DeepSched: A Deep Representation Of Scheduling Policies For Heterogeneous Distributed Systems

Mohamed Shawky | Remonda Talaat | Mahmoud Adas | Evram Youssef
SEC 2, B.N 16 | SEC 1, B.N 20 | SEC 2, B.N 21 | SEC 1, B.N 9
{mohamedshawky911, remondatalaat21, mido3ds, evramyousef}@gmail.com

***Abstract*—Task scheduling is one of the core problems in any distributed system. The ability of the system to assign tasks to its different resources to achieve the best possible performance with the least resource consumption is the main challenge in distributed systems research. The heterogeneity of the distributed systems adds another layer of complexity on top of the mentioned problem. The different execution time on different processors and different task dependencies make the scheduling problem very computationally intensive. Usually, researchers and developers turn to greedy-based methods or even recursive optimization methods to reach some optimal or sub-optimal solutions. However, due to the nature of the problem, these solutions might take very long time and usually degrade in complex situations. In this work, we propose the idea of scheduling algorithms approximation, where we approximate the complex scheduling algorithms using other models to achieve the same results but with lower execution time. Neural networks are proposed for this task, as they are great function approximators. The complexity of neural networks comes at training time, however they can maintain constant execution time in inference. This constant execution time is relatively low, as it's a simple feedforward process. This makes neural networks a perfect candidate for us, however this comes at a cost of static scheduling scenarios, which is useful in some case such as simulation workflow on supercomputers. The proposed network can achieve almost the same performance of the approximated method (HEFT) on seen data during training and slightly worse performance on unseen data with constant execution time at different input sizes.**

I. Introduction

The continuous evolution of computing power and growth of widely-distributed applications and *Internet of Things* (IoT) [1] have driven our needs for better resource allocation methods. The performance of any system critically depends on the algorithms used to schedule tasks on its resources. These systems contain limited resources that should be allocated properly, in order to perform the required tasks with maximum efficiency avoiding starvation and resource exhaustion. The general objective of any scheduling technique is to get the best possible performance with a reasonable resource utilization.

Recently, the heterogeneity of the computational systems has increased tremendously, due to the nature of the applications and the great advances in IoT and distributed systems. A *heterogeneous system* can be defined as a range of resources, different in underlying architecture, targeting different types of computational tasks. The usage of a heterogeneous system has proven to be very efficient in increasing the system performance and reducing the overall power consumption. Recent studies [2] show that *heterogeneous system* can outperform the best homogeneous systems by as much as 21%, with 23% energy savings and a reduction of 32% in Energy Delay Product. These numbers can be improved even more with better task scheduling and resource allocation methods.

However, this comes with the problem of complex task scheduling process. The task scheduling problem for a heterogeneous computing system is more complex than that for a homogeneous system, because of the different execution rates of processors and different communication rates among different processors, which add extra layers of complexity to the problem.

The first type of methods to turn to is greedy-based methods, where the best possible choice in the current situation is considered. The execution time of these methods rapidly increases with input size and their performance degrades in complex situations. However, the parameters of the scheduling problem in heterogeneous distributed systems make it very suitable for various optimization techniques. Recursive optimization techniques are very efficient in solving many optimization problem. Previous work has investigated the usage of techniques such as *genetic algorithms* [3] to solve the task scheduling problem. Also, the usage of a progressive decision making techniques such as *Reinforcement Learning* [4] for task scheduling in heterogeneous systems has been investigated.

In this paper, we investigate the ability of advanced predictive models to learn a representation of static local task scheduling problem. This learned representation is used to approximate the scheduling performance of other scheduling techniques. *Neural Networks* are our best candidate for this task. Neural networks are being used in many fields replacing traditional methods, such as image classification [5], object detection [6], neural language modelling [6], image generation [7] and others. They have proven to be more efficient than other traditional methods. The complexity of the neural networks comes in the development phases, where the network is designed and trained. However, in inference phase, neural networks can achieve very accurate results at high speed. Recent work has been done on optimizing neural network inference through various techniques, such as quantization [8] and pruning [9].

The recent advances in *Deep Learning* research are considered in designing and training our scheduling network. *Heterogeneous Earliest Time First* (HEFT) [10] is used as a target, where the network learns to approximate its performance. Also, we include a genetic algorithm experiment for the same problem for comparison. We will discuss our approach to the problem and its limitations.

II. Related Work

Since task scheduling in heterogeneous systems is a wide problem, a lot of research has been conducted in this area. Various techniques have been introduced for the problem. These techniques are categorized based on multiple aspects [11] and some of which are adapted from task scheduling methods for homogeneous systems.

A. Scheduling Categories

In this work, we focus on three basic categorizations of heterogeneous task scheduling methods.

1) Application-Specific vs. System-Specific: Heterogeneous scheduling methods can be divided into two main categories based on their target metric.

Application-Specific scheduling targets task execution speed (performance), where scheduling decisions are determined based on many parameters including application performance, task inter-dependency and the availability of resources. *System-Specific* scheduling targets resource utilization, where scheduling decisions are based on the percentage of time a resource is available or busy.

2) Global vs. Local: Also, heterogeneous scheduling methods can be divided into global and local scheduling.

Global scheduling can migrate the tasks from one processor to another, meanwhile *local* scheduling can't migrate, so once a task is assigned to a processor, it stays in its queue.

3) Static vs. Dynamic: Finally, heterogeneous scheduling methods can be divided into static and dynamic scheduling.

In *static* scheduling, the tasks required to be scheduled are defined before scheduling. However, in *dynamic* scheduling, new tasks can be introduced during the scheduling process.

B. Heterogeneous Earliest Time First (HEFT)

HEFT [10] is one of the most well-established algorithms for task scheduling in heterogeneous systems. It's a local static scheduling algorithms, which deals with a *Direct Acyclic Graph* (DAG) of tasks displaying the dependencies between different processes. Each task has a running time for each different machine and a time for communicating the results to children tasks. HEFT is essentially a greedy algorithm, so it's unable make short-term sacrifices for long term benefits. Consequently, HEFT can fail in complex situations. A wide range of algorithms [11] follows HEFT further improving the scheduling performance.

C. Optimization Algorithms

Recent research has been conducted to use optimization algorithms and predictive models to schedule tasks on heterogeneous system, since these algorithms have proven to be very efficient in solving complex computational problems.

1) Genetic Algorithms (GA): GA [3] can be used to efficiently schedule tasks in different systems. It's used for static scheduling, where a fixed population of tasks are scheduled recursively on multiple processors, based on a specific fitness function.

2) Reinforcement Learning (RL): Also, RL [4] has been a strong candidate for the problem, as the scheduling problem can be formulated as an *RL* problem, solved by various *RL* techniques. RL techniques are effective in solving progressive decision making problems, so they are more likely to be used in dynamic scheduling.

D. Neural Networks

Recent advances in *Deep Learning* have enabled neural networks to dominate many fields, such as computer vision, natural language processing, robotics and others.

Neural networks are able to perform perception tasks at a human-level accuracy. Also, neural networks are very efficient in function approximation and representation learning. These properties make neural networks a perfect choice for massive scale automation of many tasks. Some studies even investigated the usage of *Artificial Neural Networks* (ANNs) [12] in task scheduling in heterogeneous systems. Most of these studies focus on the usage of *Hopfield Nets* [13] and *Inhibitor Neurons* [12].

However, recent advances in *Deep Learning* have come up with new techniques of using neural networks in complex situations. One of the most shiny techniques are *Recurrent Neural Networks* (RNNs) [14]. RNNs have been widely used in many fields such as natural language processing [15], speech recognition [16], image captioning [17], time-series prediction [18] and others. They have proven to be very efficient in learning complex temporal information from time-series data. That's why, they are used with sequential data such as videos, speech and text. This makes RNNs our main choice, as we are dealing with sequential data which is the tasks to be scheduled.

In this paper, we make use of recent advances in *Deep Learning* such as RNNs and other techniques to learn a deep representation of the scheduling problem and use it to approximate the performance of heterogeneous scheduling methods.

III. Methodology

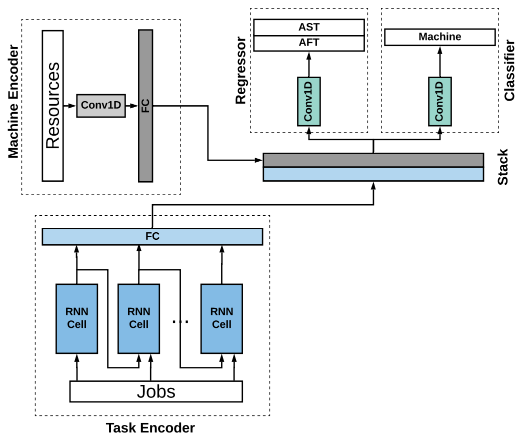

Fig. 1. The architecture of the proposed neural network (DeepSched) for distributed heterogeneous systems scheduling.

A. Motivation

Heterogeneous task scheduling is a relatively new concept, which lacks developers support. It's an active area of research, where many techniques are being developed through time. Being able to achieve the best possible performance on the given tasks, while keeping good utilization of resources is a tedious problem in heterogeneous systems. Aside from being accurate, the scheduling methods have to be fast in its execution, also they should not consume the system resources.

Heterogeneous scheduling methods are very computationally expensive. However, neural networks are relatively fast in inference and research is being conducted on optimizing inference even more. So, if a neural network can approximate the performance of current scheduling method, it can replace this method in production.

We study the ability of neural networks to approximate some heuristic methods and explore its accuracy and execution time against both heuristic and approximated methods.

B. Formulation

First, let us specify the main target of this work. As the task scheduling in heterogeneous distributed system is a vast field with lots of settings and algorithms [11], we choose our scheduling setting to be local and static. The goal of this work is to explore the ability of neural networks to approximate the performance of some well-established scheduling algorithms. In our setting, the algorithms are provided a list of tasks, each has its own execution time on different machines and a set of heterogeneous machines, on which the tasks are executed. We compare the results neural networks to some other heuristic and approximate methods.

Although, we choose local static scheduling, we assume that the neural approximation methods can work in other settings as well.

C. Baseline

Two baseline methods are considered. We choose *Heterogeneous Earliest Finish Time* (HEFT) [10] to be our heuristic baseline, as it's one of the widely-used scheduling algorithms in heterogeneous systems. Also, genetic algorithms [3] is provided as a baseline for approximate methods.

1) Heterogeneous Earliest Time First (HEFT): HEFT [10] is an old and well-established algorithm for scheduling in heterogeneous systems. It has been used as a baseline in research due to its near-optimal results. Basically, it's a greedy algorithm that chooses some process to run on some machine, if the former has the earliest finish time, with consideration of the special nature of the heterogeneous system. However, HEFT isn't practical for real-world systems, as the finish time of a process isn't usually known in advance. Also, the greedy nature of the HEFT scheduler makes it prone to failure in complex situations.

2) Genetic Algorithms: Genetic algorithms [3] have proven to be very efficient in solving some of the hardest optimization problem. Previous work on the use of genetic algorithms in task scheduling [3] is adapted to the heterogeneous setting by adding parameters for execution time of different tasks on different machines.

The targeted *fitness function* is the average waiting time W_t for all tasks. The fitness function for a specific schedule *S* is given by:

$$\mathcal{F}itness(S) = \frac{\sum_{i=1}^{N} W_{ti}}{N} \tag{1}$$

where *N* is the number of tasks in the schedule, W_t is the waiting time for a specific task.

D. Neural Networks (Deep Learning)

The main novel idea of this work is the use of *neural networks* for the scheduling problem. Neural Networks are known to be very good at learning representations and function approximation. In this work, we propose an architecture that can be used to reach a good approximated solution for the scheduling problem. *Offline Optimization* of neural networks is to be used, in order to be able to train the network on scheduling samples and then use the trained network to do inference. We avoid using *online optimization*, used in some fields such as neural style transfer [19]. This is because it's slow and we mainly care about execution time of the scheduling method.

Previous work tried to use neural networks to solve scheduling problem through *Hopfield Nets* [13] and *Inhibitor Neurons* [12]. However, we try to make use of the recent advances in *Deep Learning* to redefine the scheduling problem and solve it.

As mentioned, The provided input is a list of tasks with its information and a set of machines. We want the network to learn an implicit representation of the given tasks and machines. Then, the network learn the mapping between the learned representation and the desired schedule. In other words, the target of the network is to take two input list of information of both tasks and machines and then output for each task, the machine to be executed on, the actual start time and the actual finish time.

1) Architecture: Our architecture consists of two encoders and one decoder, as shown in Fig.1.
The first encoder is a fully convolutional network of *1D* convolutional layers, to which is machines specifications are passed. This encoder learns a representation for the machines to be able to use it later in producing the outputs.

The second encoder is an *Recurrent Neural Network* (RNN) [14], to which the tasks are passed one by one ordered by the arrival time. This network learns a representation of the tasks preserving the arrival order. RNN is chosen due to its proven effectiveness in dealing with sequential data, as they can learn complex temporal information from time sequences.

The two representations are then stacked and passed to two modules. A classification module which states the machine on which a specific task is executed. A regression module that identifies the actual start time and the actual finish time. Mainly, the two modules consist of *1D* convolutional layers.

Basically, the classifier is used during inference to produce the required schedule by determining the required machine for each task. However, the regressor is used at training along with the classifier, in order to add extra constraints on the network training and to enhance the loss function. This helps the network to learn a good representation of the scheduling problem and its parameters, also prevents the network from making some implicit mistakes in scheduling such as having two tasks executing on the same machine during the same range of time.

2) Objective Function: The cost function defined for this architecture is a sum of two criteria. The two criteria are categorical cross entropy *(CE)* on the predicted machine to run a specific job M_p and mean square error *(MSE)* on the predicted actual start time AST_p and the predicted actual finish time AFT_p. So, it can be summarized as follows:

$$\mathcal{L}oss(L) = CE(M_p, M_t) + MSE(AT_p, AT_t) \tag{2}$$

where M_p is the predicted machine, M_t is the ground truth machine, AT_p is the predicted actual time range and AT_t is the ground truth actual time range.

3) Drawbacks: The main drawback of the proposed solution is that we have to know the maximum number of machines and the maximum number of tasks to be able to define the network. These numbers are used to define the network and loss function dimensions. So, the network design has to force some static scenarios to work on. However, this can be the case in some situations such as simulation workflow in supercomputers, where the tasks, to be executed, are known before the simulation starts. So, our proposed method can be suitable for these scenarios, as it can provide static execution time for different input sizes.

IV. Experimental Setup

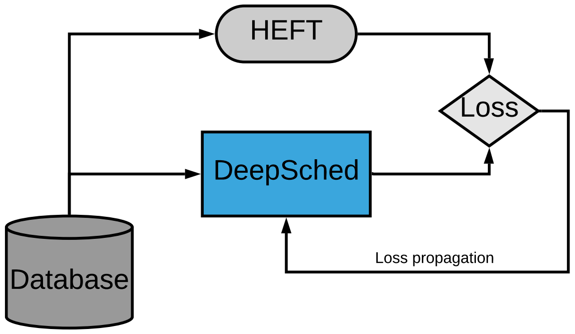

Fig. 2. The training framework of DeepSched network. A database of tasks and resources is provided to both network and HEFT. The loss of the network is measured based on HEFT produced schedule.

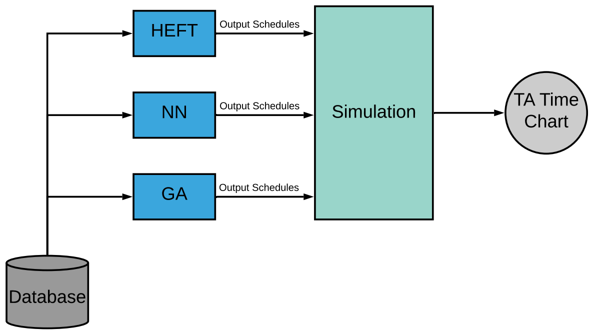

Fig. 3. The evaluation framework of DeepSched network. A database of tasks and resources is provided to all three schedulers. The output schedules are passed to a simulation stub, which runs the schedules and calculates average turnaround time.

Our evaluation methodology of the proposed network is based on comparison with two basic method. One of them is a heuristic method, which is *Heterogeneous Earliest Time First* (HEFT) and the other is an approximate method, which is *Genetic Algorithms* (GA).

The results are compared based on the execution time, as well as scheduling performance. For execution time, we measure the total time consumed by scheduling different number of tasks. For scheduling performance, we measure the average turnaround time for all scheduled tasks. Our network is supposed to achieve better execution time with similar performance to HEFT on previously seen data and slightly lower performance on unseen data.

A. Baseline

HEFT, which is a heuristic method, is used as a comparison baseline and also to generate the training data, where the network approximates HEFT performance. GA solution, which is an approximate method, is only used as a comparison baseline.

1) HEFT: The basic architecture from the original paper [10] is used in a framework with the proposed neural network shown in Fig.2. HEFT receives some data from a defined database of tasks and resources and produces the output schedule, which is used as labels for the network training data. This way the network can approximate the performance of the HEFT scheduling algorithm by learning to map the input tasks and resources to the schedule produced by HEFT on a large database.

2) Genetic Algorithms: The methodology from [3] is used to develop a genetic algorithm solution for heterogeneous systems. Extra parameters are added to the population to represent the different resources. Also, the execution time is expanded to be a value for each task running on a specific machine, instead of only one execution time value. The used fitness function is the average waiting time 1. Random initialization is used along with Crossover. The results of this method are compared to the output of the proposed network.

B. Datasets

A database of tasks and resources is adapted from *Dataset for Task scheduling in Cloud using CLoudsim* [20]. The used database consists of 100000 samples represented as a list of tasks requires to be scheduled. Each sample contains from 486 to 84654 tasks, arranged as a *directed acyclic graph* (DAG) to represent the task dependencies and each task has an execution time for each specific processor. Also, the database contains 188 resource instances (processors), on which the given tasks scheduled. The desired output of such data is to determine, for each task, the optimal processor to run on, the actual start time (AST) and the actual finish time (AFT).

The provided samples are divided into 90000 samples for training and validation and 10000 samples for testing. We maintain diversity in the test data by including different number of tasks in the test samples.

C. Implementation Details

Our proposed network consists of four main modules.

1) Machine Encoder: This module takes resource details as an input and learns a meaningful representation of them. It consists of one *Conv1D* [1] layer to extract

[1]Convolution operation performed along one dimension only, usually used for regression.

the basic features of each processor, followed by one *fully connected* layer to learn a compact representation of all processors, in order to be passed to the output modules. The common input features for each processor passed to this module are number of processor cores, core frequency and memory capacity.

2) Task Encoder: This module takes the time sequence of tasks required to be scheduled and learns a meaningful representation of them through time. *Recurrent neural networks* are used to learn the desired temporal information of the tasks. The module consists of a single *bidirectional gated recurrent unit* (GRU) [14] layer to extract temporal features of each task, followed by one *fully connected* layer to learn a compact representation of all tasks, in order to be passed to the output modules. The common input features for each task passed to this module are arrival time, burst (execution) time for each processor, task priority and memory utilization.

The outputs of the previous stages are fused with each other by stacking, in order to form one chunk of information, which is then passed to the two output modules.

3) Classifier: The target of this module is to select the optimal running processor for each task amongst the given set of processors. The input to this module is the stacked information of both resources and tasks. It consists of a single *Conv1D* layer, which is used to select running processor for each single task in the form of one hot encoding of the length of resources number.

4) Regressor: The target of this module is to predict the actual start time and the actual finish time of each task on the scheduled processor. The input to this module is the stacked information of both resources and tasks. It consists of a single *Conv1D* layer, which is used to regress the actual start and finish times for each task.

D. Training

The network is trained on HEFT scheduling outputs. Cross-entropy is used as a classification loss and mean squared error (MSE) is used as a regression loss. The summation of the two losses 2 is used to train the network using gradient descent with *Adam* optimizer and a learning rate of $1e$-5 with decay factor of 0.5 every 20 epochs. The network is trained for 100 epochs. The task encoder only takes input of maximum number of tasks, so inputs of lower dimension are zero-padded along the length dimension to the maximum number (84654 tasks).

The training takes around 35 minutes on *Nvidia GTX* 1070 and around 10 hours on *Nvidia GTX* $840m$, which is a relatively small time based on both *GPUs* performance.

The proposed network is implemented in *Python* using *PyTorch* framework [21]. Our source code is publicly available at https://github.com/DarkGeekMS/deepsched.

E. Inference

Inference is performed on the trained network using both train and test data. We use some of the train data, in order to test the ability of the network on previously seen data against HEFT. We use the same 188 resources information for all tests, however the input list of tasks is zero-padded to the maximum number of tasks, exactly like training.

F. Evaluation

Fig.3 shows our evaluation framework of the network. The evaluation of our proposed method is conducted through running inference on both seen and unseen data of different lengths. These results of the network are compared to the output of HEFT and GA methods in the same scenarios.

This is done by passing the schedule produced by the three methods through a simple simulation that runs the proposed schedule and calculates the average turnaround time of all tasks. Evaluation is also conducted on the execution time of each scheduling method, as the speed of the scheduling algorithm is one of our main concerns.

V. Discussion and Results

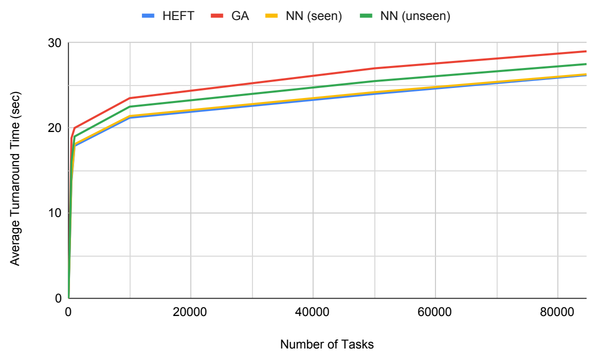

Fig. 4. Average turnaround time produced from the three discussed schedulers at different input sizes.

TABLE I
TIME COMPARISON BETWEEN THE THREE DISCUSSED SCHEDULERS AT DIFFERENT INPUT SIZES. TIME IS MEASURED IN SECONDS.

Input Size	HEFT	GA	NN
486	31.5	67.5	0.125
10000	305.5	602.5	0.125
84654	670.0	3967.0	0.125

A. Scheduling Performance

Fig.4 shows the scheduling performance of different scheduling methods at different input sizes. The proposed network shows almost the same average turnaround time as HEFT algorithm for the seen data during training. This is an indicator that the network can learn the underlying representation of the scheduling problem. However, this isn't enough, as the network needs to be tested on unseen data, too. For unseen data, the network shows slightly worse performance than HEFT, but still better than that of genetic algorithms. This indicates the ability of the network to generalize well to other inputs, as long as the network is provided by the same static scenarios and the same maximum number of tasks and resources.

These results refer to the network ability to learn a good representation to the input machines and scheduling problem. The network performance degrades with high input sizes, as it's approximating HEFT, which is basically a greedy algorithm and prone to failure at complex situations and large number of input tasks. Genetic algorithms shows similar behaviour too, due to large population size.

B. Execution Time

The main advantage of using neural networks is constant inference time. Once the network is trained, it takes a static execution time in inference, as it's a simple feedforward process. Table I shows the execution time of the discussed schedulers at 3 different input sizes. These number are produced running on *Intel i5-6600K.*

We can see that the network consumes constant time in inference with different input sizes. This is because the network is trained on the maximum input length. However, HEFT and genetic algorithms execution time increases with input size. GA time increases more rapidly than HEFT, as it's a recursive optimization method but HEFT is a greedy-based method. The performance and static execution time of the network in inference make the network very useful in the situations where static scheduling environment is provided.

C. Results Discussion

The previous results show that neural networks can approximate a scheduling algorithm for heterogeneous distributed systems, resulting in a significantly good performance and fast execution. The proposed network can perform well, even on unseen data it can produce efficient schedules.

However, this comes with a drawback of static scenarios. The neural network is trained with fixed input size, which is the maximum number of tasks and resources, in our case. At inference time, the input to the network can't exceed that number. So, we have to define the maximum number of tasks to be scheduled, as well as the maximum number of resources. Moreover, the scheduling scenario for the proposed network should be static, where all available tasks should be known before the scheduler starts.

This gives neural networks a great potential to be a good task scheduler in static scenarios, where the maximum number of tasks and resources are known. The network can be trained offline and then deployed to perform task scheduling. However, in other dynamic scenarios, neural networks might not help much.

VI. CONCLUSION AND FUTURE WORK

A. Conclusion

In this work, we propose a new approximation method of task scheduling algorithms in heterogeneous distributed system using neural networks. The proposed network is used to approximate HEFT scheduler in static scheduling scenarios. The network can perform as accurate as HEFT on seen data during training and sightly lower performance on unseen data. However, the network can maintain a constant execution time at different input sizes, while HEFT execution time increases rapidly. Moreover, the trained network can outperform some recursive optimization techniques such as genetic algorithms both for seen and unseen data, as genetic algorithms can suffer from failures and long execution time due to random initialization and large population size.

These results show that neural network, which are great function approximators, have the potential to approximate different scheduling methods for heterogeneous distributed systems. They can learn good representation of the scheduling problem, while maintaining constant execution time in inference. So, neural networks can be a better choice than other techniques in certain scenarios.

The main drawback of this method is that static scenarios must be provided. Maximum number to tasks to be scheduled and maximum number of resources must be known before training the network. However, the proposed network is relatively small and doesn't consume much time to train on low or mid-range *GPUs*.

In conclusion, It's a trade-off between constant time execution and dynamic systems, where good performance and constant execution time can be achieved in neural networks at static scenarios. This can be useful in some cases such as simulations on supercomputers, where the workflow is static and initially defined. Meanwhile, other solutions, such as RL solutions, can work well in a dynamic environment with no degradation with the complexity of the environment. This is important in IoT-based application, where the system is totally dynamic and can have different unexpected events.

B. Future Work

This work can be improved in two possible ways. *First,* other heterogeneous task scheduling methods can be approximated using neural networks. As we mentioned, the proposed method can be expanded to approximate other scheduling methods as long as static scenarios provided. Neural networks can even approximate theoretical methods, that achieves optimal results, to a great extend. These methods aren't used in real life due to there time complexity. However, if these methods are approximated using neural networks, they can be deployed in real life applications achieving optimal or near-optimal performance.

Second, Other possible approximation methods can be tried through neural networks, in order to confront dynamic scenarios. Online optimization can be a solution for dynamic scenarios. However, online network optimization can be very computationally-expensive, as the network trains during inference optimizing its parameters to adapt to new inputs. Another possible solution is to use the proposed network as a function approximator in a *Reinforcement Learning* (RL) environment, in order to be able to learn dynamically in case of new introduced tasks.

The two previously mentioned methods can provide the proposed network with extra flexibility to be able to adapt and tune its parameters to learn from new unexpected inputs introduced through dynamic environments.

REFERENCES

[1] Dr Miraz, Maaruf Ali, Peter Excell, and Rich Picking. A review on internet of things (iot), internet of everything (ioe) and internet of nano things (iont). pages 219–224, 09 2015.

[2] Ashish Venkat and Dean M Tullsen. Harnessing isa diversity: Design of a heterogeneous-isa chip multiprocessor. In *2014 ACM/IEEE 41st International Symposium on Computer Architecture (ISCA)*, pages 121–132. IEEE, 2014.

[3] Ranjeet Kumar, Dr.Rakesh, Er.Rajiv, Er Gill, and Ashwani Kaushik. Genetic algorithm approach to operating system process scheduling problem. *International Journal of Engineering Science and Technology*, 2, 09 2010.

[4] Alexandru Iulian Orhean, Florin Pop, and Ioan Raicu. New scheduling approach using reinforcement learning for heterogeneous distributed systems. *Journal of Parallel and Distributed Computing*, 117:292 – 302, 2018.

[5] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. *Commun. ACM*, 60(6):84–90, May 2017.

[6] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks, 2015.

[7] Tero Karras, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. Analyzing and improving the image quality of stylegan, 2019.

[8] Yoni Choukroun, Eli Kravchik, Fan Yang, and Pavel Kisilev. Low-bit quantization of neural networks for efficient inference, 2019.

[9] Seul-Ki Yeom, Philipp Seegerer, Sebastian Lapuschkin, Simon Wiedemann, Klaus-Robert Müller, and Wojciech Samek. Pruning by explaining: A novel criterion for deep neural network pruning, 2019.

[10] H. Topcuoglu, S. Hariri, and Min-You Wu. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Transactions on Parallel and Distributed Systems*, 13(3):260–274, March 2002.

[11] Dr. Mamta Padole and Ankit Shah. *Comparative Study of Scheduling Algorithms in Heterogeneous Distributed Computing Systems*, pages 111–122. 01 2018.

[12] Daniel Chillet, Antoine Eiche, Sébastien Pillement, and Olivier Sentieys. Real-time scheduling on heterogeneous system-on-chip architectures using an optimised artificial neural network. *Journal of Systems Architecture - Embedded Systems Design*, 57:340–353, 04 2011.

[13] Saratha Sathasivam and Wan Ahmad Tajuddin Wan Abdullah. Logic learning in hopfield networks, 2008.

[14] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling, 2014.

[15] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate, 2014.

[16] Dario Amodei, Rishita Anubhai, Eric Battenberg, Carl Case, Jared Casper, Bryan Catanzaro, Jingdong Chen, Mike Chrzanowski, Adam Coates, Greg Diamos, Erich Elsen, Jesse H. Engel, Linxi Fan, Christopher Fougner, Tony Han, Awni Y. Hannun, Billy Jun, Patrick LeGresley, Libby Lin, Sharan Narang, Andrew Y. Ng, Sherjil Ozair, Ryan Prenger, Jonathan Raiman, Sanjeev Satheesh, David Seetapun, Shubho Sengupta, Yi Wang, Zhiqian Wang, Chong Wang, Bo Xiao, Dani Yogatama, Jun Zhan, and Zhenyao Zhu. Deep speech 2: End-to-end speech recognition in english and mandarin. *CoRR*, abs/1512.02595, 2015.

[17] Md. Zakir Hossain, Ferdous Sohel, Mohd Fairuz Shiratuddin, and Hamid Laga. A comprehensive survey of deep learning for image captioning. *CoRR*, abs/1810.04020, 2018.

[18] Sharat C. Prasad and Piyush Prasad. Deep recurrent neural networks for time series prediction, 2014.

[19] Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge. A neural algorithm of artistic style, 2015.

[20] Muhammad Ibrahim. Dataset for task scheduling in cloud using cloudsim, 2020.

[21] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf,

Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library, 2019.