Master 1 MoSIG Research Project Report
Learning Job Runtimes in HPC Systems

Valentin Reis
Supervised by: Denis Trystram & Eric Gaussier.

I understand what plagiarism entails and I declare that this report is my own, original work.
Name, date and signature:

Abstract

In many HPC infrastructures, the descriptions of the tasks to be executed are subject to high uncertainty. We show that users are unreliable when estimating the run time of their jobs, and look into alternative solutions. Predictive techniques are investigated in order to infer the run time of jobs from their full description and system history. A Machine Learning technique, Random Forests is applied and compared against the state of the art. We show that the prediction obtained with this approach has good properties with respect to relevant metrics.

1 Introduction

High Performance Computing (HPC) systems are complex machinery at the frontier between research in scheduling and systems engineering. We outline two of the main difficulties that resource management software in this field have to face.

First, the ephemeral nature, and broad range of existing architectures of those systems make the development and application of theoretical results difficult. New schemes for distributing resources are more present than ever. Many recent systems have complex network and memory/hard drive sharing topologies. Moreover, the topology of HPC 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 which can deal with complex systems and adapt to their evolutions poses a challenge.

In addition, the data (i.e. the characteristics of the tasks to be executed) these systems have to work with presents many peculiarities. The nature of the information which users of the system provide is very often loose, by instance with only upper and/or lower bounds on numerical quantities 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.

As a consequence of these difficulties most free, open-source and commercial resource management software 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) policy to schedule parallel jobs on a homogeneous cluster of machines, which 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 we are headed in with this research project is to deal with the input data of the resource management systems. Accommodating for this data seems separable enough from the actual scheduling problems for work towards this objective to be rewarding. No innovative ways to query the data from the users will be studied, we will rely on existing logs from HPC systems. Instead, we seek to apply Machine Learning(ML) techniques in order to reduce uncertainty of, and extract information and/or structure from, the input data of the HPC systems. We will be working with the problem of presenting input data in the most valuable way possible to a scheduling algorithm. How to use this data to the fullest will not be discussed. When assessing the relevance of the specific information we choose to produce from the job characteristics, references from the scheduling literature and existing systems will provide ground to stand on.

1.2 Job runtimes

Most HPC resource management software (including the SLURM, OpenPBS, OpenLava and OAR software) do ask information about jobs to users, such as topological requests in terms of processing units and memory, the 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 **reqtime** in the rest of the paper. Most of these software use the **reqtime** of a job as an upper bound on its run time, and kill it should **reqtime** be violated. As a consequence, users overestimate this value, should they choose or be forced to provide it. The following section will present a statistical analysis

pertaining to this relationship.

The true run time of a job with respect to a given affected topology is of great interest, as the scheduling policies are highly dependent on this information to provide good solutions ẽftruc. We will refer to this quantity as the **runtime** of a job. It must be clear that in the context of topological heterogeneity, the **runtime** of a job is only defined with respect to a specific processing environment to which it might be affected. This can include, and is not limited to, the network topology of the processing units, the availability of shared memory, message passing costs, and the operating system supporting the computations.

1.3 Problem Statement

In this paper, the broad question of refining the data is reduced to a single variable, the **runtime**. The problem statement we are dealing with is the following.

Given a specific homogeneous HPC Cluster with negligible communication costs, how to best predict the value of the **runtime** 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 **runtime** becomes an intrinsic attribute of a job. On the contrary to the input data which is always subject to peculiarities of the various systems and software, the **runtime** is always present as a simple field in workload logs of HPC systems, such as those available at the workload archive [**?**].

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 runtime 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 which are sizeable enough to learn from. As for downplaying the impact of communication costs, we will further restrict our work to machines which do not possess overly complex topology or distribute computing nodes across more than a handful of routers. In essence, we are targeting large Beowulf clusters, supercomputers, GPU farms and mainframe clusters.

2 Motivation

2.1 Importance of runtime

Once again, we emphasise the role of the **runtime** in scheduling tasks. Virtually all results from scheduling theory use the 'clairvoyant' model [Dutot *et al.*, 2004], where **runtimes** 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 been shown [Tsafrir *et al.*, 2007] that even in the classical approach (which is to use policies that are robust to this uncertainty), there is added value when this variable is refined. In all cases, reducing uncertainty in **runtime** 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

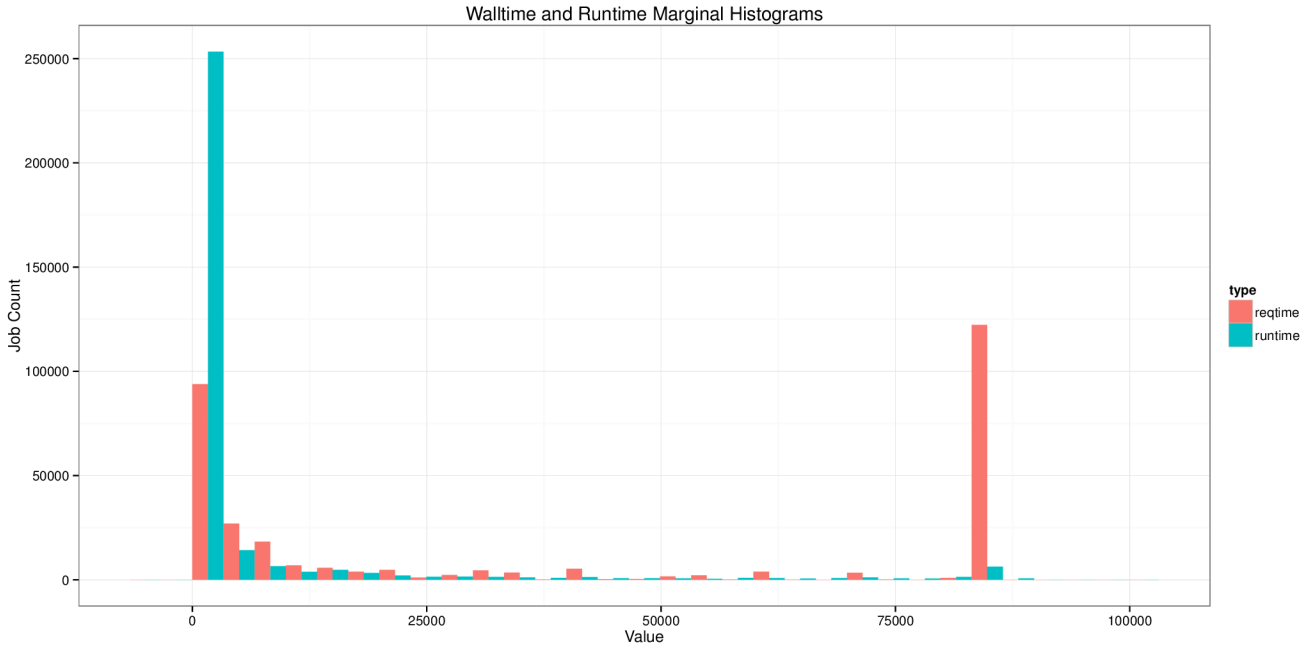

Figure 1: Marginal Histograms of the **reqtime** and **runtime** of all jobs from the CEA CURIE log.

show why a simple scheduling heuristic is applied.

2.2 Reqtime vs Runtime on a real system

The following study is conducted on a log[ref log] containing 20 months worth of data from the CURIE[ref 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 from the workload archive [**?**]. The log has several homogeneous CPU partitions and CPU+GPU partitions, however in a each partition, all nodes are identical. Jobs are allocated to partitions using user preference. The system is managed using the SLURM (Simple Linux Utility for Resource Management) software. Figure 1 shows the marginal distributions of **reqtimes** and **runtimes** on this system. The marginal distributions are already revealing, we can see that many **reqtimes** are in the 24h bin. The reason for this 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 **runtimes**, in which case the maximum value is used by the scheduler. Further looking into the relationship between reqtime and runtime, Figure 3 shows how the ratio $\frac{\textbf{runtime}}{\textbf{reqtime}}$ is distributed. This histogram indicates that a majority of users either: have very little idea about the expected **runtime** of their jobs, or overestimate very strongly its value on purpose. Sophisticated scheduling methods are not applied: under such uncertainty in the **runtime**, their performance is equivalent to the simple heuristic which is applied, namely FCFS with Backfilling [Mu'alem and Feitelson, 2001].

3 State of the art in runtime prediction

As mentioned previously, the latent question when predicting the **runtime** is how to provide the information to the scheduling algorithm. This section presents a few alternatives and state of the art methods in each case.

3.1 Predicting a value

Predicting values has been first attempted [Gibbons, 1997] by binning jobs in a predefined partitioning of their feature space and averaging values in each bin to provide an estimate. In this method, the partitionning has to be provided by the Resource Management Software or the system administrator. This method assumes the data 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 runtimes of the job's user. This method makes full use of the dependency between successive runtimes, but does not make use of the jobs's description. Its main selling point are it's simplicity and accuracy.

3.2 Predicting a distribution

An algorithms that uses a probability distribution of single job runtimes [Nissimov and Feitelson, 2008] have been proposed, with an accompanying distribution prediction tool [Nissimov, 2006]. This tool is similar to the previous method of averaging the two previous runtimes for the job's user, in the sense that it only relies on the runtime information. It treats successive runtimes of a given user as the observations of a Hidden

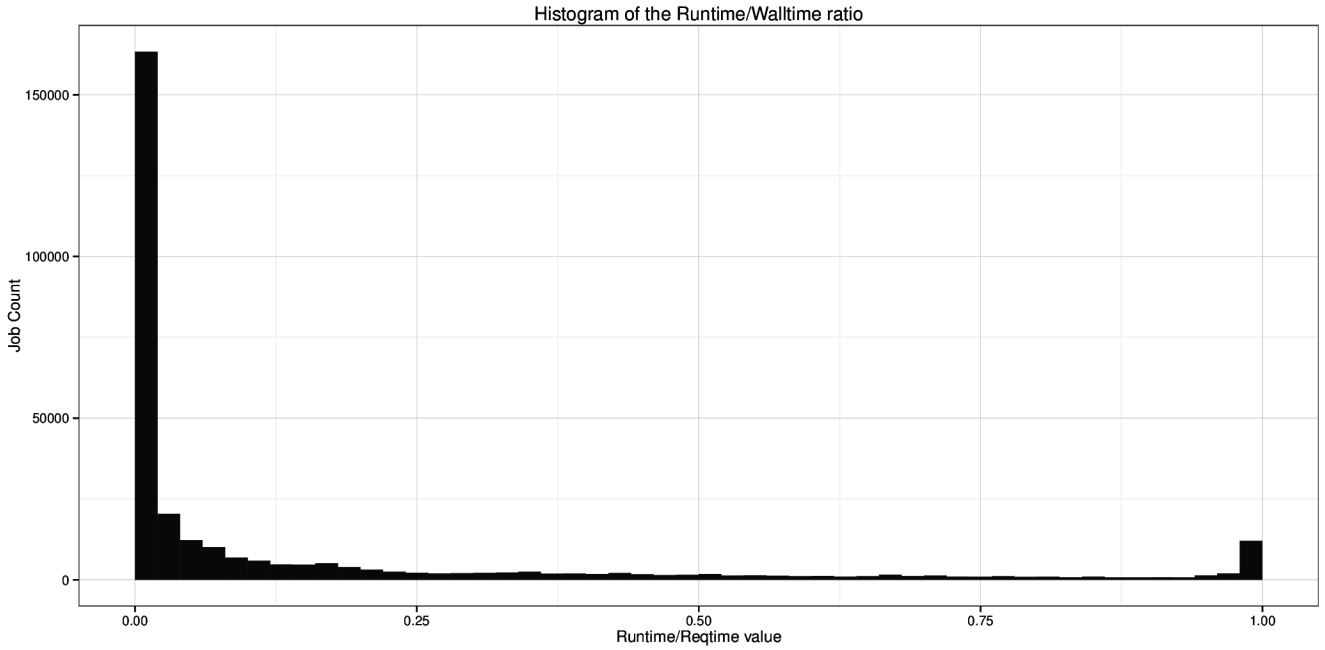

Figure 2: Histogram of the $\frac{\textbf{runtime}}{\textbf{reqtime}}$ ratio in the CURIE log.

Markov Model [Rabiner, 1989]. It does not use the job description.

4 A Regression Approach

We make the observation that single job runtime prediction might be improved by using both the 'runtime locality' (i.e. the dependency between successive runtimes) and the job features. We will experiment with a method that will allow 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 **reqtime**, number of cores required,or time of the day of the sumbission.
- 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 runtimes of jobs of the user.
- Predicted values which come from other techniques, such as averaging the two previous runtime values.

This approach has the advantage of being rather adaptable to various formulations of the job description. No two HPC systems are identical, therefore this 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 specfic ML algorithm we use is called Random Forests, and in particular the CART [Breiman, 2001] method. This technique is an ensemble learning method based upon 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 which 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 which 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 which 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

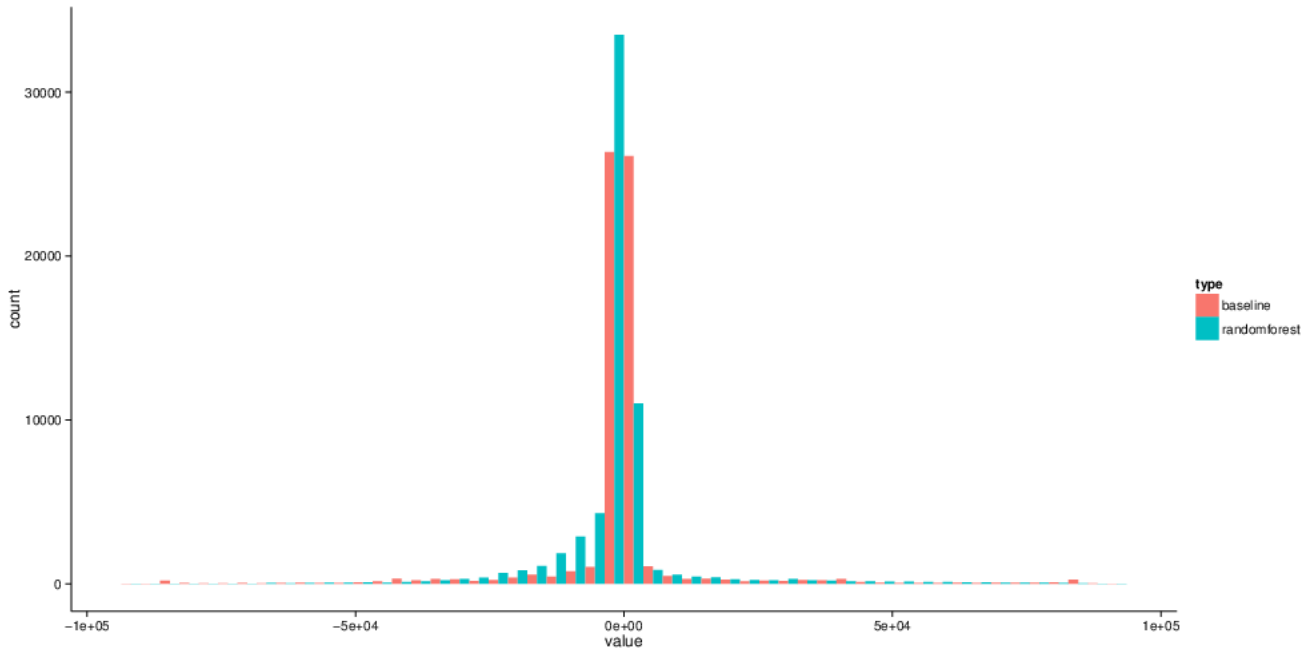

Figure 3: Histogram of the error made in the predictionfor both the baseline and random forest methods, on the 20% of the last jobs of the CURIE dataset.

the most crucial to predicting the runtime. However, given the important height and complexity of the trees, it is hard to produce more information. There exist techniques [Palczewska *et al.*, 2013] to interpret further the model, however they are not implemented in the free and open-source Random Forest packages, and we did not use them due to time constraints.

5 Preliminary Results

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

5.1 Feature List

Figure 4 shows the exact data vectors we use when running the random forests.

ID	Features
1	Processor Request
2	Reqtime
3	Average of two last Runtimes of job user
4	Last known runtime of user
5	Second to Last known runtime of job's user
6	Thinktime preceding this job
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 runtime by curr. 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

Figure 4: Vector features for random forest regression.

Features concerning the characteristics of the previous jobs of the user are not present in the CURIE log. 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. Features $UserID$, $GroupID$, $DayofWeek$, $LastStatus$, $SecondtoLaststatus$ are encoded in a one-hot fashion [1], as the package we use only builds binary decision trees on continuous attributes.

5.2 Comparison with the baseline

We compare the results from our method with the runtime averaging baseline. Figure 3 shows the histogram of the error to the true value of runtime of the real runtimes. 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 exemple is provided in the case of a list-based scheduling algorithm which optimizes fairness between users. In such a system, we are concerned with the stretch ratio of a job:

$$\text{Stretch Ratio} = \frac{\text{Time from job submit to job completion}}{\textbf{Runtime}}$$

On a highly loaded system with many users who submit jobs with runtimes of the order of a minute, a user who has a ten second job which is falsely estimated at an hour will see his stretch ratio 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 this argument for the convexity of a relevant measure of the error produced by the predictor, we argue that the classic least-squares method for estimating the quality of our predictions is appropriate. In this sense, we outperform the simple averaging of the two previous runtimes of the user, as shown in figure 5.

Baseline	$9.897605e+12$
Random Forest	$1.24186e+13$

Figure 5: Least square measure of the error of both prediction techniques on the 20% of the last jobs of the CURIE log

5.3 Feature importances

Figure 6 shows the relative importance of the different attributes. It shows that the two most relevant attributes are the baseline and the **reqtime**.

6 Conclusions and Extensions

We intend to run the experiments again with systematic full splitting on categorical attributes.

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 this work in spite of their very busy schedules!

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.

[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.

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

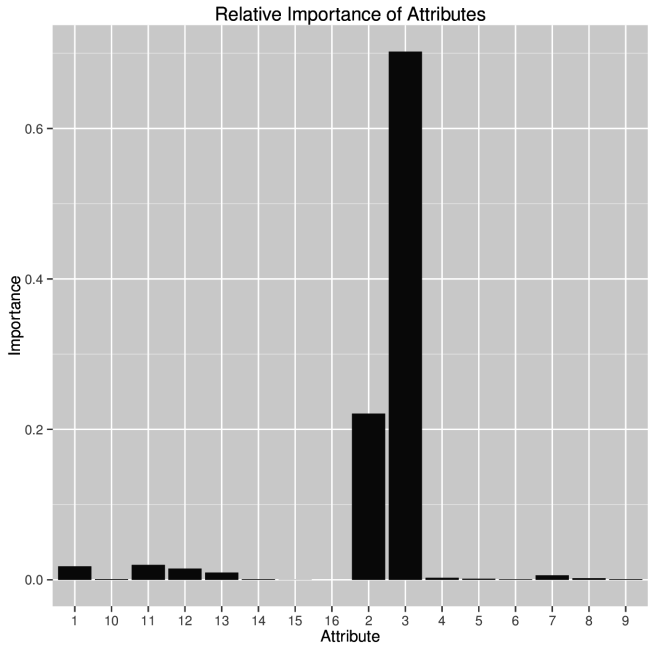

Figure 6: Relative importance of features in the Random Forests algorithm.

[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.

[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.