Hybrid Intellectual Scheme for Scheduling of Heterogeneous Workflows based on Evolutionary Approach and Reinforcement Learning

Mikhail Melnik, Ivan Dolgov and Denis Nasonov
ITMO University, Saint-Petersburg, Russia

Keywords: Workflow Scheduling, Stream Data, Cloud Computing, Supercomputer, Hybrid Approach, Evolutionary Computing, Reinforcement Learning.

Abstract: Scheduling of workload in distributed computing systems is a well-known optimization proble. A workload may include single independent software packages. However, the computational process in scientific and industrial fields is often organized as composite applications or workflows which are represented by collection of interconnected computing packages that solve a common problem. We identified three common computing modes: batch, stream and iterative. The batch mode is a classic way for one-time execution of software packages with an initially specified set of input data. Stream mode corresponds to launch of a software package for further continuous processing of active data in real time. Iterative mode is a launching of a distributed application with global synchronization at each iteration. Each computing mode has its own specifics for organization of computation process. But at the moment, there are new problems that require organization of interaction between computing modes (batch, stream, iterative) and to develop optimization algorithms for this complex computations that leads to formation of heterogeneous workflows. In this work, we present a novel developed hybrid intellectual scheme for organizing and scheduling of heterogeneous workflows based on evolutionary computing and reinforcement learning methods.

1 INTRODUCTION

This research is associated with an intensive increase in the performance of computing systems (CS) through the use of global distributed computing technologies for scientific and industrial computing, and the simultaneous complication and expansion of the scope of computing applications in research. Currently, the issue of introducing CS with exaflops performance by 2024 is being discussed, which leads not only to an increase in the number of computing elements, but also to the complexity of their architecture. Effective use of this kind of CS is a non-trivial scientific task. The computational process in the scientific and industrial fields is often not limited to the launch of a single computing package. Global distributed computing technologies deal with composite applications (CA) or workflows (WF), which are a set of interconnected services in a distributed environment that solve a common problem. CA services are represented by different software solutions working on various architectures, operating systems, and developed by different teams using various technologies. This is the main source of heterogeneity of computations in global distributed environments. At the same time, the complexity of organization of computations includes the requirement to share specialized resources, such as: graphic processors, supercomputers, elements that allow you to perform computation with dynamically changing workload of individual parts of the CA.

Here we identify three common computing modes: batch, stream and iterative. The batch mode is a classic way for one-time execution of software packages with an initially specified set of input data. Stream mode corresponds to launch of a software package for further continuous processing of active data in real time. Stream data processing can be found everywhere in our modern world. For example, data from social media, Internet of Things, many web services. Using cloud computing is the most popular option to perform data processing in the stream mode. Iterative mode is a launching of a distributed application with global synchronization at each iteration, until which distributed blocks process their part of calculations. Iterative mode is more common for scientific community, for example complex multiscale models which often are executing in supercomputer environ-

Melnik, M., Dolgov, I. and Nasonov, D.
Hybrid Intellectual Scheme for Scheduling of Heterogeneous Workflows based on Evolutionary Approach and Reinforcement Learning.
DOI: 10.5220/0010112802000211
In *Proceedings of the 12th International Joint Conference on Computational Intelligence (IJCCI 2020)*, pages 200-211
ISBN: 978-989-758-475-6

ments.

In addition, the situation is complicated by the need to organize interaction between heterogeneous modes of computing and data processing (batch, streaming, iterative). That is why we formed a concept of heterogeneous workflows (HWF). Heterogeneous workflow is a WF which has blocks with various computing architectures and principles of the organization of the execution process, requiring an individual technological approach at the level of a single computing platform. Executin of HWF may require usage of heterogeneous computing resources (cloud computing, local clusters, supercomputers) that forms global computing environment. As part of the HWF processing, it is required to organize both the execution process for each mode and the data transfer process between them. Thus, a single mechanism for managing the performance of the HWF is needed, which is able to analyze the components of the HWF components and interact with all participating platforms and technologies.

For efficient organization of computing process it is necessary to optimize execution process of HWFs. It can be optimized by scheduling technologies and algorithms. Effective scheduling allows you to reduce the amount of transferred data, arrange computing packages on resources meeting the requirements, thereby reducing the execution time of a distributed application or save money in a case of cloud computing. The workflow scheduling is a well-known optimization problem. However, in a case of HWF, we are faced with uncertainties due to the interaction between computing modes and heterogeneity of the environment.

Scheduling of HWF arises new architectural problem in distributed computing environments. It requires complex approach for efficient organization of computation. In this work we presented our developed hybrid intellectual scheme for organization and optimization of HWF execution. The scheme organizes interaction between computing modes and provides scheduling algorithms for optimization of HWF execution. The scheduling algorithms are provided for batch, stream and iterative modes and are previously demonstrated in our researches (Melnik and Nasonov, 2019)(Melnik et al., 2018)(Melnik et al., 2019). Therefore we combined our developed methods under single technology. In a batch case, we developed a scheduling algorithm that based on principles of reinforcement learning (RL) and neural networks (NN) which allows dynamic scheduling at the level of structure analysis of the computing load and infrastructure and dynamically improve the proposed solution. For optimizing the performance of streaming data processing we developed a evolutionary algorithm (EA), providing flexibility and scalability of calculations by assessing the need for computing resources. For iterative mode we also developed evolutionary algorithm which allows using the logic of computational applications to significantly reduce the execution time and balancing computing resources.

For experimental study we used a developed scenario of a problem for generation of plans for snow cleaning in a city based on data from social media. We constructed a HWF that aimed to solve the problem. The HWF is composed by software packages in batch, stream and iterative modes. We evaluated the performance of our developed hybrid scheduling scheme with a scheme based on existing perspective algorithms.

2 RELATED WORKS

Each computing mode uses specific technologies to organize data processing that have their own interfaces for data, software modules, and mechanisms for interacting with resources. Thus, the optimization of HWF includes the optimization of execution of all three modes. To do this, it is necessary to study the technologies and methods for optimizing the scheduling for each mode.

Batch. There are many various Scheduling algorithms for Batch data processing in different computing systems (supercomputers, cloud computing, grid-systems etc). The scheduling can be performed in static and dynamic cases. Static scheduling is characterized by building a schedule before execution of CA. Dynamic scheduling is happened, when schedule is built up on the fly, while CA execution happened.

In general cases, Scheduling problem is NP-complete. Because of it there is two classes of algorithms, which can solve this problem non-optimally in general: heuristic and meta-heuristic algorithms.

Heuristic algorithms are greedy; based on ranked listings; based on a specific set of deterministic rules by which computing tasks are distributed among infrastructure nodes. These rules are based on type of a computing system and the requirements of tasks on computing resources, which is not always possible, because of uncertainty of global computing systems. Heuristic algorithms in most cases are depended on specific type of computing load. The main advantage of heuristic algorithms is the speed of obtaining schedules, what is important for dynamic scheduling. The most usable and famous scheduling heuristic algorithms are CPOP (Critical Path on a Pro-

cessor), HEFT (Heterogeneous Earliest Finish Time) (Topcuoglu H., 2012), HCPT (Heterogeneous Critical Parent Trees) (Hagras and J., 2003) .

Meta-heuristic algorithms imply the implementation of evolutionary and swarm approaches. Such algorithms include: Genetic Algorithm (Nagar et al., 2018), Gravitational Search Algorithm (Choudhary et al., 2018), Simulated Annealing or Particle Swarm Optimization (Masdari et al., 2017) etc. Schemes of evolutionary and swarm algorithms can be developed directly for the Scheduling problem. Since meta-heuristic algorithms require significantly more time to obtain a solution, hybrid schemes are often built to make up this drawback by integrating heuristic algorithms at the initialization stage of meta-heuristic algorithms.The main advantages of meta-heuristic algorithms are better quality of resulting schedules and much wider range of applicability, but meta-heuristic algorithms require more time to search for quality solutions.

Despite the fact that hybrid schemes provide access to the positive qualities of heuristic and meta-heuristic algorithms, it is necessary to quickly and efficiently solve problems based on a wide range of calculations using a computing system, including the wide heterogeneity of computational models and resources. Promising for solving the scheduling problem are machine learning methods, in particular, reinforcement learning, because these methods can assimilate system monitoring data, predict the density of the computational load, and also be able to tune performance models for the most accurate assessment of the characteristics and indicators of data, computational models, resources and the quality of the resulting solutions in general.

There is a number of papers ((Hussain A., 2016), (Ismayilov G., 2020), (Vukmirovič S., 2012), (Xiao Z., 2017)) devoted to the use of machine learning approaches for solving secondary tasks, such as refining the assessment of individual tasks according to historical data, predicting changes in workload, refining the assessment of time of new task arriving. But also there are papers, where machine learning approach is used for solving Scheduling problem directly (Yao J., 2006) using Reinforcement Learning (RL) method. In this paper RL is used for specific Scheduling problem - Scheduling in Grid-systems. In work (Rashmi S., 2017) RL approach is used for scheduling tasks in MapReduce. Authors of the most promising algorithm DQTS in paper (Tong Z., 2019) consider more general formulation of the problem and use one of the RL techniques - Deep Q-learning method to get results with higher metrics (makespan and nodes load), than baseline (MAXMIN, MINMIN, FCFS algorithms) solution. But most of the papers use only basic information about computing system (number of tasks and quantity of resources) and these problem leads to less flexible scheduling system. But compared with heuristic and meta-heuristic algorithms, their advantages include: the ability to use the accumulated experience of running various WF, including changing conditions based on launch statistics; the ability to learn patterns in WF structures, resources and their combination to provide a much faster search for solutions than meta-heuristic algorithms and better than heuristics; the ability to automatically adapt to changing conditions, the nature of the load and its modes of receipt, developing new scheduling strategies, which is difficult or even impossible for heuristics and meta-heuristics methods.

Stream. Compared to batch processing, stream processing is characterized by the continuous flow of new data that require immediate processing. This creates the need for the simultaneous operation of all application operators. Due to continuity, the final amount of data to process cannot be determined. However, you can predict the density of the incoming computing load, and the resulting system load estimates can be taken into account when planning a streaming application. High-quality forecasting plays an important role, because of this, it becomes possible to adequately respond to changes by scaling (up and down) computing resources and achieve elasticity of calculations (the minimum difference between the need for resources and the amount of allocated resources).

Optimal configuration of the platform and application can increase data throughput, reduce latency and power consumption. Choosing the optimal number of nodes, the correct platform parameters, and the optimal distribution of application operators among computing nodes can ensure maximum system performance. Effective planning of streaming data processing can lead to increased productivity, resource utilization or reliability of the system as a whole, depending on the requirements of both users and providers.

For Stream data processing is widely used platforms such as Apache Storm, Spark Streaming, Flink, S4. However, the most part of algorithms are aimed for Storm (Peng B., 2015)(Agarwalla B., 2006)(Xu J., 2014).

Based on the analysis of the methods of scheduling streaming computing, it was revealed that at present this direction is at the development stage. The existing methods use explicitly available information and are not able to fully take into account the dynamics of changes and the incompleteness of

information about the incoming load into the system. Moreover, the solutions considered mainly used the Apache Storm platform for developing algorithms and conducting experiments. This necessitates the development of a generalized approach applicable to other alternative technologies (for example, Apache Flink or Spark Streaming).

Iterative. Applications running on a supercomputer are often presented as a composition of computational models of different scales (spatial or temporal) in terms of the subject area. These kinds of applications are called multiscale modeling applications. Separate levels (scales) can take the most significant part of all calculations (for example, the smallest scales). The modeling process is often iterative. In this case, the calculations are parallelized across nodes of the supercomputer and global synchronization is required at each iteration. Applications for multiscale modeling with an iterative nature of computing represent an iterative computing mode that has its own specifics in organizing and optimizing the execution process. Parallelization involves dividing the calculation area of a model into partitions, while the intensity of calculations between partitions can not only vary significantly, but also have dynamics, which greatly complicates the organization of calculations with effective scheduling.

The paper (Borgdorff J., 2014) provides a detailed analysis of the types of multiscale applications, as well as possible ways to organize their execution in a distributed environment. The authors identified three types of applications - connected, scalable and prioritized. For each type, application examples were selected and manual optimization was carried out. Experimental results shows that information about type of application significant speed up their execution.

In (Subedi P., 2019) is proposed the Stacker framework for efficiently moving data from multiscale CAs running in supercomputer environments. In particular, the work is aimed at optimizing the work with random access memory during the execution of distributed applications at various stages. But there is not ability to consider deadlines. The consideration of deadlines is a determining criterion when working with emergency computing systems, since in such systems the result often needs to be obtained by any possibility for a certain time.

In (Pollard S.D., 2019), the ways of efficient organization of parallel computing on the basis of partitioning the modeling domain and their purpose into various processor architectures are investigated. However, this article does not examine the internal characteristics of the selected areas and their balancing.

Conclusion. Despite the presence of work related to the scheduling of the HWF, a complete algorithm has not been developed that can solve the scheduling problem in the face of uncertainty at the level of distribution of tasks by resources, taking into account the specific structure of both the tasks and the resources themselves. Therefore, the application of machine learning methods to solve the task of scheduling HWFs, which partly allow for taking into account the uncertainty directly during the execution of the application itself, is an urgent direction at this stage in the development of the scientific field of optimizing the planning of distributed applications.

Also, there is no generalized algorithms for solving the problem of scheduling for distributed stream data processing applications, and therefore there is a need to develop computational optimization algorithms for different platforms and for the stream optimization problem in general. The development of methods for organizing the execution of iterative distributed applications is relevant, especially for solving scientific problems in supercomputer environments.

3 PROBLEM STATEMENT

Scheduling of HWF based on the implementation of the interaction between the modes is presented. For computational modes, individual approaches were applied that considered the specifics of each of the modes. All methods are based on the classical statement of the scheduling problem — the distribution of computational tasks over resources of a computer system. As a result, the scheduling of HWF is a combined scheduling problem, which consist of batch, stream and iterative scheduling problems with their specifics.

3.1 Batch

In the case of the batch mode, the task of workflow scheduling in the classical form is solved. For the statement of the scheduling problem, the following are introduced: a computational load model; computing environment model; optimization criteria and performance models for evaluating criteria.

For the statement of the scheduling problem, the following entities are introduced: a computational load model; computing environment model; optimization criteria and performance models for evaluating criteria. The computational load is presented in the form of a directed acyclic graph (DAG) $Wf =<$

$V,E>$, where $V = \{v_i\}$ – tasks, a $E = \{e_{i,i'}\}$ – data edges. Each task represents a computational model or application to be executed. Set of computation resources is $R = \{r_l\}$. Schedule is an ordered distribution of tasks by resources in the form $S = \{(v_i, r_l)\})$. Taking into account all the criteria $C = \{c_q(S)\}$, we can formulate a definition of the problem. The main goal is to find such an optimal schedule S_{opt}, that:

$f(S) = \sum_{(c_q \in C)} \omega_q c_q(S)$,

$f(S) \rightarrow \min_{S \in \mathbb{S}} f(S)$,

where ω_q – weights that determine the degree of importance of each criterion and normalize them and $\mathbb{S}$ is a set of all possible schedules. Criteria for optimization include: runtime; cost of resources; reliability of execution. The execution time of v_i on a particular node r_l is $ET(v_i, r_l)$ and data transfer $TT(e_{i,i'}, b_{i,i'})$ are estimated using performance models that can be constructed by analytical or statistical methods.

3.2 Stream

Scheduling for streaming data processing has several differences from scheduling in batch mode. Despite the fact that a streaming application can also have a graph structure of software components that perform various procedures on incoming data, the data arrives continuously and with dynamic intensity. Moreover, in the case of batch mode, the adjacent tasks of the computational graph are performed sequentially, while in the streaming case, all tasks of the computational graph must operate simultaneously and continuously.

An application for streaming data processing is a computational graph, where the vertices are computational operators, and the edges represent the transfer of data between operators. Such a graph is an abstract structure, operators are logical units. For direct data processing and scaling, a certain number of instances of operators (tasks) are created. Data flows continuously in the form of tuples through instances of the application graph operators. A tuple is a logically complete unit of data, i.e. can be atomically processed by the operator. Tuples can be different both in volume and in type.

The computing environment contains a set of nodes (hosts) $H = \{v\}$. Every node v - set of computing characteristics or resources allocated for computing tasks: $v = (cpu_v, ram_v, gpu_v, ...)$. Based on the resources of a node, its performance is determined in flops (float operations per second). It is assumed that tasks hosted on the same host share their resources by a fraction of their workload density. The proposed model is based on the use of cloud resources, where computing environment H can be changed during the planning process.

The computational load consists of many composite streaming applications, each of which has a set of operators. Thus, the computational load: $W = (O, E)$ is a graph, where graph nodes are operators $O = \{O_i\}$ streaming applications with dependencies set between $E = \{e_{i,j}\}$. The edge $e_{i,j}$ means a logical relationship between parent operator O_i and child operator O_j. Parent operator can be determined as $par(O_i) = (O_j \in O \forall e_{j,i} \in E)$. Similarly, the set of child operators is determined by the function $child(O_i) = O_j \in O \forall e_{i,j} \in E$. The following performance model functions are defined for each operator O_i: $RR_i(s)$ - a function that estimates the amount of resources r for the complete processing of input tuples s per unit time; $IP_{i,k}(s, r)$ - the number of incoming tuples of data received by the operator from the parent's operator $O_k \in par(O_i)$, which will be processed depending on the total number of input tuples s and the set of resources assigned to the operator; $OP_{i,j}(n)$ - the number of outgoing tuples received by the descendant operator depending on the number of processed tuples.

Schedule $S = (W', H', A)$ consist of a computational load configuration, configurations of the computing environment and distribution of operators instances among nodes. A Schedule $A = [(o_j, v_c)]_{j=1}^{|o|}$ is a set of pairs task o_j and computing node. $G = g_i(s)$ is a set of optimization criteria, which should be maximized. $C = c_i(s)$ - various boundaries, which should be performed during the optimization and schedule construction. A Schedule problem for stream data processing can be presented as:

$\sum_i^{|G|} \alpha_i g_i(S_{opt}) + \sum_i^{|C|} \beta_i c_i(S_{opt}) \geq \sum_i^{|G|} \alpha_i g_i(S') + \sum_i^{|C|} \beta_i c_i(S')$,

where α and β are used to determine the significance of a particular criterion or restriction.

3.3 Iterative

Distributed iterative applications consist of many instances of a computing package, each of which starts with its own set of parameters. Between instances of the computing package, interaction is performed at each iteration of the application through data exchange. To optimize the scheduling of iterative distributed applications, a graph performance model has been developed. The main idea of this model is based on the developed graph model for computing load allocation. The graphical allocation model is based on the graph of the internal structure of the application (consisting of blocks and the links between them) and the placement (destination) across containers and resources. Let $W = \{w_g\}$ be the computational load for the current iteration for each block of application g. h_l

is a performance of resource r_l. The model, describing the specific location, is used to: analyze and predict the performance of a distributed application; allocation changes by moving blocks between resources and containers. A cost function has been defined for moving from schedule S_1 to S_2:

$$f(S_1, S_2) = \sum \frac{d_{g1,g2}}{b_{l,l'}} \delta_l^{l'}$$

where: $\delta_l^{l'}$ is data volume to be transmitted from v_{g1} to v_{g2}, $b_{l,l'}$ denotes the data transfer speed. Function for estimating simulation time on one iteration:

$$T(S) = \max_l \left(\sum_j \frac{w_{j,j'}}{h_l} + \sum_j \sum_{j'} \frac{d_{j,j'}}{b_{l,l'}} \delta_l^{l'} \right)$$

Then the condition for the transition between schedules is determined:

$$f\left(S_{prev}, S_{new}\right) < \left(T\left(S_{prev}\right) - T\left(S_{new}\right)\right)\theta,$$

where θ is a statistically dependent value representing the rate of change of the computational load among the vertices of the computational grid.

4 METHODOLOGY

4.1 Batch NNS Algorithm

Developed scheduling algorithm Neural Network Scheduler (NNS) is based on Reinforcement Learning technique - Q-learning. Q-learning is a model-free RL technique, that learns a policy telling an agent what action to take under what states of environment. The policy determines the choice of action a from actions set A. Action at moment t is chose based on Q-function $Q(s,a)$, which is represented all reward from at this moment to the end of an episode, using rule $a_i = argmax_a(Q_t(s,a))$. Then agent do chosen action in the environment and gets new state of the system s_{t+1} and reward $r(s_t,a)$. During the process the agent accumulates data in the form of the established $s_0a_0r_0s_1a_1r_1s_2$.... Q-function is updated in accordance with the SARSA principle: $Q(s_t,a_t) = Q(s_t,a_t) + \alpha[r_t + \gamma Q(s_{t+1},a_{t+1}) - Q(s_t,a_t)]$, where α - learning rate, γ - discount coefficient.

Reinforcement learning allows you to move from classic machine learning with establishing the dependencies of the output data from the input to building a controller that establishes the relationship between system states and actions responding to states. Controller training is carried out due to interaction with the external environment (computing infrastructure) and already gained experience or historical system monitoring data.

The NNS algorithm is proposed in (Melnik and Nasonov, 2019). The NNS algorithm is an agent that is designed to effectively plan tasks for computing nodes. The main part of the agent is a neural network, which allow to evaluate and select the currently most beneficial task assignment to a resource (action). The network architecture is presented in such a way that it takes up to m tasks and n computing resources. The selected tasks and resources are encoded in vector parameters - the input state, taking into account the state of the computing environment, the structure of the workflow and the current schedule. The vector of parameters consists of several categories: general parameters of the workflow; task parameters; resource settings; parameters for the tasks set on the resources. This architecture of the scheduler allows you to perform dynamic planning without reference to the dimension of the problem (the number of tasks HWF). The main operating objects of the NNS algorithm are presented in Fig. 1.

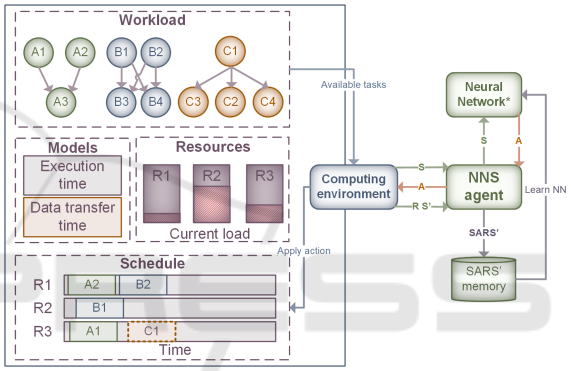

Figure 1: NNS scheduling algorithm scheme based on reinforcement learning.

The agent interacts with the computing environment, receiving from it the current state, which collects information about the load on computing resources, workload, current schedule, and performance models to evaluate task performance.

Since HWFs differ from each other, HWFs of the same structure may differ in input data and scale, because of it the relative values of parameters should be used. To go to the relative values of the parameters, we determine the theoretically worst execution time $WT(WF)$. The worst execution time is estimated as the sum of the execution time of all tasks assigned to the weakest resource in terms of performance and the time required to transfer all data through the network, regardless of the need for these transfers. This allows you to train a neural network based on HWF of any dimension, but at the same time to identify HWFs that are similar in structure and apply the experience gained during training to plan them.

The Reward function from the environment is determined as $r(s,a) = \frac{WT}{CT}\frac{i_s}{n}$, where CT - current schedule time, i_s - number of already scheduled tasks and

n - total number of tasks in the CA. In the case of a terminal state, the reward is equal to the ratio of the current execution time of the CA to its worst theoretical time.

4.2 Stream SSGA Algorithm

An evolutionary algorithm for scheduling of distributed streaming applications SSGA (Stream Scheduling Genetic Algorithm) imitates the evolutionary process, the scheme of which is presented in Fig. 2a. Fig. 2b shows an example of a graph of a streaming application, and figure 2c shows an example of representing a solution in the framework of the developed SSGA algorithm. The SSGA algorithm is growing a population of solutions that represent schedules for placing streaming applications across nodes of a computing environment. The fitness function is used to assess the quality of each solution in accordance with the specified optimization criteria and limitations. The SSGA algorithm was proposed in (Melnik et al., 2018).

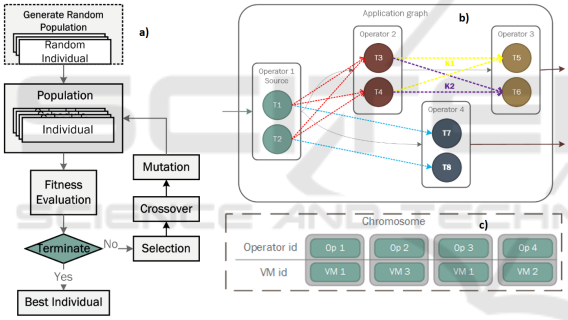

Figure 2: Scheme of the genetic algorithm (a) for stream data scheduling processing (b) and example of GA chromosome (c).

Further, a modification DSSGA was developed to solve the problem of scheduling of streaming data processing in cloud environments (for example, Amazon EC2), taking into account the possibility of launch and terminate computing nodes to ensure scalability and elasticity of calculations. Fitness function $f(S)$ is a set of criteria and restrictions: the cost of nodes $\gamma(S)$; penalty for exceeding the window for delaying the execution of data tuples $\rho_{lat}(S)$; penalty for invalid decisions $\rho_{val}(S)$; number of allocated kernels for operators; the overhead of the transition to the new schedule σ(S). Developed SSGA and DSSGA allow efficient scheduling due to a unique approach to the problem of stream data scheduling by predicting the intensity of incoming data into the system, including under conditions of uncertainty. The proposed model of streaming data processing is applicable for most popular platforms for distributed processing of streams.

4.3 Iterative. IMSGA Algorithm

The iterative computing organization algorithm is developed based on monitoring instances of a distributed iterative application IMSGA (Iterative Multiagent Scheduling Genetic Algorithm). The scheme is based on a multi-agent approach, where each intelligent agent is responsible for a certain area of computing nodes and provides its own performance models for the iteration time for its area. The master agent plans the placement of application instances by agents, receiving developed performance models from the agents themselves. During the execution of the iterative application, the master agent performs dynamic rescheduling to account for changes in the dynamics of the computational load in the selected areas of calculations. The IMSGA algorithm is proposed in (Melnik et al., 2019).

The Algorithm IMGSA have four stages. The first stage of the scheme is responsible for managing running applications. The second stage is responsible for forming virtual environment. The third stage is a distributed two-level intelligent algorithm with a high-level central core of resource quotas and a multi-agent collaborative level of settlement management. The fourth stage in the diagram is the level of a heterogeneous computing environment, which provides computing resources for calculations. The algorithm is aimed at the distribution of the cells of the modeling domain by computing resources, behind which are intelligent agents. The simulation area is given by a grid of size nxn, and there are m intelligent agents over which these cells need to be distributed. An example of a genotype and its corresponding distribution is presented in the Fig. 3.

The mutation operator moves randomly selected centers, adding normally distributed values. A two-point crossover selects centers for agents from two parents. The selection is carried out by a tournament with three participants. Each agent has its own performance model. To evaluate the fitness function, a model is launched that estimates the application modeling time using the constructed performance models. This allows the user to predict the computational load for subsequent iterations for each intelligent agent. The fitness function aggregates the simulation time of all iterations in accordance with the simulation time for each intelligent agent.

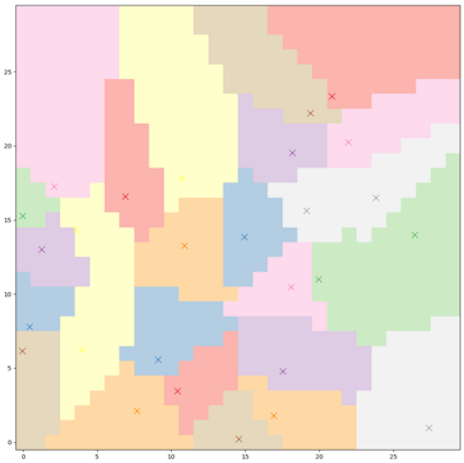

Figure 3: An example of a solution generated by a genetic algorithm in the case of modeling population mobility.

4.4 Hybrid Scheme Development

Based on the developed methods and algorithms, a generalized algorithm for the organization of the NWF is constructed, which allows for the interaction between computational modes. The developed scheme is presented in Fig. 4.

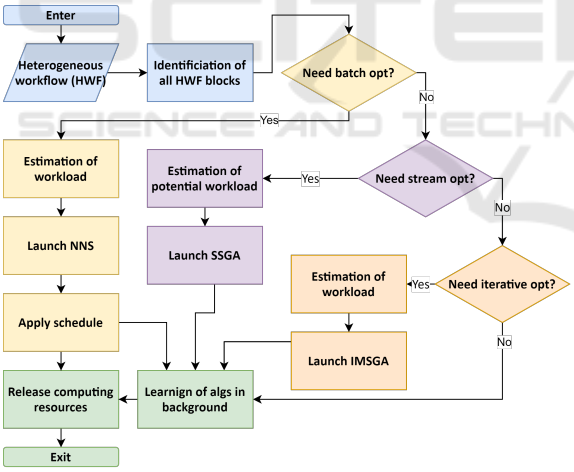

Figure 4: General scheme for organizing heterogeneous computing.

The input data is information about the received computational load in the form of the HWF structure with the characteristics of its constituent elements. The first step in organizing computations is to identify the type of application, its structure, and the structure of its blocks. Identification is based on meta-information about the components. The first level is batch scheduling for the entire HWF. To schedule batches, basic metaheuristics and heuristics are launched, the solutions of which are added to the solution repository. The developed NNS algorithm for scheduling batch data processing uses this repository in the process of training a neural network and carries out the placement of tasks for execution. If the blocks of a heterogeneous composite application are a streaming or iterative application, the planning of these packages is carried out by SSGA and IMSGA, respectively.During the execution of iterative blocks, monitoring of the load and redistribution of resources occurs if necessary. Similarly to the iterative mode, in the process of performing streaming data processing, the system is monitored and the intensity of data streams is predicted. The developed algorithm allows scheduling HWF by identifying the type of blocks included in the HWF and dividing the global HWF scheduling task into sub-tasks: batch, streaming, and iterative scheduling.The division into subtasks allows us to apply individual approaches for each mode and thereby take into account the specific features of each computational mode to build an effective schedule in conditions of uncertainty and incompleteness of information.

5 EXPERIMENTAL STUDY

5.1 Batch NNS Algorithm

We experimentally studied the ability of the developed NNS algorithm to learn and find the relationships between the proposed parameters of the vector representation of the planning problem, as well as make high-quality schedules comparable to existing analogues. The experiments were carried out on the basis of a simulation model and well-known test WFs (Montage, CyberShake, Epigenomics) provided by the workflow management system - Pegasus.

For comparison, the DQTS algorithm was implemented. The algorithm was chosen as the most promising and closest in concept (the use of reinforcement learning) among existing algorithms at the time of the study.

The results of the algorithm launches are presented in Table 1. WFs are presented in the form of workflow type (for example, Montage) and the number of tasks in WF. The results include the total execution time of the test WF and the algorithm reward for constructing the schedule. A reward is the ratio of the theoretically worst time to the found execution time. According to the results of the experiments, the developed NNS algorithm received solutions better than DQTS by 20% on average in terms of the execution time of test WFs.

Table 1: Comparison of CA scheduling results between NNS and DQTS algorithms.

CA	DQTS time, s	DQTS reward	NNS time, s	NNS reward	Improve, %
Montage50	256	2.9	206	3.8	22
Montage100	406	4.1	369	4.5	9
CyberShake50	995	2.9	807	3.6	19
CyberShake100	1550	3.4	1315	4.0	15
Inspiral50	6111	2.7	4104	4.0	33
Inspiral100	10314	2.8	7061	4.2	32
Epigenomics46	24530	2.4	19979	2.9	19
Epigenomics100	192851	2.9	170608	3.3	12

5.2 Stream SSGA Algorithm

The purpose of the experiment is to study the effectiveness of the developed SSGA algorithm. The experiments were based on the resources of the AWS cloud platform and the developed simulation model for evaluating the quality of solutions. A comparison was made of the results of the developed SSGA with the RStorm algorithm. The result of the experiments is presented in Figure 7 for the schedules built by RStorm and SSGA. Only four nodes were used in the SSGA schedule instead of the five nodes used in the RStorm schedule. Due to the optimal combination of peak and non-peak intervals of the intensities of the input data streams, the algorithm allows you to save nodes and improve resource utilization.

The developed SSGA is able to find solutions with fewer resources used (1.6 times better) and with a greater resource efficiency of 6.2% on average among experiments, even with a large measurement of the optimization problem (200 threads, 100 nodes). The results are described broader in the paper (Melnik et al., 2018).

5.3 Iterative IMSGA Algorithm

For the experiment, a scenario was developed for the movement of agents in such a way as to create an explicit dynamics of the computational load on different parts of the computational domain of the application. The aim of the experiment was to study the ability of the IMSGA algorithm to dynamically adapt the iterative application execution schedule in the process of changing the load. The experiments are described in the paper (Melnik et al., 2019).

The experimental results show that when using a uniform load allocation schedule, the default simulation time is 308 s. Schedules generated by the expert reduce execution time by only 9 seconds. When the developed execution scheme and the developed IMSGA were applied, without predicting the computational load, the average simulation time was 173 s and 158 s for cases with 5 and 10 dynamic schedules, respectively. The best results were obtained when predicting the density of the computational load for each intelligent agent. Experiments show that the algorithm is able to adapt to changes in the application. The best result is 137 s (10 times with forecasting) 55% faster in compare with default uniform schedule. In the case of predict10, the load between the computing nodes was distributed as evenly as possible.

5.4 Heterogeneous Workflow Scheduling

Scenario of Heterogeneous Workflow for Snow Cleaning Plans Generation in a City based on Data from Social Media. For experimental studies of optimizing the performance of HWFs, a scenario has been developed to build a snow removal plan through multi-agent modeling using social media data analysis. The essence of the project is the development of a multifunctional city platform for the analysis of various public and communal problems of the city based on data from specialized city resources and social media data. In particular, one of the problems is the development of an optimal plan for snow removal and its delivery to stationary snow melting and snow receiving points (with more than 100 main contractors and a total budget of around two billion rubles).The structure of application is presented in Fig. 5. The subtask of this project was the task of optimizing the performance of computing processes. Within the framework of the project, a prototype NKP was developed, which organizes all stages of data processing. In particular, the HWF includes:

- collection and preprocessing of data from social media (streaming mode, Amazon platform cloud resources);
- filtering and classification of the collected data to build a scenario for an iterative unit for modeling population mobility (batch mode, ITMO University resources);

- launch of an application for modeling population mobility in an iterative mode (iterative mode, resources of the Lomonosov supercomputer);
- analysis of the results of the iterative block for the construction of routing plans for snow removal vehicles for different areas of the city and aggregation of results (batch mode, ITMO University resources).

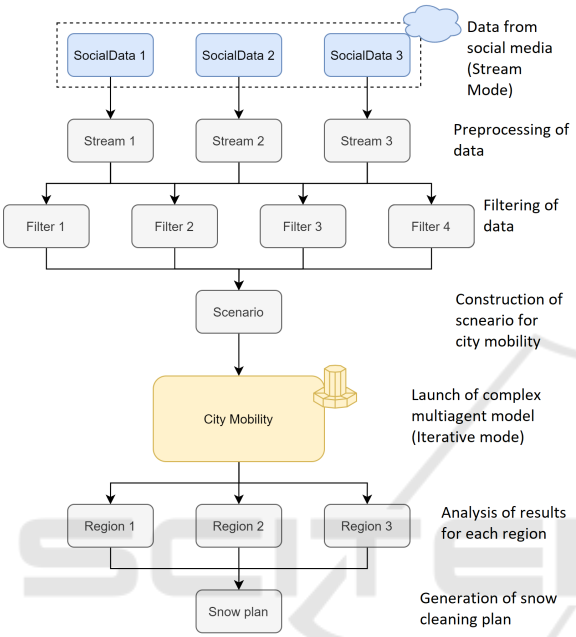

Figure 5: Scheme of a heterogeneous workflow for generation a snow cleaning plan based on data from social media.

Results. During the experiments, heterogeneous resources were used (supercomputer, cloud computing, clusters of ITMO University) and interaction between them was organized on the basis of the developed software infrastructure for planning HWF.

To conduct experimental studies on optimizing of a heterogeneous composite application, a scenario was developed for estimate a snow removal schedule based on an analysis of data from social media. As an experiment, two execution scenarios were compared. The first scenario (DEV) is based on developed methods: NNS, SSGA, IMSGA. The second scenario (ALT) uses the alternative algorithms : DQTS, R-Storm.

The first step of experimental scenario was optimization of streaming applications, which are processing data from social media. The comparison between SSGA and R-Storm result schedules are presented in Fig. 6 and Fig. 7. Algorithms scheduled a streaming workload acrossavailable cloud resources. The final load on resources at time points is represented by a red bold curve in the subplots. When the total load exceeds the available node load (500 tuples / s), a queue of tuples is accumulated, which will be processed at moments of lower resource load intensity.

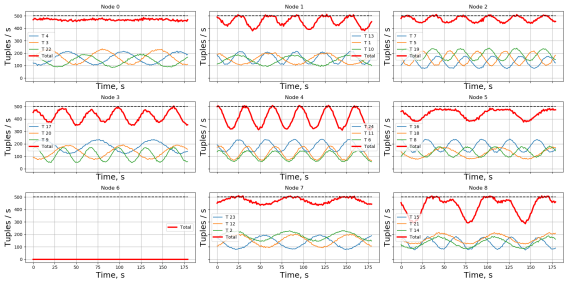

Figure 6: SSGA result schedule for stream mode.

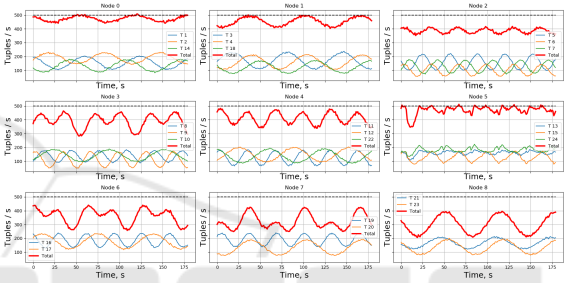

Figure 7: R-Storm result schedule for stream mode.

The second stage of the experiments is the planning of all blocks of HWF in batch mode, except for the block with iterative mode (City Mobility). The results of batch scheduling are presented in Fig. 8 for the DQTS (a) and NNS (b) algorithms. Tasks HWF's batch part were allocated across 4 computing resources with different capacity (R1 - worst node, R2 and R3 - medium, and R4 is the most performance node).

The last stage of optimization was planning of iterative block. IMSGA was compared with the basic application execution mode. The input for the block is a scenario of the movement of people in the city based on pre-processed and filtered data obtained by collecting social media data.

Based on the results of scheduling of the HWF including blocks in streaming, iterative and batch modes, Table 2 which presents the results of a comparison of two scenarios of the experiment is compiled. As a result of comparing two scenarios of HWF optimizing, the scenario (DEV) allows us to obtain a schedule with 12% less cost of cloud resources, and with the 32% less total execution time of the HWF in compare to the scenario based on alternative methods and algorithms (ALT).

Table 2: Comparison of scheduling results of HWF for snow removal plan.

Experiment	Streaming mode		Iterative mode	Batch mode	Result
	Resources cost, CPU	Avg. util., %	Exec. time, s	Exec. time, s	Total time, s
DEV	1,53	0,004	253	227	480
ALT	1,73	0,106	402	301	703
Improvement, %	12	96	37	25	32

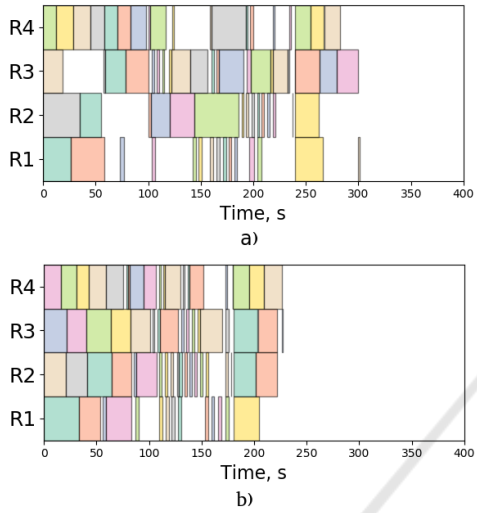

Figure 8: Result schedules of HWF performed by DQTS (a) and NNS (b).

6 CONCLUSION

In this work we formed a conception of heterogeneous workflows that are composed of computing modes: batch, stream, iterative. Each mode has its own specifics for execution in distributed computing environments. Further, we proposed a hybrid scheduling scheme that allows us to organize such a complex computation that compose execution of workflows, streaming applications and iterative multiscale models. The hybrid scheduling scheme include optimization algorithms for each of computing node. Scheduling algorithms based on reinforcement learning (batch NNS algorithm), and genetic algorithms for stream and iterative modes (SSGA and IMSGA respectively).

In the course of experimental studies of the developed families of algorithms, it was established that NNS, based on reinforcement learning, is able to learn how to solve the planning problem and build schedules with an average performance increase of 20% among the used WFs, compared to the similar DQTS algorithm. The developed SSGA streaming data processing scheduling algorithm is able to use 1.6 times fewer compute nodes to fully process data streams compared to the most famous RStorm algorithm. Due to the developed algorithm for optimizing iterative distributed applications, it was possible to reduce the execution time of the application for population mobility by 55%. The results of the HWF scheduling experiment based on the developed approaches (NNS, SSGA, IMSGA) allowed us to reduce the HWF execution time by 32% compared to alternative methods (R-Storm, DQTS) and reduce the cost of cloud resources for organizing data streaming by 12%.

ACKNOWLEDGEMENTS

This research is financially supported by The Russian Science Foundation, Agreement #17-71-30029 with co-financing of Bank Saint-Petersburg.

REFERENCES

Agarwalla B., e. a. (2006). Streamline: a scheduling heuristic for streaming applications on the grid. In *Multimedia Computing and Networking*.

Borgdorff J., e. a. (2014). Performance of distributed multiscale simulations. In *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences.*, page 372.

Choudhary, A., Gupta, I., Singh, V., and Jana, P. K. (2018). A gsa based hybrid algorithm for bi-objective workflow scheduling in cloud computing. *Future Generation Computer Systems*, 83:14–26.

Hagras, T. and J., J. (2003). A simple scheduling heuristic for heterogeneous computing environments. In *Proceedings of the Second international conference on Parallel and distributed computing.*, page 104–110.

Hussain A., e. a. (2016). A survey on ann based task scheduling strategies in heterogeneous distributed computing systems. In *Nepal Journal of Science and Technology.*, pages 69–78.

Ismayilov G., T. H. (2020). Neural network based multiobjective evolutionary algorithm for dynamic workflow scheduling in cloud computing. In *Future Generation Computer Systems.*, pages 307–322.

Masdari, M., Salehi, F., Jalali, M., and Bidaki, M. (2017). A survey of pso-based scheduling algorithms in cloud computing. *Journal of Network and Systems Management*, 25(1):122–158.

Melnik, M. and Nasonov, D. (2019). Workflow scheduling using neural networks and reinforcement learning. *Procedia Computer Science*, 156:29–36.

Melnik, M., Nasonov, D. A., and Butakov, N. (2018). Scheduling of streaming data processing with overload of resources using genetic algorithm. In *IJCCI*, pages 232–241.

Melnik, M., Nasonov, D. A., and Liniov, A. (2019). Intellectual execution scheme of iterative computational models based on symbiotic interaction with application for urban mobility modelling. In *IJCCI*, pages 245–251.

Nagar, R., Gupta, D. K., and Singh, R. M. (2018). Time effective workflow scheduling using genetic algorithm in cloud computing. *International Journal of Information Technology and Computer Science*, 10(1):68–75.

Peng B., e. a. (2015). R-storm: Resource-aware scheduling in storm. In *Middleware 2015 - Proceedings of the 16th Annual Middleware Conference.*, pages 149–161.

Pollard S.D., e. a. (2019). Evaluation of an interference-free node allocation policy on fat-tree clusters. In *Proceedings - International Conference for High Performance Computing, Networking, Storage, and Analysis.*, pages 333–345.

Rashmi S., B. A. (2017). Q learning based workflow scheduling in hadoop. In *International Journal of Applied Engineering Research.*, page 3311–3317.

Subedi P., e. a. (2019). Stacker: An autonomic data movement engine for extreme-scale data staging-based in-situ workflows. In *Proceedings - International Conference for High Performance Computing, Networking, Storage, and Analysis.*, pages 920–930.

Tong Z., e. a. (2019). A scheduling scheme in the cloud computing environment using deep q-learning. In *Information Sciences.*

Topcuoglu H., Hariri S., W. M. (2012). Performance-effective and low-complexity task scheduling for heterogeneous computing. In *IEEE Transactions on Parallel and Distributed Systems.*, page 260–274. IEEE Press.

Vukmirovič S., e. a. (2012). Optimal workflow scheduling in critical infrastructure systems with neural networks. In *Journal of Applied Research and Technology.*, pages 114–121.

Xiao Z., e. a. (2017). Self-adaptation and mutual adaptation for distributed scheduling in benevolent clouds. In *Concurrency Computation.*

Xu J., e. a. (2014). T-storm: Traffic-aware online scheduling in storm. In *Proceedings - International Conference on Distributed Computing Systems.*, pages 535–544.

Yao J., Tham C.K., N. K. (2006). Decentralized dynamic workflow scheduling for grid computing using reinforcement learning. In *IEEE International Conference on Networks*, pages 90–95.