Collection of Job Scheduling Prediction Methods

Mehmet Soysal and Achim Streit

Steinbuch Centre for Computing,
Karlsruhe Institute of Technology, Hermann-von-Helmholtz-Platz 1,
Eggenstein-Leopoldshafen, Germany
{mehmet.soysal,achim.streit}@kit.edu

Abstract. For more than two decades, researchers have been developing methods to predict HPC job run times. The methods vary from simple rule-based solutions to modern methods using machine and deep learning libraries. These studies are often developed for scientific publications and the sustainability after publication is often neglected. It is also often difficult to compare a solution with previously published, because the focus can be slightly different and missing the availability of source code. With this work, we want to start a collection of different wall time prediction methods to better compare new solutions and to promote sustainability. Therefore, this paper is submitted as a Open Scheduling Problem. The collection of source codes and results are going to be published on GitHub: https://hpc-job-scheduling-repo.github.io

Keywords: Job Scheduling · Job Prediction · Job Walltime

1 Introduction

For the execution of applications on HPC systems, a so-called job is created and submitted to a queue [1]. A job describes the application, needed resources, and requested wall time. A HPC Job Scheduler manages the queue and orders the jobs for efficient use of the resources. To plan the future usage of resources, job schedulers typically use a wall time that corresponds to the maximum execution time for each job. This wall time, also known as estimated job run time or wall (clock) time, is crucial for accurate planning. Nevertheless, users tend to request more time to prevent jobs being canceled too early. Accurate estimates of job wall time are important for many purposes, such as making better predictions about when waiting jobs will start. In addition, this information is needed when data should be staged in advance on the compute nodes [2]. There is also a online prediction system available for the XSEDE [3] ressources – KARNAK [4]. Karnak uses machine learning to provide a prediction for users either when their job will start, or how long a hypothetical job would wait before being started.

Without accurate job wall time estimation, it is almost impossible to make any preparation of the system for future job requirements. This challenge is more important if the HPC systems become larger. For future exascale systems, this can help to improve the overall efficiency significantly. For these reasons,

there are always attempts to predict the wall time more accurately. There are many different approaches and solutions. With this work we would like to start a collection of these solutions. The source code and the results will be made available on a website as a repository. Thereby, we want to ensure sustainability and if needed, the results can be reproduced by other scientists. Moreover, a better comparison of the own solution with the existing ones is easier. This repository is initially for job wall time prediction, but can be extended for other problem cases.

The remainder of this paper is structured as follows: In Section 2 we give a brief introduction to the background of job wall time predictions. We show in Section 3 what is needed to make the solutions better comparable. In Section 4 we finish with a conclusion and outlook on future work.

2 Background

The background of this work covers two areas. On the one hand, the approaches to predict job wall times and on the other hand, historical job workload logs. As a source for historical job workloads, the Parallel Workload Archive (PWA) is well established [5,6]. However, it happens from time to time that logs are used for publications which are not publicly available. The reasons for this can be manifold, e.g., to show how privacy sensitive data like directories and job names can be used for better predictability [7]. Unfortunately, this complicates the ability to compare different approaches or reproduce the work. Therefore, only a comparable and reproducible solution should be accepted into the collection.

There are several approaches to improve wall time estimates, and we can only give a brief overview in this section. The different methods can be divided into 2 different areas. One is the traditional methods and the "new" ones with the use of machine or deep learning libraries. As representatives for traditional methods, the solutions of Gibbons [8,9] and Downey [10] or also the built-in functionality of the ALEA Scheduling simulator [11]. Gibbons and Downey used historical workloads to predict job wall times. The prediction was performed based on templates. Previously collected metadata was analyzed and grouped according to similarities. Alea determines the deviation between the user estimated wall time and the used wall time is determined and applied to new jobs. It is working on a per-user basis. A new run-time estimation for a new job is computed using information about previous jobs of that user.

In recent years, **new methods** like machine and deep learning methods were used to predict resource consumption in studies [8,12–15]. As an example, Smith [16] is using genetic and greedy search methods to estimate job run times and queue wait times. This solution is used by XSEDE to predict queue wait time [3]. Other popular methods are using linear regression for predictions [17, 18], Nearest Neighbors [19,20], Regression Trees [21], or Instance Learning [16,22, 23]. Matsunga [24] used a combination of multiple machine learning methods to predict the execution time of two bioinformatics applications: BLAST [25] and RAxML [26]. These methods rely on domain experts in the machine learning

discipline to preprocess the input data and to select the correct model including the optimization of parameters. Therefore, in the recent time, automatizing the the machine learning process also gained attraction for predicting walltimes [7].

The examples given here show that there are many approaches and solutions. There are repeatedly publications in this area, but there is often a lack of comparisons with the corresponding metrics.

3 Predictions

For a sustainable and reproducible collection of tools, several conditions must be met. A source for job workloads as test data is needed. Different metrics and conditions should be specified to better compare the different solutions. Of course, the different solutions with their results should be available in a repository.

3.1 Job Workloads and Metadata

The Parallel Workload Archive offers job workload logs in SWF format [27]. It contains detailed workload logs collected from large-scale parallel systems. Every job in these data sets are represented by a sequence of lines (one job per line) containing 18 columns (job metadata). Available job metadata is listed below and Table 1 show all 40 available logs.

1. Job Number – Unique job identifier, also called JobID.
2. Submit Time – seconds starting from workload log time.
3. Wait Time – difference between Submit Time and Start Time in seconds.
4. Run Time – the actual time in seconds the job was running
5. Number of Allocated Processors – integer value of allocated cores or CPU, depends on configuration
6. Average CPU Time Used – both user and system, in seconds.
7. Used Memory – average used memory per core in kilobytes.
8. Requested Number of Processors
9. Requested Time – Wall time requested for job
10. Requested Memory – requested memory per processor in kilobytes
11. Status – a number indicating the reason why job has finished. 1 = job was completed normal, 0 = failed, and 5 = canceled. If this field can not be provided it is -1.
12. User ID – a number identifying a user.
13. Group ID – a number identifying a group.
14. Executable (Application) Number – a number to identify the application. If not available then -1.
15. Queue Number – a number identifying configured queues. It is suggested to use 0 for interactive jobs.
16. Partition Number – a number identifying configured partitions.
17. Preceding Job Number – a previous Job Number (JobID) which is the job is waiting to finish. With this a dependency between jobs can be established.

Table 1. Selected workloads and available metadata.* (Req. Number of Processors equals Allocated Number of Processors.)

	Job Number	Submit Time	Wait time	Run Time	# of Alloc. Processors	Average CPU Time Used	Used Memory	Req. # of Processors	Req. Time	Req. Memory	Status	User ID	Group ID	Exe. Number	Queue Number	Partition Number	Preceding Job Number	Think Time Preceding Job
ANL-Intrepid-2009-1	Y	Y	Y	Y	Y	n.A.	n.A.	*	Y	n.A.	n.A.	Y	n.A.	n.A.	Y	n.A.	n.A.	n.A.
CEA-Curie-2011-2	Y	Y	Y	Y	Y	n.A.	n.A.	*	Y	n.A.	Y	Y	Y	n.A.	n.A.	Y	n.A.	n.A.
CIEMAT-Euler-2008-1	Y	Y	Y	Y	Y	Y	Y	*	Y	n.A.	Y	Y	Y	Y	Y	Y	n.A.	n.A.
CTC-SP2-1995-2	Y	Y	Y	Y	Y	Y	n.A.	n.A.	n.A.	Y	Y	Y	n.A.	Y	Y	n.A.	n.A.	n.A.
CTC-SP2-1996-3	Y	Y	Y	Y	Y	Y	n.A.	n.A.	Y	n.A.	Y	Y	n.A.	Y	Y	n.A.	n.A.	n.A.
DAS2-fs0-2003-1	Y	Y	Y	Y	Y	Y	n.A.	*	Y	n.A.	Y	Y	Y	Y	Y	n.A.	n.A.	n.A.
DAS2-fs1-2003-1	Y	Y	Y	Y	Y	Y	Y	*	Y	n.A.	Y	Y	Y	Y	Y	n.A.	n.A.	n.A.
DAS2-fs2-2003-1	Y	Y	Y	Y	Y	Y	Y	*	Y	n.A.	Y	Y	Y	Y	Y	n.A.	n.A.	n.A.
DAS2-fs3-2003-1	Y	Y	Y	Y	Y	Y	Y	*	Y	n.A.	Y	Y	Y	Y	Y	n.A.	n.A.	n.A.
DAS2-fs4-2003-1	Y	Y	Y	Y	Y	Y	Y	*	Y	n.A.	Y	Y	Y	Y	Y	n.A.	n.A.	n.A.
HPC2N-2002-2	Y	Y	Y	Y	Y	Y	n.A.	*	Y	n.A.	Y	Y	Y	n.A.	n.A.	Y	n.A.	n.A.
Intel-NetbatchA-2012-1	Y	Y	Y	Y	Y	Y	Y	n.A.	n.A.	n.A.	Y	Y	Y	Y	n.A.	n.A.	Y	Y
Intel-NetbatchB-2012-1	Y	Y	Y	Y	Y	Y	Y	n.A.	n.A.	n.A.	Y	Y	Y	Y	n.A.	n.A.	Y	Y
Intel-NetbatchC-2012-1	Y	Y	Y	Y	Y	Y	Y	n.A.	n.A.	n.A.	Y	Y	Y	Y	n.A.	n.A.	Y	Y
Intel-NetbatchD-2012-1	Y	Y	Y	Y	Y	Y	Y	n.A.	n.A.	n.A.	Y	Y	Y	Y	n.A.	n.A.	Y	Y
KIT-FH2-2016-1	Y	Y	Y	Y	Y	n.A.	n.A.	*	Y	n.A.	Y	Y	Y	n.A.	Y	Y	n.A.	n.A.
KTH-SP2-1996-2	Y	Y	Y	Y	Y	n.A.	n.A.	*	Y	n.A.	Y	Y	Y	n.A.	n.A.	n.A.	n.A.	n.A.
LANL-CM5-1994-4	Y	Y	Y	Y	Y	Y	Y	*	Y	Y	Y	Y	Y	Y	Y	n.A.	n.A.	n.A.
LANL-O2K-1999-2	Y	Y	Y	Y	Y	Y	Y	*	n.A.	n.A.	Y	Y	Y	Y	Y	n.A.	n.A.	n.A.
LCG-2005-1	Y	Y	n.A.	Y	Y	n.A.	n.A.	n.A.	n.A.	n.A.	n.A.	Y	Y	n.A.	n.A.	Y	n.A.	n.A.
LLNL-Atlas-2006-2	Y	Y	n.A.	Y	n.A.	n.A.	n.A.	n.A.	n.A.	n.A.	Y	Y	n.A.	Y	n.A.	Y	n.A.	n.A.
LLNL-T3D-1996-2	Y	Y	n.A.	Y	Y	n.A.	n.A.	n.A.	n.A.	n.A.	Y	Y	n.A.	Y	n.A.	n.A.	n.A.	n.A.
LLNL-Thunder-2007-1	Y	Y	n.A.	Y	Y	n.A.	n.A.	*	n.A.	n.A.	Y	Y	n.A.	Y	n.A.	Y	n.A.	n.A.
LLNL-uBGL-2006-2	Y	Y	n.A.	Y	n.A.	n.A.	n.A.	n.A.	n.A.	n.A.	Y	Y	n.A.	n.A.	n.A.	n.A.	n.A.	n.A.
LPC-EGEE-2004-1	Y	Y	n.A.	Y	Y	Y	n.A.	*	Y	n.A.	Y	Y	Y	n.A.	Y	Y	n.A.	n.A.
METACENTRUM-2009-2	Y	Y	Y	Y	Y	n.A.	Y	n.A.	n.A.	n.A.	Y	Y	n.A.	n.A.	Y	Y	n.A.	n.A.
METACENTRUM-2013-3	Y	Y	Y	Y	Y	n.A.	n.A.	*	Y	Y	n.A.	Y	n.A.	n.A.	Y	Y	n.A.	n.A.
NASA-iPSC-1993-3	Y	Y	n.A.	Y	Y	n.A.	n.A.	n.A.	n.A.	n.A.	n.A.	Y	Y	Y	n.A.	n.A.	n.A.	n.A.
OSC-Clust-2000-3	Y	Y	Y	Y	Y	Y	n.A.	*	n.A.	n.A.	Y	Y	Y	n.A.	n.A.	n.A.	n.A.	n.A.
PIK-IPLEX-2009-1	Y	Y	Y	Y	Y	Y	Y	n.A.	n.A.	n.A.	Y	Y	Y	Y	Y	n.A.	n.A.	n.A.
RICC-2010-2	Y	Y	Y	Y	Y	n.A.	n.A.	*	Y	Y	Y	Y	Y	n.A.	Y	n.A.	n.A.	n.A.
Sandia-Ross-2001-1	Y	Y	Y	Y	Y	Y	Y	*	Y	n.A.	Y	Y	Y	Y	Y	Y	n.A.	n.A.
SDSC-BLUE-2000-4	Y	Y	Y	Y	Y	Y	n.A.	*	Y	n.A.	Y	Y	n.A.	n.A.	Y	n.A.	n.A.	n.A.
SDSC-DS-2004-2	Y	Y	Y	Y	Y	Y	n.A.	*	Y	n.A.	Y	Y	Y	n.A.	Y	Y	n.A.	n.A.
SDSC-Par-1995-3	Y	Y	Y	Y	Y	Y	n.A.	n.A.	n.A.	n.A.	Y	Y	n.A.	n.A.	Y	Y	n.A.	n.A.
SDSC-Par-1996-3	Y	Y	Y	Y	Y	n.A.	n.A.	n.A.	n.A.	n.A.	Y	Y	n.A.	n.A.	Y	Y	n.A.	n.A.
SDSC-SP2-1998-4	Y	Y	Y	Y	Y	Y	n.A.	*	Y	n.A.	Y	Y	Y	Y	Y	n.A.	n.A.	n.A.
SHARCNET-2005-2	Y	Y	Y	Y	Y	Y	Y	n.A.	n.A.	n.A.	Y	Y	n.A.	Y	n.A.	Y	n.A.	n.A.
SHARCNET-Whale-2006-2	Y	Y	Y	Y	Y	Y	Y	n.A.	n.A.	n.A.	Y	Y	n.A.	Y	n.A.	n.A.	n.A.	n.A.
UniLu-Gaia-2014-2	Y	Y	Y	Y	Y	Y	Y	*	Y	n.A.	Y	Y	Y	Y	Y	n.A.	n.A.	n.A.

18. Think Time from Preceding Job – a value indicating how long a job has to wait after a preceding job has finished before the job is started.

If more metadata is available then defined by the SWF format, these additional metadata could be published as a additional log. As an example for such a log, the historical log for the Gaia system offers a companion log with accumulated I/O for each job [28].

3.2 Metrics

To better compare the solutions, the same metrics should be used. The metrics proposed here should rather be seen as initial metrics and could change after discussion with the community. The Sci-kit library offers many different metrics to compare the quality of predictions. Thereby, the real used wall time (y) and the predicted wall time ($\hat{y}$) is passed to the corresponding function. Sci-kit offers several metrics and the metrics below are a selection of them, which might change in the future.

The mean absolute error (MAE) and the median absolute error (MedAE) measure the difference between predicted and used wall time [29, 30]. MAE is the mean over all pairs of predicted and used wall times,

$$\mathrm{MAE}(y, \hat{y}) = \frac{1}{n_{\mathrm{samples}}} \sum_{i=0}^{n_{\mathrm{samples}}-1} |y_i - \hat{y}_i|, \tag{1}$$

where y_i is the real used walltime and $\hat{y}_i$ is the predicted value of the i-th sample, and MedAE is the median value of these pairs,

$$\mathrm{MedAE}(y, \hat{y}) = \mathrm{median}(|y_1 - \hat{y}_1|, \ldots, |y_n - \hat{y}_n|). \tag{2}$$

In contrast to MAE, MedAE is robust against outliers. These metrics are widely used and easy to understand. The result indicates the deviation in seconds and are therefore very well suited to compare solutions with each other. Another suitable metric could be the Mean absolute percentage error (MAPE) which expresses the accuracy as a ratio and therefore its very intuitive interpretation [31]

$$\mathrm{MAPE}(y, \hat{y}) = \frac{100\%}{n} \sum_{i=1}^{n} \left| \frac{y_i - \hat{y}_i}{y_i} \right|. \tag{3}$$

Besides the accuracy of wall time predictions, there are other aspects that should be considered.

- Number of over- and under-estimates - An underestimation would be fatal in real scenarios, as the scheduler would cancel the job in these cases.
- Processing/Training time - In the case where methods are used to train a model, it should of course be recorded how much time is needed for the training.

- Training type - is a partial fit possible or does the model need to be trained from beginning.
- Cold start - what approach is used for the cold start.
- Prediction type - is the prediction done on a per-user basis.

Nevertheless, there are many points that impact the accuracy. However, a comparison becomes difficult if not the same dataset or metrics are used. Therefore, the data sets and metrics proposed here serve only as a starting point.

3.3 Repository

Only solutions that can be reproduced with published material should be included in the repository. This means also that solutions should be available under a free license. The solution can also be published on own pages, but it should be public so that the results can be reproduced. Preliminary repositories can be found on GitHub: `https://hpc-job-scheduling-repo.github.io`

4 Conclusion and Outlook

In this work, we propose a repository for HPC job wall time prediction approaches. This work shall serve as a starting point to collect scheduler predictions and metrics. With this, a comparison of the different solutions, approaches, and ideas should be possible. At the same time, sustainability is achieved by creating a repository for the approaches. The proposed metrics can change through feedback from the community. It is also conceivable to add further problem cases to the repository. The parallel workload archive and workload logs are a good source of HPC job workloads, but it shows that most workloads are very old and not all metadata are available. Here it might be useful to encourage the community to publish newer workloads to the archives.

5 Acknowledgement

We gratefully acknowledge funding by the Ministry of Science, Research and the Arts Baden-Württemberg and "Deutsche Forschungsgemeinschaft" (DFG).

References

1. Matthias Hovestadt, Odej Kao, Axel Keller, and Achim Streit. Scheduling in HPC Resource Management Systems: Queuing vs. Planning. In *Job Scheduling Strategies for Parallel Processing*, volume 2862, pages 1–20, 06 2003.
2. Mehmet Soysal, Marco Berghoff, Dalibor Klusáček, and Achim Streit. On the Quality of Wall Time Estimates for Resource Allocation Prediction. In *Proceedings of the 48th International Conference on Parallel Processing: Workshops*, ICPP 2019, Kyoto, Japan, pages 23:1–23:8, New York, NY, USA, 2019. ACM.
3. Xsede. `https://www.xsede.org/`.

4. Karnak start/wait time predictions. `http://karnak.xsede.org/karnak/index.html`.
5. Dror G. Feitelson, Dan Tsafrir, and David Krakov. Experience with using the Parallel Workloads Archive. *Journal of Parallel and Distributed Computing*, 74(10):2967–2982, 2014.
6. Parallel workloads archive. `http://www.cs.huji.ac.il/labs/parallel/workload/`.
7. Mehmet Soysal, Marco Berghoff, and Achim Streit. Analysis of Job Metadata for Enhanced Wall Time Prediction. In *Workshop on Job Scheduling Strategies for Parallel Processing*, pages 1–14. Springer.
8. Richard Gibbons. A historical profiler for use by parallel schedulers. *Master's thesis, University of Toronto*, 1997.
9. Richard Gibbons. A historical application profiler for use by parallel schedulers. In *Job scheduling strategies for parallel processing*, pages 58–77. Springer, 1997.
10. Allen B Downey. Predicting queue times on space-sharing parallel computers. In *Parallel Processing Symposium, 1997. Proceedings., 11th International*, pages 209–218. IEEE, 1997.
11. Dalibor Klusáček, Šimon Tóth, and Gabriela Podolníková. Complex Job Scheduling Simulations with Alea 4. In *Ninth EAI International Conference on Simulation Tools and Techniques (SimuTools 2016)*, pages 124–129. ACM.
12. Nirav H Kapadia and José AB Fortes. On the design of a demand-based network-computing system: The purdue university network-computing hubs. In *High Performance Distributed Computing, 1998. Proceedings. The Seventh International Symposium on*, pages 71–80. IEEE, 1998.
13. Ahuva W. Mu'alem and Dror G. Feitelson. Utilization, Predictability, Workloads, and User Runtime Estimates in Scheduling the IBM SP2 with Backfilling. 12(6):529–543.
14. Farrukh Nadeem and Thomas Fahringer. Using templates to predict execution time of scientific workflow applications in the grid. In *Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, pages 316–323. IEEE Computer Society, 2009.
15. Dan Tsafrir, Yoav Etsion, and Dror G Feitelson. Backfilling using system-generated predictions rather than user runtime estimates. *IEEE Transactions on Parallel and Distributed Systems*, 18(6), 2007.
16. Warren Smith. Prediction services for distributed computing. In *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International*, pages 1–10. IEEE, 2007.
17. Byoung-Dai Lee et al. Run-time prediction of parallel applications on shared environments. In *2003 Proceedings IEEE International Conference on Cluster Computing*, pages 487–491. IEEE, 2003.
18. Sena Seneviratne and David C Levy. Task profiling model for load profile prediction. *Future Generation Computer Systems*, 27(3):245–255, 2011.
19. Nirav H Kapadia, José AB Fortes, and Carla E Brodley. Predictive application-performance modeling in a computational grid environment. In *Proceedings. The Eighth International Symposium on High Performance Distributed Computing (Cat. No. 99TH8469)*, pages 47–54. IEEE, 1999.
20. Michael A Iverson, Fusun Ozguner, and Lee C Potter. Statistical prediction of task execution times through analytic benchmarking for scheduling in a heterogeneous environment. In *Proceedings. Eighth Heterogeneous Computing Workshop (HCW'99)*, pages 99–111. IEEE, 1999.

21. Tudor Miu and Paolo Missier. Predicting the execution time of workflow activities based on their input features. In *2012 SC Companion: High Performance Computing, Networking Storage and Analysis*, pages 64–72. IEEE, 2012.
22. Hui Li, David Groep, and Lex Wolters. An evaluation of learning and heuristic techniques for application run time predictions. In *Proceedings of 11th Annual Conference of the Advance School for Computing and Imaging (ASCI), Netherlands*. Citeseer, 2005.
23. Rafael Ferreira Da Silva, Gideon Juve, Mats Rynge, Ewa Deelman, and Miron Livny. Online task resource consumption prediction for scientific workflows. *Parallel Processing Letters*, 25(03):1541003, 2015.
24. Andréa Matsunaga and José AB Fortes. On the use of machine learning to predict the time and resources consumed by applications. In *Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, pages 495–504. IEEE Computer Society, 2010.
25. Stephen F Altschul, Warren Gish, Webb Miller, Eugene W Myers, and David J Lipman. Basic local alignment search tool. *Journal of molecular biology*, 215(3):403–410, 1990.
26. Alexandros Stamatakis. Raxml version 8: a tool for phylogenetic analysis and post-analysis of large phylogenies. *Bioinformatics*, 30(9):1312–1313, 2014.
27. The Standard Workload Format.
28. The university of luxemburg gaia cluster log. `https://www.cs.huji.ac.il/labs/parallel/workload/l_unilu_gaia/index.html`.
29. scikit - Mean absolute error. `http://scikit-learn.org/stable/modules/generated/sklearn.metrics.mean_absolute_error.html#sklearn.metrics.mean_absolute_error`.
30. scikit - Median absolute error. `http://scikit-learn.org/stable/modules/generated/sklearn.metrics.median_absolute_error.html#sklearn.metrics.median_absolute_error`.
31. Arnaud de Myttenaere, Boris Golden, Bénédicte Le Grand, and Fabrice Rossi. Mean absolute percentage error for regression models. *Neurocomputing*, 192:38–48, Jun 2016.