Feature analysis of backfilling heuristics

Vasilii Feofanov
UGA, LIG, INRIA
Grenoble, France
vasilii.feofanov@etu.grenoble-alpes.org

Supervised by: Denis Trystram.

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

Abstract

High Performance Computing (HPC) platforms use a scheduling system that distributes jobs between computational resources. There are numerous heuristics that offer a solution to this problem. However, the behaviour of these heuristics is difficult to explain. The goal of this research is to find features of the system workload which impact on the comparative performance of 12 popular scheduling policies. For this, we generated data using archived workload logs and applied machine learning methods that selected significant features. By this means we are able to find out basic as well as hidden properties that are inherent for the policies under consideration.

1 Introduction

Nowadays, the HPC platforms are widely used for large scale scientific computations. Usually these platforms have to receive a great amount of jobs during one work day. Because of this, a special job management system is embedded which distributes jobs between CPUs of the cluster. This scheduling problem can not be solved in explicit analytical way. Moreover, due to the uncertainty in parameters of the problem, even approximation algorithms are not able to find a solution, today. Therefore, various heuristic policies are used as a solution of the task. In fact, the analysis of policies behaviour is not straightforward. Depending on a workload, policies have different performance from each other for unknown reasons.

With no doubt, the analysis of policies can be useful for understanding what features have an impact the system. This can help in the quest to improve performance of modern HPC platforms. The goal of this research is to find out implicit properties that cause variations in the performance of each policy. This goal is achieved through data analysis. It helps to recover statistically dependency between a workload and behaviour of heuristics.

This problem is analysed in post-hoc way i.e. it is already known how much times each job waited for start and how much times it was executed, since a function of the scheduling policies is used. So, simulation is done using real data, but a posteriori. In this work the data is separated by weeks, so we aim to find features that describes performance of a policy for a given workload.

The rest of the paper is divided into five sections. In Section 2, we consider basic terms and define the scheduling problem. Then, in Section 3, work that related to the research is discussed. Section 4 describes two regression models and corresponding to them wrapper feature selection methods. In Section 5 experiments and their results are described. Section 6 concludes the article.

2 Scheduling Theory

2.1 Formulation of the problem

In this paper, a HPC platform is considered as an union of m identical resources. Over time l independent jobs are submitted concurrently. The number of resources required for job's execution is fixed and given in advance. The aim is to make an execution schedule of jobs and also minimize given criterion of system's performance.

In the problem a job (with some index j) is determined by the following characteristics:

1. actual running time p_j
2. requested by the sender running time $\tilde{p}_j$ with a condition $p_j \leq \tilde{p}_j$
3. a required number of resources q_j
4. release time sub_j, which is a time when the job was submitted
5. time $start_j$ when execution of the job was begun

It is worth to notice that p_j becomes known only after the moment when the job will be completed. That is why $\tilde{p}_j$ is usually much larger then p_j.

2.2 Performance metric

There are many cost metrics that are used to determine performance of a scheduling policy. It can be average waiting time, maximum waiting time, average/maximum of slowdown/bounded slowdown. [Feitelson, 2001; Lelong *et al.*,

2017]. In this paper average waiting time is considered, which is one of the most popular metrics:

$$AvgWait = \frac{1}{l}\sum_{j=1}^{l}\delta_j = \frac{1}{l}\sum_{j=1}^{l}(start_j - sub_j).$$

2.3 Policies

For selection of a job that is going to be executed next, job management system follows a scheduling policy. There are many simple heuristics including the following policies:

- Smallest (Largest) Area First **SAF (LAF)**
- Smallest (Largest) Ratio First **SRF (LRF)**
- Smallest (Largest) Expansion Factor First **SEXP (LEXP)**
- Shortest (Longest) Resource Requirement First **SQF (LQF)**
- Shortest (Longest) Estimated Processing Time First **SPF (LPF)**
- Last (First) Come First Served **LCFS (FCFS)**,

SEXP and LEXP polices use the notion of the job's expansion factor, which is defined as:

$$exp = \frac{wait_j - sub_j + \tilde{p}_j}{\tilde{p}_j},$$

where $wait_j$ is the waiting time of a job until the moment from which it is computed.

Besides policies that just order jobs, there are also backfilling heuristics. Suck kind of policies have a backfilling queue and have an opportunity to allow a first job from the backfilling queue to use resources that had been reserved for a job from primary queue and became free. This approach helps to improve resource utilization. That is why backfilling is popular today.

3 Related Work

Nowadays, in HPC platforms it is widely spread to use a composite scheduling approach: EASY-backfilling heuristic. This method is considered as fast, at the same time preventing job starvation. $EASY - P_R - P_B$ has two job queues: one is for reservation, which is formed by policy P_R and one formed by P_B is for backfilling. Today, EASY-backfilling is being widely studied. One of current interest is to tune parameters of the heuristics [Lelong *et al.*, 2017; Tsafrir *et al.*, 2007]. In this paper we do not try to tune. Instead of we consider heuristics of type $EASY{-}P_R{-}FCFS$, where P_R is one of 12 policies from section 2.3. And for the sake of simplicity, in the rest of paper we will denote a heuristic just by the name of a policy that is used as P_R. In this research, aforementioned 12 policies will be analysed using machine learning methods, that are become popular today in scheduling theory [Gaussier *et al.*, 2015]. As methods we use a classical method: linear regression as well as a modern and widely spread approach: random forest, which is used both for regression and classification problems [Breiman, 2001].

4 Regression analysis

4.1 The Least Square Linear Regression

The simplest way to find dependency is to represent target value as linear combinations of variables cause this value. More precisely, the linear regression model connects a dependent variable y and a vector of independent features $x_1, \ldots, x_p$ in the following way:

$$y = xw + \epsilon,$$

where $x = (1, x_1, \ldots, x_p)^T$, $w = (w_0, w_1, \ldots, w_p)$ is a vector of parameters (weights) of the model and ϵ is noise of the model which is distributed normally in following way [Rencher, 2002]:

$$\epsilon \sim N(0, d^2),$$

where d^2 is a constant value.

Let X be an input matrix which describes n p-dimensional observations. Also, let Y be response vector for the input matrix:

$$X = (x_{ij})_{i=1:n}^{j=0:p} \qquad Y = (y_i)_{i=1:n},$$

where $x_{i0} = 1$ $(i = 1 : n)$. Using this train data and linear regression equation it leads to:

$$Y = Xw + \varepsilon,$$

where $\varepsilon = (\epsilon_1, \ldots, \epsilon_n)$ is a vector of noise values, which should be independent, as the model's noise is a random variable.

The method of least squares seeks to find estimations $\hat{w}$ of model's weights w by minimizing residual sum of squares:

$$RSS = (Y - Xw)^T(Y - Xw) \xrightarrow{w} min$$

The solution of this optimization task is the following expression [Hastie *et al.*, 2009]:

$$\hat{w} = (X^T X)^{-1} X^T Y.$$

4.2 R^2 coefficient

Let $\overline{y}$ be a mean of Y and $\overline{Y} = (\overline{y})_{i=1:n}$. Then, R^2 is defined as [Rencher, 2002]:

$$R^2 = 1 - \frac{(Y - X\hat{w})^T(Y - X\hat{w})}{(Y - \overline{Y})^T(Y - \overline{Y})}.$$

R^2 is some measure quality of the model, which is equal to 1 when $Y = X\hat{w}$ and equal to 0 when RSS is big as the variance of train data.

4.3 F-test

F-test sets null hypothesis H_0 as assumption that all model's coefficients, except for the intercept, are equal to 0. Consider statistic F such that:

$$F = \frac{R^2}{1 - R^2}\frac{n - p - 1}{p}.$$

If H_0 is true, then F is distributed by Fisher distribution with p and $n - p - 1$ degrees of freedom [Rencher, 2002].

4.4 t-test

This test checks the significance of one model's coefficient. Null hypothesis assumes that the coefficient is trivial i.e. equal to 0. If it is true, then statistics t has Student distribution with $n-p-1$ degrees of freedom. Statistic t is defined as [Rencher, 2002]:

$$t = \frac{\hat{w}}{s\sqrt{d_{ii}}},$$

where d_{ii} is i-th diagonal element (counting from 0) of the matrix $(X^T X)^{-1}$ and s is such that:

$$s = \sqrt{\frac{(Y - X\hat{w})^T (Y - X\hat{w})}{n-p-1}}.$$

Summing up aforementioned paragraphs, we get a feature selection approach: build a non-trivial linear regression model and after check by t-test the significance of each coefficient of the model. The advantage of this model is its implementation simplicity and execution rapidity. The fact that the model is linear can be considered as a drawback, because it has low performance in case of non-linear data.

4.5 Decision Regression Tree

As the another approach, one consider non-parametric regression, namely, a decision tree. Assume that there is a partition of feature space $\mathbb{R}^p$ into M parts $\{R_m\}_{m=1:M}$. The regression decision tree model is represented in the following way:

$$f(x) = \sum_{m=1}^{M} w_m I\{x \in R_m\},$$

where $I\{\cdot\}$ is the indicator function and w_m are parameters that minimize sum of squared error:

$$RSS = \sum_{i=1}^{n} (y_i - f(x_i))^2.$$

It can be seen that the minimum of RSS is reached when a weight are equal to mean of all observations that belong to the corresponding part [Hastie *et al.*, 2009]:

$$\hat{w}_m = mean(y_x | x \in R_m).$$

A construction of binary decision tree is done by a greedy procedure. At each step it finds a variable j and splitting point s of j-axis such that:

$$\begin{cases} R_1(j,s)=\{(x\in\mathbb{R}^p)|x_j \le s\}, \quad R_2(j,s)=\{(x\in\mathbb{R}^p)|x_j > s\} \\ \min_{j,s}\left(\sum\limits_{x\in R_1(j,s)} (y_x - \hat{w}_1(j,s))^2 + \sum\limits_{x\in R_2(j,s)} (y_x - \hat{w}_2(j,s))^2 \right), \end{cases}$$

where $\hat{w}_1, \hat{w}_2$ are functions of j and s:

$$\hat{w}_1 = mean(y_x | x \in R_1(j,s)), \quad \hat{w}_2 = mean(y_x | x \in R_2(j,s)).$$

Each time after splitting it makes recursive calls in R_1 and R_2. The process continues until it reaches fixed minimum number of elements n_{min} in a region.

4.6 Random Forest

Despite the fact that decision tree has low bias, at the same time it has high variance [Hastie *et al.*, 2009]. To prevent this limitation random forest algorithm was developed and proposed in [Breiman, 2001]. The main idea of this method is to generate a set of de-correlated trees by means bagging procedure as well as randomly choosing features for splitting at each step. Pseudo-code of the random forest is represented in the algorithm 1.

Input: Training data (X, Y)
for *b = 1 : B* **do**
  $Z_b \leftarrow$ new bootstrap sample derived from (X, Y)
  $T_b \leftarrow$ grow a decision tree using (Z_b, n_{min}) in the following way
    1. Randomly choose $m < p$ features
    2. Determine the best split point's one of m variables
    3. Split the corresponding node in two children nodes.
end
Output: $\{T_b\}_1^B$; $f(x) = \frac{1}{B}\sum_{b=1}^{B} T_b(x)$

Algorithm 1: Random Forest

5 Experiment

5.1 Simulation

In this paper we would like to study the problem in a general way. Because of this as well as the necessity to perform a large number of simulations, which is costly operation, we neglect all topological information of the original machines. One assumes that the processors are considered as indistinguishable items and jobs can occupy any available processor of the system. Experiments in this paper use a mixture of workload logs (Table 1) derived from the Parallel Workload Archive ([Feitelson *et al.*, 2014]), which is a collection of information from different platforms. Due to instability of scheduling metrics [Tsafrir and Feitelson, 2006; Feitelson, 2001] as well as necessity of having large data, each trace has been mixed by resampling procedure proposed in [Feitelson, 2016]. For experiment's simulation a lightweight workflow system zymake [Breck, 2008] is used, which is just a build system with workflow capabilities. As policies we use EASY-backfilling heuristics (section 3) with one of 12 backfilling policy from section 2.3.

5.2 Data design

After simulation we have data that includes workload logs and the average waiting time of policies for each week. One

Name	Year	#MaxProcs	#Jobs	Duration
KTH-SP2	1996	100	28k	11Months
SDSC-SP2	2000	128	59k	24Months
SDSC-BLUE	2003	1152	243k	32Months

Table 1: Workload logs used for experiments

week is considered as a scale unit, since we do not want to take into account the difference in workload between the beginning and the end of the week. Because of this, one issue is arose: how to define features of each week. As a solution, it has been decided that one observation we formed by considering descriptive statistics of jobs characteristics. In this paper we consider such statistics as average, maximum, minimum, median, 10% and 90% quantiles. As characteristics of a job we regard following values:

1. Requested run-time $\tilde{p}$
2. Actual run-time p
3. Number of resources q
4. Area: $a_j = p_j \cdot q_j$ (or $\tilde{p}_j \cdot q_j$)
5. Ratio $rat_j = p_j / q_j$
6. Requested time over run-time $rat_j = \tilde{p}_j / p_j$

Usually, submission time is represented as cumulative over days value. But, in fact, the variable has a periodic property, which is desired to get. For this reason transformation to polar coordinates is used [Bishop, 2006]. Thence, two more characteristics can be derived:

$$7.\ \cos\left(2\pi \frac{sub_j\ \%\ spd}{spd}\right) \qquad 8.\ \sin\left(2\pi \frac{sub_j\ \%\ spd}{spd}\right),$$

where spd is number of seconds in one day and $\%$ is the modulus operator.

5.3 Results of the experiment

In this paragraph we apply regression analysis investigating the workload data. Features described in previous paragraph form input data, whereas a policy's vector of average waiting time values is considered as the target value of regression model. Hence, our mathematical model under analysis is a pool of 12 independent regressions. However, we believe that in this case the analysis is not solid since we do not take into account how other polices behave during the same week. That is why we decided to apply value shifting relatively average of all performance results for each week. The relative average waiting time is better to consider in terms of classification study that is discussed in Section 6. On the figures 1 and 2 performance values before and after transformation are illustrated.

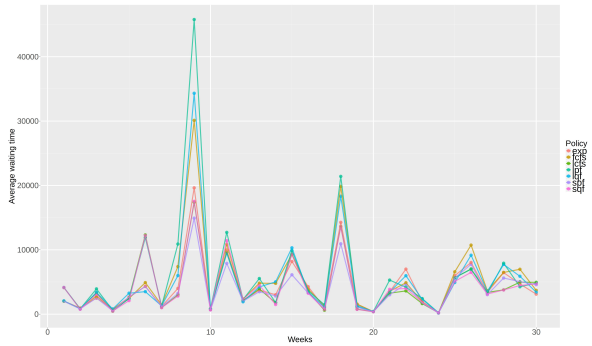

Figure 1: Average waiting time before shifting

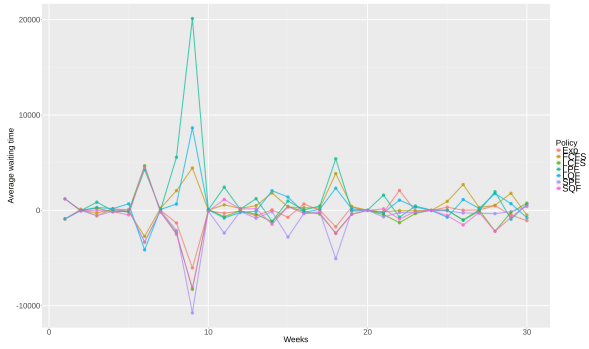

Figure 2: Average waiting time after shifting

We apply two approaches of feature selection: t-test for linear regression model and feature importance for random forest. Results that we got for the trace KTH-SP2 are illustrated in the figure 3 and 4. A green cell means that a variable was accepted as significant and red cell indicates the opposite conclusion. Results of selection for other traces it is possible to view in a github repository of this project [1].

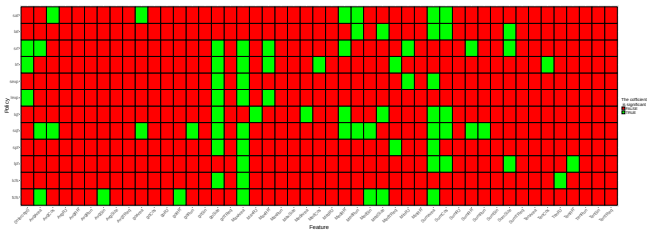

Figure 3: Results of t-test.

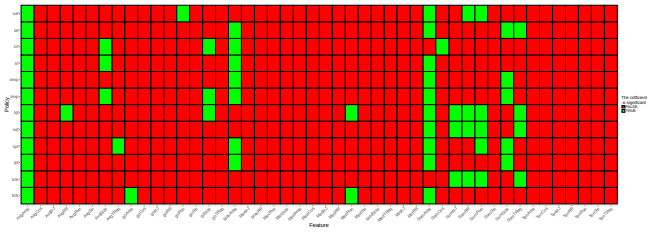

Figure 4: Results of random forest's feature importance

5.4 Analysis of results

Looking at figures 3 and 4 we can see some basic trends that are inherent for a majority of heuristics. In a table 2 it is shown features that was chosen by at least 4 policies for linear regression and random forest. It can be seen that both models indicates that 90% quantile of size, maximum area and sum of area are significant in most cases. Indeed, this fact has a

[1] https://github.com/Stonver/Feature-analysis-of-backfilling-heuristics

Linear Regression	Random Forest
	Average Area
Sum of Area	Sum of Area
Maximum Area	Maximum Area
Median of Ratio (RR)	Sum of Requested Time
Sum of Cosine	Sum of Run-Time
90% quantile of size	90% quantile of size
	Average Size

Table 2: Comparisons of selection results

Policy	Linear Regression	Random Forest
saf	11717178	10716816
laf	5097887	3616036
srf	3896098	3744861
lrf	2779184	2295166
sexp	2587307	1857527
lexp	4493848	3424834
lqf	2553505	2670536
sqf	5597682	5231832
spf	6820213	5556439
lpf	4650828	3224698
lcfs	3162729	3015185
fcfs	2231523	2014479

Table 3: Cross-validated residual sum of squares

simple interpretation. Large-size or large-area jobs increase average waiting time, so, they influence on performance significantly. The same is for sum of area which characterize the scale of workload volume.

Overall, we see that results of models differ from each other. In fact, it is needed to compare the quality of two methods. On the table 3 residual sum of squares of two approaches are illustrated. These values were computed by 5-fold cross-validation. Also, we can conduct F-test and compute R-squared for linear regression. The results are depicted on the figure 5. It can be seen that all linear regression models are significant. However, R-squared values are quite low. Probably, the reason of low values is non-linearity of the data. Also, comparing residual sum of squares of linear regression and random forest, it can be noticed that overall random forest outperforms linear regression by RSS. Hence, we can deduce that random forest gives us more reliable results of selection, which it is advised to take into consideration more. In the figure 6 feature importances derived from random forests are illustrated.

	saf	laf	srf	lrf	sexp	lexp	lqf	sqf	spf	lpf	lcfs	fcfs
F-test passed?	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
R-squared	0.40	0.51	0.27	0.33	0.37	0.28	0.23	0.30	0.34	0.47	0.15	0.30

Figure 5: F-test and R-squared of linear regression models

6 Conclusion

In this paper a statistical approach of feature analysis for scheduling heuristics is proposed. By this means we have

	Sum of Area	Max Area	Average Area	Sum of Run-Time	Sum of Req.Time	90% quantile of Size	Average Size
saf	**0.26**	0.02	0.15	0.08	0.03	0.02	0.01
laf	**0.42**	0.06	0.12	0.03	0.05	0.00	0.02
srf	0.02	0.07	**0.13**	0.04	0.02	**0.13**	0.12
lrf	**0.22**	0.07	0.19	0.03	0.01	0.03	0.07
sexp	**0.26**	0.13	0.13	0.03	0.01	0.03	0.04
lexp	**0.14**	**0.14**	0.09	0.03	0.01	0.08	0.08
lqf	0.06	0.00	0.06	0.09	0.07	**0.10**	0.03
sqf	0.18	0.01	0.08	**0.27**	0.14	0.05	0.01
spf	**0.27**	0.09	0.15	0.05	0.05	0.01	0.04
lpf	**0.39**	0.07	0.18	0.02	0.02	0.01	0.03
lcfs	0.02	0.00	0.06	**0.17**	0.11	0.01	0.01
fcfs	**0.25**	0.00	0.09	0.03	0.03	0.01	0.02

Figure 6: Feature importances of random forests

derived basic properties that influence the heuristic's performance as well as features that are important for concrete policies (Figure 3, 4, Table 6). Besides this, we also can deduce another fact. Except for aforementioned statistical conclusions, the feature analysis also helps reduce the number of model's parameters more than twice. This can be useful for another problem that we study at the same time. We noticed that the performance of the system can be improved if we will take a policy that is optimal for current week instead of using fixed one at all time. Figure 7 demonstrates this remark. The AverOfPerf column shows average waiting time of each policy among all weeks. The AvgBest column describes average waiting time of policies when they were best among all others. It can be seen that performance can be improved significantly if we are able to predict each week the best policy. Because of this, our future work is to build an online-classifier that would contextually select a scheduling policy for outgoing week. And for this feature space reduction can be extremely helpful.

policies	NumTimeBest	AverOfPerf	AvgBest
saf	6	7186.057	4337.484
laf	206	3561.800	3693.638
srf	3	5554.492	1934.373
lrf	31	4568.366	2586.903
sexp	21	4480.570	2096.751
lexp	6	5473.200	2096.105
lqf	54	4606.059	1992.889
sqf	15	6009.632	2318.681
spf	5	6260.025	1880.955
lpf	104	3775.828	3890.253
lcfs	3	5671.536	7558.707
fcfs	46	4533.650	2110.407

Figure 7: Results of random forest's feature importance

References

[Bishop, 2006] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[Breck, 2008] Eric Breck. Zymake: A computational workflow system for machine learning and natural language processing. In *Software Engineering, Testing, and Quality Assurance for Natural Language Processing*, SETQA-NLP '08, pages 5–13, Stroudsburg, PA, USA, 2008. Association for Computational Linguistics.

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

[Feitelson *et al.*, 2014] 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.

[Feitelson, 2001] Dror G. Feitelson. *Metrics for Parallel Job Scheduling and Their Convergence*, pages 188–205. Springer Berlin Heidelberg, Berlin, Heidelberg, 2001.

[Feitelson, 2016] Dror G. Feitelson. *Resampling with Feedback — A New Paradigm of Using Workload Data for Performance Evaluation*, pages 3–21. Springer International Publishing, Cham, 2016.

[Gaussier *et al.*, 2015] Eric Gaussier, David Glesser, Valentin Reis, and Denis Trystram. Improving backfilling by using machine learning to predict running times. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '15, pages 64:1–64:10, New York, NY, USA, 2015. ACM.

[Hastie *et al.*, 2009] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The elements of statistical learning: data mining, inference and prediction*. Springer, 2 edition, 2009.

[Lelong *et al.*, 2017] Jérôme Lelong, Valentin Reis, and Denis Trystram. Tuning EASY-Backfilling Queues. In *21st Workshop on Job Scheduling Strategies for Parallel Processing*, 31st IEEE International Parallel & Distributed Processing Symposium, Orlando, United States, May 2017.

[Rencher, 2002] Alvin C. Rencher. *Methods of Multivariate Analysis*. Wiley Series in Probability and Mathematical Statistics. John Wiley & Sons, Inc., Newark, New York, 2nd edition, 2002.

[Tsafrir and Feitelson, 2006] Dan Tsafrir and Dror G. Feitelson. Instability in parallel job scheduling simulation: the role of workload flurries. In *20th International Parallel and Distributed Processing Symposium (IPDPS 2006), Proceedings, 25-29 April 2006, Rhodes Island, Greece*, 2006.

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