sustainability

Article

Smart DAG Tasks Scheduling between Trusted and Untrusted Entities Using the MCTS Method

Yuxia Cheng [1], Zhiwei Wu [1], Kui Liu [1], Qing Wu [1,*] and Yu Wang [2,*]

1 School of Computer Science and Technology, Hangzhou Dianzi University, 1158 Baiyang No. 2 Street, Hangzhou 310018, China; yxcheng@hdu.edu.cn (Y.C.); wzwtime@163.com (Z.W.); m17730390391@163.com (K.L.)
2 School of Computer Science, Guangzhou University, 230 Wai Huan Xi Road, Guangzhou Higher Education Mega Center, Guangzhou 510006, China
* Correspondence: wuqing@hdu.edu.cn (Q.W.); yuwang@gzhu.edu.cn (Y.W.)

Received: 31 December 2018; Accepted: 20 March 2019; Published: 27 March 2019

Abstract: Task scheduling is critical for improving system performance in the distributed heterogeneous computing environment. The Directed Acyclic Graph (DAG) tasks scheduling problem is NP-complete and it is hard to find an optimal schedule. Due to its key importance, the DAG tasks scheduling problem has been extensively studied in the literature. However, many previously proposed traditional heuristic algorithms are usually based on greedy methods and also lack the consideration of scheduling tasks between trusted and untrusted entities, which makes the problem more complicated, but there still exists a large optimization space to be explored. In this paper, we propose a trust-aware adaptive DAG tasks scheduling algorithm using the reinforcement learning and Monte Carlo Tree Search (MCTS) methods. The scheduling problem is defined using the reinforcement learning model. Efficient scheduling state space, action space and reward function are designed to train the policy gradient-based REINFORCE agent. The MCTS method is proposed to determine actual scheduling policies when DAG tasks are simultaneously executed in trusted and untrusted entities. Leveraging the algorithm's capability of exploring long term reward, the proposed algorithm could achieve good scheduling policies while guaranteeing trusted tasks scheduled within trusted entities. Experimental results showed the effectiveness of the proposed algorithm compared with the classic HEFT/CPOP algorithms.

Keywords: DAG scheduling; trusted entities; heterogeneous; MCTS

1. Introduction

Modern organizations are increasingly concerned with their trust management. As the cloud computing paradigm prevails, more and more data security and trust issues are arising due to the public cloud infrastructures being under control of the providers but not the organizations themselves [1,2]. Therefore, a practical solution for addressing these trust issues is to deploy security sensitive tasks in the trusted entities (IT infrastructures privately managed within organizations) and those security non-sensitive tasks in the untrusted entities (IT infrastructures such as public cloud). Scheduling security sensitive and non-sensitive tasks between the trusted and untrusted entities is one of the research challenges in the trust management. Particularly, when these tasks have sequential and parallel connections, the scheduling problem becomes further complicated in distributed heterogeneous computing systems.

In distributed heterogeneous computing systems, a variety of computing resources are interconnected with high speed networks to support compute-intensive parallel and distributed applications [3,4]. In these systems, efficient task scheduling is critical for improving system performance. Especially, as the modern hardware technology evolves rapidly, diverse sets of computing hardware unit, such as CPU, GPU, FPGA, TPU, and other accelerators, constitute a more and more complex heterogeneous computing system. Modern high performance computing applications typically use the Directed Acyclic Graph (DAG) based compute model to represent an application's parallel compute tasks and their dependencies. How to schedule DAG tasks in the distributed heterogeneous computing system is an open research question.

Most parallel applications, including high performance computing (HPC) applications, machine learning applications [5] etc., use the DAG tasks model in which nodes represent application tasks and edges represent inter-task data dependencies. Each node holds the computation cost of the task and each edge holds inter-task communication cost. To improve system efficiency, the goal of DAG tasks scheduling is to map tasks onto heterogeneous computing units and determine their execution order so that the tasks' dependencies are satisfied and the application's overall completion time is minimized.

Previous research [6] has shown that the general tasks scheduling problem is NP-complete and it is hard to find an optimal schedule. Researchers [7] theoretically proved that the DAG tasks scheduling problem is also NP-complete and is more complex in practical scheduling system. Due to its key importance, the DAG tasks scheduling problem has been extensively studied in the literature.

Many traditional heuristic algorithms have been proposed, such as list scheduling algorithms [8], genetic and evolutionary based random search algorithms [9], task duplication-based algorithms [10], etc. These algorithms are mostly heuristic in restricted application scenarios, and lack generality in the adaptation of various heterogeneous hardware and rapid changing application demand [11]. The machine learning based method is a reasonable way of adapting to the ever-changing hardware and software environment by learning from past scheduling policies.

However, previous research lacks the consideration of scheduling tasks between trusted and untrusted entities. Restricting trusted tasks within trusted entities increases the scheduling complexity, and most of the previously proposed scheduling algorithms cannot be easily adapted to this scenario. Therefore, it is important to study the practical way of integrating trust management into the DAG tasks scheduling algorithm in distributed heterogeneous computing systems.

Reinforcement learning [12] could be used for learning smart scheduling policies from past experiences. Recent research has proposed task scheduling and device placement algorithms based on reinforcement learning. However, existing approaches either greatly simplify the scheduling model [13,14] that are unpractical or need a great amount of computing resources [11,15] to train the scheduling policies that are inefficient for most application scenarios.

Monte Carlo Tree Search (MCTS) [16] could be used for searching tasks scheduling policies that meet the requirement of trust management. MCTS combines the precision of tree search with the generality of random sampling. MCTS is an any-time search method that is efficient in terms of computation resource usage. To the best of our knowledge, MCTS methods are mostly developed in game domains. A few studies have been published in addressing the scheduling problems, but the trust management in the scheduling is not well studied yet.

In this paper, we propose a trust-aware adaptive DAG Tasks Scheduling (tADTS) algorithm using deep reinforcement learning and Monte Carlo tree search. The scheduling problems are properly defined with the reinforcement learning process. Efficient scheduling state space, action space and reward function are designed to train the policy gradient-based REINFORCE agent.The MCTS method is proposed to determine actual scheduling policies when DAG tasks are simultaneously executed in trusted and untrusted entities. Leveraging the algorithm's capability of exploring long term reward, we could achieve better scheduling efficiency. Experimental results showed the effectiveness of the proposed tADTS algorithm compared with the classic HEFT/CPOP algorithms. The main contributions of this paper include:

(1) We propose an accurate and practical DAG tasks scheduling model based on reinforcement learning. To the best of our knowledge, this is the first work to address the static DAG tasks scheduling problem with the reinforcement learning process. Previous research has proposed a similar model [14],

but oversimplifies the problem with assumptions of restricted machine performance, cluster status, and task classification.

(2) We designed efficient representations of state space, action space and reward function. Too large state space and action space without careful design will make the algorithm training time-consuming or even unable to converge. The reward function design also plays an important role in the reinforcement learning process.

(3) We proposed a trust-aware single-player MCTS (tspMCTS) method integrated with the DAG tasks scheduling algorithm. The proposed tspMCTS method is flexible and scalable to schedule tasks among multiple trusted and untrusted entities. The additional trust management does not increase the time complexity of tspMCTS.

The rest of this paper is organised as follows: Section 2 describes the related work. Section 3 presents the Adaptive DAG Tasks Scheduling (ADTS) algorithm design. Section 4 shows the experimental results. Finally, Section 5 concludes this paper and discusses future work.

2. Related Work

Previous research proposed different scheduling algorithms based on the characteristics of tasks computation and communication, their dependency relationships, as well as the heterogeneity of hardware. Depending on the techniques, the scheduling algorithms can be classified as traditional heuristic based algorithms and machine learning based algorithms.

DAG tasks scheduling in the distributed heterogeneous computing environment has been extensively studied. The DAG tasks scheduling algorithms could be typically divided into static and dynamic scheduling. In static scheduling [17], the tasks' runtime and data dependencies are known in advance, and the scheduling policy is determined off-line. In dynamic scheduling [18], the tasks are assigned to processors at their arrival time and the schedule policy is determined on-line. Most DAG tasks scheduling algorithms belong to static scheduling.

Traditional static DAG tasks scheduling algorithms mainly include: (1) List scheduling algorithms [8,19]. The key idea of list scheduling algorithm is to order the scheduling tasks priority list and select a proper processor for each task. (2) Clustering based algorithms [20,21]. The key idea of clustering based algorithm is to map DAG tasks to a number of clusters. Tasks assigned to the same cluster will be executed on the same processor. (3) Genetic and evolutionary based random search algorithms [9,22]. The key idea of this group of algorithms is to use random policies to guide the scheduler through the problem space. The algorithms combine the results gained from previous search with some randomizing features to generate new results. (4) Task duplication based algorithms [10,23]. The key idea of these algorithms is to duplicate some of the tasks in different processors, which reduces the communication overhead in data-intensive applications.

These DAG tasks scheduling algorithms are heuristic and mainly designed by experts, which are carefully adapted to different application scenarios. However, with the rapid development of heterogeneous hardware and ever changing applications, traditional DAG tasks scheduling algorithms cannot fully exploit system performance [11,15]. To design adaptive algorithms, researchers proposed machine learning based algorithms. In this paper, we refer to the traditional scheduling algorithm (no machine learning techniques are used) as heuristic algorithm. However, strictly speaking, the proposed algorithm also belongs to the heuristic algorithm. To distinguish between human expert experience-based algorithms and machine learning based algorithms, we denote the former as traditional heuristic algorithms.

Zhang et al. [12] first proposed using classic reinforcement learning to address job-shop scheduling problem. However, the job-shop scheduling is different from the DAG tasks scheduling problem, where DAG tasks have more complex dependencies and data communication cost. Mao et al. [13] proposed using deep reinforcement learning to solve a simplified task s scheduling problem. The policy gradient based REINFORCE algorithm is used to train a fully connected policy network with 20 neurons. However, the scheduling problem is over simplified that treats the compute cluster as a single collection of resources, which is unpractical in real systems. Orhean et al. [14] proposed reinforcement learning based scheduling approach for heterogeneous distributed systems. This approach has additional assumptions such as machine performance, cluster status, and tasks types, which can not be easily applied in real DAG tasks scheduling. Mirhoseini et al. [11,15] proposed using reinforcement learning method to optimize device placement for TensorFlow computational graphs. These methods require a large amount of hardware to train policy network. The state space and action space definitions cannot accurately reveal the DAG and hardware topologies, which results in many invalid placement trials. Though previous research has these shortcomings, the reinforcement learning based approach has demonstrated its benefits in terms of adaptiveness and better scheduling quality.

The Monte Carlo Tree Search (MCTS) method [16] combines the precision of tree search with the generality of random sampling. MCTS received significant interest due to its success in difficult games like computer Go [24]. Single-Player MCTS [25] was first proposed in the SameGame. Y. Björnsson and H. Finnsson [26] investigated the application of standard UCT [27,28] to single-player games. The MCTS method was also developed in the scheduling [29,30] and planing [31,32] applications. S. Matsumoto et al. [29] proposed a single-player MCTS method to address a re-entrant scheduling problem that managed the printing process of the auto-mobile parts supplier. A. McGovern et al. [30] proposed a basic block instruction scheduler with reinforcement learning and rollouts. However, how to leverage MCTS method to design tasks scheduling with trust management still needs further investigation.

Previous research demonstrated that DAG tasks scheduling belongs to the class of strong NP-hard problems [33]. Hence, it is impossible to construct not only a pseudo-polynomial time optimization scheduling algorithm but also a fully polynomial time approximation scheme (PTAS) unless P=NP [34,35]. To the best of our knowledge, H. Kasahara et al. [36] designed the constant-factor (1+eps) approximation algorithm for multiprocessor scheduling. However, the constant-factor approximation algorithm assumed homogeneous processors and did not consider the communication costs between tasks, which is far beyond reality in modern computer system. We plan to design a more realistic constant-factor approximation algorithm based on the branch and bound algorithm in our future work.

Unlike previous research, we proposed a new reinforcement learning based trust-aware scheduling algorithm with MCTS that defines more accurate scheduling model using DAG graph structures and efficient state/action space representations. The similarities between reinforcement learning and classic machine learning algorithms is that they both need large volumes of training data to train a model. However, unlike supervised learning that requires pre-labelled training data, the training data of reinforcement learning is obtained via online interaction with the environment and the reward function determines the label signal. The goal of our proposition is to maximize long term reward while the classic machine learning method usually minimizes the prediction error. The proposed tADTS algorithm can be used in practice in the same way as traditional static DAG tasks scheduling algorithms. This paper is an extended version of previously published conference paper in the 18th International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP 2018) [37].

3. Trust-Aware Adaptive DAG Tasks Scheduling Algorithm Design

In this section, we present the trust-aware Adaptive DAG Tasks Scheduling (tADTS) algorithm design. First, the DAG tasks scheduling problem is defined. Second, we formulate the reinforcement learning process and present the design of three key elements of RL, the state space, the action space, and the reward function. Then, we proposed the trust-aware single-player MCTS method. Finally, we show the policy gradient based training algorithm and the policy network architecture design.

3.1. Problem Definition

We leverage the definition of DAG tasks graph in distributed heterogeneous system [8]. The scheduling model consists of three parts:

(i) An application represented by a DAG tasks graph, $G = (V, E)$, where V is a set of v tasks in the application, and E is the set of e edges between tasks.

- edge $(i, j) \in E$ denotes the precedence constraint such that task n_j must wait until task n_i finishes its execution.
- $data_{i,j}$ denotes the amount of data to be sent from task n_i to task n_j.
- Each task n_i has a flag that denotes whether this task is a security sensitive or non-sensitive task.

Figure 1 shows an example of DAG tasks graph. The bold task nodes (tasks 1, 2, 9, 10) represent the security sensitive tasks that must be executed inside trusted entities.

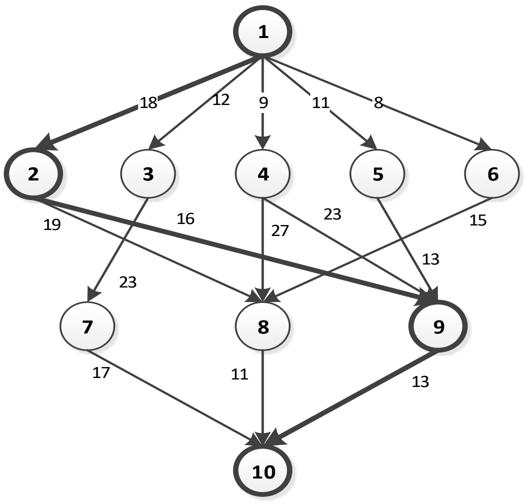

Figure 1. An example of DAG tasks graph.

(ii) A distributed heterogeneous computing system, which consists of a set Q of q heterogeneous processors with a fully connected topology.

- W is a $v \times q$ computation cost matrix, and $w_{i,j}$ denotes the execution time of task n_i on processor p_j.
- B is a $q \times q$ matrix of the data communication bandwidth between processors, and $B_{m,n}$ denotes the communication bandwidth between processor p_m and processor p_n.
- L is a q-dimensional vector that denotes the communication initialization costs of processors, and L_m denotes the initialization costs of processor p_m.
- $c_{i,j} = L_m + \frac{data_{i,j}}{B_{m,n}}$ denotes the communication cost of edge (i, j), which is for the cost of sending data from task n_i (running on p_m) to task n_j (running on p_n).
- Each processor has a flag that denotes whether this processor resides within a trusted entity or a non-trusted entity.

(iii) Performance criterion for scheduling. Before presenting the final scheduling objective function, we first define the EST (Earliest Start Time), EFT (Earliest Finish Time), AST (Actual Start Time), and AFT (Actual Finish Time) attributes.

- $EST(n_i, p_j) = max\left\{avail\,[j]\,, \max_{n_m \in pred(n_i)} (AFT(n_m) + c_{m,i})\right\}$ denotes the earliest execution start time of task n_i on processor p_j, where $avail\,[j]$ is the earliest time at which processor p_j is available for execution, and $pred(n_i)$ is the set of immediate predecessor tasks of task ni. The inner max block returns the time when all data required by task n_i has arrived at processor p_j.

- $EFT(n_i, p_j) = w_{i,j} + EST(n_i, p_j)$ denotes the earliest finish time of task n_i on processor p_j.
- $AST(n_m)$ denotes the actual start time of task n_m when it is scheduled on a processor p_j to execute.
- $AFT(n_m)$ denotes the actual finish time of task n_m after it is scheduled on a processor p_j and finishes execution.

The EST and EFT values can be computed recursively from the entry task n_{entry}, where $EST(n_{entry}, p_j) = 0$. After all tasks in a graph are finished execution, the AFT of the exit task n_{exit} is named the schedule length (also named *makespan*), which is defined as:

$$makespan = max\,\{AFT(n_{exit})\} \tag{1}$$

The objective function of the DAG tasks scheduling is to determine the assignment policies of an application's tasks to heterogeneous processors so that the schedule length is minimized.

3.2. Reinforcement Learning Formulation

Once the scheduling problem is defined, we propose to address the scheduling problem with the reinforcement learning method [38]. Figure 2 shows a brief diagram of the reinforcement learning based scheduling model. At time t, the scheduler observes the environment and receives an observation O_t. Depending on O_t, the scheduler determines an scheduling action A_t. After A_t is executed, the scheduler receives a reward R_t. The scheduler continues this process $(..., O_t, A_t, R_t, O_{t+1}, A_{t+1}, R_{t+1}, ...)$ until the end of schedule (task n_{exit} is scheduled). The observation O_t typically could be denoted as a state S_t.

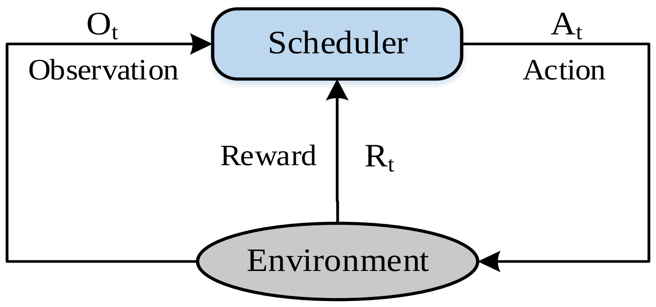

Figure 2. Reinforcement Learning Based Scheduling Model.

We use the policy gradient method to optimize the scheduling actions so that the expected total reward could be maximized. The optimization objective function is defined as:

$$J(\theta) = E_{A \sim \pi(A|G;\theta)}[R(A)|G] \tag{2}$$

where θ denotes parameters of the policy network; A denotes the scheduling policy (a sequence of actions); $\pi(A|G;\theta)$ denotes the probabilities of scheduling policy A produced by policy network (defined by parameters θ) given the DAG tasks graph and heterogeneous system G; $R(A)$ denotes the total reward under the scheduling policy A; $J(\theta)$ denotes the expected reward of the scheduling policy A.

In the reinforcement learning, the design of the state space and action space representations as well as the design of reward function are important for the algorithm's overall performance. We describe the three key elements as follows.

State space. The state space of the scheduling problem could be very large, which would include the state of the DAG tasks graph and the state of the distributed heterogeneous computing system. We design an efficient and compact representation of the state space, which is defined as:

$$S_t = [n, EST(n_i, p_1), ..., EST(n_i, p_q), w_{i,1}, ..., w_{i,q}] \tag{3}$$

where S_t is the state (observation) at time t. n denotes the number of tasks that are not scheduled so far (listed in a waiting queue). $EST(n_i, p_j)$ is the earliest start time of task n_i on processor p_j, task n_i is the current task to be scheduled. We use the task's EST on all processors (from processor 1 to processor q) to represent the state of the current system. The EST as described in Section 3.1 contains both the information of processor's load and the communication cost. Based on the Markov property, the current task's ESTs can be used as the state to summarize the previous situations before task n_i. $w_{i,j}$ is the computation cost of task n_i on processor p_j. To preserve the tasks precedence relationship in DAG, we adopt the upward rank [8] to list tasks in the waiting queue so that tasks with higher rank values are scheduled before tasks with lower rank value. Note that other task list methods are possible provided that the task precedence constraints are satisfied.

Action space. Once the state space is defined, the action space of the scheduling problem is straightforward. The action space is defined as:

$$A_t = \{p_i | p_1, ..., p_q\} \tag{4}$$

where A_t is the scheduling action at time t. p_i denotes that the scheduler assigns processor p_i for the current task in the head of the waiting queue. The possible action at each time step is to assign one of the processors (range from processor p_1 to processor p_q) for the task to be scheduled.

Reward function. The design of reward function could impact the scheduling policies, which is critical for the policy training. The reward at each time step should help guide the actual scheduling actions, and the accumulative long term reward should also reflect the final scheduling objective. Based on the above understanding, the reward function is defined as:

$$R_t = max\{EST(n_{i+1}, p_j)|_{j=1..q}\} - max\{EST(n_i, p_j)|_{j=1..q}\} \tag{5}$$

where R_t is the immediate reward at time t. Task n_{i+1} is the task in the head of waiting queue after task n_i is scheduled with action A_t at time t. The reward R_t is obtained by calculating the increment of current schedule length after task n_i is scheduled. The current schedule length is represent by $max\{EST(n_i, p_j)|_{j=1..q}\}$.

3.3. Trust-Aware Single-Player MCTS Method

Monte Carlo Tree Search (MCTS) typically has four basic steps as shown in Figure 3. The DAG tasks scheduling process can be mapped as a single-player MCTS process. We describe the four steps in detail and how each step is designed to address the trust-aware DAG tasks scheduling problem.

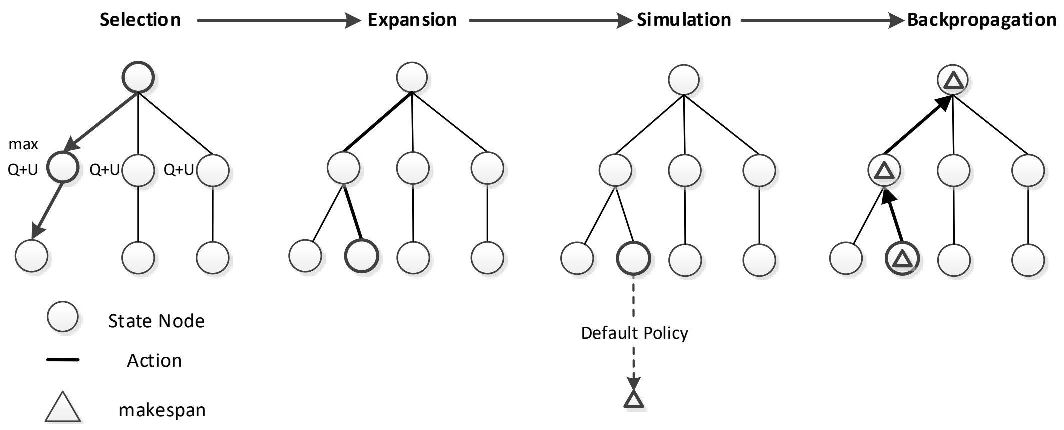

Figure 3. Basic steps of Monte Carlo Tree Search.

In the MCTS tree structure, the root node represents the beginning state of the DAG tasks scheduling, which is the initial state of the first task to be scheduled. The subsequent nodes represent the possible states reached after MCTS selects possible actions. The edges in the MCTS tree represent the scheduling actions, which are many possible combinations of actions mapping ready tasks to certain processors. The four steps of progressively building a single-player MCTS tree are described below.

Four steps are applied for each search iteration:

(1) Selection: From the root node, a child selection policy is recursively applied to descend through the MCTS tree until an expandable node is reached. An expandable node denotes a non-terminal state and has unvisited children. The child selection policy is based on the UCT algorithm [39], which selects the maximum value of UCTs among its child nodes.

$$UCT = \overline{Q_j} + 2C_p\sqrt{\frac{2\ln n}{n_j}} \tag{6}$$

Equation (6) shows the UCT calculation that addresses the exploration-exploitation dilemma in MTCS. The first term $\overline{Q_j}$ in Equation (6) represents exploitation, which is the mean makespan value of the simulated scheduling policies that visited $node_j$ so far. The second term $2C_p\sqrt{\frac{2\ln n}{n_j}}$ represents exploration, where n is the number of times the parent node has been visited, n_j is the number of times child $node_j$ has been visited and C_p is the constant parameter that controls the exploration.

(2) Expansion: According to the available actions, child nodes are added to the expandable parent node. The available actions are determined online based on the ready tasks available after the expandable parent node is visited. The ready tasks are determined depending on the ordering relations in the DAG tasks graph. The number of available actions equals the number of ready tasks multiplied by the number of allowed processors. Due to the restriction that security-sensitive tasks must be scheduled onto trusted processors, the number of allowed processors are limited for each ready task.

(3) Simulation: Starting from the leaf node, a simulation is run based on the default policy to generate subsequent schedule actions. The default policy is the output of the policy network trained in the reinforcement learning. The training of policy network $\pi(a|s,\theta)$ is described in Section 3.4. In the simulation, security-sensitive tasks are strictly limited to the trusted entities.

(4) Backpropagation: After simulation finishes, the MCTS agent obtains simulation result (the makespan of DAG tasks). Then, the simulation result is backpropagated through previously visited nodes in the MCTS tree to update their statistics (average makespan $\overline{Q}$ and visit count n). The updated node statistics are used to inform future node selection in the MCTS tree.

The decision structure helps the selection of scheduling actions at each time step during training. The online scheduler only uses the trained network to output scheduling actions. Therefore, the algorithm is similarly efficient for big trees and small trees as the MCTS tree structure is used for online reference on the top layer of the tree. However, it costs a larger number of simulation times to construct a big tree rather than a small tree. In the training phase, big trees hold many more Monte-Carlo simulation trials than small trees that could provide more accurate scheduling action selection. The advantages of the MCTS tree structure are its efficient simulation and "any-time" property that could stop simulation at any-time depending on computing resource budget. The limitations of MCTS tree structure are that it is hard to set an optimal parameter C_p to balance the tradeoff between exploration and exploitation under limited computing resources.

3.4. Training Algorithm

We train an adaptive DAG tasks scheduling agent with the REINFORCE algorithm [40] and MCTS method. The training algorithm is based on the policy gradient methods with many Monte-Carlo trials. The algorithm input consists of a differentiable parameterization $\pi(a|s,\theta)$ and the training step size α. Initially, the policy parameters θ are a set to random numbers. During the training process, we generate

N number of episodes to train the policy network. Each episode represents a complete schedule of DAG tasks, which starts from the entry task state S_0, action A_0, and the corresponding reward R_1, to the end of the exit task state S_{T-1}, action A_{T-1}, and the final reward R_T. For each step of an episode, the algorithm calculates the long term reward G with an discounted factor γ. The policy parameter θ is updated in every step with $\nabla ln\pi(A_t|S_t,\theta)$, which equals the fractional vector $\frac{\nabla\pi(A_t|S_t,\theta)}{\pi(A_t|S_t,\theta)}$ named the *eligibility vector*. Previous research [41] has proved the policy gradient theory used for the function approximation in the reinforcement learning.

As shown in Algorithm 1, the generation of an episode is based on the MCTS tree selection and the default policy simulation. Inspired by AlphaZero [42], we combined the reinforcement learning with MCTS for the DAG tasks scheduling problem. Algorithm 1 incorporates lookahead search inside the training loop that results in rapid improvement and precise and stable learning. MCTS uses the policy network to guide its simulations, which is a powerful policy improvement operator. In turn, the generated simulation episode is used to train a better policy network. Then, the better policy network is iteratively used to make the MCTS search even stronger. The iteration terminates when the number of episodes reached a predefined threshold. Thanks to the efficient exploration and exploitation structure of MCTS, the algorithm could simulate a small number of Monte Carlo trials to construct asymmetric tree structure that guides the selection of scheduling actions. Therefore, the stop criterion of N is usually set to thousands to tens of thousands depending on the scale of the scheduling problem. The detailed settings are described in Section 4.1.

Algorithm 1 REINFORCE with MCTS: Monte-Carlo Policy-Gradient Control for π_*.

```
Input: A differentiable policy parameterization $\pi(a|s,\theta)$; Algorithm parameter: step size $\alpha > 0$;
1: Initialize random policy parameter $\theta \in R$;
2: Loop for N episodes:
3:     Generate an episode $S_0, A_0, R_1, ..., S_{T-1}, A_{T-1}, R_T$,
   following MCTS tree selection and the default policy $\pi(a|s,\theta)$ simulation;
4:     Loop for each step of the episode $t = 0, 1, ..., T-1$:
5:         $G \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$
6:         $\theta \leftarrow \theta + \alpha\gamma^t G \nabla ln\pi(A_t|S_t,\theta)$
```

The reward function outputs reward R_t at each time step t. R_t is an immediate reward that is obtained by calculating the increment of current schedule length after task n_i is scheduled. Therefore, R_t is dynamically generated following different scheduling actions (policies). In the training algorithm, R_t is used to accumulatively calculate the long term reward G with a Monte-Carlo trial. Then, G is used to update the neural network parameter theta with gradient ascent.

Compared with random search, the UCT algorithm applied in the MCTS is more efficient, which has theoretical guarantee [43] of the upper confidence bound to an expected logarithmic growth of regret uniformly over n without prior knowledge regarding the reward distributions.

4. Experiments

In this section, we evaluate the proposed tADTS algorithm comparing with the classic baseline algorithms. The DAG tasks graphs are generated using the graph generator [8] to represent the real world applications. First, we present the experiment settings and the performance evaluation metrics. Then, the comparative experimental results are described in the following subsection. Note that the proposed training algorithm is under implementation, the combination of reinforcement learning and MCTS is not tested in this experiment. The following experiment shows the individual ADTS algorithm performance result.

4.1. Methodology

The experiment hardware platform is configured with two Intel Xeon E5-2600V3 processors, four NVIDIA TITAN Xp GPUs, 64 GB DDR4 memory, and 4 TB hard disk. The server is connected with S5150X-16S-EI high speed switch. The software platform is configured with ubuntu 16.04, Tensorflow 1.5, python 2.7, cuda9.1 and cudnn7.7. We generate a total of 1000 DAG tasks graphs using the graph generator [8], and simulate the DAG tasks scheduling process with a in-house simulator. The distributed heterogeneous system is configured with 3–7 heterogeneous processors with fully connected communication networks.

In the ADTS algorithm, the parameters used in the reinforcement learning are described as follows. The policy network architecture is configured with 3–5 layers of sequence-to-sequence neural networks with each layer having 10–50 neurons. The scale of policy networks depend on the problem space of DAG graphs and the heterogeneous hardware configuration. The learning rate step size α is 0.0005 and the discounted factor γ is 0.99. The number of Monte-Carlo training episodes N is configured with 2500.

In the comparative evaluation, we use the following three performance metrics.

- **Schedule Length Ratio (SLR)**. The key performance metric of a scheduling algorithm is the schedule length (makespan) of its schedule policy. As the sizes of DAG graphs are different among applications, we normalize the schedule length to a lower bound, which is named SLR. The SLR value is defined as

$$SLR = \frac{makespan}{\sum_{n_i \in CP_{MIN}} min_{p_j \in Q} \{w_{i,j}\}} \tag{7}$$

  where the CP_{MIN} denotes that the critical path of a DAG graph is based on the minimum computation costs.
- **Speedup**. The value of speedup for a given graph is the ratio of the sequential execution time to the makespan. The speedup is defined as

$$Speedup = \frac{min_{p_j \in Q} \{\sum_{n_i \in V} w_{i,j}\}}{makespan} \tag{8}$$

  where the sequential execution time is obtained by scheduling all DAG tasks to a single processor that minimizes the overall computation costs (denoted as $min_{p_j \in Q} \{\sum_{n_i \in V} w_{i,j}\}$).
- **Running time of the Algorithms**. A scheduling algorithm's running time is its execution time of producing the output schedule policy for a given DAG tasks graph. This metric represents the cost of the scheduling algorithm.

The DAG tasks graph generator uses the following parameters to quantify the characteristics of the generated DAG graphs, which is similar to [8].

- SET_V = {20,40,60,80,100}
- SET_{CCR} = {0.1,0.5,1.0,5.0,10.0}
- SET_α = {0.5,1.0,2.0}
- SET_{out_degree} = {1,2,3,4,5,v}
- SET_β = {0.1,0.25,0.5,0.75,1.0}

where SET_V denotes the number of tasks in the graph, SET_{CCR} denotes the set of parameter values of the Communication to Computation Ratio (CCR), SET_α denotes the set of parameter values of the graph shape parameter α. SET_{out_degree} denotes the set of values of out degree of a task. SET_β denotes the set of parameter values of the range percentage of computation costs on processors (β) that quantifies the heterogeneity of the processors.

4.2. Performance Comparison

In this subsection, we show the performance comparisons of four DAG tasks scheduling algorithms, the proposed ADTS algorithm, the classic HEFT algorithm and CPOP algorithm [8], and the RANDOM algorithm. The Heterogeneous Earliest Finish Time (HEFT) algorithm selects the task with the highest upward rank value at each scheduling step and assigns the selected task to the processor that minimizes its earliest finish time. The Critical-Path-on-a-Processor (CPOP) algorithm uses the summation of the upward rank and downward rank to denote a task's priority and the selected tasks with the highest priority is assigned to the critical-path processor; otherwise, it is assigned to a processor that minimizes the earliest finish time. The RANDOM algorithm selects random tasks and random processors while satisfying tasks precedence constraints. Note that we ran the RANDOM algorithm for as long as our proposed algorithm did and selected the smallest makespan among many runs.

The ADTS algorithm is non-deterministic, we show the average value of 10 individual runs in the experiment. The DAG tasks graphs are generated using the parameters listed in Section 4.1. As modern big data and machine learning based applications are mostly data-intensive, the DAG graphs are generated with a higher portion of CCR value.

Figure 4 shows the comparison of the average schedule length ratio between the ADTS, HEFT, CPOP, and RANDOM algorithms. The SLR metric represents the schedule quality of each algorithm (lower is better). The closer the SLR value to one, the better the scheduling policy. As the normalization uses the theoretical minimum computation costs, the SLR cannot be less than one.

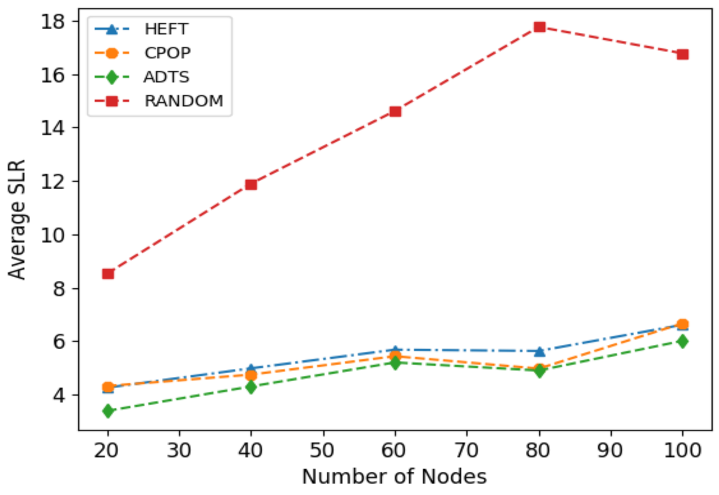

Figure 4. Comparison of the Schedule Length Ratio (SLR).

As can be seen from Figure 4, the ADTS algorithm outperforms both the HEFT and CPOP algorithms. In the 20 tasks DAG graph, the average SLR of ADTS algorithm is 3.391 and the average SLR of HEFT and CPOP are 4.262 and 4.323 respectively, which has 25% reduction of the average SLR. Similarly, in the 40, 60, 80 and 100 tasks of DAG graph scheduling experiments, the SLR of ADTS is consistently lower than both HEFT and CPOP algorithms. The lower SLR achieved by the ADTS algorithm demonstrates that the reinforcement learning could better explore the long term reward, which leads to the better scheduling policies than the traditional heuristic algorithms. Obviously, the RANDOM algorithm has the largest SLR across all settings.

Figure 5 shows the comparison of the average speedup between the ADTS, HEFT, CPOP, and RANDOM algorithms. The average speedup represents the algorithm's ability of scheduling tasks to explore parallel performance (higher is better). Note that the speedup value is calculated via dividing the sequential execution time by the makespan. The sequential execution time is represented by assigning all tasks to a single processor that minimizes the cumulative computation costs. If selecting the processor that maximizes the cumulative computation costs, the value of speedup will be higher. As can be seen from Figure 5, the ADTS algorithm achieves better speedup than HEFT and CPOP algorithms. In the 20 tasks DAG graph experiment, the speedup of ADTS algorithm is 1.087, while the speedup of HEFT and CPOP algorithms are 0.879 and 0.886 respectively. The ADTS algorithm could achieve more than 20% speedup improvement compared with HEFT and CPOP algorithms. The average speedup of the RANDOM algorithm is around 0.4 in all tested DAG graphs.

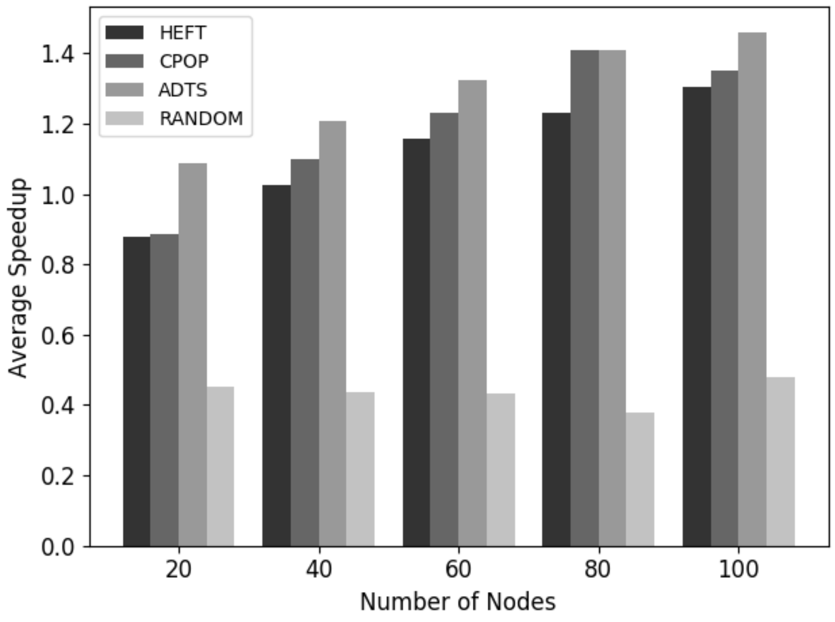

Figure 5. Comparison of the average speedup.

Figure 6 shows the comparison of the average running time of the ADTS, HEFT, CPOP, and RANDOM algorithms. The average running time of a scheduling algorithm represents the average computation costs of execution the algorithm. As can be seen from Figure 6, the ADTS algorithm has higher running time compared with the HEFT and CPOP algorithms. This is because the ADTS algorithm involves the deep neural network reference computations to produce the scheduling policy, which has higher overhead compared with the HEFT and CPOP algorithm. The CPOP algorithm has higher running time compared with the HEFT algorithm. The time complexity of both the CPOP algorithm and the HEFT algorithm is $O(e \times q)$, where e is the number of edges in the graph and q is the number of processors. The time complexity of the ADTS algorithm depends on the policy network architecture. If the neural network reference computation cost is defined as c, then the time complexity of the ADTS algorithm is $O(c \times v)$, where v is the number of tasks. As the RANDOM algorithm uses naive policy and only satisfies task precedence constraints, its time complexity is $O(n)$. However, to demonstrate that the RANDOM algorithm could not progress towards better results as the number of trials increases, we ran the RANDOM algorithm many times, which corresponded to (or even exceeded) the time spent in our proposed algorithm.

4.3. Discussion

From the above comparative performance evaluation, we observe that the reinforcement learning algorithm could achieve better scheduling policies than the classic HEFT and CPOP algorithms. However, as the deep reinforcement learning involves neural network parameters training and inference computation overhead, the running time of the ADTS algorithm is somewhat higher than the traditional heuristic greedy-based algorithms. Fortunately, the ADTS algorithm is designed for static DAG scheduling, which is acceptable for the relatively high running time considering its better schedule quality. What is more, the ADTS algorithm is non-deterministic. In some cases, the training process could not successfully converge to obtain the good policy network model. The reinforcement learning parameters tuning and the network architecture design need some trials to obtain a robust algorithm.

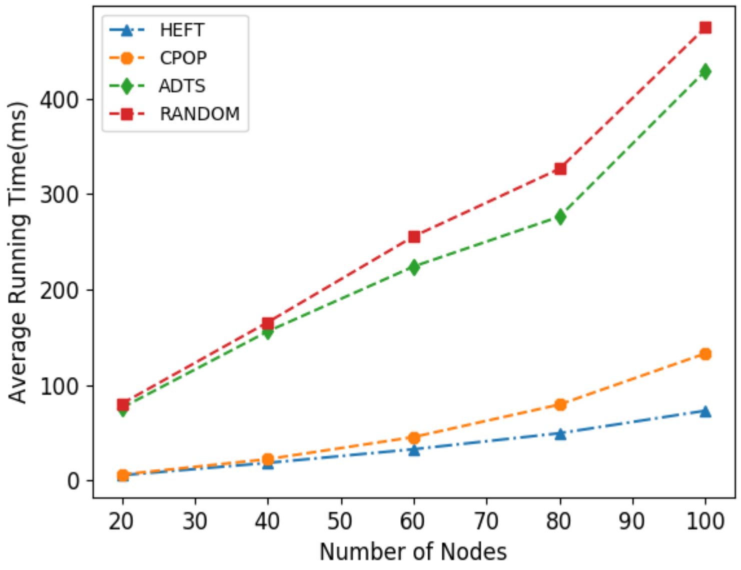

Figure 6. Comparison of the average running time.

Figure 7 shows a learning curve of the ADTS training algorithm under the 20 tasks DAG scheduling environment. As can be seen from the learning curve, the algorithm learns very fast within 400 episodes and gradually exceeds the classic HEFT algorithm after 500 episodes of training. In our experiments, some of the DAG graphs cannot be successfully trained to surpass the classic algorithms. We infer that this problem is due to the unsuitable parameters and the neural network architecture configurations. This unstable training problem needs further investigation and remains as future work.

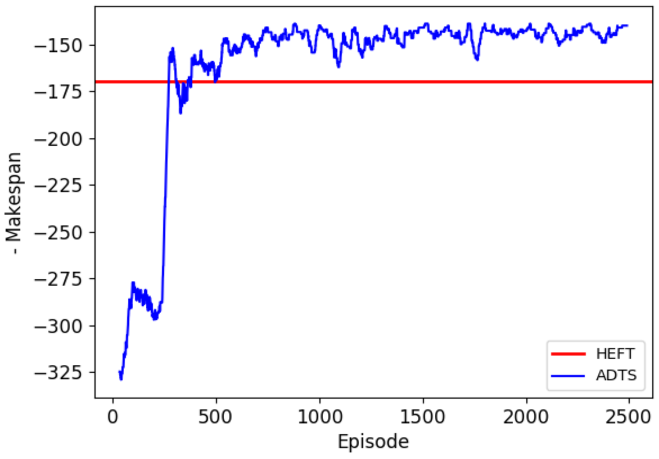

Figure 7. The learning curve of the Adaptive DAG Tasks Scheduling (ADTS) training algorithm.

5. Conclusions

In this paper, we proposed a trust-aware Adaptive DAG tasks scheduling (tADTS) algorithm using deep reinforcement learning and Monte Carlo tree search. The efficient scheduling state space, action space, and reward function were designed to train the policy gradient-based REINFORCE agent. Using the Monte-Carlo method, a large amount of training episodes were generated in a scheduling simulator and the policy network parameters were updated using the simulated episodes. Experimental results showed the effectiveness of the proposed tADTS algorithm compared with the competitive HEFT and CPOP algorithms.

In future work, we plan to investigate the method of hyperparameters tuning to achieve more stable performance of the proposed algorithm. As different hyperparameters will significantly affect the performance of the reinforcement learning and MCTS [44], it is important to find an efficient and automatic way to tune hyperparameters and study their effects in addressing the DAG tasks scheduling problem. As the proposed method is quite general and could be adapted to many applications, future works for applications of this method may include: (1) the implementation of smart task scheduler in heterogeneous high-performance computing or deep learning framework (such as TensorFlow, PyTorch) with hardware constraints, (2) the application of this method used for both online and offline usage scenarios, and (3) the variants of time-sensitive task scheduling with hard or soft deadlines.

Author Contributions: Conceptualization, Y.C. and Y.W.; methodology, Y.C.; software, Z.W.; validation, Z.W. and K.L.; formal analysis, Q.W.; investigation, Y.W.; resources, Q.W.; data curation, Z.W.; writing—original draft preparation, Y.C. and Y.W.; writing—review and editing, Y.W.; visualization, Y.C.; supervision, Q.W.; project administration, Q.W.; funding acquisition, Y.C.

Funding: This research was funded by the National Science Foundation for Young Scientists of China grant number 61802096.

Acknowledgments: Thanks to the anonymous reviewers for their helpful suggestions.

Conflicts of Interest: The authors declare no conflict of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript, or in the decision to publish the results.

References

1. Meng, W.; Li, W.; Wang, Y.; Au, M. H. Detecting insider attacks in medical cyber–physical networks based on behavioral profiling. *Future Gener. Comput. Syst.* **2018**. [CrossRef]
2. Wang, Y.; Meng, W.; Li, W.; Li, J.; Liu, W.X.; Xiang, Y. A fog-based privacy-preserving approach for distributed signature-based intrusion detection. *J. Parallel Distrib. Comput.* **2018**, 26–35. [CrossRef]
3. Wang, Y.; Meng, W.; Li, W.; Liu, Z.; Liu, Y.; Xue, H. Adaptive Machine Learning-Based Alarm Reduction via Edge Computing for Distributed Intrusion Detection Systems. 2019. Available online: https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.5101 (accessed on 26 March 2019).
4. Alajali, W.; Zhou, W.; Wen, S.; Wang, Y. Intersection Traffic Prediction Using Decision Tree Models. *Symmetry* **2018**, *10*, 386. [CrossRef]
5. Abadi, M.; Agarwal, A.; Barham, P.; Brevdo, E.; Chen, Z.; Citro, C.; Corrado, G.S.; Davis, A.; Dean, J.; Devin, M.; et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv* **2016**, arXiv:1603.04467.
6. Ullman, J.D. NP-complete scheduling problems. *J. Comput. Syst. Sci.* **1975**, *10*, 384–393. [CrossRef]
7. Mayer, R.; Mayer, C.; Laich, L. The tensorflow partitioning and scheduling problem: it's the critical path! In Proceedings of the 1st Workshop on Distributed Infrastructures for Deep Learning, Las Vegas, NY, USA, 11–15 December 2017; pp. 1–6.
8. Topcuoglu, H.; Hariri, S.; Wu, M.Y. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Trans. Parallel Distrib. Syst.* **2002**, *13*, 260–274. [CrossRef]
9. Wu, A.S.; Yu, H.; Jin, S.; Lin, K.C.; Schiavone, G. An incremental genetic algorithm approach to multiprocessor scheduling. *IEEE Trans. Parallel Distrib. Syst.* **2004**, *15*, 824–834. [CrossRef]
10. Ahmad, I.; Kwok, Y.K. On exploiting task duplication in parallel program scheduling. *IEEE Trans. Parallel Distrib. Syst.* **1998**, *9*, 872–892. [CrossRef]
11. Mirhoseini, A.; Pham, H.; Le, Q.V.; Steiner, B.; Larsen, R.; Zhou, Y.; Kumar, N.; Norouzi, M.; Bengio, S.; Dean, J. Device placement optimization with reinforcement learning. In Proceedings of the 34th International Conference on Machine Learning, Sydney, Australia, 6–11 August 2017; pp. 2430–2439.
12. Zhang, W.; Dietterich, T.G. A reinforcement learning approach to job-shop scheduling. *IJCAI* **1995**, *95*, 1114–1120.

13. Mao, H.; Alizadeh, M.; Menache, I.; Kandula, S. Resource management with deep reinforcement learning. In Proceedings of the 15th ACM Workshop on Hot Topics in Networks, Atlanta, GA, USA, 9–10 November 2016; pp. 50–56.
14. Orhean, A.I.; Pop, F.; Raicu, I. New scheduling approach using reinforcement learning for heterogeneous distributed systems. *J. Parallel Distrib. Comput.* **2018**, *117*, 292–302. [CrossRef]
15. Goldie, A.; Mirhoseini, A.; Steiner, B.; Pham, H.; Dean, J.; Le, Q.V. Hierarchical Planning for Device Placement. In Proceedings of the Sixth International Conference on Learning Representations, Vancouver, BC, Canada, 30 April–3 May 2018; pp. 1–11.
16. Browne, C.; Powley, E. A survey of monte carlo tree search methods. *IEEE Trans. Intell. AI Games* **2012**, *4*, 1–49. [CrossRef]
17. Kwok, Y.K.; Ahmad, I. Static scheduling algorithms for allocating directed task graphs to multiprocessors. *ACM Comput. Surv.* **1999**, *31*, 406–471. [CrossRef]
18. Amalarethinam, D.; Josphin, A.M. Dynamic Task Scheduling Methods in Heterogeneous Systems: A Survey. *Int. J. Comput. Appl.* **2015**, *110*, 12–18. [CrossRef]
19. Arabnejad, H.; Barbosa, J.G. List scheduling algorithm for heterogeneous systems by an optimistic cost table. *IEEE Trans. Parallel Distrib. Syst.* **2014**, *25*, 682–694. [CrossRef]
20. Palis, M.A.; Liou, J.C.; Wei, D.S.L. Task clustering and scheduling for distributed memory parallel architectures. *IEEE Trans. Parallel Distrib. Syst.* **1996**, *7*, 46–55. [CrossRef]
21. Kanemitsu, H.; Hanada, M.; Nakazato, H. Clustering-Based Task Scheduling in a Large Number of Heterogeneous Processors. *IEEE Trans. Parallel Distrib. Syst.* **2016**, *27*, 3144–3157. [CrossRef]
22. Xu, Y.; Li, K.; Hu, J.; Li, K. A genetic algorithm for task scheduling on heterogeneous computing systems using multiple priority queues. *Inf. Sci.* **2014**, *270*, 255–287. [CrossRef]
23. Meng, X.; Liu, W. A DAG scheduling algorithm based on selected duplication of precedent tasks. *Comput.-Aided Des. Comput. Graph.* **2010**, *22*, 1056–1062. [CrossRef]
24. Rimmel, A.; Teytaud, O.; Lee, C.S.; Yen, S.J.; Wang, M.H.; Tsai, S.R. Current frontiers in computer Go. *IEEE Trans. Comput. Intell. AI Games* **2010**, *2*, 229–238. [CrossRef]
25. Schadd, M.P.; Winands, M.H.; Jaap van den Herik, H.; Chaslot, G.M.J.B.; Uiterwijk, J.W.H.M. Single-Player Monte-Carlo Tree Search. In Proceedings of the International Conference on Computers and Games 2008, Beijing, China, 29 September–1 October 2008; pp. 1–12.
26. Bjornsson, Y.; Finnsson, H. Cadiaplayer: A simulation-based general game player. *IEEE Trans. Comput. Intell. AI Games* **2009**, *1*, 4–15. [CrossRef]
27. Lai, T.L.; Robbins, H. Asymptotically efficient adaptive allocation rules. *Adv. Appl. Math.* **1985**, *6*, 4–22. [CrossRef]
28. Agrawal, R. Sample mean based index policies by o (log n) regret for the multi-armed bandit problem. *Adv. Appl. Probab.* **1995**, *27*, 1054–1078. [CrossRef]
29. Matsumoto, S.; Hirosue, N.; Itonaga, K.; Yokoo, K.; Futahashi, H. Evaluation of simulation strategy on single-player Monte-Carlo tree search and its discussion for a practical scheduling problem. In Proceedings of the International MultiConference of Engineers and Computer Scientists, Hongkong, China, 17–19 March 2010; Volume 3, pp. 2086–2091.
30. McGovern, A.; Moss, E.; Barto, A.G. Building a basic block instruction scheduler with reinforcement learning and rollouts. *Mach. Learn.* **2002**, *49*, 141–160. [CrossRef]
31. Pellier, D.; Bouzy, B.; Métivier, M. An UCT approach for anytime agent-based planning. In *Advances in Practical Applications of Agents and Multiagent Systems*; Springer: Berlin/Heidelberg, Germany, 2010; pp. 211–220.
32. Chaslot, G.; De Jong, S.; Saito, J.T.; Uiterwijk, J. Monte-Carlo tree search in production management problems. In Proceedings of the 18th BeNeLux Conference on Artificial Intelligence, Namur, Belgium, 5–6 October 2006, pp. 91–98.
33. Lenstra, J.K.; Rinnooy Kan, A.H.G. Complexity of Scheduling Under Precedence Constraints. *Oper. Res.* **1978**, *26*, 22–35. [CrossRef]
34. Sahni, S.; Horowitz, E. Combinatorial Problems: Reducibility and Approximation. *Oper. Res.* **1978**, *26*, 718–759. [CrossRef]
35. Garey, M.R.; Johnson, D.S. “Strong” NP-Completeness Results:Motivation, Examples, and Implications. *J. ACM* **1978**, *25*, 499–508. [CrossRef]

36. Kasahara, H.; Narita, S. Practical Multiprocessor Scheduling Algorithms for Efficient Parallel Processing. *IEEE Trans. Comput.* **1984**, *33*, 1023–1029. [CrossRef]
37. Wu, Q.; Wu, Z.; Zhuang, Y.; Cheng, Y. Adaptive DAG Tasks Scheduling with Deep Reinforcement Learning. In *Algorithms and Architectures for Parallel Processing*; Vaidya, J., Li, J., Eds.; Springer International Publishing: Cham, Switzerland, 2018; pp. 477–490.
38. Sutton, R.S.; Barto, A.G. *Reinforcement Learning: An Introduction*; MIT Press: Cambridge, MA, USA, 2011.
39. Kocsis, L.; Szepesvári, C. Bandit based monte-carlo planning. In *European Conference on Machine Learning*; Springer: Berlin/Heidelberg, Germany, 2006; pp. 282–293.
40. Williams, R.J. Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning. In *Reinforcement Learning*; Sutton, R.S., Ed.; Springer: Boston, MA, USA, 1992; pp. 5–32.
41. Sutton, R.S.; McAllester, D.; Singh, S.; Mansour, Y. Policy Gradient Methods for Reinforcement Learning with Function Approximation. In Proceedings of the 12th International Conference on Neural Information Processing Systems, Denver, CO, USA, 29 November–4 December 1999; MIT Press: Cambridge, MA, USA, 1999; pp. 1057–1063.
42. Silver, D.; Schrittwieser, J.; Simonyan, K.; Antonoglou, I.; Huang, A.; Guez, A.; Hubert, T.; Baker, L.; Lai, M.; Bolton, A.; et al. Mastering the game of Go without human knowledge. *Nature* **2017**, *550*, 354–359. [CrossRef]
43. Auer, P.; Cesa-Bianchi, N.; Fischer, P. Finite-time Analysis of the Multiarmed Bandit Problem. *Mach. Learn.* **2002**, *47*, 235–256. [CrossRef]
44. Islam, R.; Henderson, P.; Gomrokchi, M.; Precup, D. Reproducibility of Benchmarked Deep Reinforcement Learning Tasks for Continuous Control. *arXiv* **2017**, arXiv:1708.04133.

© 2019 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (http://creativecommons.org/licenses/by/4.0/).