Master 1 MoSIG Research Project Report
Learning Job Runtimes in Homogeneous 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. Two Machine Learning techniques, Random Forests and SVMs are applied and compared against the state of the art.

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 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 [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 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. Classical results from scheduling theory use the 'clairvoyant' model[ref], where all **runtimes** of jobs with respect to any affectation on the system being modeled are known. Informally, if we want to make use of this extensive theoretical body, we must at least reduce uncertainty in this information. More recent models [ref Trystram FCFS] do [...]. Another approach [Nissimov and Feitelson, 2008] uses predicted probability distributions job runtimes in order to take scheduling decisions. 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, and out-

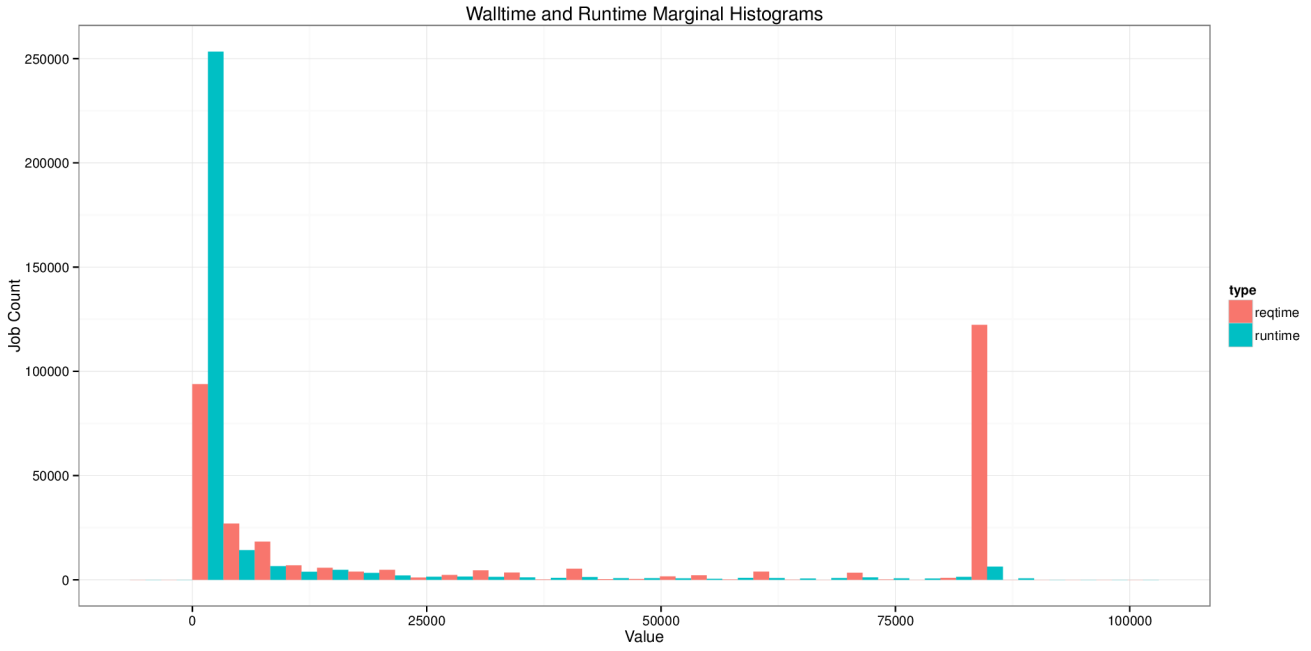

Figure 1: Marginal Histograms of the **reqtime** and **runtime** of all jobs from the CEA Curie log.

line the reasons why, in the absence of a predictive technique applied the input data, a simple 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 [ref 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 2 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 [ref backfilling]

3 State of the art in runtime prediction

As mentioned previously, the latent question when predicting the **runtime**

3.1 Predicting a value

TODO: -give the references and explain the historical methods, gibbons historical scheduler, the tsafir et al paper with mean of two last runtimes values userwise..

3.2 Predicting a distribution

TODO: -give the references and explain the probabilistic backfilling thing..

4 Our approach

WHY ML? well, there is simple little patterns everywhere

4.1 Random Forests

TODO: -explain our approach, why it could lead to better results (external info+signal locality(ref hmm thesis for locality..))

4.2 Explainability

TODO: -explain one advantage of random forests: discussion about the trees after learning..

5 Preliminary Results

TODO:results..

6 Conclusions

TODO:conclure..

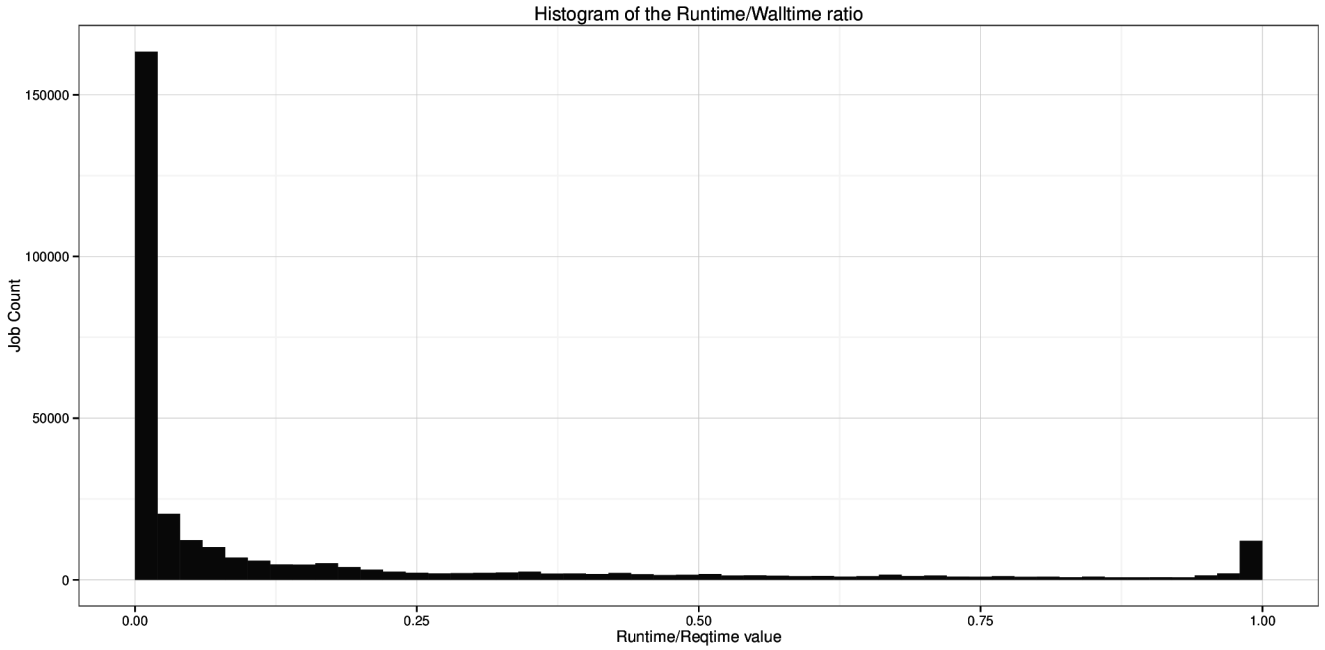

Figure 2: Histogram of the $\frac{\textbf{runtime}}{\textbf{reqtime}}$ ratio in the Curie log.

7 Acknowledgements

TODO:remercier..

References

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

[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.