Using the black-box approach with machine learning methods in order to improve job scheduling in GRID environments

DANIEL VLADUŠIČ
XLAB d.o.o.
Pot za Brdom 100, SI-1000 Ljubljana
SLOVENIA
daniel.vladusic@xlab.si

Abstract: This article focuses on mapping jobs to resources with use of off-the-shelf machine learning methods. The machine learning methods are used in the black-box manner, having a wide variety of parameters for internal cross validation. In the article we focus on two sets of experiments, both constructed with a goal of congesting the system. In the first part, the machine learning methods are used as assistants for the base resource selection algorithms, while in the second part, they are used to directly select the resource for job execution. We set up two experiments with different objectives. In the first experiment, the objective was to maximize the number of jobs finished within the deadline, while in the second experiment, the objective was to minimize the job tardiness. Finally, in the first part we were changing resource selections of simple job dispatching algorithms, while in the second part, we were employing the optimization scheme of the ALEA algorithm [15]. The results show that even with a such black-box approach of the off-the-shelf machine learning methods, we can achieve improvements or at least get comparable results.

Key–Words: scheduling, distributed systems, GRID, dispatching jobs in GRIDs, machine learning

1 Introduction

The problem description

The definition of the GRID scheduling problem, usually found in literature consists of the definition of a set of resources (e.g., machines, computational nodes, storage, network capacity, etc.) and a set of jobs (e.g., computational, storage tasks) and some optimality criterion used for quantitative measurements of the quality of the scheduling. In addition to this basic description, the definition of such problem usually also consists of some specifications about the actual GRID system and of other specific constraints.

Given this definition, we can conclude that GRID systems are typical examples of dynamic environment, where the complexity of the scheduling problem is strongly dependent on the system itself, the constraints on job execution and properties of the job itself. Namely, the resources in the GRID system may be non-uniform (e.g., machines with different speeds and number of CPUs). The same also holds for jobs, which have different computational complexities (i.e., lengths), may require different machine architectures or a specific number of processors. Furthermore, jobs may have interdependencies in terms of the order of execution, deadlines, priorities and so forth. Finally, jobs can also be migrated during their execution, which introduces another possibility for scheduling. In addition to these parameters, GRID systems can also have varying network topology and bandwith requirements. We can thus conclude that the problem space of scheduling on GRID systems is large and non-heterogenous, while the general goal of scheduling is usually to meet the users and system administrators demands. This general goal can be explicitely defined as maximization of the CPU utilization, minimization of job tardiness, etc.

Motivation

Our motive in performing these experiments was to test how the off-the-shelf machine learning approaches can be used in already existing GRID frameworks. To this end, we used implementations of machine learning algorithms available in the WEKA package [14]. Using this approach we also minimize the efforts needed for their implementation in the real frameworks. Our focus was in minimization of tardiness of jobs, in other words to maximize the number of jobs finished within the deadline. These two criteria are not, however, completely interchangeable, as minimization of tardiness may also result in a higher number of jobs that are not finished within the deadline. The success of the machine learning methods in satisfying these criteria were tested in two separate experiments and using two different sets of comparison methods.

Related work

In the area of GRID systems, the artificial intelligence algorithms and more precisely, machine learning algorithms, are used in various ways, ranging from the applications in packet routing, failure diagnosis and scheduling. In the packet routing field, the suggested automatic learning of routing tables is usually undertaken with the Q-Learning approach [1, 2, 3, 4]. The most commonly used solution is the Q-Learning approach [1] where response (positive or negative) from different routing decisions is used to learn good routing strategy through time. These machine-learning-supported systems are adaptable, both in the sense of incorporating different criteria for estimating success in the sense of adapting to the underlying (physical) networks. Q-Learning is also used for the negotiation strategies for the Service Level Agreements [17]. In the area of failure diagnosis, one of the approaches is a use of decision trees [5]. The idea behind the approach is to use the application level logs in order to predict which type of requests combined with specific nodes cause the system to fail.

In the field of GRID scheduling, machine learning is used in particular for the prediction of job complexity (i.e., the job runtime length) [12, 18]. Use of genetic algorithms is present in the field of the offline scheduling [6, 7, 8, 9], while in the past, there were also attempts to use simulated annealing [10]. Current interesting approaches towards scheduling with genetic algorithms are described in [19]. An interesting approach is also a combination of planning and scheduling, described in [11]. The advantage of such hybrid approach is in its two-level scope: the long-term planning and short-term scheduling. The approach shows interesting results on example of production planning. Finally, the approach towards GRID scheduling with use of ant colony optimisation algorithms is described in [20, 21].

Following the descriptions of the problem, our motivation and related work, we proceed with the description of the experiments.

2 The experiments

Our experiments comprise of two sets, where the first set deals with simple job dispatching algorithms and the second set is using the framework ALEA [15] as the foundation and benchmark. In both sets of experiments we used the same machine learning algorithms and procedures, described in section 2.1. The simple job dispatching algorithms, which are the focus of the first set of experiments are Random, Round Robin and First Available. While the first two do not need any additional explanations, we provide the description of the First Available algorithm in the section 2.2. The objective function in these experiments was the number of jobs executed within the deadline. The second set of experiments is presented in section 2.3. While our machine learning methods have not changed significantly, the ALEA framework offers a better benchmark to our methods, thus we are testing our methods against advanced scheduling solution. The objective function in these experiments has changed to minimization of the overall job tardiness.

2.1 The machine learning extensions

In this section we describe the setup of the machine learning algorithms used in our experiments. We used the algorithms implemented in the machine learning framework WEKA [14]. We first describe the learning procedure and then list the used algorithms and their parameters.

In order to prevent the problem of overfitting, we used 4-fold cross-validation on the learning data, where the measure for prediction performance was the Area under Receiver Operating Characteristic (AROC). The reasoning behind using this measure instead of classification accuracy (CA) was that we should use a measure which should perform well in imbalanced datasets.

The positive examples (where the job was executed within the deadline) were also given weights. In order to determine how much weight these examples should have, we also included additional evaluation of a set of weights for positive examples. The algorithm is presented as Algorithm 1. We can see that we have two loops, where the inner loop performs cross-validation using different cross-validation parameters, while the outer loop is responsible for different weights of the positive examples.

The selected algorithms for use in the experiments were: Locally Weighted Learning (LWL for short), Naive Bayes (NB for short) and decision trees (J48 for short), all implemented in WEKA. The weights we used for weighting the positive examples were: 1, 2, 5 and 10. The cross-validation parameter for LWL was the number of neighboring nodes taken into account: all, 1, 10, 50 and 100, while for J48, we used the confidence threshold for pruning with the values: 0.05, 0.1, 0.2, 0.3, 0.4 and 0.5.

The number of tests we perform in the process of learning the machine learning model is quite large, as we aim for a plug-in to the existing framework which

Algorithm 1

INPUT: machine learning algorithm ($mlAlg$) and job execution history data ($eData$).

OUTPUT: machine learning induced scheduler ($mlSched$),

```
1: weights ← MLSetup.getWeights()
2: cvPars ← MLSetup.getCVPar($mlAlg$)
3: evaluations ← {}
4: for all weight : weights do
5:     for all cvPar : cvPars do
6:         wEData ← ML.reWeight($eData$, weight)
7:         aroc ← ML.eval($mlAlg$, wEData, cvPar)
8:         evaluations.add([aroc, weight, cvPar])
9:     end for
10: end for
11: [aroc, weight, cvPar] ← evaluations.getBestAroc()
12: $mlSched$ ← ML.build($mlAlg$, weight, cvPar)
13: return $mlSched$
```

has minimal number of tuning parameters and therefore needs to use a wide array of possible parameters internally[1].

2.2 Simple dispatching rules experiments

These experiments were performed withing the PeerSim simulation environment [13]. The reason for not using a typical GRID simulation framework (e.g. ALEA) is the legacy code, which is using these extensions. The PeerSim framework was extended primarily in nodes which were used as the CPU resources. The following paragraph explains the setup from the architectural point of view.

2.2.1 The architecture

The architecture of our experiment is shown in Fig. 1. The *GridBroker*, is responsible for dispatching the jobs that arrive into the system. These jobs are executed on computational resources, marked with CPU_1 to CPU_n, where each of these resources has its own queue, marked with q_1 to q_n respectively. When a new job enters the system, the GridBroker decides which computational resource should handle the execution. When this decision has been made, the job is placed into the queue of the selected node and when the job completes, the feedback is sent back to the GridBroker, with job completion status.

[1]It should alse be noted that we performed tests with automatic attribute selection. Preliminary results of using such set up have shown significantly worse performance when compared to any of the benchmarks thus we did not pursue this possibility anymore.

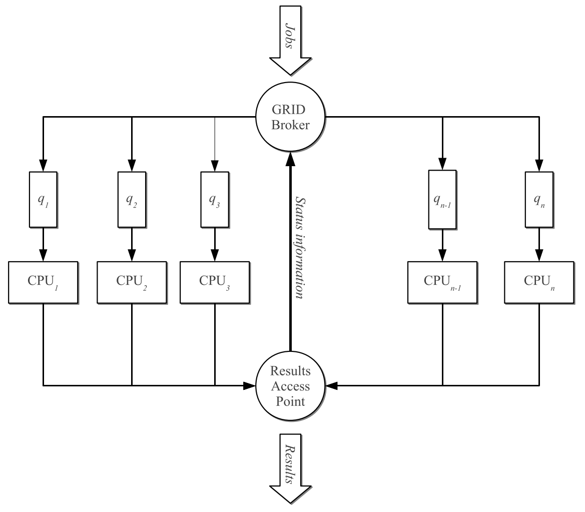

Figure 1: The architecture overview of our experimental setup.

In our case, this status is only $\mathrm{finished\_in\_time}$ or $\mathrm{not\_finished\_in\_time}$. The job results are sent to the *Results Access Point* where the users collect them.

Fig. 2 shows a more in-depth schematic view of the GridBroker module. It consists of the three submodules, used to map the job to the computational resource:

- **The base resource selector:** By base node selector algorithm we refer to one of the algorithms, usually found in this place: Random, Round-Robin, etc.
- **The machine learning resource selector:** The machine learning node selector algorithm is, in our case, a suitable off-the-shelf machine learning algorithm - e.g.: Naive Bayes, Locally Weighted Learning, etc.
- **The database:** The database serves as storage for the data about history of dispatched jobs and their status when they were finished and is thus used as the learning data for the machine learning node selector algorithms.

Algorithm 2 shows the data flow in the GridBroker module. First, the base resource selector selects the resource on which the job should be executed (i.e., default node of execution). This information is, along with the job description, passed to the machine learning resource selector, which tries to find a resource, which is possibly more suitable for execution of this job. The basic property of this algorithm is the aforementioned use of machine learning predictions only when there is a chance, based on historical data, that the machine learning selected resource of execution

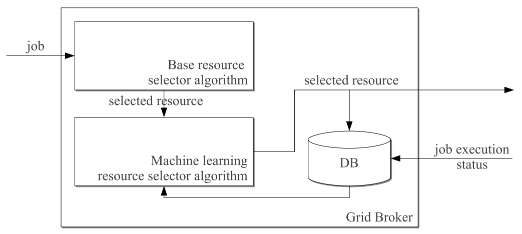
job
Base resource selector algorithm
selected resource
selected resource
Machine learning resource selector algorithm
DB
job execution status
Grid Broker

Figure 2: The architecture overview of the GridBroker module.

will perform better than the default resource of execution. Let us now briefly describe the algorithm. First, the base resource selector selects the resource of execution (i.e., the default resource of execution). The next step is to evaluate all resources with machine learning scheduling model. The data for this evaluation is job description, the resource which is being evaluated and the current system state. If the machine learning resource selector predicts the evaluated node is suitable for execution (i.e., it predicts that the job will be executed within the deadline), it is stored into the list of suitable resources. In the next step, we evaluate the probability of successful job execution of the best machine learning predicted and the default resource. Only if the chance of success with the machine learning predicted resource of execution is higher than the one of the default resource, we select the predicted resource. Otherwise, the resource of execution remains the same as selected by the default resource selector.

An important note regarding this algorithm is that it needs to go over all nodes in the system. While this is not a problem in a simulation environment, the real GRID systems may consist of thousands of resources. Still, the modern GRID systems schedule jobs only after the phase of resource discovery, which usually implements some sort of system policy or resource suitability step, hence the length of the list of available resources may not be very long. Still, we acknowledge that the algorithm must be improved for use in real systems. we must however, accept that use of off-the-shelf machine learning algorithms, without any domain tuning or use of system specific attribute, may not result in accurate predictions. Furthermore, as the learning data for these simple machine learning algorithms is obtained from the so-called base algorithm for scheduling, the algorithm which is built into the GRID framework, the predictions may be skewed towards the original resource selections. Finally, the machine learning algorithm, when it cannot predict the resource for job execution with sufficient certainty, relies to the base algorithm and these data is then used for updating the models of the machine

Algorithm 2

INPUT: job description ($jDesc$), last known system state ($sState$) and job execution history data ($eData$).
OUTPUT: resource of execution ($rExec$),

```
1: defResOfExec ← BaseSelector.getResource()
2: evaluations ← {}
3: for all resource : resources do
4:     eval ← ML.eval($jDesc, resource, sState$)
5:     if eval.getSuccess == $true$ then
6:         evaluations.add(eval)
7:     end if
8: end for
9: bestEval ← evaluations.getBest()
10: defEval ← evaluations.getEval(defResOfExec)
11: if bestEval.successProb > defEval.successProb then
12:     rExec ← bestEval.getResource()
13: else
14:     rExec ← defResOfExec;
15: end if
16: return rExec
```

learning algorithms and used for further predictions. We can thus conclude, that despite the aforementioned problems, the machine learning algorithms produce a model which takes into account past performance and more importantly, mistakes, of the scheduling algorithm.

2.2.2 Experiments and results with simple dispatching rules

We performed two sets of experiments. Each of them consisted of running five batches of jobs, where each batch consisted of 1000 jobs thus giving 5000 jobs per experiment. Descriptions of jobs consisted of the following attributes: an estimate of the time to run needed for execution of the job, the deadline for the job execution, the queue length of the node at time of job submission, the node on which the job was submitted and finally, the success of the job execution, described only with values finished_in_time or not_finished_in_time, corresponding to finished until the deadline or not.

In section 2.1 we described the machine learning algorithms that were used in our experiments, hence here we describe only the base scheduling algorithms which were used: Random, Round Robin and First Available. While the first two are commonly used, we need to explain the last one. The algorithm randomly picks a node and checks its state. The attributes it collects are whether the node is idle and the number of jobs waiting in the queue. If the node is not idle, it proceeds in round robing fashion and assigns a job to the first idle node. If there are no idle nodes, the algorithm assigns the job to the node with shortest queue. We pick the first node randomly in order to prevent skewing of the job assignments towards the start of the node list. Please note, that Random and First Available are stochastic, hence the results, when combined with our selected machine learning algorithms are not directly comparable.

The job descriptions were created randomly and scheduled to start within the first third of the simulation time. The estimate for the required execution time was also set randomly, along with the due time to finish the job (i.e., deadline), which was set according the following formula: $\text{timeToStartJob} + \text{requiredCPUTime} + \frac{1}{3}\text{random}(\text{requiredCPUTime})$. We can see that the test was indeed synthetic and that we explicitly introduced a large amount of randomness in the learning data. The reasoning for this was two-fold: first, the simulated system should reach the congestion point and second, the machine learning algorithms need to use the learning data which results from running such sets of jobs. The latter is especially important, as absence of strong patterns in data poses a problem for machine learning algorithms. The underlying system consisted of 10 heterogeneous nodes, differing in the CPU speed.

In order to provide the machine learning algorithms with a starting set of data which is used for building the initial model, we performed a test run with the learning batch of jobs. This batch of jobs consisted of 1000 jobs, which were created in a way that which prevented congestions of the system and hence produce useful and meaningful results (with strong patterns showing some of the systems' properties). These learning jobs were also created randomly, but distributed more uniformly over the simulation time than the test jobs batches. During the test experiments, the system performed an update of the machine learning scheduling model on every 100 finished jobs.

The first set of experiments consisted of running the five batches of jobs on the network of computers, with the resource CPU performance shown in Table 1. The speed marked with 1 means the normal speed (the job will execute normally), while speeds above one denote slower (the execution of the job will take longer) and speeds below one faster nodes (the execution of the job will be faster), respectively.

Results of running the first set of experiments are shown in Table 2. The table consists of three parts, where the first part shows the results, obtained with using the First Available base algorithm and following two show the use of Random and Round Robin algorithms, respectively. The rows in each part present the results for each of the machine learning algorithms, while columns show the results for each of the five batches. The meaning of the numbers is as follows: as we performed each experiment twice, once with assistance of the machine learning algorithm and once without it, we present the comparison of performance as division between the percent of jobs finished within the deadline with machine learning assistance and the percent of jobs finished within the deadline without machine learning assistance. In other words, the values higher than 1 show the advantage of machine-learning-assisted algorithms and vice-versa. The results of the first experiment show that the assistance of the Naive Bayes algorithm results in underperforming system - in majority of cases, the Naive Bayes assisted algorithm performs worse than the algorithm without the assistance. The Locally Weighted Learning and J48 algorithm perform better, but clearly, the LWL algorithm performs best. It must be noted, that not all results are comparable between the different machine-learning-assisted algorithms, as Random and First Available are not deterministic, hence we can comment only on the Round Robin algorithm. It shows that the use of machine-learning-assisted al-

p_1	p_2	p_3	p_4	p_5
1.32	1.26	0.8	1.33	0.81
p_6	p_7	p_8	p_9	p_{10}
0.91	1.06	1.35	0.97	1.52

Table 1: The relative resource CPU performance for the first set of experiments.

First Available					
J48	1.12	0.95	0.99	1.16	0.65
LWL	1.11	1.17	0.86	1.08	0.96
NB	0.95	1.02	0.94	1.28	1.05
Random					
J48	1.08	1.08	1.03	1.19	0.89
LWL	1.2	0.98	1.18	0.74	1.14
NB	0.97	0.89	1.1	0.82	0.75
Round Robin					
J48	0.98	0.99	1.39	1.03	1.04
LWL	1.16	1.05	0.92	1.01	1.04
NB	0.92	0.89	0.93	0.97	0.84

Table 2: The comparison between machine-learning-assisted and no machine learning assistance algorithms for the first set of experiments.

Round Robin-J48				
0.09	0.07	**0.09**	0.08	**0.1**
0.14	0.1	0.09	**0.15**	0.09
Round Robin-LWL				
0.05	0.06	**0.18**	0.07	**0.15**
0.17	0.08	0.03	**0.17**	0.04
Round Robin-NB				
0.11	0.12	**0.06**	0.11	**0.1**
0.11	0.12	0.12	**0.04**	0.11

Table 3: The percent of job assignments over nodes in the first experiment. The ordering of the values corresponds to the node CPU speeds in Table 1.

gorithms can improve the percent of successfully finished jobs (LWL) and also that machine learning algorithms, especially J48, are brittle (i.e., small changes in the learning data can produce big changes in the model and predictions) and thus not to be used without caution.

In order to assess the differences between machine learning algorithms, we need to focus on the Round Robin algorithm, as the assistance value can be directly compared. Table 3 shows the percentage of job assignments to resources. Its structure is the same as of the Table 1. The bold values correspond to the fastest resources. This comparison shows that only the LWL algorithm adapts to the performance of the nodes and uses faster nodes consistently more than the slower ones. The two remaining algorithms do not adapt to the performance of the nodes to such a degree.

The second set of experiments consisted of two experiments: we used a new set of the five job batches and executed them as in the first experiment. We then run another experiment, using the same set of jobs, but with an important difference: after the second batch of jobs was executed, we completely changed the properties of the nodes thus testing the performance of the machine learning algorithms under completely changed conditions. Furthermore, we did not allow for the learning set of jobs to be used nor discarding of the old results from the first two batches, thus testing the adaptability of the machine-learning-assisted algorithms. The CPU performance of the nodes before and after the change are shown in Table 4, where the node speeds marked with Normal were used in the first experiment and in the first two batches of the second experiment and the nodes marked with Changed were used after the second batch of the second experiment.

The results of the experiment described above is shown in Table 5 and divided into three parts, according to the base scheduling algorithm. Each of these parts is then subdivided according to the machine learning algorithm used and whether the node properties have been changed. Review of the second set of experiments shows that some trends are similar to the ones observed in the first set of experiments. The J48 algorithm performs in some cases very well and in others rather poorly thus showing its brittleness. The LWL and the Naive Bayes algorithm perform well in almost all cases. The performance of the latter is in stark contrast with the first set of experiments, where it performed rather poorly. Given this limited set of tests, we can thus conclude that only the LWL algorithm can be considered as stable enough to be used in real systems. Let us turn our attention to the performance before and after the change of the nodes: in majority of cases the machine-learning-assisted algorithms underperform immediately after the change (the third set of jobs) when compared to the base algorithm. In the fourth and fifth set of jobs their performance usually improves and overcomes the base algorithm. This leads to the conclusion that the algorithms can adapt to the changed circumstances, despite the fact that they also learn from the data originating from the entirely different set of nodes.

In the following section we describe our second set of experiments with dispatching of jobs using machine learning assitance.

Normal	p_1	p_2	p_3	p_4	p_5
	1.45	1.54	1.22	0.96	1.52
	p_6	p_7	p_8	p_9	p_{10}
	1.53	1.34	1.47	1.45	1.17
Changed	p_1	p_2	p_3	p_4	p_5
	1.46	1.58	1.23	1.1	0.97
	p_6	p_7	p_8	p_9	p_{10}
	0.81	1.1	1.14	1.4	0.91

Table 4: The resource relative CPU speeds for the second set of experiments.

2.3 ALEA Experiments

For these experiments we used a GRID simulation framework ALEA [15]. This simulation framework enabled us to test our machine learning extensions, described in section 2.1, against the advanced scheduling algorithm. ALEA provides two benchmark algorithms: Earliest Deadline First (EDD for short), Earliest Release Date First (ERD for short), both combined with the Tabu search algorithm. The ERD and EDD algorithms are used only when new job enters the system. Further optimizations of the schedule are made with the Tabu search algorithm. The EDD algo-

First Available						
J48	1.73	0.88	2.11	1.76	1.58	no change
			0.66	1.24	0.89	change
LWL	1.47	0.94	1.17	0.90	1.13	no change
			0.81	1.26	1.01	change
NB	1.42	0.70	1.48	2.81	2.61	no change
			0.97	0.95	0.94	change
Random						
J48	1.09	1.36	0.66	1.35	1.85	no change
			0.73	1.48	1.07	change
LWL	1.74	0.97	1.14	0.85	1.17	no change
			0.81	1.39	1.01	change
NB	1.03	0.99	0.97	1.33	0.67	no change
			1.25	1.10	1.20	change
Round Robin						
J48	0.84	1.24	1.08	0.5	1.16	no change
			0.98	1.11	1.08	change
LWL	1.16	1.45	0.75	1.15	1.01	no change
			0.80	1.13	1.08	change
NB	1.03	0.85	1.13	0.95	1.03	no change
			1.11	1.03	1.08	change

Table 5: Relative improvements of jobs being finished within the deadline for the second set of experiments.

rithm iterates over all of the computational resources and inserts the job into the schedule according to the job's deadline, i.e. the insertion is between the jobs where the predecessor is due to be finished before the current and the ancestor is due to be finished after the current job. The resource on which the increase of tardiness is the smallest, is selected to execute the job. The same principle is used in the ERD[2] insertion strategy, the only difference being the insertion according to the time of arrival of the job into the system (i.e., the release date).

The policy for new jobs is to add newly arrived job into the existing schedule (either via EDD or ERD strategy), therefore it is reusing the existing scheduling soulution (e.g., the incremental approach). Given that the schedules, derived from focusing on the newly arrived job, for each machine now exist, the Tabu search optimization is performed. This optimization is performed periodically (after a pre-determined number of new jobs has arrived into the system) and focuses on repairing the schedules of all resources in the system. In a sense, it takes the global view on the system and concentrates on all jobs that are currently in queues for execution. The jobs that are currently running are not taken into account, therefore no migration is being undertaken, despite of possibly evident tardiness of the running job.

In the first step, the Tabu search optimization finds the resource with maximum tardiness. Then, it finds the job on this resource with maximum tardiness. This job is the candidate for moving to other resources. Alternatively, if no jobs can be moved (i.e., they are already in the tabu list), the resource is also added to the tabu list. The job is moved in the following way: first, a resource with minimum tardiness is found and the job is inserted into its list according to its due date. Next, this move is evaluated against overall tardiness and if the move reduces it, the job is left in the schedule of the selected resources and added to the job tabu list (i.e., it will not be moved again). Otherwise, the schedules are reverted back to their original states and the resource is added to the tabu list of candidate resources. Note that in the Tabu search phase the jobs are always inserted according to their deadline and that the gaps (i.e., the time the CPU is idle) in schedules are not considered.

The above paragraphs present an outline of the algorithm in the ALEA simulator, which is necessary for understanding of our additions to the ALEA simulator. A more in-depth understanding of the simulator can be obtained through [15] and [16].

For our experiments we let simulator use its default settings:

- When job enters the system, it is added to the schedule of the machine with the objective to minimally increase the overall tardiness. The tests were performed with both insertion strategies - EDD and ERD, respectively.
- The optimizations of the schedules was performed periodically with Tabu search, therefore, after every 10 jobs arriving into the system, the schedules of the machines were optimized. Tabu search was set to perform 10 steps of optimization.

2.3.1 Machine Learning Additions Setup

The machine learning algorithms were added on two levels: firstly, when a new job enters the system and secondly after 10 jobs have entered the system. We hence take the same approach as with the use of Tabu search in ALEA. However, an important difference between the approaches lies in the fact that we did not check whether the moves were good, but relied only on the prediction given by the machine learning algorithm.

When a new job arrives into the system, we use the currently built model to decide to which resource it should be dispatched. The algorithm is shown in

[2]Earliest Release Date is equivalent to the First Come First Serve algorithm.

Algorithm 3

INPUT: job description ($jDesc$) and last known system state ($sState$).

OUTPUT: node of execution ($nExec$),

```
1: evaluations ← {}
2: for all node : nodes do
3:    eval ← ML.eval($jDesc, node, sState$)
4:    if eval.getSuccess == $true$ then
5:       evaluations.add(eval)
6:    end if
7: end for
8: bestEval ← evaluations.getBest()
9: nExec ← bestEval.getNode()
10: return nExec
```

Fig. 3. The main difference between this algorithm and the algorithm used in previous experiments (described in section 2.2) is in the fact that this algorithm does not rely on the base prediction anymore thus the dispatching of jobs to nodes is completely up to the machine learning algorithm. The algorithm also shows that we employed the same approach as in the previous experiments, i.e. we select the resource according to its probability of successfully execution the job.

Let us now turn towards the periodic optimization of the schedules with machine learning methods. First, we search for the resource with highest tardiness and find the job with highest expected tardiness. We then take this job out of the schedule and employ the same approach as shown in the Algorithm 3. The job is put into schedule of the resource with highest probability of the timely execution. This procedure is repeated 10 times.

The machine learning algorithms were set up as in our previous experiments, with the same parameters for optimization of the built models and the same attributes that were used for learning. We, however, did not use the LWL algorithm in these experiments, as the performance of its implementation in WEKA rapidly degraded with larger amounts of data.

The description of our machine learning extensions to the original algorithm shows that in these experiments we used only the high-level scheme of the original scheduling algorithm, as shown in Fig. 3. The selection of resources was not provided with any help from the original algorithm (i.e., there is no Base resource selector algorithm), but rather a scheme with online reparations of the schedule was employed. The basic difference between the architecture of the first set of experiments (shown in Fig. 1) and these experiments is in the position of the queues. While in the first set of experiments we had no control over queues (i.e., schedules) after we have dispatched the job, here the queues are manipulated during the periodic optimization phase.

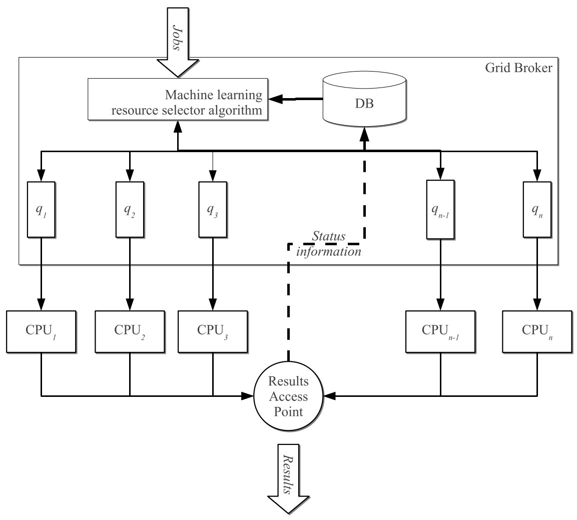

Figure 3: The architecture overview of our experimental setup.

Finally, let us sum up the differences between the ALEA and our scheme. ALEA is using the EDD and ERD insertion strategy, combined with Tabu Search. All job migrations (i.e., between schedules of different resources) are performed with insertion according to the due date of the job. The candidate resources for optimizations are found through search over all the resources and calculating their tardiness. The algorithm is also calculating the overall tardiness for the evaluation of job moves. On the other hand, our algorithm is calculating tardiness only when deciding on which job should be moved on the candidate resource. The candidate resource is the one with longest queue of jobs to be executed. These differences show that we are using significantly lower amount of information for initial dispatching and optimization of the schedules. Furthermore, our insertion strategy is always according to the due date criterion.

2.3.2 Experiments

We performed two sets of experiments. In both sets, we used 8 resources, each with 2 CPUs. The jobs that were generated were synthetic and always required only 1 CPU. The start time of the jobs and their deadlines were set by the job generation algorithm, provided in ALEA. As job tardiness is dependent on the ratio between the number of resources avaliable and jobs to execute, this setup guaranteed job tardiness.

Descriptions of jobs were the same as in experiments, described in section 2.2 and hence consisted of the following attributes: an estimate of the time to run needed for execution of the job, the deadline for the job execution, the queue length of the node at time of job submission, the node on which the job was submitted and finally, the success of the job execution, described only with values finished_in_time or not_finished_in_time, corresponding to finished until the deadline or not.

In the first set of experiments we first used the ALEA default settings to generate and execute 10000 synthetic jobs, which were used as the learning data for the machine learning algorithms in order to produce the initial predictors. We then performed 10 experiments with each of the selected machine learning methods and compared them to the results obtained with the original algorithm. During these experiments, the machine learning algorithms always used the learning data in order to initialize the predictor. In other words, they did not use any information from the previous experiments. The results are shown in Table 6, where the average tardiness of jobs is shown for each of the experiments and method used. The last line shows the average of the average tardiness over all of the experiments.

The comparison between methods shows that only use of the J48 algorithm in our scheme can be roughly compared to the ALEA algorithms. Furthermore, the J48 algorithm can be compared only to the EDD insertion strategy, while ERD strategy performs significantly better.

In the next experiment, we allowed the machine learning algorithms to incrementally learn over experiments, but we did not allow for any initial learning data before the first experiment. The machine learning algorithms performed an update of their models after 10 new jobs entered the system. The aim of this experiment was to see whether performance of the ML algorithms can be improved with use of larger datasets and through use of incremental learning.

The results are shown in Fig. 4. The mutual comparison is performed as follows: The benchmarks for our approach were ALEA Tabu search optimization strategies with EDD and ERD insertion strategy, both repeated 10 times with 1000 jobs. The average tardiness of these 10 experiments was then again averaged over the number of experiments and is hence shown as a flat line in the graph. The machine learning experiments were performed 50 times with 1000 jobs, giving 50 average tardiness results. Each dot in the graph represents the average of 10 experiments, i.e. the first dot is the average of experiments 1 to 10, the second dot is the average of the experiments 2 to 11, and so on. Thus, the graph consists of 40 experiments, where the dots are calculated in way which guarantees stability of the obtained results.

The comparison between the J48 and Naive Bayes machine learning algorithms shows that J48 performs significantly better. Furthermore, from the fourteenth experiment on, the J48 algorithm performs as well as the EDD insertion strategy with Tabu search optimization, while the ERD insertion strategy performs overall best. Both of the machine learning approaches show improvements with increasing of number of learning examples. While the J48 algorithm shows stable results with the increase of available learning data, the results of the Naive Bayes algorithm show much more instability. Even though the results are averaged over 10 experiments, we can label the performance of the Naive Bayes algorithm as erratic.

Comparison with the overall behaviour of the machine learning algorithms from this and previous experiments, described in section 2.2, where we found the LWL algorithm to be stable, while Naive Bayes and J48 algorithms were brittle, shows that the large set of experiments puts the J48 algorithm into the category of stable algorithms.

3 Discussion

We performed two sets of experiments, both using the same machine learning algorithms and infrastructure for selection of the resource for job execution. The first experiment concentrated on experiments with smaller amounts of data with three machine learning algoriths: J48, Naive Bayes and LWL. The second experiment concentrated on experiments with larger amounts of data and, due to degraded performance of the implementation of the LWL algorithm in WEKA, used only two machine learning algorithms: J48 and Naive Bayes. The objectives of the two algorithms were different: in the first set of experiments, we were maximizing the number of jobs finished within the deadline, while in the second set we minimized tardiness. The role of the machine learning algorithms differed between the two experiments: in the first set, their predictions were used only if probability of successfully finishing the job on the default selected resource was lower than the machine learning predicted one. In the second set of experiments, the machine learning algorithms were not in the role of assistant, but had complete control over selection of the resource. In both experiments we evaluated performance of the off-the-shelf machine learning approaches, aiming towards straightforward and cost-effective integration into existing GRID environments. Furthermore, almost all of the attributes used for model induction were simple and can be obtained or calculated directly. The only exception is the required job CPU time, which was always given beforehand. Still, the evaluation of job complexity may,

#	J48	Naive Bayes	ALEA-EDD	ALEA-ERD
1	6852986	7802715	4562290	3686164
2	6975198	8311823	4898042	3946878
3	5991922	9450453	4629544	3834669
4	4273658	7858624	4616085	3802694
5	4956201	8225339	5242742	4485815
6	4426126	8187952	4805775	4157972
7	4636639	10273174	4932950	4134214
8	4194367	7955454	4515571	3879128
9	4243210	11394544	4421033	3667836
10	4126020	8864409	4288354	4016526
average	*5067633*	*8832449*	*4691239*	*3961190*

Table 6: The average tardiness over 1000 jobs - the average tardiness over 10 experiments is shown in italic.

to some extent, be also predicted [12]. Finally, the number of parameters used for their self-optimization through internal cross-validation was quite large and hence enabled the use of the black-box approach, which suits the intended use the most (i.e., the plug-in architecture).

The performance of the machine learning algorithms in both experiments shows issues with stability of predictions. Furthermore, the performance of the same algorithms may be radically different in different setups of the experiments. For example, in the first set of experiments, the performance of the Naive Bayes algorithm can be either very good or rather poor. This behaviour is to some extent evident also in the second set of experiments, which shows, on a significantly larger learning dataset, instabilites in terms of performance. The J48 algorithm shows stable performance only in the second set of experiments, where larger amounts of learning data is available. We can thus conclude that, given sufficient amount of data, J48 algorithm can be used in the described setup. Finally, the LWL algorithm, while its performance was stable in the first set of experiments, could not be tested in the second set of experiments due to performance issues. Still, it is safe to assume that due to the nature of the algorithm itself, its performance should provide the required stability.

The comparison with the base or benchmark algorithms shows that the performance of the black-box approach with the off-the-shelf machine learning algorithms, can perform either well or at least comparably to some of the benchmarks. The applicability of the employed approach is dependent on the actual GRID system, available attributes and available scheduling or job dispatching algorithms.

Finally, we acknowledge the ideal circumstances of our simulations. In the real-world we cannot assume the machines and the network are ideal in terms of possibility of failure. We also directly used the time needed for computation of the job and the queue lengths of the nodes. We decided that for these experiments we omit these complications and try to assess the machine-learning-assisted job dispatching and scheduling algorithms on their own. Also, we acknowledge that we have not used the whole array of possible attributes, which are available in the real systems (e.g., user and user group to which the job belongs).

4 Conclusions

This paper describes use of off-the-shelf machine learning algorithms for mapping jobs to resources with black-box approach. The performance of the algorithms was tested in two different setups with two different objectives - maximization of jobs finished within the deadline and minimization of tardiness.

In the first set of experiments, the machine learning methods were used as assistants to the base resource selection algorithm. Furthermore, the machine learning algorithms were used cautiously and were allowed to change the decision of the base scheduling algorithm only if their prediction was, based on historical data, better. The results show that use of machine learning algorithms in the sense of correcting the base algorithm's decisions improves the number of jobs finished within the deadline to some degree. On the other hand, the results also show that these algorithms are brittle thus there is no guarantee that their use will always improve the base algorithm. Furthermore, our results also show that even the algorithm which can be considered as overall best, in some cases underperforms when compared to the base algorithm. Still, the results of the the experiment where we changed node properties between batches of jobs are encouraging as they show that our approach can adapt to completely

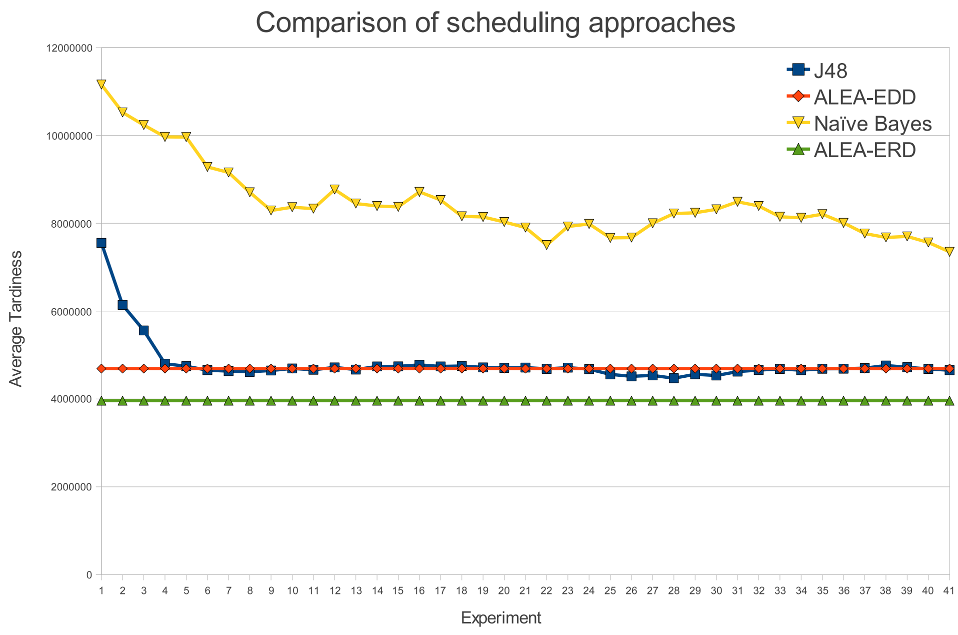

Figure 4: The comparison of performance between ALEA (EDD and ERD version) versus J48 and Naive Bayes algorithms, both learning incrementally.

different circumstances quite quickly.

In the second set of experiments the machine learning methods were used to select the resources directly. The benchmark algorithms used were ALEA with EDD and ERD job insertion strategy, combined with Tabu search. Use of the machine learning methods in this set of experiments only followed the optimization scheme of the original algorithm, while trying to use even less information about the system. The results show that the machine learning algorithms, in this scheme, are as good as the ALEA with EDD job insertion strategy and Tabu search, but perform worse than ALEA with ERD job insertion strategy and Tabu search optimization algorithm.

The machine learning algorithms were used as black boxes, using internal cross validation for parameter tuning and without any inspections of the induced models. The properties of the system were learned using a simple batch of jobs, while the testing was performed with jobs and system, which guaranteed congestions of the system.

Finally, given the number and type of experiments, we also acknowledge that more experiments need to be performed. Our future plans consist of use of real system data, an increase of the number and diversity of experiments (i.e., different number of nodes, system and network failing, different loads on the system, use of user data, etc.) and inclusion of additional base and machine learning algorithms (e.g., ALEA-EG-EDF with Tabu search, Neural networks, etc.).

References:

[1] J. Boyan, M. L. Littman, Packet routing in dynamically changing networks: a reinforcement learning approach, *Advances in Neural Information Processing Systems*, 7, 1994, pp. 671–678.

[2] G. Di Caro, M.Dorigo, AntNet: Distributed stigmergetic control for communications networks, *Journal of Artificial Intelligence Research*, 9, 1998, pp. 317–365.

[3] D. D. Clark, C. Partridge, J. C. Ramming, J. T. Wroclawski, A knowledge plane for the internet, *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, Karlsruhe, Germany, 2003, pp. 3–10.

[4] T. Itao, T. Suda, T. Aoyama, Jack-in-the-net: Adaptive networking architecture for service emergence, *IEICE Transactions on Communications*, E84-B(10), 2001.

[5] M. Chen, A. X. Zheng, J. Lloyd, M. I.Jordan, E. Brewer, Failure Diagnosis Using Decision Trees, *First International Conference on Autonomic Computing (ICAC'04)*, New York NY, USA, 2004, pp. 36—43.

[6] E. K. Burke, J. .P. Newall, A Multi-Stage Evolutionary Algorithm for the Timetable Problem,

IEEE Transactions on Evolutionary Computation, 3.1, 1999, pp. 1085–1092.

[7] Genetic Algorithms And Highly Constrained Problems: The Time-Table Case, *Proceedings of the First International Workshop on Parallel Problem Solving from Nature*, Dortmund, Germany, Lecture Notes in Computer Science 496, Springer-Verlag, 1990, pp. 55–59.

[8] M. W. Carter, G. Laporte, and S. Y. Lee. Examination timetabling: Algorithmic strategies and applications. Working Paper 94-03, University of Toronto Dept of Industrial Engineering, January 1995.

[9] B. Paechter, A. Cumming, and H. Luchian. The use of local search suggestion lists for improving the solution of timetable problems with evolutionary algorithms. *Lecture Notes in Computer Science 993* (AISB Workshop on Evolutionary Computing), Springer-Verlag, Berlin, 1995, pp. 86–93.

[10] J. M. Thompson and K. A. Dowsland. General cooling schedules for a simulated annealing based timetabling system. In: *The Practice and Theory of Automated Timetabling: Selected Papers from the 1st International Conference*, Lecture Notes in Computer Science 1153, Springer-Verlag, Berlin, 1996, pp. 345-364.

[11] D. E. Smith, J. Frank, A. K. Jonsson, Bridging the Gap Between Planning and Scheduling, Knowledge Engineering Review, 2000.

[12] H. Li, L. Wolters, An Investigation of Grid Performance Predictions Through Statistical Learning, *First Workshop on Tackling Computer Systems Problems with Machine Learning Techniques (SysML)*, Saint-Malo, France, 2006.

[13] PeerSim. PeerSim. http://peersim.sourceforge.net/.

[14] I. H. Witten and E. Frank, *Data Mining: Practical machine learning tools and techniques*, 2nd Edition, Morgan Kaufmann, San Francisco, 2005.

[15] D. Klusek, L. Matyska, H. Hana. Alea Grid Scheduling Simulation Environment, *Parallel Processing and Applied Mathematic* Springer Verlag, Berlin, 2007, pp 1028–1038.

[16] ALEA Simulation Framework, version 1. http://www.fi.muni.cz/ xklusac/alea/download/source1.zip.

[17] J. Li, R. Yahyapour, Learning-Based Negotiation Strategies for Grid Scheduling, *S*ixth IEEE International Symposium on Cluster Computing and the Grid (CCGRID'06), 2006, pp. 576–583.

[18] W. Smith, Prediction Services for Grid Computing. In: *T*he 22nd IEEE International Parallel and Distributed Processing Symposium, April 2007.

[19] C. Franke, *Design and Evaluation of Multi-Objective Online Scheduling Strategies for Parallel Machines using Computational Intelligence*, PhD Thesis, University of Dortmund, 44221 Dortmund, Germany, 2006.

[20] Z. Xu, X. Hou, J. Sun, Ant algorithm-based task scheduling in grid computing. In: *E*lectrical and Computer Engineering 2003, 2003, pp. 1107–1110.

[21] G. Ritchie, J. Levine, A hybrid ant algorithm for scheduling independent jobs in heterogeneous computing environments, In:*Third Workshop of the UK Planning and Scheduling Special Interest Group (PLANSIG 2004)*, December 2004.