Improving Backfilling using Learning to Rank algorithm

Jad Darrous
Supervised by:
Eric Gaussier and Denis Trystram
LIG - MOAIS Team

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

Abstract

While High Performance Computing (HPC) systems became more and more available and widespread, scheduling in these systems faces many challenges that prevent schedulers from reaching the desired performance. Improving scheduling performance leads to an immediate improvement on the HPC environment which means higher throughout and shorter response time. Scheduling is a hard problem and no straightforward solutions exist. Knowing that a lot of historical data is available, a Machine Learning technique called *Learning to Rank* is employed to optimize the jobs priorities in the queue of the EASY-Backfilling scheduler. We got an improvement up to **68%** in respect of the Average Bounded-Slowdown scheduling measure. This work is the first milestone toward a new approach for improving schedulers in a HPC environment using Machine Learning.

1 Introduction

Scheduling in High Performance Computing (HPC) systems is a critical and important part in the global performance of a Resource Management Software (RMS). The RMS is the middleware that lays between the users and the cluster hardware to manage the allocation of resources to the submitted jobs. In the past decades, many researches have been conducted to improve the scheduling algorithms. Multiprocessor scheduling is NP-Hard problem, thus no optimal algorithm exists to find the optimal solution in polynomial time. As a consequence, the proposed algorithms use heuristics to achieve the desired performance.
Improving the scheduler performance has many direct effects on the response time, energy consumption and resource utilization, to name a few.

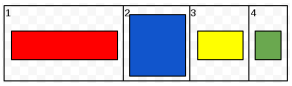
In recent years, more and more machine learning technologies have been used to train ranking models, and a new research area named *Learning to Rank* (L2R) has gradually emerged. Learning to Rank has a wide range of applications in Information Retrieval, Natural Language Processing, and Data Mining. Learning to rank task is to automatically construct a ranking model using training data, such that, the model can sort new objects according to their degrees of relevance, preference, or importance. Therefore, it is used extensively by search engines to order the response documents of a user's search query. We used the same concept and adapt it in the field of scheduling in order to rank jobs for a scheduler in a way that maximizes its performance.

In this work, we try to incorporate a ranking technique borrowed from the Machine Learning and IR domains into the scheduling domain to improve the performance of a scheduler.

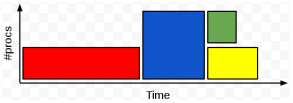
2 Problem description

First Come First Served (FCFS) [Schwiegelshohn and Yahyapour, 1998] is the most basic batch scheduling algorithm where jobs are considered in order of arrival. In other words, the job that is submitted first should run first. Each job specifies the number of processors it requires and is placed in a First-In-First-Out (FIFO) queue upon arrival. When a sufficient processors are available, the scheduler start the job at the head of the queue. Otherwise, it blocks until its requirement is fulfilled. As a consequence, many computational resources are wasted.

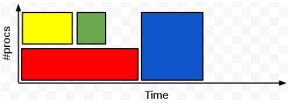
Backfilling is a scheduling optimization that allows a scheduler to make better use of available resources by running jobs out of order. It requires that each job specifies its maximum execution time, normally done by the user. The scheduler can determine the earliest time when the needed resources will become available for the highest priority job to start. Consequently, it can also determine which jobs can be started without delaying this job.
By letting some jobs execute out of order, other jobs may get delayed. Backfilling will never completely violate the FCFS order where some jobs are never run (a phenomenon known as "starvation"). In particular, jobs that need to wait are typically given a *reservation* for some future time.
Enabling backfilling allows the scheduler to start other, lower-priority jobs as long as they do not delay the highest priority job. Backfilling offers significant performance improvement by filling in holes, favoring short and small jobs.



(a) The waiting queue of jobs



(b) Without backfilling



(c) With backfilling

Figure 1: The backfilling mechanism

A simple schema for the backfilling is presented in Figure 1. The waiting queue of the scheduler is illustrated (Figure 1(a)). Without backfilling, the jobs are executed in order of arrival (Figure 1(b)). Enabling backfilling will allows some jobs to start execution before its order if no delay will be caused to other jobs (Figure 1(c)). With backfilling, idle processors can be exploited.

Backfilling, in which small jobs move forward to utilize the idle resources, was introduced by Lifka [Lifka, 1995]. This was done in the context of EASY, the Extensible Argonne Scheduling sYstem, which was developed for the first large IBM SP1 installation at Argonne National Lab.

The order of selecting jobs from the waiting queue is FIFO order. Other criteria (e.g. priorities) can be employed. In this work, we try to learn the best order for jobs to be scheduled in the waiting queue. And this done by learning the best way to rank jobs using Learning to Rank algorithm.

3 Related work

While the concept of backfilling is quite simple, many variations with different parameters have been proposed and showed a good performance improvement.

One variation is related to the *n*umber of reservations. The original backfilling with reservation do one reservation for the first job in the queue. But the backfilled job may cause a delay for jobs in the waiting queue other than the first one. To handle this situation a reservation for all the jobs is made, which is called "conservation reservation" but this method lead to performance degradation. In the middle of these two extremes, Chiang et al. suggest that making up to four reservations is a good compromise [Chiang *et al.*, 2002].

Another parameter is the *o*rder of queued jobs. The original EASY scheduler, and many other systems and designs, use a first come, first served (FCFS) order [Lifka, 1995]. Flexible backfilling combines three types of priorities: an administrative priority set to favor certain users or projects, a user priority used to differentiate among the jobs of the same user, and a scheduler priority used to guarantee that no job is starved [Talby and Feitelson, 1999]. The Maui scheduler has a priority function that includes even more components [Jackson *et al.*, 2001].

A final parameter is the *a*mount of look ahead into the queue. All previous backfilling algorithms consider the queued jobs one at a time, and try to schedule them. But the order in which jobs are scheduled may lead to resources fragmentation. The alternative is to consider the whole queue at once, and try to find the set of jobs that together maximize desired performance metrics. This can be done using dynamic programming, leading to optimal packing and improved performance [Shmueli and Feitelson, 2003].

One of the assumptions for backfilling is that the runtime of the jobs is already known. Runtime estimation is usually done by the user when submitting the job. Jobs are threatened to be killed if they run more than their estimated runtimes. Predicting runtime, using Machine Learning algorithms, rather than estimating it by users shows an improvement for the backfilling even with simple prediction as the average of the two last available running times of the user's jobs [Tsafrir *et al.*, 2007]. A paper from my team MOAIS is about to be published in this domain also.

Because of the advantages it offers, Learning to Rank has been gained increasing attention especially in the past several years, and it become one of the most active research areas in Information Retrieval (IR). Many methods have been proposed and applied to IR applications (e.g., [Cao *et al.*, 2006] and [Cohen *et al.*, 1999]). A public datasets are also released by Microsoft Research [Qin *et al.*, 2010] by the LEROT project to be as a benchmark dataset for the research in this domain, in addition to about 100 of research papers have been listed on the project website[1]. To be applicable on big datasets, Google research publish a paper [Sculley, 2009] to minimize the complexity of pairwise learning. Yahoo! also organized a challenge for Learning to Rank [Chapelle and Chang, 2011].
L2R is also employed in other domains. In Content-based Image Retrieval (CBIR), the algorithm is used to accurately rank the returned images [Faria and Veloso, 2010]. Another application is for contextual advertising to predict the probability that users will click on ads [Tagami *et al.*, 2013].

An important part of EASY-BF is ordering. In the original EASY, the FCFS used as a priority to select backfilled jobs. In EASY++, Shortest Job First (SJF) is used [Tsafrir *et al.*, 2007]. Considering the big available historical data about clusters workloads, and knowing that the problem of finding the best priority is a Hard problem (i.e. no polynomial algorithm exist) employing a learning algorithm to solve the problem is a potential candidate. To the best of our knowledge, there is no other work that investigates using ranking algorithms to order jobs in the waiting queue of the EASY scheduler.

4 Materials and Methods

4.1 Machine Learning

Machine Learning is the set of algorithms that are used to solve complex problems that do not have a specified algorithm to solve them. For example: handwriting recognition, face detection, plagiarism detection, stock market prediction.. and many others. Learning algorithms learn from data; they try to find a mapping function between the input and output that fit the available data (training set) and performs well on

[1] http://research.microsoft.com/en-us/um/beijing/projects/letor/paper.aspx

the newly unseen data (test set). The data set is a set of instances. Each instance is represented as a vector of features. The data set is split into training set that will be used in the training phase to learn the hypothesis, and a test set that will be used to evaluate the generated hypothesis. The parameters that are used to generate the hypothesis that perform the best on the training set are used to build the hypothesis again on the complete data set. The data is a key success for learning algorithms, so preparing and pre-processing the data is a critical part in the final results.

The two main and broad categories in Machine Learning are *Supervised Learning* and *Unsupervised Learning*. In the supervised learning, a single target value is associated with each instance. if the target value belongs to a finite set; the problem is called classification. if the target value is a real number; it is called regression. The goal is to predict the target value for the new data. On the other hand, no target value is associated with each instance in the unsupervised learning. The algorithm groups similar instances together according to their features. The new data is assigned to the most similar group. The algorithms in this categories are called clustering algorithms.

Learning algorithms have two learning techniques: *Batch learning* and *Online learning*. In the batch learning (offline learning) the hypothesis is generated from the available data and used later to predict or classify the new data and the hypothesis remains unchanged. In another way, in the online learning the hypothesis is build and updated for each new available instance, so it can adapt for the new data and getting ride of the training phase at the beginning. This technique is the only solution in cases where data become available sequentially.

4.2 Learning to Rank algorithm

Learning to rank is a supervised learning task to generate a ranking model. The training data consists of lists of items with some partial order specified between items in each list. This order is typically induced by giving a numerical or ordinal score for each item. The purpose of the ranking model is to rank, i.e. produce a permutation, of items in new, unseen lists in a way which is "similar" to rankings in the training data in some sense. Learning to rank is mostly used to rank documents according to a query in which the relative ranking between the retrieved documents is more important than the absolute importance of each document. Learning to rank problems are categorized into three categories by their input representation and loss function: Pointwise, Pairwise and Listwise. In the point-wise approach the ranking problem is reduced to regression or classification on single objects. Similarly, in the pair-wise approach the ranking problem is reduced to pair-wise classification problem. But in the list-wise approach, the algorithm tries to minimize ranking losses directly on the input list. A more extensive introduction about the Learning to Rank algorithm is presented by *H*ang LI [Li, 2011].

Pairwise approach

The pairwise approach does not focus on accurately predicting the relevance degree of each item (as the pointwise does); instead, it cares about the relative order between two items. In this sense, it is closer to the concept of "ranking" than the pointwise approach.
In the pairwise approach, ranking is transformed into pairwise classification. The algorithm takes as input a set of ordered items, each item is of the form $(\mathbf{x}, y)$ where $\mathbf{x} \in \mathbb{R}^n$ represent the item in a n-dimensional space and $y \in \mathbb{N}$ denoting its rank, and the goal is to find a weight vector $\vec{w}$ that gives a higher score for an item a ordered before another item b. That is, the goal of learning is to minimize the number of miss-classified items pairs. Items from different groups are not comparable. To do that, the algorithm build the item $a-b$ and tries to classify it either as positive instance $(+1)$ or as negative instance (-1). This classification can be done using any classical supervised algorithm.

Ranking SVM

Many previous studies have shown that Ranking SVM [Herbrich *et al.*, 2000] [Joachims, 2002] is an effective algorithm for ranking. The algorithm was published by Thorsten Joachims in 2002 [Joachims, 2002]. Ranking SVM generalizes SVM to solve the problem of ranking: while traditional SVM works on documents, Ranking SVM adopts partial-order preference for document pairs as its constraints. The optimization formulation of Ranking SVM is as follows:

$$min(\frac{1}{2}w^T w + C\sum_{i,j,k}\varepsilon_{i,j,q})$$

4.3 Learning to Rank + Backfilling

In this work, we apply a ranking algorithm to find the best priority that orders the jobs in the waiting queue according to an objective function. The best priority is learned by feeding the algorithm with the simulated workload that gives the best value of the objective function after simulating it with different priorities. As a result, a score will be assigned to each job. This score will be used later as a priority indicator for selecting jobs from the waiting queue to be backfilled.

Training data preparation

The training data set for the L2R algorithm should be provided as set of ordered lists (i.e. queries, in the IR domain) these lists should be independent and identically distributed (i.i.d.). For the workload log files, the splitting is done by cutting the training set into multiple continuous lists each contains the same number of jobs. shuffling the training set before splitting it does not preserve the scheduler state so the training lists will not consistent for training.

Features selection

Each job is represented as a vector in the space R^5. We decide to represent each job by 5 features: Submit Time, Wait Time, Run Time, Number of Allocated Processors and Requested Time. More details about these features can be found in The Standard Workload Format website[2]. All the used features are normalized to be in range $[0..1]$, feature normalization leads to a faster convergence and let all the feature contribute equally to the predicted value.

[2]http://www.cs.huji.ac.il/labs/parallel/workload/swf.html

4.4 The objective function

One of the commonly used measures in scheduling is the *Slowdown* or *stretch*, which is defined for a job as follows:

$$sld = \frac{T_w + T_r}{T_r}$$

where T_w is the waiting time of the job (in the queue of the scheduler - the time between submission and the start of execution) and T_r is the running time for that job. But, by this measure, short jobs with reasonable delay have too large slowdown values. For example, a 1 second job delayed for 20 minutes suffer from a slowdown of 1200. An improved version is the *bounded slowdown* as suggested by [Feitelson *et al.*, 1997], which is defined as follows:

$$bsld = max\left\{\frac{T_w + T_r}{max\{T_r, \tau\}}, 1\right\}$$

where T_w and T_r are the same as before, and τ is a constant. In literature, τ is generally set to 10 seconds. We will use this value in the experiments. For the previous example, we got a bounded slowdown of 120.
To apply this measure on the complete workload with N jobs, we take the *Average bounded slowdown*:

$$AVGbsld = \frac{1}{N}\sum_i bsld(job_i)$$

5 The proposed algorithm

In this section, we present the logical flow of the algorithm. As most of batch Machine Learning algorithm, there are two main phases: the training phase and the testing or evaluation phase.

The training phase is done offline to generate the hypotheses. This hypotheses is used later to give a score to the jobs. The training phase begin by extracting and preparing the data set as described above. After that, each list is simulated using the MAUI scheduler (EASY + backfilling) with different priorities to select from the waiting queue; In real life, the weights that form the priorities are usually selected randomly, we select the weight for FCFS and Slowdown among others because they give better results. The list that gives the best value for the used objective function after simulation, is selected. A relative score is given for each job in the selected lists; the earlier the job is finished, the higher score is given. The selected lists are fed to the Learning to Rank algorithm. As a result, the hypotheses (or ranking model) is generated. This hypotheses is used later for scoring jobs.

For the testing phase; First, the test set is simulated using all the priorities as before and the best value for the objective function is considered. Then, a relative score is given for each job by the hypotheses. The test set is simulated again using the score as priority for the backfilling.

6 Experiments and Evaluation

6.1 Testbed

The workload logs used in the experiments, presented in table 1, are extracted (Expect Metacentrum) from the *P*arallel Workloads Archive [Feitelson *et al.*, 2014] and the cleaned version is used as it the recommended one. Additional filters applied to the available cleaned versions to remove dirty jobs that have missing fields or inconsistent fields: requested cores greater than max processors, runtime less than zero, submit time less than zero and required time greater than runtim. Metacentrum is extracted from the personal website of Klusáček[3]. Since the workloads were generated at different sites, by machines with different sizes and reflect different load condition, we can say that the following results are truly representative.

The simulation done by a fork from the open source batch scheduler simulator *p*yss[4]. The source of this simulator, the source of the implemented algorithm and all the experiments are available online[5].

For the ranking algorithm, the SVM^{rank} library[6] is used; a C library that implement an efficient Ranking SVM algorithm as described in [Joachims, 2006].

Table 1: Workloads used in the experiments

Name	Year	#CPU	#Jobs	Duration
KTH SP2	1996	100	28k	11 Months
CTC SP2	1996	338	77k	11 Months
SDSC SP2	2000	128	59k	24 Months
SDSC BLUE	2003	1,152	243k	32 Months
CEA CURIE	2012	80,640	295k	3 Months
Metacentrum	2013	3,356	495k	6 Months

6.2 Learning parameters

The provided workload files are split into training set and test set by the ratio 70:30. The training set is split again into multiple lists (e.g. 128, 256, 512, 1024 ..). We select the number of lists that gives the best value for the objective function, a discussion about the relation between the number of lists and the AVGbsld value obtained comes later. Table 2 shows the number of partitions for each workload used during experiments.

Table 2: Number of training lists

Name	#Training lists
KTH SP2	256
CTC SP2	256
SDSC SP2	512
SDSC BLUE	1024
CEA CURIE	1024
Metacentrum	1024

6.3 Experiments flowchart

Figure 2 shows a flowchart of the experiments. The starting point is the desired workload and the final output is the per-

[3] http://www.fi.muni.cz/ xklusac

[4] pyss - the Python Scheduler Simulator, available at https://code.google.com/p/pyss/

[5] https://github.com/jad-darrous/predictsim

[6] http://www.cs.cornell.edu/people/tj/svm_light/svm_rank.html

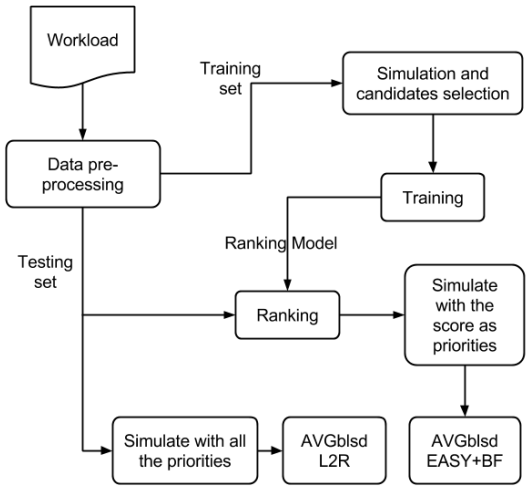


Figure 2: Flowchart of the experiments

formance gain using ranking algorithm compared to EASY + BF.

6.4 Results

Table 3 presents a comparison of the AVGbsld value obtained using EASY+BF and L2R. The presented values are measured on the test set. The EASY+BF column is the standard EASY+BF scheduler, the L2R column contains the value of the objective function when using the score of the ranking algorithm as a priority. The last column represents the improvement percentage.

Table 3: Average Bounded-Slowdown on Test set

Name	EASY+BF	L2R	Improvement
KTH SP2	34.74	21.45	61.9%
CTC SP2	35.48	21.13	67.9%
SDSC SP2	82.24	60.82	35.2%
SDSC BLUE	29.60	17.53	68.8%
CEA CURIE	24.08	2.65	809.9%
Metacentrum	10.14	10.14	-0.1%

Figure 3 shows the relation between the number of training lists and the AVGbsld measured for the workload *SDSC BLUE*. in Figure 4, the time needed for the training phase is presented as a function of the number of training lists for the log *SDSC BLUE*.

6.5 Discussion

The previous table shows a large variance of improvement. Usually in Machine Learning algorithms, the more data available for training the more the algorithm learn the underlying model and a higher improvement is got. Unless for the last workload, The improvement is increased according to the size of the workload. The huge improvement on the log *CEA CURIE* and the no improvement of *Metacentrum* can be partially explained by the used features for the learning and the

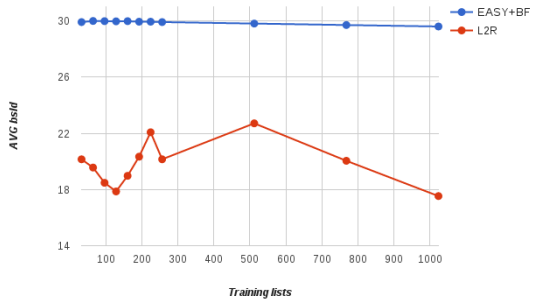


Figure 3: AVGbsld computed for different numbers of training lists of the SDSC BLUE log

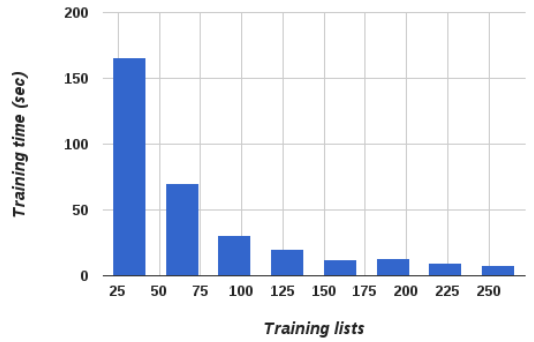


Figure 4: The execution time needed for training as a function of the number of training lists of the SDSC BLUE log

structure of these logs, but the accurate explanation will remain an open question that needs more experiments and in depth analysis of the workloads.

Increasing the number of lists for the training set does not have a big impact in terms of AVGbsld, it is just the training time that changes; the more training list, the shorter training time (exponential function) as seen in Figure 4. The time is measured in seconds but the relative values is the more important than absolute values of time.

Later, experiments have been done to use the generated hypotheses from one workload with other one to find out if the same hypotheses many be used with different workload. The results was as expected; the performance is decreased. This can be explained by considering the fact that the characteristics of the training jobs are different from that of the jobs in the test set. As a consequence, no general hypotheses is found by our experiments that can adapt and increase the performance on other workloads.

7 Conclusions and Perspectives

Scheduling in HPC is an active research area for its critical role in the currently available large-scale clusters. To improve the scheduler performance, we proposed to use a Machine Learning algorithm called Learning to Rank, mostly used in the IR domain, to rank jobs to be backfilled in the waiting queue. Our experiments show an improvement up to **68%** on average and an improvement up to **809%** times on CEA-

Curie workload. The results are reported according to the Average Bounded-Slowdown as an objective function, and compared with EASY+BF scheduler.

Including more features that not just represent the jobs, but represent the execution environment of the scheduler is an important step in order to get a more accurate results.

Another direction to continue is to use the online learning technique instead of batch learning (i.e. offline learning). By online learning, the hypotheses is build and updated while the scheduler is running. Two benefits can be gained from this approach: First, no training phase at the beginning. Second and most important, the hypotheses can be updated constantly according to the newly upcoming jobs which leads to a more accurate scoring for the future jobs. Current research in the IR domain with Learning to Rank have been proposed an online learning software that ".. learns directly from interactions with users .." [Schuth *et al.*, 2013]. Applying online learning in the scheduling domain will allow the scheduler to adapt constantly and in a dynamic way to the handled workload which will lead to an additional improvement on the overall scheduling performance.

8 Acknowledgements

We would like to thanks the contributors of the Parallel Workload Archive, Victor Hazlewood (SDSC SP2), Travis Earheart and nancy Wilkins-Diehr (SDSC Blue), Lars Malinowsky (KTH SP2), Dan Dwyer and Steve Hotovy (CTC SP2), Joseph Emeras (CEA Curie), and of course Dror Feitelson. The Metacentrum workload log was graciously provided by the Czech National Grid Infrastructure. I wish to express my sincere thanks to my supervisors Denis and Eric for their time, effort and great remarks. And I would like to thanks David Glesser for his big help during my internship.

References

[Cao *et al.*, 2006] Yunbo Cao, Jun Xu, Tie-Yan Liu, Hang Li, Yalou Huang, and Hsiao-Wuen Hon. Adapting ranking svm to document retrieval. In *Proceedings of the 29th Annual International ACM SIGIR Conference*, 2006.

[Chapelle and Chang, 2011] Olivier Chapelle and Yi Chang. Yahoo! learning to rank challenge overview. In *Proceedings of the Yahoo! Learning to Rank Challenge*, 2011.

[Chiang *et al.*, 2002] Su-Hui Chiang, Andrea C. Arpaci-Dusseau, and Mary K. Vernon. The impact of more accurate requested runtimes on production job scheduling performance. In *8th International Workshop on Job Scheduling Strategies for Parallel Processing*, 2002.

[Cohen *et al.*, 1999] William W. Cohen, Robert E. Schapire, and Yoram Singer. Learning to order things. *J. Artif. Int. Res.*, 1999.

[Faria and Veloso, 2010] Fabio F. Faria and Veloso. Learning to rank for content-based image retrieval. In *International Conference on Multimedia Information Retrieval*, 2010.

[Feitelson *et al.*, 1997] Dror G. Feitelson, Larry Rudolph, Uwe Schwiegelshohn, Kenneth C. Sevcik, and Parkson Wong. Theory and practice in parallel job scheduling. In *Job Scheduling Strategies for Parallel Processing*, 1997.

[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*, 2014.

[Herbrich *et al.*, 2000] Ralf Herbrich, Thore Graepel, and Klaus Obermayer. *Large Margin Rank Boundaries for Ordinal Regression*. 2000.

[Jackson *et al.*, 2001] David Jackson, Quinn Snell, and Mark Clement. Core algorithms of the maui scheduler. In *Job Scheduling Strategies for Parallel Processing*. 2001.

[Joachims, 2002] Thorsten Joachims. Optimizing search engines using clickthrough data. In *Proceedings of the Eighth ACM SIGKDD International Conference*, 2002.

[Joachims, 2006] Thorsten Joachims. Training linear svms in linear time. In *Proceedings of the 12th ACM SIGKDD*, 2006.

[Li, 2011] Hang Li. A short introduction to learning to rank. *IEICE Transactions*, 2011.

[Lifka, 1995] David A. Lifka. The anl/ibm sp scheduling system. In *Workshop on Job Scheduling Strategies for Parallel Processing*, 1995.

[Qin *et al.*, 2010] Tao Qin, Tie-Yan Liu, Jun Xu, and Hang Li. Letor: A benchmark collection for research on learning to rank for information retrieval. *Inf. Retr.*, 2010.

[Schuth *et al.*, 2013] Anne Schuth, Katja Hofmann, Shimon Whiteson, and Maarten de Rijke. Lerot: An online learning to rank framework. In *Proceedings of the Workshop on Living Labs for Information Retrieval Evaluation*, 2013.

[Schwiegelshohn and Yahyapour, 1998] Uwe Schwiegelshohn and Ramin Yahyapour. Analysis of first-come-first-serve parallel job scheduling. In *9th SIAM Symposium on Discrete Algorithms*, 1998.

[Sculley, 2009] D Sculley. Large scale learning to rank. *NIPS Workshop on Advances in Ranking*, 2009.

[Shmueli and Feitelson, 2003] Edi Shmueli and DrorG. Feitelson. Backfilling with lookahead to optimize the performance of parallel job scheduling. In *Job Scheduling Strategies for Parallel Processing*. 2003.

[Tagami *et al.*, 2013] Yukihiro Tagami, Shingo Ono, Koji Yamamoto, Koji Tsukamoto, and Akira Tajima. Ctr prediction for contextual advertising: Learning-to-rank approach. In *7th International Workshop on Data Mining for Online Advertising*, 2013.

[Talby and Feitelson, 1999] D. Talby and D.G. Feitelson. Supporting priorities and improving utilization of the ibm sp scheduler using slack-based backfilling. In *13th International on Parallel and Distributed Processing*, 1999.

[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.*, 2007.