New Scheduling Approach using Reinforcement Learning for Heterogeneous Distributed Systems

Alexandru Iulian Orhean[a], Florin Pop*[a], Ioan Raicu[b]

[a] *Computer Science Department, Faculty of Automatic Control and Computers, University Politehnica of Bucharest, Romania*
[b] *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

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

Preprint submitted to Journal of Parallel and Distributed Computing *March 1, 2017*

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

Supercomputers are formed of dedicated machines, that are connected with each other throughout a well organized and fast network, and have high performances, but usually have high costs and are specialized in solving certain problems. Computer clusters or grid systems made of commodity hardware are the favorite solution both in the industry and in the academia, because of its low costs and highly configurable characteristic, given by the frameworks and platforms that run on the systems. One such framework is the Apache Hadoop Ecosystem [2] that implemented the successful data processing model MapReduce [3], published by Google. The framework has grown over the years integrating components that assure data replication and consistency, fault-tolerance, security, safe execution of scalable parallel applications. One of the most important enhancements to the Hadoop environment was the separation of the resource negotiator, known as YARN [4], the advantage being the ease of customization. This intended versatility of YARN confirms the importance of the scheduling process in the efficiency of the system and of the applications.

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

Machine learning is a vast domain of artificial intelligence, that has grown in popularity because of its simple recipes or algorithms that give programs the capability to learn patterns, behavior, models and functions, and use that information to make better decisions or actions in the future. Machine learning can be classified as: supervised learning, where a training data set is given and the agent learns how to predict output values of certain input targets; unsupervised learning, where an agent learns a certain structural organization of the input data or the relationship of the elements of the data set; and reinforcement learning, where an agent is given certain rewards corresponding to the utility of an action or decision relative to the world model, with which the agent interacts. Neural networks and deep learning are the most prominent concepts that roam in artificial intelligence, as a result of their capacity to find more efficient solutions than heuristic approaches. These are used in many classification problems [6, 7, 8, 9]. Regarding the problem of task scheduling in distributed systems, the machine learning box will use reinforcement learning algorithms to schedule the tasks in the given cluster of computers.

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

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

2. Background and related work

2.1. Related work

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

Experiments show that machine learning algorithms can achieve great performances in scheduling tasks. Temporal difference, a classic reinforcement learning algorithm, has been shown to be able to solve the scheduling problem [16], but with the help of a neural network that learned the evaluation functions over states. Other investigations have shown that queuing models combined with reinforcement learning techniques permit optimization of the tasks scheduling process at a finer granularity [17]. Mechanism that learn best scheduling strategies [18], from a list of methods that were created to improve certain metrics in cloud computing, have also been proposed, letting an agent decide from past experiences which strategy is more appropriate giving a set of conditions. Exotic research, circumvent the process of learning scheduling policies from past experiences by interacting with the environment or from other heuristic strategies, but from expert human or synthetic demonstrations [19].

2.2. Reinforcement Learning Algorithms

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

2.2.1. Q-Learning

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

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

Algorithm 1 Q-learning

function Q-LEARNING($s_{initial}, s_{terminal}, \alpha, d$)
  initialize $Q[S, A]$
  $s \leftarrow s_{initial}$
  while $s! = s_{terminal}$ **do**
    select a
    $r = R(s, a)$
    $s^{'} = T(s, a)$
    $Q[s, a] \leftarrow Q[s, a] + \alpha(r + d \cdot \max_{a'} Q[s^{'}, a^{'}] - Q[s, a])$
    $s \leftarrow s^{'}$

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

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

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

where:

- α - represent the learning rate. It influences at what extent does the new acquired information influence the old information. The learning rate take values between 0 and 1, the inferior extremity meaning that the agent will not learn anything, while the superior extremity determines the agent to learn only the most recent information. In deterministic environments, usually α takes values closer or equal to 1, while in worlds with stochastic transition dynamics, lower values are preferred.

- d - is the discount factor and it determines how much does a future reward influence the present one. As with the learning rate parameter, the discount factor takes values between 0 and 1, the inferior extremity making the agent not to consider future rewards at all, while values closer to the superior extremity will determine the agent to aim for the long-term high reward.

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

State-Action-Reward-State-Action is another reinforcement learning method in which the agent learn an optimal state-action policy using an on-policy strategy. Q-learning uses a off-policy strategy learning the optimal policy with disregard to the actual exploration that is being carried out. Sometimes the actions of an agent can generate large negative rewards, thus the strategy to update the value of the policy according to the exploration path it took can become and improvement. The latter strategy refers to off-policy learning, and SARAS is an algorithm that learns in this way. Algorithm 2 describes the computational steps of the SARSA method.

Algorithm 2 SARSA

function SARSA($s_{initial}, s_{terminal}, \alpha, d$)
    initialize $Q[S, A]$
    $s \leftarrow s_{initial}$
    select a
    while $s! = s_{terminal}$ **do**
        $r = R(s, a)$
        $s^{'} = T(s, a)$
        select $a^{'}$
        $Q[s, a] \leftarrow Q[s, a] + \alpha(r + d \cdot Q[s^{'}, a^{'}] - Q[s, a])$
        $s \leftarrow s^{'}$
        $a \leftarrow a^{'}$

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

2.2.3. Monte-Carlo Technique

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

3. Reinforcement Learning Model for Scheduling

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

3.1. Scheduling Problem Definition

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

- $nrpe_i$ the number of processing elements of node N_i, $nrpe_i \in \mathbb{N}^*$;
- $mips_i$ the computational power of node N_i, measured in million instructions per second, $mips_i \in \mathbb{N}^*$;
- $storage_i$ the storage capacity of node N_i, measured in bytes, $storage_i \in \mathbb{N}^*$;
- bw_i the communication bandwidth of node N_i, measured in Megabits per second, $bw_i \in \mathbb{N}^*$.

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

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

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

A task assignment schedule P is described as a tuple of $< Pt, Pv >$, where Pt contains n subsets of tasks and Pv contains n subsets of edges such that:
- $\forall t_k \in Pt_i$, $t_k \notin Pt_j$, with $i \neq j$ and $1 \le i, j \le n$ and
- $\forall v(t_k, t_p) \in Pv_i$, $t_p \notin Pt_j$, with $i \neq j$ and $1 \le i, j \le n$.

An example of a DAG in which the tasks have been assigned to the nodes is presented in Figure 1. The labels $t_1 \dots t_{10}$ represent the task and the nodes of the DAG, while the edges show the dependencies between the tasks. After the scheduling process, as it can be observed from the picture, the tasks found in the red rectangle will be executed by $machine_1$, the tasks from the blue rectangle will be executed by $machine_2$, and the tasks from the red rectangle will be executed by the $machine_3$. 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:

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

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

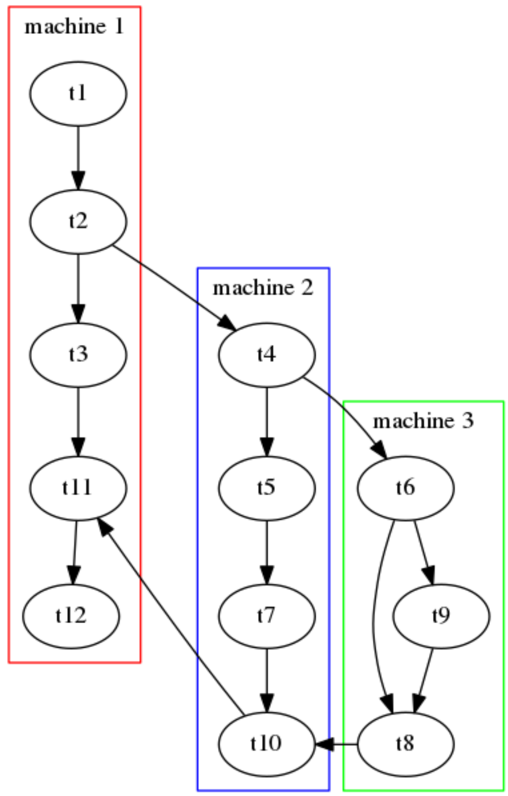

Figure 1: Task DAG example.

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

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

The most basic scheduling problem definition has a set of nodes or machines, that have their characteristics similar to the ones found in modern grid systems, and a task DAG that needs to be scheduled such that the entire dag achieves optimal execution time. For simplicity, it is firstly presumed that all the tasks from the DAG have similar execution times and resource requirements, such that if all of those tasks would execute on a single machine sequentially, the execution time of each task would be approximately the same. This means that there is no variation in task execution caused by the internal structure of the tasks themselves. An optimal total execution time is going to be determined by the correct disposition of the tasks onto the nodes such that the hardware infrastructure and the DAG layout are properly exploited.

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

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

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

The states S of the world are defined as follows:

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

where:

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

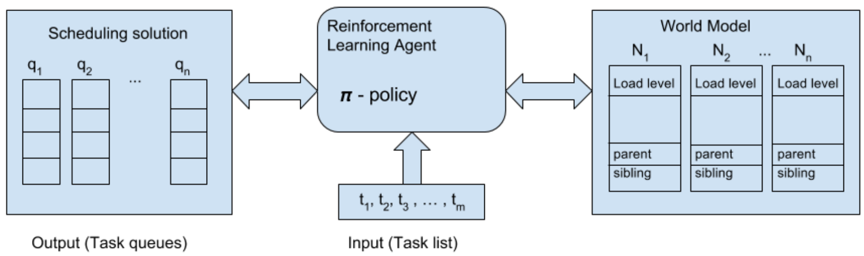

Figure 2: Conceptual model with first layer of complexity.

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

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

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

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

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

3.3. Second Layer of Complexity: dynamic cluster status

The first layer of complexity presumed that all the task have the same internal structure, and that there is only one job running at one time. In reality, a modern cluster systems has many jobs running and being scheduled, the true load of each machine influencing the quality of the policy learned from the world of the previous section. The advantage of the proposed scheduling solution is that it allows the extension of the world model without submitting heavy changes to the whole algorithm. As seen in Figure 3 each node should add a new attribute through which it can inform about it status. Every time the distributed system wants to use the agents policy to schedule, it should update the status fields of the world and inform the agent that he can start scheduling.

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

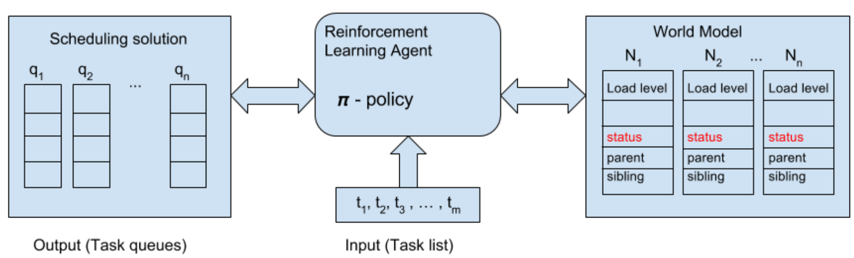

Figure 3: Conceptual model with second layer of complexity.

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

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

A task attribute added to the world model would increase the precision of the agents policy and would allow to schedule complex DAGs with variate tasks. But in order to better determine the type of a task, another component should be added to the main concept, a component that classifies incoming tasks and compresses their characteristics, to simplify the world model and reduce the number of states. Figure 4 is a visual representation of the concept with the task classifier extension. With the addition of the last component the model is complete, and should be able to learn scheduling policies that decrease the total execution time of jobs and improve overall performance of distributed systems like clusters and grid.

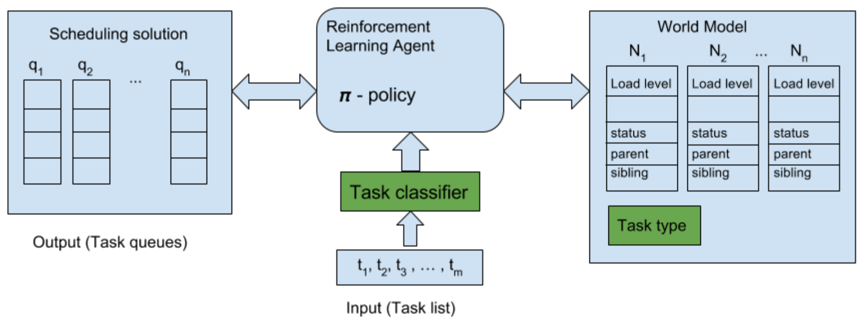

Figure 4: Conceptual model with third layer of complexity.

4. Machine Learning Box Architecture

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

Figure 5 shows a simple architectural model of the Machine Learning Box of the basic structural and functional components. The application is designed to support parallelism and easily scale into a big system that could handle many clients. Each major component runs, in the demo application, in a separate thread, this separation giving the system the capability to eventually move each component on dedicated servers. The domain database can be moved to a system that permits data replication and has integrated consistency and backup protocols, so that the already learned policies and registered world models are never lost and are always accessible. The possibility to run simultaneously many reinforcement learning agents can speed up the learning process and can lead to faster ways to find optimal policies on large world models.

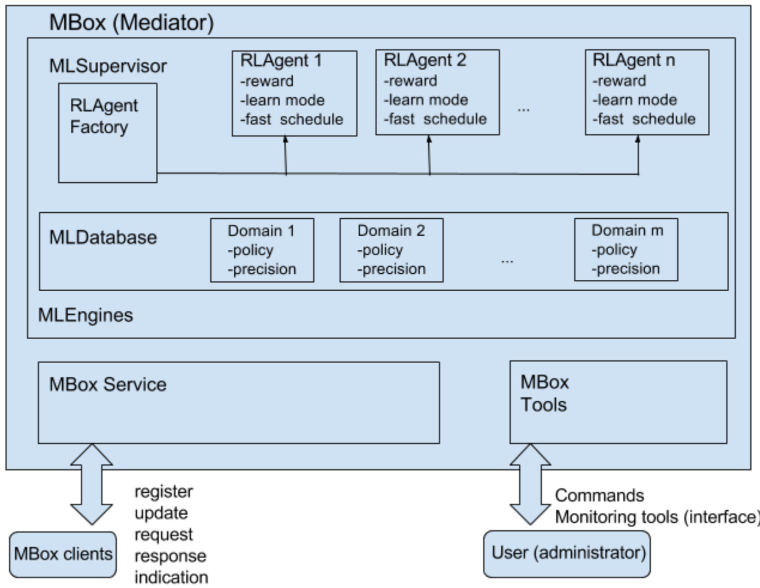

Figure 5: Visual representation of the MBox architecture, encapsulating the main components of the platform, their relationship and the interfaces provided.

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

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

The clients that want to use the scheduling service must firstly make a request, to the administrators or owners of the machine learning box instance, for the allocation of a number of MBox schedulers. Then, by using a library of common classes and interfaces, they must implement and integrate a custom unit that uses the remote allocated MBox scheduler to register new domains, update existing ones, make scheduling requests for jobs and indicate the execution time of the scheduling solution. If the remote scheduler fails, then it is the duty of the local unit to deal with it, providing alternatives. As future work, the integration of learning from logs could prove to be useful, as the learning agents could learn from past solutions without having the need to physically test the scheduling solution.

In the following subsections all the major components of the machine learning box application will be presented, offering details about the implementation and the design decisions.

4.1. MBox Service

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

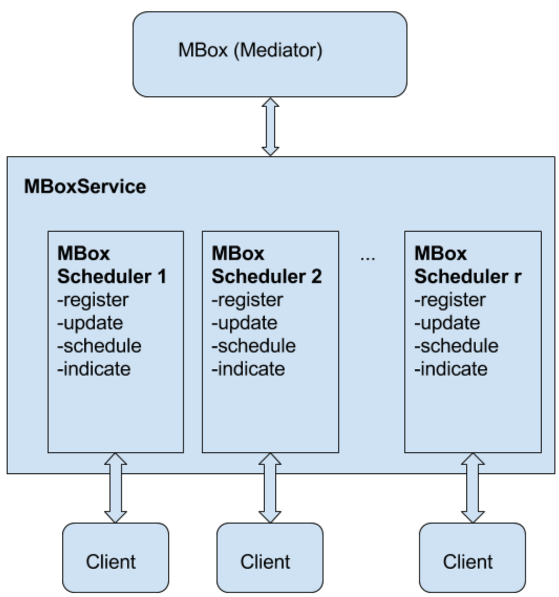

Figure 6: Visual representation of the MBox Service, of its internal structure and of its interaction with the clients.

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

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

4.2. MBox Machine Learning Engine

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

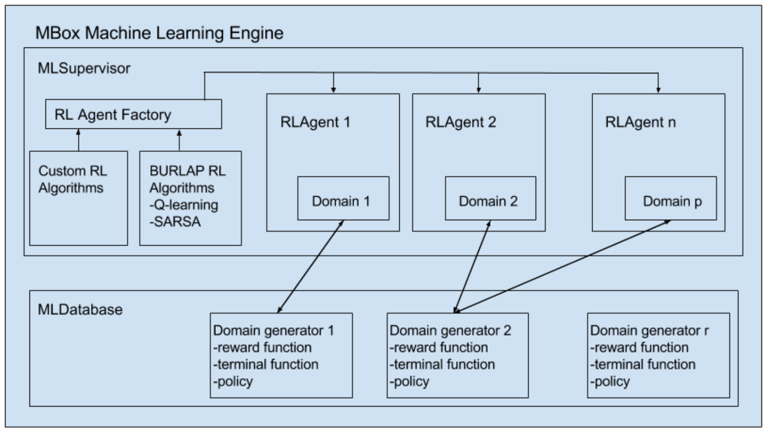

Figure 7: Visual representation of the MBox Machine Learning Engine, encapsulating the communication between the domanis from the MLDatabase and the learning agents, monitored by the MLSupervisor.

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

BURLAP is a java code library for developing scheduling and learning algorithms. It offers a highly flexible framework for defining states and actions, supporting discrete, continuous and relational domains. There are several scheduling and learning algorithms implemented in the library and ready to use, and it allows the extension and creation of new ones. It also contains a suite of analysis tools for the visualization of the domains and for the performance of the running agents.

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

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

4.3. MBox Scheduling with WorkflowSim

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

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

5. Results

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

5.1. Theoretical Limit

Considering the first layer of complexity, the model had the following parameters:

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

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

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:

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

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

5.2. Experimental results

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

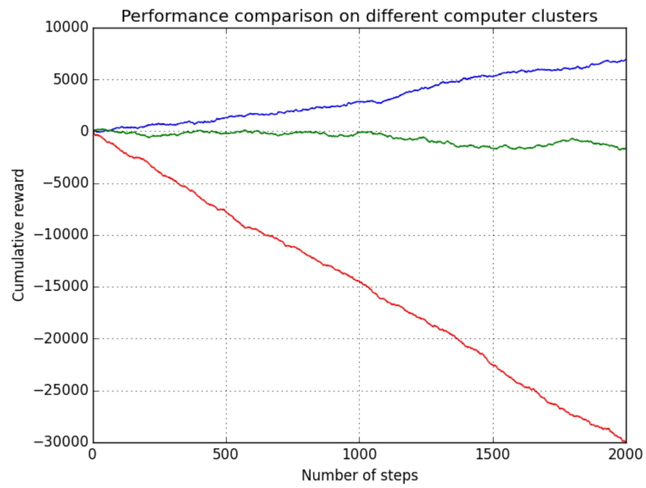

Figure 8: Cumulative reward performance evaluation of Q-learning algorithm. 2 nodes - blue, 4 nodes - green, 8 nodes - red.

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

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

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

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

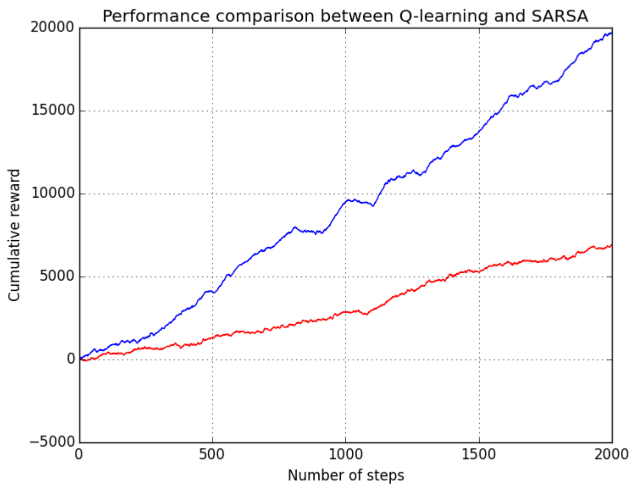

Figure 9: Q-learning (blue) vs SARSA (red).

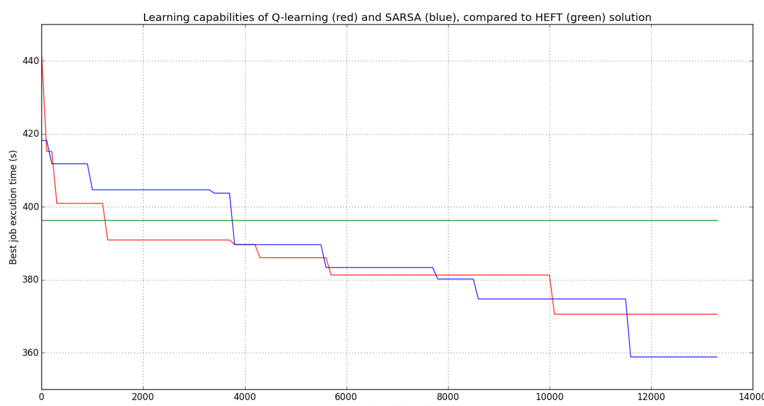

Figure 10: Learning capabilities of Q-learning (red) and SARSA (blue), compared to HEFT (green) solution.

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

The conclusion is that the proposed model has combined the characteristics of all the nodes or machines of the cluster, resulting in a huge world of states that cannot be properly explored by a reinforcement learning agent. Machine Learning algorithms have a though time dealing with such problems, and in order for those techniques to work they would need auxiliary help from other heuristics and strategies. Given the fact that the theoretical model has shown the limitations of the proposed algorithm, further experimentation would have been redundant. Regarding the comparison of other scheduling algorithms, most of other scheduling algorithms do not need iterations to arrive at a more mature state. The performance of the reinforcement learning method will be lower at the beginning, but it will surpass the classic algorithms after an enough number of iterations.

6. Conclusion

As computer clusters and distributed systems become more and more popular, the need to improve the performance of such systems becomes a challenge, that if properly mastered could accelerate the evolution of science or could reset the positions of industry giants. It is clear that the schedulers from such systems have a certain impact on the efficiency of parallel systems. Machine learning and artificial intelligence are gaining ground, intelligent solutions, that learn from the past and adapt, will become the norm when dealing with complex problems. Task schedulers in distributed systems would benefit greatly from intelligent agents, learning from past mistakes, exploring and finding new solutions that no human might have though before.

In this paper, a platform, offering scheduling solutions as a service based on machine learning agents, was described, and a reinforcement learning world model for scheduling was proposed. The platform, know as the Machine Learning Box, allows further development of scheduling algorithms and an easy integration process. The application model can easily be mapped on parallel systems, in order to scale and increase the overall efficiency. The learning model proved to have its limitations, due to the complex nature of a distributed system and the proposed codification as a world of states. While this codification works on smaller systems, the more nodes were added to the system the larger the world got, leaving the reinforcement learning agent incapable of properly learning an optimal policy.

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

Acknowledgment

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

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

We would like to thank the reviewers for their time and expertise, constructive comments and valuable insight.

References

[1] R. Kumar, N. Gupta, S. Charu, S. K. Jangir, Architectural paradigms of big data, in: National Conference on Innovation in Wireless Communication and Networking Technology–2014, Association with the Institution of Engineers (INDIA), 2014.

[2] T. White, Hadoop: The Definitive Guide, 3rd Edition, O'Reilly - O'Reilly Media, 2012.

[3] J. Dean, S. Ghemawat, Mapreduce: simplified data processing on large clusters, Communications of the ACM 51 (1) (2008) 107–113.

[4] A. C. Murthy, V. K. Vavilapalli, D. Eadline, J. Niemiec, J. Markham, Apache Hadoop YARN: Moving Beyond MapReduce and Batch Processing with Apache Hadoop 2, Pearson Education, 2013.

[5] J. D. Ullman, Np-complete scheduling problems, Journal of Computer and System sciences 10 (3) (1975) 384–393.

[6] P. Liu, K.-K. R. Choo, L. Wang, F. Huang, Svm or deep learning? a comparative study on remote sensing image classification, Soft Computing (2016) 1–13.

[7] L. Wang, J. Zhang, P. Liu, K.-K. R. Choo, F. Huang, Spectral–spatial multi-feature-based deep learning for hyperspectral remote sensing image classification, Soft Computing (2016) 1–9.

[8] Y. Hu, J. Yan, K.-K. R. Choo, Pedal: a dynamic analysis tool for efficient concurrency bug reproduction in big data environment, Cluster Computing 19 (1) (2016) 153–166.

[9] Z. Xu, H. Zhang, V. Sugumaran, K.-K. R. Choo, L. Mei, Y. Zhu, Participatory sensing-based semantic and spatial analysis of urban emergency events using mobile social media, EURASIP Journal on Wireless Communications and Networking 2016 (1) (2016) 44.

[10] J. MacGlashan, The brown-umbc reinforcement learning and planning (burlap) java code library (http://burlap.cs.brown.edu/) (2016).

[11] D. E. Hershkowitz, J. MacGlashan, S. Tellex, Learning propositional functions for planning and reinforcement learning, in: 2015 AAAI Fall Symposium Series, 2015.

[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.

[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.

[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.

[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.

[16] W. Zhang, T. G. Dietterich, A reinforcement learning approach to job-shop scheduling, in: IJCAI, Vol. 95, Citeseer, 1995, pp. 1114–1120.

[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.

[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.

[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.

[20] D. L. Poole, A. K. Mackworth, Artificial Intelligence: foundations of computational agents, Cambridge University Press, 2010.

[21] S. Russle, P. Norvig, Artificial Intelligence A Modern Approach, Third Edition, 2009.