Analysis of Job Metadata for Enhanced Wall Time Prediction

Mehmet Soysal(✉), Marco Berghoff, and Achim Streit

Steinbuch Centre for Computing (SCC), Karlsruhe Institute of Technology (KIT), Hermann-von-Helmholtz-Platz 1, 76344 Eggenstein-Leopoldshafen, Germany
{mehmet.soysal,marco.berghoff,achim.streit}@kit.edu

Abstract. For efficient utilization of large-scale HPC systems, the task of resource management and job scheduling is of highest priority. Therefore, modern job scheduling systems require information about the estimated total wall time of the jobs already at submission time. Proper wall time estimates are a key for reliable scheduling decisions. Typically, users specify these estimates, already at submission time, based on either previous knowledge or certain limits given by the system. Real-world experience shows that user given estimates are far away from accurate. Hence, an automated system is desirable that creates more precise wall time estimates of submitted jobs. In this paper, we investigate different job metadata and their impact on the wall time prediction. For the job wall time prediction, we used machine learning methods and the workload traces of large HPC systems. In contrast to previous work, we also consider the jobname and in particular the submission directory. Our evaluation shows that we can better predict the accuracy of jobs per user by a factor of seven than most users, without any in-depth analysis of the job.

1 Introduction

For the execution of applications on HPC systems, a so-called job is created and submitted to a queue. A job describes the application, needed resources, and requested wall time. An HPC scheduler manages the queue and orders the jobs for efficient use of the resources. The jobs are waiting in the queue until the requested resources are available. The scheduler allocates the resources and starts the job [1]. For planning future usage of the resources, schedulers typically use a wall time that corresponds to the maximum execution time for each job. This wall time, also known as estimated job runtime or wall clock time, is usually given by the user, or a default value of the system is applied.

Often, users could be able to do a reasonable job runtime estimation, because they have detailed knowledge about their jobs. Nevertheless, the users tend to request more time then the job needs, to prevent jobs being terminated too early by the scheduler. This detailed knowledge is not available without interviewing the user. Without this knowledge, it is difficult for the scheduler to perform exact resource planning. Without accurate job wall time estimation, it is almost

D. Klusáček et al. (Eds.): JSSPP 2018, LNCS 11332, pp. 1–14, 2019.
https://doi.org/10.1007/978-3-030-10632-4_1

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. The project ADA-FS [2] (as part of the DFG-funded priority program 1648 "Software for Exascale Computing" SPPEXA) focuses on pre-staging of input data for massively parallel jobs. Previous to the data staging it is going to deploy a private filesystem across the allocated nodes. For this, it is essential to know on which nodes a queued job will be executed. The scheduler predicts these nodes based on the user given wall times of the already running jobs. Hence, precise wall time estimates are critical.

In this paper, we take a closer look at the individual metadata and examine their impact on the prediction. Machine learning methods are used to determine the influence of additional metadata. In particular, we use previously unconsidered metadata for jobs in the workload traces of our HPC systems to support the machine learning methods, e.g., information about the working directory of jobs, which typically contains valuable information about the jobs itself.

The remainder of this paper is structured as follows: In Sect. 2 we give a brief introduction to machine learning and similar approaches. We show in Sect. 3 how we prepared our historical data and also explain metrics to rate our results. In Sect. 4 we present the results on the used metadata and finish with a conclusion and outlook to future work in Sect. 5.

2 Related Work

2.1 Predicting Job Walltimes

Enhanced predictions of HPC job wall time can be used to improve the scheduling performance [3]. With exact information about the runtime of a job, the scheduler can predict more accurately when sufficient resources are available to start queued jobs. [4]. However, the user requested wall time is not close to the real used wall time. Gibbons [3,5], and Downey [4] use historical workloads to predict the wall times of parallel applications. They predict wall times based on templates. These templates are created by analyzing previously collected metadata and grouped according to similarities. However, both approaches are restricted to simple definitions. Smith et al. [6,7] applied greedy and genetic search techniques to identify similar jobs and partition them into categories. The studies as mentioned above use templates to find similarities and use these for wall time predictions. In our evaluation, we do not use templates. In the recent years, the machine learning algorithms are used to predict resource consumption in several studies [5,8–13]. However, these studies do not take into account the additional metadata we do. There is also a online prediction system available for the XSEDE [14] resources – KARNAK [15]. Karnak also uses machine learning to provide a prediction for users either when their job will start, or how long a hypothetical job would wait before starting on selected XSEDE resources. For this prediction the requested wall time, processors, queue, and the system has to be provided. In our evaluation we consider more metadata.

2.2 Machine Learning

Machine learning (ML) is about knowledge retrieval from data. It can also be understood as statistical learning and predictive analytics. In general, machine learning is a method to learn from a set of samples with a target value and use the learned data to predict target values from unknown samples. For our evaluation, we use a supervised machine learning approach [16].

In supervised learning, an algorithm is used to train a model with input data and its associated output. This process is called training. A trained model predicts the desired output value from new input samples. However, the success of the method relies on expert knowledge in the machine learning discipline, to preprocess the input data and to select the correct model including the optimization of parameters. These tasks are very complicated and time-consuming. Therefore, there is a high demand for automatizing the machine learning process, so the use of *Automatic Machine Learning (AUTOML)* has gained high acceptance in a variety of domains. In our evaluation, the AUTOML library auto-sklearn [17] (based on scikit-learn [18,19]) is used to automate the complex work of machine learning optimization. In a classical ML process, different models and systems are explored until the best is chosen. Auto-sklearn estimates the best performing model out of a range of various classifiers or pre-processors. The training of the model can be time and resource consuming until an accurate model is found. Therefore, the training can be time-limited.

3 Methods

For the prediction of the wall times, machine learning models are trained with historical workload data. A large collection of parallel workload traces is available online [20]. The Parallel Workloads Archive offers workload traces in the standard workload format [21]. Our workload traces enhances this standard workload format. For example, it also offers the initial working directory (IWD) or the jobname. Here, we have to note that these metadata fields may contain privacy-sensitive information which should not be published online.

For our evaluation, we use the recorded workloads from two of the HPC-systems at the Karlsruhe Institute for Technology, the ForHLR I + II [22,23]. These workload traces are generated for the accounting system to track resource consumption. The reason for the job termination is not recorded in these logs, e.g., for technical reasons, by the user, or by the scheduler. It should also be taken into account that there are similar problems with the parallel workload archives as described by Feitelson et al. [24].

Incorrect entries have been removed from our records. Also, jobs with shortly used wall time are eliminated, which indicates technical problems. 177 users remain for the ForHLR I cluster with a total of 169 358 jobs. 107 143 jobs from 135 users stay for the ForHLR II cluster for our evaluation. The used metadata entries from the traces are explained below.

3.1 Job Metadata

The *used walltime* describes the real runtime used by the job in seconds. Jobs with a used wall time less than 60 s are discarded, to ignore jobs with mistakes in the start script. The aim of our approach is to predict a value close to the used wall time, hence, this is the target value for the AUTOML model.

Like mentioned earlier, the human given *requested walltimes* are anything but ideal. The requested wall time is considered because users might use smaller wall times for short test runs and longer wall times for the simulation. The maximum wall time on both machines is three days. The users can omit to specify the requested wall time, then the default value of 10 min is automatically applied. The requested wall time is an integer representing the required time in seconds.

Queues are associated with each job. Usually, the system operators define available resources for the specific queues, e.g., higher priority or specialized hardware. Both clusters offer a "develop" queue with higher priority but reduced maximum wall time. The queue name is converted into a categorical value to make it usable for machine learning.

The requested *taskcount*—the number of requested cores— is a user requested value. Some users tend to use a small task count to test their simulations before they run the real workload. Like wall time, the task count is used as an integer value. For example, Fig. 1 shows a subset of categorized datasets from two users based on the requested wall time and task count. User A with 249 jobs used two different task counts. The used wall time of the jobs are within a small range and can be easily predicted. User B has submitted 214 jobs, all with identical task count. Here, the used wall time is spread over a large range. This shows that the given input values (requested wall time and task count) are not sufficient to predict the job wall time for user B.

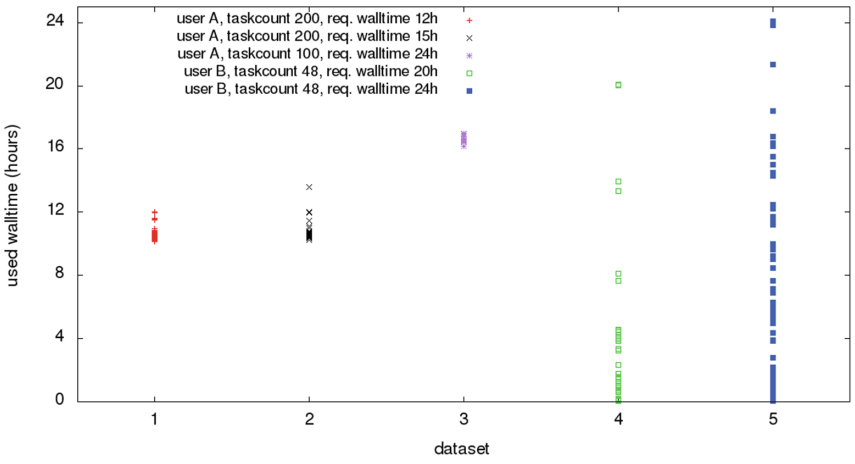


Fig. 1. Categorized dataset for two users, based on the requested walltime and taskcount.

SubmitTime represents the time of job submission, *StartTime* the time when the job starts to execute. The users usually observe jobs submitted during business hours. Submitting jobs right before the weekend result in unobserved runs. If users see unplanned behavior in their job, they will cancel them and restart. While this behavior is very user specific, jobs submitted and started at the the weekend will likely run until the simulation finishes or the requested wall time ends and the job gets canceled by the scheduler. To take this observation into account, the day of the week and the hour of the day was sampled from the SubmitTime and StartTime (Table 1).

Table 1. Example creation of the input matrix for two jobnames.

jobname	myjob	jan	feb	10	16	18
myjob_jan.16-18	1	1	0	0	1	1
myjob_feb.10-18	1	0	1	1	0	1

At least two job parameters are a free chosen string by the user, the *jobname* and the *initial working directory (IWD)*. While some users set a specific jobname for each job, others do not use a jobname at all. There are also some users, which use a specific jobname for specific parts of a work, e.g., preprocessing, simulation, and post-processing. More interestingly, in real-world, it can often be observed, that users organize their jobs by either jobname or the IWD. We split both parameters into smaller components to gather additional information. For the jobname, we use a generic regular expression to split the string by the following delimiter `"_|-| |."`. The split string is then converted to a matrix. For splitting the IWD a regular expression is used to separate the directory path into three components. The first part points to the parallel file systems. This separates jobs using the regular home file system or the optimized and faster scratch file system. The second and third parts contain the directory, where the third part is the basename of the working directory. Table 2 shows a small example, how directories are converted into a matrix for the machine learning.

Table 2. Example creation of the input matrix for directory names.

IWD	/p1/joe-abc	/p3/joe-xyz	sim	data	run_a	data/run	x_1	x_2	x_3
/p1/joe-abc/sim/run_a	1	0	1	0	1	0	0	0	0
/p3/joe-xyz/data/run_a	0	1	0	1	1	0	0	0	0
/p1/joe-abc/data/run/x_1	1	0	0	0	0	1	1	0	0
/p1/joe-abc/data/run/x_2	1	0	0	0	0	1	0	1	0
/p1/joe-abc/data/run/x_3	1	0	0	0	0	1	0	0	1

3.2 Metrics

The built-in metrics from the scikit library [25] are used to evaluate the trained models. Scikit offers several metrics for the regression tasks. The R^2 score (coefficient of determination) provides a metric how well the trained model will predict new samples. It is defined by

$$R^2(y, \hat{y}) = 1 - \frac{\sum_{i=0}^{n_{\text{samples}}-1} (y_i - \hat{y}_i)^2}{\sum_{i=0}^{n_{\text{samples}}-1} (y_i - \bar{y})^2}, \tag{1}$$

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

$$\bar{y} = \frac{1}{n_{\text{samples}}} \sum_{i=0}^{n_{\text{samples}}-1} y_i, \tag{2}$$

where $\bar{y}$ as the average of y_i. The best possible value is 1.0 which corresponds to a perfect prediction. The R^2 score can also be negative and indicates a badly trained model [26]. Other metrics are the mean absolute error (MAE) [27] and the median absolute error (MedAE) [28]. Both measure the difference between predicted and used wall time. MAE is the mean over all pairs of predicted and used wall time,

$$\text{MAE}(y, \hat{y}) = \frac{1}{n_{\text{samples}}} \sum_{i=0}^{n_{\text{samples}}-1} |y_i - \hat{y}_i|. \tag{3}$$

MedAE is the median value of these pairs,

$$\text{MedAE}(y, \hat{y}) = \text{median}(|y_1 - \hat{y}_1|, \ldots, |y_n - \hat{y}_n|). \tag{4}$$

In contrast to MAE, MedAE is robust against outliers. The individual users' historical workload traces are divided into two parts. For this purpose, scikit-learn provides a function that divides the data into a test dataset and a training dataset. The default value is to use 25% as test data and the remaining as training data. A random selection decides which records are added to which set [29]. The training dataset is used to train the machine model, called training set. A high R^2 score for the training data implies that AUTOML was able to find a good model. The other part is used to test the trained model, called test set. A high R^2 score on the test set indicates that the trained model makes good predictions. In our case, this means that the predicted wall time of the job is close to the used wall time. Figure 2 shows the results of AUTOML trained with all above mentioned metadata: requested wall time, task count, initial working dir, jobname, class, start time, and submit time. Each point represents a pair of the R^2 scores from the training and test set for a specific user of the machines (ForHLR I+II). In Fig. 2 an accumulation of the pairs in the right upper half can be seen, which indicate that a well-trained model for most of the users are found. Some low scores for the ForHLR II users indicates, that better model for ForHLR I users are found.

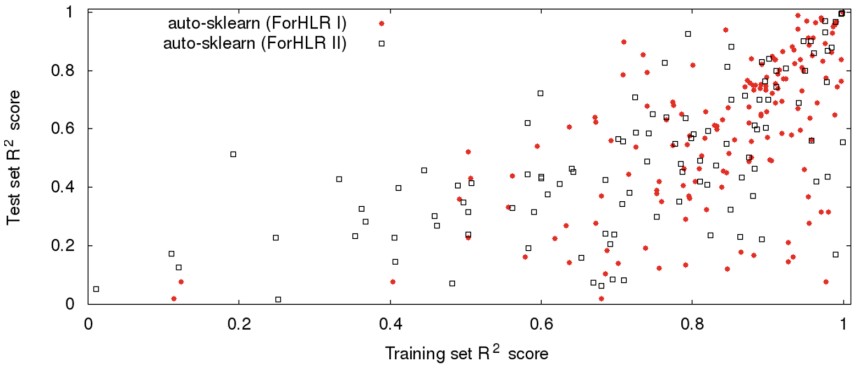


Fig. 2. X-Axis R^2 score on training samples, Y-Axis R^2 score on test samples for ForHLR I+II with 20 min auto-sklearn.

4 Results

The AUTOML model predicted wall times are compared to the user requested wall times. Therefore, a separate model for each user is trained, and then the R^2 score for the prediction calculated. Figure 3 shows a cumulative distribution plot of the accuracies for the models for the different users for the ForHLR I. For the training of the AUTOML model, we used a training set with the requested walltime as metadata. Besides, we extend the requested wall time with other metadata records, e.g., req. wall time + task count, req. wall time + jobname, req. wall time + start time, and so on. Finally, all available metadata are used to train the AUTOML model. Ideally, a curve should be flat at the beginning and rise late (high R^2 prediction scores). In contrast, 80 % of users have a negative R^2 score based on user estimated wall times on ForHLR I. In Fig. 5a and b these results are grouped into four categories of the R^2 scores for the ForHLR I+II. The ranges less 0 and from 0 to 0.5 show a really bad and bad trained model. Whereas in the two ranges from 0.5 to 0.8 and 0.8 to 1 indicate a good and excellent trained model. The four fields have been selected to illustrate the improvement in the individual areas. Adding the different fields of the metadata the number of user in the low R^2 ranges decreases and increases in the high ranges. A model trained with all metadata shows the best results. Similar results for ForHLR II are plotted in Figs. 4 and 5b. Based on user estimated walltimes on ForHLR II over 90 % of the users have a negative R^2 score. In the Table 3 we used the metric MAE (mean absolute error) to present the results based on time. For this purpose, we have grouped the users according to the wall time accuracy. The last line shows the MAE value of the user's requested wall time as the prediction and compares it to the used wall time of the jobs. While only eight users have a mean absolute error less than 30 min, over 127 users are more than 6 h mean absolute error with their requested wall time. While AUTOML achieves even with a few metadata fields good results.

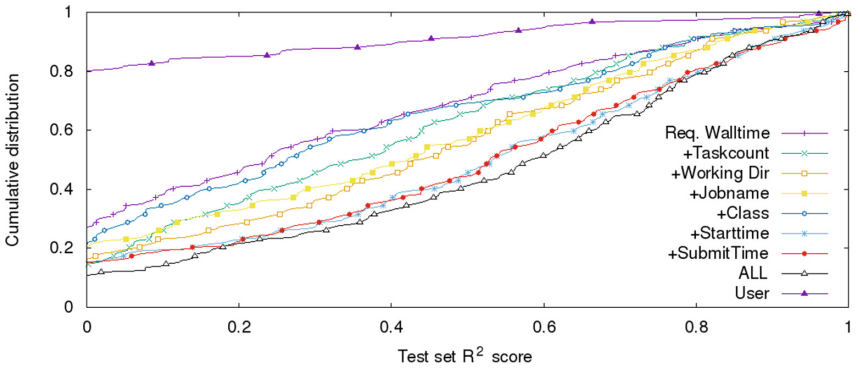


Fig. 3. X-axis R^2 score on test samples, Y-axis cumulative distribution for ForHLR I.

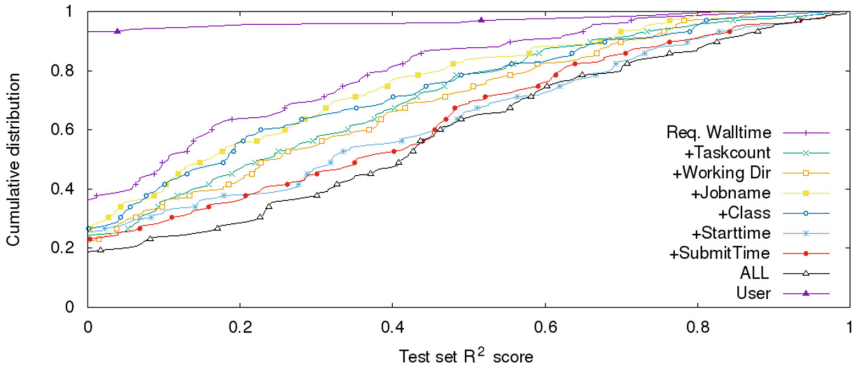
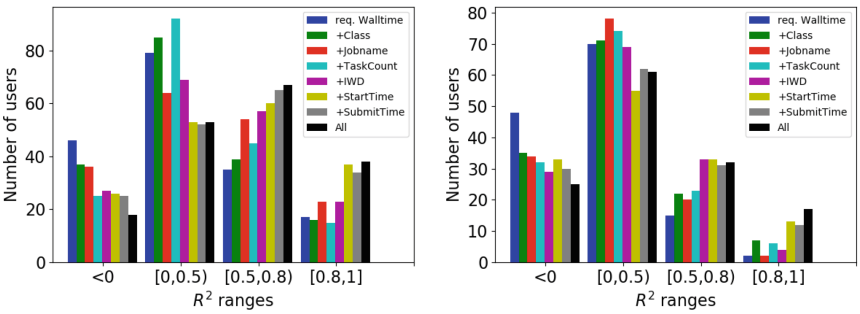


Fig. 4. X-axis R^2 score on test samples, Y-axis cumulative distribution for ForHLR II.

Table 3. Number of users categorized in mean absolute error (MAE) values for the ForHLR I+II.

	ForHLR I				ForHLR II			
Req. Walltime	30min	30min-3h	3h-6h	6h-	30min-3h	30min-3h	3h-6h	6h-
+req. Walltime	22	42	29	84	28	40	25	42
+IWD	28	44	34	71	32	47	14	42
+StartTime	29	49	41	58	30	45	25	35
+SubmitTime	32	47	43	55	31	42	25	37
+TaskCount	28	39	29	81	32	42	18	43
+Jobname	24	45	33	75	30	40	21	44
+Class	26	41	32	78	32	39	19	45
ALL	31	52	42	52	33	46	21	35
User	8	22	20	127	10	21	21	83

It is noticeable that on both machines the start time and submit time make a significant contribution to accuracy. In Figs. 6 and 7 we show the results using the date components of submittime and starttime. For this, we used the requested wall time together with the hour of day component (Fig. 6) of the start time and submit time. The day of the week component is presented in Fig. 7. The third line show the results of all job metadata (requested walltime, taskcount, initial working dir, jobname, class, starttime, and submittime), while using only the date components for submittime and starttime.



(a) Y-axis number of users for ForHLR I (b) Y-axis number of users for ForHLR II.

Fig. 5. Histogram of categories of the R^2 score

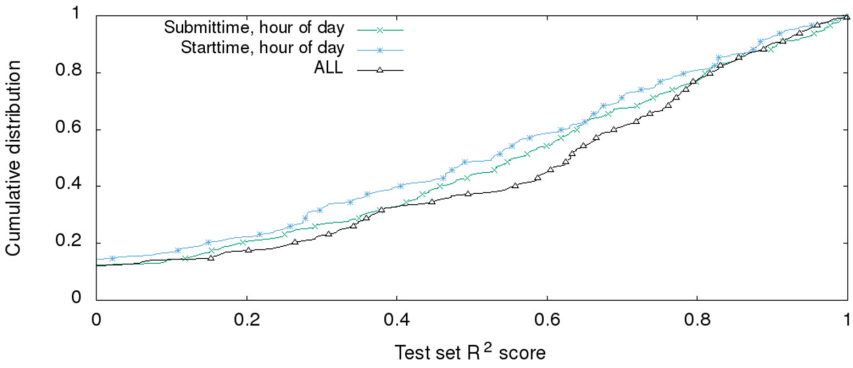


Fig. 6. X-axis R^2 score on test samples, Y-axis cumulative distribution for ForHLR I with only StartTime hour.

All three lines in both figures are very close together, which indicates that the components, hour of day and day of week, from StartTime and SubmitTime provide significant information for the model.

In Figs. 8 and 9 the comparison of the user prediction and AUTOML trained models can be seen. Figure 8 the mean aboslute error (MAE) in hours. Figure 9

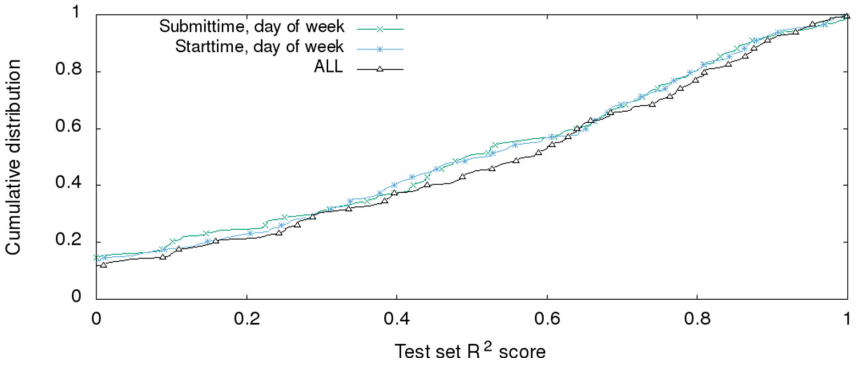


Fig. 7. X-axis R^2 score on test samples, Y-axis cumulative distribution for ForHLR I with only StartTime weekday.

shows the cumulative distribution for the median absolute error (medAE) in hours and in both Figures the horizontal line at 0.6 on the Y-Axis represents 60 % of the users. The user estimations, of both clusters, has a medAE deviation of about 7.4 h. A model trained with AUTOML shows for 60 % of the users a medAE of approximately 1 h on the ForHLR I and 1.4 h for the FORHLR II. Figure 8 shows the difference for 60 % of the users using MAE as a metric. Using AUTOML improves the accuracy from 15.4 h (user estimation) to 4.6 h (AutoML). The accuracy of the predictions for the ForHLR II improve from almost 16 h to only 3 h by AUTOML. Taking into account that only job metadata is used without a knowledge of the payload, this is a good result. More detailed knowledge about the job could result in better predictions, but require in-depth knowledge about the tools and applications of the users. This can not be done in an automated way for the vast number of users.

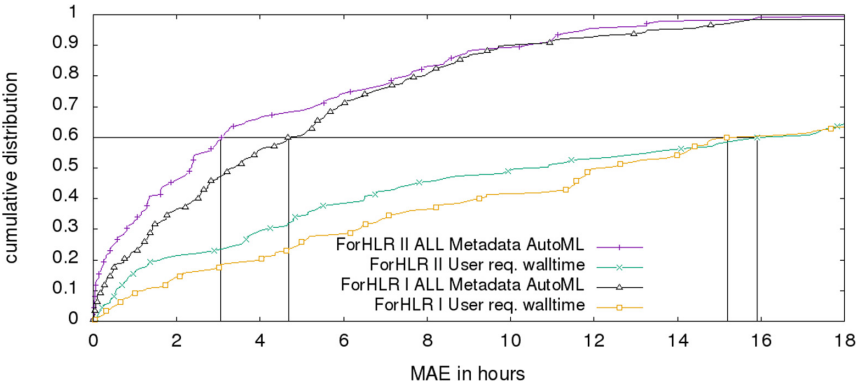


Fig. 8. Comparison of MAE for ForHLR I+II. X-axis mean absolute error in hours, Y-axis cumulative distribution.

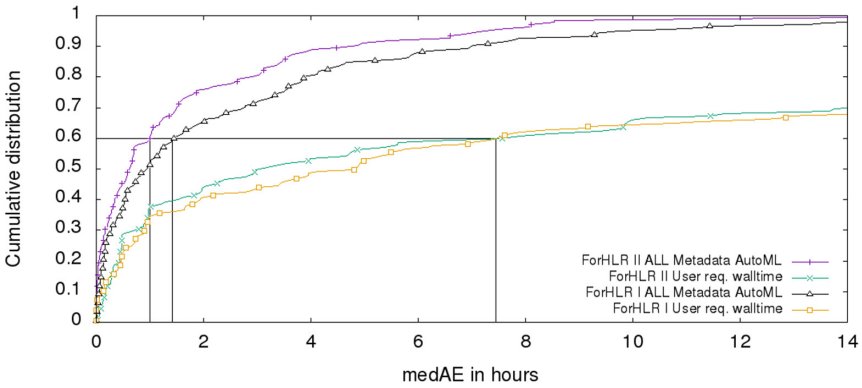


Fig. 9. Comparison of medAE for ForHLR I+II. X-axis Median absolute error in hours, Y-Axis cumulative distribution.

A trained model can be saved to disk for model persistence. This file can be then loaded within seconds and subsequently be used for further predictions [30].

The size of the compressed trained model files are up to 280 MB per user on the ForHLR I and 214 MB on the ForHLR II. The average size is around 23 MB per user. We use a generic regular expression of the job name and the working directory, without any further processing. This leads to a drastic increase in the dimension of the input data. For top users, this results in up to 5000 columns for input parameters. These high numbers are caused by the user giving his directories and jobs names which contain many characters of our regular expression. Analyzing this high dimension can cause the well-known problem "curse of dimensionality" [31,32]. The problem describes issues with high dimensional data. Data with increasing dimension becomes sparse. This sparsity is problematic for any machine learning methods. In our case, a user creating a new directory for every job could make the information of the working directory useless for us, because no more correlations can be recognized as each date is individual. AUTOML use a principal component analysis [33] to reduce the dimension of the input parameter by omitting meaningless columns. This helps to reduce this issue, but this process needs more time and resources to recognize meaningless dimensions. Another approach could be to apply a compression algorithm on these user chosen strings or to discard meaningless names right before training the machine model.

5 Conclusion and Future Work

In this work, we have shown that all the chosen job metadata contains information which improves the wall time predictions. In particular, the previously unnoticed metadata for the initial working directory and jobname provide an additional source of information. Automatic prediction of job wall times is possible without an in-depth analysis of user data and behavior. Good prediction

models can be trained with very simple job metadata without having precise knowledge of the user's work. The expressiveness of the job metadata depends on the operating model of each supercomputer and the way the users use that machine.

A further examination with workload traces from other machines will be conducted in the future, e.g., from other HPC systems in Germany, as our approach uses general job metadata as long as enough metadata like the jobname and initial working directory is available. These are available in most schedulers. The preprocessing of the data is also generic so that no cluster-specific parameters are used.

Providing these predictions to the scheduler is the next step. This means that the scheduler can use the prediction as a basis for its planning; instead, the user requested wall time. The user requested wall time could be used as a guaranteed wall time and the planning could be done with the predicted wall time.

Another necessary investigation is the accuracy of predictions over time. A model that has been trained could become inaccurate over time due to a change of the user behavior, i.e., submitting different workloads and applications in the jobs. Another topic is the fundamental problem with unknown users, known as the cold start. In our approach, a model is trained for each user. There is no model for new users which we can use for prediction. A possible solution for the cold start problem could be an evaluation with a model containing all the job records of all users.

Acknowledgement. This work inside of the project ADA-FS is funded by the DFG Priority Program "Software for Exascale Computing" (SPPEXA, SPP 1648), which is gratefully acknowledged.

References

1. Hovestadt, M., Kao, O., Keller, A., Streit, A.: Scheduling in HPC resource management systems: queuing vs. planning. In: Feitelson, D., Rudolph, L., Schwiegelshohn, U. (eds.) JSSPP 2003. LNCS, vol. 2862, pp. 1–20. Springer, Heidelberg (2003). https://doi.org/10.1007/10968987_1
2. Oeste, S., Kluge, M., Soysal, M., Streit, A., Vef, M., Brinkmann, A.: Exploring opportunities for job-temporal file systems with ada-fs. In: 1st Joint International Workshop on Parallel Data Storage and Data Intensive Scalable Computing Systems (2016)
3. Gibbons, R.: A historical application profiler for use by parallel schedulers. In: Feitelson, D.G., Rudolph, L. (eds.) JSSPP 1997. LNCS, vol. 1291, pp. 58–77. Springer, Heidelberg (1997). https://doi.org/10.1007/3-540-63574-2_16
4. Downey, A.B.: Predicting queue times on space-sharing parallel computers. In: 11th International Proceedings on Parallel Processing Symposium, pp. 209–218. IEEE (1997)
5. Gibbons, R.: A historical profiler for use by parallel schedulers. Master's thesis, University of Toronto (1997)
6. Smith, W., Foster, I., Taylor, V.: Predicting application run times using historical information. In: Feitelson, D.G., Rudolph, L. (eds.) JSSPP 1998. LNCS, vol. 1459, pp. 122–142. Springer, Heidelberg (1998). https://doi.org/10.1007/BFb0053984

7. Smith, W., Taylor, V., Foster, I.: Using run-time predictions to estimate queue wait times and improve scheduler performance. In: Feitelson, D.G., Rudolph, L. (eds.) JSSPP 1999. LNCS, vol. 1659, pp. 202–219. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-47954-6_11
8. Matsunaga, A., AB Fortes, J.: 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, pp. 495–504. IEEE Computer Society (2010)
9. Kapadia, N.H., AB Fortes, J.: On the design of a demand-based network-computing system: the purdue university network-computing hubs. In: Proceedings of the Seventh International Symposium on High Performance Distributed Computing, pp. 71–80. IEEE (1998)
10. Mu'alem, A.W., Feitelson, D.G.: Utilization, predictability, workloads, and user runtime estimates in scheduling the ibm sp2 with backfilling. IEEE Trans. Parallel Distrib. Syst. **12**(6), 529–543 (2001)
11. Nadeem, F., Fahringer, T.: 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, pp. 316–323. IEEE Computer Society (2009)
12. Smith, W.: Prediction services for distributed computing. In: IEEE International Parallel and Distributed Processing Symposium, IPDPS 2007, pp. 1–10. IEEE (2007)
13. Tsafrir, D., Etsion, Y., Feitelson, D.G.: Backfilling using system-generated predictions rather than user runtime estimates. IEEE Trans. Parallel Distrib. Syst. **18**(6), 789–803 (2007)
14. Xsede. https://www.xsede.org/
15. Karnak start/wait time predictions. http://karnak.xsede.org/karnak/index.html
16. Mohri, M., Rostamizadeh, A., Talwalkar, A.: Foundations of Machine Learning. MIT Press, Cambridge (2012)
17. Feurer, M., Klein, A., Eggensperger, K., Springenberg, J., Blum, M., Hutter, F.: Efficient and robust automated machine learning. In: Cortes, C., Lawrence, N.D., Lee, D.D., Sugiyama, M., Garnett, R. (eds.) Advances in Neural Information Processing Systems, vol. 28, pp. 2962–2970. Curran Associates Inc., New York (2015)
18. Pedregosa, F., et al.: Scikit-learn: machine learning in Python. J. Mach. Learn. Res. **12**, 2825–2830 (2011)
19. Buitinck, L., et al.: API design for machine learning software: experiences from the scikit-learn project. In: ECML PKDD Workshop: Languages for Data Mining and Machine Learning, pp. 108–122 (2013)
20. Parallel Workloads Archive. http://www.cs.huji.ac.il/labs/parallel/workload/
21. The Standard Workload Format. http://www.cs.huji.ac.il/labs/parallel/workload/swf.html
22. Forhlr i, kit/scc. https://www.scc.kit.edu/dienste/forhlr1.php
23. Forhlr ii, kit/scc. https://www.scc.kit.edu/dienste/forhlr2.php
24. Feitelson, D.G., Tsafrir, D., Krakov, D.: Experience with using the parallel workloads archive. J. Parallel Distrib. Comput. **74**(10), 2967–2982 (2014)
25. scikit - regression metrics. http://scikit-learn.org/stable/modules/model_evaluation.html#regression-metrics
26. scikit - r2 score. http://scikit-learn.org/stable/modules/generated/sklearn.metrics.r2_score.html#sklearn.metrics.r2_score
27. scikit - mean absolute error. http://scikit-learn.org/stable/modules/generated/sklearn.metrics.mean_absolute_error.html#sklearn.metrics.mean_absolute_error

28. scikit - median absolute error. http://scikit-learn.org/stable/modules/generated/sklearn.metrics.median_absolute_error.html#sklearn.metrics.median_absolute_error
29. scikit - datasset spliting
30. scikit - model persistence. http://scikit-learn.org/stable/modules/model_persistence.html
31. Bellman, R.: Dynamic Programming. Courier Corporation, North Chelmsford (2013)
32. Hughes, G.: On the mean accuracy of statistical pattern recognizers. IEEE Trans. Inf. Theory **14**(1), 55–63 (1968)
33. Pearson, K.: LIII. on lines and planes of closest fit to systems of points in space. Lond, Edinb, Dublin Philos. Mag. J. Sci. **2**(11), 559–572 (1901)