Learning Job Run-times in HPC Systems

Valentin Reis
Supervised by: Eric Gaussier & Denis Trystram.

Abstract

In many HPC infrastructures, the descriptions of the tasks to be executed are subject to high uncertainties. In this work, we show that users are unreliable when estimating the run time of their jobs, and we look for alternatives. Predictive techniques for inferring the run time of jobs from either their description or system history are surveyed, and a Machine Learning technique (Random Forests) is used to combine these approaches. Experiments emphasize that the predictions obtained with our approach have correct properties with respect to relevant metrics, and outperform state of the art approaches.

1 Introduction

High Performance Computing (HPC) systems are more and more complex. There is a need for studies and tools for efficient use, at the frontier of two different fields: Scheduling and allocation of jobs, and Systems and Middleware. We outline two of the main difficulties that Resource Management Software (RMS) in this field have to face.

First, the ephemeral nature and broad range of existing architectures of these systems make it difficult to develop and apply of theoretical results. New schemes for distributing resources frequently appear in the industry. Current systems can have complex network topologies and memory/hard drive sharing architectures. Moreover, the topology of those systems can now change on a hourly to monthly basis, since the hardware of distributed systems can be reconfigured or extended continually. Finding scheduling and resource management strategies that can deal with complex systems and adapt to their evolutions is a big challenge.

In addition, the data (i.e., the characteristics of the tasks to be executed) that the RMS are provided with has many peculiarities. The nature of the information that the users of the system provide is very often loose: only upper and/or lower bounds on numerical quantities may be provided. For instance, and this will be the focus of this paper, the run time of a given job on a specific system is seldom known in advance, but many cluster management software ask the users for an upper bound on this quantity, promising a sooner and better allocation.

As a consequence of these difficulties most free, open-source and commercial RMS use simple heuristics that can provide bounds on their performance and/or guarantee a few functional properties. An example of such a heuristic is the First Come First Serve (FCFS) [Schwiegelshohn and Yahyapour, 1998] policy to schedule parallel jobs on a homogeneous cluster of machines. This policy starts jobs as soon as possible, in the exact order they were submitted. Among other properties, such as robustness to lack of information about the amount of time jobs will run on the system, this strategy guarantees the avoidance of starvation.

1.1 Research Direction

The general direction followed within this research project is to deal with the input data of the RMS. The treatment of this data seems to us separable enough from the actual scheduling problem so that working towards this objective may be rewarding. Methods to query information from the users will be studied. We will instead rely on existing logs from HPC systems and seek to apply Machine Learning (ML) techniques in order to reduce the uncertainty of (and extract information from) the data. The goal is to present the input data in the most valuable way possible to a scheduling algorithm. The question of how to use this data in a scheduling algorithm will not be addressed beyond providing references to appropriate scheduling algorithms.

1.2 Job run-times

Most HPC resource management software (including SLURM [Yoo *et al.*, 2003], TORQUE [Staples, 2006], and OAR [Capit *et al.*, 2005]) do ask information about jobs to users, such as topological requests in terms of processing units and memory, name of the executable, miscellaneous functional requirements and, last but not least, the expected run time of the job with respect to its hardware requirements. This user-provided estimate of the run time of a job on a specific system will be referred as **required-time** in the rest of this paper. Most of these software use the required-time of a job as an upper bound on its run time, and kill it if this required-time is violated [1]. As a consequence, users overesti-

[1] See [Yoo *et al.*, 2003] and [Capit *et al.*, 2005]

mate this value when they to provide it. Section 2 will present a statistical analysis pertaining to this relationship.

The actual run time of a job with respect to a given topology is of great interest, as scheduling policies are highly dependent on this information to provide good solutions [Dutot *et al.*, 2004]. We will refer to this quantity as the **run-time** of a job. It must be stated clearly that in the context of topological heterogeneity, the run-time of a job is only defined with respect to a specific processing environment to which it might be affected. The parameters determining the run-time can include, and is not limited to, network topology of the processing units, availability of shared memory, message passing costs, and the operating system(s) supporting the computations.

1.3 Problem Statement

In this paper, the question of refining the data is reduced to a single variable, the run-time. The problem statement that is dealt with in this paper is the following.

Given a specific homogeneous HPC Cluster with negligible communication costs, how to best predict the value of the run-time of a job?

The choice of dealing with homogeneous distributed machines without communication costs in a first approach has the interesting property of separating the data treatment from the scheduling and interaction with the system. In this case, the run-time becomes an intrinsic attribute of a job. On the contrary to the input data, which are subject to peculiarities of the various systems and software, the run-time is consistently present as a simple field in workload logs of HPC systems, such as those available at the Parallel Workload Archive [Feitelson *et al.*, 2012].

This problem statement implies a latent question: How to communicate the prediction to the rest of the system (e.g., single-value, probability density, confidence factor)? Alternatives will be discussed but ultimately, the focus will be on single-valued predictions. As mentioned previously, our approach is to use machine learning techniques. We will be inferring run-time from the job characteristics by learning from system logs, and since this is a value in $[0, +\infty[$, this is a supervised learning problem, namely regression.

We will be careful to only learn our models on logs from homogeneous systems, or homogeneous subsets of systems that are sizeable enough to learn from. As for downplaying the impact of communication costs, we will further restrict our work to machines that do not possess complex topology or distribute computing nodes across more than a handful of routers. In essence, this paper targets large clusters, supercomputers, GPU farms and mainframe clusters.

This paper is organised in the following manner: Section 2 provides motivation for predicting the run-time. Section 3 surveys existing prediction techniques, and Section 4 introduces our approach. Preliminary results are introduced in Section 5 and we conclude on our findings in Section 6.

2 Motivation for Prediction

2.1 Importance of run-time

Once again, we emphasise the role of the run-time in scheduling. Virtually all results from scheduling theory use the 'clairvoyant' model [Dutot *et al.*, 2004], where run-times of jobs with respect to all possible affectations on the system are known in advance. Intuitively, in order to use this extensive theoretical body, we would need to reduce uncertainty in this variable. It has even been shown [Tsafrir *et al.*, 2007] to some extent that in the classical approach (using policies such as FCFS, which are robust to this uncertainty), there is added value when this variable is refined. In all cases, reducing uncertainty in run-time is critical to the success of the approach used to schedule jobs.

As mentioned before, existing solutions use much simpler heuristics. We will now focus on a particular system in order show why a simple scheduling heuristic is applied.

2.2 Required-time vs Run-time on a real system

The following study is conducted on a log containing 20 months worth of data from the CURIE supercomputer operated by the French government-funded research organization CEA (Commissariat à l'Énergie Atomique). It contains more than 300,000 jobs, submitted from February 2011 to October 2012 by 900 users in the 'cleaned' version, which is referenced in the archive as *CEA-Curie-2011-2.1-cln.swf.* The log has several homogeneous CPU partitions and CPU+GPU partitions, however in each partition all nodes are identical. Jobs are allocated to partitions using user preference. The system is managed using the SLURM RMS. Figure 1 shows the marginal distributions of required-times and run-times on this system. The marginal distributions are already revealing: almost half of the required-times are in the 24 hour bin.

The reason behind this highly filled bin is that 24h is both the maximal value and the default one. On this system, users may choose not to provide an estimate for their job's run-times, in which case the maximum value is used by the scheduler. We already see here that the asking for the required-time is not a suitable option. Further looking into the relationship between required-time and run-time, Figure 2 shows how the $\frac{\text{Run-time}}{\text{Required-time}}$ ratio is distributed.

This histogram indicates that a majority of users either: have very little idea about the expected run-time of their jobs, or overestimate very strongly its value on purpose. Sophisticated scheduling methods are not applied: under such uncertainty in the run-time their performance is equivalent to the simple heuristic that is applied in practice, namely FCFS with Backfilling [Mu'alem and Feitelson, 2001].

3 State of the art in run-time prediction

As mentioned previously, the latent question when predicting the run-time is how to provide the information to the scheduling algorithm. This section presents alternatives and state of the art methods for both the single value and distribution prediction cases.

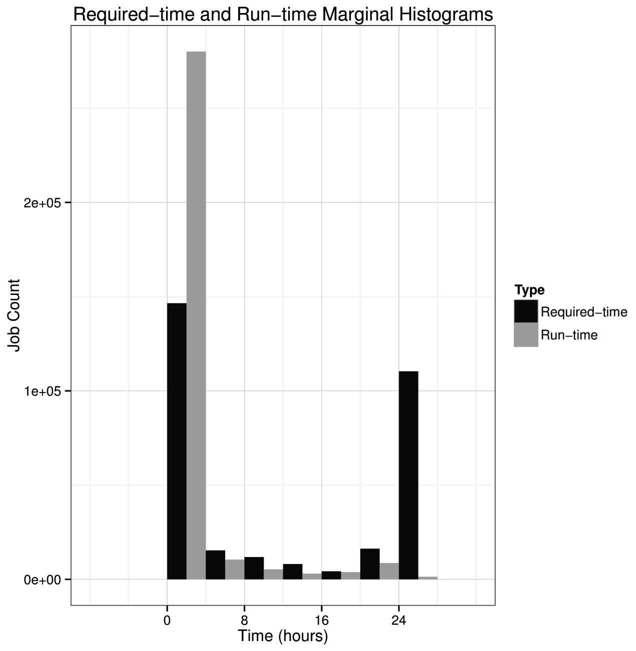

Figure 1: Marginal Histograms of the required-time and run-time of all jobs from the CEA CURIE log.

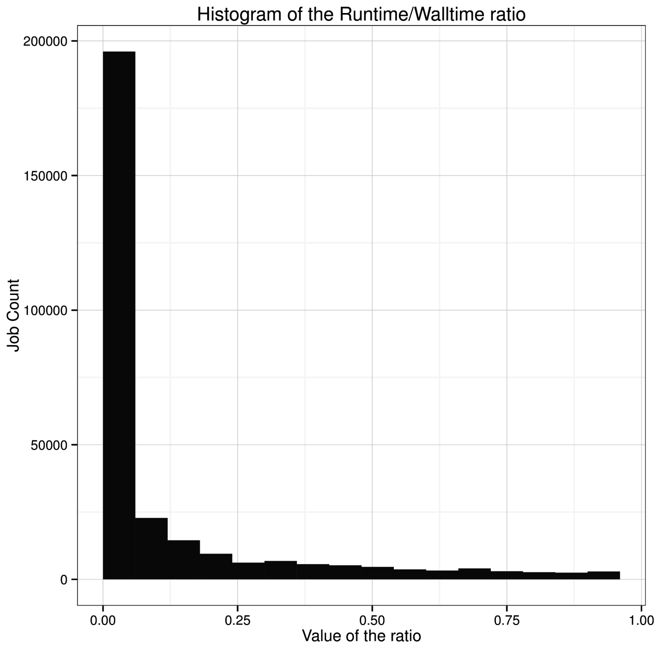

Figure 2: Histogram of the $\frac{\text{Run-time}}{\text{Required-time}}$ ratio in the CURIE log.

3.1 Predicting a value

Single-value prediction has first been attempted [Gibbons, 1997] by partitioning jobs in a predefined partitioning of their feature space and averaging values in each partition to provide an estimate. In this method, the partitioning has to be provided by the RMS or the system administrator. This method assumes the job characteristics to be identically distributed and independent. It does not make use of dependency between successive jobs. Moreover, the binning has to be obtained trough careful statistical analysis of the specific system and population.

A more simple approach [Tsafrir *et al.*, 2007] averages the two last available run-times of the job's user. This method makes full use of the dependency between successive run-times, but does not make use of the job's description. Its main selling point are its simplicity and accuracy.

An advantage of single value prediction lies in the existence of many scheduling algorithms that can use.

3.2 Predicting a distribution

An algorithm that uses a probability distribution of single job run-times [Nissimov and Feitelson, 2008] has been proposed, with an accompanying distribution prediction technique [Nissimov, 2006]. This technique is similar to the previous method of averaging the two previous run-times for the job's user, in the sense that it only relies on the run-time information. It treats successive run-times of a given user as the observations of a Hidden Markov Model [Rabiner, 1989]. It does not use the job description.

4 A Regression Approach

We make the observation that single job run-time prediction might be improved by using both the 'run-time locality' (i.e., the dependency between successive run-times) and the job features. We will experiment with a method that will enable us to bridge this gap. The chosen method is to perform regression on vectors containing the following features:

- The continuous and discrete attributes from the job's description, such as required-time, amount of cores required, or time of the day of the submission.
- The attributes of the last *n* jobs of this user, where *n* is to be chosen as to balance between the cost of fitting the model and its accuracy.
- Attributes from the user and the system, such as the amount of nodes the user is currently using on the system, or the mean run-times of jobs of the user.
- Predicted values that come from other techniques, such as averaging the two previous run-time values.

This approach has the advantage of being rather adaptable to various formulations of the job description. No two HPC system's RMS are identical, therefore a generic ML approach provides a clear added value, as it is now the algorithm's responsibility deal with the particularities of the input data on each system. It is however difficult to provide validation of this aspect of the specific algorithm we will use, and this will not be in the scope of this paper.

4.1 Random Forests

The specific ML algorithm we use is called Random Forests, and in particular the CART [Breiman, 2001] method. This technique is an ensemble learning method that uses Decision Tree Learning [Breiman *et al.*, 1984]. Decision Trees partition the feature space: in a tree, the splitting is performed by using a threshold for continuous features and a all-way split on categorical features. Decision Trees are usually learned on a training set with a recursive top-down greedy algorithm that optimizes a function of the tree and the training set. Such a function can be for instance the Information Gain [Kullback and Leibler, 1951] of the tree on the training set. A criterion on the purity of leaves and/or amount of vectors that fall therein is used to stop the construction. Training set real output values are then averaged in each leaf to give predictions: When regressing a data point, it is sent trough the tree and the algorithm outputs the value of the corresponding leaf. The CART algorithm functions in the following manner:

- Training:
  - Randomly partition the training data.
  - Build decision trees on each partition.
- Predict a value by combining (e.g., average or linear combination) the results from all the trees.

4.2 Interpretation of the model

This approach has the advantage of producing an explainable model. Nodes that are, on average, higher up the decision trees are more important in the decision process. By ranking input vector attributes according to their average height in the trees, we will be able to gain insight about which feature is the most crucial to predicting the run-time. However, given the important height and complexity of the trees, it is hard to produce more information. There exist techniques [Palczewska *et al.*, 2013] that may allow to interpret further the model, however they are not implemented in the free and open-source Random Forest packages. We did not use them, because of time constraints.

5 Preliminary Results

We run the CART algorithm on a dataset built from the CURIE log, which was analyzed in Subsection 2.2. In a first approach, we choose to validate our approach by training the algorithm with the first 80% of the job/run-time associations and predicting the last 20% of the run-times. We then compare our results with an existing popular baseline, namely averaging the two last available run-time values from the job user [Tsafrir *et al.*, 2007], in order to validate our method.

5.1 Feature List

Table 1 shows the exact data vectors we use when running the random forests.

Features concerning the characteristics of the previous jobs of the user are not present in the CURIE log. are extracted by replaying the log with the Simpy [Muller and Vignaux, 2003] discrete event simulation package.

The experiments are run using the Scikit-Learn [Pedregosa *et al.*, 2011] package. Categorical features *user ID*, *group ID*, *day of week*, *last Status*, and *second to Last status* are encoded in a one-hot fashion[2], as the package we use only builds binary decision trees on continuous attributes.

[2]The one hot encoding of a n-valued categorical attribute consists of n binary attributes where each one corresponds to a category.

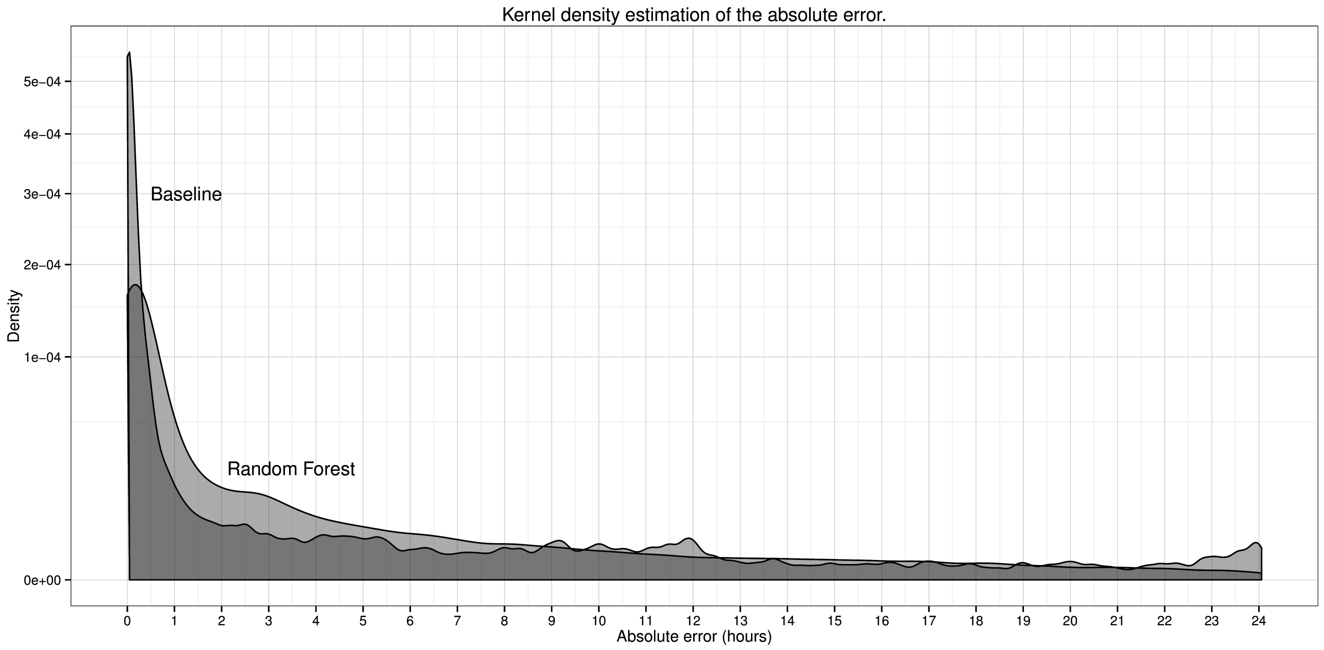

Figure 3: Kernel Density Estimation of the squared error made in the prediction for both the baseline and random forest methods, on the 20% of the last jobs of the CURIE dataset.

ID	Features
1	Processor Request
2	Required-time
3	Average of two last Run-times of job user
4	Last known run-time of user
5	Second to Last known run-time of job's user
6	Time since last job's end, if known.
7	Maximum length of already running job of job's user
8	Sum of already running jobs of job's user
9	Amount of jobs of this user running
10	Average run-time by currently running jobs of user
11	Total cores currently used by user
12	User ID
13	Group ID
14	Day of Week
15	Last known job exiting status of user
16	Second to Last known job exiting status of user

Table 1: Vector features for random forest regression.

5.2 Comparison with the baseline

We compare the results from our method with the run-time averaging baseline. Figure 2 shows the histogram of the error to the true value of run-time of the real run-times. We can see that our method provides a rather thin-tailed distribution of the error. This is a rather interesting attribute because some scheduling policies are impacted by this error in a convex manner. An example is provided in the case of a list-based scheduling algorithm, which optimizes fairness between users. In such a system, we might be concerned with the stretch ratio of a job:

$$\text{Stretch Ratio} = \frac{\text{Time from job submit to job completion}}{\text{Run-time}}$$

On a highly loaded system with many users submitting jobs with run-times of the order of a minute, a user who has a short, ten second job that is falsely estimated to a run-time of an hour will see his job highly delayed. Its stretch ratio will be highly degraded. So far we can not see any convexity in the cost function associated with our error. In practice, we are often concerned with minimizing the maximum stretch ratio value across all jobs and users. Using the maximum value introduces convexity and we would now like to avoid extreme values of the error at all costs.

Because of such arguments for the convexity of a relevant measure of the error produced by the predictor, we argue that the classic Mean Squared Error (MSE) metric is appropriate for estimating the quality of our predictions. The MSE metric of a predicted set of runtime values $\{r_{pred,i}\}$ with respect to a set of true values $\{r_i\}$, for i in $[\![1,N]\!]$ is:

$$MSE(\{r_{pred,i}\}) = \frac{1}{N}\sum_{i=1}^{N}(r_{pred,i} - r_i)^2$$

This is a general treatment of the question of how to measure the quality of the prediction: when validating a prediction technique on a specific scheduling algorithm, a measure specific to this algorithm should be used. Nonetheless, as minimizing the prospect of extraordinary events is a rather practical concern of system administrators, a convex measure is proper. According to the MSE measure, our method outperforms the simple averaging of the two previous run-times of the user, as shown in Table 2. However, the Mean Absolute Error(MAE) is higher. The standard error is lowered by our method.

Measure	Baseline	Random Forest
MSE	1.98497515×10^8	1.58202218×10^8
MAE	4.680836×10^3	5.551237×10^3
Standard Error	5.312812×10^1	4.512378×10^1

Table 2: Performance of both prediction techniques on the 20% of the last jobs of the CURIE log

5.3 Feature importance

Figure 4 shows the relative importance of the job attributes. It indicates that the two most relevant attributes are the baseline prediction and the required-time of a job. In this sense, the model succeeds in combining job characteristics and temporal dependence. Low dependency on the User ID and Group ID attributes may be attributed to the one-hot encoding we performed.

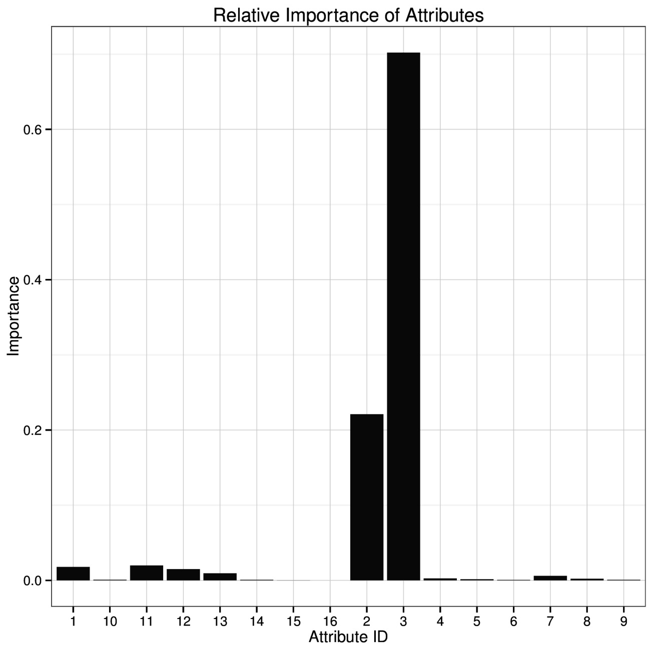

Figure 4: Relative importance of features in the trained Random Forest model.

6 Conclusions and Perspectives

We have shown that single-valued run-time prediction can be improved by using both job characteristics and run-time locality. Our method joins these aspects by performing regression on an extended set of features. It outperforms a popular existing method according to the sum-of-squares metric, which we argue is appropriate given its convexity. A series of extensions to this work are possible:

- Use a Online Random Forests algorithm. This is necessary in order to apply the method to a real system.
- Try the same method again with systematic full splitting on categorical attributes. This might add to the usefulness of the categorical attributes.
- Validate the approach on more data sets.
- Explore applications of the same methodology of mixing signal locality and job characteristics to single job runtime *distribution* prediction.
- Justify our approach by further adding to the existing literature concerning the usefulness of run-time prediction. An experimental and/or theoretical study could be done to assess the impact of the squared error on various popular and/or highly effective (e.g., classic algorithms to the strip packing problem [Dutot *et al.*, 2004]) scheduling algorithms that optimize various objectives. The impact of the signed error to run-time could also be studied, indeed some algorithms such as FCFS with backfilling have asymmetric cost functions to errors in run-time values. This could lead to a theoretical or experimental model of a cost function that could be used in a decision theoretic framework with distribution-based prediction. Such an asymmetric cost function could also be used when learning the decision trees.

7 Acknowledgements

I would like to thank both my advisors for the opportunity to work on this problem and the time they spent to meet and discuss it in spite of their busy schedules. Thank you Denis for convincing me to do a research master!

References

[Breiman *et al.*, 1984] Leo Breiman, Jerome Friedman, Charles J Stone, and Richard A Olshen. *Classification and regression trees*. CRC press, 1984.

[Breiman, 2001] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[Capit *et al.*, 2005] Nicolas Capit, Georges Da Costa, Yiannis Georgiou, Guillaume Huard, Cyrille Martin, Grégory Mounié, Pierre Neyron, and Olivier Richard. A batch scheduler with high level components. In *Cluster Computing and the Grid, 2005. CCGrid 2005. IEEE International Symposium on*, volume 2, pages 776–783. IEEE, 2005.

[Dutot *et al.*, 2004] Pierre-François Dutot, Grégory Mounié, and Denis Trystram. *Handbook of Scheduling*, chapter Scheduling Parallel Tasks - Approximation Algorithms. Number 26. CRC press, joseph leung edition, 2004.

[Feitelson *et al.*, 2012] Dror G. Feitelson, Dan Tsafrir, and David Krakov. Experience with the parallel workloads archive. 2012.

[Gibbons, 1997] Richard Gibbons. A historical application profiler for use by parallel schedulers. In DrorG. Feitelson and Larry Rudolph, editors, *Job Scheduling Strategies for Parallel Processing*, volume 1291 of *Lecture Notes in Computer Science*, pages 58–77. Springer Berlin Heidelberg, 1997.

[Kullback and Leibler, 1951] S. Kullback and R. A. Leibler. On information and sufficiency. *The Annals of Mathematical Statistics*, 22(1):79–86, 03 1951.

[Mu'alem and Feitelson, 2001] Ahuva W. Mu'alem and Dror G. Feitelson. Utilization, predictability, workloads, and user runtime estimates in scheduling the ibm sp2 with backfilling. *IEEE Trans. Parallel Distrib. Syst.*, 12(6):529–543, June 2001.

[Muller and Vignaux, 2003] K Muller and Tony Vignaux. Simpy: Simulating systems in python. *ONLamp. com Python Devcenter*, 2003.

[Nissimov and Feitelson, 2008] Avi Nissimov and Dror G. Feitelson. Probabilistic backfilling. In *Proceedings of the 13th International Conference on Job Scheduling Strategies for Parallel Processing*, JSSPP'07, pages 102–115, Berlin, Heidelberg, 2008. Springer-Verlag.

[Nissimov, 2006] Avi Nissimov. Locality and its usage in parallel job runtime distribution modeling using HMM. Master's thesis, The Hebrew University, 2006.

[Palczewska *et al.*, 2013] A. Palczewska, J. Palczewski, R. Marchese Robinson, and D. Neagu. Interpreting random forest classification models using a feature contribution method. *ArXiv e-prints*, December 2013.

[Pedregosa *et al.*, 2011] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[Rabiner, 1989] Lawrence Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

[Schwiegelshohn and Yahyapour, 1998] Uwe Schwiegelshohn and Ramin Yahyapour. Analysis of first-come-first-serve parallel job scheduling. In *Proceedings of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '98, pages 629–638, Philadelphia, PA, USA, 1998. Society for Industrial and Applied Mathematics.

[Staples, 2006] Garrick Staples. Torque resource manager. In *Proceedings of the 2006 ACM/IEEE Conference on Supercomputing*, SC '06, New York, NY, USA, 2006. ACM.

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

[Yoo *et al.*, 2003] Andy B Yoo, Morris A Jette, and Mark Grondona. Slurm: Simple linux utility for resource management. In *Job Scheduling Strategies for Parallel Processing*, pages 44–60. Springer, 2003.