192	Password cracker

with 11 benchmarks, 5 different *unwanted* applications, and also with 6 months of typical supercomputer usage.

5.1 Platform

We run all of our experiments on Volta, a Cray XC30m supercomputer located at Sandia National Laboratories. Volta is composed of 13 fully-connected routers, with 4 nodes each, leading to a total of 52 compute nodes. The operating system used is SLES 11 (SUSE Linux Enterprise Server) with kernel version 3.0.101. Each node has 64 GB of memory and two Intel Xeon E5–2695 v2 CPUs with 12 2-way hyper-threaded cores.

LDMS is a scalable monitoring system deployed on Volta. We use the memory metrics collected from `/proc/meminfo` and `/proc/vmstat`, CPU usage information from `/proc/stat`, and network usage information from Cray network interface card (NIC) counters. 721 metrics from every node every second in total.

5.2 Applications

Representative Applications. We pick a collection of 11 benchmarks and proxy applications, described in the upper section of Table 1. We choose these applications to be representative of characteristic HPC workloads. All representative applications use MPI, and are compiled with the Cray compilers. For each application, we use 3 different input configurations, and we run the applications on 4 nodes. We also run miniAMR, miniMD, miniGhost and Kripke on 32 nodes with an additional input. We run each application on the maximum number of hardware threads available that the application can utilize.

[3] minerd: `www.github.com/pooler/cpuminer`, BFGminer: `www.github.com/luke-jr/bfgminer`, xenon: `www.github.com/xenoncat/equihash-xenon`, davidjaenson: `www.github.com/davidjaenson/equihash`, tromp: `www.github.com/tromp/equihash`, John the Ripper: `www.openwall.com/john`

Unwanted Applications. These are applications that are usually not allowed on supercomputers such as cryptocurrency miners and password crackers. The tromp, davidjaenson, and xenon miners are from an open source miner competition [29]; BFGminer and minerd are popular miners for mining with CPUs. Xenon is single-threaded, so we execute 48 copies per node. Other cryptocurrency miners are multi-threaded, so we execute them one copy per node, using 48 threads. John the Ripper is a popular password cracking application which supports MPI; we execute it one rank per hardware thread. The inputs for John the Ripper are various password formats; and for the cryptocurrency miners, the inputs are the different types of cryptocurrencies. Due to ethical considerations, we ran all of the unwanted applications in benchmark mode to ensure that none of the cryptocurrency mined was connected to the main blockchains.

Typical Volta Usage. This data includes unlabeled applications run by 28 unique Volta users, consisting of 58,366 jobs, from August 2016 until January 2017. Our controlled experiments are removed from these runs.

5.3 Baseline Technique

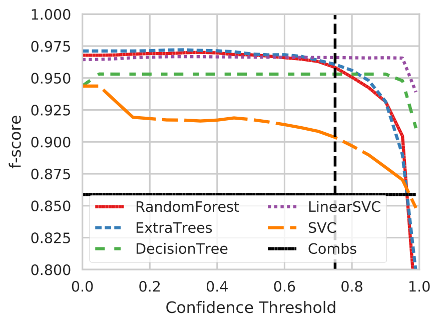
Combs et al. [8] have proposed a technique (referred to as *Combs*) for application detection using power data instead of performance monitoring data. Combs uses a similar feature extraction approach, but in contrast to our method, it extracts serial correlation, non-linearity, self-similarity, chaos, and trend from the timeseries, as well as skew, kurtosis, serial correlation and non-linearity from the timeseries with the trend component removed. Furthermore, Combs et al. normalized maximum and median with the minimum for each timeseries to generate two additional features. Their method uses a random forest classifier and does not have a method for labeling unknown applications, so we do not implement any thresholding for Combs' method.

6 Evaluation

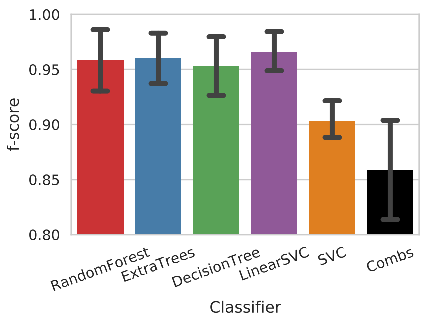
We evaluate the capability of Taxonomist in detecting applications with a variety of workloads and scenarios. First, we examine the classification performance in identifying known applications with new input configurations. Then, we evaluate the performance in labeling unknown applications.

For all tests, we first perform 5-fold cross validation, where we split the whole data into five sets with equal distributions of applications with the original data set. We then train five different Taxonomist instances using four of the sets. For testing, we use the fifth set that was removed from training data. For the normalization and hyperparameter selection steps, Taxonomist performs another 5-fold cross-validation on the training set.

For the results, we report the *F-Score*, which is a widely used measure of classifier performance. For binary classification, F-Score is defined as the harmonic mean of *precision* and *recall.* Precision is the ratio of true positives to the number of all positive predictions, and recall is the ratio of true positives



(a) F-scores for classifiers, vertical dashed line indicates the chosen confidence threshold.



(b) F-scores for classifiers at the chosen confidence threshold, 0.75. Error bars indicate the 95% confidence interval.

Fig. 4: F-scores with one input configuration removed from training. In most cases, the applications are correctly identified in spite of the unknown input configuration.

to the number of all actual positives in the data set. F-Score ranges between 1 (best) and 0 (worst). All of our results are multi-class; therefore we calculate the average precision and recall for each class, and take the harmonic mean to calculate the overall F-score.

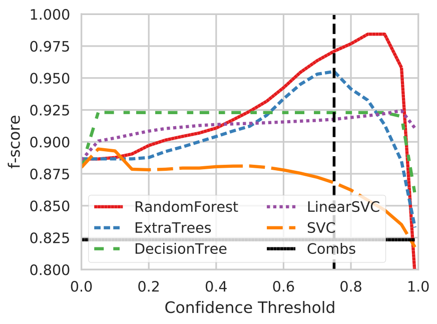
Full Data Set. Table 2 shows the 5-fold cross validation results on the 11 representative applications. All of the results except the baseline technique (Combs) have an F-Score of over 0.99. However, this scenario where the training data contains all applications and all input configurations is unrealistic. SVM with the linear kernel (LinearSVC) performs better than the rbf kernel (SVC). This is likely due to the large data set with many features and datapoints, and this behavior is consistent with the literature [16].

Table 2: Five-fold cross validation results with the full data set.

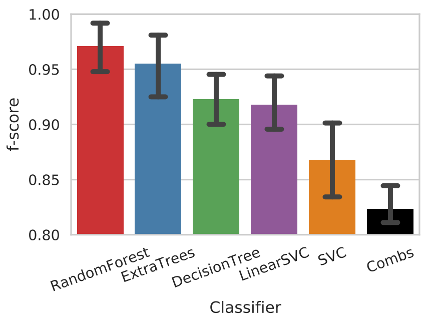
Classifier	Precision	Recall	F-score
RandomForest	1.000	1.000	1.000
ExtraTrees	1.000	1.000	1.000
DecisionTree	0.998	0.998	0.998
LinearSVC	0.999	0.999	0.999
SVC	0.994	0.994	0.994
Combs	0.932	0.931	0.931

Detecting Applications with Unknown Input Configurations. Applications' resource usage is affected by their input configurations. To evaluate Taxonomist's robustness against input configurations that are not in the training set, we remove one of the input sets from the training set. For the test set, we keep the cross validation folds the same. Figure 4 shows that the classification is successful unless the confidence threshold is over 0.9, in which case the unknown input configurations are marked as unknown applications.

Detecting Unknown Applications. Figure 5 shows classification results with one application removed from the training set. If the removed application is labeled as unknown, we mark it as a correct prediction. In the majority of



(a) F-scores for classifiers, vertical dashed line indicates the chosen confidence threshold.



(b) F-scores for classifiers at the chosen confidence threshold, 0.75. Error bars indicate the 95% confidence interval.

Fig. 5: F-scores with one application removed from the training set. With the correct confidence threshold choice, the unknown application can be correctly identified.

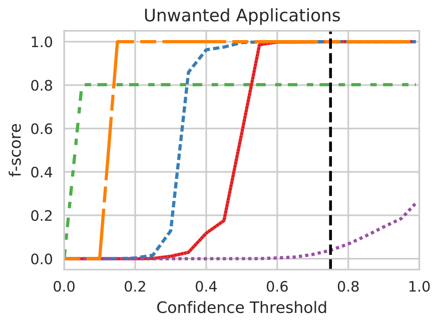
the cases, the unknown application is correctly identified as such. The lowest F-Scores are for the BT and SP applications, which are both partial differential equation solvers and they have been shown to have similar behavior [18]. Hence, the classifiers tend to mispredict SP and BT.

The confidence threshold that gives the maximum value for the average F-scores of the unknown input and unknown application cases is 0.75, and Random Forest is the classifier that gives the best average F-score.

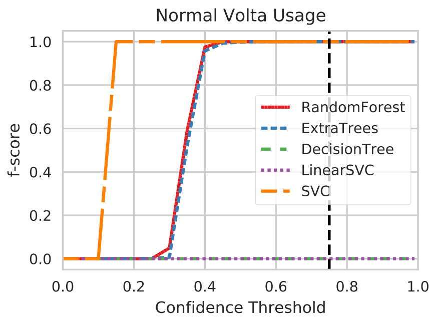
Unwanted Applications and Typical Volta Usage. We show Taxonomist's ability to identify unknown applications from different domains by testing with unwanted applications such as bitcoin miners, shown in Fig. 6a, and with 6 months of Volta usage data, shown in Fig. 6b. In both of these tests, we train Taxonomist with the 11 representative applications, and consider the unknown label to be correct. Random Forest, Extra Trees and SVC have an almost perfect F-score for identifying any of these applications as unknown. Combs is not shown, because it is unable to identify unknown applications.

Feature Importance. In order to present the importance of different statistical features and metrics, we train a decision tree for each application, using all of the data from the 11 applications. To compare feature importances, we use *Gini reduction*, which is used to measure the reduction of heterogeneity in the data. A feature that can divide the data set well has a high Gini reduction, which means the resulting divided data sets are more homogeneous. We use the implementation in Python `scikit-learn` library (`sklearn.DecisionTreeClassifier.feature_importances_`).

In the decision trees corresponding to our 11 applications, we calculate the total Gini reduction of features extracted using the 11 statistics (§ 4.2), and

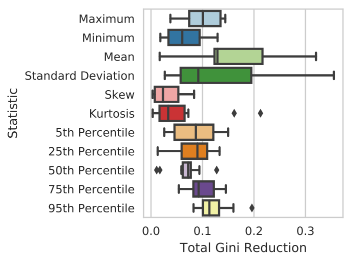


(a) F-scores when tested with bitcoin miners and password crackers.

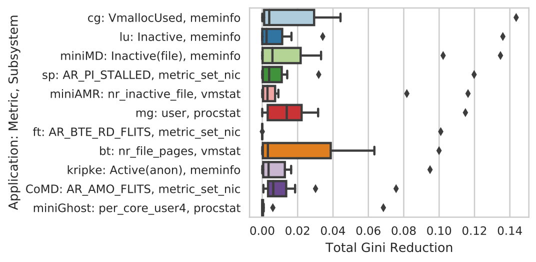


(b) F-scores when tested with HPC applications that are not known to the classifiers.

Fig. 6: The classifiers can correctly identify unknown applications, whether they are HPC applications or bitcoin miners and password crackers.



(a) The importance of each statistical measure.



(b) The most important metric for each 11 applications and the metric's source subsystem.

Fig. 7: The importance of different metrics and statistics. Box-plots are constructed using the different decision trees for each application. The box shows the quartiles while the whiskers show the rest of the distribution except outliers, which are points away from the low and high quartiles by more than $1.5 \times IQR$.

report it in Fig. 7a. The box-plots are constructed using the data from the decision trees, and the individual importance values from the trees are summed up. Fig. 7b shows the most important metric from each decision tree. The important metric and subsystem[4] are highly application specific.

[4] metric-set-nic: Cray network counters [9], vmstat: `/proc/vmstat`, meminfo: `/proc/meminfo`, procstat: `/proc/stat`, AR stands for AR-NIC-RSPMON-PARB-EVENT-CNTR

7 Conclusion

We have presented Taxonomist, a technique for classifying applications in supercomputers with the help of readily available monitoring data. The technique builds classifiers from historical data, and detects new applications while being robust to new input configurations of applications. We have evaluated Taxonomist using a comprehensive data set including controlled experiments and real-world workloads and demonstrated F-scores of over 95%.

Data Availability Statement and Acknowledgment. The datasets generated during and/or analyzed during the current study are available in the Figshare repository: https://doi.org/10.6084/m9.figshare.6384248 [4].

This work has been partially funded by Sandia National Laboratories. Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC., a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under Contract DE-NA0003525.

References

1. ASCR cybersecurity for scientific computing integrity. DOE Workshop Report (2015)
2. Agelastos, A., et al.: The lightweight distributed metric service: A scalable infrastructure for continuous monitoring of large scale computing systems and applications. In: International Conference for High Performance Computing, Networking, Storage and Analysis (SC). pp. 154–165 (2014)
3. Agelastos, A., et al.: Toward rapid understanding of production hpc applications and systems. In: IEEE International Conference on Cluster Computing. pp. 464–473 (2015)
4. Ates, E., Tuncer, O., Turk, A., Leung, V.J., Brandt, J., Egele, M., Coskun, A.K.: Artifact for Taxonomist: Application detection through rich monitoring data (2018). https://doi.org/10.6084/m9.figshare.6384248
5. Auweter, A., et al.: A case study of energy aware scheduling on supermuc. In: Kunkel, J.M., Ludwig, T., Meuer, H.W. (eds.) Supercomputing. pp. 394–409. Springer International Publishing, Cham (2014)
6. Bailey, D., et al.: The nas parallel benchmarks. The International Journal of Supercomputing Applications **5**(3), 63–73 (1991)
7. Bhatele, A., Mohror, K., Langer, S.H., Isaacs, K.E.: There goes the neighborhood: Performance degradation due to nearby jobs. In: SC'13. pp. 41:1–41:12. ACM, New York, NY, USA (2013)
8. Combs, J., et al.: Power signatures of high-performance computing workloads. In: Proceedings of the 2nd International Workshop on Energy Efficient Supercomputing. pp. 70–78. E2SC '14, IEEE Press, Piscataway, NJ, USA (2014)
9. Cray: Aries hardware counters (s-0045-20). Tech. rep. (2015), http://docs.cray.com/books/S-0045-20/S-0045-20.pdf
10. Dart, E., Rotman, L., Tierney, B., Hester, M., Zurawski, J.: The science DMZ: A network design pattern for data-intensive science. In: SC'13. pp. 1–10 (2013)

11. DeMasi, O., Samak, T., Bailey, D.H.: Identifying hpc codes via performance logs and machine learning. In: Proceedings of the First Workshop on Changing Landscapes in HPC Security. pp. 23–30. ACM, New York, NY, USA (2013)
12. Dongarra, J., et al.: The international exascale software project roadmap. Int. J. High Perform. Comput. Appl. **25**(1), 3–60 (2011)
13. Egele, M., Woo, M., Chapman, P., Brumley, D.: Blanket execution: Dynamic similarity testing for program binaries and components. In: 23rd USENIX Security Symposium. pp. 303–317. USENIX Association, San Diego, CA (2014)
14. Florez, G., Liu, Z., Bridges, S.M., Skjellum, A., Vaughn, R.B.: Lightweight monitoring of mpi programs in real time: Research articles. Concurr. Comput. : Pract. Exper. **17**(13), 1547–1578 (2005)
15. Heroux, M.A., et al.: Improving Performance via Mini-applications. Tech. Rep. SAND2009-5574, Sandia National Laboratories (2009)
16. Hsu, C.W., Chang, C.C., Lin, C.J., et al.: A practical guide to support vector classification. Tech. rep. (2003), https://www.csie.ntu.edu.tw/~cjlin/papers/guide/guide.pdf
17. Kunen, A., Bailey, T., Brown, P.: Kripke-a massively parallel transport mini-app. Tech. rep., Lawrence Livermore National Laboratory, Livermore, CA (2015)
18. Ma, C., et al.: An approach for matching communication patterns in parallel applications. In: IEEE International Symposium on Parallel Distributed Processing. pp. 1–12 (2009)
19. NERSC: Number of NERSC users and projects through the years (2016), www.nersc.gov/about/nersc-usage-and-user-demographics/number-of-nersc-users-and-projects-through-the-years/
20. Office of Inspector General: Semiannual report to congress (2014), https://www.nsf.gov/pubs/2014/oig14002/oig14002.pdf
21. Peisert, S.: Fingerprinting communication and computation on HPC machines. Lawrence Berkeley National Laboratory (2010). https://doi.org/10.2172/983323
22. RedLock CSI Team: Lessons from the cryptojacking attack at Tesla. Tech. rep. (2018), https://blog.redlock.io/cryptojacking-tesla
23. Rosenberg, E.: Nuclear scientists logged on to one of russia's most secure computers — to mine bitcoin. The Washington Post (2018)
24. Skinner, D., Wright, N., Fuerlinger, K., Yelick, K., Snavely, A.: Integrated performance monitoring ipm (2009), ipm-hpc.sourceforge.net/
25. Thebe, O., Bunde, D.P., Leung, V.J.: Scheduling restartable jobs with short test runs. In: Frachtenberg, E., Schwiegelshohn, U. (eds.) Job Scheduling Strategies for Parallel Processing. pp. 116–137. Springer Berlin Heidelberg (2009)
26. Tuncer, O., Ates, E., Zhang, Y., Turk, A., Brandt, J., Leung, V.J., Egele, M., Coskun, A.K.: Diagnosing Performance Variations in HPC Applications Using Machine Learning, pp. 355–373. Springer International Publishing, Cham (2017)
27. Wang, X., Smith, K., Hyndman, R.: Characteristic-based clustering for time series data. Data Mining and Knowledge Discovery **13**(3), 335–364 (2006)
28. Whalen, S., Peisert, S., Bishop, M.: Multiclass classification of distributed memory parallel computations. Pattern Recognition Letters **34**(3), 322 – 329 (2013)
29. Zcash Electric Coin Company: Zcash open source miner challenge (2016), www.zcashminers.org