On Domain based Task Allocation in Network Powered by Computing

Preprint · January 2024

DOI: 10.13140/RG.2.2.16273.40805

CITATIONS
0

READS
4

4 authors, including:

Evgeniy Stepanov
Lomonosov Moscow State University
7 PUBLICATIONS **11** CITATIONS
SEE PROFILE

Smelyasnskiy Ruslan
Lomonosov Moscow State University
3 PUBLICATIONS **0** CITATIONS
SEE PROFILE

Ruslan. L. Smeliansky
Lomonosov Moscow State University
122 PUBLICATIONS **847** CITATIONS
SEE PROFILE

On Domain based Task Allocation in Network Powered by Computing

Stepanov E.P.[a,*], Smeliansky R.L.[a], Balashov V.V.[a], Plakunov A.V.[b], Xia Zhu[c], Jianing Pei[c], Zhen Yao[c]

[a] *Lomonosov Moscow State University Russia*
[b] *Applied Research Center for Computer Networks Russia*
[c] *Huawei Technologies Co. Ltd. China*

Abstract

The problem of task allocation in Network Powered by Computing (NPC) infrastructure is considered. NPC is a new generation of computational infrastructure, where the space of computational resources is considered as unbounded, i.e. not localized in any specific DC, HPC or Edges. The key contribution of this paper is Domain based Task Allocation (DoTA) method for NPC infrastructure that distributes the tasks between different domains. Each domain is a local space of computational resources that can include several locations like DC, HPC or Edges and has a centralized designated node that is responsible for selecting the best candidates in the domain for the task execution. Then designaeted nodes cooperatively select the single candidate that meets the task requirements and provides the most even resource distribution. This paper presents DoTA detailed description and experimental comparison with heuristic algorithm.

Keywords: MARL, task allocation, NPC

1. Introduction

Recently, the landscape of computational infrastructure is in dramatic changes under the pressure of application requirements [1, 2]. The suit of the properties of modern applications can be summarized as follows: distributed, self-sufficient, work in real time, elastic, cross-platform, actively interact and synchronize, and are easy to update. The definitions of these terms are in [1].

For further understanding, it is important to recognize that an application is made up of interrelated components, which we will refer to as application functions (ApF). The analysis of requirements of modern application to the computational infrastructure presented in [2, 3, 4] shows the trend of ubiquitous application deployment. We are moving to the era when data processing resources and data transmission resources form an unbounded space for computing - computational infrastructure. Further we will call such computational infrastructure Network Powered by Computing (NPC).

This paper is focused on application function distribution in NPC environment. Briefly NPC functional architecture can be described as following. It consists of several planes:

- data processing (DP);
- data transmission (DT);
- data processing control (DPC);
- data transmission control (DTC);
- administration, orchestration and management (AOM).

Application functions will require computing resources, that are represented by DP plane. To deliver ApF input data to allocated computing resources and result to the user (that may be represented by other ApF), the overlay network is required, presented by DT plane. The control for ApF placement and data transmission is provided by DPC and DTC planes respectively. Also DPC plane deduces the service-level agreement (SLA) requirements for the connection between ApFs and pass it to DTC plane. Finally, AOM plane orchestrates interactions between ApFs in accordance with application topology, collects NPC resource consumption statistics by every AF, secures management and administration of NPC.

The focus of this paper is DPC plane operation, especially the problem of AF placement. The interaction between DPC and DTC planes is quite important, however the more simplified problem is considered - DPC plane will request the channel with the highest bandwidth. More details about DTC plane operation can be found in [5].

According to NPC architecture, DPC operates on the special facility that is responsible for the integration of every data processing resource (computational node - CN) with the data transmission network (DTN) and the external sources of computational service requests. Call this facility NPC router (NPCR). NPCR replaces several devices at once - a task manager, a traffic and task router, VPN-gateway, CPE, and supplies the following functionality:

- distribution of application functions (ApF)/ virtual

*Corresponding author

network functions (VNF) across computational nodes (CN) of DP plane;

- decision making: is it worth to execute the certain ApF/VNF on the CN connected to this current NPCR or not;
- forwarding ApF/VNF that was not accepted by the current facility under some reason to other facilities where their computational resources are much more promising from the point of Application execution efficiency as a whole;
- optimal data traffic routing as between ApF as between corresponding VNF;
- provision of the transport connection that meets the required Service Level Agreement (SLA).

In those cases when the NPCR provides the input for external sources of data, applications, computational service requests to the NPC resources, we will call such NPCR as pole.

This paper has been dedicated only to one problem from the list above – ApF/VNF distribution. ApF/VNF will be called further as *task*, that need be allocated to computing node, satisfying memory, computing and time requirements. In other words, this paper considers the task allocation problem in NPC environment. The proposed solution can be applied mot only in NPC, but also in environments like CFN [3], CPN [2] etc.

Recent works [7, 8, 9, 10, 11, 12, 13] on load balancing methods showed focus on Machine Learning (ML) methods, especially multi-agent reinforcement learning (MARL). The scale and rate of task allocation requests in NPC do not let us get the proper solution for speed and accuracy of task allocation by traditional centralized methods. Our previous work [5] also showed benefits in applying Multi-agent Reinforcement Learning approach to traffic distribution problem in NPC. That is why the research keeps focus on ML.

Based on examining of the works related to ML methods for task allocation a new method is proposed called Domain-based Task Allocation (DoTA). The idea is a division of NPCR network into domains; task allocation problem is solved in each domain. Then designated nodes of the domains collectively make a decision to which domain the task should be sent. This method combines multi-agent reinforcement learning and domain-based approach to achieve fast task allocation solution maximizing the fraction of satisfied requests.

The paper is organized as follows. In the section 2 we describe the task allocation optimization problem in NPC environment. The related works that deal with close problem by ML algorithms are discussed in the section 3. The proposed new method DoTA is presented in the section 4. The results of the simulation research are presented in the section 5.

2. Task Allocation Problem Statement

Let us formulate the load balancing problems in terms of NPC. The task allocation problem considered in this paper is stated as follows.

2.1. Input

- NPC network $G : (V, E, W)$ of computational nodes, where
  - V - set of nodes.
    Each node is a system with an uniform computing environment and its own scheduler. Each node has a corresponding NPCR, that can receive the task allocation request.
  - E - set of edges (network links) between nodes
  - $W = \{\omega_(i, j)\}$ - the weight matrix, $\omega_(i, j)$ denotes the bandwidth between nodes i and j

  Each node i is described by a pair $\{c_i, s_i\}$, where
  - c_i - the total computing resources (CPU cycles per second) of node i;
  - s_i - the total storage resources (memory) of node i;
- Set J of dynamically arriving computational tasks, where each task $j \in J$ has the following specification:
  - t_j - arrival time of the task j;
  - n_j - arrival node of the task j;
  - m_j - size of input data for the task j;
  - $\widehat{c_j}$ - amount of computing resources (CPU cycles) required by the task j, which defines the execution time e_{ji} - execution time estimation of the task j on node i;
  - $\widehat{s_j}$ - amount of storage resources (memory) required by the task j;
  - p_j - maximum allowed response time (transmission + processing) of the task j; we assume the size of task execution results to be small, so their transmission time can be ignored.

The task allocation problem is considered in streaming form, so that the NPC router operates in the timeline of tasks arrival and has no information on the future tasks. Meanwhile the history of previous task arrivals and allocations is available to the NPCR.

The NPC router must choose the execution (destination) node for every task and find a route for the task data transmission from the initial node to the destination node. Same bandwidth is allocated for the task transmission in each route link, and this allocation is not changed during the task transmission.

There are no task queues on the nodes. On every node, preemptive earliest deadline first (EDF) scheduling scheme

is used; therefore tasks execution schedule for a node is determined by task start times (when task arrives to the destination node) and task deadlines (calculated from maximum response times). The reason for choosing preemptive EDF is that it is an optimal scheme if task postponing is not allowed [6], which is our case.

2.2. Output

Tasks execution schedule W, including :

- assignment of tasks to the nodes;
- tasks execution schedule for each node;
- schedule of links bandwidth allocation;
- set U of unscheduled tasks.

A task becomes unscheduled if the NPCR is unable to assign it to a node without violating the correctness constraints on the schedule.

2.3. Correctness constraints

Schedule must be correct, that is:

- every task in it (except tasks from U) finishes within its maximum response time;
- memory capacity of the nodes is never overloaded;
- network links are never overloaded.

2.4. Objective function

The objective function is formulated for the time interval (t_1, t_2) and incorporates the following components:

1. computing resources utilization averaged among nodes:

$$\Gamma(t_1, t_2) = \frac{1}{|V|} \sum_{i=1}^{(|V|)} \frac{c^i_{used}(t_1, t_2)}{C_i(t_1, t_2)} \quad (1)$$

where $c^i_{used}(t_1, t_2)$ is the amount of node's computing resources (cycles) used in the interval (t_1, t_2), and $C_i(t_1, t_2)$ is the total computing resources of the node in this interval.

2. time-averaged storage resources utilization, averaged among nodes:

$$\Delta(t_1, t_2) = \frac{1}{|V|} \sum_{i=1}^{(|V|)} \frac{s^i_{avg}(t_1, t_2)}{s_i} \quad (2)$$

where

$$s^i_{avg}(t_1, t_2) = \frac{1}{t_2 - t_1} \sum_{j=0}^{k-1} (l_{j+1} - l_j) h_j \quad (3)$$

where l_j are time instances, that belong to (t_1, t_2), in which memory utilization is changed due to task start or finish; additionally, $l_0 = t_1$ and $l_k = t_2$; h_j is the new memory utilization in t_j.

The objective function for an interval (t_1, t_2) is:

$$\Phi(t_1, t_2, W) = \sum_{i=1}^{|V|} (\gamma[(\frac{c^i_{used}(t_1, t_2)}{C_i(t_1, t_2)} - \Gamma(t_1, t_2))^2 + (\frac{s^i_{avg}(t_1, t_2)}{s_i} - \Delta(t_1, t_2))^2]) + \delta|U| \quad (4)$$

where γ, δ are positive constant factors.

2.5. Optimization problem

$$\min_{\{W\}} \Phi(0, T, W) \quad (5)$$

where $\{W\}$ is the set of correct schedules, T is the final time of system operation (when the last task is finished).

Minimization of function Φ is aimed at:

1. balancing the utilization of CPU and memory among the nodes (the component with γ factor);
2. limiting the number of unscheduled tasks (the component with δ factor).

Relative importance of the sub-goals 1) – 2) is specified by the values of corresponding factors.

3. Related Work

This section contains the survey of machine learning (ML) based solutions for the related dynamic task allocation problems. The solutions are compared by the following criteria: state, action spaces, rewards, kind of architecture (centralized, decentralized or decentralized with centralized critic) , class of neural network, constraints and how to deal with them within machine learning methods.

The paper [7] proposes TapFinger, a distributed scheduler of ML tasks for edge clusters. It minimizes the total completion time by solving the task placement problem and fine-grained multi-resource allocation problem separately. TapFinger is based on multi-agent reinforcement learning (MARL) and includes several techniques to make it efficient, like a heterogeneous graph attention network (HAN) as the MARL backbone, a tailored task selection phase in the actor network, and the integration of Bayes' theorem and masking schemes. The authors propose two schedulers: a single-task scheduler and a multi-task scheduler. The first one can process one task at a time. This scheme is generalized for the multi-task scheduling case, in which a sequence of tasks is scheduled simultaneously. TapFinger's design can mitigate the expanded decision space and yield fast convergence to the optimal scheduling solutions .

Key features: Deep Reinforcement Learning (DRL), Actor-Critic, Multi-Agent (MA), HAN.

Working with constraints: constraints are enforced by the invalid action masking module. Its purpose is to

identify and mask all the invalid actions in the combinatorial action space to prevent actors from predicting invalid resource allocation. Then zero probability is set to all the actions that either exceed the resource capacity or provide less than the minimum required amount of resources. Additionally, the conflict resolution module is used when several edge clusters choose the same task. This module transforms the attention scores in the output of the task selection phase into task-conditioned probabilities using Bayes' theorem and restricts the maximum number of tasks that can be allocated to each edge cluster.

The paper [8] implements the concept of cloud automation to reduce the manual intervention and improve the resource management for large-scale cloud computing workloads. The authors propose four deep and reinforcement learning-based scheduling approaches to automate the process of scheduling large-scale workloads to cloud computing resources, while reducing both the resource consumption and task waiting time. These approaches are: reinforcement learning (RL), deep Q networks (DQN), recurrent neural network long short-term memory (RNN-LSTM) and deep reinforcement learning combined with LSTM (DRL-LSTM).

Key features: DRL, Q-learning, Recurrent neural network (RNN), Long short-term memory (LSTM)

Working with constraints: It is assumed that the resource demands of each task are known upon arrival and for a task to be assigned to a specific VM the necessary amount of resources must be available on this VM. The paper does not provide details on how this constraint is checked.

The paper [9] proposes a MARL scheduling framework to cooperatively learn fine-grained task placement policies, towards the objective of minimizing task completion time. To achieve topology-aware placements, the proposed framework uses hierarchical graph neural networks to encode the data center topology and server architecture. In view of a common lack of precise reward samples corresponding to different placements, a task interference model is further devised to predict interference levels in face of various co-locations, for training of the MARL schedulers.

Key features: DRL, Actor-Critic, Multi-Agent, Hierarchical GNNs with convolutional layers

Working with constraints: Hierarchical GNN: there are two GNNs on each scheduler, one for encoding the inner graph with the constraints of its nodes and one for inter-scheduler graph. Constraints are checked before task allocation. If resources in the partition managed by a scheduler are not sufficient for hosting a task that the scheduler receives, the scheduler will forward the task to another scheduler. Schedulers can exchange their observations of server load status , link bandwidth usage and concurrent task placements. Once a task is placed, it will run to completion without preemption.

The paper [10] introduces KaiS, a learning based scheduling framework for edge-cloud systems to improve the long-term throughput rate of request processing. There are several features of KaiS worth to mention. The first one is a coordinated multi-agent actor-critic algorithm to cater to decentralized request dispatch and dynamic dispatch spaces within the edge cluster. The second one is the use of graph neural networks to embed system state information and combine the embedding results with multiple policy networks to reduce the orchestration dimensionality by stepwise scheduling. Finally, there is a two-time-scale scheduling mechanism to harmonize request dispatch and service orchestration. The paper describes the implementation design of the above algorithms to deploy them in Kubernetes environment.

Key features: DRL, Actor-Critic, Multi-Agent, Centralized Critic, GNN

Working with constraints: To avoid, as much as possible, the situation that an agent dispatches a request to an edge node with insufficient resources, a resource context for each agent is calculated before dispatch. The resource context is a binary vector that is used to filter out invalid dispatch actions.

In edge-based distributed deep learning (DL), the head of a cluster of edge nodes usually schedules all the DL training jobs from the cluster nodes. The cluster head knows and takes in account all the loads of cluster nodes, but the use of centralized scheduling scheme may lead to overload of the cluster head itself. To handle this problem, the paper[11] proposes a MARL system that enables each edge node to schedule its own jobs using RL. To avoid action collision, in which multiple nodes may schedule tasks to the same node and make it overloaded, an approach called Shielded Reinforcement learning based DL training on Edges is proposed. In this approach, each edge node schedules its own jobs using multi-agent RL. The "shield" deployed in a node checks action collisions and provides alternative actions to avoid the collisions. As the central shield node for the entire cluster may become a bottleneck, a decentralized shielding method is proposed, in which different shields are responsible for different regions in the cluster, and they coordinate to avoid action collisions on the region boundaries.

Key features: DRL, Multi-Agent, DNN

Working with constraints: Each cluster has a shield deployed in the cluster head that has high resource capacity. After an edge node makes a scheduling decision for its job, it reports its decision to the shield in its cluster. The shield collects the decisions of all edge nodes in its cluster and checks action collisions, i.e. the actions that make an edge node overloaded by hosting the tasks from multiple edge nodes. The shield then provides alternative actions to avoid the action collisions.

The paper [12] focuses on task offloading to edge servers in the domain of vehicular fog computing. It is noted that an edge server may have high load when a large number of mobile vehicles offload their tasks to it, causing many tasks either to experience long processing times or to be dropped, the latter particularly for latency-sensitive tasks. The authors state that most existing methods are largely

limited to training a model from scratch for new environments. This is because these methods focus more on model structures with fixed input and output sizes, impeding the transfer of trained models across different environments. To solve these problems, the paper [12] proposes a decentralized task offloading method based on transformer and policy decoupling-based multi-agent actor-critic scheme. The paper first introduces a transformer-based long sequence forecasting network for predicting the current and future queuing delay of edge servers to resolve load uncertainty. Second, the actor network is redesigned using transformer-based temporal feature extraction network and policy decoupling network. The feature extraction network can adapt to various input sizes through a transformer that accepts different tokens built from the raw input. The policy decoupling network provides a mapping between the transformer-based embedding features and offloading policies utilizing self-attention mechanism to address various output dimensions.

Key features: DRL, Actor-Critic, Multi-Agent, policy decoupling network, transformer-based temporal feature extraction network

Working with constraints: There is a penalty for every task that misses its deadline, as well as for every violated resource constraint. All penalties sum up to the system cost. Without knowing the offloading decisions of other vehicles and the resource allocation of edge servers, each vehicle minimizes the system cost through locally optimal offloading policies. The sum of all vehicle decisions' costs (i.e. penalties) is defined as the system cost. The optimization goal is to minimize the system cost, thus minimizing the constraints violation.

The paper [13] proposes a new DRL-based task allocation process that allows cooperating agents to act automatically and learn how to communicate with other neighboring agents to allocate tasks and share resources. Through learning capabilities, agents will be able to reason conveniently, generate an appropriate policy and make a good decision. Experiments show that it is possible to allocate tasks using deep Q-learning as a part of distributed task allocation approach.

Key features: DRL, Q-learning, Multi-Agent, DQN

Working with constraints: If the agent to which the task initially arrives cannot fulfill the task resource requirements, it tries to discover neighboring agents with sufficient resources. If there is no success in it, the agent passes the task to another agent, increasing the counter of hops. This next-hop agent tries to do the same (discover / pass) until an agent with enough resources is found, or maximum number of hops is reached (meaning an allocation failure). Partial execution of the task along this path is allowed, so that only the remaining part is passed along.

Conclusion

In most of the considered papers, the solutions relied on graph neural networks. Moreover, in some papers hierarchical GNNs are used, or a two-level approach is implemented (first, agents choose a cluster, and then resources inside cluster). We follow by the same approach in the proposed method and divide the network into several domains. This partitioning scheme is described in the next section.

The survey has shown that there are two approaches to impose constraints on the neural network output values in MARL:

1. Reward penalty if constraints are violated.
2. Setting the probability of invalid action to 0.

We follow the second approach, as it guarantees that all constraints are met.

4. Proposed algorithm for domain-based task allocation

4.1. Overview of the algorithm

In the preliminary step, performed offline, the proposed algorithm partitions the network into domains and selects a designated node for every domain. This is done to find a balance between a fully centralized task allocation scheme (prone to overloads of the decision maker) and a fully distributed scheme in which any node can make allocation decisions (based on very limited information). Agents are located on these designated nodes. Every agent is responsible for its own domain and keeps track of workload allocated to nodes of this domain.

The main algorithm operates online, in the timeline of tasks arrival. The tasks are processed in order of their arrival; if several tasks enter simultaneously, they are processed in some arbitrary order.

For every task j entering the NPC network, following steps are performed:

1. Every agent chooses the candidate node for the task j within this agent's domain. Only nodes that can execute the task without violation of memory and timing constraints (deadlines) can be chosen.
2. Final choice of the node for task j is performed among the nodes proposed in step 1.
3. Transmission and execution of task j is performed:
    - bandwidth for the task j is allocated along the transmission route, which is pre-calculated in step 1;
    - task j is transferred from its original node to the destination node, then the allocated bandwidth is freed;
    - task j is executed on the destination node according to preemptive EDF scheme.

A task becomes unscheduled (added to the set U) in case no suitable nodes are found for it in step 1.

Steps 1 and 2 use neural networks for making the choices. Training of these networks is performed in advance.

4.2. NPC partitioning into domains

The NPC partitioning subproblem is stated as follows:

1. Domains must contain almost equal numbers of NPC nodes (balanced partitioning). This allows even distribution of decision making burden between the agents, and presumably leads to balanced convergence times during the training of neural networks on the domain designated nodes.
2. Interdomain channels number is minimized (cut-edge minimization). This condition simplifies construction of task transfer routes between domains.

Partitioning is performed by METIS algorithm [14], with the number of domains as input, and *ufactor* parameter controlling the allowed imbalance of the numbers of nodes in the obtained partitions.

The designated node within the domain is selected among the border nodes of this domain. A node of a domain is a border node if it has at least one link to the nodes of some other domains. The number of its neighboring nodes from other domains is called *border-degree*. The node with the highest border-degree is selected as domain designated node.

Figure 1 shows a network with two domains, with designated nodes marked red.

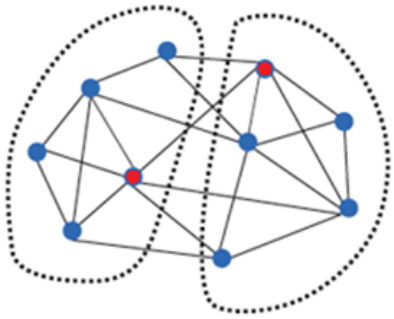

Figure 1: A network with two domains

To estimate the task transmission time and simplify transmission route construction, a transit graph G^T is obtained from the network G:

1. G^T retains the edges, connecting the different domains, and their incident nodes (i.e. border nodes). Edge weights are same as in G.
2. New edges, connecting the border nodes from the same domain, are added to G^T. The weight of such edge is the minimum bandwidth on the maximum capacity path between its nodes inside the domain, constructed in G.

We assume such graph is available on every designated node. Weights of its edges are updated to reflect residual (i.e. free) bandwidth in the network links.

4.3. Candidate node selection in every domain

We propose to select the candidate node for execution of a new task j in a domain using a MARL-based algorithm with proximal policy optimization (PPO). Every agent makes its choice regarding its domain, and acts independently of other agents. In the experimental study, this approach is compared to a greedy heuristic based algorithm.

Our previous research on ML-based traffic engineering [5], as well as the survey of related works on task allocation techniques, shows the advantages of GNN and PPO approaches.

The selection of candidate nodes for a task in every domain is performed by MARL as follows:

1. Task j enters on the NPC node n_j. The task specification $(m_j, \widehat{c_j}, \widehat{s_j}, p_j, q_j)$ is sent to the designated node of the domain the node n_j belongs to. From there it is broadcast to all other domain designated nodes. The time it takes for the specification to reach all domain designated nodes is assumed insignificant.
2. Each domain designated node has the state of all nodes and links within its domain: a) for a link: available and total bandwidth; b) for a node: task schedule, available and total amount of memory and computing resources. Each domain designated node updates the transit graph G^T to reflect the current link bandwidth occupation.
3. Each domain designated node independently chooses one node within its domain that is the most appropriate to process the task j. The node can only be chosen if all the following constraints are met: a) there is a path with non-zero available bandwidth from n_j to the chosen node; b) allocating the task j to the chosen node will not violate the memory and timing constraints of other tasks allocated to this node; c) the task j will be completed within its maximum response time.
4. The output of each domain designated node is one node inside of its domain. If there were no valid nodes, then the output is empty. This output and the chosen node's state (task schedule, available and total amount of memory and computing resources) is sent through the network to the designated node of the domain the node n_j belongs to. The time it takes for this information to reach the domain designated node is considered insignificant.

The neural network is used to choose a node in step 3. Input to the network is:

- Task specification $(m_j, \widehat{c_j}, \widehat{s_j}, p_j, q_j)$.
- State of every domain node: CPU and memory load calculated based on the current node's schedule on the interval $[t_j, T]$, where T is the maximum finish time of the current task j among all the nodes of the domain.

The neural network calculates the relative probabilities of domain nodes to be chosen as candidates, then a random choice is made according to these probabilities. Before performing this choice, probabilities for nodes that do not meet the constraints are set to zero.

The route of task transmission to a given node is the maximum capacity path (MCP) from the entrance node. The MCP is constructed by a Dijkstra-based algorithm [15]. The algorithm is applied to the transit graph G^T, with added nodes and links of the original and destination nodes' domains. Same bandwidth is reserved for all links of the selected route, equal to the minimum available bandwidth on the links of this route.

4.4. Checking the constraints

When a node is checked for being able to execute the newly entered task, both memory and timing constraints are taken in account.

Memory constraints are checked directly by comparing the sum of required storage resources $\widehat{s_j}$ and input data size m_j with the available storage resources s_i on the node.

To check the timing constraints for task j on a node:

1. Time of task j arrival to the considered node is calculated, based on task input data size m_j and bandwidth of the MCP.
2. Preemptive EDF schedule is rebuilt for the node, taking in account the added task j, with its deadline set to $t_j + p_j$.
3. If all tasks in the schedule meet the deadlines, the timing constraints are met, otherwise they are violated and the task j cannot be allocated to this node.

In step 2, the schedule is rebuilt starting from the task j arrival to the considered node, and taking in account only the remaining parts of the other tasks on the node. Note that execution of task j affects the execution of the tasks previously allocated to the node and not yet completed, in case the deadline of j is earlier than deadlines of some of these tasks.

4.5. Final selection of the node

The best node among the node candidates provided by the domain designated nodes is chosen by the same PPO approach, as in algorithm to choose a node candidate in a domain. Since the constraints were already checked by the domain designated nodes, all candidates are valid.

The neural network takes as input the same transit graph and the state of each candidate node (task schedule, available and total amount of memory and computing resources), and evaluates the relative probabilities of the candidate nodes to be finally chosen. Then the final choice is made according to these probabilities.

4.6. Neural network training scheme

The proposed method is based on the actor-critic approach characterized by agent separation into two decision making entities: the actor and the critic. The reason behind this separation is to allow a policy (the probability to take a certain action for an observed state) improvement through an estimation of the state-value function, combining both value-based and policy-improvement algorithms. The critic approximates the state-value function $\hat{V}_s$, while the actor updates and improves a model of the stochastic policy $\hat{\pi}$ by taking into account the critic estimation while maximizing the total expected reward.

The reward for a single applied action is evaluated as the difference in the objective function before and after the current task was placed:

$$\Phi(t, t'|\textit{task was not placed}) - \Phi(t, t'|\textit{task was placed}),$$

where $\Phi(t_1, t_2|X)$ is the value of $\Phi(t_1, t_2)$ calculated for scenario X of tasks placement; t is current time; t' is time when all tasks entered before t would be completed.

This reward is given to all agents present in NPC: ones that select the best node within the domain and the one that selects the final node from domain node candidates.

Training takes place after a large enough history of actions and rewards is accumulated. The interval of accumulation of this data is called an episode. In the proposed algorithm the episode is defined as a complete schedule for a given set of tasks, i.e. the episode starts when the first task enters and ends when the last task is processed.

5. Experimental results

The goal of the experimental research was to evaluate the algorithm according to the following criteria:

1. objective function mean value and mean deviation;
2. number of unscheduled tasks;
3. task placement decision time.

The efficiency of the proposed algorithm is evaluated by criteria 1 and 2 and compared with a heuristic algorithm result and reference value, obtained in a special way described below. The overhead of the proposed algorithm is evaluated by criterion 3 taking into account the neural network operation time and the message exchange time.

5.1. Simulation methodology

This research considers two topologies:

- Topology with 16 nodes (Figure 2). This topology has a good structure with clear domains and its designated nodes. Each color represents individual domain and the nodes with larger size are designated nodes;

- Relaxed-caveman graph with 56 nodes and 392 edges (Figure 3). This type of graph is noteworthy because they represent variations of the graph with "perfect" communities, i. e. disconnected cliques that can be represented by distributed datacenters belonging to different owners.

Domains were constructed by described method in section 4.2 using METIS algorithm.

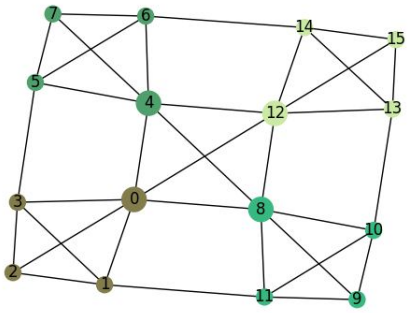

Figure 2: Symmetric topology with 16 nodes

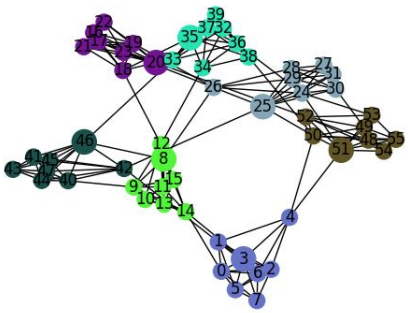

Figure 3: Random topology with 56 nodes

The workload J for experiments is generated by constructing EDF schedule for each NPC node, evenly filling the resources of the NPC network. Then the task specification is assembled from the obtained EDF schedule. This allows us to know the sub-optimal solution beforehand, where all tasks will be distributed across CNs. The value of objective function Φ for this sub-optimal solution will be called the reference value. Also the fraction of available resources to total NPC resources is the parameter of the generator. The simulation was carried out for the value of this parameter being 0.9. Two task sets generated for 16-node topology contained 503 and 1548 tasks. The task set for 56-node topology contained 1248 tasks.

To run the simulation experiments, the algorithm was implemented in Python using tensorflow library (version 2.11).

Every simulation experiment consists of the following steps:

1. Initialize initial EDF as an empty schedule.
2. Get the next task from the workload J.
3. Update NPC resources load based on the new task entering time.
4. Run DoTA algorithm to select CN for execution.
5. Calculate objective function value Φ.
6. If current task is not the last one, go to step 2.

The proposed DoTA method is evaluated by the number of episodes to converge to a sub-optimal solution, the number of allocated tasks, the value of objective function Φ. These values, except for number of episodes for training, are compared to the reference value and the value obtained by the heuristic algorithm.

The heuristic algorithm is a distributed algorithm that runs on each domain designated node. Before the start of the heuristic algorithm, the ring path consisting of all domain designated nodes is determined by solving *travelling salesman problem*. Upon arrival of a new task, available resources information is transmitted along the ring path, and each designated node appends the part related to its domain. Then the domain with the most amount of available resources is selected according to the greedy algorithm.

In the experiments $\gamma = 1, \sigma = 0.1$, however the figures below omit the penalty part of the objective function to show how evenly workload is distributed among NPC nodes. The unscheduled tasks are shown on a separate figure.

5.2. Efficiency estimation results

Figures 4-5 show 1000 and 400 episodes of training for task set sizes of, respectively, 503 and 1541 tasks on 16-node topology. The figures show two components of the objective function - memory deviation from average, computing resource deviation from average - and their sum. The horizontal lines stand for objective function value obtained from the heuristic algorithm and from reference schedule.

The figures show that the proposed algorithm requires 100-200 episodes to reach objective function values around 0.07 which do not improve further. Memory and computing resource deviation from average differ insignificantly from each other, so the algorithm optimizes both of these values equally. For both sets of tasks the proposed algorithm shows better values than the heuristic algorithm. Also, figures 6 and 7 show that the proposed algorithm placed more tasks in both cases.

Figure 8 shows the results of training case of 400 episodes and for the task set size of 1248 on 56-node topology. In this case objective function values stabilize around 0.4 which is nearly the same as the heuristic algorithm.

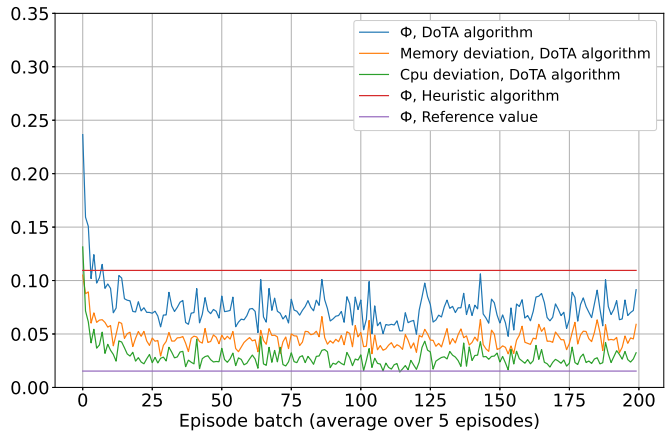

Figure 4: Training performance on 16-node topology and 503 tasks

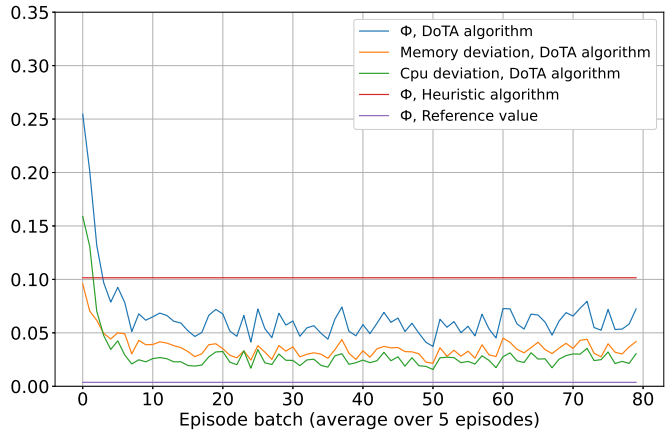

Figure 5: Training performance on 16-node topology and 1541 tasks

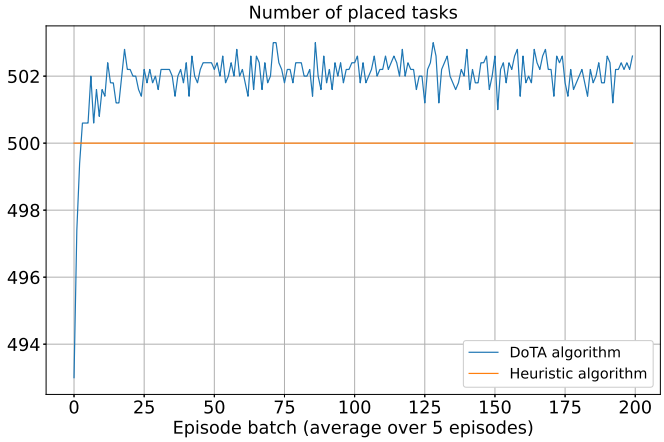

Figure 6: Number of placed tasks on 16-node topology and 503 tasks

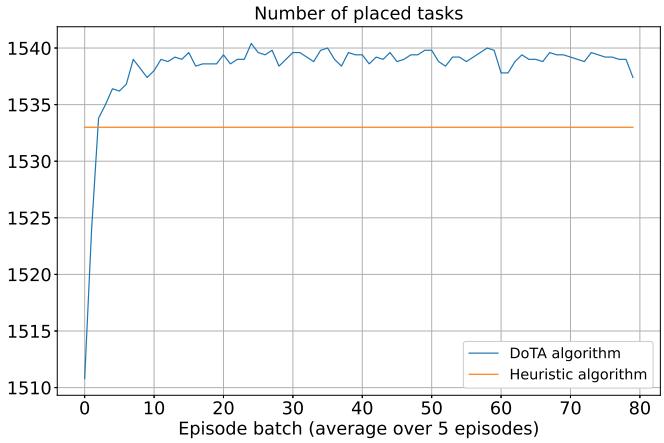

Figure 7: Number of placed tasks on 16-node topology and 1541 tasks

Figures 9 and 10 show aggregated results from 5 runs on the same input data to demonstrate the stability and spread of the values obtained by the algorithm. The figures show that on the 16-node topology after the algorithm reaches a stable objective function value around 0.07 at 100th episode. The objective function value oscillates between 0.04 and 0.12 which is almost in the middle of reference value and heuristic algorithm's value. On the 56-node topology the algorithm improves objective function value until the 300th episode. It has a significantly larger spread, oscillating from 0.2 to 0.6. However its values are still better compared to the heuristic algorithm.

The proposed algorithm shows stable results on small topology which are better than heuristic algorithm's results. However on large topology the advantage over the heuristic algorithm is smaller and the spread is larger. This difference in results can be attributed to a greatly increased size of the valid schedule space with a larger topology, since it scales with both the task set size and the amount of nodes.

5.3. Overhead estimation results

Overhead is estimated by training time and decision making time. The training consists of the gradient calculation and the neural network weight update. Experiments showed that the training time has a linear dependence of a number of tasks therefore the results are averaged for 1 task.

Table 1 presents training time and decision making time of 1 task for different topology and domain sizes. The most important points to note from this table are: training time does not depend on the topology size and is insignificant compared to decision making time because a single task set for large topology can have hundreds of tasks and training happens at the end of the task set. Decision making time of 1 task strongly depends on the size of the domain, but not the whole topology, since the largest jump in execution time happens between large and extra large topology, which goes from about 12 nodes per domain to about 60 nodes per domain.

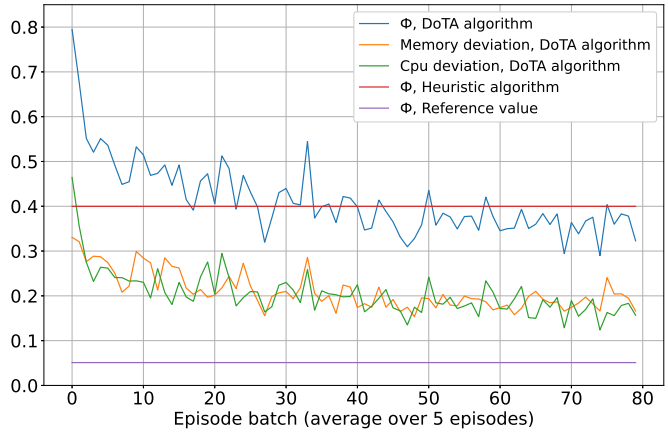

Figure 8: Training performance on 56-node topology and 1248 tasks

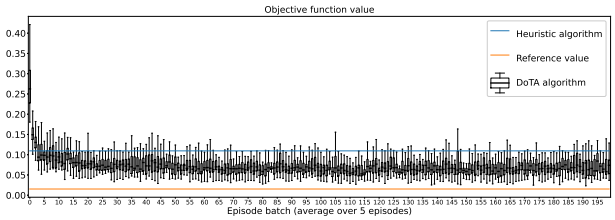

Figure 9: Aggregated results from 5 runs on 16-node topology and 503 tasks

6. Conclusion

The proposed DoTA method is based on the domain approach, where NPC network is divided to the domains with its designated nodes. This approach is designed with preparation for the large-scale networks, where it is hard to maintain and operate the whole network from the single node. It was shown that the proposed multi-agent reinforcement learning method outperforms the heuristic algorithm and gives the results close to the reference value for 16-node and 56-node topologies.

The future research of the proposed algorithm will include the modeling on large-scale topology. It is still open challenge how to model large-scale network of multiple agents, supporting reinforcement learning algorithms.

Also the perspective research direction is to consider the interaction between DPC and DTC planes, that will allow to fine tune not only the load of computing nodes, but also the load of the network.

7. Acknowledgements

Authors would like to thank Garkavy Ivan, Savitskiy Ilia, Tsvetkova Vera and Morozova Veronika, students of Lomonosov Moscow State University for their contribution to the experimental part of this research.

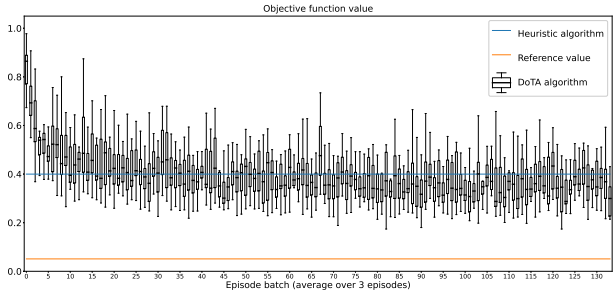

Figure 10: Number of placed tasks on 16-node topology and 1541 tasks

Topology	NN training time	Decision making time (without training)
Smallest (4 nodes, 1 domain)	1.04 ms	1.1 ms
Small (16 nodes, 4 domains)	1.04 ms	1.5 ms
Medium (56 nodes, 7 domains)	1.1 ms	2.5 ms
Large (123 nodes, 10 domains)	1.05 ms	2.7 ms
Extra large (625 nodes, 10 domains)	1.07 ms	6.9 ms

Table 1: Training time and decision making time of 1 task for DoTA algorithm

References

[1] R. Smeliansky, "Network Powered by Computing," 2022 International Conference on Modern Network Technologies (MoNeTec), 2022, pp. 1-5, DOI: 10.1109/MoNeTec55448.2022.9960771

[2] Yukun Sun, Bo Lei, Junlin Liu, Haonan Huang, Xing Zhang, Jing Peng, Wenbo Wang Computing Power Network: A Survey. arXiv:2210.06080

[3] L. Geng and P.Willis, "Compute First Networking (CFN) scenarios and requirements," in IETF RTGWG Working Group, 2019.

[4] R. Smelyanskiy Network Powered by Computing: Next Generation of Computational Infrastructure. In Edge Computing - Technology, Management and Integration (ISBN 978-1-83768-862-3)

[5] E. Stepanov et al. "On Fair Traffic allocation and Efficient Utilization of Network Resources based on MARL." Preliminary on ResearchGate, Available from: `https://www.researchgate.net/publication/371166584_On_Fair_Traffic_allocation_and_Efficient_Utilization_of_Network_Resources_based_on_MARL`

[6] W. A. Horn. "Some simple scheduling algorithms." Naval Research Logistics Quarterly. 1974. P. 177-185.

[7] Li, Yihong, et al. "Task Placement and Resource Allocation for Edge Machine Learning: A GNN-based Multi-Agent Reinforcement Learning Paradigm." arXiv preprint arXiv:2302.00571 (2023).

[8] Rjoub, Gaith, et al. "Deep and reinforcement learning for automated task scheduling in large-scale cloud computing systems." Concurrency and Computation: Practice and Experience 33.23 (2021): e5919.

[9] Zhao, Xiaoyang, and Chuan Wu. "Large-scale Machine Learning Cluster Scheduling via Multi-agent Graph Reinforcement Learning." IEEE Transactions on Network and Service Management (2021).
[10] Han, Yiwen, et al. "Tailored learning-based scheduling for kubernetes-oriented edge-cloud system." IEEE INFOCOM 2021-IEEE Conference on Computer Communications. IEEE, 2021.
[11] Sen, Tanmoy, and Haiying Shen. "Distributed Training for Deep Learning Models On An Edge Computing Network Using Shielded Reinforcement Learning." 2022 IEEE 42nd International Conference on Distributed Computing Systems (ICDCS). IEEE, 2022.
[12] Gao, Zhen, Lei Yang, and Yu Dai. "Fast Adaptive Task Offloading and Resource Allocation via Multi-agent Reinforcement Learning in Heterogeneous Vehicular Fog Computing." IEEE Internet of Things Journal (2022).
[13] Noureddine, Dhouha Ben, Atef Gharbi, and Samir Ben Ahmed. "Multi-agent Deep Reinforcement Learning for Task Allocation in Dynamic Environment." ICSOFT. 2017.
[14] G. Karypis and V. Kumar. Multilevel k-way partitioning scheme for irregular graphs. Journal of Parallel and Distributed Computing, 48(1): 96–129, 1998.
[15] Punnen A.P. "A linear time algorithm for the maximum capacity path problem." European Journal of Operational Research. Vol. 53, No. 3. 1991. P. 402-404.