Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 156 (2019) 29–36

Procedia Computer Science

www.elsevier.com/locate/procedia

8th International Young Scientist Conference on Computational Science

Workflow scheduling using Neural Networks and Reinforcement Learning

Mikhail Melnik*, Denis Nasonov

ITMO University, 49 Kronverksky Pr., Saint-Petersburg 197101, Russia

Abstract

The development of information technologies entails a nonlinear growth of both volumes of data and the complexity of data processing itself. Scheduling is one of the main components for optimizing the operation of the computing system. Currently, there are a large number of scheduling algorithms. However, even in spite of existing hybrid schemes, there remains a need for a scheduling scheme that can quickly and efficiently solve a scheduling problem on a wide range of possible states of the computing environment, including the high heterogeneity of computational models and resources, and should have an ability to self-adapt and self-learn. At present, artificial intelligence and neural networks are the most popular methods for working with data and solving a wide range of problems, but they are not developed enough to solve the scheduling problem. Therefore, in this paper we propose a scheduling scheme based on Artificial Neural Networks and the principles of Reinforcement Learning.

Keywords: workflow scheduling; artificial intelligence; reinforcement learning; cloud computing.

1. Introduction

The development of information technologies entails a nonlinear growth of both volumes of data and the complexity of data processing itself. To follow the growing requirements for data processing, existing technologies, platforms and cloud services are being developed and modernized, providing increasingly high-quality capabilities for organizing computation and scaling. The main direction of development of these technologies is the update of hardware and corresponding software, the development of automation and monitoring tools for computing pipelines [3]. However, the development of methods for optimization of internal processes is also a crucial part that is required to achieve maximum computing performance. Scheduling is one of the main components for optimizing the operation of the computing system. The main aim of the scheduling is to distribute workload over resources of computing infrastruc-

E-mail address: mihail.melnik.ifmo@gmail.com

1877-0509
10.1016/j.procs.2019.08.126

ture. Efficient scheduling of tasks can reduce the amount of data transferred, allocate the computation of models on computational resources corresponding to their requirements, thereby reducing the waiting time and execution time of tasks [13].

Currently, there are a large number of scheduling algorithms [12, 15] for various types of computational workload (workflows, streaming topologies) and for various computing environments (grids, clouds, clusters, supercomputers). Heuristic [11, 6] and metaheuristic [8, 9] algorithms can be classified as the main classes of scheduling algorithms. However, even in spite of existing hybrid schemes [5, 14], there remains a need for a scheduling scheme that can quickly and efficiently solve a scheduling problem on a wide range of possible states of the computing environment, including the high heterogeneity of computational models and resources, and should have an ability to self-adapt and self-learn. At present, machine learning and neural networks are the most popular methods for working with data and solving a wide range of problems, but they are not developed enough to solve the scheduling problem.

The main problem is the discreteness of the problem, which does not allow us to clearly understand the type and structure of the input and output data for constructing algorithms based on machine learning. In [17, 16], there are researches on the use of reinforcement learning (RL) for the scheduling problem, but the problem statements used in these works have a superficial view aimed at scheduling in Grid systems with the distribution of tasks according to external resources of Grids. In [1], RL was used to plan MapReduce tasks, and in [2] the similar approach was aimed at planning several composite applications in the cloud environment. The disadvantages of these works are that they consider only basic information (the number of tasks or the number of resources) for describing a set of states and do not use such models as neural networks. There are two highly related to our research works with similar ideas. In [10] authors proposed a RL based scheduling scheme with their definition of input states for learning agents and its integration in MLBox framework. In this work, the state is composed of a list of characteristics of computing resources, taking into account that all tasks have approximately the same execution time, but have dependencies among themselves. In [7], authors developed another RL scheduling algorithm based on pointer networks. Their input states and actions are composed as production of all tasks and resources. For each pair (task, resource) in input state following parameters are defined: runtime, failure probability, communication cost and a number of depended tasks. Moreover, authors analyzed their neural architecture and built performance models for evaluation of runtime and failures probabilities of tasks. In comparison to these works, our main idea is focused on analyzing and formalizing a more complete view of the scheduling problem in terms of RL that implies the formation of input states and possible actions considering the structure of workflows.

Therefore, the study of artificial intelligence methods for solving the scheduling problem is a relevant area, and in this paper we propose a scheduling scheme, based on Artificial Neural Networks (ANN), principles of Reinforcement Learning (RL) and analysis of workflow structure.

2. Problem statement

2.1. Workflow scheduling problem

For the workflow scheduling problem it is required to define workload and computing environment. Workload is presented as a directed acyclic graph (DAG) $Wf =< T, E >$ where $T = \{t_i\}$ is a set of tasks and $E = \{e_{j,k}\}$ is a set of edges. Each task t_i represents a computational model or application that should be executed. An edge $e_{j,k}$ between tasks t_j and t_k corresponds to data dependencies between them. In this case, task t_k is a child task and it can't start its execution before it receives all required input data from parent task t_j. Two functions are defined for convenience, $succ(t)$ is a set of children of task t and $pred(t)$ returns its parent tasks. Tasks without parents are called initial tasks and tasks without children are called end tasks. Multiple workflows may exist in workload and we can merge all their tasks in one DAG by a pseudo-task. Computing environment is composed of a set of physical or virtual computing resources $N = \{n_i\}$. Each computing node n_i has its characteristics and specifications such as number of CPU cores, RAM. Often, within the scope of scheduling problem, resources are characterized by their capacity, expressed in million instructions per second (MIPS). Resources are connected via set of networks which are presented as a set of bandwidths $B = \{b_i\}$. Schedule is an ordered assignment of tasks to resources in the form $S = \{(t_i, n_j)\}$. There are various criteria (makespan, cost, reliability) and constraints (deadline, budget) in the field of scheduling problem, but in this work we perform optimization only by makespan - total execution time of workflow. For estimation of

makespan it is required to define performance models for tasks execution time and data transfers times between tasks. Task's execution time on node is defined as $ET(t,n)$. Time to transfer data from parent task t_i to its child t_j is defined as $TT(t_i,t_j) = e_{i,j}/b_k$, where b_k is a bandwidth which connects nodes where t_i and t_j were assigned accordingly. If both tasks were assigned on one node, then transfer time becomes zero. We assume that data can start moving between nodes immediately after the completion of the parent task. Therefore, full execution time of a task is a sum $FT(t,n) = max_{t_i \in pred(t)}(TT(t_i,t))+ET(t,n)$ and actual finish time of task $AFT(t)$ defines the point in time when the task will be completed, taking into account all dependencies and queues in the schedule. Thus, workflow total execution time is defined as a finish time of last executed task $makespan = max_{t_i \in T}(AFT(t_i))$. The aim of workflow scheduling problem is to find the optimal schedule S_{opt} that will minimize makespan of workflow.

2.2. Reinforcement learning

RL is a form of machine learning techniques where agent (scheduling algorithm) forms an optimal strategy under experience of interaction with the external environment (computing environment). The strategy determines a choice of the action a_i from the set of possible actions A under the current state s from the state space S. The action at time moment t is selected on the basis of the constructed utility function $Q(s,a)$ for evaluation of actions according to the principle $a_t = argmax_a Q_t(s_t,a)$. When interacting with the environment and taking an action, the agent enters a new state s' and receives a reward r from the environment $r(s,a)$, which is used to train the evaluation function $Q(s,a)$. During the operation of the agent, it accumulates data in the form of a sequence $s_o a_o r_o s_1 a_1 r_1 s_2$.... The evaluation function is updated according to the SARSA principle: $Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha[r_t + \gamma Q(s_{t+1},a_{t+1}) - Q(s_t,a_t)]$, where α is a learning rate and γ is a future actions importance factor (in experiments α anneals from 0.9 to 0.1 and γ is set to 0.5). To solve complex problems, including a continuous state space, the approximation function $f(s,a)$ may be constructed in a form of prediction models such as neural networks.

3. Neural network scheduling algorithm

Developed in this research neural network scheduling (NNS) algorithm is based on reinforcement learning or Q-learning principles. NNS algorithm represents an agent that aimed to perform effective scheduling of tasks across computing nodes. The main part of algorithm is a neural network that allows you to evaluate and select the most advantageous assignment of a task to a resource (action) at the moment. The network's architecture is presented in such a way that it accepts up to m tasks and n computing resources. Selected tasks and resources are encoded in the vector parameters - the input state s taking into account the state of the computing environment, workflow structure and the current schedule. Content of parameter vector consists of several categories: general workflow parameters; tasks parameters; parameters of resources; parameters for assigned on resources tasks. This architecture of the scheduler allows perform dynamic scheduling without reference to the dimension of the problem. It should be noted that if it is necessary to scale the computations, several scheduler agents can work at the same time and be responsible for different tasks or resources, including the fact that they can have a different structure and be trained under various conditions. The main acting entities of the NNS algorithm are presented in Fig. 1.

The algorithm's agent interacts with the computing environment, receiving from it the current state s that aggregates information about the load of the computing resources, workload, current schedule and performance models for estimation of tasks execution. When input state s is encoded into parameters vector, it is fed to the input of the neural network. Neural network predicts the most effective action a - selection of task that should be scheduled at this moment of time and selection of node for this task. NNS agent transmits this action to computing environment that applies it and adds the task to the schedule for the specified resource. When the task has been scheduled, computing environment estimates current workflow execution time, its new state s' and evaluates a reward r for NNS agent. NNS agent saves received sequence $\{s,a,r,s'\}$ in its memory. While received state is not terminal, NNS agent performs new actions according to new input states. SARS memory storages data in a form of $sars'$ vectors for neural network learning that takes place in the background.

The main focus of this work is to form a set of parameters when encoding the state of the environment. It is assumed that the algorithm should be able to perform dynamic scheduling of heterogeneous computational load. As workflows differ from each other, so same workflows may differ in input data and scale. This leads to the need for the algorithm

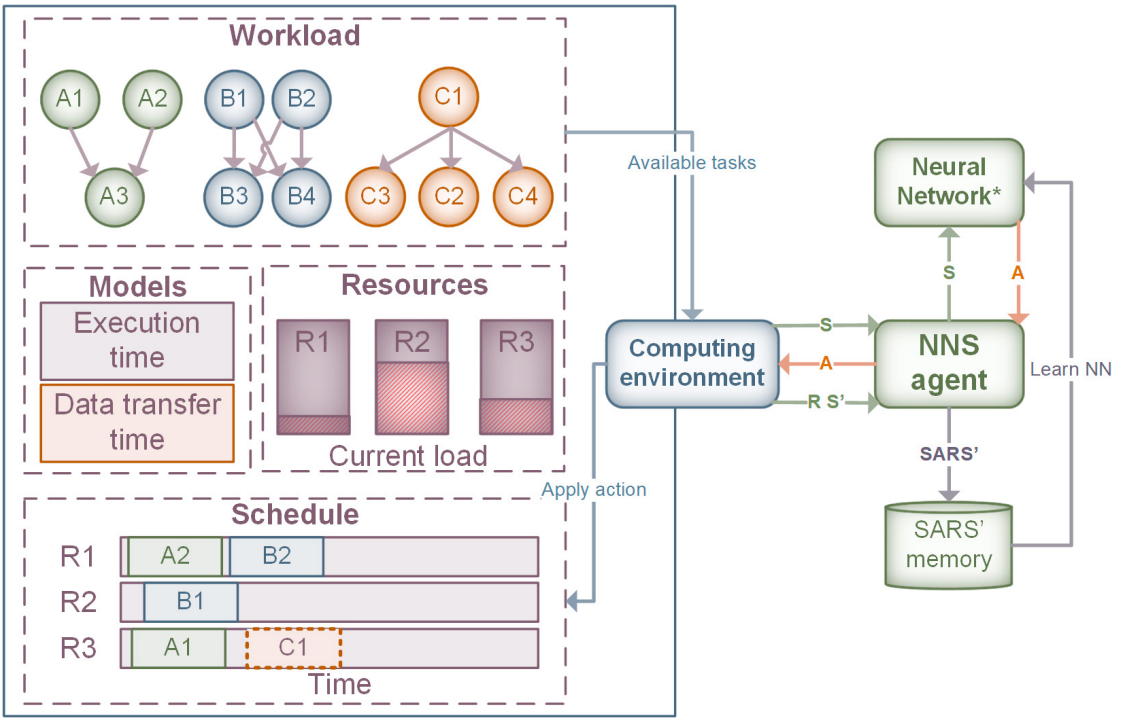

Fig. 1. Neural Network Scheduling (NNS) scheme

to operate with relative values. For this purposes, in order to go over to the relative values of the parameters, we define theoretically the worst time of workflow's execution $WT(Wf)$. Theoretical the worst execution time is estimated as the sum of the execution time of all tasks assigned to the weakest by capacity resource and the time it takes to transfer all data through the network, regardless of the need for these transfers. The most part of following state's parameters are defined with the usage of WT.

In Table 1 we present all parameters that we tried to use for NNS algorithm. We define column 'Relative' to highlight these parameters as relative to Wf's worst execution time. Moreover, we add column 'Use' to indicate which parameters were used in the experimental study. For example, parameters 'width', 'height' and numbers of parents or children were not used since they can't accurately show how weighty these values are when schedule various Wfs. Despite the fact that without them our algorithm showed better results, in our future works we will try to add them in another relative form. In current form of our input state, the relationship between tasks is sufficiently determined by the parameter tr_data.

With the set of used parameters, the dimension of input state vector calculated as $5+n+3*m+2*n*m$, where m - number of tasks and n - number of nodes for which our NNS agent was constructed. For example, size of input vector for architecture with 5 tasks and 4 nodes becomes 64. Only available tasks can be considered for selection by NNS scheme. If size of available tasks less than m, empty tasks' parameters are set to zero. Number of actions corresponds to $n*m$. Reward function we estimated by the following equation:

$$r(s,a) = \frac{worst\_time}{current\_makespan} * \frac{scheduled\_tasks}{n}$$

In a case of terminal state, reward just equals to ratio of Wf's resulted makespan and its theoretical worst time. The scheme of neural network and its parameters is presented in Fig. 2.

Table 1. Parameters of input state vector.

Type	Name	Relative to *WT*	Description	Used
Wf	scheduled_tasks	-	Ratio of the number of completed tasks to their total number	+
Wf	width	-	Maximum number of parallel tasks	-
Wf	height	-	Height of Wf graph	-
Wf	runtime_weight	+	Ratio of execution time of all tasks to the Wf's worst time	+
Wf	runtime_med	+	Ratio of the median of tasks' execution time to the worst time	+
Wf	data_weight	+	Ratio of all data transfer times to worst time	+
Wf	data_med	+	Ratio of the median of data transfer times to worst	+
Wf	parents_max	-	Maximum amount of parent tasks	-
Wf	parents_avg	-	Average amount of parent tasks	-
Wf	children_max	-	Maximum amount of children among tasks	-
Wf	children_avg	-	Average number of children	-
Node	load_time	+	Ratio of resource release time to worst time	+
Task	t_runtime	+	Ratio of task's execution time to worst	+
Task	t_data	+	Ratio of task's input data transfer time to worst	+
Task	t_parallel	-	Number of parallel tasks for this	-
Task	t_parents	-	Number of parent tasks of this	-
Task	t_children	-	Number of children of this task	+
Task, Node	tr_comptime	+	Ratio of task's execution time on specific node to worst	+
Task, Node	tr_data	+	Ratio of data transfer time for this specific task on specific node to worst	+

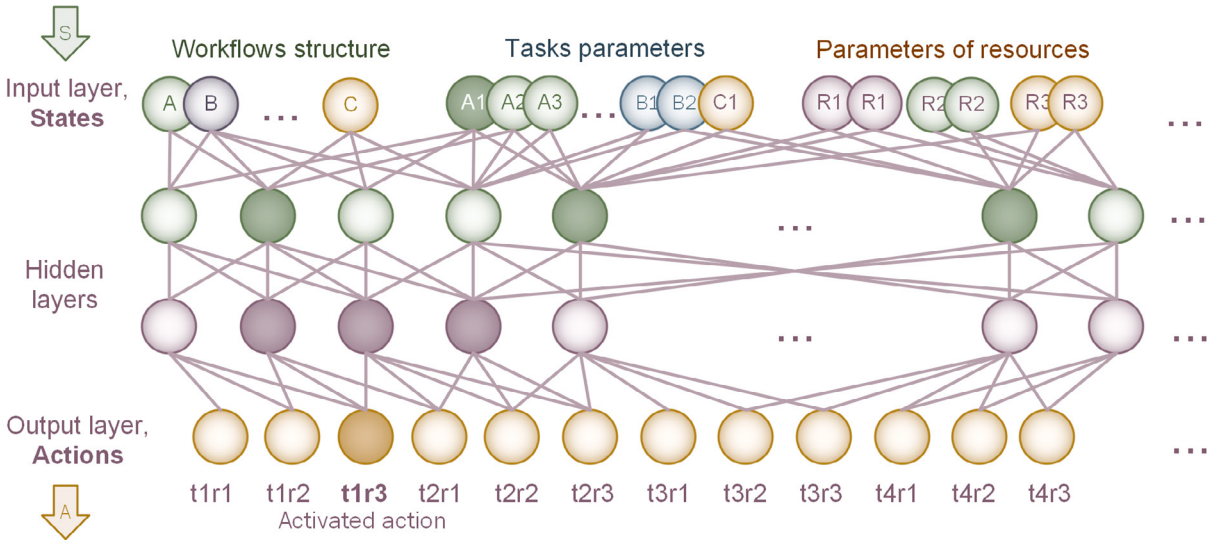

Fig. 2. Structure of neural network for workflow scheduling

4. Experiments

4.1. Simulation environment

To carry out experimental studies, the simulation model was developed for the implementation of workflow's execution in a computing environment. Well-known workflows from Pegasus [4] were used as input data to conduct the experiments. This input data presents logs of workflows' execution and contains runtimes for all tasks *t_runtime*, volumes of their input and output data and dependencies between tasks. Computing nodes are presented in a form of their capacities. In our experiments, we defined a list of four resources with capacities of 4, 8, 8, 16, which can

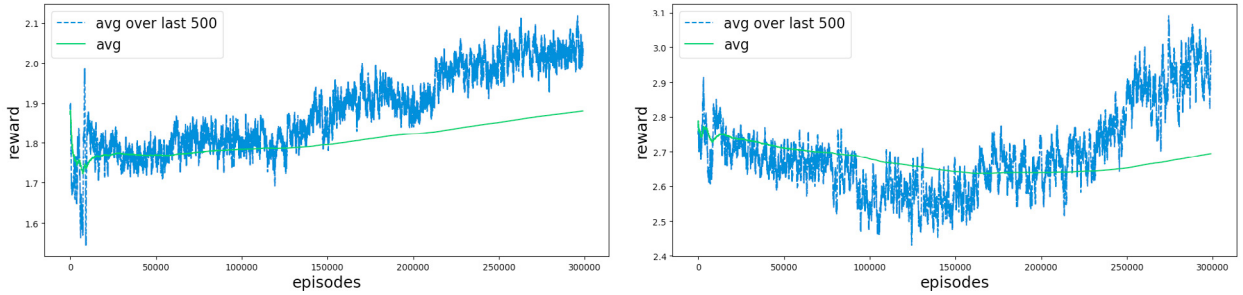

Fig. 3. Evolution of rewards during NNS training on the set of (left) small workflows; (right) medium workflows

corresponds to their number of CPU cores. Task's execution time depends on node's capacity and is estimated by the following equation:

$$ET(t,n) = \frac{t_{runtime}}{log(n_{capacity} + 1)}$$

Such a set of nodes was made in order to highlight one less and one more powerful resources. Network's bandwidth was set to 10 MB/s. We selected two classes of workflows accordingly to their size: small workflows with 5-10 tasks (Gene2life, Floodplain, Scoopsmall, Leadadas, Molsci); medium workflows with 25-30 tasks (Montage, CyberShake, Inspiral, Epigenomics).

4.2. Learning actor structure

For the experiments we build NNS algorithm with a such architecture that takes 5 tasks and 4 nodes. Its neural network's structure was set to $64 \times 1024 \times 512 \times 256 \times 20$ full-connected network including input and output layers with ReLU activations and Adam as optimizer. Experiments are structured in such a way that the NNS agent is trained on one of the selected sets of workflows. The scheduling process consists of following steps:

1. select a workflow from input list of workflows
2. initialize new schedule
3. encode input state s
4. predict action a by NNS agent
5. apply action to schedule
6. receive reward r and next state s', save $sars'$ in NNS' memory
7. if s' is not terminal, continue workflow's scheduling, go to 1.

These steps corresponds to one episode where one chosen workflow is scheduled. We perform neural network's update each 10 episodes with $batch\_size = 128$ input vectors taken from NNS agent's memory (memory size is 1000).

4.3. Results

In the first part of experiments we trained our NNS algorithm on the set of small workflows. The NNS agent performed 300000 episodes. Received reward values in aggregated form are presented in Fig. 3 for both sets: small workflows and medium workflows. During this learning process algorithm starts to learn how to assign tasks more effectively. It starts trying to place tasks on the most performance resource R4 and in cases where parallel tasks are exists, it places it in parallel on average-performance nodes R2 or R3. Examples of schedules of Gene2life workflow at initial and final stages of NNS learning are pictured in Fig. 4. Tasks execution time goes through the X axis, which is bounded to the right by the theoretical worst possible time. Resources are marked along the axis Y with their schedules for execution of tasks accordingly. Even though R4 executes tasks faster than R2 or R3 on 28% and then R1 on 76% it

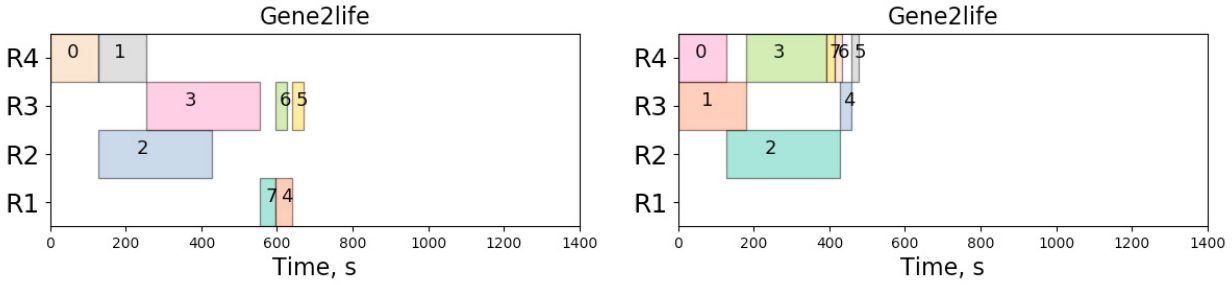

Fig. 4. Examples of Gene2life's schedules at initial stages (left) and final stages (right)

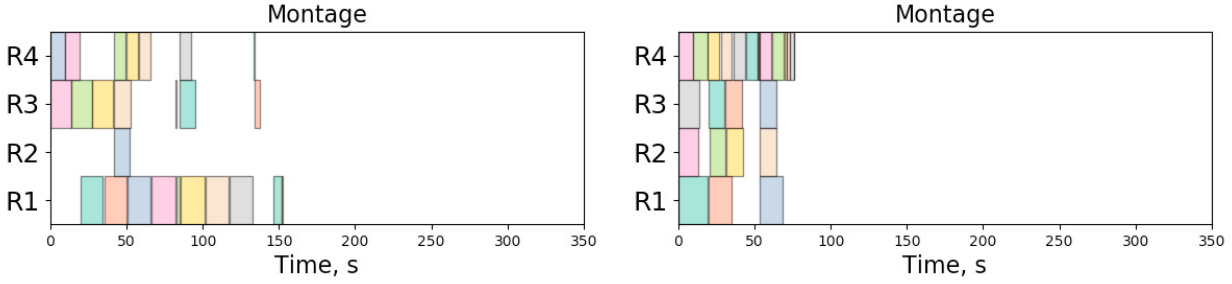

Fig. 5. Examples of Montage's schedules at initial stages (left) and final stages (right)

is not easy for the algorithm to catch this difference within the receiving rewards. Nevertheless, Gene2life's makespan has been reduced by 39% from 670 to 480 seconds.

In the second part of experiments we conduct the same structured NNS algorithm but with the set of medium-sized workflows as input. All workflows in both sets have various structure and degree of parallelism. Moreover, they differ in the ratio of execution times to data transfer times in both relative and absolute values. However, over episodes, the algorithm also learned how to produce more quality schedules. For example, schedules of Montage workflow at initial and final stages are presented in Fig. 5.

Results of workflows' makespan reduction during the NNS learning process is shown in Table 2. According to results, NNS algorithm learned how to produce effective schedules with average improvement of 76.7% across all workflows in compare to schedules which were performed at initial steps. The difference between workflows also lies in the possible improvement over the theoretical execution time. This leads to a large dispersion of reward values and complicates the learning process for different workflows and even for random workload. However, in future works we plan to improve encoding process of input state to perform more relative and suitable set of parameters for heterogeneous workload. Moreover, an important direction of our future works is to rebuild neural network with the usage of RNN and LSTM that is crucial since there is evident connection between tasks order and their execution on nodes.

5. Conclusion

In this work we introduced a workflow scheduling algorithm NNS that based on principles of artificial intelligence and reinforcement learning. The main idea of this research is to analyze workflow scheduling problem's context and build an encoder to provide vectored form of this context. We form such a set of parameters which relates to workflow's general structure, tasks' characteristics, nodes and performance models which are possible to estimate tasks' execution time. Experiments were conducted on two benchmark sets of workflows with various sizes and structures. Results of experiments showed that built NNS algorithm is able to learn how to provide qualitative schedules in terms of workflows' makespan.

Table 2. Results of makespan reduction during NNS learning process for all workflows.

Workflow	Tasks	makespan initial (s)	makespan final (s)	improvement
Gene2life	8	669.42	478.51	39.9%
Floodplain	7	142217.51	101138.32	40.6%
Scoopsmall	6	2717.22	1290.40	110.5%
Leadadas	6	7069.87	5148.50	37.3%
Molsci	5	1442.71	900.01	60.3%
Montage	25	152.30	76.10	100.1%
CyberShake	30	893.65	465.44	92.0%
Inspiral	30	3799.69	2484.03	52.9%
Epigenomics	24	20990.28	8156.4	157.3%

Acknowledgements

This research is financially supported by The Russian Foundation for Basic Research, Agreement #18-37-00416 and The Russian Science Foundation, Agreement #17-71-30029 with co-financing of Bank Saint Petersburg.

References

[1] Basu, A., 2017. Learning based workflow scheduling in hadoop. Int. J. Appl. Eng. Res 12, 3311–3317.
[2] Cui, D., Ke, W., Peng, Z., Zuo, J., 2015. Multiple dags workflow scheduling algorithm based on reinforcement learning in cloud computing, in: International Symposium on Computational Intelligence and Intelligent Systems, Springer. pp. 305–311.
[3] Da Cunha Rodrigues, G., Calheiros, R.N., Guimaraes, V.T., Santos, G.L.d., De Carvalho, M.B., Granville, L.Z., Tarouco, L.M.R., Buyya, R., 2016. Monitoring of cloud computing environments: concepts, solutions, trends, and future directions, in: Proceedings of the 31st Annual ACM Symposium on Applied Computing, ACM. pp. 378–383.
[4] Deelman, E., Vahi, K., Juve, G., Rynge, M., Callaghan, S., Maechling, P.J., Mayani, R., Chen, W., Ferreira da Silva, R., Livny, M., Wenger, K., 2015. Pegasus: a workflow management system for science automation. Future Generation Computer Systems 46, 17–35. URL: http://pegasus.isi.edu/publications/2014/2014-fgcs-deelman.pdf, doi:10.1016/j.future.2014.10.008. funding Acknowledgements: NSF ACI SDCI 0722019, NSF ACI SI2-SSI 1148515 and NSF OCI-1053575.
[5] Delavar, A.G., Aryan, Y., 2014. Hsga: a hybrid heuristic algorithm for workflow scheduling in cloud systems. Cluster computing 17, 129–137.
[6] Ijaz, S., Munir, E.U., 2018. Mopt: list-based heuristic for scheduling workflows in cloud environment. The Journal of Supercomputing , 1–29.
[7] Kintsakis, A.M., Psomopoulos, F.E., Mitkas, P.A., 2019. Reinforcement learning based scheduling in a workflow management system. Engineering Applications of Artificial Intelligence 81, 94–106.
[8] Liu, L., Zhang, M., Buyya, R., Fan, Q., 2017. Deadline-constrained coevolutionary genetic algorithm for scientific workflow scheduling in cloud computing. Concurrency and Computation: Practice and Experience 29, e3942.
[9] Nasonov, D., Melnik, M., Shindyapina, N., Butakov, N., 2015. Metaheuristic coevolution workflow scheduling in cloud environment, in: 2015 7th International Joint Conference on Computational Intelligence (IJCCI), IEEE. pp. 252–260.
[10] Orhean, A.I., Pop, F., Raicu, I., 2018. New scheduling approach using reinforcement learning for heterogeneous distributed systems. Journal of Parallel and Distributed Computing 117, 292–302.
[11] Rahman, M., Hassan, R., Ranjan, R., Buyya, R., 2013. Adaptive workflow scheduling for dynamic grid and cloud computing environment. Concurrency and Computation: Practice and Experience 25, 1816–1842.
[12] Singh, L., Singh, S., 2013. A survey of workflow scheduling algorithms and research issues. International Journal of Computer Applications 74.
[13] Varghese, B., Buyya, R., 2018. Next generation cloud computing: New trends and research directions. Future Generation Computer Systems 79, 849–861.
[14] Vasile, M.A., Pop, F., Tutueanu, R.I., Cristea, V., Kołodziej, J., 2015. Resource-aware hybrid scheduling algorithm in heterogeneous distributed computing. Future Generation Computer Systems 51, 61–71.
[15] Wu, F., Wu, Q., Tan, Y., 2015. Workflow scheduling in cloud: a survey. The Journal of Supercomputing 71, 3373–3418.
[16] Yao, J., Tham, C.K., Ng, K.Y., 2006. Decentralized dynamic workflow scheduling for grid computing using reinforcement learning, in: 2006 14th IEEE International Conference on Networks, IEEE. pp. 1–6.
[17] Zeng, B., Wei, J., Liu, H., 2009. Dynamic grid resource scheduling model using learning agent, in: 2009 IEEE International Conference on Networking, Architecture, and Storage, IEEE. pp. 67–73.