Analysis of XDMoD/SUPReMM Data Using Machine Learning Techniques

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, Ryan Rathsam
Center for Computational Research
University at Buffalo, State University of New York
Buffalo, NY

***Abstract*— Machine learning techniques were applied to job accounting and performance data for application classification. Job data were accumulated using the XDMoD monitoring technology named SUPReMM; they consist of job accounting information, application information from Lariat/XALT, and job performance data from TACC_Stats. The results clearly demonstrate that community applications have characteristic signatures which can be exploited for job classification. We conclude that machine learning can assist in classifying jobs of unknown application; in characterizing the job mixture; and in harnessing the variation in node and time dependence for further analysis.**

Keywords—XDMoD; Open XDMoD; SUPReMM; machine learning; HPC monitoring; TACC_Stats; application classification

I. XDMoD AND OPEN XDMoD WITH SUPREMM

High Performance Computers are a combination of complex commodity level compute servers, network gear, and storage subsystems. While the change to commodity level components for supercomputers has been beneficial in terms of cost, it has made managing them to ensure they are operating optimally much more challenging. Given the important role they play in science and engineering and the fact that they are almost always oversubscribed, with a backlog of jobs waiting to run, there has been a great need for an effective tool to manage supercomputers to ensure that all the subcomponents are operating optimally and that application codes are running efficiently.

The National Science Foundation recognized the need for a comprehensive management tool for its supercomputing portfolio and in 2010 awarded UB's Center for Computational Research (CCR) with a 5-year program to, among other things, develop the XD Metrics on Demand (XDMoD) tool for XSEDE, the world's largest distributed supercomputing infrastructure for open, scientific research [1].

Such data driven management requires a good set of tools for data collection and analysis at different complexity and scale. While there have been some tools for system measurement (see for e.g. [17]) they have been targeted at application tuning and do not use system usage data of the system being targeted. We started with the core functionality of data collection as described in earlier reports on XDMoD [2-9]. Initial attempts at analysis discussed in these earlier reports were largely comprised of characterization. We present here more complex analyses of this data using modern machine learning based tools to extract more useful information from this system data. Such an analysis will be of use in both a posteriori/ in-situ characterization of system usage and predictive analysis of jobs.

The XDMoD framework has been described previously [2-9] and here we will provide only a brief description in order to provide a context for the discussion of the application of machine learning technology to XDMoD data. XDMoD ingests and organizes data on computer system behavior from multiple sources and then maps that data into metrics required for overall system management. Metrics provided by XDMoD include: number of jobs, CPUs consumed, wait time, and wall time, with minimum, maximum and the average of these metrics, in addition to many others. These metrics can be broken down by: field of science, institution, job size, job wall time, NSF directorate, NSF user status, parent science, person, principal investigator, and resource. The success of XDMoD in helping to manage the NSF portfolio of supercomputing sites led to the development of Open XDMoD, an open source version of XDMoD for use by academic and industrial HPC centers [7]. Released in late 2013, Open XDMoD has already been deployed at well over 60 institutions globally. In addition to usage metrics, quality of service (QoS) metrics and application code performance metrics (flops, IO rates, network metrics, etc) for all applications running on a given resource are also available. QoS is measured through application kernels [2,7,8] and job level performance is measured through the SUPReMM project [4-7,9], both of which are described in greater detail below.

Application kernels are designed to provide quality of service metrics for HPC systems. Typically, HPC facilities do not have a mechanism to monitor the quality of service they provide to their end-users. Instead, often times the end-users are the "canaries in the coal mine" who report problems to center support personnel when their jobs suddenly run poorly or fail to run altogether. The key idea behind the application kernels is to periodically run a series of computationally lightweight benchmarks and applications through the normal user submission queues, in order to proactively detect problems with the hardware and software. The resulting application kernel job data is ingested into the XDMoD or Open XDMoD data warehouse, and process control algorithms automatically detect underperforming application

978-1-4673-6598-7/15 $31.00 © 2015 IEEE
DOI 10.1109/CLUSTER.2015.114

kernels and notify support staff. The application kernels are designed to span all aspects of the HPC cluster operation (compute, storage, and network).

The most pertinent enhancement of XDMoD and Open XDMoD for the purposes of this paper is SUPReMM. The SUPReMM (Integrated HPC **S**ystems **U**sage and **P**erformance of **R**esources **M**onitoring and **M**odeling) project integrates tools for the collection of job performance data with the powerful data warehousing and analysis capabilities of XDMoD. While multiple open source data collection tools are supported by SUPReMM including Performance Co-Pilot (PCP) [11] and TACC_stats [3-6,12], the data presented in this paper are from TACC_Stats and Lariat (soon to be replaced by XALT). Lariat captures, among other things, the application path and environment module information for all jobs that are launched using ibrun on the Texas Advanced Computing Center (TACC) machines. TACC_Stats, an enhancement to the conventional Linux "sysstat" system-wide performance monitor, collects a range of node-level performance information [4-6] and generates aggregated job-centric views of performance data. In addition to the information gathered by systat, TACC_Stats records hardware performance counter values, parallel filesystem metrics, and high-speed interconnect usage, resolving the measurements by job and core. The basic component is a collector executed on every compute node, both at the beginning and end of each job (via batch scheduler prolog and epilog scripts) and at periodic intervals (via cron every ten minutes by default). Data collection takes place in the background, incurs very low overhead, and requires no modification to user or system applications. TACC_Stats is open source and is freely available for download [13].

The remainder of this paper is organized as follows: Section II gives a general introduction to using machine learning algorithms on SUPReMM data, Section III gives a detailed description of using machine learning to classify jobs by application and finally Section IV summarizes what has been learned and discusses future work.

II. Machine Learning Analysis of SUPReMM Data

The set of data generated by SUPReMM consists of hundreds of metrics in the broad categories of CPU usage, memory usage, network usage and I/O usage. These metrics are typically averaged over all nodes for the duration of the job and reported as summaries of job activity. However, the sheer volume of data makes it challenging to convert this data into information that would be of use in understanding HPC operations. Based on our experience with the performance data, we have selected the metrics that appear to be of the greatest importance and made them accessible through XDMoD but even with this reduced set of metrics it can be difficult to holistically evaluate the job mixture. Since the SUPReMM data has a large number of samples, a moderate number of predictor attributes, and can be labeled in several possible ways we feel that it is well suited to machine learning and data discovery techniques such as classification, dimensionality reduction, and clustering that other workers have applied to similar data, see for example [10,18] and references therein.

For the purposes of this analysis, we selected metrics for the set of all jobs run on TACC Stampede in 2014 consuming more than one minute of wall time. This resulted in a total of 1,683,850 individual jobs of which 969,641 were labeled with application names extracted from Lariat data, 238,929 contained application names but were unable to be categorized using the executable path or name (e.g., “data”, “main”, or other user-compiled codes), and 475,280 where application information was unavailable (e.g., they were run outside of ibrun). A brief general description of the most important metrics is included in Table 1.

Table 1: Brief description of SUPReMM metrics included.

Metric	Description
CPU SYSTEM, CPU USER, CPU IDLE	Fraction of CPU time spent in kernel mode, user mode, or idle.
CPLD	Clock ticks per L1D cache: Average ratio of clock ticks to L1D cache loads per core.
CPI	Clock tics per instruction: Average ratio of clock ticks to instructions per core.
FLOP	Total floating point operations.
MEMORY USED	Total memory used per node. This value excludes the OS buffer cache and kernel memory.
MEMORY TRANSFERRED	Total memory bandwidth in bytes per second.
ETHERNET TRANSMIT	Total bytes transmitted over the ethernet device per node.
INFINIBAND RECEIVE	Total bytes received over the InfiniBand device per node.
HOME WRITE, SCRATCH WRITE	Total bytes per node written to the indicated directory file.
LUSTRE TRANSMIT	Total data transmitted by the Lustre filesystem driver per node.
LOCAL DISK READ IOS, LOCAL DISK READ BYTES	The total local disk reads in bytes per second or operations per second.
NODES	Number of nodes on which the job was executed.
…..COV	When COV is appended to the metric name, it indicates the coefficient of variation of the metric over the various nodes in the job, calculated by computing the standard deviation of the data for the nodes and dividing by the mean value, is used rather than the mean value of the metric itself.

There are a number of different ways to view the SUPReMM data and types of analyses that can be performed. Our initial investigations involved automatic identification of underperforming or inefficient applications. We manually classified a set of 80,000 SUPReMM data points as efficient or inefficient. Where inefficient jobs were defined to be those with < 30% CPU USER; CPI values < 2; CPLD > 0.1, CATASTROPHE (a metric indicating a shut down of the CPU activity partway through the job) < 0.2; or CPU USER IMBALANCE (a metric indicating some CPUs are not being used) > 1. The data were selected to be completely separable and only intended to test different machine learning classification tools. Initially we tried three different types of machine learning classifiers: Naïve Bayesian (nb) [14],

Support Vector Machine (svm) [14] and Random Forest (rF) [15]. The Naïve Bayesian classifier performed very poorly on this problem, which is not surprising since the *a priori* data distributions are not normal and the metrics are known to be correlated. We then investigated the svm and rF classifiers which do not require normally distributed data or linearly independent attributes and should be well suited to this case where a simple classification decision is needed. Using a class-balanced training sample of 116,371 jobs and test sample of 77,553 jobs, both the svm and the rF were very successful as job application classifiers on this test data set and achieved nearly 100% correct classification of the withheld test data for this simple problem. The support vector machine classifier is a supervised learning model which works by using the training data to construct hyperplanes that separate one class from all of the others. The best hyperplane is that which produces the largest separation between the given class and all of the others. The random Forest classifier works on a completely different algorithm. The rF classifier uses the training data to construct an ensemble of decision trees. The classification of the test data is then done by polling this ensemble.

We next applied the svm and rF classifiers to identify jobs as either successful or failed using the job exit code as a label. Although both classifiers trained very well, they were not very successful in predicting the success or failure status of the jobs in the withheld test data. This is not entirely surprising, since the exit code for a job is typically the exit code of the last operation that occurred in the run script. It is entirely possible that, while an application completed successfully, a failure code was returned. For example, a remove or grep operation in the run script could return an unsuccessful exit code. The conclusion is that the available SUPReMM data does not contain the requisite information to perform this particular classification task. We plan to revisit this issue when more SUPReMM attributes, especially time dependent variation metrics are added to the SUPReMM data set. Finally, we applied the machine learning technology to the important and practical task of classifying jobs by their application. The next section will describe the results of this analysis.

III. Classification of Jobs Using Machine Learning

We will describe the results from an analysis that has generated very promising results, that is, the machine classification of jobs by their application name based on performance data (e.g., VASP, LAMMPS, etc.) For facilities that collect the job executable information (e.g., by using Lariat or XALT), job data is associated with an application by comparing the path of the job executable and matching this to a known list of 121 community applications. While the applications for many jobs are identified by this method, there are two sets of job data that go unidentified. For the first set, called "Uncategorized", we cannot identify any known application from the executable path. Many of these jobs are user-compiled codes and test applications with names such as "a.out", "main", "data", etc. The second set, predominantly those jobs not launched using ibrun on the TACC facilities have no Lariat data available and are labeled "NA" (Not Available). We set out to develop a machine learning solution for automatic classification of jobs to learn more about these unclassified jobs and find out whether they are similar enough to the known applications to classify them.

We used the svm classifier from the R package e1071 [14], using the radial basis kernel tuned with gamma = 0.1 and cost = 1000. We defer discussion on the SUPReMM metrics included in the model to later when we discuss their importance in detail. We trained on an application-balanced mixture of 100,000 jobs from Stampede. The classifier was able to correctly predict 99.95% of the known jobs in the training set. We applied this trained svm to a test mixture containing another 100,000 jobs (based on 20 unique applications), and found that 97% of the jobs were properly classified by application. Table 2 summarizes the 20 X 20 confusion matrix for this classification process, with each row representing the results for one of the 20 unique applications. Table 2 was constructed such that column 1 represents the actual application with the number of times it was correctly classified in parentheses. The second column includes the off diagonal elements of the matrix which list all the misidentified applications along with the number of times each application was misidentified in parentheses. For example, in the first row, AMBER was misidentified as GROMACS 5 times. Based on the off-diagonal elements of the confusion matrix, two factors seem to enter into the misclassifications. The first is the number of jobs the particular applications have in the native job mix. For example VASP and NAMD are the two most used applications in the mixture at 33% and 17%, respectively. VASP is often mistaken for NAMD while NAMD is most often mistaken for VASP. This could possibly be ameliorated by weighting the classes or using a non-native job mixture. The other type of misclassification is mistaking two similar types of applications. For example, GROMACS is most often misclassified as LAMMPS where both are molecular dynamics simulation codes. This will be further expanded upon when we discuss classification of groups of similar applications.

The fraction of correctly classified applications was a very encouraging result; however, one final step is required before the svm classifier could be applied to the two unclassified job data sets. Since it is very likely that there are applications in the Uncategorized and NA data sets which are not in the set of community applications on which the svm classifier was trained, it is necessary to look at the probability that the jobs are properly classified. The svm classifier will find the best match for the job, but if the probability of the classification is very low, below some selected threshold value, we would conclude that the job was not classified. For example, if a job is classified as application A with only one percent higher probability than application B, how confident are we that this is the correct classification? Figure 1, below, shows a plot indicating the percent of the test jobs that are classified as a

function of the threshold probability. For example, over 85% of the test jobs are considered classified, even if we require a 90% probability threshold. Note that for the high probability classifications the correctly classified and the classified curves are in complete alignment. As the probability requirement is relaxed, although more jobs are classified, the correctly classified curve drops below as some jobs are incorrectly classified. The user can select the probability threshold to match the requirements of the classification process depending on how important it is to classify as many jobs as possible and how important it is to have as high a correct classification ratio. For example, Figure 1 shows that over 90% of the jobs can be classified while incurring very few misclassifications.

The svm and rF classifiers were extensively tested and compared. While true ROC (receiver operating characteristic) curves can only be made for binary selection processes, one can compare the fraction of jobs classified correctly to the fraction classified incorrectly as a function of probability threshold as described in Equation 1. Each point (x,y) is a function of *t,* the probability threshold. *P* and *C* are logical vectors indicating whether or not a given job classification has met the threshold and has been classified correctly or incorrectly at that threshold, respectively. *N* is the total number of jobs classified correctly or incorrectly. This plot is shown in Figure 2 and compares the performance of the two different algorithms. Both classifiers do an excellent job on this classification problem and approach the ideal behavior.

Equation 1

$$(x,y) = f(t) = \left(\frac{\sum(P_t \wedge C_{correct})}{N_{correct}}, \frac{\sum(P_t \wedge C_{incorrect})}{N_{incorrect}}\right)$$

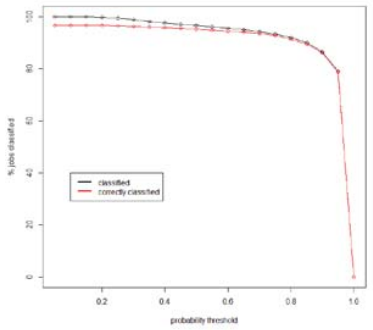

Figure 1 Plot indicating the percentage of jobs that are classified (black) and those that are correctly classified (red) in the test data set ***as a function of probability threshold****. Over 90% of the jobs are classified at the cost of only a minimal number of misclassified jobs.*

Finally, we applied the svm classifier to the Uncategorized and NA jobs. Figure 3 shows the result of the percentage of

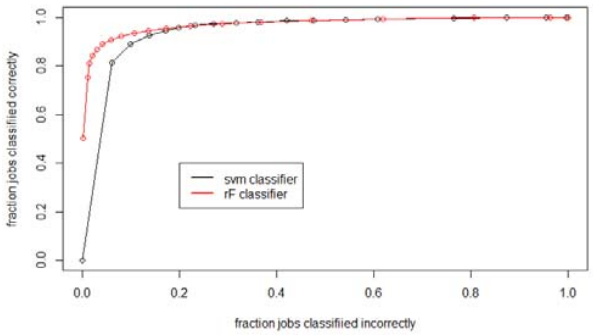

Figure 2 ROC-like plot comparing the svm (black) and rF (red) classifiers ***as a function of probability threshold*** *from 1.0 to 0.05 in decreasing steps of 0.05.*

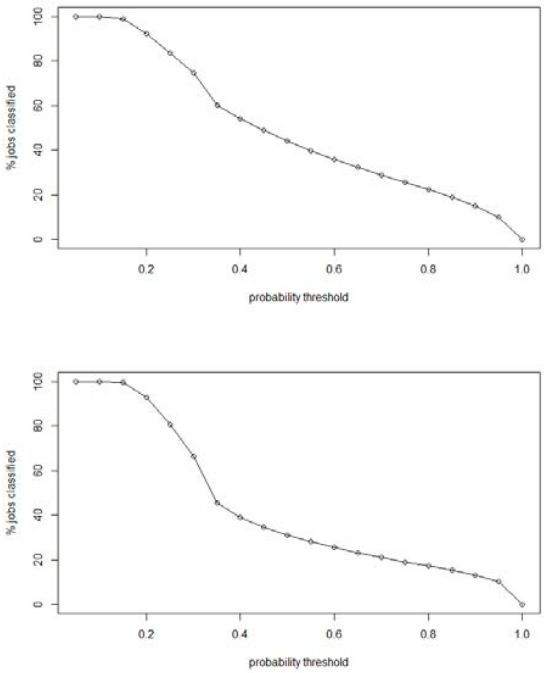

Figure 3 Plots indicating the percentage of jobs that are classified as a function of probability threshold in the uncategorized data set (top) and the NA data set (bottom). The low number of jobs classified ~20% or less indicates that the uncategorized and NA data sets are very different from the majority of the job mixture where the jobs are classified by application. Compare to Figure 1 above.

jobs that can be classified as a function of probability threshold. Very few jobs can be classified, on the order of 20% or less, for a ~0.8 probability threshold. The contrast between Figures 1 and 3 is striking. Although it is disappointing that a larger percentage of the unclassified jobs could not be classified, it is perhaps not surprising since the uncategorized job data sets mainly consist of custom

applications not in the set of community applications that make up the bulk of the job mix.

The classification of the Uncategorized and NA data sets by application indicated that these job mixtures were very different from the job mixture for which the applications were known. We therefore tried to group the applications into broad categories that were determined by the type of problems solved by the applications to see if at least a general classification by application type could be made. The applications were grouped together into 12 broad categories by application type: Math, Matlab, Benchmark, E/M,photonics, Lattice Quantum Chemistry, Python, Astrophysics, Quantum Chemistry, Molecular Dynamics, Computational Fluid Dynamics, Extended System Quantum Chemistry and Unknown. An svm classifier was trained and was able to classify known applications from a test set into the broad categories with a 97% success rate, see Table 3 below. The only groups that are not well classified are those which are represented by a very small number of jobs such that the svm trainer had a very limited data set to work with. This could be modified by using a non-native training mixture that includes more of the less common groups or simply lumping the less common groups in with the Unknown group. Given this success, the trained classifier was applied to the Uncategorized and NA data sets. Figure 4 shows the probability threshold plot for the Uncategorized and NA data sets. The distribution of this data is very similar and only slightly improved over the simple application plots shown in Figure 3. This again highlights how different the Uncategorized and NA job sets are from the mainstream characterized job set.

Table 3. Classification by general application type

group_name	number	% mix	% correct
Astrophysics	8200	2.94	87.01
benchmark	1238	0.44	76.27
CFD	10405	3.74	90.63
E&M,photonics	2922	1.05	98.39
Lattice QCD	344	0.12	89.38
Math	766	0.28	73.70
Matlab	125	0.04	91.43
MD	111102	39.90	98.71
Python	2041	0.73	65.67
QC	7670	2.75	94.60
QC,ES	120236	43.18	98.25
Unknown	13414	4.82	87.52

An analysis of the importance that particular attributes play in the various machine learning models is crucial in understanding the SUPReMM data as well as the idea that applications can be positively identified using a signature. Unfortunately, the e1071 svm packages in R and even the service package caret [16] do not provide direct programmatic access to this information. However, the randomForest package does provide information on the attribute importance and has allowed us to identify a subset of the attributes that have the most impact on the correct classification of applications. Although there may be some differences between the two different classification algorithms, we expect that the attribute importance should be similar to the extent that each is able to take advantage of the information content of the data and we plan to test this in the future.

Figure 5 shows an output of the attribute importance plot from the randomForest model. It is generated by permuting only a single variable in the presence of all other variables in the model. The larger the effect that this variation produces, the more important the attribute is to the model. Examining Figure 5, we see that the 4 most important attributes are: MEMORY USED, CPI, CPU SYSTEM, and CPLD. The covariance attributes were added to see if SUPReMM data

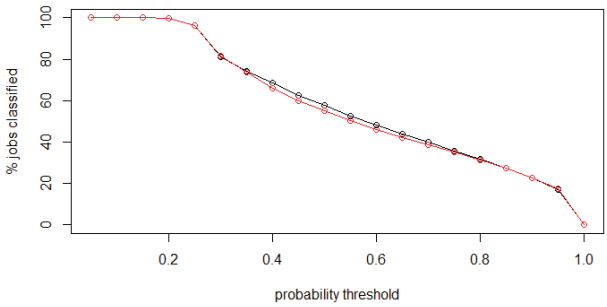

Figure 4 Plot indicating the percentage of jobs that are classified into one of 12 broad categories as a function of probability threshold in the Uncategorized (black) and NA (red) data sets. The low number of jobs classified ~20% or less indicates that the Uncategorized and NA data sets are very different from the majority of the job mixture where the jobs are classified by application. Compare to Figures 1 and 3 above.

which would be difficult for a human observer to interpret, could provide information that would improve the model. Looking at the importance plot it is clear that this is the case. These four most important attributes are mainly concerned with CPU and memory usage. Noticeably absent is any IO or network attributes. The next 6 attributes, MEMORY USED COV through LUSTRE TRANSMITTED COV still seem to be making a contribution, albeit significantly less. Looking at this list there are CPU, memory and IO attributes (including several COV attributes) but no non-IO related network contributions. The final 20 attributes, CPI COV through the end, are making diminishing contributions. Note that all of the non-IO networking attributes are in this final category. This attribute importance analysis indicates that the memory usage, CPU usage and to a lesser extent IO usage contribute to the application signature but network usage contributes very little. There is one caveat that should be noted when analyzing the importance data from Figure 5. Some of the variables are highly correlated. Since attribute importance is computed by permuting only a single variable in the presence of all others, if a variable is highly correlated with another variable it may appear to be relatively unimportant since it adds very little to

the model when its correlated mate is present but it may seem much more important if that correlated variable was removed from the model. For example CPU USER, CPU IDLE and CPU SYSTEM always add up to one. CPU SYSTEM was found to be the third most important variable while CPU IDLE and CPU USER were well down the list. It may be that if CPU SYSTEM was removed from the model that CPU IDLE or CPU USER would substantially increase in importance.

Figure 5 Plot indicating the importance of the various SUPReMM data attributes in the randomForest classifier model. The top 4 on the list are the most important differentiating between applications. The next 6 seem to still be contributing and the final 20 are making at most small contributions.

In order to gauge the role that these more important attributes play in classification accuracy, we began by training a randomForest classifier using the full set of available attributes to achieve a 97% success rate on the test data set. Removing five highly correlated attributes such as the number of file device IOPs and read/write rates also resulted in a 97% success rate, as expected. We then examined the mean decrease in accuracy (i.e., attribute importance) values generated by the random forest classifier trained on the full set of attributes and programmatically varied a cutoff value to identify attributes falling below the cutoff. For each value of the cutoff, we removed these less important attributes and trained a new model using from 43 to 1 attribute. Figure 6 shows the resulting classifier accuracy as a function of the number of predictors used. While the accuracy of the each new model steadily decreased, it remained at or above 90% until the number of attributes fell below five. In most models these five attributes (CPI, CPLD, CPU SYSTEM, MEMORY USED, MEMORY USED COV) are the most important components of the application signature and models including only these 5 produce an accuracy of 90%. Interestingly, this set does not include any attributes directly related to filesystem or network I/O. The inclusion of the number of bytes read for both local disk and network filesystem increases the overall model accuracy by only 3%. Other combinations of attributes can produce results with similar accuracy. For example, the combination of CPU SYSTEM, MEMORY USED COV, LOCAL DISK READS, INFINIBAND READS COV, and SCRATCH WRITES also yields approximately a 90% accuracy.

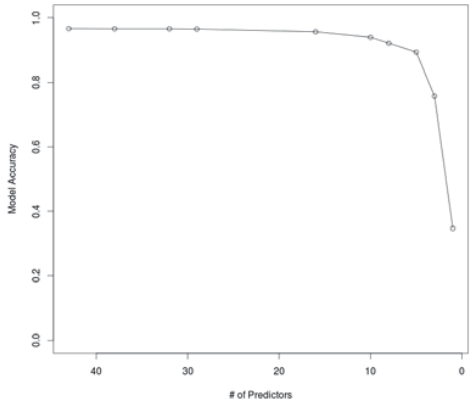

Figure 6 Plot of model accuracy vs. number of predictors.

IV. Discussion and Future Work

A number of conclusions can be drawn from this preliminary study using machine learning algorithms to analyze SUPReMM job data. Clearly, machine learning can make a real contribution to the analysis of SUPReMM performance data. For example, this analysis tells us that we are capturing most of the usage of the popular community applications based on parsing the job scripts from jobs that are recognizably launched as an identifiable application. Our analysis demonstrates that very few instances of these community applications are missed because the application was launched in a manner such that we missed the data or were not able to identify the application. The present analysis clearly supports the idea that these community codes can be identified based on a signature derived from performance data collected during their execution and that the SUPReMM data is sufficient to establish a signature for each application. Furthermore, even using the reduced set of SUPReMM

metrics currently included in the XDMoD data warehouse, this signature can be used as the basis for automatic machine learning classification. Other SUPReMM metrics can be added to the job data that may further improve the job classifiers. In fact, during the course of this analysis, we added some attributes that looked at the variation in the recorded metrics rather than their mean values that made a real contribution to improving the machine classification. This is an interesting point, because it means that we are taking advantage of information from the SUPReMM data set using machine learning that would not be otherwise useful. We have made some preliminary randomForest models in which time dependent attributes rather than the mean attributes were used for the classification. These models worked very well and were approximately as good as the models using mean attributes. One advantage of the time dependent attributes is that they have the potential to serve as the basis for cross platform classifier models. Some initial efforts developing time dependent attribute based cross platform classification models showed limited success. They were superior to the mean based cross platform classifiers, however any cross platform classifier can only be useful if the job mixes on the two platforms contain similar applications.

Finally, we have just made one initial step in the possible applications of this technology. We do plan to develop the machine learning technology that was explored in this work into production tools for use in XDMoD. We anticipate extending this study in a number of different directions. We will explore the classification of jobs by various means: by efficiency, and by whether they fail or go to completion. Another avenue for investigation is to refine the classification of jobs by broad grouping based on the similarity of the applications that was briefly initiated. We may also attempt to account for time-dependent performance data, for example metrics that spike at different times during the execution of a job such as I/O, network traffic, or FLOPS. The question can be asked if jobs from these applications grouped together are more similar to one another than they are to jobs from other groups of applications. This analysis can also be extended to other machines; we can look for similarities and differences across different platforms and architectures. We can also add other data such as the time based variation of metrics that can potentially be analyzed and contribute to some of the previously listed analyses. The data for these metrics has already been collected, it is just a matter of reprocessing the job set to include them in the data warehouse. Finally, such machine learning techniques can be applied to perform a multivariate regression analyses on job data sets. For example a regression analysis can be applied to the application kernel data available in XDMoD. The key to the application kernels is that the same set of applications are run repeatedly using identical inputs. We have done some initial svm and rF regression analysis of the application kernel data. Initial efforts have been successful in modeling wall time on Stampede for all of the application kernels. Such a regression analysis of this SUPReMM data may reveal useful information on HPC performance.

Acknowledgments

This work is supported by the National Science Foundation under grant number OCI 1203560 for SUPReMM and grant number OCI 1025159 for the technology audit service for XSEDE.

REFERENCES

[1] J. Towns, T. Cockerill, M. Dahan, I. Foster, K. Gaither, A. Grimshaw, V. Hazlewood, S. Lathrop, D. Lifka, G. D. Peterson, R. Roskies, J. R. Scott, and N. Wilkins- Diehr "XSEDE: Accelerating Scientific Discovery", Computing in Science and Engineering Vol 16 No 5, p 62 (2014).

[2] T. R. Furlani, M. D. Jones, S. M. Gallo, A. E. Bruno, C. Lu, A. Ghadersohi, R. J. Gentner, A. Patra, R. L. DeLeon, G. von Lazewski, L. Wang and A. Zimmerman, "Performance Metrics and Auditing Framework Using Applications Kernels for High Performance Computer Systems", Concurrency and Computation: Practice and Experience. Vol 25, Issue 7, p 918, (2013). DOI:10.1002/cpe.2871 XDMoD: http://xdmod.ccr.buffalo.edu; [March 30, 2014].Open XDMoD: http://xdmod.sourceforge.net/; [March 30, 2014].

[3] T. R. Furlani, B. I. Schneider, M. D. Jones, John Towns, David L. Hart, Steven M. Gallo, Robert L. DeLeon, Charng-Da Lu, A. Ghadersohi, R. J. Gentner, A. K. Patra, G. von Laszewski, F. Wang, J. T. Palmer, and N. Simakov. 2013. "Using XDMoD to facilitate XSEDE operations, planning and analysis", In *Proceedings of the Conference on Extreme Science and Engineering Discovery Environment: Gateway to Discovery* (XSEDE '13). ACM, New York, NY, USA, , Article 46 , 8 pages. DOI=10.1145/2484762.2484763

[4] C. D. Lu, J. Browne, R. L. DeLeon, J. Hammond, W. Barth, T. R. Furlani, S. M. Gallo, M. D. Jones, and A. K. Patra. 2013. "Comprehensive job level resource usage measurement and analysis for XSEDE HPC systems.", In *Proceedings of the Conference on Extreme Science and Engineering Discovery Environment: Gateway to Discovery* (XSEDE '13). ACM, New York, NY, USA, , Article 50 , 8 pages. DOI=10.1145/2484762.2484781

[5] J. C. Browne, R. L. DeLeon, C. D. Lu, M. D. Jones, S. M. Gallo, A. Ghadersohi, A. K. Patra, W. L. Barth, J. Hammond, T. R. Furlani, R. T. McLay, "Enabling Comprehensive Data-Driven System Management for Large Computational Facilities", In *Proceedings of SC13: International Conference for High Performance Computing, Networking, Storage and Analysis* (SC '13). ACM, New York, NY, USA, Article 86, 11 pages. DOI=10.1145/2503210.2503230

[6] J. C. Browne, R. L. DeLeon, A. K. Patra, W. L. Barth, J. Hammond, M. D. Jones, T. R. Furlani, B. I. Schneider, S. M. Gallo, A. Ghadersohi, R. J. Gentner, J. T. Palmer, N. Simakov, M. Innus, A. E. Bruno, J. P. White, C. D. Cornelius, T. Yearke, K. Marcus, G. von Laszewski, F. Wang, "Comprehensive, Open Source Resource Usage Measurement and Analysis for HPC Systems", invited paper accepted to special issue of Concurrency and Computation: Practice and Experience dedicated to XSEDE13 conference, **26**, 2192-2209, (2014).

[7] J.T. Palmer, S.M. Gallo, T. R. Furlani, M. D. Jones, R. L. DeLeon, J. P. White, N. Simakov, A. K. Patra, J. Sperhac, T. Yearke, R. Rathsam, M. Innus, C. D. Cornelius, J. C. Browne, W. L. Barth, R.T. Evans, "Open XDMoD: A Tool for the Comprehensive Management of High Performance Computing Resources", accepted Computing in Science and Engineering, July – August 2015.

[8] N. A. Simakov, J. P. White, R. L. DeLeon, A. Ghadersohi, T. R. Furlani, M. D. Jones, S. M. Gallo, A. K. Patra, "Application Kernels: HPC Resources Performance Monitoring and Variance Analysis", accepted *Concurrency and Computation: Practice and Experience* (2015).

[9] Todd Evans, William Barth, Robert Mclay, James Browney, Leonardo Fiahloy, Robert DeLeon, Thomas Furlani, Steven Gallo, Matthew Jones, Abani Patra, "Comprehensive resource use monitoring for HPC systems with TACC stats" In HUST '14 Proceedings of the First International Workshop on HPC User Support Tools. DOI 10.1109/HUST.2014.7

[10] Hao Zhang, Haihang You, Bilel Hadri, Mark Fahey, "HPC Usage Behavior Analysis and Performance Estimation with Machine Learning Techniques", Proceedings of The 2012 World Congress in Computer Science.

[11] Performance Co-Pilot: http://www.pcp.io

[12] http://sebastien.godard.pagesperso-orange.fr

[13] http://github.com/TACCProjects/tacc_stats

[14] David Meyer, Evgenia Dimitriadou, Kurt Hornik, Andreas Weingessel, Friedrich Leisch, Misc Functions of the Department of Statistics (e1071), TU Wien (R package e1071).

[15] A. Liaw and M. Wiener (2002). Classification and Regression by randomForest. R News 2(3), 18-22.

[16] Kuhn, M. (2015). Classification And REgression Training (caret), R package version 6.0-47, URL http://topepo.github.io/caret/.

[17] X-Ray : A tool for automatic measurement of hardware parameters. / Yotov, Kamen; Pingali, Keshav; Stodghill, Paul. QEST 2005 - Proceedings Second International Conference on the Quantitative Evaluation of SysTems. Vol. 2005 2005. p. 168-177 1595793.

[18] Joseph Emeras, Sébastien Varrettey, Mateusz Guzek, Pascal Bouvry, "Evalix: Classification and Prediction of Job Resource Consumption on HPC Platforms", 19th Workshop on Job Scheduling Stratergies for Parallel Processing (2015).

Table 2. Results from sym classifier confusion matrix**

Application	Application Predicted by sym Classifier
AMBER (1916)	CHARMM (3), CP2K (5), GADGET (1), GROMACS (5), LAMMPS (17), NAMD (16), OPENFOAM (3), PYTHON (5), Q-ESPRESSO (5), VASP (68), WRF (3)
ARPS (1173)	CACTUS (1), FLASH4 (3), LAMMPS (1), NAMD (3), OPENFOAM (1), VASP (18), WRF (5)
CACTUS (1619)	ARPS (1), CHARMM (1), ENZO (5), FD3D (2), FLASH4 (1), LAMMPS (6), NAMD (2), PYTHON (4), Q-ESPRESSO (3), VASP (56), WRF (9)
CHARMM++ (6779)	AMBER (1), CP2K (4), ENZO (1), FLASH4 (6), GROMACS (9), LAMMPS (12), NAMD (53), OPENFOAM (3), Q-ESPRESSO (4), SIESTA (5), VASP (50), WRF (1)
CHARMM (1491)	AMBER (3), CHARMM++ (1), CP2K (1), FLASH4 (2), GADGET (7), GROMACS (3), LAMMPS (22), NAMD (15), OPENFOAM (1), PYTHON (1), Q-ESPRESSO (2), VASP (66), WRF (3)
CP2K (1407)	CACTUS (1), CHARMM++ (3), ENZO (4), FLASH4 (2), GADGET (2), LAMMPS (1), NAMD (5), PYTHON (1), Q-ESPRESSO (4), VASP (75), WRF (2)
ENZO (775)	AMBER (1), ARPS (1), CACTUS (3), CHARMM (1), CP2K (1), GADGET (14), LAMMPS (5), OPENFOAM (4), PYTHON (11), Q-ESPRESSO (11), SIESTA (2), VASP (39), WRF (10)
FD3D (1564),	CHARMM (1), ENZO (1), VASP (12), WRF (1)
FLASH4 (907)	ARPS (2), CACTUS (3), CHARMM++ (6), CHARMM (2), CP2K (4), ENZO (4), FD3D (1), GADGET (5), IFORTDDWN (1), LAMMPS (9), NAMD (4), OPENFOAM (1), PYTHON (6), Q-ESPRESSO (6), VASP (61), WRF (29)
GADGET (594)	AMBER (1), ARPS (2), CACTUS (4), CHARMM (4), CP2K (1), ENZO (19), FLASH4 (3), GROMACS (2), LAMMPS (4), NAMD (3), OPENFOAM (2), PYTHON (4), Q-ESPRESSO (8), SIESTA (1), VASP (53), WRF (12)
GROMACS (7692)	AMBER (12), ARPS (1), CACTUS (3), CHARMM++ (4), CHARMM (3), ENZO (2), FLASH4 (6), GADGET (1), LAMMPS (79), NAMD (39), OPENFOAM (15), PYTHON (1), Q-ESPRESSO (9), SIESTA (2), VASP (35), WRF (4)
IFORTDDWN (835)	VASP (12)
LAMMPS (12093)	AMBER (32), ARPS (4), CACTUS (13), CHARMM++ (8), CHARMM (13), CP2K (1), ENZO (2), FD3D (1), FLASH4 (7), GADGET (5), GROMACS (63), NAMD (53), OPENFOAM (7), PYTHON (18), Q-ESPRESSO (8), SIESTA (4), VASP (145), WRF (3)
NAMD (17058)	AMBER (9), CACTUS (2), CCARMM++ (51), CHARMM (3), ENZO (2), FLASH4 (8), GROMACS (13), LAMMPS (55), OPENFOAM (2), PYTHON (4), Q-ESPRESSO (13), SIESTA (4), VASP (100), WRF (13)
OPENFOAM (1301)	AMBER (5), ARPS (1), CACTUS (9), ENZO (4), FLASH4 (1), GROMACS (20), LAMMPS (26), NAMD (4), PYTHON (9), Q-ESPRESSO (10), VASP (42), WRF (15)
PYTHON (671)	AMBER (1), ARPS (3), CHARMM (2), CP2K (1), ENZO (7), FD3D (2), FLASH4 (11), GADGET (4), GROMACS (1), LAMMPS (15), NAMD (8), OPENFOAM (15), Q-ESPRESSO (20), SIESTA (1), VASP (95), WRF (11)
Q-ESPRESSO (2300)	AMBER (5), ARPS (1), CACTUS (13), CHARMM++ (15), CP2K (3), ENZO (12), FD3D (1), FLASH4 (10), GADGET (11), GROMACS (7), LAMMPS (21), NAMD (14), OPENFOAM (14), PYTHON (19), SIESTA (11), VASP (188), WRF (13)
SIESTA (1029)	CACTUS (1), CHARMM++ (2), CHARMM (1), CP2K (4), GADGET (2), GROMACS (1), LAMMPS (6), NAMD (13), PYTHON (1), Q-ESPRESSO (12), SIESTA (1029), VASP (44)
VASP (32499)	AMBER (16), ARPS (2), CACTUS (17), CHARMM++ (30), CHARMM (24), CP2K (18), ENZO (11), FD3D (7), FLASH4 (13), GADGET (8), GROMACS (7), IFORTDDWN (1), LAMMPS (94), NAMD (58), OPENFOAM (8), PYTHON (20), Q-ESPRESSO (51), SIESTA (7), WRF (27)
WRF (2983)	ARPS (2), CACTUS (5), CHARMM (1), CP2K (2), ENZO (12), FD3D (2), FLASH4 (19), GADGET (7), LAMMPS (8), NAMD (14), OPENFOAM (2), PYTHON (11), Q-ESPRESSO (9), SIESTA (2), VASP (103)

***Column 1 is the application and the number in parenthesis in this column indicates the number of times it was correctly identified by the sym classifier. For example, AMBER was correctly identified 1916 times. Column 2 shows, for the application in the first column, the number of times it was incorrectly identified as each other application. Thus for AMBER in row one, the sym classifier incorrectly identified AMBER as GROMACS 5 times. Zero off diagonal elements are not shown.*