# Using Machine Learning in Trace-driven Energy-Aware Simulations of High-Throughput Computing Systems

A.Stephen McGough,
Noura Al Moubayed
School of Engineering and Computing Sciences
Durham University
Durham, UK
stephen.mcgough@durham.ac.uk
noura.al-moubayed@durham.ac.uk

Matthew Forshaw
School of Computing Science
Newcastle University
Newcastle, UK
matthew.forshaw@newcastle.ac.uk

## ABSTRACT

When performing a trace-driven simulation of a High Throughput Computing system we are limited to the knowledge which should be available to the system at the current point within the simulation. However, the trace-log contains information we would not be privy to during the simulation. Through the use of Machine Learning we can extract the latent patterns within the trace-log allowing us to accurately predict characteristics of tasks based only on the information we would know. These characteristics will allow us to make better decisions within simulations allowing us to derive better policies for saving energy.

We demonstrate that we can accurately predict (up-to 99% accuracy), using oversampling and deep learning, those tasks which will complete while at the same time provide accurate predictions for the task execution time and memory footprint using Random Forest Regression.

## Keywords

Trace-Driven; Simulation; Machine Learning

## 1. INTRODUCTION

Computer simulation is a powerful tool for understanding how systems in the real world interact, and the impact that changes to these systems may have. As we become more aware, and concerned, about energy consumption of these systems simulation can aid understanding and help in the identification of better systems and their management.

High-Throughput computing, where thousands of tasks can be executed over a distributed collection of computers, expends significant energy through code execution. Through efficient management of these systems we can significantly reduce the overall energy consumption. Management changes could include selecting more energy-efficient computers when working in a heterogeneous environment [28], selection of computers least susceptible to preemption by their primary users [20], or removing tasks which are failing to complete (due to misconfiguration or broken code) [22].

When performing such simulations we require an input stream of tasks which represent the users' interaction with the system. One approach here is to capture the real stream of events which are produced from the live system – this is often referred to as a trace-log and leads to a trace-driven simulation. This, however, has a number of drawbacks. The system being simulated may not exist yet; there may be a desire to modify the characteristics of the input as one of the changes you make with the simulation; or the time required to capture the trace-log from the real system may be too long. This leads to trace-logs having significant value among researchers. However, exchanging such trace-logs may not be easy as they can contain confidential information.

Alternatively, a synthetic trace-log may be produced from probability models. These models may be derived from statistical analysis of real input streams or probability models chosen for their similarity to known usage patterns. If no existing system exists then the latter case is the only choice. In either case such probability models normally lack the fidelity exhibited within real trace-logs. In prior work [19] we have generated synthetic trace-logs derived from statistical analysis of the HTCondor users at Newcastle University in order to upscale the system load. Although this generated data which matched the statistical characteristics of the original trace-log the newly generated data failed to exhibit the finer-grained characteristics of the original trace data.

In this work we propose a different approach in which we exploit the ability for Machine Learning techniques to over-sample the original trace-logs in order to generate additional data. Using this approach, we can both extend the timeframe covered by our original trace-log but also increase (or decrease) the number of system users. As the Machine Learning approach builds up a fine-grain model from the initial trace-log this has the potential to capture the fidelity apparent within the original trace-log but lost when a probability model is used. This approach has two clear advantages, new synthetic data can be generated for an arbitrarily length of time (and density) while capturing the same fidelity as the original trace-log but also as the data is still synthetically generated it can be safely exchanged with other researchers without fear of breaking confidentiality.

Once we are able to produce synthetic trace-logs of high enough fidelity there are still a number of problems which



*ICPE '17 Companion, April 22 - 26, 2017, L'Aquila, Italy*



ISBN 978-1-4503-4899-7/17/04…$15.00

DOI: http://dx.doi.org/10.1145/3053600.3053612

need to be overcome. Solving these problems for the simulation would also allow us to solve the corresponding problem in the real-world and are often the focus of the investigations which lead us to developing simulations in the first place. These problems include:

***1) Estimation of task execution time:*** In order to identify the best computer to run a task on, or indeed whether to run the task in the first place, we need an estimation of task execution time. Although the actual execution time is held within the trace-log this is not information we are privy to at the time of task submission. Prior work [5, 24, 26] has widely criticised the quality of user provided estimates. Instead we propose the use of Machine Learning techniques to estimate these times more accurately based on a model of both the individual user and other similar users.

***2) Identification of Miscreant tasks:*** In prior work [22] we have explored the problem of miscreant tasks – tasks which in a shared use system, where task can be evicted from resources due to higher priority interactive users logging in, are repeatedly started on computers just to lose control of that computer. A miscreant task may be one where the code will never complete due to bugs or misconfiguration. Alternatively it could be a task which has been unlucky in its selection of computers to run on. We see this as a classification exercise where we can attempt to identify those tasks which are bad thus removing them quickly from the system so that they waste less energy.

***3) Identification of computers which are unlikely have interactive users logging in during the time-scale of a task's execution:*** If we can select a computer where no interactive user will log in before the task completes then we remove the need for the task to be evicted and thence restarted on another computer – thus wasting significant energy. We have previously demonstrated that Reinforcement Learning can be used to increase the chance of a task being deployed to an appropriate computer [20]. Here we seek to go further by building a model for when interactive users log in and log out. Through this model we can better predict the most appropriate computer to deploy tasks to.

***4) Estimation of task memory footprint:*** Although users can specify memory requirements for their tasks these estimates are often missing or significantly greater than that required. Instead we can develop a regression approach to better estimate this value allowing for better selection of computers on which to run the task.

We discuss related work in Section 2 before performing an analysis of the data from a HTC system in Section 3. We justify our approach of how using Machine Learning can be applied to a HTC system through preliminary results in Section 4. Finally we conclude the paper in Section 5.

## 2. RELATED WORK

Machine learning techniques have been applied to the operational management of High-Throughput and High- Performance computing systems. These approaches incorporate prior knowledge of system behaviours to optimise policy decisions made within the system – e.g. resource allocation, scheduling, migration, and power management – and have been shown to be able to adapt to temporal variations in resource availability and performance, and offered workload.

A number of approaches consider infrastructure-level management. Gao [12] apply an ensemble of deep neural networks using DeepMind, to model the operation of Google data centres and predict Power Usage Effectiveness (PUE), yielding an error rate of 0.4% for a typical PUE of 1.1. Machine Learning approaches receive further attention in failure modelling within HPC systems; e.g. in the prediction of memory issues [3] and hard disk failure [13].

Berral *et al.* [8] use machine learning to predict energy consumption of virtual machines, using these predictions to inform resource provisioning and scheduling decisions including migration and consolidation. Further, these approaches have been applied in the context of systems comprising multiple, geographically distributed datacentres [7].

Curtis-Maury *et al.* [11] use artificial neural networks (ANN) to determine target levels of concurrency on multicore processors optimising performance and energy consumption.

ML approaches have been applied to predict the characteristics of tasks. Matsunaga [18] explore the spatiotemporal utilisation of resources by mainstream HPC applications, including BLAST. Rodrigues *et al* [25] evaluate the application of a number of machine learning techniques to predict task memory requirements in an HPC setting. Insights gained through these approaches are used to inform decisions made by the LSF scheduler. Techniques include Support Vector Machines (SVMs), Random Forests, Multilayer Perceptrons (MLPs) and $k$ Nearest Neighbours (KNN).

Barrett *et al* [6] consider a Q-learning approach to the auto-scaling of cloud applications deployment. Seeking to address the dimensionality issues associated with Reinforcement Learning approaches by adopting a hybrid approach.

## 3. ANALYSIS OF A HTC SYSTEM

A High Throughput Computing (HTC) system comprises of a large collection of computers, which may not be owned by the manager of the HTC system. Often these computers are provided on a voluntary basis in which the computer owners expect the HTC system to relinquish use at any time when the owner wishes to use their computer(s) – often referred to as task eviction. Evicted tasks are normally restarted a different computer. Two commonly used HTC systems are BOINC [4] and HTCondor [27].

High Throughput Computing systems collect and maintain meta-data about the tasks which have been submitted for execution. For HTCondor this is in the form of a ClassAd which can be inspected through condor_history or Quill [16]. The set of elements stored within this ClassAd are described within the HTCondor manual [29]. All absolute times are stored in seconds relative to the UNIX epoch.

The stored meta-data contains such information as when the task was submitted, which resource the task is running on and when the task completed. Some of this meta-data is known as the task is submitted (name of executable, time of submission, command line arguments, etc...) whilst further meta-data is only known while the task is within the system (current execution time, number of invocations of the task due to prior evictions, etc...) with the last meta-data only available on task completion (execution time, size of data files returned, etc...). This can result in a large number of artefacts held in the meta-data – e.g. the HTCondor system at Newcastle University contains 102 different artefacts.

In general we wish to be able to predict those artefacts which will only become known when a task completes from the other artefacts which we can see before task completion.

| Artefact Name | Description |
|---|---|
| ClusterID | A "Cluster" is a group of tasks |
| ProcID | The ID of a Task within a Cluster |
| QDate | Submission time (secs since Unix epoch) |
| Owner | person submitting the task |
| User | equivalent to Owner |
| Command | Command/executable run for the task |
| Args | arguments passed to the executable |

**Table 1: Artefacts available before task execution**

## 3.1 Newcastle HTCondor Data

Our analysis here is based on the HTCondor set-up at Newcastle University which was deployed in October 2005 and replaced in 2011; this gives us an archive of five years. The vast majority computers were in open access clusters around the university providing ∼2000 computers running Windows XP. There is a fair degree of heterogeneity between clusters. A number of staff added their own computers and a small number (∼ 100) of Linux computers were available. The HTCondor system has one central manager and one main submission machine though some projects had their own submission machine. In the rest of this section we analyse the history logs from the main submission machine and use this as motivation for the subsequent work.

Out of the 102 artefacts within the Newcastle HTCondor logs we can identify seven artefacts which are both available when the task is submitted and of relevance to the work here. These are depicted in Table 1.

After the task completion there are four artefacts which are added containing information which would be relevant for running a simulation. These are depicted in Table 2. In order to perform simulations we require the task duration which is not clearly defined within the set of artefacts. However, it can be derived from the four artefacts as:

$$\begin{aligned} Duration = \quad & EnteredCurrentStatus \\ & -JobCurrentStartDate \end{aligned} \tag{1}$$

if the task was successful (JobStatus=4). This may be an overestimate of the actual execution time if HTCondor placed the task into a suspend state at some point. However, suspensions are rare and normally followed by an eviction, which would prevent this from being a successful execution. This does not work when check pointing and migration is used. If however, the task failed to complete and was terminated (JobStatus=3) then we can only assume that the task would run for at least as long as it was in the system:

$$Duration = EnteredCurrentStatus - QDate.$$

We can also compute how much time a successful task wasted in the system due to incomplete executions:

$$\begin{aligned} Successful\ task \quad &= \quad RemoteWallClockTime \\ wasted\ time \quad & \quad -Successfulexecutiontime. \end{aligned}$$

There were a total of 642,298 task submissions between the 17th October 2005 and the 31st December 2010. The breakdown of these tasks is show in Table 3. These figures seem promising for energy efficient use of the HTCondor system until we look at the total times for these tasks.

The total compute time consumed by HTCondor was 1, 610, 913, 772 seconds (just over 51 years, 29 days) this is shown in Table 4. However, the time consumed by tasks that were eventually terminated was just over 51%. Only 35% of time consumed by HTCondor went to successfully completing tasks the remaining 15% went on executions that were evicted. In effect for every 1 second of useful execution time required from HTCondor 2.8 seconds of actual computing time was needed. Even if we negate tasks removed by the user this still leaves 1.4 seconds HTCondor time for every successful second of execution. Clearly not energy efficient.

We now justify the adoption of a ML approach to estimate a number of the initially unknown artefacts.

***1) Task Execution time:*** Figure 1 shows the profile of execution times for a single user within the HTCondor log. It can be seen that this profile does not conform to a simple statistical model. Prior attempts to apply such models to this data has yielded poor results [21]. A regression approach could be applied to this data taking into account such factors as the time of day that the task was submitted, the executable run and the user submitting the task.

***2) If the task is good:*** Prior work has shown that monitoring a miscreant task allows us to identify bad tasks. However, this is at the expense of execution runs wasting energy. If instead we could first classify tasks – even if this were only to say "high chance of bad task" this would allow us to ter-

| Artefact Name | Description |
|---|---|
| JobCurrentStartDate | Time the task started executing (measured in Seconds since Unix epoch). If a task has run multiple times, this is the measurement for the latest run. |
| EnteredCurrentStatus | epoch time when the task entered its current status (listed in JobStatus) |
| JobStatus | 3 is a failed task, 4 is a successful task |
| ImageSize | Memory usage of the task when it ran |

**Table 2: Artefacts available after task execution**

| | Number | Percentage |
|---|---|---|
| Total number of tasks | 642,298 | 100% |
| - Which completed | 469,184 | 73% |
| - - Without wasted time | 415,807 | 65% |
| - - With wasted time | 53,377 | 8% |
| - Which were removed | 173,114 | 27% |
| - - Removed before execution | 164,059 | 26% |
| - - Removed after some execution | 9,055 | 1.4% |

**Table 3: Breakdown of HTCondor tasks**

| Total time | Seconds | Time | Percent |
|---|---|---|---|
| used by HTCondor | 1610913772 | 51y 29d | 100% |
| - successful execution | 559065399 | 17y 265 d | 35% |
| Wasted | 1051848373 | 33y 129d | 65% |
| - tasks that completed | 228979785 | 7y 95d | 14% |
| - for removed tasks | 822868588 | 26y 33d | 51% |

**Table 4: Breakdown of time used by HTCondor**

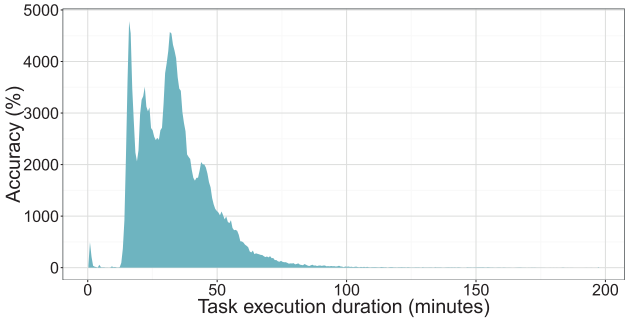


**Figure 1: Task execution profile for a single user**

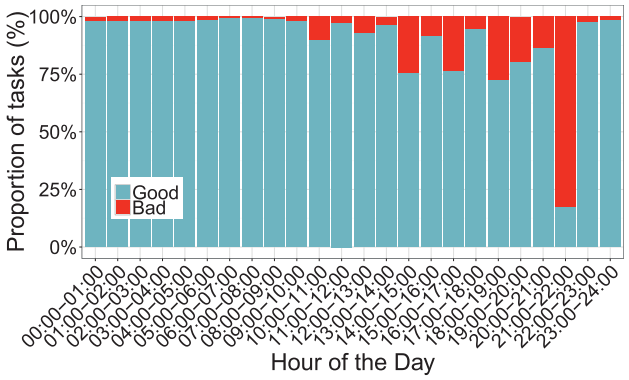


**Figure 2: Good and Bad tasks by hour of submission**

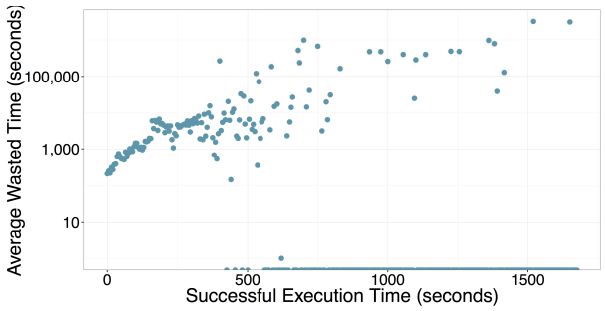


**Figure 3: Wasted time against successful execution**

minate miscreant executions earlier. Factors which could influence the chances of a task being bad could include the user, executable and the time of day when the task was submitted. Figure 2 shows the proportion of good and bad task submissions for each hour of the day during 2010. It can be seen that the lowest rate of bad task submissions is between 22:00 and 09:00. This is most likely due to the fact that tasks submitted overnight will be automatic submissions of known good work. Whilst during "normal" hours the users will be developing scripts and trying them out.

***3) Expected wasted time per task:*** Figure 3 illustrates the relationship between wasted time for a task in relation to the actual execution time. This correlation is not unexpected as the longer running tasks will have a higher probability of being evicted as they spend longer running. If we know an estimate for the wasted time for a given task we can monitor how much time it has indeed been wasting and thus know whether it is likely to be a bad task.

# 4. PRELIMINARY RESULTS

## 4.1 Data Pre-processing:

Before applying any machine learning approaches to analyse the data, a pre-processing step is necessary to reshape the data into a format that is easier to learn from. QDate was converted to four different features (month of year, day of month, day of week, hour of day). JobCurrentStartDate and EnteredCurrentStatus were substituted by

$$Wait = JobCurrentStartDate - Qdate, \quad (2)$$

and

$$JobDuration = EnteredCurrentStatus - JobCurrentStartDate$$

The owner ID and command features were converted to unique numeric features. The Args feature was removed as it can be arbitrary and without much useful information from a scheduling point of view. JobStatus, ImageSize, ClusterID, and ProcID were kept as described above.

Tasks that were killed before receiving any execution were removed from the dataset (128,069 in total). This results in 433,782 samples each with 12 features to be used for analysis.

## 4.2 Analysis

We tackle the analysis of the data from four perspectives.

***I. Prediction of JobStatus:*** Predicting the possibility of a task to fail could be very beneficial in eliminating potentially time wasting tasks or giving them a lower priority in the queue. The challenge of building a machine learning model here is the imbalance nature of the training data, i.e. the number of available instances of one class is much bigger than the second skewing the performance of the machine learning model towards the majority class. In this data set the number of failed tasks is 4189, against the number of successful tasks: 429,593, with a fail to success ratio of 0.97%. This can be a real challenge for any classification approach [10]. In [1] we used the concept of over-sampling the minority class to balance the performance of the classifier for both classes. Here we used the same approach and employed Synthetic Minority Over-sampling Technique (SMOTE) [9] as the over-sampling algorithm.

As an input feature to the classifier we utilised nine of the 12 pre-processed features. ImageSize and JobDuration were excluded as they can only be available after the task is completed. JobStatus is used as the label.

Figure 4 shows the results of applying two standard machine learning approaches: Logistic Regression (LR) [15], and Linear Discriminant Analysis (LDA) [17] on the dataset without over-sampling (columns 1 and 3) and with over-sampling of the minority class using SMOTE. The results clearly show the potential for detecting the failed tasks. It also shows the critical role over-sampling is playing in enhancing the accuracy of the minority class (failed tasks) without significantly reducing the accuracy of detecting the successful tasks.

***Prediction of ImageSize:*** Predicting the memory usage of the task can be beneficial in optimising the usage of the system's resources. This is a regression problem as the ImageSize is not a categorical value. JobStatus and JobDuration were excluded from the feature set as in the previous analysis. A random forest regressor [14] is used to model the data. The accuracy of the model is measured using

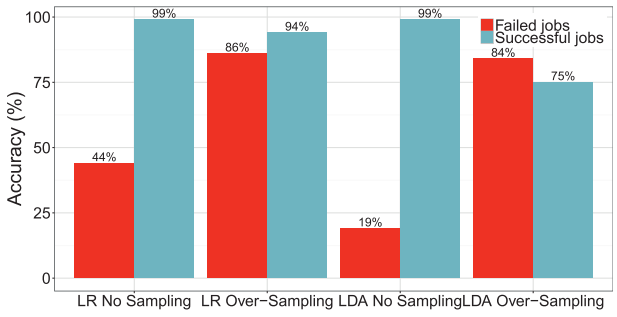


**Figure 4: Prediction of the JobStatus. Red indicates the accuracy of predicting the failure of a task. Blue represents the accuracy of predicting the success of the task. Four classification approaches are tested: Logistic Regression (LR) with sampling of the minority class (failed tasks) using SMOTE, and without any sampling, Linear Discriminant Analysis (LDA) with and without sampling.**

normalised coefficient of determination $r^2$. $r^2$ measures the variance of the observed ImageSize data explained by the predicted data [23]. A value of 1 is the best possible score. Our model scores $r^2 = 0.961$ (Figure 5) which means we can predict with high confidence the memory footprint of a task.

***Prediction of Task Duration:*** Following the prediction of ImageSize, the prediction of JobDuration is very crucial for the schedular to plan the available resources. A similar random forest regression model is built after excluding JobStatus and ImageSize from the feature set. Figure 6 shows the results of this model with $r^2 = 0.974$. This can be used as a factor to decide the priority of the time consuming tasks in the queue or to schedule such tasks at times with less or no load in the system.

***Outlier Detection:*** One of the potentially useful information is the ability to detect uncommon behaviour in the system. This could be, for example, a task that is taking a very long time to process. In order to detect these outliers a model of the "common" usage of the system is necessary. This model is built based on the previous use cases. Here we used our outlier detection approach [2] based on an unsupervised deep learning model: Stacked Denoising Autoencoders (SDA) [30]. The method builds an expanding deep neural network that tries to reconstruct the input features and measure the difference between the reconstructed sample against the actual input. Data samples which result in high reconstruction error are considered outliers of the SDA model. Figure 7 demonstrate the output of the outlier detection system on all the tasks submitted by all 15 users. The highlighted samples are examples of outliers. By further analysis of these particular tasks, we find that they consume over 38 times the average memory and around 1000 times the average task duration.

## 5. CONCLUSIONS AND FUTURE WORK

In this paper we have provided motivation for the use of Machine Learning for the identification of trends and patterns within trace-logs for High Throughput Computing systems. Our argument being that there are latent patterns within these trace-logs which if exposed will allow for better resource usage leading to saved energy.

We go further to show that machine learning techniques

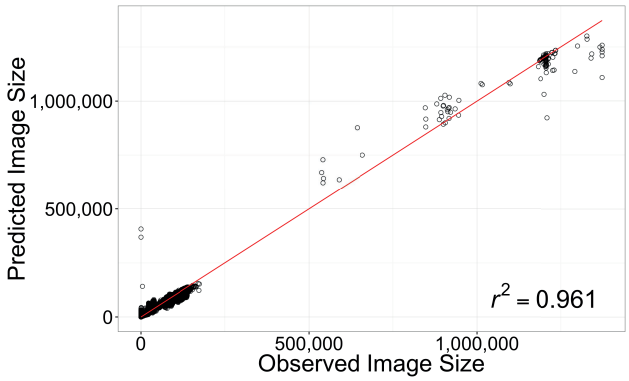


**Figure 5: Using a random forest regression model to predict the ImageSize. Red line is the equilibrium.**

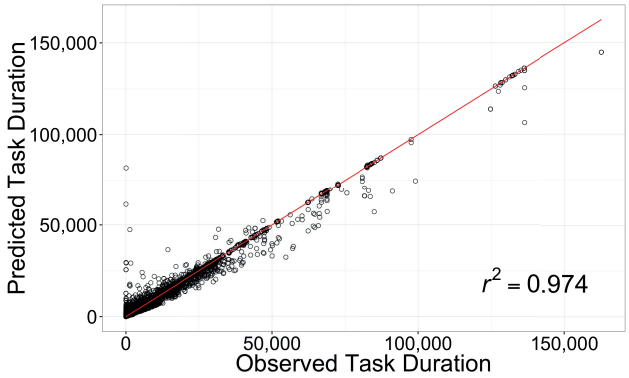


**Figure 6: Using a random forest regression model to predict Duration. Red line is the equilibrium.**

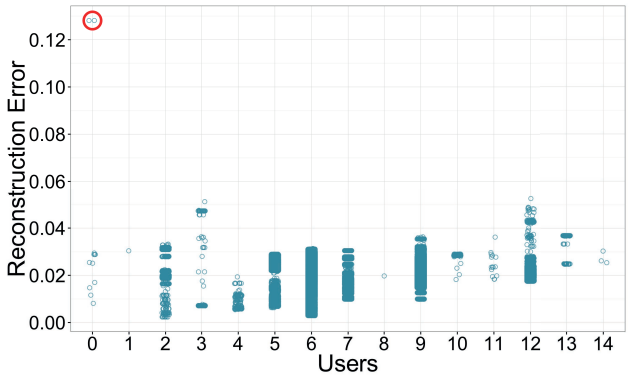


**Figure 7: Reconstruction error of the users data as generated by a stacked denoising auto-encoder.**

such as SMOTE and Linear Discriminant Analysis can be combined to accurately predict if tasks will complete with an accuracy up to 99% even in the presence of imbalanced training data. We further demonstrate that using Random Forest Regression can accurately predict both task image size and duration. We finish by showing that Stacked denoising Autoencoders can be used to predict those tasks which are likely to be problematic within the system.

Our ongoing work seeks to evaluate the operational benefits these approaches yield when applied within a production environment.

# 6. REFERENCES


[1] N. Al Moubayed, B. Awwad Shiekh Hasan, and A. S. McGough. Enhanced detection of movement onset in eeg through deep oversampling. In *International Joint Conference on Neural Networks (IJCNN 2017)*. IEEE, 2017.

[2] N. Al Moubayed, T. Breckon, P. Matthews, and A. S. McGough. Sms spam filtering using probabilistic topic modelling and stacked denoising autoencoder. In *International Conference on Artificial Neural Networks*, pages 423–430. Springer, 2016.

[3] J. Alonso, J. Torres, J. L. Berral, and R. Gavalda. Adaptive on-line software aging prediction based on machine learning. In *DSN*, pages 507–516. IEEE, 2010.

[4] D. P. Anderson. Boinc: A system for public-resource computing and storage. In *Grid Computing, 2004*, pages 4–10. IEEE, 2004.

[5] C. Bailey Lee, Y. Schwartzman, J. Hardy, and A. Snavely. Are user runtime estimates inherently inaccurate? In D. Feitelson, L. Rudolph, and U. Schwiegelshohn, editors, *Job Scheduling Strategies for Parallel Processing*, volume 3277 of *LNCS*, pages 253–263. Springer Berlin Heidelberg, 2005.

[6] E. Barrett, E. Howley, and J. Duggan. Applying reinforcement learning towards automating resource allocation and application scalability in the cloud. *CCPE*, 25(12):1656–1674, 2013.

[7] J. L. Berral, R. Gavalda, and J. Torres. Power-aware multi-data center management using machine learning. In *ICPP*, pages 858–867. IEEE, 2013.

[8] J. L. Berral, Í. Goiri, R. Nou, F. Julià, J. Guitart, R. Gavaldà, and J. Torres. Towards energy-aware scheduling in data centers using machine learning. In *ACM e-Energy*, pages 215–224. ACM, 2010.

[9] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer. Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, 16:321–357, 2002.

[10] N. V. Chawla, N. Japkowicz, and A. Kotcz. Editorial: special issue on learning from imbalanced data sets. *ACM Sigkdd Explorations Newsletter*, 6(1):1–6, 2004.

[11] M. Curtis-Maury, K. Singh, S. A. McKee, F. Blagojevic, D. S. Nikolopoulos, B. R. De Supinski, and M. Schulz. Identifying energy-efficient concurrency levels using machine learning. In *Cluster Computing, 2007 IEEE International Conference on*, pages 488–495. IEEE, 2007.

[12] J. Gao. Machine Learning Applications for Data Center Optimization. *Google White Paper*, 2014.

[13] G. Hamerly, C. Elkan, et al. Bayesian approaches to failure prediction for disk drives. In *ICML*, 2001.

[14] T. K. Ho. Random decision forests. In *Document Analysis and Recognition, 1995., Proceedings of the Third International Conference on*, volume 1, pages 278–282. IEEE, 1995.

[15] D. W. Hosmer Jr, S. Lemeshow, and R. X. Sturdivant. *Applied logistic regression*, volume 398. John Wiley & Sons, 2013.

[16] J. Huang, A. Kini, E. Paulson, C. Reilly, E. Robinson, S. Shankar, L. Shrinivas, D. Dewitt, and J. Naughton. An overview of Quill: A passive operational data logging system for Condor. https://www.cs.wisc.edu/condordb, 2007.

[17] A. J. Izenman. Linear discriminant analysis. In *Modern multivariate statistical techniques*, pages 237–280. Springer, 2013.

[18] A. Matsunaga and J. A. Fortes. On the use of machine learning to predict the time and resources consumed by applications. In *IEEE/ACM CCGRID*, 2010.

[19] A. McGough, C. Gerrard, J. Noble, P. Robinson, and S. Wheater. Analysis of power-saving techniques over a large multi-use cluster. In *Dependable, Autonomic and Secure Computing (DASC), 2011 IEEE Ninth International Conference on*, pages 364–371, Dec 2011.

[20] A. S. McGough and M. Forshaw. Reduction of wasted energy in a volunteer computing system through reinforcement learning. *Sustainable Computing: Informatics and Systems*, 4(4):262 – 275, 2014. Special Issue on Energy Aware Resource Management and Scheduling (EARMS).

[21] A. S. McGough, M. Forshaw, C. Gerrard, P. Robinson, and S. Wheater. Analysis of power-saving techniques over a large multi-use cluster with variable workload. *Concurrency and Computation: Practice and Experience*, 25(18):2501–2522, 2013.

[22] A. S. McGough, M. Forshaw, C. Gerrard, and S. Wheater. Reducing the number of miscreant tasks executions in a multi-use cluster. In *Cloud and Green Computing (CGC), 2012 Second International Conference on*, pages 296–303, Nov 2012.

[23] N. J. Nagelkerke. A note on a general definition of the coefficient of determination. *Biometrika*, 78(3):691–692, 1991.

[24] S. Niu, J. Zhai, X. Ma, M. Liu, Y. Zhai, W. Chen, and W. Zheng. Employing checkpoint to improve job scheduling in large-scale systems. In *Job Scheduling Strategies for Parallel Processing*, pages 36–55. Springer, 2013.

[25] E. R. Rodrigues, R. L. Cunha, M. A. Netto, and M. Spriggs. Helping HPC users specify job memory requirements via machine learning. In *Proceedings of the Third International Workshop on HPC User Support Tools*, pages 6–13. IEEE Press, 2016.

[26] S. Srinivasan, R. Kettimuthu, V. Subramani, and P. Sadayappan. Characterization of backfilling strategies for parallel job scheduling. In *Parallel Processing Workshops, 2002. Proceedings. International Conference on*, pages 514–519. IEEE, 2002.

[27] T. Tannenbaum, D. Wright, K. Miller, and M. Livny. Condor: a distributed job scheduler. In *Beowulf cluster computing with Linux*, pages 307–350. MIT press, 2001.

[28] G. Terzopoulos and H. D. Karatza. Power-aware bag-of-tasks scheduling on heterogeneous platforms. *Cluster Computing*, 19(2):615–631, 2016.

[29] The Condor Team. Condor manual. http://www.cs.wisc.edu/condor/manual/, Oct 2010. University of Wisconsin.

[30] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of Machine Learning Research*, 11(Dec):3371–3408, 2010.