

# New Scheduling Approach using Reinforcement Learning for Heterogeneous Distributed Systems

Alexandru Iulian Orhean<sup>a</sup>, Florin Pop<sup>\*a</sup>, Ioan Raicu<sup>b</sup>

<sup>a</sup>Computer Science Department, Faculty of Automatic Control and Computers, University Politehnica of Bucharest, Romania

<sup>b</sup>Department of Computer Science (CS), Illinois Institute of Technology (IIT)

---

## Abstract

Computer clusters, cloud computing and the exploitation of parallel architectures and algorithms have become the norm when dealing with scientific applications that work with large quantities of data and perform complex and time-consuming calculations. With the rise of social media applications and smart devices, the amount of digital data and the velocity at which it is produced have increased exponentially, determining the development of distributed system frameworks and platforms that increase productivity, consistency, fault-tolerance and security of parallel applications. The performance of such systems is mainly influenced by the architectural disposition and composition of the physical machines, the resource allocation and the scheduling of jobs and tasks. This paper proposes a reinforcement learning algorithm for the scheduling problem in distributed systems. The machine learning technique takes into consideration the heterogeneity of the nodes and their disposition within the grid, and the arrangement of tasks in a directed acyclic graph of dependencies, ultimately determining a scheduling policy for a better execution time. This paper also proposes a platform, in which the algorithm is implemented, that offers scheduling as a service to distributed systems.

*Keywords:* Scheduling, Distributed Systems, Machine Learning, SARSA.

---

## 1. Introduction

2 The constant evolution of technology has grown in tandem with the quantity  
3 of data generated from scientific experiments and research. But in the last few  
4 years, due to the increase in popularity of social media applications and the  
5 rise of smart devices, such as smartphones, smartwatches and health monitor  
6 gadgets, smart city solutions, like intelligent semaphores, and the Internet of  
7 Things trend, the amount of data generated has grown exponentially. Proper

---

\*Corresponding author, Tel.: +40-723-243-958; Fax: +40-318-145-309; *Email address:* florin.pop@cs.pub.ro.

8 analysis of that information, combined with the insight offered by data gathered  
9 by organizations and institutions, could prove useful in taking right decisions or  
10 in the prevention of catastrophes. In order to store, access, analyze and process  
11 large volumes of information, that is produced at a fast rate, new paradigms  
12 needed to be explored [1], paradigms that make use of parallel and distributed  
13 architectures and suitable algorithms. Computer clusters, computer grids and  
14 modern supercomputers have become the most popular systems when dealing  
15 with the challenges of big data or with intensive parallel applications.

16 Supercomputers are formed of dedicated machines, that are connected with  
17 each other throughout a well organized and fast network, and have high per-  
18 formances, but usually have high costs and are specialized in solving certain  
19 problems. Computer clusters or grid systems made of commodity hardware are  
20 the favorite solution both in the industry and in the academia, because of its  
21 low costs and highly configurable characteristic, given by the frameworks and  
22 platforms that run on the systems. One such framework is the Apache Hadoop  
23 Ecosystem [2] that implemented the successful data processing model MapRe-  
24 duce [3], published by Google. The framework has grown over the years inte-  
25 grating components that assure data replication and consistency, fault-tolerance,  
26 security, safe execution of scalable parallel applications. One of the most im-  
27 portant enhancements to the Hadoop environment was the separation of the  
28 resource negotiator, known as YARN [4], the advantage being the ease of cus-  
29 tomization. This intended versatility of YARN confirms the importance of the  
30 scheduling process in the efficiency of the system and of the applications.

31 Scheduling in distributed systems represents a broad subject, given the com-  
32 plexity of modern computer clusters and the nature of applications that run in  
33 them. Scheduling may refer to job or task scheduling or resource allocation. The  
34 scheduling can be dynamic, deciding for current running jobs and tasks, or it  
35 could schedule in advance the assignment of tasks from a given workflow. Mod-  
36 ern systems allocate virtual machines or containers, that have reduced resource  
37 capabilities, and form clusters of heterogeneous nodes in which applications can  
38 run. The general scheduling problem is a NP-hard [5] problem and it is difficult  
39 to find a general heuristic method to solve it. In this paper it is discussed the  
40 problem of assigning tasks, that can be represented as a directed acyclic graph of  
41 dependencies, on a given set of machines in order to obtain better performances.

42 Machine learning is a vast domain of artificial intelligence, that has grown  
43 in popularity because of its simple recipes or algorithms that give programs  
44 the capability to learn patterns, behavior, models and functions, and use that  
45 information to make better decisions or actions in the future. Machine learning  
46 can be classified as: supervised learning, where a training data set is given and  
47 the agent learns how to predict output values of certain input targets; unsuper-  
48 vised learning, where an agent learns a certain structural organization of the  
49 input data or the relationship of the elements of the data set; and reinforcement  
50 learning, where an agent is given certain rewards corresponding to the utility  
51 of an action or decision relative to the world model, with which the agent in-  
52 teracts. Neural networks and deep learning are the most prominent concepts  
53 that roam in artificial intelligence, as a result of their capacity to find more effi-

54 cient solutions than heuristic approaches. These are used in many classification  
55 problems [6, 7, 8, 9]. Regarding the problem of task scheduling in distributed  
56 systems, the machine learning box will use reinforcement learning algorithms to  
57 schedule the tasks in the given cluster of computers.

58 The intent of this paper is to explore the scheduling problem in distributed  
59 systems, through the perspective of reinforcement learning algorithms. In or-  
60 der to be able to integrate machine learning methods in systems that use task  
61 schedulers, this paper proposes the implementation of a Machine Learning Box  
62 (MBox). The MBox application uses the BURLAP library [10, 11, 12] for the  
63 implementation of the reinforcement learning agents, the domains and the world  
64 models of the scheduling problem. BURLAP offers a simple and configurable  
65 interface for the implementation of various planning and learning algorithms,  
66 and it has a collection of machine learning algorithms ready for use. It also  
67 offers a suite of analysis tools for the visualization of domains and agent perfor-  
68 mance. The Machine Learning Box offers scheduling services, through the Java  
69 RMI API, to distant or local clients. The clients use remote allocated sched-  
70 ulers, in order to register different world models, that characterize the number  
71 of machines for which the scheduling will take place, and send schedule request,  
72 receiving a response with the scheduling solution. As an example, the Work-  
73 flowSim [13] toolkit was used for the testing and performance evaluation of the  
74 scheduling solution.

75 The rest of the content is organized as follows. Related work is presented in  
76 Section 2, along with the the most known and most used reinforcement learning  
77 algorithms. In Section 3 the scheduling problem in distributed systems is defined  
78 and the proposed reinforcement learning model is discussed. In Section 4 the  
79 Machine Learning box architecture and design is detailed with the result being  
80 analyzed in Section 5. Section 6 draws conclusions and discusses future work.

## 81 2. Background and related work

### 82 2.1. Related work

83 Given the problem of scheduling possible parallel tasks, that are dependent  
84 on one another, in distributed and heterogeneous systems, there are many re-  
85 searches and experiments published, some of them using heuristic approaches  
86 while others using evolutionary algorithms. Genetic algorithms [14] represent a  
87 class of suitable solutions, due to the natural affinity between the task schedul-  
88 ing solution and the representation of individual from the populations that a  
89 genetic computer program works with. Hybrid solutions, that combine different  
90 strategies, such as heuristic optimizations, definition of statistical models and  
91 artificial intelligence techniques [15], show great promise in solving the schedul-  
92 ing problem.

93 Experiments show that machine learning algorithms can achieve great perfor-  
94 mances in scheduling tasks. Temporal difference, a classic reinforcement learn-  
95 ing algorithm, has been shown to be able to solve the scheduling problem [16],  
96 but with the help of a neural network that learned the evaluation functions over

97 states. Other investigations have shown that queuing models combined with  
98 reinforcement learning techniques permit optimization of the tasks scheduling  
99 process at a finer granularity [17]. Mechanism that learn best scheduling strate-  
100 gies [18], from a list of methods that were created to improve certain metrics in  
101 cloud computing, have also been proposed, letting an agent decide from past ex-  
102 periences which strategy is more appropriate giving a set of conditions. Exotic  
103 research, circumvent the process of learning scheduling policies from past expe-  
104 riences by interacting with the environment or from other heuristic strategies,  
105 but from expert human or synthetic demonstrations [19].

## 106 2.2. Reinforcement Learning Algorithms

107 Reinforcement learning [20] [21] represents a class of machine learning algo-  
108 rithms in which the agent learns how to behave in a world through the positive  
109 and negative rewards that it receives. The rewards do not appear after each  
110 action the agent takes in the world, but only when it achieved a certain point  
111 of interest. Through multiple iterations the agent must realize which of the ac-  
112 tions led to the specific compensation. Initially having no idea on what are the  
113 consequences of every action, an agent must explore the world in order to better  
114 understand the purpose. Reinforcement learning algorithms always encounter  
115 the explore versus exploit dilemma, in which an agent must decide if it should  
116 follow a course of actions or try to different paths. On one hand, if an agent  
117 commits to much on exploration it will not be able to learn anything valuable,  
118 on the other hand through exploitation it might not be able to discover the  
119 optimal sequence of steps that have the maximum utility.

### 120 2.2.1. Q-Learning

121 The Q-learning algorithm is a reinforcement learning technique, in which the  
122 agent tries to learn an optimal state-action policy based on a sequence of state-  
123 action-rewards, that represent the interactions the agent had with the world.  
124 This method does not require the model of the world to be known, computing the  
125 utilities of state-actions in order to maximize the reward. The optimal policy is  
126 realized through the selection of the best state-actions according to the utility  
127 values learned. Formally the Q-learning technique consists of an agent, a set  
128 of states  $S$  of the world, a set of actions  $A$ , a definition of how actions change  
129 the world  $T : S \times A \rightarrow S$ , also known as transition dynamics, a set of rewards  
130  $R : S \times A \rightarrow \mathbb{R}$  for each actions, a table of utilities  $Q : S \times A \rightarrow \mathbb{R}$  and a  
131 policy  $\pi : S \rightarrow A$ . The agent's goal is to maximize the reward and in order to  
132 do so, it must learn which is the best action taken from each state, the optimal  
133 action having the highest long-term reward. For such a solution to be effective  
134 an agent should run multiple training episodes for the purpose of exploring and  
135 finding the optimal policy. Algorithm 1 describes the Q-learning algorithm in a  
136 deterministic and finite world.

137 One of the most important factors of a Q-learning algorithm is the selection  
138 step of the action from a given state. The strategy used determines if the agent  
139 tend to explore new paths or to exploit currently known solutions. If an agent

---

**Algorithm 1** Q-learning

---

```
1: function Q-LEARNING( $s_{initial}, s_{terminal}, \alpha, d$ )
2:   initialize  $Q[S, A]$ 
3:    $s \leftarrow s_{initial}$ 
4:   while  $s \neq s_{terminal}$  do
5:     select  $a$ 
6:      $r = R(s, a)$ 
7:      $s' = T(s, a)$ 
8:      $Q[s, a] \leftarrow Q[s, a] + \alpha(r + d \cdot \max_{a'} Q[s', a'] - Q[s, a])$ 
9:      $s \leftarrow s'$ 
```

---

140 chooses first the actions that were unexplored, then it will not be able to use  
141 the utilities it had learned in the previous episodes. If the agent chooses first  
142 the best action, if the world is deterministic it might get stuck on a known and  
143 well traveled path that might not represent the optimal policy.

144 There are also two fine-tuning parameters in the utility value update function  
145 that characterize the performance of a reinforcement learning algorithm given  
146 a certain world or problem:

$$Q[s, a] \leftarrow Q[s, a] + \alpha \left( r + d \cdot \max_{a'} \{Q[s', a']\} - Q[s, a] \right) \quad (1)$$

147 where:

- 148 •  $\alpha$  - represent the learning rate. It influences at what extent does the new  
149 acquired information influence the old information. The learning rate takes  
150 values between 0 and 1, the inferior extremity meaning that the agent will  
151 not learn anything, while the superior extremity determines the agent to  
152 learn only the most recent information. In deterministic environments,  
153 usually  $\alpha$  takes values closer or equal to 1, while in worlds with stochastic  
154 transition dynamics, lower values are preferred.
- 155 •  $d$  - is the discount factor and it determines how much does a future reward  
156 influence the present one. As with the learning rate parameter, the dis-  
157 count factor takes values between 0 and 1, the inferior extremity making  
158 the agent not to consider future rewards at all, while values closer to the  
159 superior extremity will determine the agent to aim for the long-term high  
160 reward.

### 161 2.2.2. State-Action-Reward-State-Action (SARSA)

162 State-Action-Reward-State-Action is another reinforcement learning method  
163 in which the agent learn an optimal state-action policy using an on-policy strat-  
164 egy. Q-learning uses a off-policy strategy learning the optimal policy with disre-  
165 gard to the actual exploration that is being carried out. Sometimes the actions  
166 of an agent can generate large negative rewards, thus the strategy to update the  
167 value of the policy according to the exploration path it took can become and  
168 improvement. The latter strategy refers to off-policy learning, and SARAS is

169 an algorithm that learns in this way. Algorithm 2 describes the computational  
170 steps of the SARSA method.

---

**Algorithm 2** SARSA

---

```
function SARSA( $s_{initial}, s_{terminal}, \alpha, d$ )  
2:   initialize  $Q[S, A]$   
       $s \leftarrow s_{initial}$   
4:   select  $a$   
      while  $s! = s_{terminal}$  do  
6:      $r = R(s, a)$   
        $s' = T(s, a)$   
8:     select  $a'$   
        $Q[s, a] \leftarrow Q[s, a] + \alpha(r + d \cdot Q[s', a'] - Q[s, a])$   
10:     $s \leftarrow s'$   
        $a \leftarrow a'$ 
```

---

171 The difference between SARSA and Q-learning can be seen in the utility  
172 value update function, SARSA choosing the utility from taking action from the  
173 exploration path rather than the best one. The rest of the parameters remain  
174 the same as the ones in the Q-learning algorithm.

### 175 2.2.3. Monte-Carlo Technique

176 Monte-Carlo Tree Search is a famous artificial intelligence algorithm, that  
177 runs a number of fast random sampled simulations to expand the tree with  
178 promising moves. Q-learning and SARSA methods have conceptual roots in the  
179 Monte-Carlo techniques, but one optimization could refer to the utility value  
180 update function. Instead of computing the utility of the action-state at the  
181 moment the action occurred, all the decisions will be remembered in a list and  
182 at when it arrived at a terminal state the utilities would be updated in the  
183 reverse order of apparition. This technique could fasten approximation of the  
184 optimal policy, but if not analyzed carefully, with regard to the problem at  
185 hand, it could create delusions for the agent.

## 186 3. Reinforcement Learning Model for Scheduling

187 Scheduling concurrent tasks, that have dependencies between each other,  
188 in a distributed heterogeneous system of computers is a complex and difficult  
189 endeavor. To be able to solve this problem, it must be reduced at a much simpler  
190 level, without losing sight of the core behavior and model. Keeping in mind  
191 that a simple problem is easier to solve, in this section the scheduling problem  
192 in distributed systems is defined in a formal manner, presenting afterwards the  
193 codification of the scheduling process under the form of three additive layers  
194 that add more complexity and come closer to the scheduling problem. The  
195 abstract model of the scheduling process and the reinforcement learning aspects  
196 will be formally defined and explained for each cumulative layer.

197 *3.1. Scheduling Problem Definition*

198 Let there be  $n$  nodes, physical computers or virtual machines, that are con-  
 199 nected through a network and can communicate with each other, with  $n \in \mathbb{N}^*$ .  
 200 Each node  $N_i$  has a set of attributes  $\langle nrpe_i, mips_i, ram_i, storage_i, bw_i \rangle$ , with  
 201  $1 \leq i \leq n$ , where:

- 202 •  $nrpe_i$  the number of processing elements of node  $N_i$ ,  $nrpe_i \in \mathbb{N}^*$ ;
- 203 •  $mips_i$  the computational power of node  $N_i$ , measured in million instruc-  
 204 tions per second,  $mips_i \in \mathbb{N}^*$ ;
- 205 •  $storage_i$  the storage capacity of node  $N_i$ , measured in bytes,  $storage_i \in$   
 206  $\mathbb{N}^*$ ;
- 207 •  $bw_i$  the communication bandwidth of node  $N_i$ , measured in Megabits per  
 208 second,  $bw_i \in \mathbb{N}^*$ .

209 Let  $T$  be a set of tasks, that defines a job, and  $V$  be a set of edges cor-  
 210 responding to a directed acyclic graph (DAG), where the tasks are nodes and  
 211 the edges represent the dependencies between the tasks, with  $|T| \in \mathbb{N}^*$  and  
 212  $|V| \in \mathbb{N}^*$ . Then  $v(t_i, t_j)$  represents an edge in the graph and it tells that task  
 213  $t_j$  is dependent on task  $t_i$ , with  $t_i \neq t_j$ .

214 The function  $c(t_i, n_j)$ , defined as  $c : T \times N \rightarrow \mathbb{R}_+$ , returns the execution  
 215 time of task  $t_i$  that ran on the machine  $n_j$ , with  $t_i \in T$  and  $n_j \in N$ .

216 The function  $d(v(t_k, t_p), n_i, n_j)$ , defined as  $d : V \times N \times N \rightarrow \mathbb{R}_+$ , returns the  
 217 communication time between task  $t_k$  and  $t_p$ , while  $t_k$  is running on  $n_i$  and  $t_p$   
 218 is running on  $n_j$ , with  $v < t_k, t_p > \in V$  and  $n_i, n_j \in N$ .

219 A task assignment schedule  $P$  is described as a tuple of  $\langle Pt, Pv \rangle$ , where  
 220  $Pt$  contains  $n$  subsets of tasks and  $Pv$  contains  $n$  subsets of edges such that:

- 221 -  $\forall t_k \in Pt_i, t_k \notin Pt_j$ , with  $i \neq j$  and  $1 \leq i, j \leq n$  and
- 222 -  $\forall v(t_k, t_p) \in Pv_i, t_p \notin Pt_j$ , with  $i \neq j$  and  $1 \leq i, j \leq n$ .

223 An example of a DAG in which the tasks have been assigned to the nodes is  
 224 presented in Figure [1](#). The labels  $t_1 \dots t_{10}$  represent the task and the nodes of  
 225 the DAG, while the edges show the dependencies between the tasks. After the  
 226 scheduling process, as it can be observed from the picture, the tasks found in the  
 227 red rectangle will be executed by *machine*<sub>1</sub>, the tasks from the blue rectangle  
 228 will be executed by *machine*<sub>2</sub>, and the tasks from the red rectangle will be  
 229 executed by the *machine*<sub>3</sub>. This was an example of a scheduling solution.

The time elapsed to execute all the tasks from the subset of node  $N_i$ , after  
 all of the tasks have been assigned and a schedule  $P$  has been formed, can be  
 expressed as:

$$time_i = \sum_{t_x \in Pt_i} c(t_x, n_i) + \sum_{v(t_k, t_y) \in Pv_i} d(v < t_k, t_y >, n_j, n_i) \quad (2)$$

230 Finding the optimal task assignment schedule  $P$  that minimizes the max-  
 231 imum of the  $time_i$ , represents the definition of task scheduling problem. The  
 232 objective of the machine learning box scheduler is to learn to schedule tasks in

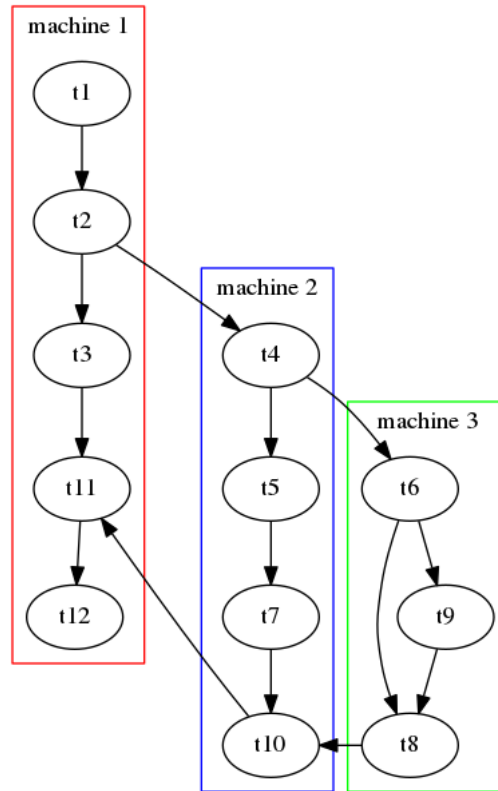


Figure 1: Task DAG example.

233 order to obtain an optimal schedule for a given cluster of machines, with the ob-  
 234 servation that each machine has its own internal process scheduler. The smaller  
 235 the execution time from the slowest queue the more efficient the scheduler is, a  
 236 queue being a list of tasks assigned to a certain node. Initial it is considered that  
 237 there is only one job running in the entire system, but adding the complexity  
 238 of multiple already running jobs would mean to add dynamic attributes to the  
 239 nodes, such as load, status etc.

### 240 3.2. First Layer of Complexity: task DAG scheduling and machine performance

241 The most basic scheduling problem definition has a set of nodes or machines,  
 242 that have their characteristics similar to the ones found in modern grid systems,  
 243 and a task DAG that needs to be scheduled such that the entire dag achieves  
 244 optimal execution time. For simplicity, it is firstly presumed that all the tasks  
 245 from the DAG have similar execution times and resource requirements, such  
 246 that if all of those tasks would execute on a single machine sequentially, the  
 247 execution time of each task would be approximately the same. This means that



248 there is no variation in task execution caused by the internal structure of the  
 249 tasks themselves. An optimal total execution time is going to be determined  
 250 by the correct disposition of the tasks onto the nodes such that the hardware  
 251 infrastructure and the DAG layout are properly exploited.

252 Usually reinforcement learning agents have the possibility to move in a world  
 253 and interact with the entities that reside in that world. But the scheduling  
 254 problem has no environment in which the agent can move. The dynamics of  
 255 the world can be imagined as a group of people that stand at a table, and one  
 256 of them, the agent, must assign a set o papers with math problems. The math  
 257 problems are of the same difficulty and some problems depend on the result of  
 258 others. The job of the agent is to learn how to spread the math problems such  
 259 that all of the tasks finish in the shortest time. The agent has no information  
 260 regarding the capabilities of the people that solve problems but he knows that  
 261 they won't change their seats or leave.

262 Having the previous example in mind, the agent represents the entity that,  
 263 at one moment of time, must determine to whom to assign a task. The agent  
 264 will have several episodes to train and find out what disposition of tasks onto  
 265 the nodes is best and obtains the lowest total execution time. But this will  
 266 work only if the tasks have no dependencies. In order to give the agent the  
 267 perception regarding DAG structure, each node must signal, at one moment of  
 268 time, if it has in its queue a task that represents the parent of this current task  
 269 and if it has tasks that have the same parent, meaning they could be executed in  
 270 parallel. The last information that the agent needs, is the number of tasks that  
 271 he assigned to each node at the moment of time when it must decide where to  
 272 assign the current task. The number of tasks is too precise metric and creates a  
 273 large space of possible world states, a better solution being the introduction of  
 274 a precision factor telling the agent the percentage of tasks assigned relative to  
 275 the total number of tasks. Now that them main elements have been identified  
 276 the formal definition is as follows:

277 Let  $n$  be the number of nodes or machines in the computer cluster. Given a  
 278 precision  $p$  and a list of  $m$  tasks, that have dependencies, the scheduling problem  
 279 is defined as the finding of the best scheduling scheme for the tasks assignment  
 280 to the execution queues such that the total execution time to be minimum.  
 281 Figure 2 holds a visual representation of the main conceptual model.

The states  $S$  of the world are defined as follows:

$$S = \{\langle load\_level_i, parent_i, sibling_i \rangle | 1 \leq i \leq n\}. \quad (3)$$

282 where:

- 283 •  $load\_level_i$  represents the number of tasks currently assigned to queue  $q_i$   
 284 relative the given percentage precision  $p$ ,  $load\_level_i \in \mathbb{N}$ ;
- 285 •  $parent_i$  informs if in  $q_i$  a father of the current task resides,  $parent_i =$   
 286  $\{true, false\}$ ;
- 287 •  $sibling_i$  informs if in  $q_i$  there are sons of the same father,  $sibling_i =$   
 288  $\{true, false\}$ .

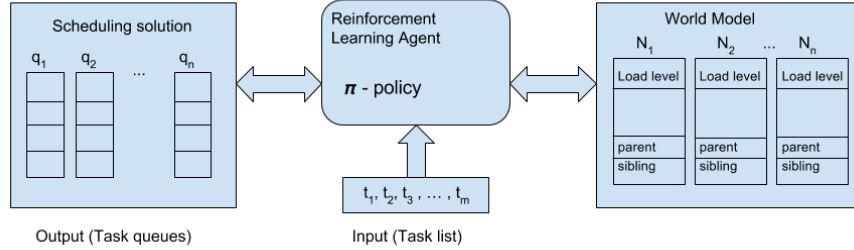


Figure 2: Conceptual model with first layer of complexity.

289 The set of actions  $A$  is represented by the index of the node to which the  
 290 current task is going to be assigned.

291 The transition dynamics  $T$  has a hidden world model, where precision of task  
 292 numbers is highest, but it offers through the simplification of the states a smaller  
 293 world space. When an action takes place the hidden world is consulted and the  
 294 new corresponding state is returned. Even though the simplified world may  
 295 seem non-deterministic, due to the deterministic nature of the hidden world,  
 296 every action will determine the transition to a single state. The terminal state  
 297 is characterized by an empty list of tasks to schedule.

298 The reward  $r$  will remain null throughout the whole scheduling process, with  
 299 the exception of the terminal state, when the execution time of the solution is  
 300 compared with a base value and the reward either gains a positive or a negative  
 301 value, depending on the performance of the execution schedule.

302 The reinforcement learning agent is going to learn a policy on how to schedule  
 303 the current task knowing the placement of other tasks that he might depend  
 304 on, or tasks that can possibly run in parallel with, through the  $parent_i$  and  
 305  $sibling_i$  attributes. After a number training episodes the agent will also learn  
 306 which node has more computational power and try to assign more tasks on its  
 307 queue, but also taking in consideration the dependencies between them or the  
 308 opportunity to run concurrently.

309 The advantage of using a precision parameter to define the number of nodes  
 310 from a queue is that, the learning agent and its policy will not be dependent  
 311 on the number of tasks or tasks structure, allowing to test the learn policy on  
 312 various DAGs.

### 313 3.3. Second Layer of Complexity: dynamic cluster status

314 The first layer of complexity presumed that all the task have the same in-  
 315 ternal structure, and that there is only one job running at one time. In reality,  
 316 a modern cluster systems has many jobs running and being scheduled, the true  
 317 load of each machine influencing the quality of the policy learned from the

318 world of the previous section. The advantage of the proposed scheduling so-  
 319 lution is that it allows the extension of the world model without submitting  
 320 heavy changes to the whole algorithm. As seen in Figure 3 each node should  
 321 add a new attribute through which it can inform about its status. Every time the  
 322 distributed system wants to use the agent's policy to schedule, it should update  
 323 the status fields of the world and inform the agent that he can start scheduling.

324 An optimization would be to initially learn a policy without taking into  
 325 consideration the status attribute, and then redistribute the utilities to the new  
 326 scheme hoping that the new attribute would not influence the actual policy very  
 327 much.

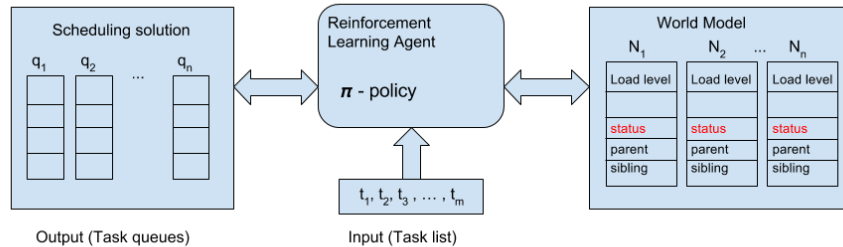


Figure 3: Conceptual model with second layer of complexity.

### 328 3.4. Third Layer of Complexity: variable tasks and task classification

329 The final layer of complexity taken into consideration in this paper, is the one  
 330 regarding internal structure of the task. Until now the model considered that  
 331 all the tasks have the same internal structure, and if all the tasks were to run  
 332 on a single machine, one at a time, each execution time would be approximately  
 333 the same. In the first two layers this was not possible, because the agent had no  
 334 information regarding the task that it had to assign. The policy learned when  
 335 and where to schedule a task to a certain queue, from history and from the  
 336 *load\_level* attribute. The tasks in a job can vary in purpose and functionality  
 337 drastically, having a great influence on the execution time if not placed properly.

338 A task attribute added to the world model would increase the precision of  
 339 the agent's policy and would allow to schedule complex DAGs with variate tasks.  
 340 But in order to better determine the type of a task, another component should  
 341 be added to the main concept, a component that classifies incoming tasks and  
 342 compresses their characteristics, to simplify the world model and reduce the  
 343 number of states. Figure 4 is a visual representation of the concept with the  
 344 task classifier extension. With the addition of the last component the model is  
 345 complete, and should be able to learn scheduling policies that decrease the total

346 execution time of jobs and improve overall performance of distributed systems  
 347 like clusters and grid.

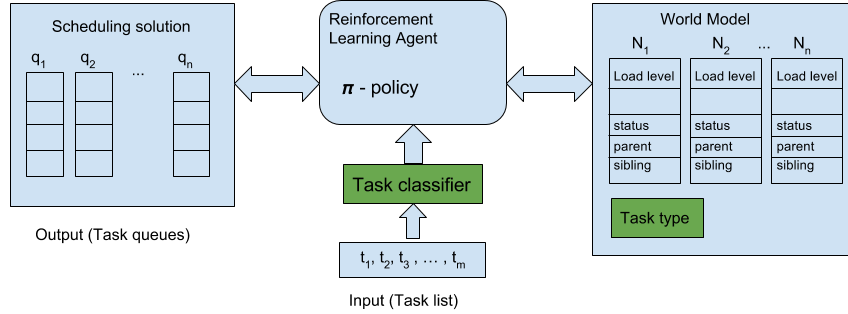


Figure 4: Conceptual model with third layer of complexity.

#### 348 4. Machine Learning Box Architecture

349 The Machine Learning Box (MBox) is an application that offers scheduling  
 350 services to other distributed systems or computer clusters. The scheduling en-  
 351 gines use machine learning algorithms and agents to learn optimal scheduling  
 352 policies for different machine setups. A client must firstly register a domain or  
 353 a world definition. After that it can make scheduling requests of jobs that are  
 354 meant to train the reinforcement learning agents, or it can request fast schedul-  
 355 ing solutions for critical operations. MBox responds to the request with a task  
 356 scheduling scheme, and informs the system if it requires an indication regarding  
 357 the execution time of the scheduled set of tasks. The application is written in  
 358 Java 8 and it has a library of necessary classes and interfaces for the implemen-  
 359 tation of local MBox client. The MBox client uses Java RMI to communicate  
 360 with the MBox service module, and must request a valid remote instance of  
 361 a MBox scheduler. MBox schedulers are initialized through a command line  
 362 interface from the server side, that can start monitoring tools for performance  
 363 and system status analysis.

364 Figure 5 shows a simple architectural model of the Machine Learning Box of  
 365 the basic structural and functional components. The application is designed to  
 366 support parallelism and easily scale into a big system that could handle many  
 367 clients. Each major component runs, in the demo application, in a separate  
 368 thread, this separation giving the system the capability to eventually move  
 369 each component on dedicated servers. The domain database can be moved to a  
 370 system that permits data replication and has integrated consistency and backup  
 371 protocols, so that the already learned policies and registered world models are  
 372 never lost and are always accessible. The possibility to run simultaneously many

373 reinforcement learning agents can speed up the learning process and can lead  
 374 to faster ways to find optimal policies on large world models.

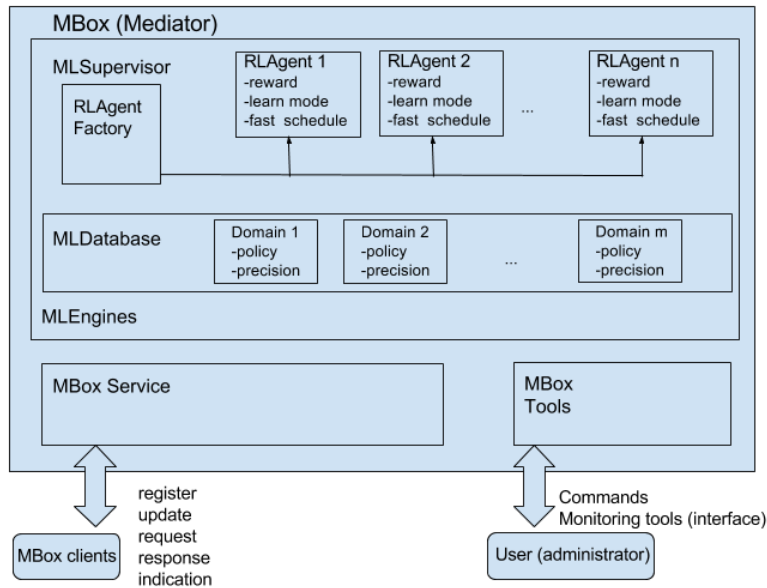


Figure 5: Visual representation of the MBox architecture, encapsulating the main components of the platform, their relationship and the interfaces provided.

375 Following the mediator design pattern, the demo application has all the  
 376 major components incorporated in a single class, through which communication  
 377 is assured. The Mediator starts all of the other components at initialization  
 378 and waits for their graceful termination when the program is ended. It also  
 379 represents a communication medium and it can be replaced by physical software  
 380 with little modification to the rest of the components.

381 The MBox has a command line interface used for configuration and perform-  
 382 ance monitoring. A user with administrative clearance can create, destroy or  
 383 display the status of the machine learning agents. The remote scheduler objects  
 384 are also created and managed through that interface. The performance of the  
 385 system currently relies on the utility tools offered by the BURLAP library, but  
 386 they only allow to observe the machine learning elements not the entire system.  
 387 The command line interface can be remotely accessed through already secure  
 388 ssh connections, via the terminals offered by the operating system. A command  
 389 line interface was preferred instead of a graphical user interface, because it can  
 390 be remotely accessed for configuration without needing X11 sessions or other  
 391 graphical engines, scripts and wrappers can be created to automate certain tasks  
 392 and from the perspective of the system administrator it offers versatility.

393 The clients that want to use the scheduling service must firstly make a re-  
 394 quest, to the administrators or owners of the machine learning box instance,

395 for the allocation of a number of MBox schedulers. Then, by using a library  
 396 of common classes and interfaces, they must implement and integrate a custom  
 397 unit that uses the remote allocated MBox scheduler to register new domains,  
 398 update existing ones, make scheduling requests for jobs and indicate the execu-  
 399 tion time of the scheduling solution. If the remote scheduler fails, then it is the  
 400 duty of the local unit to deal with it, providing alternatives. As future work,  
 401 the integration of learning from logs could prove to be useful, as the learning  
 402 agents could learn from past solutions without having the need to physically  
 403 test the scheduling solution.

404 In the following subsections all the major components of the machine learn-  
 405 ing box application will be presented, offering details about the implementation  
 406 and the design decisions.

#### 407 4.1. MBox Service

408 The role of the MBox Service component and Java class is to allow clients to  
 409 connect to the main application and acquire the offered services. It holds and  
 410 manages a list of MBox Scheduler objects, that represent the remote objects  
 411 that the client will use. Figure 6 depicts a visualization of the structural design  
 412 of the MBox Service, with regard to the external components and entities with  
 413 whom it communicates. The interaction with the rest of the classes is realized  
 414 through the MBox mediator and from whom it receives command to add new  
 415 **MBox Scheduler** objects or remove existing ones.

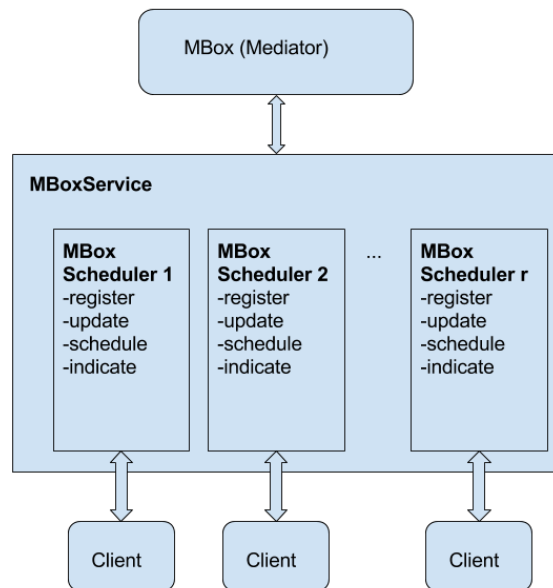


Figure 6: Visual representation of the MBox Service, of its internal structure and of its interaction with the clients.

416 A client does not interact directly with MBox Service, instead it looks for  
417 the methods of a remote MBox Scheduler, calling them according to its needs.  
418 The remote scheduler object then checks the validity of the call and proceeds  
419 to execute the corresponding action with the help of the MBox Service. In  
420 the demo application, The MBox Scheduler offers to the client four methods:  
421 **register**, **update**, **schedule** and **indicate**.

- 422 • **register** method receives as a parameter an object that describes the  
423 domain or world of the scheduling problem, that holds information about  
424 the nodes (how many, what are their characteristics, dynamic properties  
425 like status etc), and about the precision of the world model; When it is  
426 called, the designated MBox Scheduler send upward the command to the  
427 MBox Service, to create a new MLDomain and store it in the MLDatabase;
- 428 • **update** method receives as a parameter an update object that describes  
429 the domain or world of the scheduling problem, that updates an already  
430 created MLDomain from the MLDatabase;
- 431 • **schedule** method receives as a parameter a MBox request object, that  
432 stores the list of task that need to be scheduled and other information  
433 related to the type scheduling (learning mode or fast mode); it returns  
434 a response object, that contains the status of the scheduling (successful,  
435 learning, failed) and a scheduling solution of the tasks;
- 436 • **indication** is used to inform the scheduler about the execution time of  
437 the solution that it provided; It is essential for a learning job to send that  
438 information to the MBox Scheduler, so that it can transmit that data  
439 forward to the learning agent, in order to finish the learning process and  
440 estimate a reward;

#### 441 4.2. MBox Machine Learning Engine

442 The MBox Machine Learning Engine is the core of the MBox application,  
443 incorporating the environment for creating and running reinforcement learn-  
444 ing agents, world domains and tasks scheduling algorithms. The module was  
445 designed keeping in mind the benefits of parallel computing and the versatil-  
446 ity of the application. The two sub-components, the **MLSupervisor** and the  
447 **MLDatabase**, that define the functionality of this module, are placed in the  
448 same module in the demo application, but can be separated, so that they can  
449 run more efficiently. The MLSupervisor can be placed on a high performance  
450 parallel system, in order to exploit the advantages of running multiple learning  
451 agents concurrently on different threads and on different machines, while the  
452 MLDatabase can be placed on a distributed system that offers advanced stor-  
453 age techniques with accent being put on data replication and consistency. In  
454 Figure 7 there can be seen the structural disposition and the relation between  
455 the elements of the MBox Machine Learning Agent. This major component has  
456 been written in Java 8 and uses the BURLAP library for the implementation

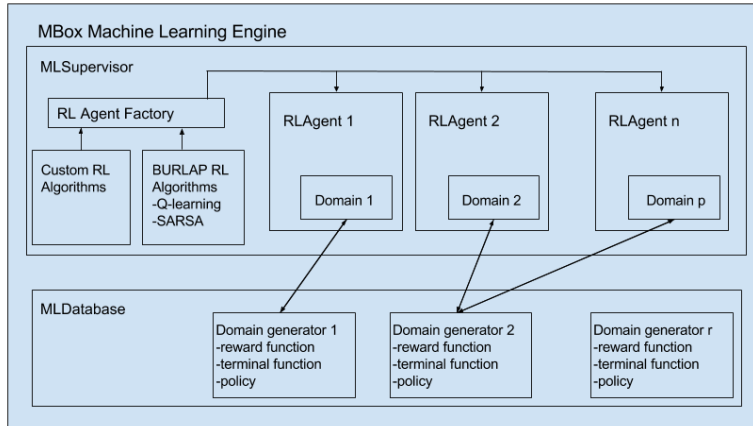


Figure 7: Visual representation of the MBox Machine Learning Engine, encapsulating the communication between the domains from the MLDatabase and the learning agents, monitored by the MLSupervisor.

457 of the world models, the machine learning agents and reinforcement learning  
 458 algorithms.

459 BURLAP is a java code library for developing scheduling and learning algo-  
 460 rithms. It offers a highly flexible framework for defining states and actions, sup-  
 461 porting discrete, continuous and relational domains. There are several schedul-  
 462 ing and learning algorithms implemented in the library and ready to use, and  
 463 it allows the extension and creation of new ones. It also contains a suite of  
 464 analysis tools for the visualization of the domains and for the performance of  
 465 the running agents.

466 The role of the MLDatabase is to store scheduling world models in the form  
 467 of Domain Generators. Each Domain Generator describes a different model of  
 468 the node architecture in a cluster of computers or groups of virtual machines in  
 469 which job are scheduled in the form of task assignment. When a client invokes  
 470 the register method, the MBoxService sends a command to the MLDatabase,  
 471 which check if that domain wasn't registered before, and if it hasn't it register  
 472 it in the form of a Domain Generator. The Domain Generator contains all the  
 473 information from the parameter of the register method, the reward function, the  
 474 terminal function and the scheduling policy. The reward function is activated  
 475 when the agent is in learning mode and needs to asses the performance of a  
 476 scheduling solution, which is done by comparing the current execution time  
 477 with a baseline execution time. Initially the base time will be uninitialized and  
 478 the first solution will become the baseline execution time. The reward will be  
 479 calculated as the subtraction between the baseline and the current execution  
 480 time. The sign of the result tell if the reward was a positive one or a negative  
 481 one. The advantage using Domain Generator comes from the possibility to  
 482 exploit parallel computing and generate new domains for each agent that wants  
 483 to start a learning process. The combination of the policy utilities learned from



484 agents that have run on similar domains represents an important factor in the  
485 performance of the entire platform.

486 The MLSupervisor has the role of creating, initializing and running new  
487 learning agents, when the MBoxService has a MBoxScheduler invoking the  
488 schedule method. This module creates a new thread for each learning agent,  
489 gives them their respective domain and starts the learning process. The demo  
490 application does not have the possibility to select the machine learning schedul-  
491 ing algorithm on a schedule request, but for future works this functionality  
492 could be added. The reinforcement learning algorithms implemented in the  
493 BURLAP library are compatible with the definition of the scheduling prob-  
494 lem model. Nevertheless, the MLSupervisor permits custom implementation of  
495 planning and reinforcement learning algorithms, due to the high flexibility of  
496 the BURLAP library.

#### 497 *4.3. MBox Scheduling with WorkflowSim*

498 A use scenario would firstly imply an administrative user from the server  
499 side to create a new MBoxScheduler. That can be realized through the MBox  
500 application command line interface. It is presumed that the Machine Learning  
501 Box application is already running. After the remote object was created and  
502 initialized, it is time for the client to do its part. The client must implement  
503 a custom module in the distribute system, using the compatibility library from  
504 the MBox repository. An example of an implementation will be presented later  
505 in this section. After the implementation the custom module should find the  
506 remote methods and obtain access to them. The clients module should firstly  
507 register the world model used in task scheduling. This is done also through  
508 the use of the classes and interfaces from the MBox library. After a registry  
509 request the custom module can send schedule requests to the remote MBox  
510 Scheduler. Initially the reinforcement learning agent will not return efficient  
511 task assignment schedules. In order for it to become more intelligent it must  
512 learn, and this is realized through learning job, defined as jobs that will run  
513 on the system for a large number of times, or through the analysis of the logs.  
514 The last form of learning has not been implemented and consists an idea for  
515 future work. Given enough time and enough learning episodes the reinforcement  
516 learning agents will become more proficient at realizing efficient task assignment  
517 schedules and so improve the performance of the client system.

518 To test the MBox application, the WorkflowSim 1.0 was used. WorkflowSim  
519 is an open source simulator of workflows represented as DAGs. It can simulate  
520 large concentrations of nodes that form heterogeneous systems, node delays and  
521 even node failure. Using this simulation platform the MBox Scheduler can be  
522 tested without causing any harm to real computer clusters or grid systems.  
523 WorkflowSim comes with a rich set of jobs organized as directed acyclic graphs  
524 with different disposition of tasks, inspired from real scientific applications, that  
525 can be represented as a workflow.

526 **5. Results**

527 The section contains the observations made upon the reinforcement learning  
528 model used in the MBox application, and will firstly consider the theoretical  
529 expectations, followed by the experimental results. The demo application had  
530 only the first layer of complexity, that was described in section 3, implemented.

531 *5.1. Theoretical Limit*

532 Considering the first layer of complexity, the model had the following pa-  
533 rameters:

- 534 •  $n$  heterogeneous nodes, with  $n \in \mathbb{N}^*$ ;
- 535 •  $m$  tasks that form a DAG of dependencies, with  $m \in \mathbb{N}^*$ ;
- 536 • precision  $p$ , with  $p \in \mathbb{N}^*$ ;
- 537 • a set of states  $S = \{\langle load\_level_i, parent_i, sibling_i \rangle | 1 \leq i \leq n\}$ .

538 If  $val(x)$  = list of all possible values  $x$  can take, then  $|val(load\_level_i)| = p$ ,  
539  $|val(parent_i)| = 2$ ,  $|val(sibling_i)| = 2$ .

Given the parameters above, the number of states a node can have can be calculated:

$$|S_i| = p \cdot 2 \cdot 2 = 4p \tag{4}$$

Given the number of states for a single node, the number of world states can be calculated, knowing that the world state is a concatenation of the state of all the nodes:

$$|S| = (4p)^n \tag{5}$$

From the last result, it can be deduced that the number of states grows at a magnitude given by the number of nodes; For example, if there is a cluster with  $n = 10$  and  $p = 10$ , then:

$$|S| = (4 \cdot 10)^{10} = 4^{10} \cdot 10^{10} = 1.048576 \cdot 10^{16} \tag{6}$$

540 The conclusion is that the number of states grows too fast to the number  
541 of nodes from the cluster, for an agent or a group of agents to properly learn,  
542 using reinforcement learning. For a smaller cluster the reinforcement learning  
543 agent would be able to find the optimal policy, the number of tasks in a job  
544 accelerating the learning process.

545 *5.2. Experimental results*

546 Even it is hard to measure the performance of reinforcement learning al-  
547 gorithms, one form of evaluation might give some valuable insight. The plot  
548 of the cumulative reward as a function of the number of steps tells how fast  
549 and how good is the policy that the agent deduced after a certain amount of  
550 steps. The slope of the plot tells how good is the policy after it stabilized, the  
551 descending portion shows how much reward was wasted before it could improve

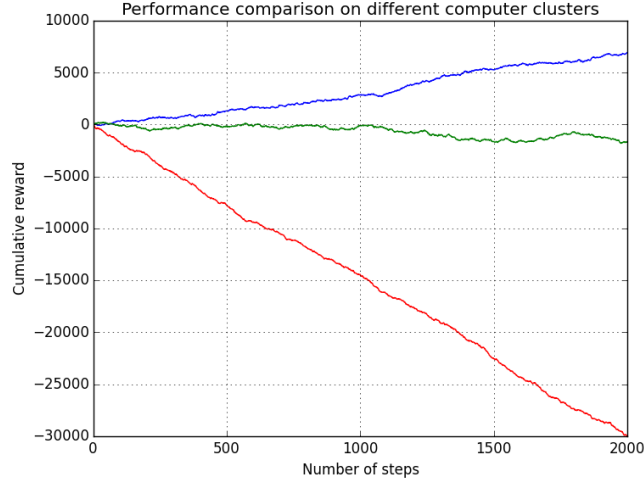


Figure 8: Cumulative reward performance evaluation of Q-learning algorithm. 2 nodes - blue, 4 nodes - green, 8 nodes - red.

552 and the point of intersection with zero shows how much time took the algorithm  
 553 to recuperate the lost reward.

554 Figure 8 depicts a comparison in performance of Q-learning algorithm on  
 555 three scenarios: a cluster formed of two nodes (blue plot), a cluster formed of  
 556 4 nodes (green plot) and a cluster formed of 8 nodes (red plot). The cluster  
 557 are heterogeneous and the training job remained constant through the steps.  
 558 It is clear that the more nodes are added to the distributed system the more  
 559 hard it got for the agent to learn a good policy. This reflects the theoretical  
 560 observations.

561 Figure 9 shows a comparison between the Q-learning algorithm (red plot)  
 562 and SARSA (blue plot) on a cluster formed of two nodes. Experimental results  
 563 have shown that SARSA behaves better than Q-learning, but it must be taken  
 564 into consideration the fact that the reward is dynamically calculated in the  
 565 first step, becoming the baseline for future steps with great influence on the  
 566 utility distribution of the policy values. If the baseline sets high standards the  
 567 majority of the rewards will be negative, while low initial baseline value could  
 568 lead to higher utility values.

569 Given enough time the reinforcement learning agents using the two algo-  
 570 rithms are able to find better solutions for a given job, determining faster ex-  
 571 ecutions even than classic algorithms. The results from Figure 10 depict an  
 572 experiment in which a job, composed of 100 tasks, runs multiple times on a  
 573 heterogeneous cluster of four nodes, using Q-learning, SARSA and HEFT as  
 574 scheduling algorithms. After each step, that comprised of 100 iterations, the  
 575 best solution of each reinforcement learning method is selected and the job is  
 576 run again, the learning agents switching from a dynamically balanced policy

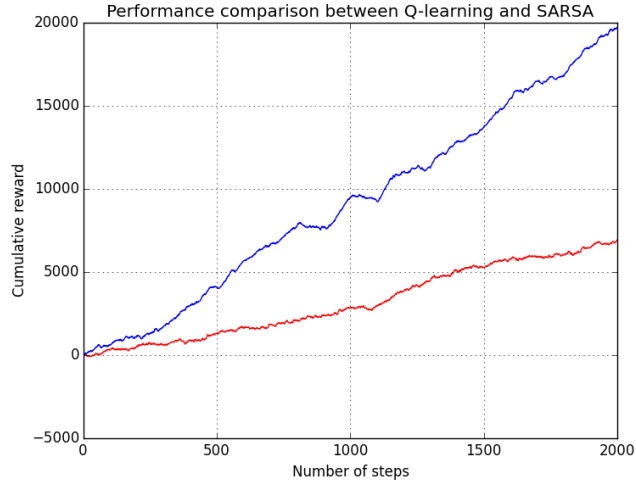


Figure 9: Q-learning (blue) vs SARSA (red).

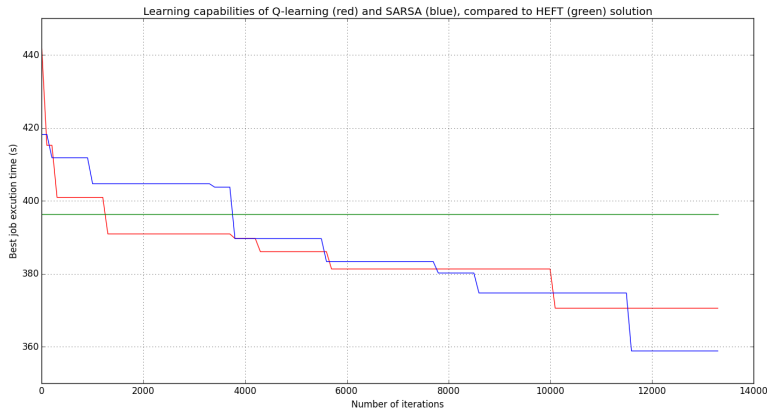


Figure 10: Learning capabilities of Q-learning (red) and SARSA (blue), compared to HEFT (green) solution.

577 between exploration and exploitation to only exploitation, thus obtaining the  
 578 time of the best solution found.

579 The conclusion is that the proposed model has combined the characteristics  
 580 of all the nodes or machines of the cluster, resulting in a huge world of states  
 581 that cannot be properly explored by a reinforcement learning agent. Machine  
 582 Learning algorithms have a tough time dealing with such problems, and in  
 583 order for those techniques to work they would need auxiliary help from other

584 heuristics and strategies. Given the fact that the theoretical model has shown  
585 the limitations of the proposed algorithm, further experimentation would have  
586 been redundant. Regarding the comparison of other scheduling algorithms, most  
587 of other scheduling algorithms do not need iterations to arrive at a more mature  
588 state. The performance of the reinforcement learning method will be lower at  
589 the beginning, but it will surpass the classic algorithms after an enough number  
590 of iterations.

## 591 **6. Conclusion**

592 As computer clusters and distributed systems become more and more popu-  
593 lar, the need to improve the performance of such systems becomes a challenge,  
594 that if properly mastered could accelerate the evolution of science or could re-  
595 set the positions of industry giants. It is clear that the schedulers from such  
596 systems have a certain impact on the efficiency of parallel systems. Machine  
597 learning and artificial intelligence are gaining ground, intelligent solutions, that  
598 learn from the past and adapt, will become the norm when dealing with com-  
599 plex problems. Task schedulers in distributed systems would benefit greatly  
600 from intelligent agents, learning from past mistakes, exploring and finding new  
601 solutions that no human might have though before.

602 In this paper, a platform, offering scheduling solutions as a service based  
603 on machine learning agents, was described, and a reinforcement learning world  
604 model for scheduling was proposed. The platform, know as the Machine Learn-  
605 ing Box, allows further development of scheduling algorithms and an easy inte-  
606 gration process. The application model can easily be mapped on parallel sys-  
607 tems, in order to scale and increase the overall efficiency. The learning model  
608 proved to have its limitations, due to the complex nature of a distributed sys-  
609 tem and the proposed codification as a world of states. While this codification  
610 works on smaller systems, the more nodes were added to the system the larger  
611 the world got, leaving the reinforcement learning agent incapable of properly  
612 learning an optimal policy.

613 For future work the platform could be extended to support other types of  
614 algorithms and scheduling methods. Naturally the efficiency and bottlenecks of  
615 a parallel implementation of the platform could be analyzed. As for the rein-  
616 forcement learning used in scheduling in distributed systems, other techniques  
617 should be experimented, as well as other world models that could reduce the  
618 number of states and enhance the method. Worth exploring would be a model  
619 that does not combine the states of each node of the cluster, but creates indi-  
620 vidual policies that give utilities to the action of refusing or accepting a task to  
621 be assigned.

## 622 **Acknowledgment**

623 The research presented in this paper is supported by projects: *DataWay*:  
624 Real-time Data Processing Platform for Smart Cities: Making sense of Big Data

625 - PN-II-RU-TE-2014-4-2731; *MobiWay*: Mobility Beyond Individualism: an In-  
626 tegrated Platform for Intelligent Transportation Systems of Tomorrow - PN-II-  
627 PT-PCCA-2013-4-0321; *CyberWater* grant of the Romanian National Authority  
628 for Scientific Research, CNDI-UEFISCDI, project number 47/2012; *clueFarm*:  
629 Information system based on cloud services accessible through mobile devices,  
630 to increase product quality and business development farms - PN-II-PT-PCCA-  
631 2013-4-0870.

632 We would like to thank the reviewers for their time and expertise, construc-  
633 tive comments and valuable insight.

## 634 References

- 635 [1] R. Kumar, N. Gupta, S. Charu, S. K. Jangir, Architectural paradigms of  
636 big data, in: National Conference on Innovation in Wireless Communica-  
637 tion and Networking Technology–2014, Association with the Institution of  
638 Engineers (INDIA), 2014.
- 639 [2] T. White, Hadoop: The Definitive Guide, 3rd Edition, O’Reilly - O’Reilly  
640 Media, 2012.
- 641 [3] J. Dean, S. Ghemawat, Mapreduce: simplified data processing on large  
642 clusters, Communications of the ACM 51 (1) (2008) 107–113.
- 643 [4] A. C. Murthy, V. K. Vavilapalli, D. Eadline, J. Niemiec, J. Markham,  
644 Apache Hadoop YARN: Moving Beyond MapReduce and Batch Processing  
645 with Apache Hadoop 2, Pearson Education, 2013.
- 646 [5] J. D. Ullman, Np-complete scheduling problems, Journal of Computer and  
647 System sciences 10 (3) (1975) 384–393.
- 648 [6] P. Liu, K.-K. R. Choo, L. Wang, F. Huang, Svm or deep learning? a  
649 comparative study on remote sensing image classification, Soft Computing  
650 (2016) 1–13.
- 651 [7] L. Wang, J. Zhang, P. Liu, K.-K. R. Choo, F. Huang, Spectral–spatial  
652 multi-feature-based deep learning for hyperspectral remote sensing image  
653 classification, Soft Computing (2016) 1–9.
- 654 [8] Y. Hu, J. Yan, K.-K. R. Choo, Pedal: a dynamic analysis tool for efficient  
655 concurrency bug reproduction in big data environment, Cluster Computing  
656 19 (1) (2016) 153–166.
- 657 [9] Z. Xu, H. Zhang, V. Sugumaran, K.-K. R. Choo, L. Mei, Y. Zhu, Participa-  
658 tory sensing-based semantic and spatial analysis of urban emergency events  
659 using mobile social media, EURASIP Journal on Wireless Communications  
660 and Networking 2016 (1) (2016) 44.
- 661 [10] J. MacGlashan, The brown-umbc reinforcement learning and planning  
662 (burlap) java code library (<http://burlap.cs.brown.edu/>) (2016).

- 663 [11] D. E. Hershkowitz, J. MacGlashan, S. Tellex, Learning propositional functions for planning and reinforcement learning, in: 2015 AAAI Fall Symposium Series, 2015.  
664  
665
- 666 [12] J. MacGlashan, M. L. Littman, Between imitation and intention learning, in: Proceedings of the 24th International Conference on Artificial Intelligence, AAAI Press, 2015, pp. 3692–3698.  
667  
668
- 669 [13] W. Chen, E. Deelman, Workflowsim: A toolkit for simulating scientific workflows in distributed environments, in: E-Science (e-Science), 2012 IEEE 8th International Conference on, IEEE, 2012, pp. 1–8.  
670  
671
- 672 [14] F. Pop, C. Dobre, V. Cristea, Genetic algorithm for dag scheduling in grid environments, in: Intelligent Computer Communication and Processing, 2009. ICCP 2009. IEEE 5th International Conference on, IEEE, 2009, pp. 299–305.  
673  
674  
675
- 676 [15] N. R. Satish, K. Ravindran, K. Keutzer, Scheduling task dependence graphs with variable task execution times onto heterogeneous multiprocessors, in: Proceedings of the 8th ACM international conference on Embedded software, ACM, 2008, pp. 149–158.  
677  
678  
679
- 680 [16] W. Zhang, T. G. Dietterich, A reinforcement learning approach to job-shop scheduling, in: IJCAI, Vol. 95, Citeseer, 1995, pp. 1114–1120.  
681
- 682 [17] Z. Peng, D. Cui, J. Zuo, Q. Li, B. Xu, W. Lin, Random task scheduling scheme based on reinforcement learning in cloud computing, Cluster Computing 18 (4) (2015) 1595–1607.  
683  
684
- 685 [18] N. George, K. Chandrasekaran, A. Binu, An objective study on improvement of task scheduling mechanism using computational intelligence in cloud computing, in: 2015 IEEE International Conference on Computational Intelligence and Computing Research (ICCIC), IEEE, 2015, pp. 1–6.  
686  
687  
688
- 689 [19] M. Gombolay, R. Jensen, J. Stigile, J. Shah, Apprenticeship scheduling: Learning to schedule from human experts, in: Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI), New York City, NY, USA, 2016.  
690  
691  
692
- 693 [20] D. L. Poole, A. K. Mackworth, Artificial Intelligence: foundations of computational agents, Cambridge University Press, 2010.  
694
- 695 [21] S. Russle, P. Norvig, Artificial Intelligence A Modern Approach, Third Edition, 2009.  
696

## Biographies



**Alexandru Iulian ORHEAN**, Computer Science diplomat engineer graduated University Politehnica of Bucharest, Faculty of Automatic Control and Computers in 2016, and now he is PhD student at Department of Computer Science at Illinois Institute of Technology. His research interests are in Cloud System and Big Data, especially in resource-aware scheduling, machine learning techniques for distributed systems, multi-criteria optimization, Cloud middleware tools, Big Data applications design and implementation.



**Florin POP** received his PhD in Computer Science at the University POLITEHNICA of Bucharest in 2008. He received his MSc in Computer Science in 2004 and the Engineering degree in Computer Science in 2003, at the same University. He is Associate Professor within the Computer Science Department and also an active member of Distributed System Laboratory. His research interests are in scheduling and resource management, multi-criteria optimization methods, Grid middleware tools and applications development, prediction methods, self-organizing systems, contextualized services in distributed systems. He is the author or co-author of more than 150 publications (books, chapters, papers in international journals and well-established and ranked conferences). He served as guest-editor for *International Journal of Web and Grid Services* and he is Managing Editor for *International Journal of Grid and Utility Computing*. He was awarded with “Magna cum laude” distinction for his results during his PhD, one IBM Faculty Award in 2012 or the project “*CloudWay – Improving resource utilization for a smart Cloud infrastructure*”, two Prizes for Excellence from IBM and Oracle (2008 and 2009), *Best young researcher in software services Award*, FP7 SPRERS Project, Strengthening the Participation of Romania at European R&D in Software Services in 2011 and two Best Paper Awards. He worked in several international (EGEE III, SEE-GRID-SCI, ERRIC) and national research projects in the distributed systems field as coordinator and member as well. He is a senior member of the IEEE, ACM and euroCRIS.



Dr. Ioan Raicu is an associate professor in the Department of Computer Science (CS) at Illinois Institute of Technology (IIT), as well as a guest research faculty in the Math and Computer Science Division (MCS) at Argonne National Laboratory (ANL). He is also the founder (2011) and director of the Data-Intensive Distributed Systems Laboratory (DataSys) at IIT. He has received the prestigious NSF CAREER award (2011 - 2015) for his innovative work on distributed file systems for extreme-scales. He was a NSF/CRA Computation Innovation Fellow at Northwestern University in 2009 - 2010, and obtained his Ph.D. in Computer Science from University of Chicago under the guidance of Dr. Ian Foster in March 2009. He is a 3-year award winner of the GSRP Fellowship from NASA Ames Research Center. His research work and interests are in the general area of distributed systems. His work focuses on a relatively new paradigm of Many-Task Computing (MTC), which aims to bridge the gap between two predominant paradigms from distributed systems, High-Throughput Computing (HTC) and High-Performance Computing (HPC). His work has focused on defining and exploring both the theory and practical aspects of realizing MTC across a wide range of large-scale distributed systems. He is particularly interested in resource management in large scale distributed systems with a focus on many-task computing, data intensive computing, cloud computing, grid computing, and many-core computing. Over the past decade, he has co-authored over 100 peer reviewed articles, book chapters, books, theses, and dissertations, which received over 7248 citations, with a H-index of 35. His work has been funded by the NASA Ames Research Center, DOE Office of Advanced Scientific Computing Research, the NSF/CRA CIFellows program, and the NSF CAREER program. He has also founded and chaired several workshops, such as ACM Workshop on Many-Task Computing on Clouds, Grids, and Supercomputers (MTAGS), the IEEE Int. Workshop on Data-Intensive Computing in the Clouds (DataCloud), the ACM Workshop on Scientific Cloud Computing (ScienceCloud), and IEEE Int. Workshop on Collaborative methodologies to Accelerate Scientific Knowledge discovery in big data (CASK). He is on the editorial board of the IEEE Transaction on Cloud Computing (TCC), the Springer Journal of Cloud Computing Advances, Systems and Applications (JoCCASA), and the Springer Cluster Computing Journal (Cluster). He has been leadership roles in several high profile conferences, such as HPDC, CCGrid, Grid, eScience, Cluster, ICAC, and BDC. He is a member of the IEEE and ACM.