

# We are IntechOpen, the world's leading publisher of Open Access books Built by scientists, for scientists

6,300

Open access books available

171,000

International authors and editors

190M

Downloads

Our authors are among the

154

Countries delivered to

TOP 1%

most cited scientists

12.2%

Contributors from top 500 universities



WEB OF SCIENCE™

Selection of our books indexed in the Book Citation Index  
in Web of Science™ Core Collection (BKCI)

Interested in publishing with us?  
Contact [book.department@intechopen.com](mailto:book.department@intechopen.com)

Numbers displayed above are based on latest data collected.  
For more information visit [www.intechopen.com](http://www.intechopen.com)



Chapter

# Network Powered by Computing: Next Generation of Computational Infrastructure

*Ruslan Smeliansky*

## Abstract

This paper is an extended version of my talk on the MoNeTec-2022. It gives a detailed presentation of the concept Network Powered by Computing. The main differences from the previously published one are that the functional architecture of the NPC is presented, the main problems on the way to its implementation are formulated, the mathematical statements of the problems of control and management of the resources in the NPC environment by methods of multi-agent optimization are given, the existence of a solution to these problems is justified, and the relationship between the problem of control in such an infrastructure and the Barabási-Albert model is shown. An example of the predicting execution time of services in the NPC environment is given.

**Keywords:** cloud computing, edge computing, software-defined network, network function virtualization, software defined wide area network, green computing, zero footprint

## 1. Introduction

MIT professors J.L. Hennessy and D.A. Patterson in their 2018 Turing Award lecture gave an excellent overview of the history of computer architecture and the lessons of this history in Ref. [1]. In the conclusion of this paper, they wrote: “The next decade will see a Cambrian explosion of novel computer architectures, meaning exciting times for computer architects in academia and in industry.” This “explosion” brings big opportunities for computational infrastructure. To paraphrase the title of their paper, we can say that we are going through “A New Golden Age for Computational Infrastructure.” The term computational infrastructure will be treated here as the architecture of computational infrastructure.<sup>1</sup> Organization of computations is one of the pillars of human civilization. Therefore, it is important to understand the main trends and prospects of its development, to understand what problems will need to be solved.

---

<sup>1</sup> This paper is extended version of my talk on the MoNeTec-2022 published in [2].

If we have a look at the history of the progress of computing infrastructure from the individual use of the first computers, the packaged organized computation on stand-alone computers to the client-server paradigm of computation based on high-speed data communication networks and large-scale (giant-like) data centers (DC), then the main lesson of this history is that the main drivers of this progress were the requirements of applications. Nowadays, the computational infrastructure paradigm is moving from “build your own” to the new one—“consume as a service” where a business does not need to buy and develop its own computational infrastructure, rent channels that connect it to the public network, hire expensive professionals for system and network administration, and so on. In the new paradigm, one can request and get the resources and services they need based on the model of “pay as you go.” Also, computing paradigm based on the giant-like DC is being replaced by a new one—small cloud edges in [3]. Our applications became more and more real-time applications. So, the time for communication between user terminal and large scale DC where our ability to control communication delays became more and more critical for application operation. The increased restrictions on the interaction time between the application and the terminal device led to a contradiction with the concept of computing based on large-scale DC. Carbon footprint of such organized computation is a heavy burden on our ecology.

The necessity of this change has come from the requirements of new applications with their real-time interactivity, video streaming, and 5G communication. Over the past 10 years, cloud computing as the computing paradigm has completely changed the landscape of computational infrastructure; for example, see [4–7]. In the paper [3], I wrote: “It significantly contributed to the growth of both the number of data processing centers (DCs) and their size, the increase in throughput capacity of backbone channels [5], the increase in equipment density: virtualization of IT equipment in cloud architectures allowed to fit into one rack what previously required 10 racks improving and developing the capabilities of personal gadgets, various types and uses of sensors, the development of data transmission technologies such as OTN, 5G networks, network convergence, the emergence of SDN and NFV technologies gave impetus to the development of a large number of real-time applications (RT applications) in Ref. [8]. Here are just some examples of such applications: smart city, smart home, healthcare (especially its areas such as surgery, telemedicine, emergency cardiology), interactive games, training, augmented reality, agriculture, infrastructure for scientific multidisciplinary research in [7], social communications, energy management systems (smart grid), wireless sensors embedded in a variety of robots, monitoring and control of transportation systems and facilities, assembly lines and production lines, gas and oil pipelines.

An important aspect of the computing infrastructure is power consumption. According to 451 Research, a technology research group within S&P Global Market Intelligence that provides a holistic view of innovation across the entire enterprise IT landscape, the computing power as well as the engineering equipment of all DC worldwide is estimated in 2022 about 200 GW in [11]. This means that in 2022 the energy consumption of all DC was  $200 \text{ GW} \times 24 \text{ hours} \times 365 \text{ days} = 1,752 \cdot 10^{12} \text{ Wt hours}$ ! Thus, the carbon footprint of the contribution of computational infrastructure was about  $1.8 \cdot 10^{15} \text{ W/year}$  (53% of the US). This means that the organization of the computing infrastructure has a significant impact on the ecology of our environment.

## 2. Properties suite of modern applications

In my talk, I emphasized: “the main driving force of the development of infrastructure for computing, its operating environment, programming languages and tools have always been the needs of applications” [2]. The set of these needs can be summarized as follows: distribution, self-sufficiency, real-time, elasticity, cross-platform, interaction and synchronization, and update-friendly. The definitions of these terms can be found in [2]. Nevertheless, for a better understanding of the further text, it is necessary to explain the term self-sufficiency, which is defined in [2] as: “application is no longer represented by code and source data only, it is accompanied by a description of the structure of the interconnection of the components (hereinafter application services) that make up the application, setting the required level of their productivity, explicitly formulated requirements for computing and network resources, data storage and access to data resources, intended timeframes for computing and data transmission in the form of a service level agreement (SLA), application launching procedures. This description is written in a special language, an example of which one can be the TOSCA language [6] (hereinafter such a description will be called Application Operation Specification—AOS)” [2].

## 3. Computational infrastructure requirements

Here, let me briefly list the requirements for the computational infrastructure described in [2]: behavior predictability, security, availability, reliability and fault tolerance, efficiency and fairness, virtualization, scalability, and serverless. The serverless was defined there [6] as follows: “[T]he infrastructure should automatically place application components in a way that allows them to interact according to the application structure, and in a way that ensures that the SLA requirements of the application are met, while minimizing infrastructure resources utilization”.

In order for NPC to be able to meet the requirement of efficiency and predictability and serve as the computational infrastructure for applications, in the above sense (everywhere below the app.), its behavior and functioning must meet the requirements such as:

- “predictability of time of execution of application components and their interaction time (data transfer) according to AOS;
- predictability of the characteristics of data transfer between application components along overlay channels;
- availability of a variety of virtualized network functions (VNF hereinafter) and other traffic engineering (TE) methods on DTN channels based on machine learning algorithms for distribution, balancing, shaping, filtering to control, and manage QoS of an overlay channel (further channel);
- reliable isolation of control plane and data plane in DTN from errors in network equipment, as well as isolation of different data flows, malicious influences in these planes” [2].

For predictability of the characteristics of data transfer between application components, it is necessary to:

- “set and guarantee variation ranges of end-to-end delay and jitter in DTN;
- guarantee the probability of packet loss in the DTN at the level corresponding to the SLA application;
- make the usability of the available bandwidth of DTN channels be maximal (mass overuse of resources is prohibited, such as flooding, broadcasting);
- exclude the unpredictable transmission delays caused by DTN, such as packet delays due to failure of order, retransmission, overload feedback, etc.;
- predict how much time will take the execution of certain application service on some computer installation to meet application SLA requirements” [2].

Techniques and methods for predicting the execution time of services and applications will be considered later.

#### **4. NPC functional architecture**

NPC functional architecture was presented in my talk [2]. Let me briefly repeat the main statements from there: “The computational infrastructure with properties above, we will call Network Powered by Computing (NPC)—it is a software-driven infrastructure, which is a tight software-driven integration of various computer installations with a high-speed DTN. Such an NPC is a fully manageable, programmable, virtualized infrastructure. In other words, the NPC becomes Computer!

The NPC organization should be based on the federative principle. Each federate has its own administration and possess an independent authority in whose jurisdiction there is a certain amount of computing, telecommunication, storage resources. Federate transfers part of these resources to the Federation authority, which forms and monitors a unified policy for their use.

Here is a summary of what a functional NPC architecture should look like. The core layers are the Application, Application Services, and Network Functions (ASNF) layer, the NPC Infrastructure Control (NPCIC) layer, NPC computational, networking, storage resources (NPCR) layer, and the E2E Orchestration, Administration, and Management (OAM) layer, responsible for organizing, administering and managing the NPC infrastructure.

The ASNF layer is responsible for application representation development: it’s code and it’s data, it’s Application Operation Specification (AOS) representing application services (AS) and virtualized network functions (VNF) necessary for the operation of the application, specification of the data transmission network between application components. Specification of a network should include the topology for data transmission between application components, the requirements to the quality of service of the channels (QoS) that should include such characteristics as available bandwidth, admissible delay, admissible probability of packet loss, admissible jitter variation range. Based on this information and on SLA for the application as a whole ASNF calculates SLAs for each AS” [2].

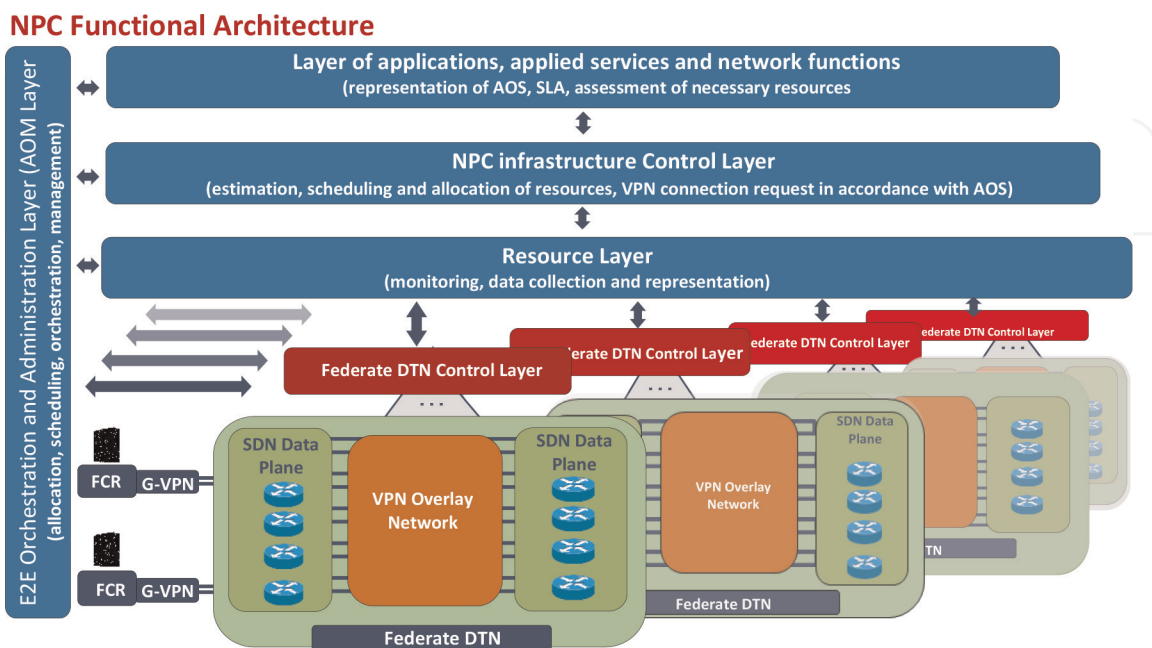
NPCIC functionality provides scheduling and assignment of the application components to NPC resources in accordance with AOS and predicted time for computing, for data transfer, determination, collection, and aggregation of resources as necessary to comply with QoS resource requirements in accordance with the SLA of the application based on the current state of the resources, creating an overlay network according to AOS (topology, QoS channels, and security management). It is at this level that it is determined which VNFs will be required and where in the data transmission network.

The NPCR is responsible for a unified representation of the state of heterogeneous NPC resources, monitoring their current states, and predicting their states for the nearest future. The last functionality is the cornerstone to make NCP behavior be predictable.

The OAM layer orchestrates interactions of the application components in accordance with the AOS, collects resources consumption data for every application components, and manages the security and administration of NPC.

The basis for building NPC is formed by the technologies of software-defined networks (SDN) and network functions virtualization (NFV). Taking into account that the scaling range of network functions is wide and works in real time, the NPC will require low time complexity algorithms to optimize resource scheduling and resource allocation. And given the operation speed, as for data transmission networks and for computer installations, it becomes clear that only suboptimal solutions to the emerging optimization problems will be available based on ML methods. The functional architecture of the NPC described above is shown in **Figure 1** below.

Now, briefly consider the interaction components of NPC functional architecture. AOS can have two types of components: network functions (VNF) for managing data flows (traffic engineering) and application services for data processing and computational services (like modeling, simulation, etc.). The first type of components (network functions) is placed either in the DTN control plane (applications of SDN controller) or directly in network devices. Examples of VNF are NAT, Firewall, BRASS, balancers, shapers, and so on. Components of the second type—application



**Figure 1.** Functional architecture of network powered by computing [2].

services—are placed in a virtualized form (on virtual machines or containers) or directly on computing resources (servers, edges, data centers (DC), and HPC installations) of federates (in **Figure 1** above, they are shown as FCR in the form of racks).

In [2], I emphasize that the application programmer is not required to foresee and explicitly insert the necessary VNFs or their chains into his application. VNFs can be automatically integrated into AOS applications by means of NPCIC, just as compilers or application libraries do: “plug in” the necessary functionality into the application code. NPCIC can do this based on the monitoring data of available channels QoS supplied by NPCR. Based on this data, NPCIC has also calculated the routs for the application with the given AOS.

As it can be seen in **Figure 1** that DTN of NPC may consist in the form of several local DTN of different federates. Each local DTN is SD-WAN overlay with control plane and data plane. In control plane of every local DTN, it is assumed that there are controllers with the set of network applications, channels QoS control and monitoring block, network security center with PKI management, and distributed ledger (DL) of overlay tunnels (chain of overlay channels).

When AOS application services interact through the DTN, the SDN controller of the overlay network “catches” the request for data transfer, accessing DL of the overlay network tunnels to find the proper tunnel. If there is no one there, then the SD-WAN controllers apply to the control center for application services in the NPCIC layer. If the DL does not have the required tunnel or its validity period has expired, then the SD-WAN controllers from the control plane apply to the NPCIC layer that, in cooperation with E2E OAM Orchestrator (end-to-end orchestrator), forms the data necessary for SD-WAN controllers to build the desired tunnel.

As mentioned in [2]: “NPC can run applications in three modes—pro-active, active, and mixed. In pro-active mode, application services are loaded in advance on the federate resources in passive state (in the form of code and data on external or internal memory). In this case, when running the application, it is only necessary to activate the required application services according to AOS on those computing resources that will ensure SLA compliance with the specific call to a specific application. It should be clear from the above that different references to the same application may have different SLAs. Same application but with different SLA is treated as different one.

The active mode involves loading the code and data of the necessary application/ computing services in accordance with the AOS on the computing resources of the federates on demand in such a way as to ensure compliance with the SLA of the application. Mixed mode involves a combination of proactive and active modes, i.e. some of the application services and network functions are already pre-installed and are only being activated, the rest ones are loaded and activated upon request”.

## **5. The main problem statements**

In [2], there is a list of problems that NPC infrastructure should provide solutions to like the method of distribution (distributed vs. centralized) of computing resources between flows of computing/application services and request to them in a given NPC mode of operation (proactive, active, or mixed), method for optimal control of data flows in the interaction of computing/application services, and method for managing resource monitoring, prediction of the state of overlay network channels, selection of the optimal overlay network channel with the best QoS to meet the requirements of the SLA of the application, congestion control management, minimization of end-to-

end delay, bandwidth monitoring, scheduling flows in queues and so on, scaling the NPC control plane and the data plane when changing the NPC scale, optimal channel routing of the overlay DTN, fair distribution of channel bandwidth, balancing data flow between computing/application services, and allocation of channel bandwidth on demand [8]. In the NPCR layer, it is necessary to choose the resource state scan frequency, data presentation format, and so on.

Here, the statements for the most important problems are considered:

1. Optimal distribution of the given set of application services on the NPC resources;
2. Distribution of a given set of services for proactive mode of operation of the NPC.

First of all, let us define the formal model of NPC.

Denote:

NPC as  $\Gamma = (V, A)$ , where

$V = CN \cup SN \cup P$ , where

$CN$ —set of NPC computational nodes,

$SN$ —set of NPC VPN gateways,

$P$ —set of  $\Gamma$  poles.

$A = \{(v_i, v_j) \mid v_i, v_j \in V\}$ —multiple channels of overlay network.

$Q(l_{vi,vj}, \Delta t) = (B, D, L, J)$  is the function defined on  $A$ , where.

$\Delta t$  — interval of time;

$B = (\hat{b}, \bar{b})$  —  $\hat{b}, \bar{b}$  band width of  $l_{vi,vj}$  in terms:  $\hat{b}$ —average and  $\bar{b}$ —maximum on  $\Delta t$ ;

$D = (\underline{d}, \hat{d}, \bar{d})$ —delay on  $\Delta t$  in terms:  $\underline{d}$ — minimum,  $\hat{d}$  —average,  $\bar{d}$ —maximum

RTT;

$L$ —percentage of lost packets;

$J = (\hat{j}, \bar{j})$ —jitter  $\Delta t$  in terms:  $\hat{j}$ —average,  $\bar{j}$ —maximum.

$CN = \{cn_i = \langle cr, m, h \rangle\}$ , where  $cr, m, h \in N$ —set of integers.

$cr$ —number of cores (possible with characteristics);

$m$ —amount of RAM;

$h$ —storage size.

$P = \{p_i\}$ , where  $p_i$ —source of requests/applications flows, which is characterized by the function of distributing the probability of requests/applications each with its SLA. Consider that the same request/application but with different SLAs are different requests/applications. The same  $p_i$  can be a source of requests for different applications. Each request is characterized by an application ID and a specific SLA—execution time plus result delivery time.

$AS$ —set of application services, each characterized by the required computing resources, memory resources, and storage resources ( $cr, m, h$ ).

$VNF$ —set of virtual network functions with required resources (computing, memory, and storage)—( $cr, m, and h$ ) presented in AOS.

Here, for simplicity, we suppose that one application service/network function is always allocated per one  $cn_i$  and will consider an application as a chain of application services from  $AS$ . However, in general, it could be a directed acyclic graph (DAG). So, chain of application services (ASC) is  $W = \{w_k = (s^k_1, \dots, s^k_l)\}$ , where  $s^i_j \in AS \cup VNF, s^i_j = \langle cr, m, h \rangle$ .



Denote the function  $ET: (AS \cup VNF) \times CN \rightarrow R$ , where  $R$  – set of rational numbers. We will interpret  $ET$  as the estimation of execution time element from  $AS \cup VNF$  on the certain  $cn_i \in CN$ . This function can be represented as a matrix, where columns correspond to elements from  $AS \cup VNF$  and rows correspond to elements from  $CN$ ; entries are execution times + data transfer time to the next  $s_j^i$  along the chosen channel. We will consider how to build this function in the next section.

In these terms, the problem of the optimal distribution of SFC on NPC can be formulated as follows:

Construct the mapping  $F: W \rightarrow \Gamma$  for a given set  $P$  in such a way that

1. Meets the SLA requirements for all  $w_i$  from  $W$ ;
2. Under the condition of minimizing the objective function, for example, in the following form:

$$F = \min \sum_1^{|CN|} \left[ \alpha \frac{\bar{c}_i}{c_i} + \beta \frac{\bar{h}_i}{h_i} + \gamma \left( \left( \frac{\bar{c}_i}{c_i} - \Theta \right)^2 + \left( \frac{\bar{h}_i}{h_i} - \Delta \right)^2 \right) \right] \quad (1)$$

where

$\alpha, \beta, \gamma$ —constant values;

$c_i, h_i$ — $cn_i$  resources are used;

$\bar{c}_i, \bar{h}_i$ — $cn_i$  resources and queue length averaged over usage time;

$\Theta, \Delta$  —the entire NPC resource usage averaged over time.

The  $\alpha, \beta, \gamma$  values are the subject of adjustment of the application services allocation control. It is required to find the distribution of the AOS component  $w_k \in W$  in such a way as to minimize the objective function  $F$  from Eq. (1), that is, in the table representing the  $ET$  function, one needs to add the application service ID in those positions that correspond to the appropriate resources.  $F$  from Eq. (1) gives us the set  $\{cn_i\}_w$  of which it is necessary to select only those that are connected by channels that form a path in NPC corresponding to SLA ( $w_k$ ).

The problem of distribution of application services over the resources of the NPC in proactive mode can be stated as: for given  $\Gamma, AS, W$ , and  $P$ , it is required to build a matrix  $X$  with dimension  $|X| = |AS| \times |CN|$  where  $x_{ij} = 1$ ; if  $s_i$  can be located on  $cn_j$ , it is subject to the following conditions:

1. The constraints of none  $cn_j$  and none of  $(v_i, v_j) \in A$ , incident to  $cn_j$ , from  $\Gamma$  are violated;
2.  $\forall w_i \in W$ , SLA applications always be met for any  $p_j \in P$ .

It is clear that for both problems, we have first of all proof for the existence of the solutions. Consider solving the problem of placing a chain of services in a NPC. It is divided into three subtasks:

1. Construct a set of all possible placements  $s_j^k$  for a given  $w_k$  on  $cn$  nodes from  $CN$ ;
2. Select only those placements that meet the requirements of the SLA of the application and the limits on the resources of the  $cn$  nodes;

3. Among  $cn$  nodes selected at step 2, choose only those for which the NPC topology contains a route on which the selected  $cn$  nodes are ordered according to the order on  $w_k$ , that is, if  $s_j^k$  was assigned to  $cn_i$  and  $s_{j+1}^k$  to  $cn_r$ , then  $(cn_i, cn_r) \in A$ , which provides the necessary parameters for data transfer between  $cn_i, cn_r$ .

Since the length  $w_i$ , the number of computational nodes in CN, is limited, the problem certainly has a solution. You just need to go through a finite number of combinations. Let us estimate the space of possible solutions for some chain  $w$  from  $W$ . Let the length of any  $w_k \in W$  not exceed  $N$ , where  $N = |CN|$  and  $|w_k| = k \leq N$ . Then, the number of application service locations for  $w_k$  equals  $\sum_{i=0}^{k+1} C_k^i = 2^k$ . The number of possible placements of these  $2^k$  substrings over NPC can be estimated by the following expression:  $\sum_{i=0}^{k+1} C_N^i C_k^i$ . This expression can be evaluated under the condition that the lengths  $w_k \in W$  are uniformly distributed on  $[1, N]$ , as follows:

$$\sum_{i=0}^{l+1} C_N^i C_l^i < \sum_{i=0}^{l+1} 2^l C_l^i = 2^l 2^N \approx 2^{\frac{3N}{2}}. \quad (2)$$

Even  $N \cong 100$  gives us the estimation of this expression  $2^{150} > 10^{100}$  (googol) options. This estimation must also be multiplied by  $|W|$ . The problem under consideration has a solution due to the finiteness of the number of options. If we recall that we are considering the simplest case, when the application is a chain of services, then it should be clear that the space of possible options in the case when the application is a directed acyclic graph (DAG) will be multiply increase: in this case, the estimation of (Eq. (2)) will need to be multiplied by the number of paths in this DAG. It is clear that classical mathematical optimization methods will not meet time restrictions to solve the problems. It is natural to consider the solution in the direction of splitting the solution space on domains and searching the solution in parallel in every domain based on ML technics.

The urgency to use ML methods is also argued by the following reasons. The NPC model described above is actually two random graphs. One is formed by data flows between the chain of application services that arise as a result of the action of poles from the set  $P$ . The vertices of this graph, let us call it an information flow graph (IFG)—application services, arcs—data flows between them. Both the first and second are of a stochastic nature and are determined by random processes initiated by poles. The second graph is the topology of the NPC network. This is also a random graph. Its dynamics are determined by the availability of resources of NPC, the failure dynamic of which is a set of random processes. The control in this model is the mapping of IFG graph to NPC graph. The optimality of control in this model is such a mapping of the IFG graph to the NPC graph, in which the lifetime of the information chain is minimal, subject to all restrictions on NPC resources.

Please pay attention that in the IFG graph, the number of the nodes is not fixed. It is a scale-free graph in terms of the Albert-Laslo Barabashi model from [10]. If NPC accepts the mobile edges, then NPC graph is also a scale-free graph. In this case, it seems that this model is very similar to the Barabási-Albert model described in [11]. Here and there, we have random scale-free graphs' interaction. However, the properties of such models in a computer world have never been investigated. The applicability and adequacy of such models in relation to computer networks require awareness and research.

The most appropriate mathematical technique for optimal control seems to be suitable for that is the multi-agent optimization (MA) technique. There are several approaches to MA optimization. **Centralized approach** assumes the presence of a control center and that agents are deployed one per node. Each agent forms local state-status of its node. The control center collects the status of each agent, makes a decision based on the optimization policy, and sends each agent a management impact. Another possible MA optimization approach induced by the NPC graph structure. The structure (topology) is divided into domains and agents that exchange information about their state only within the domain. The agents belong to the same domain called **neighboring** (interconnected) **agents**. In this case, each agent knows its local state and states of its neighbors. Information exchange is limited by neighboring agents only. Based on local and neighbor-based information, each agent decides on the optimal strategy. The way of topology dividing into domains has great importance when using the MA approach in control. Experiments with the usage of MA optimization for the routing have shown that by adjusting the domain size, it is possible to achieve the optimal combination of convergence and quality optimum solution of the routing problem.

The third option—**independent agents**. Each agent knows its local state. Each agent judges the control strategy and actions of other agents based on its experience. The agent implements control decisions in accordance with its local optimization strategy and based on its observations.

Thus, the choice of approach to MA optimization is another challenge for the problem under consideration. At the same time, it is important to bear in mind that:

- a. there are no mathematical models that guarantee convergence to the optimal solution;
- b. the constraint of the deviation from the optimal solution is not guaranteed.

## 6. Application services execution time prediction

### 6.1 Problem statement

The application services execution time prediction problem can be described in the following way. Let we know  $ID(cn)$ —ID for every computational node  $cn$ , for every  $w_k$  and every  $s_j^k$  it is known the set of  $cn$  computing installations identifiers (IDs), a set of variants of the  $s_j^k$  parameters (for more details see below), the amount of resources requested for known executions of the  $s_j^k$  with its parameter variants on certain  $cn$  node, and the execution time of this  $s_j^k$  with its input data variant and on some of  $cn \in N$ . Neither the source code of the program nor its binary code or the architecture of computer installations are unknown. We emphasize that only the  $s_j^k$  execution time is known on some  $cn$  nodes but not for each. Further, the application service will be treated simply as a program; for the sake of simplicity of terminology, we will use the term program and denote  $P_i$ . Because the set of  $s_j^k$  is limited ( $|W|$  is finite, any  $|w_k|$  is finite), there is a numeration that for every  $s_j^k$  defines the unique index  $i$ .

We will use the following notations for the problem statement:

1.  $P_i$ —unique program ID;

2.  $\{P_1, P_2, \dots, P_N\} = \{P_i\}$ —the set of IDs of programs under consideration;
3.  $\{Arg^{(i)}_1, Arg^{(i)}_2, \dots, Arg^{(i)}_{A_i}\} = \{Arg^{(i)}_j\}_{j=1}^{A_i}$ —the set where each  $Arg^{(i)}_j$  is a set of the values of input parameters program  $P_i$ : amount of input data and program environment parameters (number of processes and number of threads). We will call this set as **arguments** of  $P_i$ .
4.  $[(P_i, Arg^{(i)}_1), (P_i, Arg^{(i)}_2), (P_i, Arg^{(i)}_2) \dots, (P_i, Arg^{(i)}_{A_i}), (P_i, Arg^{(i)}_1)]$ —a history of  $P_i$ ;
5.  $cn_i$ —unique computational node ID;
6.  $\{cn_1, cn_2, \dots, cn_N\} = \{cn_i\}_{i=1}^N$ —a set of  $M$  unique IDs of computer installations;

By the notions above, the problem statement can be specified as following:

- **Given**

$\{P_i\}_{i=1}^N$ — $N$  program IDs;  
 $\{Arg^{(i)}_j\}_{j=1}^{A_i}$ —the set of arguments  $P_i$  program;  
 $\{cn_i\}_{i=1}^N$ —computational node IDs;  
 $V = \{(P_i, Arg^{(i)}_j), cn_k\}$ , where  $i = 1, M, j = 1, A_i, k = 1, N$  and  $|V| = (\sum_{i=1}^M A_i) \cdot N$ ;  
 $T(v)$  where  $v$  is the partial defined function on  $V$ . The values of  $T(v)$  is the execution time the program  $P_i$  with arguments  $Arg^{(i)}_j$  on computational node  $cn_k$ .

- **Required**

Redefine the values of the function  $T(v)$  at the undefined points of  $V$ .

The problem of estimating the execution time of a program on a computer is a classic problem that has been known since the 1960s. The problem still exists in many forms, for example, for worst-case execution time estimation and for different computer architectures [12–15]. Different ways for this problem were proposed as analytical [16] and statistical [17], based on program behavior analysis [18], time series prediction [19], and neural networks [20]. Execution time can be predicted from test runs [21]. All algorithms predicting program execution time mentioned above use the history of program executions. Their main drawback is that all of them are applicable only when the histories of program execution are known for the certain computer installation; that is, to predict the execution time on a certain computer installation, there is the need to know the whole history of the program executions on this computer installation.

As we will demonstrate below, to predict the program execution time on some computer nodes, just some running histories of this program on them are sufficient. There is no need to know each program run on each computational node. The accuracy of the prediction depends on the number of program running histories on computer installations from a certain set. To solve the execution time prediction problem, the following technique was used:

1. Let us represent the information about  $V$  set and function  $T(v)$  as a matrix where each row corresponds to pair  $(P_i, Arg^{(i)}_j)$  and each column to the computational node  $cn_k$ . The intersection of row and column is the execution

time of the corresponding program  $P_i$  with parameters  $Arg^{(i)}_j$  on  $cn_k$ . We will denote such matrix as  $PC$ ;

2. Form  $PC^*$  matrix (matrix closure) by the information about representative set of computer installations and the histories of execution of various programs on these computer installations. By this way, we will get everywhere defined function  $T^*(v)$  on  $V = \{(P_i, Arg^{(i)}_j), cn_k\}$ , where  $i = 1, M, j = 1, Ai, k = 1, N$ ;
3. Delete an arbitrary number of entries from  $PC$  to get partially defined  $T(v)$  on the set  $V$ . Obtain thinned matrix— $PC$  matrix.
4. Apply the developed prediction algorithm to redefine the values of the function  $T(v)$  at uncertain points. This algorithm defines function  $Or(v)$  in each point of the set  $V$ .  $Or(v)$  coincides with the function  $T(v)$  where it is defined and at other points, defines the predicted execution time.
5. Estimate the quality of the results of the developed prediction algorithm by the following metric:

$$PredictionError(P_i, Arg^{(i)}_j, C_k) = (|predict - target|/target) \quad (3)$$

where  $predict$  is predicted time if program  $P_i$  with parameters  $Arg^{(i)}_j$  on  $cn_k$ ,  $target$  is a true execution time of program  $P_i$  with the same parameters. The total prediction error is calculated as an average of the errors calculated using the Eq. (3) for all programs.

To form the  $PC$  matrix, the datasets from the website [22] dated 8 June 2021 (the datasets on this website are periodically updated) was used. These datasets contain the description of total amount of resources of numerous computer installations and the results of executions of programs with various input parameters. From this site, we took three datasets with execution results of programs as MPI as OpenMP on a wide range of computers. These programs cover numerous application areas [23].

Naturally, the question arises: why, when developing the method for estimating the execution time, MPI programs were taken? The fact is that this class of programs is used primarily on supercomputers. It is well known that the execution time of a supercomputer program is very dependent on its architecture. Therefore, if we manage to develop a time estimation method for this case, then for calculators used in traditional servers, it will certainly be no worse.

The brief description of the selected datasets are presented in **Table 1**.

The used data was uploaded on github in [24] along with the developed algorithm.

Name	Number of computer installations	Number of programs	Type
MPIL2007	180	12	MPI
MPIM2007	437	13	MPI
ACCEL OMP	30	15	OpenMP

**Table 1.**

Datasets of MPI and OpenMP programs executions on various computer installations from [22].

It has been recognized that the considered problem is very similar to the problems solved in recommender systems, in which matrix decomposition algorithms as in [25] are very widely used in various combinations. The proposed solutions were developed as a combination of several algorithms. They are run in parallel with the same PC matrix, each of which makes its own prediction. At the end, all predictions are averaged. The resulting value is considered the expected execution time of the program. This technique is called ensemble averaging in [26]. Three algorithms were chosen: ridge regression, Pearson correlation, and matrix decomposing.

1. Ridge regression: An unknown execution time of program  $P_i$  with parameters  $Arg^{(i)}_j$  on computer installation  $cn_k$  is redefined by ridge regression in [27] based on known execution times of the program  $P_i$  with the same parameters on all other computer nodes;
2. Cliques: Firstly, group computer installations using the Pearson correlation coefficient. Secondly, redefine unknown execution time of program  $P_i$  with parameters  $Arg^{(i)}_j$  on computer installation  $cn_k$  based on execution times of the group of computer nodes where  $cn_k$  belongs.
3. Matrix decomposition: Apply matrix decomposition in [28] of the *PC matrix* in order to fill in empty entries of the *PC matrix*;
4. It has to be noted that if the program was executed with different sets of input parameters, this program is represented by multiple rows in the *PC matrix*.

*Ridge regression* algorithm is used as is. It is only worth to mention that ridge regression is used to predict the value in empty entries in a row of *PC matrix*. If there is an empty value in a row of the *PC matrix*, this value is prediction using all known values. In fact, the problem of interpolation is solved. If the columns corresponding to the computer installations in the *PC matrix* are ordered by performance (this data can be taken from the description of the computer installations), then this type of regression can get quite well prediction. Ridge regression works well on dense matrices with a small number of empty entries in *PC matrix*.

*Pearson correlation* is proposed to estimate the proximity of the set of vectors of program execution times. Since the algorithm is used as is and no novelty was introduced into it is just recalled below. The columns of the *PC matrix* are considered as such sets in other words. The correlation between the columns  $cn_i$  and  $cn_j$  shows how close in performance different computational nodes running these programs are. If the Pearson correlation between these columns is close to 1, then  $cn_i$  and  $cn_j$  are close to each other from the point of view of performance on the given set of the program executions. In this case, the estimate of the program execution time for  $cn_j$  can be obtained by multiplying by the constant of the time estimate on the  $cn_i$  node.

The procedure for distributing computing nodes into groups consists of the following steps:

1. Calculate Pearson correlation for each pair of columns of the *PC matrix* –  $N^*(N-1)/2$  pairs, where  $N$  is the number of computational nodes;

2. Build the graph of cliques:
  - a. Each vertex represents  $cn$ ;
  - b. An edge between the  $cn_i$  and  $cn_j$  exists if the Pearson correlation between the corresponding columns in the PC matrix is modulo greater than some threshold. The value of threshold is an algorithm parameter; this way brings us a graph where nodes are  $cn_i$ , and arcs are correlated pairs of computational nodes.
3. Find all groups such that any two vertices in the group are connected by an edge. To do this, we use the algorithms from [29, 30].
4. Each group contains computing nodes  $cn$  such that the program execution times on different nodes have a linear dependence, that is, one can be obtained from the other by multiplying by some coefficient. It should be borne in mind that the same calculator can be included in different groups.
5. The resulting cliques are groups of computational nodes.

To search for groups, a special algorithm was developed, presented in [24], whose complexity does not exceed  $H^3$ , the size of groups is less than  $3H^{H/3}$ . This algorithm was described with detail in [31].

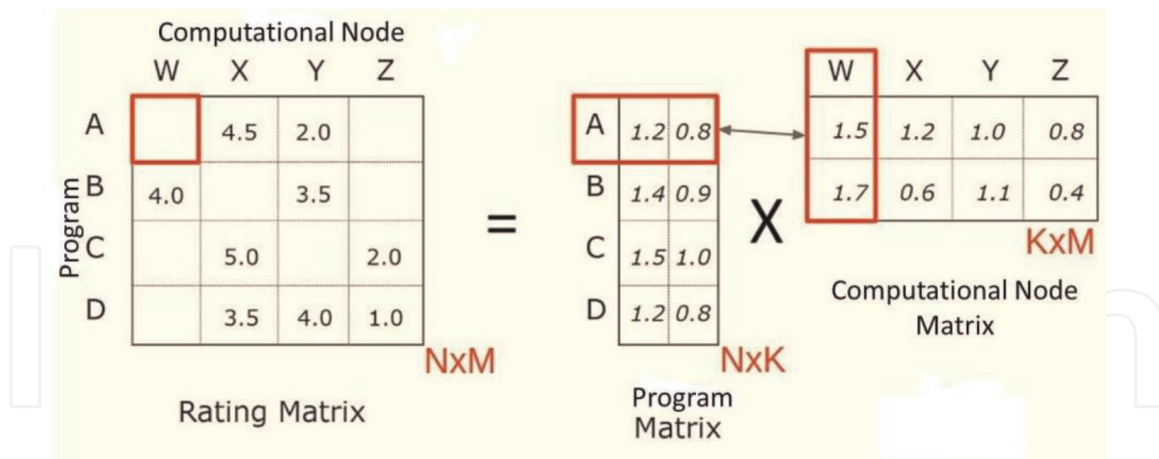
However, if threshold for the value of Pearson correlation is close to 1, then further prediction algorithm described in [32] is pretty good even if some vertexes in the cliques would be missed.

If it is not possible to calculate Pearson correlation (e.g., the considered program  $P$  has not been run on any of the computer installations from clique) but corresponding row in *PC matrix* for  $P$  program is non-empty, then one needs to use ridge regression for the prediction. See step 4 above.

The error of prediction for the algorithm presented above can be estimated by Eq. (3).

*Matrix decomposition:* Because this stage in the proposed algorithms ensemble we consider as our main contribution to the considered problem, we will spend more space on its description. As mentioned above, the problem of programs execution time prediction is very similar to the problem solved in the recommender systems. In these systems, there are usually two types of objects, the relationships between which are measured. For example, such objects can be users and movies, users and books, users and goods, and so on. The relationship between them is often a measure to what extent the user prefers some movies, books, or goods. It is called goods rating. One can build a rating matrix where the rows (or columns) correspond to movies, books, or goods, and the columns (or rows) correspond to users. This matrix is often sparse, since there are a lot of users and objects, and users cannot physically rate all the objects. The problem that solve the recommender systems, is to determine the ratings of all users for all objects; in other words, the system has to fill in the empty entries in the rating matrix.

Let us consider the following analogy: users are computer installations, the goods are programs, and the ratings are execution times. Thus, the computer installations “rate” the programs, and the smaller rating (execution time), the better the computer installation meets the program.



**Figure 2.**  
 Matrix decomposition.

The problem of filling empty entries in recommender systems is solved by matrix decomposition method in [23]. We propose to use this technique to solve the problem under consideration.

The application matrix decomposition techniques to some matrix result in two or more matrices, the product of which gives the approximation of the original one. For empty entries of the original matrix, that is, for unknown values, the product of the matrices gives values that estimate the unknown values.

PC matrix decomposition allows one to get a vector representation of programs and computational nodes, which have a remarkable property: the scalar product of the vector representation of the program and the vector representation of the nodes is the program execution time on the node. The vector representations of programs and computational nodes are called as embeddings of programs and computational nodes, respectively. Embedding techniques and methods of applying embeddings are very well-known in such areas as NLP in [33], topic modeling in [34], and recommender systems in [25].

**Figure 2** demonstrates matrix decomposition. The rows are programs, and columns are computers that should be rated. The matrix entries are execution times corresponded to the pairs (program, computer). Some entry could be empty.  $K$  is a parameter of the decomposition and is a subject of tuning to get the admissible accuracy. The result of the decomposition procedure of rating matrix of size  $N \times M$  is program matrix of size  $N \times K$  and computer matrix of size  $K \times M$ . The rows in program matrix are vectors represented execution times for the program on the corresponded computers. The columns are vector representations of the “computational power” the corresponded computer for the programs under consideration.

In our study, ALS algorithm [35] was selected.

As mentioned above, three algorithms were chosen: ridge regression, Pearson correlation, and matrix decomposition for the program execution time prediction. As it was said previously, the averaging ensemble in [26] of them to improve the accuracy of the predicted execution time was developed. These algorithms can be combined into an ensemble of algorithms to improve the accuracy of the prediction in [26].

## 6.2 Experimental study

Here, the results of experimental studies of the algorithms comparison are presented.



The purposes of the experiments are analysis of the quality of prediction results:

1. based on grouping computational nodes by the Pearson correlation;
2. based on ALS matrix decomposition algorithm. Selecting the parameter  $K$ —the number of components in the vector representation of programs and computer installations;
3. by the ensemble of algorithms.
4. the data for experimental study were used from the website <https://spec.org>. The quality of the proposed solution is estimated by the prediction error and prediction accuracy is calculated according to Eq. (3).

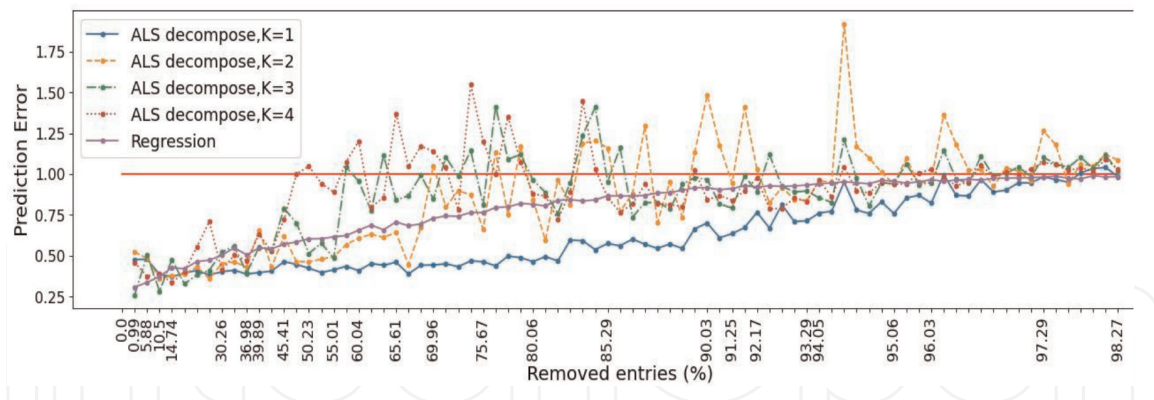
#### 6.2.1 Analysis of the quality of prediction based on grouping computer installations by Pearson correlation

Dataset MPIM2007 with 13 programs and 437 computer installations from [22] was used for the experiments. The algorithm based on grouping computer installations by Pearson correlation is very sensitive to the presence of outliers in the data, as well as to what extent the *PC matrix* is low-density. In order to analyze the quality of prediction by this algorithm, three experiments were conducted. As a basic algorithm, the ridge regression algorithm was used. Each experiment was conducted according to the following methodology. In each row, only one value was removed, and then, the prediction was made based on the remaining values in this row of the matrix according to the algorithm above.

In the first experiment, the execution time is predicted by the ridge regression algorithm; as a result, the prediction error was 0.25 or 25%. In the second and third experiments, grouping algorithms based on Pearson correlation were used; threshold for correlation value was 0.97; the grouping resulted in 46 groups with two and more computer installations and 27 groups with only one computer installation. In the second experiment, execution time was predicted only for groups with size greater than or equal to 2; groups with size 1 were ignored. As a result, the prediction error was 0.068 or 6.8%. In the third experiment, execution time was predicted by Pearson correlation algorithm for groups with size greater than or equal to 2, but for groups with size 1, ridge regression algorithm was used. As a result, the prediction error was 0.115 or 11.5%. Thus, the accuracy of prediction on dense matrices is 88.5%.

#### 6.2.2 Analysis of the quality of prediction based on ALS matrix decomposition algorithm

Dataset MPIM2007 with 13 programs and 437 computer installations from [22] was used for the experiments. Experiments were conducted according to the methodology described in Section 4. To study the quality of the predictions based on ALS matrix decomposition algorithm, 4 experiments were made with  $K = 1, 2, 3, 4$ . The results of the matrix decomposition were compared with each other, as well as with ridge regression algorithm that was chosen as the basic prediction algorithm. In **Figure 2**, X-axis is the percentage of empty entries in the *PC matrix* (which randomly was removed from it); Y-axis is the prediction error. Also, for comparison, the result of predictions by the Pearson correlation algorithm was added to **Figure 2**. According to the plots in **Figure 2**, the conclusion can be made that the ridge and cliques



**Figure 3.**  
 Ridge regression and ALS matrix decomposition with  $K = 1, 2, 3, 4$ .

algorithms work well on dense matrices (up to 1% percentage of empty entries). The matrix decomposition technique with  $K = 1$  gives the best results for sparse matrices in which the number of empty entries is more than 15%. Even if 80% of the entries are removed from the *PC matrix*, they can be predicted with an accuracy of 60%.

Thus, as a result of experiments, we can conclude that the matrix decomposition technique with  $K = 1$  gives the best solutions when the percentage of empty entries in the *PC matrix* is more than 15%.

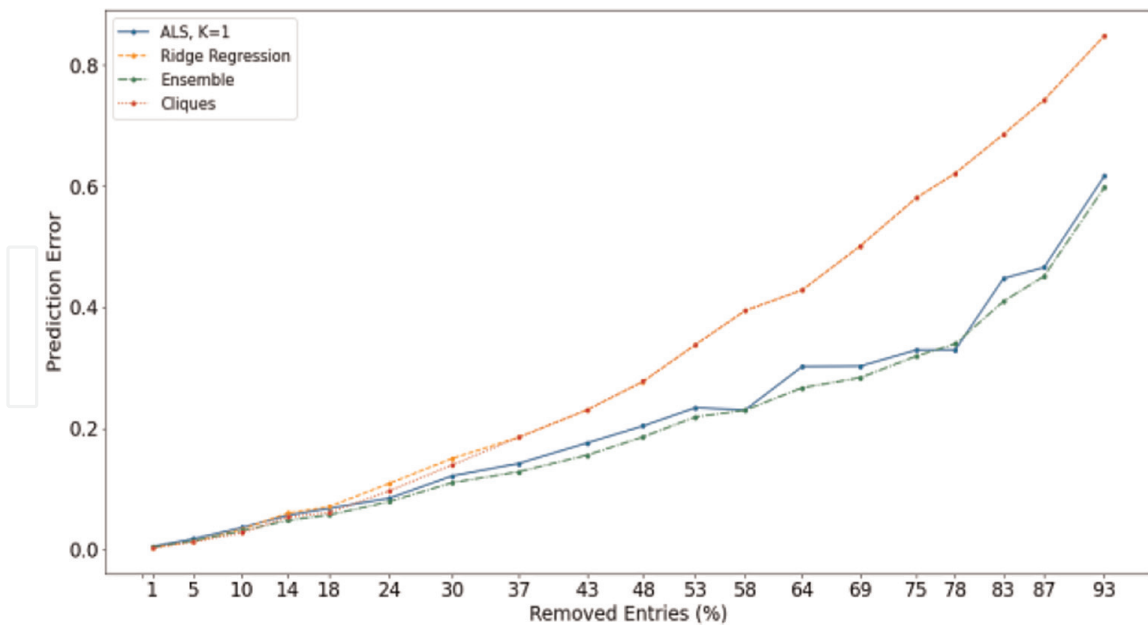
An important advantage of using the matrix decomposition technique is the vector representations (embeddings) of programs, and computer installations in case  $K = 1$  are the points in a space of dimension 1. So, the total ordering on the set of computer installations could be defined, and one can work with them as scalars. **Figure 3** shows the less embedding of computational node, the less execution time of the corresponded program.

### 6.2.3 Analysis of the prediction quality of an ensemble of algorithms

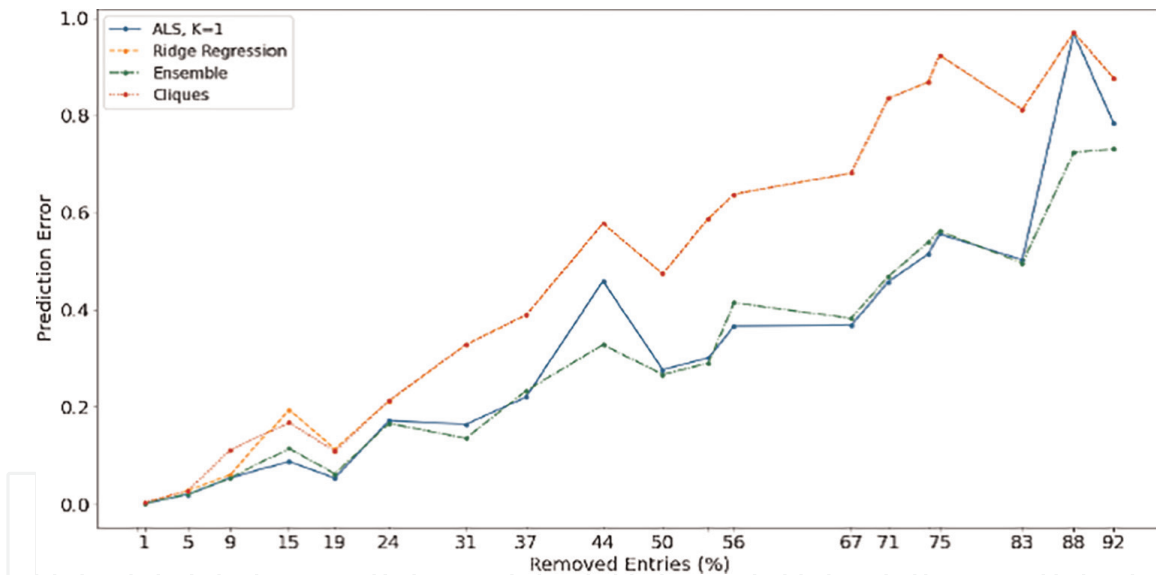
The ensemble averaging described by (Eq. (3)) was used. The prediction result was compared with the following algorithms: *ridge regression*, *Cliques*, and *ALS* with  $K = 1$ . All three datasets—MPIL2007, MPIM2007, and ACCEL OMP from [22] were used for the experiments. Experiments were conducted according to the methodology described in Section 4. As we can see in **Figure 3**, the best estimations give us the *ALS* algorithm when the matrix sparsity is at least 14%. According to **Figure 4**, the ensemble averaging is better when the matrix sparsity is over 14% up to 94%. One more testing of the proposed method was done of the dataset ACCEL OMP that covers 15 programs and 30 computers. The results of this experiment are shown in **Figure 5**.

## 7. Organization of NPC computing resources

As I presented in [2], the computing node (CN) could be as Edges in [36] as supercomputer or HPC installation. Existing data center construction approaches demand high quality of communication channels service, to ensure availability of service, and very high capital construction costs of a centralized data center. Significant problems of traditional DC are scaling and low level of resource utilization due to the lack of a centralized management system and orchestration system [37].



**Figure 4.** Results of ridge, cliques, matrix decomposition, and an ensemble of algorithms on MPIM2007 (1–94%).



**Figure 5.** Results of Ridge, Cliques, Matrix decomposition and an ensemble of algorithms on ACCEL OMP (1–92%).

The advantages of building a NPC based on edges over the traditional approach have been discussed in detail in [38] and characterized by: reduction of transport requirements by proximity of the service copy to the final consumer, reducing the cost of organizing a data center due to the absence of the need to build a centralized data center, efficient scaling through the use of a centralized cloud platform, increasing the efficiency of the network due to a centralized management and orchestration system, and the proximity of the service to the client. The problems of organizing the control plane and the data plane in edges are in many ways similar to those that were already listed above for the NPC Federate DTN control layer (see **Figure 1**). The main difference—the decision-making speed should be much higher.

Another important point is the following. Managing and optimizing power has been a long-standing challenge in computer systems, with many fundamental techniques, and has been increasingly receiving attention. Today, a wide range of applications such as IoT services, ML inference, data analytics, and scientific computing now use Functions as a Service (FaaS) offerings of cloud platforms such as Amazon Lambda, Azure and Google Functions, and so on. Widespread application deployment model has nowadays become serverless computing, where applications are partitioned into small, fine-grained “functions”, whose execution is managed by the cloud platform as in [39]. However, the constant evolution of cloud abstractions and usage models poses new energy efficiency challenges.

Serverless features can provide unique opportunities to reduce cloud heat generation by reducing the level of concentration of computing power in a relatively small location and programming model. Many power-saving techniques, such as workload migration and on-demand scaling, that are difficult for regular VMs and containers, can be significantly easier to develop and optimize for serverless functions that can “run anywhere”[40]. In this way, FaaS can provide new power tools for cloud platforms to quickly and finely move applications to environmentally friendly locations, which will be especially useful for distributed cloud edges powered by renewable energy such as solar and wind, as in [41]. The FaaS programming model also allows power management at the functional level. A function can have multiple implementations that differ in power consumption and performance. This may allow the cloud provider to use the appropriate feature implementation based on power availability and application performance constraints. Finally, features are reused, enabling data-driven and machine learning methods such as transfer learning that can be used for general and practical energy management.

One possible approach to energy management is a computing infrastructure model based on the concept of a cloud data center network and cloud edges where computing can be scheduled based on energy consumption. In the FaaS model, functions are not tied to any specific servers or locations and can potentially “run anywhere” as long as the runtime platform has access to the function’s code dependencies and the container/VM “image”. By separating computing from its location, serverless computing allows us to run functions in the most power-efficient location. This location independence can be an extremely efficient technique for resilient computing but is often challenging for other workloads. Because renewable energy sources (such as solar and wind) can be fickle, the availability of servers powered by them is temporary [42].

AI routing in distributed edge clouds can offer different trade-offs between energy and carbon emissions depending on location, time, and availability of resources and hardware. Functions can be run on the edge to ensure low latency. The trade-off between power consumption and performance adds a new dimension to the discussion of future cloud architectures. While edge clouds may have performance and security/privacy benefits, their energy benefits require further analysis.

As you know, for maximum efficiency of program execution, a certain set of hardware and their configuration is required. Several attempts have already been made to implement the approach of dynamically adapting the architecture to the application, that is, see [32]. Currently, to meet this need, it is proposed to use the resource disaggregation approach in [27]. Its essence is as follows. Data centers have been using the monolithic server model for over 20 years, where each server has a motherboard that houses all types of hardware resources, typically including the processor, memory chips, storage devices, and network cards. Resource disaggregation involves dividing the server’s hardware resources into standalone network-

connected devices that applications can access remotely. Applications must be given virtualized and secure access to hardware resources, and data centers must support these applications with tools that ensure their good performance.

One of the possible approaches to manage energy consumption and carbon footprint is the computational infrastructure model based on the concept network of cloud DC and cloud edges, where computation can be organized by energy and carbon-based scheduling. In FaaS model, functions are not tied to any specific servers or locations and can potentially be “run anywhere”, as long as the execution platform has access to the function’s code dependencies and the container/VM “image”. By decoupling computation from its location, serverless computing allows us to run functions at the most energy-suitable location. Thus, even though individual functions may not be energy efficient, they can be run in carbon-friendly locations to achieve better carbon efficiency. This location independence can be an extremely potent technique for sustainable computing, but is often challenging for other workloads. Since renewable energy (such as solar and wind) can be intermittent, the availability of servers powered by them is only transient [42].

AI routing in distributed edge clouds can offer different energy/carbon trade-offs depending on location, time, and resource and hardware availability. Functions can be run on the edge to provide low latency. The energy versus performance trade-off adds a new dimension to the discussion on future cloud architectures. While edge clouds may have performance and security/privacy advantages, their energy benefits need additional analysis.

In the history of computer architecture, there have been many attempts to make the architecture of computers dynamically adoptable to the structure of the algorithm of the program that it executes [32]. However, all of them were not very successful. Currently, a new direction is gaining momentum—resource disaggregation. This direction does not use the monolithic model of server that hosts all types of hardware resources like CPU cores, memory chips, storages, and NICs. In resource disaggregation, server is split on individual devices connected by a high-speed network. The user can choose the architecture for the virtual machine and the configuration of individual devices to ensure the most efficient execution of his applications. Application has virtualized and reliable access to all devices. This approach to computer architecture requires a rethinking of the concept of the operating system. The traditional view of it has already become obsolete. But this is a topic for a separate post.

## **8. Conclusion**

The Network Powered by Computing (NPC) concept of next generation computational infrastructure was presented. This concept based on the convergence of data communication networks with computing facilities like DC, edge, and HPC centers united by the functional architecture is presented in the article. The NPC concept is the incarnation of the slogan I got from Jhon Gadge from Sun Microsystems: “Network is a computer”. Here, we considered the functional architecture of NPC, and the main problems on the way of its implementation are described. The presented concept allows to achieve deep automation in the management of resources of this infrastructure, load distribution, and energy consumption through the use of methods based on machine learning algorithms.

The issue of organizing the ASNf layer, which, together with the OAM and the NPCIC layers, is essentially new generation of the operating environment—an

analogue of the traditional operating system. But this is an independent, large topic that requires a separate publication.

## **Acknowledgements**

The author is grateful to Professor V. Korolev and Professor A. Borisov who read the manuscript and gave suggestions for its improvement and to Ph.D. student Andrey Chupakhin, who performed the entire cycle of experiments to evaluate the proposed method for predicting MPI program execution time.

Also, thanks to Marina Trukhova who helped a lot with shaping the text of this paper.

## **Conflict of interest**

The author declares no conflict of interest.

## **Author details**


Ruslan Smeliansky

Department of Computational Mathematics and Cybernetics, Moscow State University, Moscow, Russia

\*Address all correspondence to: [smel@cs.msu.su](mailto:smel@cs.msu.su)

## **IntechOpen**

---

© 2023 The Author(s). Licensee IntechOpen. This chapter is distributed under the terms of the Creative Commons Attribution License (<http://creativecommons.org/licenses/by/3.0>), which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited. 

## References

- [1] Hennessy JL, David A. Patterson. A new golden age for computer architecture. *Communication of the ACM*. 2019;**62**(2):48-60
- [2] Smeliansky R. Network powered by computing. In: *International Conference on Modern Network Technologies (MoNeTec)*. 2022. pp. 1-5. DOI: 10.1109/MoNeTec55448.2022.9960771
- [3] Smeliansky R. Hierarchical edge computing. In: *Proceedings of International Scientific and Technical Conference Modern Computer Network Technologies (MoNeTeC)*. 2018. pp. 1-11. DOI: 10.1109/MoNeTeC.2018.8572272
- [4] Topology and Orchestration Specification for Cloud Applications [Internet]. Available from: <http://docs.opensis-open.org/tosca/TOSCA/v1.0/os/TOSCA-v1.0-os.html> [Accessed: December 10, 2022]
- [5] Elasticity [Internet]. Available from: <https://wa.aws.amazon.com/wellarchitected/2020-07-02T19-33-23/wat.concept.elasticity.en.html> [Accessed: December 13, 2022]
- [6] "What is Serverless Computing?" *ITPro Today*. December 13, 2021 [Retrieved: 23 March 23, 2022]
- [7] Smeliansky R. MC2E: The environment for interdisciplinary research. *Engineering Letters*. 2021;**6**: 40-54
- [8] Demonstration of the Bandwidth on Demand Application Based on the SDN Controller RunOS [Internet]. 2021. Available from: <https://www.youtube.com/watch?v=TuzDDZT4NL4> [Accessed: December 11, 2022]
- [9] Total Datacenter Power [Internet]. Available from: <https://451research.com/total-data-management> Accessed: 2022-12-12]
- [10] Albert-László Barabási Network Science. The Barabási-Albert Model [Internet]. Available from: <https://barabasi.com/f/622.pdf>
- [11] Albert-László Barabási Network Science. Chapter 3 [Internet]. Available from: <http://networksciencebook.com/chapter/3#degree-distribution>
- [12] Random Forest Algorithm [Internet] [Online]. Available from: [https://www.stat.berkeley.edu/breiman/RandomForests/cc\\_home.htm](https://www.stat.berkeley.edu/breiman/RandomForests/cc_home.htm) [Accessed: April 4, 2022]
- [13] Gonzalez MTA. Performance Prediction of Application Executed on GPUs Using a Simple Analytical Model and Machine Learning Techniques [Online]. 2018. Available from: <http://doi.org/10.11606/t.45.2018.tde-06092018-213258> [Accessed: April 4, 2022]
- [14] Snavelly A, Carrington L, Wolter N, Labarta J, Badia R. A framework for performance modeling and prediction, SC '02. In: *Proceedings of the 2002 ACM/IEEE Conference on Supercomputing*. Baltimore, MD, USA; 2002. pp. 21-21. DOI: 10.1109/SC.2002.10004
- [15] Nadeem F, Alghazzawi D, Mashat A, Fakeeh K, Almalaise A, Hagraas H. Modeling and predicting execution time of scientific workflows in the grid using radial basis function neural network. *Cluster Computing*. 2017;**20**(3): 2805-2819. DOI: 10.1007/s10586-017-1018-x

- [16] Yero EJH, Henriques MAA. Contention-sensitive static performance prediction for parallel distributed applications. *Performance Evaluation*. 2006;**63**(4):265-277
- [17] Wu Q, Datla VV. On performance modeling and prediction in support of scientific workflow optimization. In: *Proceedings of the 2011 IEEE World Congress on Services, SERVICES '11*, IEEE Computer Society. 2011. pp. 161-168
- [18] Li H, Groep D, Templon J, Wolters L. Predicting job start times on clusters. In: *CCGRID '04 in Proceedings of the 2004 IEEE International Symposium on Cluster Computing and the Grid*. 2004. pp. 301-308
- [19] Liu X, Chen J, Liu K, Yang Y. Forecasting duration intervals of scientific workflow activities based on time-series patterns. In: *Proceedings of the IEEE Fourth International Conference on eScience, eScience '08*. 2008. pp. 23-30
- [20] Lee BC, Brooks DM, de Supinski BR, Schulz M, Singh K, Mckee SA. Methods of inference and learning for performance modeling of parallel applications. In: *Proceedings of the 12th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, PPOPP '07*. 2007. pp. 249-258
- [21] Yang LT, Ma X, Mueller F. Cross-platform performance prediction of parallel applications using partial execution. In: *Proceedings of the 2005 ACM/IEEE Conference on Supercomputing, SC '05*, IEEE Computer Society. 2005. pp. 40-44
- [22] Standard Performance Evaluation Corporation [Internet]. 2021, Available from: <https://spec.org> [Accessed: December 10, 2022]
- [23] Benchmark Documentation [Internet]. Available from: <https://spec.org/mpi2007/Docs/> [Accessed: December 13, 2022]
- [24] MPI Program Execution Time Prediction Algorithms from the Article. *IFIP Networking 2022 SLICES Workshop* [Internet]. Available from: <https://github.com/andxeg/> [Accessed: December 11, 2022]
- [25] Ricci F, Rokach L, Shapira B, Kantor PB. Recommender System Handbook [Internet] [Online]. Available from: <https://link.springer.com/book/10.1007/978-0-387-85820-3> [Accessed: April 4, 2022]
- [26] Dzˇeroski S, Panov P, Zˇenko B. Machine learning, ensemble methods. In: Meyers R, editor. *Encyclopedia of Complexity and Systems Science*. New York: Springer; 2009
- [27] Ridge Regression [Internet] [Online]. Available from: [https://www.ncss.com/wp-content/themes/ncss/pdf/Procedures/NCSS/Ridge\\_Regression.pdf](https://www.ncss.com/wp-content/themes/ncss/pdf/Procedures/NCSS/Ridge_Regression.pdf) [Accessed: April 4, 2022]
- [28] Cheng CM, Jin XQ. Matrix decomposition. In: Alhaji R, Rokne J, editors. *Encyclopedia of Social Network Analysis and Mining*. New York, NY: Springer; 2018
- [29] Skiena SS. Clique and Independent Set and Clique. §6.2.3 and 8.5.1: *The Algorithm Design Manual*. New York: Springer-Verlag; 1997. p. 144 and pp. 312-314
- [30] Johnston, H.C. Cliques of a graph- variations on the Bron-Kerbosch algorithm. *International Journal of Computer and Information Sciences* 5,



209-238 1976. Available from: <https://doi.org/10.1007/BF00991836> [Accessed: December 10, 2022]

[31] Chupakhin A, Kolosov A, Smeliansky R, Antonenko V. New approach to MPI program execution time prediction. 10.48550/arXiv.2007.15338

[32] Smeliansky RL, Antonenko VA, Stepanov EP, Chupakhin AA, Kolosov AM. Chapter 1: On HPC and cloud environments integration. In: Performance Evaluation Models for Distributed Service Networks. Cham, Switzerland: Springer Nature Switzerland AG; 2020

[33] Li Y, Yang T. Word embedding for understanding natural language: A survey. In: Srinivasan S, editor. Guide to Big Data Applications. Studies in Big Data. Vol. 26. Cham: Springer; 2018

[34] Aggarwal CC. Matrix factorization and topic modeling. In: Machine Learning for Text. Cham: Springer; 2018

[35] Gabor Takacs and Domonkos Tikk. Alternating least squares for personalized ranking. In Proceedings of the Sixth ACM Conference on Recommender Systems (RecSys '12). Association for Computing Machinery. New York, NY, USA; 2012. pp. 83-90. Available from: <https://doi.org/10.1145/2365952.2365972> [Accessed: December 11, 2022]

[36] Brun R, Urban L, Carminati F, Giani S, Maire M, McPherson A, Patrick G. 1993. GEANT: de tector description and simulation tool No. CERN-W-5013. CERN

[37] Masanet E, Shehabi A, Lei N, Smith S, Koomey J. Recalibrating global data center energy-use estimates. Science. 2020;**367**(6481):984-986

[38] Antonenko V, Petrov I, Smeliansky RL, Huang Z, Chen M, Cao D, et al. MC2E: Meta-cloud computing environment for HPC. In: Performance Evaluation Models for Distributed Service Networks. Cham, Switzerland: Springer Nature Switzerland AG; 2019

[39] Schleier-Smith J, Sreekanti V, Khandelwal A, Carreira J, Yadwadkar NJ, Popa RA, et al. What serverless computing is and should become: The next phase of cloud computing. Communications of the ACM. 2021;**64**:7684

[40] Prateek Sharma Indiana University Bloomington [Internet]. Challenges and Opportunities in Sustainable Serverless Computing. Available from: <https://cgiluddy.indiana.edu/~prateeks/papers/hotcarbon22.pdf> [Accessed: December 10, 2022]

[41] Chien AA. Driving the cloud to true zero carbon. Communications of the ACM. Jan. 2021;**64**(2):5-5

[42] Singh R, Sharma P, Irwin D, Shenoy P, Ramakrishnan K. Here today, gone tomorrow: Exploiting transient servers in data centers. IEEE Internet Computing. 2014;**18**