МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
имени М.В. ЛОМОНОСОВА

Факультет
вычислительной математики
и кибернетики

ПРОГРАММНЫЕ СИСТЕМЫ И ИНСТРУМЕНТЫ

Тематический сборник

№ 22

МАКС Пресс

МОСКВА — 2022

МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
имени М. В. ЛОМОНОСОВА

Факультет вычислительной математики и кибернетики

ПРОГРАММНЫЕ СИСТЕМЫ И ИНСТРУМЕНТЫ

Тематический сборник

№ 22

Под общей редакцией
чл.- корр. РАН, профессора Р. Л. Смелянского

МАКС ПРЕСС

Москва – 2022

УДК 519.6+517.958
ББК 22.19
П75

https://elibrary.ru/zabkdl

Печатается по решению Редакционно-издательского совета факультета вычислительной математики и кибернетики МГУ имени М.В. Ломоносова

Программные системы и инструменты : Тематический сборник / Под ред. Смелянского Р.Л. – М: Издательский отдел факультета ВМК МГУ (лицензия ИД №05899 от 24.09.2001г.; МАКС Пресс, 2022, №22. – 160 с.

П75

ISBN 978-5-89407-629-4 (ВМК МГУ имени М.В. Ломоносова)
ISBN 978-5-317-06895-0 (МАКС Пресс)
https://doi.org/10.29003/m3106.978-5-89407-610-2

Данный выпуск сборника составлен по материалам работ студентов и аспирантов, получивших рекомендации научного семинара кафедры Автоматизации Систем Вычислительных Комплексов. Редколлегия сборника продолжает традицию его издания в память о первом руководителе кафедры АСВК Королёве Л.Н. В предлагаемом читателю тематическом сборнике публикуются статьи, посвященные проблемам современных компьютерных сетей, методам планирования вычислений и балансировки вычислительной нагрузки, задачам и алгоритмам дискретной оптимизации, методам кодирования и визуализации информации.

Статьи сборника будут интересны студентам, аспирантам и специалистам в области разработки прикладных программных систем.

Ключевые слова: Информационно-телекоммуникационные технологии, машинное обучение, нейронные сети, блокчейн, программно-конфигурируемые сети, сетевое кодирование, планирование вычислений, балансировка нагрузки, облачные вычисления, эволюционные алгоритмы, имитационное моделирование, сбор медицинской информации, Интернет вещей, мультиагентные системы, визуализация сетевой среды, анализ выборок..

УДК 519.6+517.958
ББК 22.19

ISBN 978-5-89407-629-4
ISBN 978-5-317-06895-0

Оглавление

СБОРНИК
«Программные системы и инструменты»

От редколлегии:

Тематический сборник “Программные системы и инструменты” был инициирован Львом Николаевичем Королевым в 2000 году, как площадка, где студенты и молодые ученые могли бы публиковать свои достижения. Все эти годы Лев Николаевич неустанно вел его, отбирал публикации, редактировал. Редколлегия сборника продолжает эту традицию в память об этом выдающимся человеке.

Этот выпуск сборника составлен по материалам работ студентов, получивших рекомендации научного семинара кафедры Автоматизации Систем Вычислительных Комплексов, которую Л.Н. Королев создал и бессменно руководил.

Редколлегия

Pluzhnikova D. R., Antonenko V. A., Sakharov F.V.

BORIS: Prototype of BGP routing system based on blockchain technology

Introduction

The Internet is now divided into subsystems, these components are nevertheless built on a centralized architecture. This leads to security issues, such as trust issues [1]. For example, because the system is built on the trust of the system to the central node, it will be enough for an attacker to gain access to this node and then the attacker will own the entire system. Now BGP is the main routing protocol on the Internet. The routing information is very important, that's why BGP requires TCP-like reliability. Once a TCP connection is established, network nodes start their communication by exchanging specific BGP messages to establish a session. After the BGP neighbors have moved to the stable state of ESTABLISHED, which means that the correct version of BGP is running and all settings are consistent, they proceed to exchange route information. For this purpose, the UPDATE messages are used. Each UPDATE message can contain information about one new route or about deleting a group of old routes. Before saving the new path to the routing table, the latter passes through the filters defined by the AS routing policies.

BGP does not provide confidentiality, and BGP messages can be replayed [2]. This means that an attacker can resend the UPDATE message, which causes an invalid route to be present in the router's BGP routing table. In addition, no mechanism provides authentication of the origin of messages for BGP. In this case, attackers can impersonate one of the BGP nodes, so this approach can be used to introduce non-existent routes by using BGP message interception and analysis [5].

Consider the following attack as an example: BGP uses an algorithm of several steps, thanks to which the protocol selects only one path and places it in the local routing table. There are many ways to influence this choice. For example, consider the `AS_PATH` attribute (an attribute that describes which autonomous systems you need to go through to reach the destination network). It consists of a sequence of numbers, each autonomous system adds its AS number to the UPDATE message in the order of the queue, and the router, all other things being equal, will choose the shortest route.

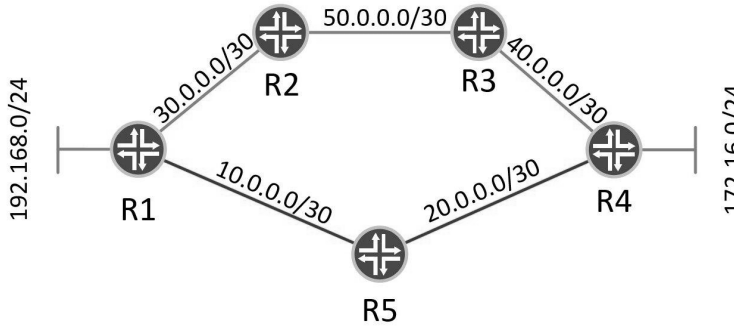

Fig 1. Network topology. The optimal route through R1 to R4 goes through R2 and R3

In Fig 1, R1 and R4 act as border provider routers, both routers announce networks 192.168.0.0/24 and 172.16.0.0/24, respectively, from their AS 1 and 2. Each provider has an agreement with the backbone providers (R2 and R3) that R1 and R4 will drive traffic through them, and AS 2 and 3 will provide the necessary bandwidth, reliability and protection. Also in this scheme there is an R5 provider. By changing the length of the AS path, the AS, which includes the R5 router, can become a transit AS for traffic between AS 1 and 2, pass all traffic through itself, which can lead to both traffic interception and a DDOS attack on a system that is not ready for additional loads.

One of the main problems with BGP is that the path information in the received message must be trusted. Thus, the announcement of this information must be confirmed, for example, by an authoritative AS, that is, an AS that we can trust. However, BGP does not provide an authoritative hierarchy that allows a malicious or compromised AS to create and advertise and announce new non-existent or malicious routes.

In this regard, many BGP security solutions have been developed, such as SecureBGP (S-BGP) [3], Secure Origin BGP (SoBGP) [4], and Cross-Domain Route Verification (IRV) [5]. The disadvantage is that they each depend on a central, trusted node, which is enormous management overhead and complex.

1. Related work

There are many mechanisms for securing BGP. We will consider the most competitive protocols.

A. `Secure-BGP (Secure Border Gateway Protocol)`

Secure-BGP (Secure Border Gateway Protocol) Secure Border Gateway Protocol (S-BGP) is a modification of BGP designed to address security concerns. S-BGP uses digital signatures and X.509 certificates to create and validate BGP UPDATE messages advertised by AS [3]. Secure-BGP is based on three different security mechanisms that seek to meet the security requirements of BGP. The S-BGP architecture uses security elements such as public key infrastructure (PKI), evidence, and Internet protocol security (IPsec).

S-BGP eliminates many of the security issues in BGP and provides security for the exchange of messages between ASs. However, it is susceptible to UPDATE message replay attacks. In addition, an attacker could remove signatures from messages, and further ASs would be unable to verify them. One of the biggest problems with S-BGP is its complexity. This protocol is expensive to adopt and performance is degraded by the evidence and signature calculation for each UPDATE message. In addition, S-BGP has some storage requirements for route evidence, which makes it even more difficult to use.

S-BGP provides guarantees for secure BGP communications. Exchange paths and IP prefixes are verified and proven to be valid in destination autonomous systems using evidence. In addition, the integrity and confidentiality of updates are achieved through the use of the IPSec protocol. However, the high security of S-BGP requires a trade-off. Namely, its evidence storage requirements and performance degradation due to signature verification in every UPDATE message could discourage ISPs from deploying it to their ASs.

B. `SoBGP (Sender Secure Border Gateway Protocol)`

Secure Origin Border Gateway Protocol (soBGP) is a modification of BGP designed to address BGP security issues and improve communication reliability between peers. Like S-BGP, it uses PKI to authenticate and authorize objects on the network [4].

soBGP relies on several mechanisms, mainly certificates, to provide various security services for cross-domain routing. soBGP uses four types of certificates to verify peer prefixes and paths:

- Authentication certificate

- Authorization certificate
- Prefix Policy Certificate
- AS Policy Certificate

soBGP does not address all BGP security concerns. It has much weaker path authentication compared to S-BGP. The main difference between the approaches used by soBGP and S-BGP is that S-BGP provides dynamic path checking. This means that S-BGP speakers can view the topology and path of a message in real time. In contrast, soBGP uses databases that provide a static view of the topology and the paths within it. The UPDATE message received may have had a path from the changed topology that has not yet been reflected in the soBGP database. Topology changes can be accommodated by reapplying ASPolicyCert.

soBGP requires the database to be deployed on the AS side, certificate propagation can cause some deployment problems. The fact that there is an additional SECURITY message type can cause some difficulties in protocol deployment and backward compatibility.

soBGP provides a more flexible and lightweight solution than S-BGP for solving BGP security problems. The protocol can be tuned in such a way as to achieve a trade-off between security and cost. However, it does not provide path integrity, good source authentication, and an attacker can still invade the system. In addition, it has a weak protection mechanism for path authentication and PKI key distribution. Also, additional databases are required for path and policy validation, which complicates deployment.

C. IRV (Inter-Domain Route Verification)

Inter-Domain Route Validation (IRV) is the least centralized BGP security solution [4].

Each AS on the network contains an IRV server. Upon receipt of the UPDATE message, the BGP speaker will contact the IRV server in its AS to verify the correctness of the message received. The IRV server will verify the received request by requesting the IRV server from the AS referenced in `AS_PATH`. To check all ASs in `AS_PATH`, the IRV server will need to request information from all relevant IRV servers.

The main problem with IRV is that it must be able to connect to IRV servers in other ASs to check the path. This complicates the configuration and maintenance of the IRV server as it must communicate with other IRV servers.

In addition, deploying an IRV server requires a separate virtual machine or server that must be highly available and fault tolerant.

Unlike S-BGP and soBGP, IRV operations are independent of the routing protocol. Its security check is completely decoupled from BGP, allowing for more flexibility. However, there are some issues with deploying and maintaining the IRV server that might render this solution suboptimal for real-world scenarios.

D. `BGP-LSChain`

The router downloads IP prefixes from the blockchain network which it inserts into an update message. Before this step, the administrator must upload a list of IP prefixes for the router. Nodes in blockchain subsequently verify it. Also, it checks if there is the valid entry stored in the blockchain for every prefix of the list. This architecture includes only prefixes that are verified to be assigned to the AS of the router. Thus, routers add to their configuration only valid IP prefixes from the blockchain network and will advertise through BGP only legitimate information. This approach protects against administrator accidental misconfiguration or malicious appropriation of IP prefix of other AS. Nevertheless, it does not affect the dissemination of prefixes that have been learned from neighboring ASes [6].

The architecture consists of a set of smart contracts that represent participants and their operations in the RPKI model.

The advantage of this architecture is that the admin can track changes that happened in the past. Every modification of the configuration file of any device is stored in the blockchain. It is simplified to find out when the configuration file was changed and what was modified. The disadvantage of this architecture lies precisely in the presence of an administrator node, which makes the architecture more centralized.

E. `RouteChain`

RouteChain uses a bihierarchical blockchain structure and Clique consensus protocol to facilitate fast and tamper-proof route management [7].

While the blockchain ledger provides a validation source for all prefixes, Clique enables swift consensus among ASes over the nature prefix broadcast. Combined, these two properties enable RouteChain to act as a standalone security service that can be incrementally deployed in parallel with the current operations of ASes. We validate achievable objectives of RouteChain through discrete-event simulations, and our

results show that RouteChain can effectively curtail a BGP attack while it is in its initial stage.

The disadvantage of the system is the need to divide routers into groups. This algorithm is non-trivial and requires different approaches for different networks, and, accordingly, high implementation costs.

E. Summary

The Fig.2 and Fig.3 show the summary of the work of each of the considered protocols.

Approach	**Source Authentication**			
	Design	***Security***	***Expenses***	
			Time	***Memory***
S-BGP	Hierarchical. Local PKI memory used	High	Low	High
SoBGP	Hierarchical. A separate PKI database is used	High	Low	Low
IRV	Separate IRV servers	High	Low	Low
BGP-LSChain	Checking through the administrator	High	High	Low
Route Chain	Clique consensus protocol	High	Average	Average

Fig 2. Summary of the source authentication

Approach	Path Authentication			
	Design	*Security*	*Expenses*	
			Time	*Memory*
S-BGP	Signing a message	High	High	High
SoBGP	Topology map	Low	Low	Low
IRV	Database	Average	High	Low
BGP-LSChain	Blockchain system	Average	Average	Average
Route Chain	Blockchain system	High	Average	Average

Fig 3. Summary of the path authentication

There are tradeoffs in the protocols reviewed that are not optimal for all scenarios. For example, S-BGP and IRV have strong security, but there are some performance and deployment issues. On the other hand, soBGP authentication is easier to deploy but lacks some security features. When it comes to blockchain solutions, RouteChain has deployment issues, and BGP-LSChain is still somewhat of a centralized solution.

As a result, there is still room for improvement and development of a better, more reliable and safer solution.

2. Blockchain Substrate

The prototype of the blockchain-based system is being developed focused on eliminating the problems that arise with the BGP security modifications described above. It shows how the blockchain can be used to validate messages and how to distribute information between nodes safely and securely by decentralizing the system.

This section discusses the main tools used in the implemented solution, such as the Substrate blockchain framework developed by Parity [11].

Blockchain is a technology that organizes a system consisting of a chain of blocks, each of which contains information about the previous ones. It is stored on all computers of the system participants at the same time, and the connection between the blocks is provided not only by numbering, but also by the fact that each block contains its own hash sum and the hash sum of the previous block. Storing copies of the

blockchains on different computers independently makes it extremely difficult to change the information already included in the blocks, inasmuch as to change the information in one of the blocks, you will also have to edit all subsequent blocks.

You can also check the contents of the blocks because each block contains information about the previous one. All the blocks are arranged in a single chain, which contains information about all the operations that have ever been performed in the system. Wherein, the created block will be accepted by other users if the numeric value of the header hash is equal to or less than a certain number.

Each block, in turn, consists of transactions. A transaction is considered complete and valid if its format and signatures are verified, and the transaction itself is grouped with several others and recorded in a block. There are two types of transactions: signed and unsigned.

Signed transactions contain the signature of the account that issued the transaction, and must pay a fee for including the transaction in the chain. Since the value of including signed transactions in the chain can be determined before they are executed, they can be transmitted in the network between nodes with a low risk of spam. In some cases, unsigned transactions can also be used, but the logic behind verifying them can be complex. Since the transaction is not signed, the commission is not paid. Because of this, there is no economic logic in the transaction queue to prevent spam. Unsigned transactions also lack a one-time number, making it difficult to protect against replay.

The client-side API of any Substrate-based blockchain allows a user to subscribe to a feed of events that take place in the runtime as the chain makes progress. Thus, Substrate notifies the user of a certain change in the state of the chain. The Substrate runtime module can generate events when it wants to notify external objects of changes or conditions in the runtime. The developer determines what events are generated in his module, what information is contained in these events, and when these events are generated.

3. Quagga BGP

Quagga [12] consists of several separate programs (daemons), each of which performs a specific function. For example, the bgpd daemon implements the BGP protocol. The main role in Quagga is played by the zebra demon. It receives route information from daemons that implement specific protocols and selects the best routes obtained from these sources. After that, these routes are passed to the Linux kernel, which transmits user traffic.

Each individual route in the routing table can be represented as a prefix with which different route information is associated. The Quagga routing table is simply a set of such prefixes with additional information associated with them. The zebra daemon stores all routes that have been passed to it or have been configured in the zebra itself. Storing all routes allows you to quickly select a new best route if for some reason the current best route no longer exists.

When a new route is received, its routing information is added to the beginning of the linked list, after which all routes for this prefix are sequentially scanned and the best one is selected. When a route is deleted, it is removed from the list of routes for this prefix and the procedure for choosing the best route is started in the same way.

The BGP table is very similar in structure to the zebra routing table. Each prefix corresponds to several different routes received from different sources, but different BGP attributes are used instead of the administrative distance and metric. For each route, a pointer is stored to the BGP neighbor from which the route was received, which allows you to use the corresponding data about the neighbor, for example, the connection type (IBGP or EBGP), its router id and IP address. When a new route is obtained or deleted, a procedure is started that selects the best one for this prefix by using a sequential pairwise comparison of routes.

First of all, when a new packet from a neighbor is received, it is parsed. Then the `AS_PATH` loop is checked, that is, it checks that the `AS_PATH` does not contain the autonomous system number to which the router belongs.

Then, various route filtering mechanisms are implemented if they are configured for the BGP neighbor from which the packet was received.

If the route has successfully passed all the filtering stages, then the incoming route-map is applied to it. Here you can flexibly modify the attributes of the received BGP route, or filter out the route. The last step is to ask zebra for the validity of the next-hop and the metric before it.

4. Substrate-based solution

The blockchain provides the structure of a system that uses the blockchain to increase the security of BGP [9]. Blockchain is a solution for an environment in which the parties do not have to trust each other, but must cooperate [10].

Advantages of using the blockchain:

- All transactions occur between nodes without any intermediary side. Moreover, there is no need for a third party for authentication.

- Immutability of announcement and the ability to re-track the BGP route chain. In addition, the route is checked by several parties and is more reliable.

The goal of the work is to create a fully distributed decentralized routing system.

Thus, the system has a common global blockchain. With each update, all AS in the group check to see if their path has changed with the update.

The functionality provided by the blockchain allows for a secure mechanism for storing and retrieving data in a publicly verifiable and immutable manner. For this reason, the blockchain provides a storage mechanism for collecting data and reaching consensus among participants.

BGP UPDATE messages can be verified using signed transactions running on the blockchain platform.

Unlike some blockchains, Substrate splits the requirement to reach consensus into two separate phases [11]:

- Block authoring is the process nodes use to create new blocks.

- Block finalization is the process used to handle forks and choose the canonical chain.

Before reaching consensus, some nodes in the blockchain network must be able to produce new blocks. For our Substrate-based solution we have chosen a Proof of work computing-based scheduling algorithm so that we can get a fully decentralized network without any trusted nodes.

In proof-of-work consensus models, any node can produce a block at any time if the node has solved a computationally-intensive problem. Proof-of-work block authoring is not slot-based and does not require an authority set. In proof of work, anyone can produce a block at any time, so long as they can solve a computationally challenging problem (a hash preimage search). Solving the problem takes CPU time, and thus nodes can only produce blocks in proportion with their computing resources.

Each block header contains a reference to its parent block, so you can trace the chain back to its genesis. Forks occur when two blocks reference the same parent. Block finalization is a mechanism that resolves forks such that only the canonical chain exists.

A fork choice rule is an algorithm that selects the best chain that should be extended. In our prototype we use a GRANDPA. In the GRANDPA protocol, the longest chain rule simply says that the best chain is the longest chain.

GRANDPA provides block finalization. It does not author blocks; it just listens to gossip about blocks that have been produced by some authoring engine such as proof-of-work in the case of the prototype in question. GRANDPA validators vote on chains, not blocks, i.e. they vote on a block that they consider "best"and their votes are applied transitively to all previous blocks. Once more than 2/3 of the GRANDPA authorities have voted for a particular block, it is considered final.

When the router wants to send an Update message, it sends it as a signed transaction, thereby becoming a node of this blockchain network. This means that in order to interact with the blockchain and participate in the network, each AS router must run its own blockchain node.

The transmitted route is stored in the block of the blockchain network. Thus, the host must transmit the router id and AS number in its message, as shown in Fig.4. To add this route to the route map quagga, it is passed as a command to manually add a route, after which quagga independently adds it to the system, relying on protocols and restrictions on a specific host. In the future, it is planned to transfer all route checks directly to the blockchain so that the addition takes place without any interaction with the quagga system.

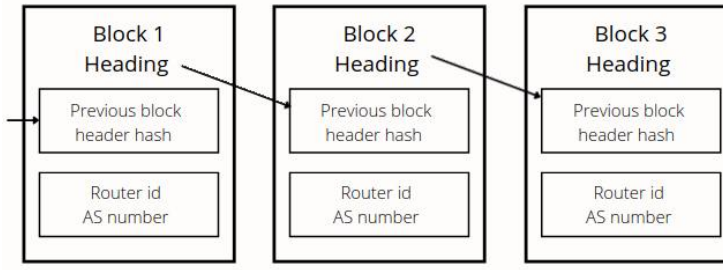

Fig 4. Block structure

After verifying the message and confirming the transaction, the blockchain interacts with other network nodes via Event messages by sending messages to every node on the network. Now the BGP node does not need to send UPDATE messages to other nodes. Using the event module, the blockchain informs network nodes about changes in the routing table and the need for its local update.

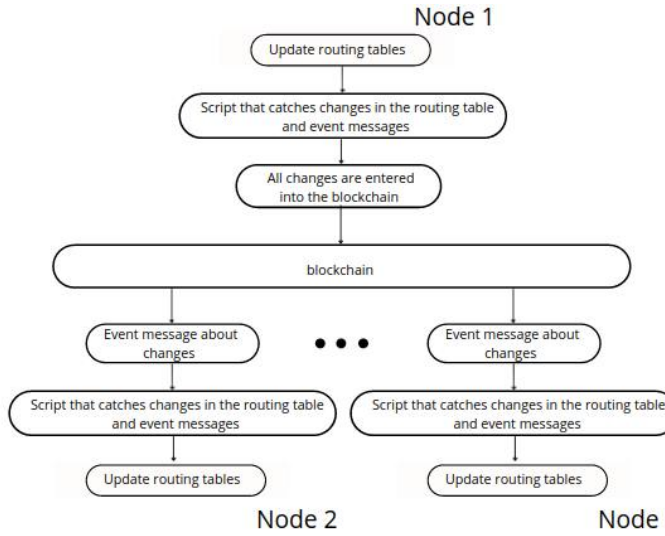

Fig 5. Prototype work

The Fig.5 shows a diagram of work of the prototype.

5. Review of penalty systems

After creating a prototype of the BGP dynamic routing protocol on the Internet based on blockchain technology, it is necessary to create a trusted environment.

Since routers trust all received messages and update their routing tables with almost no additional checks, there are quite a lot of options for violating the correctness of the routing table. Based on this, it was decided to introduce a system of fining routers for incorrect behavior in the system into the prototype. As an example of this behavior, you can take the termination of a BGP session or the transmission of an incorrect route such as:

- Non-existent route
- Is not the shortest route
- Invalid next hop
- Incorrect Local preference inside AS (for example, too high for this route to be selected)

Entities participating in the system should be such that any damage far outweighs any benefit that can be obtained from incorrect behavior in the system. The potential consequences of being fined as a result of improper behavior should be serious enough to take the incorrect router out of the system. The protocol controlling the blockchain application should monitor the behavior of participants. Failure to fulfill agreed obligations may result in sanctions such as denial of service, cancellation of anonymity or loss of funds in the account or other types of fines.

There are many mechanisms for fining nodes in blockchain systems, let's consider the most competitive ones.

A. `Cosmos SDK`

In the Cosmos SDK system, penalties for nodes for incorrect behavior may include, but are not limited to:

- `Burning` some part of their share
- `Depriving` them of the opportunity to vote on future blocks for a certain period of time

At any time, any number of validators are registered in the system, in each block they can propose and vote on blocks. Such validators are at risk if some kind of protocol error is committed. For each of these validators, a record is stored that contains information related to the life and other attributes associated with the violation.

The Cosmos SDK implements a system for each validator that allows you to find the validator only once in case of an error. For example, if the validator is incorrectly configured and it signs several old blocks twice, the validator will be punished only for the first one.

B. `Crypto.org`

In this system, validators are responsible for signing or proposing a block in each round of consensus. To strengthen it, it is necessary to impose punishment for the bad behavior of validators.

In particular, functionality is provided that aims stimulate actions observed on the network, such as incorrect checks. Penalties may include the loss of some part of their bet, the loss of their ability to perform network functions for a certain period of time, collect rewards, etc.

Punishments for the validator are triggered when they either make a mistake or do not interact with the system for a long time.

To configure the behavior of validators when fines are imposed, network parameters such as parameters for tracking uptime are used;

the maximum percentage of blocks with erroneous/missed checks allowed for the account; the deadline for removing the validator from the system in case of an error; the percentage of funds reduction when the validator makes an error; the percentage of funds reduction when the validator does not it works.

C. Summary

After reviewing the fining systems in other blockchain systems, three main areas were identified that should be taken into account in order for fining participants to make sense in our model:

- Fining for a certain amount in the presence of some conditional unit.
- Deprivation of the possibility of voting on future blocks for a certain period of time.
- Complete exclusion of the router from the blockchain system.
- The choice of the type of fine should be determined based on the architecture of the system, as well as the number of fines already received by the router and the severity of the error.

6. Creating a trusted environment

In the proposed architecture of the trusted environment (Fig. 6), an interface is implemented through which all routers will be able to track the changes made and report the received incorrect route. In the case of several requests about the incorrectness of one of the routes, the validator sends a new Update message canceling this route, that is, removing it from the node routing tables, and the node that sent the incorrect route is fined a certain amount.

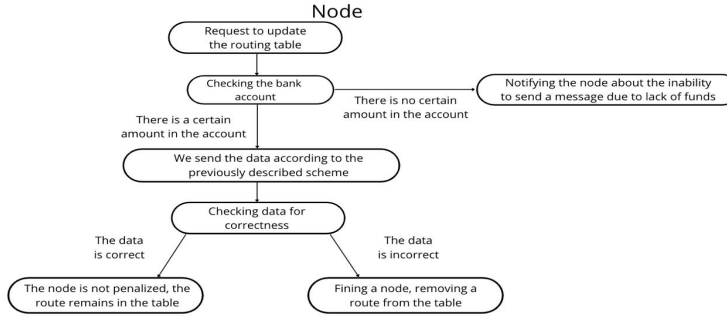

Fig 6. Trusted environment

7. Discussion

All studies were emulated in a system with the following indicators:

- `OS: Ubuntu 21.04`
- `Processor cores: 8`
- `Processor base frequency: 2.3 GHz`
- `RAM: 16 GB`
- `Rust Version: Rust nightly-2020-10-01`

The blockchain queues all received events and sends them to the block in the received order. Therefore, even if several events were received at the same time, the time of their total processing will be linearly dependent on the number of events received.

When testing the developed prototype of interaction, the results presented in Fig.7 and in Fig.8 were obtained.

	Number of events (transmitted routes)			
	1	***10***	***100***	***1000***
Average working time (sec)	0.0807	0.8798	9.3947	89.487
Average processing time for one route (sec)	0.0807	0.0879	0.0939	0,0894
Average running time of the intermediate script (sec)	0.0065	0.0731	0.6834	6.6108
Average running time of the intermediate script for one route (sec)	0.0065	0.0073	0.0068	0.0066

Fig 7. System prototype test data

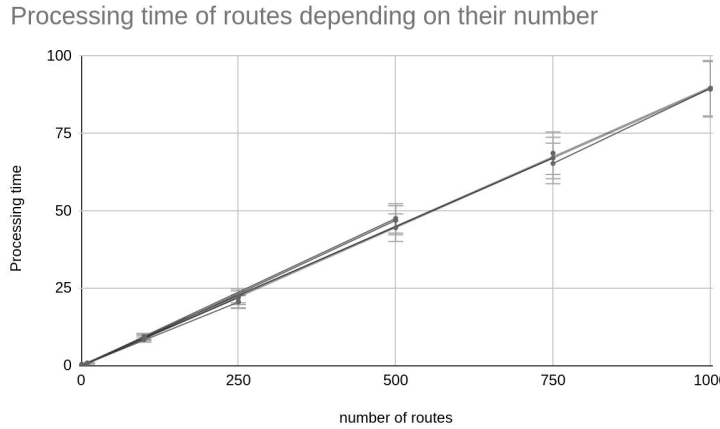

Fig 8. Processing time of routes depending on their number

The research was carried out under the Ubuntu 21.04 system. Events were sent to the blockchain to change the routing table on the node at the same time. As can be seen from the table, the average time for processing an event does not change, and the total time, as expected, linearly depends on the number of events, since they are buffered into a queue, after which they are sequentially sent to the nodes.

For comparison, the average transmission time of a single route in Quagga under the same conditions is 0.0002 s.

The increase in operating time is due to the TCP blockchain connection, as well as due to the parsing of routes and their inclusion in the blockchain.

The statistics of the consumed memory of the blockchain node, displayed in Fig.9, shows a linearly increasing function.

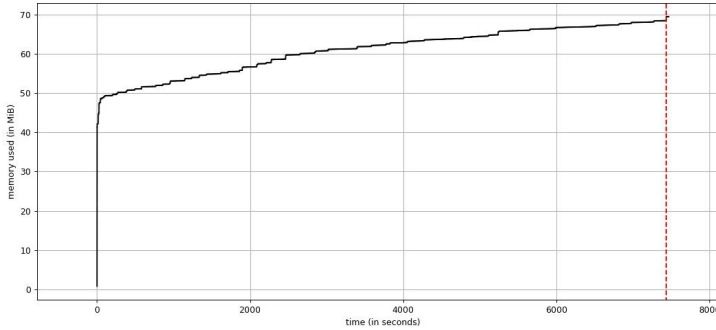

Fig 9. Consumed memory of the node

An increase in the memory consumption function occurs due to the need for local storage of the entire blockchain on a given node. This leads to high memory consumption when working with a large system for the prototype under study.

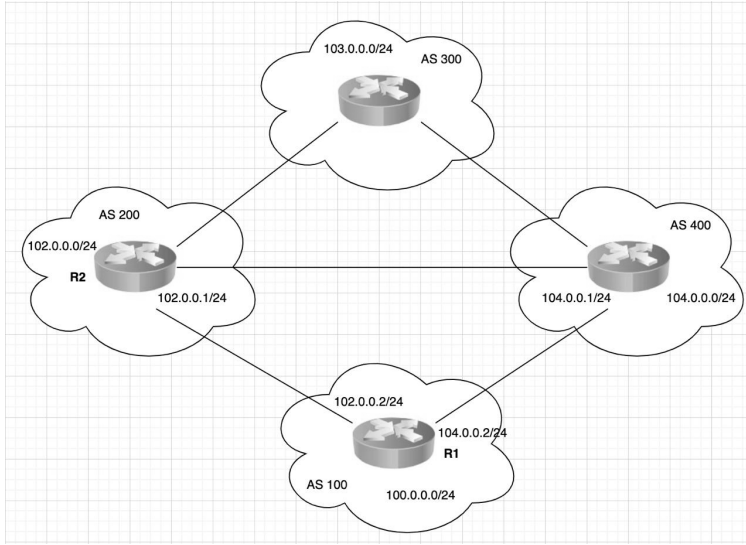

Fig 10. Network device

To investigate the correctness of the prototype, a network of 4 AS was created (Fig. 10). When viewing the routing table of the router R1 when configuring quagga, a routing table was obtained (Fig. 11):

```
Router# show ip bgp

BGP table version is 22, local router ID is 100.0.0.1
Status codes: s suppressed, d damped, h history, * valid, > best, i -
internal,
              r RIB-failure, S Stale, m multipath, b backup-path, x
best-external
Origin codes: i - IGP, e - EGP, ? - incomplete

  Network          Next Hop        Metric  LocPrf  Weight  Path
*> 100.0.0.0/24    0.0.0.0           0               32768   i
*> 102.0.0.0/24    104.0.0.1                         0       400 200 i
*>                 102.0.0.1         0               0       200 i
*> 104.0.0.0/24    102.0.0.1                         0       200 400 i
*>                 104.0.0.1         0               0       400 i
*> 103.0.0.0/24    102.0.0.1                         0       200 300 i
*>                 104.0.0.1                         0       400 300 i
```

Fig 11. Source routing table

The same routing table (Fig. 11) was obtained using a developed prototype based on blockchain technology. The results of the prototype and the quagga system are the same.

After that, an experiment was conducted, during which complaints were sent from routers about routes to R2, during which they were removed from the routing tables. Now the following picture is visible on R1 (Fig. 12):

```
Router# show ip bgp

BGP table version is 22, local router ID is 100.0.0.1
Status codes: s suppressed, d damped, h history, * valid, > best, i -
internal,
              r RIB-failure, S Stale, m multipath, b backup-path, x
best-external
Origin codes: i - IGP, e - EGP, ? - incomplete

  Network          Next Hop        Metric  LocPrf  Weight  Path
*> 100.0.0.0/24    0.0.0.0           0               32768   i
*> 104.0.0.0/24    104.0.0.1         0               0       400 i
*> 103.0.0.0/24    104.0.0.1                         0       400 300 i
*> 102.0.0.0/24    104.0.0.1                         0       400 200 i
```

Fig 12. Routing table after deleting routes

After that, the same routing table (Fig. 12) was obtained on R1 in the scheme without a route to R2.

Conclusion

This article has reviewed the issue of BGP routing security. As its solution, the implementation of a prototype BGP routing system based on blockchain technology was proposed, which made it possible to ensure the security of transmitted routing data while maintaining a relatively high speed of information processing. The blockchain system, in contrast to the previously proposed solutions based on centralized systems, provides greater reliability of the system's functioning due to the abandoning of a main node, an attack on which can lead to the failure of the entire system.

Even though the prototype [13] is slower than the Quagga system and requires memory to store the entire blockchain at each node, in the long term it provides greater security when transferring routes due to verification of the sender of transactions, decentralization of the system, invariability of route chain announcements and the possibility of re-tracking them. Also the advantage of the prototype is the simplicity of system deployment.

Perhaps, in the future, this system, if it cannot completely replace the ordinary BGP routing, can become a good solution for modernizing systems where the security of route transmission plays a key role.

References

1. O. Ulusoy, *Research issues in peer-to-peer data management.* inIEEEISCIS, 2007, pp. 1–8.

2. M. Saad, A. Anwar, A. Ahmad, H. Alasmary, M. Yuksel, and A. Mohaisen, *Routechain: Towards blockchain-based secure and efficient bgp routing.* inIEEE ICBC, 2019, pp. 210–218.

3. S. T. Kent, *Securing the border gateway protocol.* The Internet Protocol Journal, vol. 6, no. 3, pp. 2–14, 2003.

4. R. White, *Securing BGP through secure origin BGP (soBGP).* Business Communications Review, vol. 33, no. 5, pp. 47–53, 2003.

5. G. Goodell, W. Aiello, T. Griffin, J. Ioannidis, P. D. McDaniel, and A. D. Rubin, *"Working around BGP: An incremental approach to*

improving security and accuracy in interdomain routing. in NDSS, 2003.

6. Ma, X.; Xu, D.; Dang, H, *BGP-LSChain: An Inter-domain Link State Sharing Framework Based on Blockchain.* In Proceedings of the 2019 International Conference on Blockchain Technology, Honolulu, HI, USA, 15–17 March 2019; pp. 19–23.

7. Saad, M.; Anwar, A.; Ahmad, A.; Alasmary, H.; Yuksel, M.; Mohaisen, A., *RouteChain: Towards blockchain-based secure and efficient BGP routing.* In Proceedings of the 2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC), Seoul, Korea, 14–17 May 2019; pp. 210–218.

8. *BGP hijacking overview. Routing incidents prevention and defense mechanisms* `https://www.noction.com/blog/bgp-hijacking`

9. A. Hari and T. Lakshman, *The internet blockchain: A distributed, tamper-resistant transaction framework for the internet.* inACM HotNets, 2016.

10. Q. Xing, B. Wang, and X. Wang, *gpcoin: Blockchain-based internet number resource authority and BGP security solution.* iSymmetry, vol. 10,no. 9, p. 408, 2018.

11. Parity technologies, paritu.io.
`https://www.parity.io/substrate/`

12. Quagga software.
`https://www.quagga.net/`

13. Prototype code.
`https://github.com/Pluzhnikovadari/Substrate`

Абрамов А. В., Чупахин А. А.

Построение однопроцессорного расписания с минимизацией пикового использования ресурса при помощи муравьиного алгоритма

Введение

Эффективность использования ресурсов в вычислительных системах (ВС) тесно связана с планированием заданий. Частью указанного процесса является решение задачи составления расписаний. Традиционно решение таких задач направлено на минимизацию общего времени выполнения всех заданий, но часто появляются дополнительные ограничения, связанные с вычислительными ресурсами. Причём нередко такие ресурсы оказываются недешёвыми. Таким образом, при построении ВС возникает вопрос: как определить необходимый для ВС объём ресурсов, которого точно хватит для организации вычислительного процесса и не окажется в избытке?

В данной работе рассматривается проблема построения оптимального расписания с точки зрения уменьшения пикового потребления памяти ВС. ВС состоит из одного процессора, способного в каждый момент времени обрабатывать лишь одно задание. На вход алгоритму планирования подаётся ориентированный ациклический граф заданий, в котором рёбра представляют собой зависимости по данным между ними. Результат выполнения каждого задания занимает некоторый объем памяти ВС до тех пор, пока не выполнятся все непосредственные потомки данного задания. Необходимо построить расписание, минимизирующее пиковое потребление памяти.

В работе приведена математическая постановка рассматриваемой задачи, показана ее NP-трудность, проведён обзор различных алгоритмов построения расписаний. Для решения поставленной задачи были выбраны муравьиный алгоритм (МА) и алгоритм имитации отжига (ИО). Приведены их реализации, а также предложены модификации для МА, в числе которых локальный поиск и подход Ant-Q, основанный на обучении с подкреплением. Проведено экспериментальное исследование свойств разработанных алгоритмов на синтетических данных и данных, предоставленных компанией ООО «Техкомпания Хуавэй».

1. Постановка задачи

Модель прикладной программы может быть представлена как ориентированный ациклический граф $G = (V, E), |V| = n, |E| = m$. Каждой вершине графа соответствует задание $\{p_i\}_{i=1}^{n}$, каждой дуге — передача данных между заданиями. Если $\exists(p_i, p_j) \in E, 1 \leq i, j \leq n$, то для выполнения задания p_j необходим результат выполнения задания p_i. При этом задание p_j далее будем называть потомком задания p_i.

ВС состоит из одного процессора SP, который в каждый момент времени способен выполнять только одно задание, и памяти SR. Каждое задание выполняется на процессоре SP без прерываний. После выполнения задание $p_i, 1 \leq i \leq n$, захватывает $r_{p_i} \geq 0$ памяти SR для хранения результата. Далее происходит высвобождение памяти, занимаемой такими заданиями $p_j, 1 \leq j \leq n, (p_j, p_i) \in E$, для которых p_i является последним выполненным потомком.

Расписание HP сформировано, если для всех заданий из G определён порядок их выполнения на процессоре SP. Фактически HP представляет из себя перестановку заданий из G. Далее будем говорить, что расписание допустимо $HP \in HP^{*}_{\{1-3\}}$, если оно удовлетворяет набору ограничений $\{1-3\}$:

1. каждое задание должно быть назначено на процессор SP;
2. процессор SP в каждый момент времени выполняет не более одного задания;
3. частичный порядок, заданный графом зависимостей G, сохранен в HP.

Минимизируемой целевой функцией является максимальное по всему расписанию количество занятой в ВС памяти. Пусть задано расписание $HP \in HP^{*}_{\{1-3\}}$. Обозначим за $f^{k}_{HP}, 1 \leq k \leq n$, количество занятой в ВС памяти для k-ой позиции в расписании HP. Пусть A — множество заданий, расположенных в расписании HP на позициях $1, ..., k$. Пусть $B \subset A$ — множество заданий, расположенных в расписании HP на позициях $1, ..., k-1$, у каждого из которых все потомки расположены на позициях от 1 до $k-1$. Тогда f^{k}_{HP} вычисляется по следующей формуле:

$$f^{k}_{HP} = \sum_{p_j \in A} r_{p_j} - \sum_{p_i \in B} r_{p_i} \tag{1}$$

Пусть известны значения $\{f_{HP}^k\}_{k=1}^n$ для расписания HP. Тогда значение целевой функции f_{HP} для такого расписания:

$$f_{HP} = max_{1 \le k \le n} f_{HP}^k \tag{2}$$

Требуется для модели прикладной программы G найти такое расписание $HP_* \in HP_{\{1-3\}}^*$, для которого достигается минимальное значение целевой функции:

$$f_{HP_*} = min_{HP \in HP_{\{1-3\}}^*} f_{HP} \tag{3}$$

2. Обзор предметной области

Задача в текущей постановке является задачей о минимизации максимальных затрат (ММЗ) [1]. В работах [2, 3] показано, что в общем случае, т.е. без дополнительных ограничений, упрощающих поиск решений, задача является NP-полной.

Существует множество различных подходов для решения NP-полных задач. Для ограничения классов рассматриваемых алгоритмов были выделены следующие критерии:

1. отличие решаемой в статье задачи от поставленной в данной работе;
2. тип алгоритма: рандомизированный или детерминированный;
3. конструктивный или итерационный алгоритм;
4. порядок сложности алгоритма — экспоненциальный или полиномиальный;
5. число параметров алгоритма.

Известны некоторые точные методы решения NP-полных задач, в числе которых метод ветвей и границ и различные методы динамического программирования, но они имеют экспоненциальную сложность и применяются для задач небольших размерностей и ограниченных классов данных. В работе рассматривается задача в общем виде, где число вершин в графе заданий может достигать $2 * 10^6$, поэтому было решено ограничиться приближёнными методами, решающими поставленную задачу.

В некоторых частных случаях существуют полиномиальные алгоритмы решения задачи о ММЗ. Так, в работах [3, 4] для частного случа графа вычислений в виде двоичного дерева

предлагаются конструктивные решения поставленной задачи. Оба алгоритма находят оптимальное решение за полиномиальное время. В работе [5] приводится полиномиальный алгоритм для частного случая задачи о ММЗ, когда отношения предшествования заданы в виде последовательно-параллельного графа.

В данной работе рассматривались следующие классы алгоритмов:

1. жадные алгоритмы;
2. алгоритмы имитации отжига [6];
3. генетические алгоритмы [8];
4. муравьиные алгоритмы [9].

Результаты обзора приведены в табл. 1.

	Рандомизированный/ детерминированный	**Конструктивный/ итерационный**	**Порядок сложности**	**Число парам-ов**
Жадный алгоритм	Детерминированный	Конструктивный	Зависит от жадного критерия	1
Имитация отжига	Рандомизированный	Итерационный	$n \cdot [e^{\frac{T_0}{T_{min}}} - 1]$	3
Генетический алгоритм	Рандомизированный	Итерационный	Зависит от генетических операторов	4-5
Муравьиный алгоритм	Рандомизированный	Конструктивно-итерационный	$O(n^2 \cdot it \cdot a)$	7

Таблица 1: Обзор алгоритмов планирования вычислений; n — количество заданий, T_0, T_{min} — начальная и конечная температуры в алгоритме ИО, it, a — число итераций и число муравьёв в МА.

3. Описание разработанного алгоритма

3.1 Основные понятия и формулы

Напомним, что формально расписание HP представляет из себя перестановку заданий из G, поэтому каждое задание p_j в нём привязано к позиции $i, 1 \leq i, j \leq n$.

В алгоритме используются следующие параметры:

1. $epochs\_count, ants\_count$ — количество итераций и муравьёв, соответственно. На каждой итерации каждый из муравьёв строит своё решение, опираясь на опыт предшественников.

2. η_{ij} — эвристическая информация (видимость) о том, насколько «хорошей» кажется постановка допустимого задания p_j на место $i, 1 \leq i, j \leq n$.

3. τ_{ij} — «след» (в природе: феромонный след). После каждой итерации данный параметр корректируется. Параметр показывает насколько «хорошим» оказалась постановка допустимого задания p_j на место $i, 1 \leq i, j \leq n$. То есть это накопленная статистическая информация о качестве выбора задачи p_j для постановки на место i.

4. ρ — коэффициент распада феромона, $\rho \in [0, 1]$.

5. q_0 — пороговое значение для вероятности перехода, $q_0 \in [0, 1]$.

6. α, β — влияние следа и видимости (соответственно) на вероятность перехода.

Эвристическую информацию η_{ij} будем вычислять по следующей формуле:

$$\eta_{ij} = \frac{1}{f_{HP}^{ij}} \tag{4}$$

где f_{HP}^{ij} — значение целевой функции для i-ой позиции в расписании при постановке на неё задания p_j. То есть, формально говоря, η_{ij} — величина обратная к ресурсу, который окажется занят при постановке задания p_j на место $i, 1 \leq i, j \leq n$ в расписании. Чем больше ресурсов окажется занято, тем меньше величина η_{ij}.

Матрица вероятностей перехода выглядит следующим образом:

$$P_{ij} = \begin{cases} \frac{[\tau_{ij}]^\alpha [\eta_{ij}]^\beta}{\sum_{h \in D_i} [\tau_{ih}]^\alpha [\eta_{ih}]^\beta}, j \in D_i, \\ 0, j \notin D_i \end{cases} \tag{5}$$

где D_i — множество заданий, допустимых для планирования на шаге i. Значение P_{ij} показывает вероятность постановки задания p_j на место $i, 1 \leq i, j \leq n$ в расписании.

3.2 Муравьиный алгоритм

На вход алгоритму подаётся направленный ациклический граф заданий G. Так как частичный порядок, заданный в G, должен сохраняться в HP, каждому муравью в процессе построения решения необходим список возможных переходов. Для этого на шаге i вводится множество D_i, содержащее такие задания, у которых все предки располагаются на местах $1, ..., i-1$ в уже построенной части HP.

Шаги алгоритма:

1. Сгенерировать расписание, на каждом шаге i выбирая из D_i задание, у которого значение η_{ij} максимально. Пусть f^0 — значение целевой функции (2) для такого расписания.
2. Положить $\tau_{ij} = \tau_0 = \frac{1}{f^0}, 1 \leq i, j \leq n$.
3. Положить $n\_ant = 1$.
4. Пока $n\_ant \leq ants\_count$:
   (a) Положить $i = 0$.
   (b) Пока $i \leq n$:
   - Вычислить множество D_i и значения η_{ij} и P_{ij} для всех $j \in D_i$ (4-5).
   - $q = uniform(0, 1)$.
   - Из множества D_i задание p'_j выбирается по следующему правилу:

$$p'_j = \begin{cases} argmax_{h \in D_i}\{[\tau_{ih}]^\alpha [\eta_{ih}]^\beta\}, q \leq q_0 \\ p'_j \text{ вычисляется в зависимости от } P_{ij}, q > q_0 \end{cases} \quad (6)$$

   - Задание p'_j ставится на выполнение.
   - Вычислить f_i по (1).
   - Локальное обновление следа (см. Раздел 3.3).
   - $i = i + 1$.

   (c) Вычислить f_{n_ant} по формуле (3).
   (d) $n\_ant = n\_ant + 1$.
5. Глобальное обновление следа (см. Раздел 3.3).
6. Запомнить наилучшее решение BS (решение с наименьшим значением целевой функции среди всех решений, которые нашли муравьи).
7. Шаги 3-6 выполняются $epochs\_count$ раз.

3.3 Обновление феромонного следа

Локальное обновление следа происходит после планирования каждого задания в локальном расписании. Элемент матрицы следа τ_{ij} обновляется по следующему правилу:

$$\tau_{ij}(t+1) = (1-\rho) \cdot \tau_{ij}(t) + \rho \cdot \tau_0 \tag{7}$$

Глобальное обновление следа происходит в конце каждой итерации. Элемент матрицы следа τ_{ij} обновляется по следующему правилу:

$$\tau_{ij}(t+1) = \begin{cases} (1-\rho) \cdot \tau_{ij}(t) + \rho \cdot \Delta\tau_{ij}(t), (i,j) \in BS \\ (1-\rho) \cdot \tau_{ij}(t), (i,j) \notin BS \end{cases} \tag{8}$$

где $\Delta\tau_{ij}(t) = \frac{1}{f^*}$, f^* — значение целевой функции для наилучшего решения BS по всем итерациям.

3.4 Модификации муравьиного алгоритма

Локальный поиск

Локальный поиск заключается в итеративном применении операции (количество итераций — это параметр алгоритма, по умолчанию 20) перестановки случайно выбранного задания на допустимую позицию, т.е. такую позицию, при постановке на которую не нарушается корректность расписания. Локальный поиск применяется в конце каждой итерации ко всем решениям, полученным на текущей итерации.

Ant-Q

Ant-Q [10] — это расширение классического алгоритма муравьиных колоний, который переформулируется в терминах обучения с подкреплением. В Ant-Q подходе рассматривается система кооперирующихся агентов (муравьёв), изучающих пространство решений. Агенты взаимодействуют между собой посредством обмена информации, представленной в форме AQ-значений.

Цель подхода Ant-Q состоит в обучении матрицы AQ-значений таким образом, чтобы они способствовали нахождению хороших решений. Обучение AQ-значений происходит по следующей формуле:

$$AQ(i,j) \leftarrow (1-\rho)\cdot AQ(i,j)+\rho\cdot(\Delta AQ(i,j)+\gamma\cdot max_{k\in D_{i+1}} AQ(i+1,k)) \quad (9)$$

где ρ — распад AQ-значений, аналогичный распаду феромона в муравьином алгоритме, $\Delta AQ(i,j)$ — изменение AQ-значений, аналогичное изменению феромона $\Delta\tau_{ij}$, а γ — коэффициент влияния качества следующего перехода.

В общем случае $\Delta AQ(i,j)$ можно использовать как в локальном, так и в глобальном обновлении AQ-значений соответственно. В данной работе локальное усиление ($\Delta AQ(i,j)$) всегда равно нулю, в то время как глобальное усиление вычисляется по следующей формуле:

$$\Delta AQ(i,j) = \begin{cases} \frac{W}{f^*}, (i,j) \in BS, \\ 0, (i,j) \notin BS \end{cases} \quad (10)$$

где W — коэффициент глобального усиления, а f^* — целевая функция для наилучшего решения BS.

Схема Ant-Q практически совпадает со схемой муравьиного алгоритма, описанного выше. Отличие состоит лишь в том, что для осуществления перехода на каждом шаге Ant-Q учитываются также характеристики следующего перехода. Вместо матрицы следов феромона используется матрица AQ-значений, и правила локального и глобального обновления следов феромона (7)-(8) заменяются на правило обновления AQ-значений (9)-(10).

4. Экспериментальное исследование свойств алгоритма

4.1 Описание входных данных

Для проведения исследований был разработан генератор случайных графов, который конструирует графы с параметрами (число вершин и рёбер, вес вершин) из равномерного распределения.

Также для исследований использовался комплект данных, предоставленных ООО «Техкомпания Хуавэй». Комплект содержит 500 графов с 30-100 вершинами. Каждая вершина может иметь не более двух предков и произвольное количество потомков. Для каждой вершины определён объём ресурса, занимаемый результатом задания, соответствующего данной вершине. Объём

занимаемого ресурса принимает одно из следующих значений: 16384, 65536, 262144.

4.2 Подбор значений параметров алгоритмов

Значения параметров алгоритмов устанавливаются до этапа построения расписания, поэтому возникает задача подбора оптимальных значений параметров, при которых алгоритм будет сходиться к глобальному оптимуму, или получать решение, близкое к нему. В данной работе следующие параметры МА подбирались по сетке: ρ, q_0, α, β (см. раздел 3.1). В качестве подбираемых параметров для алгоритма Ant-Q рассматривались следующие: W, γ (см. раздел 3.4).

В таблице 2 приведены параметры сгенерированных графов. Плотность графа — отношение числа рёбер графа к максимальному числу рёбер для данного числа вершин. Было сгенерировано по 50 графов каждого размера: mini, small, ..., ultralarge.

Результаты подбора для МА и Ant-Q приведены в таблицах 3 и 4 соответственно. На каждом графе алгоритмы запускались по 50 раз. Среднее значение f_{HP} — это среднее арифметическое значений целевых функций по всем запускам. Разброс f_{HP} — среднеквадратическое отклонение значений целевой функции по всем запускам.

	mini	small	medium	large	extralarge	ultralarge
Число вершин	15	30	45	60	90	120
Минимальная плотность	0,2	0,5	0,3	0,4	0,4	0,4
Максимальная плотность	0,7	0,8	0,7	0,8	0,8	0,8

Таблица 2: Параметры сгенерированных наборов данных

Датасет	q_0	α	ρ	β	Ср. знач. f_{HP}	Разброс f_{HP}	Ср. знач. t в с.	Разброс t в с.	Ср. знач. *iter*	Разброс *iter*
mini	0,2	0,4	0,1	0,7	255,8	0	0,24	0,02	309,39	7,46
small	0,4	0,4	0,3	0,9	787,67	0	0,9	0,06	300,65	0,64
medium	0,4	0,6	0,1	0,8	1124,21	0,52	1,8	0,19	337,53	24,04
large	0,2	0,9	0,1	0,9	2414,12	0,02	6,05	0,25	307,22	5,65
extralarge	0,2	0,9	0,1	0,9	2536,14	0,06	6,45	0,27	307,82	5,98
ultralarge	0,2	0,6	0,1	0,9	3389,58	0	9,9	0,49	305,46	4,58
general	0,4	0,4	0,1	0,8	1751,28	0,09	4,24	0,2	309,67	7,24

Таблица 3: Результаты для наилучших параметров на каждом наборе данных (mini, small, ..., ultralarge) и на всех наборах в среднем (general) для MA.

Датасет	γ	W	Ср. знач. f_{HP}	Разброс f_{HP}	Ср. знач. t в с.	Разброс t в с.	Ср. знач. *iter*	Разброс *iter*
mini	0,3	0,1	255,8	0	0,23	0,02	129,24	2,4
small	0,9	0,3	787,67	0	0,79	0,03	125,33	0,36
medium	0,9	0,5	1124,3	0,48	1,47	0,09	134,24	6,79
large	0,9	0,9	2539,48	0,03	5,48	0,2	128,11	2,5
extralarge	0,9	0,9	2536,16	0,05	5,59	0,18	128	2,9
ultralarge	0,9	0,9	3440,45	0	8,2	0,34	127,55	1,85
general	0,9	0,5	1780,71	0,14	3,74	0,13	129	2,94

Таблица 4: Результаты для наилучших параметров на каждом наборе данных (mini, small, ..., ultralarge) и на всех наборах в среднем (general) для алгоритма Ant-Q.

Из таблиц видно, что на маленьких наборах данных (mini, small) и на плотных графах с числом вершин больше 100 разброс значений целевой функции равен нулю. Это связано с размером и плотностью графов: чем больше (меньше) плотность (число вершин) графа, тем меньшее число различных корректных для данного графа расписаний возможно составить. Поэтому перебор возможных решений сокращается, и алгоритм может быстро сходится как к локальным, так и глобальным оптимумам.

4.3 Точность алгоритмов

Для подсчёта оптимумов значений целевой функции использовался полный перебор с отсечением. Оптимумы подбирались на графах с числом вершин не больше 80-ти, причём графы с числом вершин больше 60-ти имеют преимущественно плотности больше $0,7$.

В таблице 5 представлены отклонения алгоритмов от оптимальных значений, а также разброс значений целевой

функции, который представляет собой среднеквадратическое отклонение, посчитанное по относительным отклонений значений целевой функции от оптимальных значений по всем запускам. Видно, что МА с подобранными значениями параметров показывает наименьшее отклонение от оптимумов среди всех остальных алгоритмов. МА со случайными значениями параметров показывает второй результат, ИО — показывает наихудший результат согласно среднему отклонению и разбросу (39%). Это говорит о том, что значения параметров МА, подобранные для одного класса графов, не всегда показывают наилучшие результаты на другом классе графов, из чего следует, что имеет смысл разбивать наборы входных данных на классы и подбирать параметры для каждого класса отдельно.

Алгоритм	Среднее отклонение от оптимума (%)	Разброс
МА	7,46	0
МА с оптимальными параметрами	7,26	0
ИО	7,64	0,39

Таблица 5: Сравнение оптимального решения с алгоритмами: МА с подобранными параметрами, МА со случайными параметрами, ИО [6].

4.4 Сравнение алгоритмов

Сравнение алгоритмов проводилось на графах, предоставленных ООО «Техкомпания Хуавэй» (см. Раздел 4.1). Результаты приведены на графиках 1-2. На оси абсцисс расположены номера графов, отсортированных в лексикографическом порядке по числу вершин и числу рёбер.

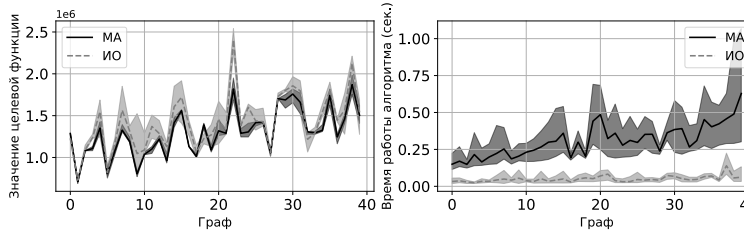

Рис. 1: Слева изображена зависимость целевой функции, а справа зависимость времени работы алгоритма (в сек.) от графа для алгоритма ИО и МА.

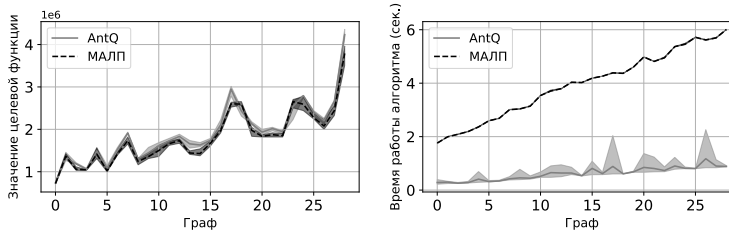

Рис. 2: Слева изображена зависимость целевой функции, а справа зависимость времени работы алгоритма (в сек.) от графа для алгоритма Ant-Q и МА с локальным поиском (МАЛП).

В среднем МА находит решения оптимальнее, чем ИО в 73% случаев, но в 46% случаев на каком-то из запусков в серии запусков ИО удалось найти решение оптимальней решения, найденного МА. В среднем МАЛП нашёл решения оптимальней, чем МА, ИО и Ant-Q, в 56%, 83% и 97% случаев соответственно, что является наилучшим показателем. Но если обратить внимание на графики времени работы каждого алгоритма, то можно заметить, что МАЛП сильно уступает остальным алгоритмам. Это связано с тем, что помимо построения *ant_count* решений в каждой эпохе он также запускает на каждом из решений локальный поиск из 20 итераций, что сильно сказывается на времени работы.

Заключение

В работе рассматривалась задача построения однопроцессорного расписания с минимизацией пикового использования ресурса. Установлено, что, во-первых, наилучшие значения параметров, подобранные для МА на одном классе данных, не всегда оказываются наилучшими для других классов, поэтому следует подбирать значения параметров для каждого класса в отдельности, если они заранее известны, или же использовать различные адптивные методики, которые производят подбор параметров во время построения конкретного расписания и в данной работе не рассматривались.

Во-вторых, для плотных или же маленьких графов МА быстро сходится к какому-то оптимуму (не обязательно к глобальному) и застревает в нём. Это можно видеть из оценки его разброса и качества полученного решения.

В-третьих, число параметров МА и его модификаций сильно превосходит число параметров алгоритма ИО, что затрудняет подбор для них наилучших значений.

В-четвёртых, наилучшие результаты смог показать МАЛП, но время его работы много больше времени работы всех остальных алгоритмов. Дальнейшие исследования предполагается направить на устранение описанных проблем.

Реализация разработанных алгоритмов вместе с результатами экспериментального исследования публично доступны на Github [10].

Литература

1. Michael R. Garey, David S. Johnson, *Computers and intractability* Freeman San Francisco, 1979, vol. 174.

2. Abdel-Wahab, Hussein Mohamed, *Scheduling with applications to register allocation and deadlock problems*, 1976.

3. Ravi Sethi, *Complete register allocation problems* // SIAM journal on Computing, 1975, vol. 4, no. 3, pp. 226–248.

4. Roman R. Redziejowski, *On arithmetic expressions and trees* // Communications of the ACM, 1969, vol. 12, no. 2, pp. 81–84.

5. Hussein M. Abdel-Wahab, Tiko Kameda, *Scheduling to minimize maximum cumulative cost subject to series-parallel precedence constraints* // Operations Research, 1978, vol. 26, no. 1, pp. 141–158.

6. Зорин В. А, Костенко В. А. *Алгоритм имитации отжига в задачах построения многопроцессорных расписаний* // Автоматика и телемеханика, 2014, №10, с. 97–109.

7. Marco Dorigo, Luca Maria Gambardella, *A study of some properties of Ant-Q* // International Conference on Parallel Problem Solving from Nature, 1996, pp. 656–665.

8. Mitchell Melanie, *An Introduction to Genetic Algorithms* // MIT Press, 1998.

9. Dorigo Marco, Mauro Birattari, and Thomas Stutzle, *Ant colony optimization* // IEEE computational intelligence magazine, vol. 1, no. 4, pp. 28–39.

10. Abramov A.V., *Scheduling Problem Source Code*, https://github.com/Crotokot/SchedulingProblem

Бахмуров А. Г.

Опыт преподавания курса "Имитационное моделирование в исследовании и разработке вычислительных систем"и планы по развитию курса

1. Понятие имитационного моделирования и некоторые его особенности

Имитационное моделирование - это развиваемый с конца 50-х годов XX века мощный и универсальный метод исследования динамических систем различной природы, в том числе, вычислительных систем и компьютерных сетей. Имитационная модель объекта - это описание поведения объекта во времени при помощи алгоритма, с учётом влияния внешних воздействий на объект.

Процесс создания и исследования имитационной модели в общем виде состоит из следующих этапов: определение цели моделирования; построение концептуальной модели; построение и отладка имитационной модели; валидация имитационной модели; планирование и выполнение экспериментов с моделью; анализ результатов моделирования и выводы. По сути, экспериментальное исследование поведения реального объекта заменяется исследованием поведения, воспроизводимого его имитационной моделью.

При программной реализации имитационной модели особенно важно сократить усилия программиста и уменьшить количество ошибок программирования, так как, во-первых, зачастую имитационная модель создаётся для конкретной цели исследования и "выбрасывается" по его окончании, а, во-вторых, "эталон" , с которым можно сверить функционирование модели как программы для ЭВМ, не доступен (иначе бы можно было исследовать **его**, а не строить модель), и в ходе валидации имитационная модель может быть многократно переписана. Поэтому при создании имитационных моделей широко используются соответствующие библиотеки для универсальных языков программирования, а также специализированные языки и программные системы.

2. Имеющиеся учебные пособия и курсы, необходимость создания данного курса

Следует в первую очередь отметить классические книги, охватывающие весь процесс построения и исследования модели: Р. Шеннон [1], Лоу и Кельтон [2], Советов и Яковлев [3]. В этих книгах подробно рассматривается "философия" имитационного моделирования, математические методы планирования эксперимента и обработки результатов. Однако, в них, вышедших не позже 2001 г, не рассматриваются современные программные средства построения имитационных моделей, такие как OMNet++, ns-3. Известны также учебники Замятиной [4], по ИМ в экономике [5], по моделированию сетей связи [6]. В этих курсах также узок круг рассматриваемых инструментальных средств. Во всех упомянутых источниках не рассматривается сам объект моделирования - в данном случае вычислительные системы и сети.

В силу сказанного выше, было принято решение создать новый учебный курс на основе материалов нескольких авторов, взяв за основу книгу Р. Шеннона [1]. Ряд тем (понятие производительности ВС, наблюдение за работой ВС) был включён в курс под влиянием методической разработки Р.Л. Смелянского [7], с доработкой под современные реалии.

При начальной разработке курса и последующей его доработке автор стремился следовать следующим принципам:

- в основу содержания курса должен быть положен процесс создания и использования имитационной модели;
- перед тем, как излагать некоторый метод исследования (в нашем случае это - ИМ), следует рассказать об особенностях объекта исследования (вычислительные системы) и об основных целях исследования (в нашем случае - это анализ и синтез производительности, проверка правильности функционирования);
- в курсе должны быть рассмотрены основные подходы к представлению имитационных моделей и модельного времени в них: дискретно-событийный, с непрерывным временем, агентный, несмотря на то, что в основном используется дискретно-событийный подход;
- дать в курсе дополнительные сведения о вычислительных системах, средствах наблюдения и моделях, которые не

изучаются в других курсах кафедры АСВК, но представляются полезными для студентов.

3. Содержание курса по темам

Тема 1. Вычислительные системы (ВС) как объект исследования и разработки. Вводится понятие ВС как совокупности аппаратного и программного обеспечения. Кратко рассматриваются наиболее важные виды аппаратных средств ВС: серверы, суперкомпьютеры, центры обработки данных, встроенные вычислительные системы, сетевые устройства. Перечисляются специфические требования к услояиям и режимам использования различных видов ВС, указываюся соответствующие их особенности. Например: конструктивное исполнение сервера должно обеспечить его круглосуточную работу с возможностей резервирования и быстрой замены относительно малонадёжных компонентов; для суперкомпьютера, в отличие от центра обработки данных, характерен режим пакетной обработки вычислительно сложных заданий, с относительно небольшим числом параллельно выполняемых заданий.

Тема 2. Производительность и эффективность ВС. Вводится понятие производительности ВС, определяются основные характеристики производительности - пропускная способность и время отклика. Описываются единицы измерения производительности. Вводятся понятия номинальной (технической) производительности ВС и системной производительности. Приводятся примеры программ для измерения производительности (эталоны, англ. benchmarks). Указывается, что наиболее надёжный способ измерения производительности для доступной в реальности ВС - наблюдение за работой ВС при выполнении программы. Вопросы наблюдения за работой ВС подробно будут рассмотрены в теме 4.

Тема 3. ВС как совокупность ресурсов. В этой теме развивается представление о ВС. Вводятся понятие ресурса и понятие рабочей нагрузки, как потока обращений к ресурсам. Вводятся уровни детальности рассмотрения ВС: совокупность вычислительных узлов, уровень общей архитектуры узла, уровень микроархитектуры. Приводятся примеры ресурсов на этих уровнях. Даётся частично формализованное описание задач анализа и синтеза производительности в терминах ресурсов и рабочей нагрузки.

Тема 4. Наблюдение за работой ВС. Рассматриваются как аппаратные, так и программные наблюдатели. Примеры аппаратных наблюдателей: отладочные регистры процессора; Intel Processor Tracing; Performance Monitoring Counters для процессоров Intel, AMD, ARM; интерфейс JTAG; мониторы периферийных интерфейсов PCI, USB, CAN и других. В качестве программных наблюдателей уровня ОС рассматриваются: интерфейсы к счётчикам процессора и ОС, в частности, утилита perf; средства измерения времени; отладчик gdb (как напоминание), средства профилирования программы; средство Valgrind (подчёркивается, что оно способно не только обнаруживать ошибки работы с динамической памятью, но и некоторые ошибки синхронизации потоков, а также выполнять профилирование.

Тема 5. Понятие модели, виды моделей. Дискретно-событийные имитационные модели. Как и в других курсах ИМ, даётся определение модели. Перечисляются важные примеры целей моделирования: исследование объекта, прогнозирование и проектирование, обучение, обмен информацией. Приводится классификация моделей: физические (натурные, полунатурные, масштабные, аналоговые); знаковые, в том числе, математические (структурные, функциональные). В свою очередь, функциональные модели делятся на аналитические, алгоритмические и имитационные. Приводятся примеры аналитических моделей производительности ВС. Приводится программа простейшей потактовой модели функционирования кэш-памяти. Далее, описывается принцип дискретно-событийного имитационного моделирования. Он иллюстрируется на примере модели сервера с двумя клиентами. Далее показывается, как переписать модель в объектно-ориентированном виде, с использованием библиотеки SimpleSim. Рассматриваются две формиы представления модели с использованием этой библиотеки: в виде объектов, реагирующих на посылку сообщения (эквивалентно планированию события для объекта) и в виде объектов-процессов, имеющих свои потоки управления и работающих параллельно в модельном времени.

Тема 6. Формальные модели функционирования ВС часть 1. Формальные модели нужны для представления концептуальной модели (о ней речь подробнее будет сказано в теме 9) перед построением программы имитационной модели. Рассматриваются диаграммы переходов состояний (StateCharts) - расширение известных студентам из курса дискретной математики конечных автоматов Мура. Эти диаграммы входят в состая языка

UML, а также используются в некоторых системах имитационного моделирования.

Тема 7. Профессиональная система ИМ на примере OMNet++. Рассматриваются основные возможности системы: описание модели в виде набора простых и составных модулей, порты которых соединяются каналами передачи сообщений. Описываются правила построения простого модуля на языке C++, даётся обзор возможностей языка NED для связывания простых модулей в составной модуль. Приводится обзор возможностей интегрированной среды разработки моделей. Даётся краткое описание библиотек моделей.

Тема 8. Формальные модели часть 2. Рассматриваются: процессы с сообщениями, сети Петри. Расширением сетей Петри являются системы с очередями (E-сети), которые поддерживаются во многих языках и системах ИМ, начиная с языка GPSS. Приводится модель сервера с клиентами, переписанная для систем моделирования OMNet++ и AnyLogic. Примечание: вопросы анализа логических свойств сетей Петри (достижимость, живость, безопасность) подробно не рассматриваются, даётся лишь краткое упоминание и ссылка на книгу Питерсона [8].

Тема 9. Этапы жизненного цикла имитационной модели. Этапы рассматриваются по Р. Шеннону: постановка цели моделирования; построение концептуальной модели; проверка адекватности концептуальной модели; выбор языка и средств моделирования; программирование модели; отладка модели; проверка адекватности модели; планирование и проведение экспериментов с моделью; анализ результатов. Проводится рассуждение о том, что в ИМ есть "наука", а что - "искусство". Показывается построение модели сетевого коммутатора, с целью сравнения вариантов построения с очередями на входе и на выходе. Даны краткие соображения по приемам отладки и проверки адекватности (валидации) моделей. Примечание: более детально математические методы, используемые при проверке адекватности модели и планировании эксперимента, рассматриваются в теме 10.

Тема 10. Математическая поддержка имитационного моделирования. В основном, в данной теме рассматривается применение методов теории вероятностей и математической статистики. Рассматриваются причины появления случайных величин в имитационных моделях. Кратко рассматриваются задачи: оценка необходимого числа экспериментов для снижения дисперсии результата до требуемого порога; подбор распределения случайной величины по выборке. Рассматривается аналитическая

модель простейшей системы массового обслуживания M/M/1, приводятся формулы загруженности обслуживающего прибора и длины очереди. Эта модель очень часто используется для валидации моделей систем с очередями. Далее, рассматриваются способы выбора наборов входных параметров, позволяющие сократить количество экспериментов (стратегическое планирование по Шеннону) и способы оценки необходимой продолжительности эксперимента (тактическое планирование).

Тема 11. Обзор возможностей системы моделирования ns-3. Система ns-3 ориентирована на построение моделей компьютерных сетей, со стеком протоколов TCP/IP. В ней имеется большое количество моделей сетевых протоколов уровней L1-L4. Рассмотрена иерархия компонентов ns-3, описаны процессы "сборки" стека протоколов на узле и соединения узлов. Показаны возможности по наблюдению за работой модели (сбор трассы).

Тема 12. Моделирование с непрерывным временем. Хотя для моделирования ВС используется в основном дискретное время, тема представляется важной для целевой аудитории, так как непрерывные модели очень полезны, например, для моделирования внешней среды объёкта, управляемого системой реального времени. Также "непрерывное время" используются для моделирования методов обработки сигналов в цифровой связи. Наконец, изучение данного подхода к построению моделей нужно для целостного восприятия принципа имитационного моделирования.

Изложение начинается с простейшего примера модели движения стрелок часов для подсчета количества их пересечений за 12 часов. Далее, описывается подход системной динамики и иллюстрируется примером модели в системе AnyLogic. Кратко упоминаются популярные средства Matlab и Simulink. Несколько более подробно рассматриваются их заменители с открытым кодом SciLab и Xcos. Приводится пример модели входного выпрямителя блока питания. Приводятся примеры элементов библиотек непрерывных моделей, применяемых для моделирования аналоговых систем управления (примечание: физические реализации указанных элементов могут использоваться для построения аналоговых моделей).

Тема 13. Эмуляторы системы команд ЦП. Эмулятор, по сути дела, является имитационной моделью ЦП, ориентированной главным образом на **замещение** моделируемого объекта - центрального процессора. Тем не менее, рассказывается об эмуляторах, обнспечивающих наблюдение за работой ЦП на уровне его микроархитектуры. Рассматриваются основные принципы построения эмуляторов: интерпретация, статическая трансляция,

динамическая трансляция. Описываются уровни точности учёта времени. Приводится пример решения, которое обеспечивает высокую скорость эмуляции при сохранении потактовой точности. Даётся краткий обзор эмулятора QEMU. Коротко рассматривается описанный в теме 4 Valgrind как эмулятор абстрактной вычислительной машины.

Тема 14. Моделирование аппаратных средств ВС на уровне интегральных схем. В начале темы описывается предмет исследования. Приводится классификация СБИС по области применения. Приводится таблица затрат на подготовку производства БИС, из которой следует невыгодность производства БИС малыми тиражами. Описываются вентильные матрицы и программируемые логические матрицы. Приводятся примеры зарубежных и отечественных ПЛИС, а также примеры отечественных процессоров и систем-на-кристалле (СнК). Далее, перечисляются типовые уровни моделирования (и проектирования) СБИС: электрических цепей; вентилей; регистровых передач; системный. Кратко описываются: язык Verilog, библиотека SystemC. Описываются основные этапы процесса проектирования БИС, указывается роль моделирования и оптимизации в нём. Вводится понятие виртуального прототипа аппаратных средств ВС.

Каждая из тем может требовать от половины лекции до двух. Общий объём оценивается в 13-14 двухчасовых лекций и будет уточнён после прочтения курса в 2022 г.

4. Практические задания

Планируются к выдаче следующие задания:

Задание 1. Измерение характеристик производительности программы на ПК. Преобразование программы для улучшения измеренных характеристик.

Задание 2. Модификация простой имитационной модели сервера с двумя клиентами. Добавление нового клиента, сервера, планировщика.

Задание 3. Построение диаграммы состояний для ряда объектов (по вариантам), в том числе процесса ОС Linux.

Задание 4. Построение имитационной модели некоторого фрагмента сети из библиотечных компонентов и её исследование.

Задания выдаются в четырёх вариантах. Разделение по вариантам выполняют сами студенты в начале курса.

Заключение

Описанный курс частично построен на основе классической литературы по имитационному моделированию. Однако, в нём вместо программных средств 70-80х годов рассматриваются современные средства построения имитационных моделей, в частности, OMNet++, ns-3, Scilab+Xcos, с открытым исходным кодом. Кратко упоминается популярная отечественная система AnyLogic, у которой есть бесплатная студенческая лицензия. В курсе также рассматриваются смежные (по отношению к имитационному моделированию) темы: производительность ВС, наблюдение за работой ВС, эмуляторы, некоторые аспекты аппаратных средств ВС, роль ИМ в разработке этих средств.

К перспективам развития следует отнести: увеличение числа примеров построения концептуальной модели (как в учебных материалах, так и в задачах); подготовка дополнительных примеров валидации модели; построение имитационных моделей для использования в других курсах кафедры АСВК (“Введение в сети ЭВМ”, “Дополнительные главы компьютерных сетей”), в том числе, с привлечением студентов к работе.

Литература

1. Р. Шеннон. Имитационное моделирование систем – искусство и наука. – М:Мир, 1978.

2. Аверилл М.Лоу, В. Дэвид Кельтон. Имитационное моделирование. 3-е издание. // СПб:Питер, 2004. – 847 с.

3. Б.А. Советов, С.А. Яковлев. Моделирование систем – М:Высшая школа, 2001.

4. О.М. Замятина. Моделирование систем – Томск, 2009.

5. Н.Н. Лычкина. Имитационное моделирование экономических процессов. Учебное пособие. - М:Инфра-М - 2014. - 254 с.

6. Боев В. Д. Компьютерное моделирование: Пособие для практических занятий, курсового и дипломного проектирования в AnyLogic7:. — СПб.: ВАС, 2014 — 432 с.

7. Р.Л. Смелянский. Методы анализа производительности вычислительных систем. Методическая разработка. Изд-во МГУ, 1990. (название будет уточнено)

8. Дж. Питерсон. Теория сетей и моделирование систем. – М:Мир, 1984.

Бахмуров А. Г., Бодров А. О.

Масштабирование приложения с помощью средств docker swarm и HAproxy

Введение

Приложения для сбора и обработки данных, например, в сетях Интернета вещей, могут собирать данные с большого количества (сотни, тысячи) клиентов одновременно. Во избежание роста задержек на обработку данных и их потерь, необходима масштабируемость приложения, т.е. увеличение его производительности (при росте числа клиентов) пропорционально дополнительным ресурсам (оборудованию), причём автоматическое. В настоящей статье авторы рассмотрели масштабирование на примере разрабатываемого на факультете ВМК МГУ сервиса сбора медицинских данных. Однако, описанный подход и архитектура могут быть применены и для других приложений.

1. Виртуализация приложения

Хорошей практикой является виртуализация приложения. Виртуализация позволяет обеспечить независимость приложения от вычислительной инфраструктуры. В настоящее время основными способами виртуализации являются:

1. Виртуальные машины (ВМ)
2. Контейнеризация [1]

1.1 Виртуальные машины (ВМ)

Виртуальная машина – это программная и/или аппаратная система, эмулирующая аппаратное обеспечение некоторой вычислительной платформы (называемой гостевой платформой) и исполняющая программы для гостевой платформы на платформе-хозяине, изолирующая от платформы-хозяина приложения, двоичные файлы и операционные системы. Виртуальная машина обладает всеми виртуальными устройствами и виртуальным жёстким диском, на который устанавливается

гостевая операционная система, драйверы устройств и прочие компоненты. Виртуальное оборудование с точки зрения приложения функционирует, как реальное, при этом виртуальная машина изолирована от реального компьютера, однако может иметь общие каталоги файловой системы с ним. Различные варианты виртуализации гостевой платформы и соответствующее ПО рассматриваться в данной работе не будут.

Использование ВМ вызывает дополнительные расходы на виртуальное оборудование, запуск гостевой ОС и поддержку необходимого окружения для работы приложений.

1.2 Контейнеры

Контейнеризация – это метод легковесной виртуализации, при котором вместе упаковываются и изолируются от домашней ОС приложение, двоичные файлы и библиотеки. В отличие от виртуальных машин контейнеры обеспечивают виртуализацию на уровне операционной системы [2], а не аппаратного обеспечения (см. Рис.1). Благодаря виртуализации на уровне ОС, то есть изоляции только приложения, двоичных файлов и библиотек, возможно быстрое развёртывание, простое масштабирование. Также уменьшается объём занимаемого места на жёстком диске по сравнению с виртуализацией с помощью виртуальных машин.

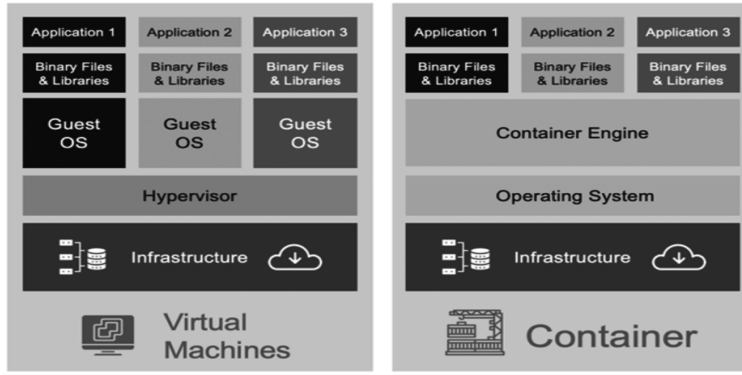

Рис.1 Сравнение технологий виртуализации виртуальных машин и контейнеров

2. Описание реализации

2.1 Исходные данные

Схема работы подвергаемого масштабированию приложения

В качестве примера приложения, на котором была рассмотрена реализация масштабируемости, взята разрабатываемая на факультете ВМК МГУ реализация сервиса мониторинга физиологических показателей человека. На рис.2 показана упрощённая схема работы данного сервиса [3].

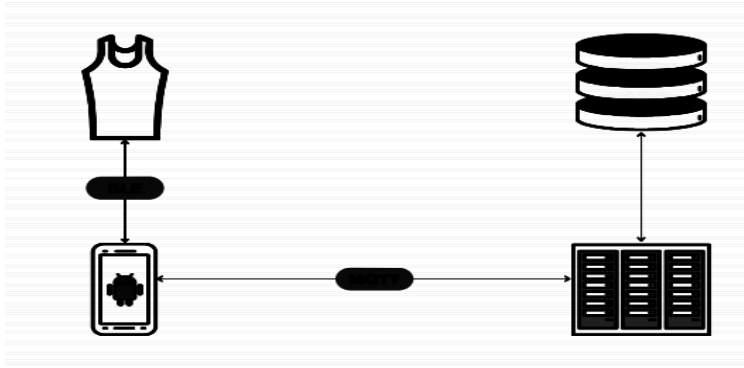

Рис.2 Схема работы сервиса по сбору медицинской телеметрии

В работе сервиса можно выделить следующие этапы:

1. Показатели физиологического состояния пользователя считываются с помощью датчиков умной одежды Hexoskin, затем отправляются на мобильный телефон на операционной системе Android по протоколу Bluetooth Low Energy (BLE).

2. Считанные по BLE показатели принимаются клиентским приложением на мобильном телефоне и отправляются на сервер по протоколу MQTT.

3. Серверное приложение записывает полученные данные в базу данных, откуда их можно выгружать для обработки.

Передача данных между клиентом и сервером выполняется по протоколу MQTT[4].

MQTT (message queuing telemetry transport) – это сетевой протокол прикладного уровня, работающий поверх TCP/IP. MQTT

ориентирован на обмен сообщениями между устройствами по принципу издатель-подписчик. Клиент может быть, как издателем или подписчиком, так и иметь обе роли одновременно. Обмен происходит через центральный сервер, который называется брокером (см. Рис.3). Клиенты взаимодействуют с брокером через TCP-соединение. Каждое публикуемое сообщение содержит поле темы (topic, топик), и, собственно, публикуемые данные. Клиент имеет возможность подписаться на определенные топики, если заинтересован в получении конкретных данных. Топики имеют иерархическую структуру в виде дерева, что упрощает их организацию и доступ к данным.

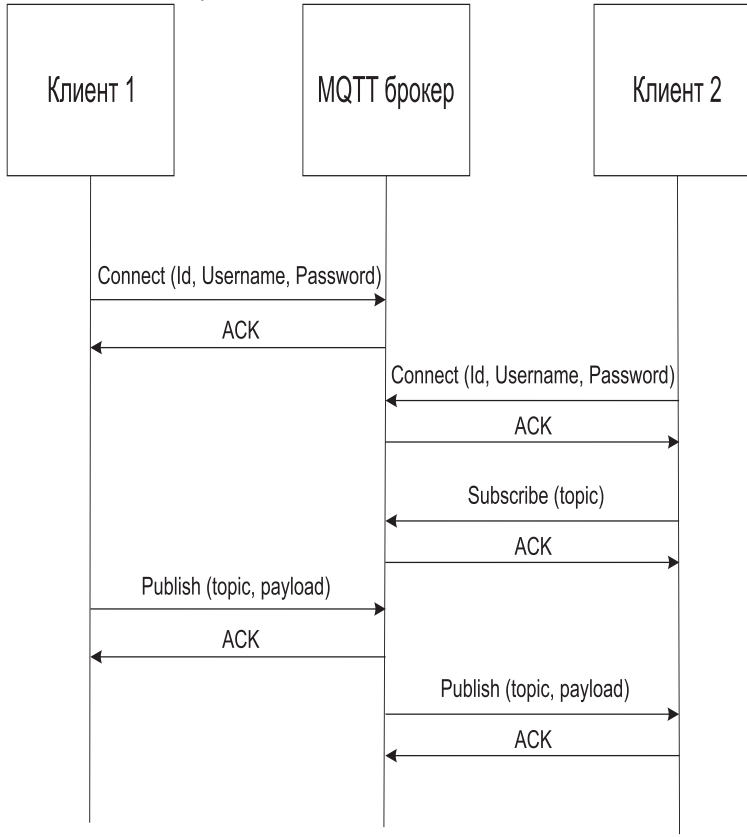

Рис.3 Схема работы протокола MQTT

Для протокола MQTT выделяется порт 1883. При использовании TLS (криптографического протокола, обеспечивающего защищённую передачу данных) для защиты

соединения между клиентом и сервером вместо 1883 используется порт 8883.

Масштабирование приложения и балансировка нагрузки

Под масштабируемостью следует понимать способность системы справляться с увеличением рабочей нагрузки путём добавления ресурсов. Масштабируемость можно разделить на горизонтальную и вертикальную. Горизонтальная масштабируемость, рассматриваемая в данной статье, - это запуск в параллельную работу нескольких экземпляров приложения, каждый из которых обрабатывает некоторую часть запросов пользователей. В обеспечение горизонтального масштабирования и отказоустойчивости используется балансировка нагрузки – процесс распределения задач (запросов) между несколькими серверами, исполняющими экземпляры приложения.

В качестве балансировщика нагрузки было выбрано средство HAProxy [5].

Средство HAProxy может быть запущено в двух различных режимах – TCP и HTTP. Режимы различаются доступной функциональностью. В режиме TCP каждая публикация клиента в рамках одного TCP-соединения приходит к одной конкретной копиии брокера. В режиме HTTP каждый новый топик приходит на новый брокер. В настоящей работе стоит задача балансировки самого TCP-соединения, а протокол MQTT работает поверх TCP, поэтому HAProxy используется в режиме TCP.

Клиент отправляет запросы HAProxy, выступающему в качестве балансировщика, и получает ответ на каждый. HAProxy распространяет запросы по узлам с копиями серверного приложения, взаимодействуя в каждом случае с одним MQTT-брокером (см. Рис.4) [6]. Узел может либо принять запрос, если он не перегружен, либо вернуть значение null или ошибку, тогда HAproxy передаст запрос следующему узлу. Если ни один из узлов не сможет предоставить услугу, то HAproxy вернёт клиенту null или ошибку [7]. Распределение нагрузки между контейнерами идёт в соответствии с выбранным для HAproxy алгоритмом балансировки. Для описываемой реализации выбран алгоритм Least Connections, он предназначен для соединений с длительным временем жизни. Least Connections является динамическим алгоритмом балансировки, и каждый запрос передаётся на наименее нагруженный сервер.

Обоснование архитектуры средства балансировки и масштабирования приложения

При разработке структуры соединений между брокером MQTT и экземпляром приложения будем исходить из требования избежать:

- дублирования данных в БД, чтобы не тратить ресурсы на последующее удаление дубликатов;
- потери данных;
- “узких мест”.

При подключении единственного брокера ко всем экземплярам приложения возникает проблема «узкого места» при взаимодействии клиентов с брокером. Также возникает дублирование попадающих с базу данных сообщений. Если избежать этого и распределить клиентов по экземплярам приложения, то возникает проблема синхронизации. При выходе из строя одного из экземпляров приложения необходимо сообщить об этом остальным (иначе произойдёт потеря данных), а это само по себе является нетривиальной задачей.

Если каждый экземпляр приложение подписывается на каждый из имеющихся брокеров, то также возникает проблема дублирования сообщений.

Поэтому каждый экземпляр приложения соединен с одним брокером, и каждая такая пара упакована в Docker-контейнер (см. Рис.4). Клиент устанавливает TCP-соединение через балансировщик с одним из брокеров, которые не перегружены. Балансировщик также упакован в контейнер. При выходе из строя одного из контейнеров, приложений или брокеров, TCP-соединение с соответствующим MQTT-брокером разрывается, и переустанавливается на другой работающий и свободный брокер, по выбору балансировщика.

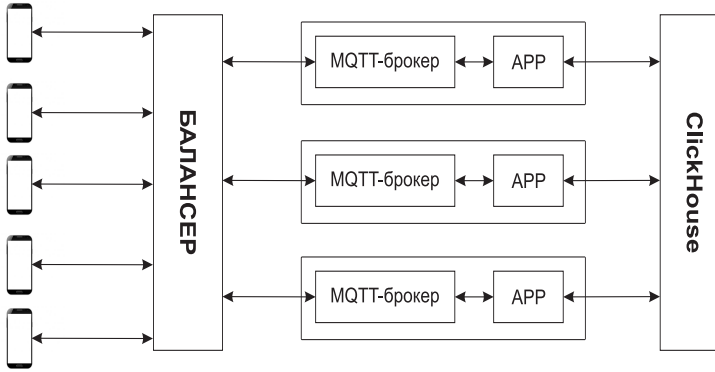

Рис.4 Балансировка нагрузки между копиями серверной части приложения

3. Автоматическое масштибирование приложения

Анализ программных средств для автоматического масштабирования Docker-контейнеров показал, что для данного проекта не обязательно использовать популярное средство Kubernetes (k8s), можно использовать Docker Swarm, имеющее намного меньший порог вхождения в технологию и намного более простое в настройке. Docker Swarm (также называемое SwarmKit) – это средство для управления кластерами и контейнерами, которое интегрировано в Docker Engine. Кластер Docker Swarm состоит из docker-хостов, которые называются нодами: каждая нода может быть управляющей (manager) или подчиняться управляющей (worker), см. Рис. 5.

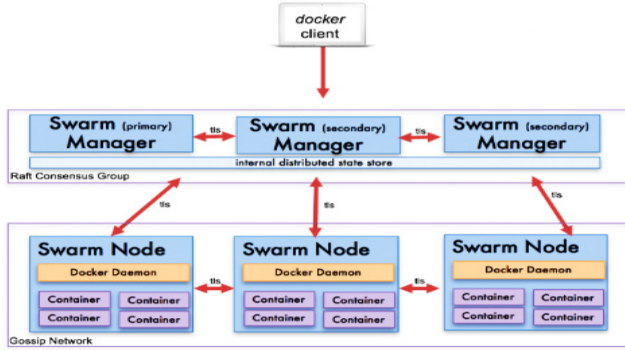

Рисунок 5 - Структура кластера docker swarm [8]

Ноды можно запускать на одном физическом или виртуальном хосте или же распределять их по нескольким хостам. Чтобы развернуть приложение на нодах, всегда нужна управляющая нода (manager). Сервисом называются задачи, которые назначены какой-то из нод. Сервис – ключевое понятие экосистемы swarm и основной элемент взаимодействия пользователя с системой. Ноды worker получают и исполняют запросы от менеджера, а также уведомляют менеджера о текущем состоянии данных им заданий.

При создании сервиса в SwarmKit указываются его параметры: количество копий, доступ к сети, открытые порты и другие сервисы, доступные извне. Преимущество архитектуры docker swarm перед отдельно работающими контейнерами в том, что при прекращении работы одной из нод, её задачи перекладываются на другую, а не теряются. Ещё одним преимуществом использования swarm-режима является то, что при изменении конфигурации контейнеров отсутствует необходимость перезапускать их вручную. Docker сам обновит конфигурацию и перезапустит контейнеры с новыми параметрами.

По умолчанию запуск контейнеров в swarm-режиме недоступен, чтобы запустить swarm-режим достаточно проинициализировать swarm на одну текущую ноду:
docker swarm init

Затем необходимо создать две сети: одну – для мониторинга, вторую – для непосредственной работы приложения:
docker network create -d overlay monitoring
docker network create -d overlay iomt

После выполнения данной команды текущая нода становится менеджером. Далее можно присоединять новые ноды (как manager, так и worker) к созданному кластеру. Для настройки контейнеров

будем использовать стандартное средство Docker docker-compose. Для этого опишем необходимые нам контейнеры Orbiter, контейнер с сервисом run.py и mqtt-брокером, web-site (контейнер с nginx и gunicorn) и haproxy. Все контейнеры, кроме Orbiter, используют свой Dockerfile для построения пользовательского образа. В swarm-режиме недоступно построение образов «на лету», поэтому необходимо создать хранилище, куда и откуда будут загружаться построенные образы. Сделать это можно следующей командой:
docker service create –name registry –publish published=5000,target=5000 registry:2

Загрузить образы в хранилище можно следующей командой:
X docker-compose push

Затем, следующая команда непосредственно разворачивает ноду, используя загруженные в созданный выше репозиторий образы и публичные образы с Dockerhub, и сервисы на ней:
docker stack deploy –compose-file docker-compose.yml iomt

Чтобы провести масштабирование в автоматическом режиме, нужно настроить систему мониторинга, для которой была создана сеть monitoring. Система мониторинга позволит контролировать загруженность сервиса сбора и обработки данных через специальные метрики. В настоящей работе будем анализировать количество запросов, проходящих через HAproxy, так как данная метрика напрямую коррелирует с загруженностью непосредственно серверов. В зависимости от величины этой метрики будет производиться масштабирование. Контейнер с Orbiter увеличивает или уменьшает количество контейнеров с MQTT-брокером и MQTT-клиентом после запросов, которые ему отсылает Alertmanager [9], программное средство, которое обрабатывает предупреждения по изменяющимся системным параметрам. Alertmanager следит за метриками на платформе мониторинга нашей системы. В качестве такой платформы использован Prometheus [10]. Он получает системные метрики балансировщика HAproxy через Haproxy-exporter [11], специальное ПО для экспорта статистики и системных метрик HAproxy в Prometheus. Таким образом, автомасштабируемая платформа, представленная в настоящей работе, имеет архитектуру, представленную на рисунке 6.

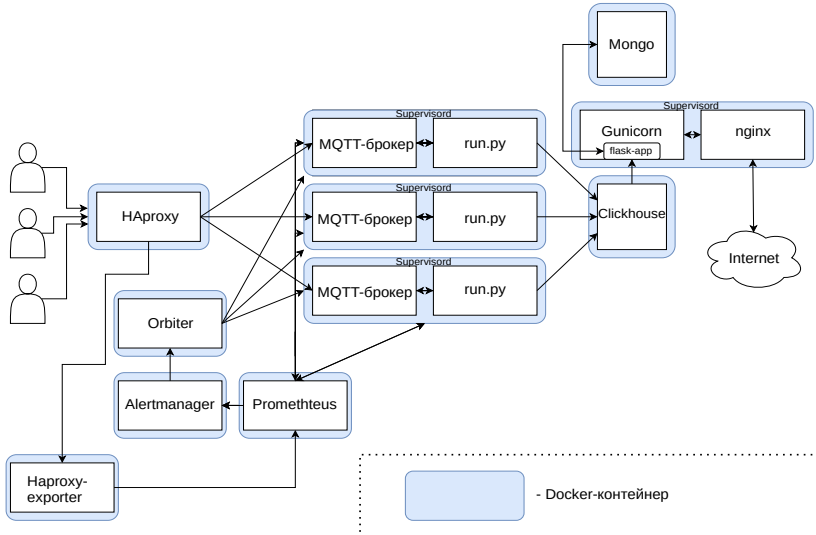

Рис.6. Архитектура автомасштабируемой платформы

4. Экспериментальное исследование

В качестве экспериментального исследования проведено нагрузочное тестирование с помощью утилиты mqtt-benchmark [12]. Цель тестирования – замерить среднюю скорость доставки запросов от клиента до приложения в зависимости от количества контейнеров. В экспериментах участвует масштабируемая версия платформы. Запросы отправляются через haproxy на одну из копий MQTT-брокера. Параметр QoS (Quality of Service) обозначает соглашение между отправителем и получателем, которое гарантирует доставку сообщений [13]. Выделяют три уровня гарантии доставки сообщений QoS:

- Максимум один раз (0);
- Хотя бы один раз (1);
- Ровно один раз (2).

По умолчанию запуск происходит с QoS=1. В проводимом эксперименте мы также будем запускать тестирование с таким параметром (QoS=1 по умолчанию). Нагрузочное тестирование выполняется посредством команд следующего вида:

mqtt-benchmark –broker tcp://172.30.7.214:1344 –count 20 –clients 200 –wait 10 –topic c/test - -format text –ramp-up-time 5

Здесь

- –count - количество запросов от каждого из клиентов (что обозначает, по сути, длительность эксперимента);
- –clients - количество клиентов, запущенных параллельно и, соответственно, параллельно отправляющих запросы;
- –wait – время ожидания подтверждения прихода сообщения (мс);
- –topic – топик для отправляемых сообщений;
- –ramp-up-time – таймаут на создание клиентов (с). Сообщения от каждого клиента по умолчанию отправляются с интервалом в 1 секунду.

В качестве измеряемых величин используем количество дошедших сообщений до MQTT-брокеров в течение времени, равному wait, и среднее время доставки сообщений до них. Будем изменять количество контейнеров и параллельно подключённых клиентов (–clients) и наблюдать, как изменяются заданные нами метрики в зависимости от данных величин. В результате экспериментального исследования были получены результаты, изображённые на рисунках 7 и 8. Как видно из диаграмм, гипотеза об увеличении доли дошедших сообщений (часть сообщений не доходит в любом случае, так как специально для эксперимента установлено малое время ожидания их прихода с помощью флага –wait) и уменьшении среднего времени доставки сообщения с увеличением числа контейнеров для разработанного решения поставленной задачи, подтвердилась.

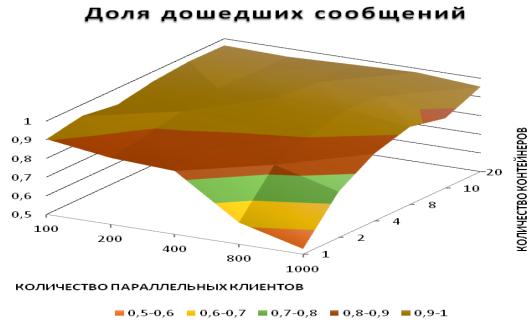

Рис.7 Доля дошедших сообщений до MQTT-брокера

Рис.8 Среднее время доставки сообщений до MQTT-брокера

5. Благодарности

Авторы благодарят сотрудника компании "Яндекс"А.А.Любицкого за техническую помощь и консультации по используемым в проекте программным средствам.

Заключение

Разработанная архитектура с использованием нескольких программных средств с открытым исходным кодом позволяет улучшить производительность приложения за счёт автоматического горизонтального масштабирования и балансировки нагрузки. Это было подтверждено в ходе эксперимента, поставленного над платформой мониторинга, разрабатываемой на факультете ВМК МГУ. Подобная архитектура довольно универсальна и применима к любым приложениям, использующим MQTT в своей работе, поэтому может использоваться и в других разработках.

Литература

1. A.M.Potdar, D.G.Narayan, S.Kengond, and M.M.Mulla, “Performance Evaluation of Docker Container and Virtual Machine,” Procedia Comput. Sci., vol. 171, no. 2019, pp. 1419–1428, 2020, doi: 10.1016/j.procs.2020.04.152.

2. Akf partners. [Электронный ресурс]. – режим доступа

   `https://akfpartners.com/growth-blog/vms-vs-containers`

3. Ю.В. Аникевич “Разработка и реализация серверной части платформы для сбора и обработки медицинской телеметрии”, Дипломная работа, 2021. [Электронный ресурс]. – режим доступа:

   `https://drive.google.com/file/d/1cdTAQjV7uynd5N90CJDL1wrhwguX1cQ0/view`

4. MQTT Version 3.1.1 documentation. [Электронный ресурс]. - режим доступа

   `http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.pdf`

5. HAproxy Documentation. [Электронный ресурс]. - режим доступа:

   `https://www.haproxy.com/`

6. “How to build an high availability MQTT cluster for the Internet of Things” [Электронный ресурс] – режим доступа

   `https://medium.com/@lelylan/how-to-build-an-high-availability-mqtt-cluster-for-the-internet-of-things-8011a06bd0`

7. Marischa Elveny, Ari Winata, Baihaqi Siregar and Rahmad Syah “A tutorial: Load balancers in a container technology system using docker swarms on a single board computer cluster”, EEO, vol. 19, issue 4, no. 2020, pp.744-751, doi: 10.17051/ilkonline.2020.04.178

8. T. Yudi Hadiwandra, Feri Candra “High Availability Server Using Raspberry Pi 4 Cluster and Docker Swarm”, IT journal research and development, 2021, vol. 6. Issue 1, pp.43- 51, doi: 10.25299/itjrd.2021.vol6(1).5806

9. Alertmanager Documentation. – [Электронный ресурс] – режим доступа:

   https://prometheus.io/docs/alerting/latest/alertmanager/

10. Prometheus Documentation. – [Электронный ресурс]

    https://prometheus.io/docs/introduction/overview/

11. Haproxy-exporter Github. – [Электронный ресурс]

    https://github.com/prometheus/haproxy_exporter

12. MQTT-benchmark Github. – [Электронный ресурс]

    https://github.com/krylovsk/mqtt-benchmark

13. “Quality of Service (QoS) 0,1 and 2 MQTT Essentials: Part 6”, hivemq. – [Электронный ресурс]

    https://www.hivemq.com/blog/mqtt-essentials-part-6-mqtt-quality-of-service-levels/

Гуров С. И., Смелянский Р. Л., Ержанов Ж. Р.

Сетевое кодирование в троичных полях

Введение

В статье предложен новый метод сетевого кодирования, на основе перехода от двоичного к троичному представлению данных. Предложенный метод позволяет сократить задержки при передаче данных, поскольку отпадает необходимость агрегировать один и тот же кадр с несколькими другими кадрами в разные пакеты. Также становится не нужным обеспечивать асинхронную доставку двух или более агрегированных кадров в одну и ту же точку сети для выделения исходного кадра. Кроме того, предложенный метод устраняет влияние блокировок на выходе коммутатора на его пропускную способность, позволяя сократить повторную передачу данных из-за ошибок передачи.

1. Сетевое кодирование

Сетевое кодирование является относительно новой технологией в построении компьютерных сетей передачи данных. Оно позволяет увеличить пропускную способность сети, повышая её эффективность. Суть данной технологии заключается в том, что узлы сети на L2 уровне вместо пересылки традиционных кадров агрегируют несколько кадров в один, без увеличения размера полученного агрегированного кадра. Промежуточные коммутаторы однозначно восстанавливают исходные кадры по нескольким полученным агрегированным [1].

Для агрегирования кадров часто применяют метод называемый *линейным сетевым кодированием (Linear Network Coding, LNC)* [2]. В нём для агрегирования используется операция побитовой суммы XOR двух кадров. Достоинства и недостатки этого метода хорошо изучены [2].

Достоинствами LNC являются повышение пропускной способности, защита от атак типа Wiretapping, использование естественных свойств беспроводных технологий, а также сокращение задержки на выходе коммутатора. Недостатками — дополнительная вычислительная нагрузка на коммутаторы сети, большая уязвимость перед скомпрометированными

коммутаторами, сложность внедрения в существующие сетевые устройства, а также проблемы дублирования и задержки из-за асинхронной передачи кадров в сети. При использовании LNC неизбежно дублирование одних и тех же кадров в нескольких агрегированных. К тому же кадры приходят к месту назначения разными путями, т. е., вообще говоря, асинхронно. Поэтому узел вынужден будет ждать, пока не получит всю информацию, нужную для декодирования агрегированного кадра.

2. Поля Галуа. Помехоустойчивое избыточное кодирование

Мы предлагаем рассмотреть другой подход к сетевому кодированию, а именно перейти от двоичного к троичному алфавиту представления данных, что обеспечит более компактное представление передаваемых кадров. Для будем рассматривать передаваемые данные как элементы конечного поля (поля Галуа). Напомним кратко известные свойства этих полей.

Поля Галуа, символически $GF(q)$, содержат конечное число $q = p^t$ элементов, которое называют *порядком* данного поля. Здесь p — простое число, *характеристика* поля $GF(p^t)$, а t — натуральное, степень его *расширения*. При $t = 1$ поле Галуа называют *простым*.

Все поля одинакового порядка q эквивалентны с точностью до изоморфизма. Мультипликативная группа поля Галуа является циклической. Элементы поля Галуа $GF(p^t)$ могут быть представлены как многочлены $f(x)$ степени не выше $t - 1$ с коэффициентами из простого поля $GF(p) = \{0, 1, \ldots, p - 1\}$, x — формальная переменная.

Такое представление элементов конечного поля называют *полиномиальным*. Тогда сложение и умножение элементов поля Галуа определяется как обычные сложение и умножение многочленов. При этом сложение коэффициентов проводят по $\bmod p$, а умножение — по модулю некоторого неприводимого многочлена степени t над $GF(p)$.

Поля Галуа находят большое применение в телекоммуникации. Их используют для построения избыточных кодов, исправляющих или обнаруживающих ошибки передачи данных. При таком кодировании вместо исходного слова (*сообщения*) передают записанное в некотором алфавите представляющее его *кодовое слово*. Традиционно длину сообщения обозначают k, а кодового слова — n, $n \geq k$. При $k = 0$ или $n = k$ говорят о *тривиальных*

кодах. Часто кодовое слово представляет собой исходное сообщение, пополненное дополнительными символами, называемые *избыточными*. Число $n - k$ избыточных символов также традиционно обозначают m. *Скорость кода* R есть отношение k/n.

Код есть совокупность всех кодовых слов. *Мощность кода* над полем $GF(q)$ обозначим $Q = q^k$, $q = p^t$. Код C образует k-мерное подпространство мощности $Q = |p^{tk}|$ линейного векторного пространства $GF^n(p^t)$ мощности $|p^{tn}|$.

Минимальное хэммингово расстояние между словами кода C называется его *кодовым расстоянием*, символически $d(C)$ или просто d. Ясно, что если при передаче произошло искажение не более $r = \lfloor (d-1)/2 \rfloor$ символов кодового слова (без их вставок или выпадений), то возникшие ошибки можно исправить декодированием принятого слова в ближайшее кодовое. Будем рассматривать коды, ориентированные на исправление ошибок только указанного типа.

Считаем далее, что сообщениями являются все двоичные слова длины k, а кодовые слова задаются над полем Галуа $GF^n(3^t)$. Тогда код C с кодовым расстоянием d обозначатся $[n, k, d]_{3^t}$. При этом ясно, что элемент расширенного поля $GF(p^t)$ можно представить как блок из t элементов простого поля $GF(p)$.

3. Троичная система счисления

Традиционно для представления сообщений и кодовых слов используют двоичный алфавит $GF(2) = \{0, 1\}$. Однако в последнее время рассматриваются и применяется недвоичные коды над $GF(2^s)$, элементами которых являются не биты, а блоки (на практике чаще всего байты: $s = 8$) битов. В данной работе предлагается использовать сжатие данных, основанное на переходе к алфавиту кодирования элементами поля характеристики 3.

Для оценки эффективности той или иной системы счисления важнейшим параметром является её *экономичность* — запас чисел, которые можно записать в данной системе с помощью определенного количества знаков. В p-ичной системе счисления используя z знаков можно представить не менее $p^{\lceil z/p \rceil}$ чисел. Например, используя $z = 30$ знаков в десятичной системе счисления можно представить $10^3 = 1\,000$ различных чисел (по 10 цифр в каждом разряде), в двоичной — $2^{15} = 32\,768$, в троичной — $3^{10} = 59\,049$.

Легко показать, что троичная система является наиболее экономной из всех систем счисления (с целым основанием) [3]. Ясно

что, например, при числе входов больше двух, перенос в следующий разряд в троичном сумматоре будет происходить реже, чем в двоичном [3,4]. Всё это делает её троичную систему удобной как для реализации сжатия сообщений, так и при конструирования средств вычислительной техники.

Троичная система счисления также обладает рядом других важных свойств. Удобной формой представления троичных чисел наряду с *естественной*, использующей алфавит $\{0, 1, 2\}$, является т. н. *сбалансированная система* с алфавитом $\{-, 0, +\}$, в котором полагают $- < 0 < +$. Правила сложения и умножения тритов в сбалансированной системе очевидны, и мы не будем их здесь указывать. В табл. 1 даны приведены представления всех девяти возможных двузначных чисел в этой системе.

-4	-3	-2	-1	0	$+1$	$+2$	$+3$	$+4$
$--$	-0	$-+$	$0-$	00	$0+$	$+-$	$+0$	$++$

Таблица 1: Троичные двузначные числа в сбалансированной системе

Сбалансированная система обладает следующими полезными свойствами [3,5]:

- позволяет обойтись без использования дополнительного или обратного кода;
- противоположное число находится инвертированием ($+ \leftrightarrow -$) слова;
- при сравнении чисел по величине не нужно учитывать знак числа, что в два раза уменьшает время операции ветвления по знаку.

4. Два способа обеспечения помехоустойчвости при преобразовании $2 \to 3$

Как было отмечено, использование троичной системы в промежуточных узлах сети увеличивает производительность коммутаторов, за счет уменьшения длительности блокировки на выходе.

Отметим, что наибольную пропускную способность имеют коммутаторы с буферизацией на выходе (минимальная задержка

при загрузке не более 90%). Однако, такие коммутаторы предъявляют высокие требования к объему и скорости буферной памяти [6].

Предположим, что на одном и том же выходе коммутатора появилось s кадров. длиной ℓ бит каждый. Соберем все кадры в буфере в битовую матрицу $S_{s\times\ell}$. Каждый столбец этой матрицы можно представить как элемент поля $GF(2^s)$.

Определим тернарное поле $GF(3^n)$ минимального расширения n, для которого имеется инъективное отображение

$$F : GF(2^s) \to GF(3^n), \quad n = \lceil s/\log_2 3 \rceil . \tag{1}$$

Для реальных рассматриваемых величин расширений значение $\log_2 3$ можно округлить до $1.58 = \mu$. Инъективность F обеспечит однозначное восстановление двоичного кода по его троичному образу.

При переходе от двоичных данных к троичным оказывается возможным дополнительно обеспечить помехозащищённость передаваемой информации. Это возможно сделать двумя путями.

Во-первых, поскольку размер области определения отображения F строго меньше размера области значения, *избыточное пространство* $GF(3^n) \setminus \operatorname{Im} F$ можно использовать для передачи дополнительных битовых сообщений, для исправления или обнаружения ошибок, для передачи управляющей информации и т. д.

Во-вторых, при преобразовании F можно прямо использовать помехозащищённое кодирование, получая те или иные избыточные коды в различных полях характеристики 3.

5. Избыточное кодирование двоичных данных

Вначале рассмотрим пример. Пусть $s = 8$. Тогда $|GF(2^s)| = 256$, $n = \lceil 8/\mu \rceil = 6$, $|GF(3^n)| = 729$, т. е. примерно 2.85 раз больше, чем $|\mathrm{Im}F|$. Величину $L = 3^n/2^s$ назовём *избыточностью*.

Для преобразования столбцов исходной двоичной матрицы $S_{s\times\ell}$ в элементы $f(x)$ поля Галуа $GF(2^s)$, воспользуемся взаимнооднозначным соответствием между многочленами $f(x)$ с коэффициентами из $GF(2)$ и двоичными векторами, предполагая, что элементы столбца (сверху вниз) суть коэффициенты многочлена про убыванию степеней от $s-1$ до 0. Аналогично, после преобразования (1) многочлены, в виде элементов $GF(3^n)$,

можно представить как столбцы, где верхний элемент соответствует старшей степени многочлена, второй сверху элемент соответствует коэффициенту при степени $n-2$, и т. д.

Таким образом, входная матрица S размера $8 \times \ell$ преобразуется в матрицу T размера $6 \times \ell$. При одинаковой скорости пакетизации, получим выигрыш в скорости освобождения буфера $8/6 = 1.33$ раза, т. е. более 30%.

Покажем, как можно использовать избыточность. Поскольку $|729/256| \approx 2.85 > 2$, значит вместе с каждым столбцом высоты 8 можно также однозначно закодировать дополнительный бит данных. Если же изначально группировать столбцы, например, по 10, то с каждой группой столбцов получится закодировать 10-битное дополнительное сообщение.

Если в рассматриваемом примере выбрать $n = 7$, то избыточность будет равна приблизительно 8.54, т. е. почти в 3 раза больше, чем при $n = 6$. В этом случае, с каждым столбцом вместе с битом данных может быть передан ещё и один трит. Как можно видеть, при увеличении скорости освобождения буфера уменьшается размер избытка, и наоборот. Возникает естественный вопрос: как выбрать золотую середину?

Пусть, например, избыток используется для кодирования с исправлением ошибок. Количество исправляемых ошибок напрямую зависит от количества дополнительных разрядов в сообщении. Поэтому, если среда передачи данных не очень надежна (например, радиопередача), то имеет смысл пожертвовать выигрышем в скорости освобождения буфера, но увеличить избыточность. И наоборот, если среда передачи данных надежна (например, витая пара), то можно максимально уменьшить избыточность. Получается, что значение n зависит от s и от внешних факторов.

Ещё одним важным допущением является предположение об одинаковой длине всех кадров. Это предположение не всегда выполняется. Решением может быть приведение всех кадров к одинаковой длине, путем заполнения всех недостающих позиций, например, нулями. При этом, для получения оригинального кадра из “приведенного” будет использоваться информация о длине из заголовка кадра или дополнительное, закодированное в избытке сообщение.

На основе сказанного, предлагаемый метод можно сформулировать следующим образом:

1. В зависимости от внешних факторов и информации о коммутаторах в сети, выбираются значения для s и n.

2. В буфере выходного порта коммутатора собирается s кадров.

3. Кадры приводятся к одинаковой длине ℓ и преобразовываются в матрицу S размера $s \times \ell$.

4. Для каждого столбца размера s матрицы S производится преобразование (1) в столбец размера n, помимо исходного сообщения включающий и дополнительное.

5. Полученная матрица T размера $n \times \ell$ отправляется на физический уровень устройства передачи данных.

6. Устройство, принявшее матрицу T, применяет обратное преобразование $GF(3^n) \to GF(2^s)$ и однозначно восстанавливает исходные s пакетов.

6. Преобразование $GF(2^s) \to GF(3^n)$

В предыдущем разделе было продемонстрировано, как можно использовать избыточность в преобразовании $GF(2^8) \to GF(3^6)$. Конкретнее, с каждым столбцом высоты 8 кодируется дополнительный бит данных, что задействует 2 единицы из 2.85 возможных. Можно видеть, что полученная избыточность используется не целиком, а потому приведем пример преобразования, которое задействует всю избыточность.

Рассмотрим, как можно это сделать на том же примере преобразования $GF(2^s) \to GF(3^n)$, при $s = 8$, $n = 6$. Далее будет приведена общая схема, не зависящая от конкретных значений s и n. Пусть каждый входной столбец высоты 8 рассматривается как двоичная запись числа, где нижний элемент столбца отвечает за младший бит числа. Таким образом, каждый столбец представляет число из отрезка $[\,0, 255\,]$. Здесь и далее используется троичный алфавит $\{\,0, 1, 2\,\}$ с естественным порядком $0 < 1 < 2$.

Аналогично каждый выходной столбец высоты 6 может представлять запись числа в троичной системе счисления из отрезка $[\,0, 728\,]$. Пусть на первом шаге преобразование $GF(2^8) \to GF(3^6)$ преобразовывает двоичную запись числа из отрезка $[\,0, 255\,]$ в его троичную запись.

Сложность вычисления этого преобразования линейно зависит от длины n столбца c. Заметим, что для чисел из отрезка $[\,0,\, 242\,] = [\,0,\, 3^5 - 1\,]$, первый (старший) трит c_1 в троичной записи кодового слова (числа) c всегда равен 0. Для чисел из отрезка $[\,243,\, 255\,]$, имеем $c_1 = 1$.

Зная эту информацию заметим, что если значение полученного кодового слова c находится на отрезке $[\,13, 242\,]$, то его можно однозначно декодировать без знания значения c_1. В случае, когда $0 \leq c \leq 12$ или $243 \leq c \leq 255$, слово c однозначно декодируется значением в старшем трите: если $c_0 = 0$, то имеет место первый случай, а если $c_0 = 1$, то второй. Таким образом, на втором шаге преобразования $GF(2^8) \to GF(3^6)$ входное число сравнивается с пороговыми значениями 13 и 242 и далее, если c —

- находится в отрезке $[\,13, 242\,]$, то $c_1 \in \{\,0, 1, 2\,\}$;
- находится в отрезке $[\,243, 255\,]$, то $c_1 \in \{\,1, 2\,\}$;
- находится в отрезке $[\,0, 12\,]$, то в $c_1 = 0$ и в этом никакого дополнительного сообщения сообщения переслать не получится.

Пусть полученный столбец представляет число, которое равновероятно принимает значения из отрезка $[\,0, 255\,]$. Тогда дополнительно к основному сообщению с вероятностью $13/256$ — один бит, с вероятностью $230/256$ — один трит, и с вероятностью $13/256$ нельзя ничего дополнительно передать нельзя. Иными словами, имеем случайную величину χ, равномерно распределённую на $[\,0, 255\,]$, для которой в первом случае можно передать два различных сообщения, в которых будет зашифровано исходное число, во втором случае – три, и в последнем случае — лишь одно.

Для такой случайной величины, математическое ожидание числа различных сообщений, в которых можно зашифровать исходное число имеет вид:

$$\mathrm{E}[\chi] = 3 \cdot \frac{230}{256} + 2 \cdot \frac{13}{256} + 1 \cdot \frac{13}{256} = \frac{3^6}{2^8} \approx 2.85.$$

Из полученного значения следует, что предложенный метод кодирования использует всю избыточность, которая появляется при отображении $GF(2^8) \to GF(3^6)$.

Обобщим этот метод для s и n, удовлетворяющим ограничениям (1).

1. На вход подается бинарный столбец дины s, представляющий двоичную запись некоторого числа $b \in [\,0,\, 2^s - 1\,]$.
2. В зависимости от положения входного числа относительно пороговых значений $h_1 = 2^s - 3^{n-1}$ и $h_2 = 3^{n-1} - 1$ получаем выходной столбец следующим образом —

$0 \le b \le h_1 - 1$**:** первый элемент выходного столбца ставим равным 0, остальные элементы — в соответствии с троичной записью входного числа;

$h_2 + 1 \le b \le 2^s - 1$**:** первый элемент выходного столбца ставим равным 1 или 2, в зависимости от передаваемого дополнительного сообщения;

$h_1 \le h_2 \le r$**:** первый элемент выходного столбца ставим равным 0, 1 или 2, в зависимости от передаваемого дополнительного сообщения.

В общем случае, для вышеописанной случайной величины χ, равномерно распределённой на $[\,0,\, 2^s - 1\,]$, математическое ожидание числа различных сообщений, в которых можно зашифровать исходное число имеет вид:

$$\mathrm{E}[\chi] = 3 \cdot \frac{(3^{n-1} - 1) - (2^s - 3^{n-1}) + 1)}{2^s} + \\ + 2 \cdot \frac{(2^s - 1) - 3^{n-1} + 1)}{2^s} + \\ + 1 \cdot \frac{(2^s - 3^{n-1} - 1) - 0 + 1}{2^s} = \frac{3^n}{2^s}.$$

Итак, описанное преобразование работает следующим образом: имеется три отрезка, в которое может попасть входное число. В зависимости от того, в какой отрезок оно попадет, первый элемент выходного столбца может принимать либо одно, либо два, либо три значения. Из этого следует, что с какой-то вероятностью, передать дополнительное сообщение вообще не получится. Для борьбы с этим недостатком можно для шифрования одного дополнительного сообщения группировать несколько столбцов.

Рассмотрим преобразование $GF(2^8) \to GF(3^6)$. Пусть помимо самих битов, вместе с оригинальным столбцом надо отправить дополнительное сообщение размера один бит. Если преобразовывать лишь один столбец, то этот дополнительный бит можно будет отправить с вероятностью $1 - 13/256 = 243/256 \approx 0.95$. Если же для отправки этого сообщения преобразовывать два столбца вместо одного, то вероятность, что этот бит будет закодирован вместе с основным сообщением равна $1 - (13/256)^2 \approx 0.997$.

Заметим, что в зависимости от s и n удовлетворяющих ограничениям в (1), значение избыточности преобразования $GF(2^s) \to GF(3^n)$ находится в интервале $(1, 3)$.

Это значение можно увеличить, добавляя новые элементы к выходным столбцам. Иными словами, вместо преобразования $GF(2^s) \rightarrow GF(3^n)$, рассматривается преобразование $GF(2^s) \rightarrow GF(3^{n+1})$. В таком случае значение избытка будет находиться в интервале $(3, 9)$, то есть на каждый входной столбец можно зашифровать ещё как минимум один дополнительный трит данных.

В общем случае получим, что каждый дополнительный элемент выходного столбца можно использовать для передачи ещё одного дополнительного трита данных. Относительно входных и выходных пакетов это будет означать, что к выходной матрице будет добавлена ещё одна строка. К сожалению, добавление нового элемента к выходному столбцу означает, что в канал надо отправить на один кадр больше. Получается, что за избыточность мы платим скоростью освобождения буфера коммутатора. Потеря скорости, в свое время, зависит от значения параметра s.

В рассмотренном выше примере $s = 8$ и $n = 6$. Скорость освобождения буфера увеличилась в $8/6$ раз. Пусть теперь к выходным столбцам будет прибавляться ещё один элемент. Получим, что входные $s = 8$ кадров будут преобразованы в выходные $n + 1 = 7$ кадров. Добавив ещё один трит избыточности, мы потеряем $8/6 - 8/7 \approx 0.19$ единиц скорости.

Рассмотрим теперь другой пример, в котором $s = 48$, тогда $n = 31$. Теперь имеем, что добавление нового элемента в выходные столбцы влечет потерю в скорости освобождения буфера, равную $48/31 - 48/32 \approx 0.05$, что почти в четыре раза меньше, чем в первом примере, хотя прирост в избыточности остался таким же.

7. Коды, исправляющие ошибки

Дополнительные данные, которые вышеописанным способом можно закодировать при преобразовании перехода от двоичного представления данных к троичному. можно использовать для кодирования с исправлением ошибок (Forward Error correction, FEC). В отличие от кодирования с обнаружением ошибок, кодирование с исправлением ошибок помогает бороться с перегрузками в сети, но при этом требует больше избыточности в кодословах. В нашем случае этот недостаток нивелируется “бесплатными” битами и тритами, которые появляются вследствие преобразования $GF(2^s) \rightarrow GF(3^n)$.

В качестве кодирования с исправлением ошибок предлагается рассмотреть коды Рида–Соломона (RS), как наиболее эффективные

относительно затрачиваемой избыточной информации.

В общем случае схема кодирования выглядит следующим образом.

1. Выбирается значение t, определяющее порядок $q = 3^t$ поля Галуа характеристики 3.

2. Определяют примитивный многочлен $f(x)$ степени s с коэффициентами из элементов $GF(q) = \{0, 1, \ldots, q-1\}$. Многочлен $f(x)$ используется для полиномиального представления элементов расширенного поля.

3. Выбирается желаемая длина кодового слова n, $n < q^s$, и параметр r — число исправляемых ошибок (замен символов). Длина кодового слова измеряется в символах — элементах поля $GF(3^t)$, для представления которого можно использовать троичный вектор длины t.

4. К каждому передаваемому сообщению, состоящему из $n - 2r$ символов, приписывается $2r$ проверочных символов.

Если в процессе передачи, до r символов (включительно) кодослова искажено, то ошибочные символы будут вычислены, а также будут вычислены символы, которые должны были быть переданы изначально.

Адаптируем описанную выше схему для преобразования (1). Напомним, что изначально преобразование применялось к столбцам входной матрицы S размера $s \times \ell$, а на выходе получалась матрица T размера $n \times \ell$. Согласно описанной в конце предыдущего параграфа процедуре, можно (не теряя сильно в скорости освобождения буфера коммутатора), добавить j элементов к каждому выходному столбцу в преобразовании $GF(2^s) \to GF^n(3^t)$, что эквивалентно добавлению j строк в выходную матрицу. В итоге получено преобразование вида $GF(2^s) \to GF^{n+j}(3^t)$, вследствие которого, для выходной матрицы $n \times \ell$ получено $j \times \ell$ дополнительных тритов, которые как раз и будут использованы как $2 \times r$ избыточных символов.

Рассмотрим пример. Пусть на вход подана бинарная матрица S размера 48×54. Такая матрица соответствует 324 байтам данных. Изначально она, согласно преобразованию $GF(2^48) \to GF(3^{31})$, преобразуется в троичную матрицу T размера 31×54.

Начнем построение схемы кодирования по шагам, описанным ранее:

1. Положим $n = 6$, и тогда размер поля будет $3^6 = 729$.

2. В качестве примитивного порождающего возьмем многочлен $f(x) = x^6 + x + 2$ с коэффициентами из $GF(3)$, строим поле $GF(3^6)$.

3. Примем значение параметра $r = 18$ (при организации космической связи часто встречаются коды, которые исправляют 16 ошибок на 255 байт [9]). Длина основного сообщения равна $s - 2t = (54 \times 31)/6 = 279$ символов (символ — элемент поля $GF(3^6)$, который представим в виде 6 тритов). Количество проверочных символов равно $2r = 36$ символов, или $36 \cdot 6 = 216$ трит. Множество проверочных тритов поместится в $216/54 = 4$ строки выходной матрицы. В результате получаем суммарную длину кодового слова $s = 279 + 36 = 315$ символов или 1890 трит, которые представимы в виде матрицы размера 35×54.

4. Этот шаг выглядит следующим образом. Каждый раз, к матрице размера 31×54 прибавляется новые 4 строки, с 36 проверочными символами. Добавление этих четырех строк уменьшит скорость освобождения буфера на 13.9% относительно матрицы с 31 строками, что не так существенно, по сравнению с общим выигрышем в скорости в 27% относительно входной матрицы с 48 строками. Затем вся матрица, в виде 35 кадров отправляется в канал.

Сделаем некоторые замечания, касательно представленной схемы кодирования. Во-первых, по сравнению с "чистым" преобразованием, то есть без добавления проверочных строк, скорость освобождения буфера уменьшилась на $48/31 - 48/35 \approx 0.18$.

Во-вторых, сравним традиционное кодирование Рида–Соломона (16 исправляемых ошибок на 255 байт) и представленное кодирование (18 исправляемых ошибок на 315 символов).

8. Помехоустойчивые коды над $GF(3^t)$

Рассмотрим теперь второй подход к обеспечению помехозащищённости, заключающийся в передаче по каналам связи кодовых слов некоторого избыточного кода над некоторым полем характеристики 3.

Тернарные коды чётности описаны в [5,7]. Контрольный трит у них равен инверсии суммы проверяемых тритов (в сбалансированной системе), и результат поразрядного сложения всех тритов кодового слова дает 0. При одиночной ошибке номер

ошибочного трита определяется модулем синдрома, а тип ошибки — его знаком. Код Хэмминга $[13, 10, 3]_3$ в сбалансированной системе явно выписан в [7].

Рассматриваемый подход состоит в построении инъективного отображения

$$\overline{F} : GF^s(2) \to GF^k(3^t) < GF^n(3^t) \tag{2}$$

(здесь знак $<$ означает, что $GF^k(3^t)$ — подпространство пространства $GF^n(3^t)$). Инъективность будет обеспечиваться при

$$s \le \mu k t < \mu n t. \tag{3}$$

Рассмотрим параметры возможных помехоустойчивых кодов $C = [n, k, d]_{3^t}$ применительно к нашей задаче построения отображения (2). Такой код исправляет до $r = \lfloor (d-1)/2 \rfloor$ ошибок передачи троичных данных по каналу связи.

Рассмотрим сначала коды над простым полем Галуа характеристики 3 ($t = 1$).

1. *Троичный код Голея* $[11, 6, 5]_3$ с длиной кодового слова $n = 11$ трит, из которых $k = 6$ информационных исправляет 2 ошибки. Для такого кода имеем $Q = 3^6 = 729$, $s = \lfloor \log_2 729 \rfloor = 9$. То есть вместо 9 бит сообщения передается кодовое слово длиной 11 трит, из которых 2 могут быть исправлены, если в процессе передачи произошли ошибки. Избыточность кода $L = 3^6/2^9 = 1.42$.

Вывод: канал, допускающий ошибки в 2-х тритах из 11 — очень низкого качества, избыточность кода высока, и поэтому кодирование, ориентированное на такой канал неэффективно с обеих точек зрения.

2. *Троичные коды Хэмминга* $[n = (3^m - 1)/2, k = n - m, 3]_3$. У таких кодов n трит кодируют максимум $s = \lfloor \mu n \rfloor$ информационных бит с возможностью исправления одного трита.

Для нахождения конкретного кода будем последовательно выбирать значение m избыточных разрядов.

$m = 1$ — тривиальный код.

$m = 2$, код $[4, 2, 3]_3$. Вместо $s = \lfloor \mu \cdot 2 \rfloor = 3$ бит передается 4 трита, из которых 2 информационных. Канал, в котором возможна одна ошибка на 4 (3-ичных) символа представляется очень плохим, поэтому этот вариант кодирования вряд ли будет востребованным. Избыточность кода $L = 3^2/2^3 = 1.125$ не мала.

$m = 3$, код $[13, 10, 3]_3$. Вместо $s = \lfloor \mu \cdot 10 \rfloor = 15$ бит передается 13 трит, из которых 10 информационных. Представляется, что канал допускающий одну ошибку на 13 трит не эффективен, к тому же он имеет значительную избыточность $L = 3^{10}/2^{15} = 1.8$.

$m = 4$, код $[\,40, 36, 3\,]_3$. Вместо $s = \lfloor \mu \cdot 36 \rfloor = 57$ бит передается 40 трит, из которых 36 информационных. Крайне малая избыточность $L = 3^{36}/2^{57} \approx 1.04$ говорит о практически однозначном отображении. Представляется, что канал, допускающий одну замену в 40 тритах встречается достаточно часто. Кодирование 57 бит 36-ю тритами назовём *почти оптимальным*.

$m = 5$, код $[\,121, 116, 3\,]_3$. Вместо $s = \lfloor \mu \cdot 116 \rfloor = 183$ бит передается 121 трит, из которых 116 инфомационых. $L = 3^{116}/2^{183} \approx 1.81$. Такой код кажется неплохим, но избыточность его $L = 3^{116}/2^{183} \approx 1.81$ достаточно высока.

$m > 5$. Так как зависимость длины кодового слова от m — экспоненциальная, то рассматривать большие значения m не имеет смысла: канал, который допускает одну ошибку в сообщении длины, скажем, 567 бит ($m = 6$) не позволит получить выигрыш от перехода в троичную систему.

Рассмотрим далее коды над расширенными полями Галуа характеристики 3 ($t > 1$).

3. *Коды Хэмминга* в общем случае имеют вид $\left[n = \frac{q^m-1}{q-1}, k = n - m, 3\right]_q$, $q = 3^t$. Здесь передаваемым символом является не трит, а элемент поля Галуа $GF(3^t)$, представляемый на практике блоком из t тритов.

Рассматриваем случаи при $t > 1$, перебирая значения $m = n - k$ числа избыточных символов.

3.1. При всех t и $m = 1$ имеем тривиальный код.

3.2. $t = 2 \Rightarrow q = 3^2 = 9$.

$m = 2$ — код $[\,10, 8, 3\,]_9$ с параметрами $R = 8/10 = 0.8$ и $L = 3^{2\cdot 8}/2^{25} \approx 1.282$.

$m = 3$ — код $[\,91, 88, 3\,]_9$, $R = 725/728 \approx 0.996$, $L = 3^{2\cdot 88}/2^{278} \approx 1.194$.

$m = 4$ — код $[\,820, 826, 3\,]_9$, $R = 820/826 \approx 0.993$, $L = 3^{2\cdot 826}/2^{2618} \approx 1.29$.

Представляется, что случаи $m > 4$ не интересны практически.

3.3. $t = 3 \Rightarrow q = 3^3 = 27$.

$m = 2$ — код $[\,28, 26, 3\,]_{27}$ с параметрами $R = 26/28 \approx 0.929$, $L = 3^{3\cdot 26}/2^{123} \approx 1.544$.

$m = 3$ — код $[\,757, 754, 3\,]_{27}$ с параметрами $R = 754/757 \approx 0.996$, $L = 3^{3\cdot 754}/2^{123} \approx 1.137$.

$m \geq 4$ — коды, имеющие длины более 20 тыс. символов в нашей задаче нереальны.

3.4. $t = 4 \Rightarrow q = 3^4 = 81$.

$m = 2$ — код $[\,82, 80, 3\,]_{81}$ с параметрами $R = \approx 0.976$, $L \approx 1.139$.

$m = 3$ — код $[\,6643, 6640, 3\,]_{81}$ и коды с бóльшими значениями m вряд ли представляют интерес.

4. *Коды Рида–Соломона (примитивные)* имеют вид $[\,n = q - 1, k, n - k + 1\,]_q$, $q = 3^t$, $k < n$. Известно, что коды RS являются достигают границы Синглтона, то есть для невозможно построить код, который исправлял бы столько же ошибок, но имел параметр k выше.

Рассмотрим коды Рида–Соломона, использующие почти оптимальное представление 57-и бит 36-ю тритами.

Имеем $3^{36} = (3^\tau)^{36/\tau}$, где $\tau \in \{\,1, 2, 3, 4, 6, 12, 18, 36\,\}$. Тогда $36/\tau = k$ в соответствии с (2).

Будем строить RS-коды вида $[\,n = 3^\tau - 1, k = 36/\tau, d = n - k + 1\,]_{3^\tau}$, где τ пробегает значения всех делителей 36.

При $\tau = 1, 2$ имеем $k > n$ и соответствующих RS-кодов не существует.

$\tau = 3 \Rightarrow k = 12 < n = 26 = 3^3 - 1$ — существует RS-код $[\,26, 12, 15\,]_{3^3}$ скорости $R = 12/26 \approx 0.857$, исправляющий 7 (27-ричных) символов. Представляется, что такой код вполне приемлем.

$\tau = 4 \Rightarrow k = 9 < n = 80 = 3^4 - 1$ — существует RS-код $[\,80, 9, 72\,]_{3^4}$, исправляющий 35 (81-ричных) символов. Скорость кода $R = 9/80 \approx 0.11$ крайне невелика.

$\tau = 6, 12, \ldots$ — параметры этих кодов становятся практически неприемлемыми из-за мизерной скорости.

Заключение

В работе были рассмотрены теоретическае аспекты нового подхода к проблеме сжатия данных без потерь. Рассмотренный подход предолагается использовать для технологии сетевого кодирования, а именно, кодирования передаваемых кадров в промежуточных узлах сети с помощью нового более компактного представления. Рассмотренный подход имеет ряд достоинств, которые стоит отдельно отметить.

Во-первых, за счет перехода в новый алфавит, уменьшается число кадров, циркулирующих внутри сети при сетевом кодировании.

Во-вторых, уменьшаются затраты на повторную отправку кадров, что является обязательным при линейном сетевом кодировании, так как появляется возможность эффективно использовать механизмы исправления ошибок. В-третьих, устраняется необходимость дублирования кадров, а также

решается проблема асинхронной доставки кадров в конечные узлы сети. С другой стороны, в таком подходе имеется ряд недостатков.

Сеть все также уязвима перед скомпрометированными коммутаторами; к тому же потеряно свойство защищенности линейного сетевого кодирования от атак типа Wiretapping [8]

В добавок к этому, на устройства внутри сети будет налагаться дополнительная вычислительная нагрузка. Также недостатком остается сложность внедрения такого подхода в уже существующую инфраструктуру.

В работе предложено два алгоритма, первый из которых позволяет использовать всю избыточность, которая возникает при использовании предложенного метода сетевого кодирования. Второй алгоритм – демонстрация сочетаемости этого метода с технологией Forward Error correction. Также в работе приведены основные параметры для некоторых видов кодов Хэмминга и Рида–Соломона и выделены наилучшие.

Предложенный метод оставляет несколько открытых проблем:

- Значения s и n выбираются заранее и не учитывают текущего состояния коммутатора. Коммутатору надо будет ждать пока в буфере не накопится n кадров, что не есть хорошо. Требуется исследовать методы, в которых количество пакетов может быть переменным.

- Вопрос о группировке столбцов для максимизации вероятности отправки дополнительного сообщения. Требуется найти оптимальные отношения между размером дополнительно кодируемого сообщения и количеством группируемых столбцов.

- Следует провести анализ применимости рассмотренных кодов в зависимости от параметров каналов связи, рассматривая также операции сокращения и выкалывания кодов.

Литература

1. C. Fragouli, E. Soljanin. Network coding fundamentals, Now Publishers Inc, 2007.

2. R. Ahlswede et al. “Network information flow,” in IEEE Transactions on information theory, vol. 46, N 4, 2000, pp. 1204–1216.

3. С. В. Фомин. Системы счисления. М.: Наука, 1987.

4. T. Kerins, E. Popovici, and W. P. Marnane. Fully paramaterisable Galois Field arithmetic processor over $GF(3^m)$ suitable for elliptic curve cryptography, Proceedings of the International Conference on Microelectronics, 24, vol. 2, 10.1109/ICMEL.2004.1314938, pp. 739–742.

5. D. Efanov. Ternary parity codes: Features. 1-5. 10.1109/EWDTS.2019.8884414, 2019, pp. 315-319.

6. Р. Л. Смелянский. Компьютерные сети: в 2 т., Т. 1, Системы передачи данных. — М.: Издательский центр Академия, 2011.

7. Ю. С. Владимирова, Х. Рамиль Альварес. “Huffman compression and self-correcting Hamming codes on ternary machines,” unpublished.

8. O. Al-Khaleel et al. “High performance FPGA-based decimal-to-binary conversion schemes for decimal arithmetic,” in Microprocessors and Microsystems, vol. 37.3, 2013, pp. 287–298.

9. S. Leilei et al. “Integration of Reed–Solomon codes to licklider transmission protocol (LTP) for space DTN,” in IEEE Aerospace and Electronic Systems Magazine, vol. 32.4, 2017, pp. 48–55.

Казантаев А. Д., Чупахин А. А.

Алгоритм балансировки нагрузки в распределенной вычислительной системе с применением мультиагентного обучения с подкреплением

Введение

По мере роста популярности мобильных устройств в наступающую эпоху 5G, мобильные приложения, особенно для задач, требующих больших вычислительных ресурсов, таких как онлайн-игры, распознавание лиц и дополненная или виртуальная реальность на основе определения местоположения, столкнулись с ограниченными вычислительными возможностями устройств [1]. Для сохранения качества восприятия услуг (*QoE — Quality of Experience*) была предложена технология периферийных вычислений с множественным доступом (ПВМД) [2] в качестве решения проблемы роста спроса на вычисления, запрашиваемые приложениями, при ограниченных ресурсах мобильных устройств.

В последнее время глубокое обучение с подкреплением стало эффективным методом решения задач обучения с подкреплением с большими пространствами состояний и действий [3], которые возникают при решении задач распределения заданий и ресурсов. Существуют работы по распределению заданий на основе глубокого обучения с подкреплением для сетей ПВМД, однако эти подходы часто не применимы к задачам, возникающим на практике, из-за использования централизованных алгоритмов для принятия всех решений.

Как правило, балансировка заданий выполняется в системах вычислителей, где задачи должны быть распределены между несколькими вычислителями. Мультиагентное обучение с подкреплением — *Multi-Agent Reinforcement Learning (MARL)* — обучает множество агентов взаимодействовать в общей среде: сотрудничать или конкурировать. Эта концепция контрастирует с одноагентным обучением с подкреплением, где состояние окружающей среды меняется исключительно в результате действий отдельных агентов.

В данной работе предлагается алгоритм мультиагентного обучения с подкреплением, способный решить задачу балансировки

нагрузки в неоднородной РВС.

1. Постановка задачи

Дана неоднородная распределенная вычислительная система, представленная в виде полного графа $G = (V, E)$:

- V — множество вычислительных узлов (ВУ) РВС;
- $|V| = N$ — количество ВУ РВС;
- $v_i \in V$, $i = 1, \ldots, N$ — i-й ВУ;
- $\tilde{V}$, $\tilde{V} \in V$ — множество периферийных ВУ;
- E — множество двунаправленных каналов передачи данных между каждой парой ВУ. Так как граф G является полным, количество таких каналов: $|E| = \frac{N*(N-1)}{2}$;
- $\{v_i, v_j\} \in E$, $v_i, v_j \in V$, $i \neq j$, $i = 1, \ldots, N$, $j = 1, \ldots, N$ — двунаправленный канал передачи данных между ВУ v_i и ВУ v_j;
- c_i, $i = 1, \ldots, N$ — вычислительный ресурс (ресурс центрального процессора) ВУ v_i;
- X — множество вычислительных заданий, поступающих в РВС. Каждое задание имеет уникальный идентификатор в виде натурального числа;
- $x_j \in X$ — вычислительное задание с идентификатором $j \in \mathbb{N}$;
- $q_i = (q_1, q_2, \ldots, q_m)$, $m \in \mathbb{N}$ — очередь с заданиями узла v_i. Задания в q_i упорядочены по времени поступления на ВУ. Дисциплина обработки очереди — *FIFO*;
- $F(\cdot) \in \{F_p, F_e, F_u\}$ — функция распределения, которая определяет вероятность того, что за заданный промежуток времени, в систему поступит хотя бы одно задание. F_p — распределение Пуассона, F_e — распределение Эрланга, F_u — равномерное распределение;
- $W = (w_{i,j})$ — матрица времени передачи заданий между узлами. $w_{i,j}$ — время, затрачиваемое на передачу задания между узлами v_i и v_j. Время передачи задания не зависит от его размера. Так как между ВУ v_i и ВУ v_j каналы передачи

данных двунаправленные, то $w_{i,j} == w_{j,i}$, $i = 1, \ldots, N$, $j = 1, \ldots, N$

На примере распределения Пуассона поясним, как определяется вероятность поступления задания в РВС. Пусть λ — интенсивность поступления заданий в промежуток времени Δt, тогда вероятность поступления в РВС m заданий в Δt определяется следующим образом согласно функции распределения Пуассона: $P_m = \frac{\lambda^m}{m!} e^{-\lambda}$.

Неоднородная РВС работает со структурой временных шагов, шаг считается дискретным и индексируется натуральным числом $T = \{1, 2, \ldots, t\}, t \in \mathbb{N}$. Количество заданий, поступивших в РВС заранее неизвестно, известно только множество различных заданий X, которые могут поступать в систему. Каждое задание $x_j \in X$ равновероятно может поступить в систему, время поступления определется функцией распределения $F(\cdot) \in \{F_p, F_e, F_u\}$.

Задание $x_j \in X$ описывается следующими характеристиками:

- $t_j \in T$ — время поступления задания x_j в систему (это характеристика задания, заранее о времени поступления в РВС планировщику неизвестно);
- $lt_j \in T$ — дедлайн выполнения задания x_j, момент времени в будущем относительно начала работы РВС;
- $rt_j = \{rt_{j,1}, \ldots, rt_{j,N}\}$ — вектор времени выполнения задания x_j на каждом ВУ. $rt_{j,i}$ — время выполнения задания x_j на ВУ v_i;
- $cr_j = \{cr_{j,1}, \ldots, cr_{j,N}\}$ — вектор используемого заданием x_j вычислительного ресурса на каждом ВУ. $cr_{j,i}$ — требуемое количество ресурса для выполнения задания x_j на ВУ v_i.

Текущее состояние $St_i(t) = (c_i(t), q_i(t))$ ВУ v_i в момент времени $t \in T$ описывается следующими характеристиками:

- $c_i(t)$ — вычислительные ресурсы ВУ v_i в момент времени $t \in T$. На ВУ v_i одновременно может выполняться несколько заданий, пока достаточно свободных ресурсов ВУ. Таким образом, $c_i(t) = \sum_{x_j \in X'} cr_{j,i}$, где X' — множество заданий, выполняющихся одновременно на ВУ v_i;
- $q_i(t)$ — длина очереди ВУ v_i в момент времени $t \in T$. Пусть X'' — это все задания в очереди в момент времени $t \in T$, тогда $q_i(t)$ — это время, необходимое для полного завершения всех работ $x_j \in X''$.

ВУ пересылают информацию о $c_i(t)$ и $q_i(t)$ другим устройствам — время на пересылку таких сообщений не учитывается.

Между заданиями отсутствует зависимость по данным, то есть нет заранее заданного частичного порядка их выполнения. Если задание попало в очередь или начало выполнение на одном из ВУ, оно не может быть прервано и отправлено на другое. ВУ может оставить задание в своей очереди либо переслать свободному ВУ. Решение "оставлять" или "пересылать" принимается в момент поступления задания на ВУ. Сначала проверяется, не наступил ли дедлайн выполнения задания: если наступил, задание "сбрасывается", если дедлайн не наступил, то ВУ может оставить или переслать задание другому ВУ в зависимости от локального алгоритма принятия решения. Задание "сбрасывается", если ВУ решило оставить его в своей очереди, но дедлайн его выполнения наступит раньше, чем задание успеет начать и закончить выполняться на ВУ. Таким образом, задание x_j может пересылаться между ВУ в составе РВС до тех пор, пока не наступит дедлайн ее выполнения (в этом случае задача будет считаться невыполненной), либо задача останется в очереди некоторого ВУ v_i и выполнится до дедлайна.

Вычислительным путем задания $CP(x_j)$, $x_j \in X$ будем называть последовательность ВУ, на которых оно побывало после попадания в РВС: $CP(x_j) = [v_{k_1}, v_{k_2}, ..., v_{k_{l_j}}]$. Тогда по вычислительному пути $CP(x_j)$ можно вычислить время пребывания в РВС задания x_j: $Trans_{CP}(x_j) + Queue_{CP}(x_j) + Comp_{CP}(x_j)$, где:

- $Trans_{CP}(x_j) = \sum_{i=1}^{l_j-1} w_{k_i,k_{i+1}}$ — время, потраченное на пересылку x_j между ВУ $v_{k_1}, v_{k_2}, ..., v_{k_{l_j}}$;
- $Queue_{CP}(x_j) = q_{k_{l_j}}(l_j)$ — время нахождения задания x_j в очереди ВУ $v_{k_{l_j}}$, которое приняло решение оставить задание;
- $Comp_{CP}(x_j) = rt_{j,k_{l_j}}$ — время выполнения задания x_j на ВУ $v_{k_{l_j}}$.

Слагаемые $Queue_{CP}(x_j)$ и $Comp_{CP}(x_j)$ можно вычислить только для тех заданий, у которых не наступил дедлайн к моменту времени $t_j + Trans_{CP}(x_j)$, а также для тех заданий, которые успеют выполниться на узле $v_{k_{l_j}}$ до дедлайна, т.е. $(t_j + Trans_{CP}(x_j) + Queue_{CP}(x_j) + Comp_{CP}(x_j)) \leq lt_j$. Таким образом, для просроченных заданий верно либо $t_j + Trans_{CP}(x_j) \geq lt_j$, либо

$(t_j + Trans_{CP}(x_j) + Queue_{CP}(x_j) + Comp_{CP}(x_j)) > lt_j$. Обозначим количество выполненных в срок заданий к моменту времени $t \in T$ через $FT(t) = |\{x_j | (t_j + Trans_{CP}(x_j) + Queue_{CP}(x_j) + Comp_{CP}(x_j)) \leq lt_j\}|$. Обозначим количество просроченных заданий к моменту времени $t \in T$ через $ET(t)$.

Целевой функцией, используемой для оценки эффективности распределения заданий в РВС к моменту времени $t \in T$, в контексте рассматриваемой задачи выступает:

$$\mathfrak{F}(t) = \frac{ET(t)}{M} + \sum_{i=1}^{N} \left(\alpha \frac{\bar{c}_i}{c_i} + \beta \frac{\bar{q}_i}{q_i} + \gamma \left(\left(\frac{\bar{c}_i}{c_i} - \Theta \right)^2 + \left(\frac{\bar{q}_i}{q_i} - \Delta \right)^2 \right) \right), \quad (1)$$

где:

1. α, β, γ — $const \in [0,1]$, $\alpha + \beta + \gamma = 1$;
2. M — количество заданий, поступивших в РВС к моменту времени $t \in T$;
3. $\frac{ET(t)}{M} \in [0,1]$ — доля просроченных заданий относительно всех заданий, поступивших в РВС к моменту времени $t \in T$;
4. $\bar{c}_i \in [0, c_i], \bar{q}_i \in [0, q_i]$ — усредненное за всё время от 1 до t количество используемых вычислительных ресурсов и усредненная длина очереди ВУ v_i: $\bar{c}_i = \frac{\sum_{\tau=1}^{t} c_i(\tau)}{t}$, $\bar{q}_i = \frac{\sum_{\tau=1}^{t} q_i(\tau)}{t}$;
5. $\frac{\bar{c}_i}{c_i} \in [0,1]$ — усредненная по времени доля используемого вычислительного ресурса ВУ v_i;
6. $\frac{\bar{q}_i}{q_i} \in [0,1]$ — усредненная по времени доля заполненности очереди на ВУ v_i;
7. $\Theta, \Delta \in [0,1]$ — усредненные доли используемых ресурсов и усредненные доли заполненности очередей на всех ВУ РВС за всё время от 1 до t: $\Theta = \frac{\sum_{i=1}^{N} \frac{\bar{c}_i}{c_i}}{N}$, $\Delta = \frac{\sum_{i=1}^{N} \frac{\bar{q}_i}{q_i}}{N}$.
8. $\left(\frac{\bar{c}_i}{c_i} - \Theta \right)^2$ — квадрат отклонения усредненной по времени доли используемого вычислительного ресурса ВУ v_i относительно усредненных долей используемых ресурсов всех ВУ РВС за все время от 1 до t;

9. $\left(\frac{\bar{q}_i}{q_i} - \Delta\right)^2$ — квадрат отклонения усредненной по времени доли заполненности очереди на ВУ v_i относительно усредненных долей заполненности очередей на всех ВУ РВС за всё время от 1 до t;

В данной работе рассматриваются два критерия эффективности работы РВС: доля просроченных заданий относительно всех заданий, поступивших в систему за время $t \in T$(первое слагаемое в (1)), и сбалансированное использование ресурсов РВС (второе слагаемое в виде суммы в (1)). Сбалансированность используемых ресурсов (вычислительных ресурсов и очереди ВУ v_i) учитывается благодаря включению в целевую функцию слагаемых 5, 6 (усредненные по времени доли используемых ресурсов) и 8, 9 (отклонения долей используемых ресурсов каждого ВУ относительно средней доли используемых ресурсов для всех ВУ). Таким образом, многокритериальная задача оптимизации сведена к однокритериальной.

Требуется найти алгоритм распределения заданий, минимизирующий целевую функцию (1).

2. Стохастическая игра

Представим стохастическую игру для моделирования распределения заданий при балансировке нагрузки в неоднородной распределенной вычислительной системе с минимизацией системных затрат. В данной неоднородной РВС каждое ВУ рассматривается как агент, изучающий свою оптимальную политику построения расписания путем взаимодействия со средой. Политика — стратегия, которая применяется агентом для принятия решения о следующем действии на основе текущего состояния. Политика позволяет оценить вероятность того, что определенные действия приведут к вознаграждениям или положительным состояниям. Это игра с N агентами, в которой каждый агент изучает свою политику действий. Каждый агент наблюдает за состоянием окружающей среды $s_i(t) \in S_i$, после чего выбирает одно действие $a_i(t) \in A_i$: отправка на вычислительный узел $v_i \in V$. Каждый агент получает вознаграждение $r_i(t) = r_i\left(s_i(t), a_1(t), \cdots, a_N(t)\right)$, целевую функцию (1), и переходит в следующее состояние $s_i(t+1) \in S_i$, в соответствии с действиями всех участвующих в игре агентов.

Стохастическая игра — это обобщение марковского процесса принятия решений в мультиагентном случае, также называемое

марковской игрой, которая обозначается кортежем $< \mathcal{S}, N, \mathcal{A}, \mathbb{P}, \mathcal{R} >$:

- $\mathcal{S}$ — состояние среды, включающее в себя состояние каждого агента, $\mathcal{S} = \mathcal{S}_1, \mathcal{S}_2, \cdots, \mathcal{S}_N$;
- N — число агентов;
- $\mathcal{A}$ — набор совместных действий, $\mathcal{A} = \mathcal{A}_1, \mathcal{A}_2, \cdots, \mathcal{A}_N$;
- $\mathbb{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \in [0, 1]$ — функция вероятности перехода между состояниями;
- $\mathcal{R} = \{\mathcal{R}_1, \cdots, \mathcal{R}_N\}$ — функции наград всех агентов.

3. Формулировка стохастической игры балансировки нагрузки

Основываясь на определении стохастической игры, сформулируем рассматриваемую задачу балансировки нагрузки в неоднородной РВС как стохастическую игру, определяя каждый элемент в кортеже $< \mathcal{S}, N, \mathcal{A}, \mathbb{P}, \mathcal{R} >$ (см. раздел 2).

- *Пространство действий* $\mathcal{A}$. В предложенной неоднородной РВС каждое вычислительное устройство v_i представляется агентом, который в любой момент времени $t \in T$ выбирает действие $a_i(t) \in A_i$. Действие $a_i(t)$ представляет собой номер ВУ $i = 1 \cdots N$, на который нужно отправить задание. Если действие совпадает с номером самого ВУ, то задание добавляется в его очередь. Таким образом, пространство действий стохастической игры балансировки нагрузки представляется как $A = \prod_{i \in N} A_i, A_i \in \mathbb{N}$.
- *Пространство состояний* $\mathcal{S}$. В момент времени $t \in T$ для каждого отдельного ВУ состояние задается следующим образом:

$$s_i(t) = (NT_i(t), TR_i(t), TRT_i(t), MTL_i(t)), \tag{2}$$

где:

  - $NT_i(t) \in [0, 1]$ — количество заданий в очереди ВУ v_i в момент времени $t \in T$;
  - $TR_i(t) \in [0, 1]$ — отношение длины очереди ВУ v_i в момент времени $t \in T$ к максимальной длине очереди ВУ v_i;

– $TRT_i(t) \in [0,1]$ — время выполнения всех заданий в очереди ВУ v_i в момент времени $t \in T$;

– $MTL_i(t) \in [0,1]$ — время выполнения самого короткого задания в очереди ВУ v_i в момент времени $t \in T$.

Все элементы состояний представляются вещественными числами и нормализованы по максимальному соответствующему элементу всей системы.

- *Награда* $\mathcal{R}$. Вознаграждение $r_i\left(s_i(t), a_i(t), a_{-i}(t)\right)$ вычислительного устройства v_i в состоянии $s_i(t)$ представляет собой прибыль v_i при выполнении действия $a_i(t)$, в то время как другие ВУ выполняют действия $a_{-i}(t) = (a_1(t), \cdots, a_{i-1}(t), a_{i+1}(t), \cdots, a_N(t))$ в момент времени $t \in T$. Так как традиционно в стохастических играх награда маскимизируется, в качестве функции награды рассматривается немного измененная целевая функция, введенная в разделе 1. В качестве первого слагаемого рассматривается не доля просроченных заданий, а доля выполненных в срок заданий, а второе слагаемое рассматривается с обратным знаком. Таким образом, функция награды выглядит следующим образом:

$$\mathfrak{R}(t) = \frac{FT(t)}{M} - \sum_{i=1}^{N}\left(\alpha\frac{\bar{c}_i}{c_i} + \beta\frac{\bar{q}_i}{q_i} + \gamma\left(\left(\frac{\bar{c}_i}{c_i} - \Theta\right)^2 + \left(\frac{\bar{q}_i}{q_i} - \Delta\right)^2\right)\right), \quad (3)$$

Это нужно, чтобы алгоритм обучения с подкреплением максимизировал функцию награды, что приводит к минимизации целевой функции. Максимизация функции награды заключается в максимизации первого слагаемого и минимизации второго слагаемого в 3. Это возможно делать независимо, если назначать положительные награды для действий, максимизирующих первое слагаемое и действией, минимизирующих второе слагаемое, и отрицательные награды для остальных действий.

Политика $\pi_i\left(s_i, a_i\right)$ представляет собой отображение из каждого состояния $s_i \in S_i$ и $a_i \in A_i$ с вероятностью $\pi_i\left(s_i, a_i\right) = \mathbb{P}\left(a_t = a \mid s_t = s\right) \in [0,1]$. ВУ v_i в состоянии s_i имеет смешанную политику $\pi_i\left(s_i\right) = \{\pi_i\left(s_i, a_i\right) \mid a_i \in A_i\}$. Следовательно, в стохастической игре совместная политика для N агентов определяется вектором политики $\pi = \left(\pi_1\left(s_1\right), \pi_2\left(s_2\right), \cdots, \pi_N\left(s_N\right)\right)$, где каждая отдельная политика принадлежит каждому отдельному

ВУ. Основываясь на вероятностных политиках, мы можем определить функцию ценности состояния s_i при стратегии π для ВУ v_i:

$$V_i\left(s_i, \pi_i\right) = E\left[\sum_{\tau=0}^{+\infty} \gamma^\tau r_i\left(s_i(t+\tau), \pi_i\left(s_i(t+\tau)\right)\right) \mid s_i(t) = s_i, \pi_i\right], \quad (4)$$

где $E[\cdot]$ — математическое ожидание перехода в состояние s_i при политике π_i, а $\gamma \in [0,1]$ — коэффициент дисконтирования (определяет разницу в важности между текущим и будущими вознаграждениями, например, если $\gamma = 0.9$ и есть вознаграждение в размере 5 после 3 шагов, настоящая стоимость этого вознаграждения $5 * 0.9^3$).

Переход из состояния $s_i(t)$ в новое состояние $s_i(t+1)$ определяется совместной политикой всех ВУ. В момент времени $t \in T$ каждое вычислительное устройство v_i выбирает политику $\pi_i\left(s_i\right)(t)$ в состоянии $s_i(t)$, чтобы максимизировать свое дисконтированное вознаграждение $V_i\left(s_i, \pi_i\right)$, а затем получает текущее индивидуальное вознаграждение, основанное на совместной политике π. Целью каждого ВУ является изучение оптимальной политики π_i^*, а оптимальные политики других агентов изучаются как $\pi_{-i}^* = \left(\pi_1^*, \cdots, \pi_{i-1}^*, \pi_{i+1}^*, \cdots, \pi_N^*\right)$. Переформулируем (4): $V_i\left(s_i, \pi_i, \pi_{-i}\right) = E\left[\sum_{\tau=0}^{+\infty} \gamma^\tau r_i\left(s_i(t+\tau), \pi_i\left(s_i(t+\tau)\right), \pi_{-i}\left(s_i(t+\tau)\right)\right) \mid s_i(t) = s_i\right]$, где

$$\pi_{-i}\left(s_i(t)\right) = \left(\pi_1\left(s_1(t)\right), \ldots, \pi_{i-1}\left(s_{i-1}(t)\right), \pi_{i+1}\left(s_{i+1}(t)\right), .., \pi_N\left(s_N(t)\right)\right)$$

.

4. Обзор существующих решений

Задача балансировки нагрузки в неоднородной вычислительной системе является NP-трудной задачей [4]. NP-трудные задачи подразумевают использование эвристических алгоритмов для их решения, а конкретно для данной задачи были использованы муравьиные алгоритмы, генетические алгоритмы, алгоритм имитации отжига [5]. В последние годы стала популярна парадигма машинного обучения “обучение с подкреплением” [6].

Обзор показал, что для решения задачи балансировки нагрузки более целесообразно представить задачу как мультиагентный марковский процесс принятия решений (ММППР) ([9]-[12]), нежели

Статья	Мульти-агентность	Децентрали-зованность	Неоднородность	Равномерность задействования ресурсов	Минимизация времени выполнения работ	Алгоритм Обучения с Подкреплением
[7]	–	–	+	–	+	*DDPG* + *LSTM*
[8]	–	–	+	+	+	Q-Обучение
[9]	+	–	+	–	+	*MA-DDPG*
[10]	+	+	–	–	+	*MA-DDPG*
[11]	+	–	+	–	+	*MA-DDPG*
[12]	+	+	–	+	–	*MA-DDPG*

Таблица 1: Сравнение алгоритмов балансировки нагрузки.

обычный марковский процесс принятия решений ([7], [8]), так как это дает возможность создать децентрализованную систему обучения с подкреплением.

Большинство рассмотренных алгоритмов нацелено на решение задачи минимизации времени выполнения заданий, почти не решается задача равномерного распределения ресурсов и их влияния на количество завершенных в срок заданий (см. целевую функцию (1)).

Основным алгоритмом обучения с подкреплением, который используют исследователи в своих работах, является *DDPG* [14] и его мультиагентная версия, однако в данной работе за основу взят алгоритм *PPO* [13], являющийся улучшенной версией *DDPG* и алгоритма Исполнителя-Критика (Actor-Critic) [15].

5. Описание разработанного алгоритма

На Рис. 1 показана структура *MARL* для неоднородной вычислительной системы. В момент времени $k \in T$ каждое ВУ v_i в состоянии $s_i(k)$ выполняет действие $a_i(k) \in \pi_i(s_i)$. Таким образом, РВС переходит в новое совместное состояние $S(k+1) = \{s_1(k+1), \cdots, s_i(k+1), \cdots, s_N(k+1)\}$ после применения совместного действия $A(k) = \{a_1(k), \cdots, a_i(k), \cdots, a_N(k)\}$. Каждое ВУ получает локальную информацию о среде $o_i(k)$ и вознаграждение $r_i(k)$. Поскольку будущее состояние $S(k+1)$ зависит только от текущего состояния $S(k)$ и предпринятого действия $A(k)$, этот алгоритм *MARL* формулируется в виде стохастической игры, описанной в разделе 3.

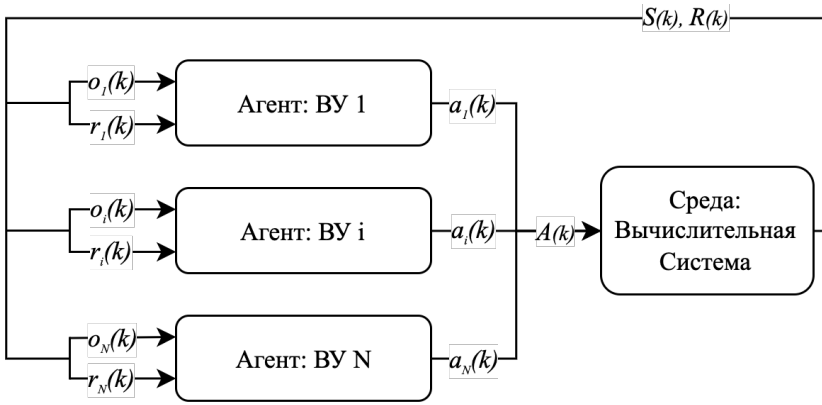

Рис. 1: Представление структуры MARL алгоритма задачи балансировки нагрузки в вычислительной системе.

Для реализации разработанного алгоритма было принято решение описать среду стохастической игры на языке *Python* с использованием библиотеки *OpenAI Gym*. Данная библиотека позволяет разрабатывать алгоритмы обучения с подкреплением и проводить их сравнение.

При инициализации класса среды *OpenAI Gym* требуется с помощью семантики *gym spaces* описать пространства *действий* и *состояний*. Так как состояния в стохастической игре балансировки нагрузки введены как множество вещественных чисел (см. раздел 3), используется класс *Box*, а для описаний действий используется класс *Discrete*. Кроме этого требуется описать методы *reset* и *step*.

Метод *reset* (см. Алгоритм 1) используется для возвращения состояния среды в начальное положение — это требуется для итерационного обучения агентов после прохождения заданного числа шагов итерации. В конце каждой итерации алгоритма предлагается сохранять число невыполненных заданий — это требуется для оценки эффективности работы алгоритма.

Метод *step* (см. Алгоритм 2) описывает алгоритм итерационного шага, возвращающий новые состояния и награды всех агентов, взаимодействующих со средой, принимающий на вход словарь действий каждого агента. В нём описывается основная логика алгоритма, включая пересылку заданий между ВУ, выполнение заданий, отслеживание очередей, вычисление полученных при действиях из *action_ dict* состояний и наград.

Функция RESET
    $time = 0$
    Если $drop\_count \neq None$ **тогда**
        Функция Сохранить количество невыполненных заданий
    Конец условия
    $drop\_count = 0$
    Функция Заново инициализировать информацию о РВС
    $state = []$
    Для всех i в num_agents **выполнять**
        $state_i = 0$
    Конец цикла
    Возвратить $state$
Конец функции

Алгоритм 1: Метод *reset()* среды *Gym*.

Функция STEP($action\_dict$)
    $state-$ пустой список длины num_agents
    $rew-$ пустой список длины num_agents
    Для всех i в num_agents **выполнять**
        $rew[i]$ = число завершившихся заданий на шаге $time$
        Если На данном шаге пришли задания на ВУ v_i **тогда**
            Функция Выбрать первое, оставшиеся отправить на случайные ВУ
            Если Директивный срок задания > длины очереди $action\_dict[i]$ **тогда**
                $rew[i] = rew[i] - 0.5$
            Конец условия
            Если $action\_dict[i] \neq i$ **тогда**
                Функция Отправить на ВУ $action\_dict[i]$
            Конец условия
        Конец условия
    Конец цикла
    Для всех i в num_agents **выполнять**
        Пока Есть свободные ресурсы процессора **выполнять**
            Функция Активировать задания в порядке *FIFO*
        Конец цикла
    Конец цикла
    Функция Заполнение $state$
    Функция Нормализация $state$
    $drop =$ число заданий с истёкшим директивным сроком
    Функция Удаление заданий с истёкшим директивным сроком
    $time = time + 1$
    Возвратить $state, rew$
Конец функции

Алгоритм 2: Метод *step(action_ dict)* среды *Gym*.

На каждом ВУ РВС располагается обучаемый агент. Num_agents — количество ВУ в РВС, N согласно математической постановке задаче (см. раздел 1). $Action\_dict$ — ассоциативный массив, ключами которого являются индексы ВУ (например, для v_i

индекс равен i), значения — индекс другого ВУ (если действие заключается в пересылке задания на другой ВУ), тот же самый индекс ВУ (если ВУ оставляет задание в своей очереди).

Реализованные алгоритмы публично доступны на GitHub [16].

6. Описание программной реализации

6.1 Требования к программному модулю

Приведем основные требования, которым должен удовлетворять программный модуль:

- Наличие подмодуля генерации исходных данных;
- Наличие консольного интерфейса запуска экспериментов;
- Наличие подмодуля автоматического запуска экспериментов и обработки результатов;
- Возможность визуализации результатов экспериментов;
- Кроссплатформенность, возможность работать на вычислительных системах под управлением различных операционных систем.

6.2 Подмодуль генерации исходных данных

Для экспериментального исследования алгоритма мультиагентного обучения с подкреплением для решения задачи балансировки нагрузки в неоднородной РВС был разработан генератор исходных данных на языке программирования *Python*, данных о вычислительных ресурсах, времени отправки сообщений между ВУ и длинах очереди агентов.

При генерации данных требуется задать число шагов в итерации, от которого зависит количество заданий, число всех агентов и получающих задания извне. Время прибытия заданий определяется с помощью распределений: равномерного, Пуассона и Эрланга.

Было сгенерировано 360 наборов данных, различающихся указанными параметрами и разделенными на соответствующие группы.

6.3 Подмодуль среды *Gym*

Класс среды стохастической игры 2 является наследником класса *MultiAgentEnv* библиотеки RLlib [17], который позволяет нескольким агентам работать в одной среде. Наследование даёт нам доступ к использованию методов *MultiAgentEnv* и возможность переопределять их в классе среды, как в случае с *reset* (Алгоритм 1) и *step* (Алгоритм 2).

При инициализации класса среда загружает данные об исходном расписании и описываемой вычислительной системе, сгенерированных модулем генерации данных 6.2. Исходное расписание представляет собой *pandas Dataframe* [18], то есть двумерный массив с именованными столбцами, который позволяет легко обращаться к данным, изменять их и проводить различные операции с помощью методов класса *Dataframe*, как в функции отправки заданий на другое вычислительное устройство.

6.4 Подмодуль визуализации результатов экспериментов

Для инициализации экземпляра класса визуализации требуется подать ему на вход название эксперимента и число шагов итерации в выбранном эксперименте. Первое нужно для доступа к директории с результатами, которые мы хотим графически изобразить, второе — для правильного отображения результатов по оси абсцисс.

Выделяются два метода класса: отрисовка вариации числа заданий с истекшим директивным сроком и вариации значений целевой функции. Оба метода проходят по всем сохраненным в директории результатам, запоминая минимальное и максимальное значения для каждой точки оси абсцисс, после чего с помощью библиотеки *matplotlib* [19] строят закрашенную область между полученными значениями.

6.5 Подмодуль запуска экспериментов

Данный модуль базируется на использовании *Ray Tune API*, средства запуска экспериментов с выбранными параметрами. Метод *tune.run(·)* принимает на вход словарь параметров эксперимента, включающий в себя как общую информацию о числе запусков, наименовании среды, выбранного алгоритма обучения с подкреплением, так и конфигурационные данные для алгоритма. В данной работе используется алгоритм обучения с подкреплением — *PPO*. Для того чтобы использовать данный алгоритм, РВС была

описана в терминах *OpenAI Gym* (см. раздел 5). Подразумевается, что данный алгоритм работает в составе целевой системы РВС постоянно. Периодически происходит фаза обучения и настройки алгоритма обучения с подкреплением на текущий профиль нагрузки и характеристики потока поступающих заданий. Более подробно про фазу обучения, скорость сходимости алгоритма описано в разделе 7.2.

7. Экспериментальное исследование

7.1 Цели экспериментального исследования

Сформулируем цели экспериментального исследования разработанного алгоритма:

1. Исследовать влияние числа периферийных агентов на сходимость числа просроченных заданий и целевой функции;

2. Исследовать влияние распределения заданий на сходимость числа просроченных заданий и целевой функции.

7.2 Методика экспериментального исследования

В различных экспериментах общее количество ВУ в графе РВС равняется либо 10, либо 15. Длина очередей на каждом ВУ одинаковая и равняется 20 (очередь может хранить не более 20 заданий одновременно). В экспериментах, когда общее количество ВУ равняется 10, количество периферийных ВУ может равняться 5, 7, или 9. В экспериментах, когда общее количество ВУ равняется 15, количество периферийных ВУ может равняться 5, 7, 9, 11, 13 или 15. В данной статье приведены результаты экспериментального исследования для двух распределений времен поступления заданий в РВС: распределения Пуассона и равномерного распределения. Входные данные генерировались в виде *Dataframe* [18], каждая строка которого соответствовала характеристикам одного задания (см. описание характеристик заданий в разделе 1). В среднем в одном комплекте входных данных было около 300 заданий.

Стоит отметить, что алгоритм обучения с подкреплением работает эпизодами. Каждый эпизод совершаются одни и теже действия: задания поступают в РВС, затем многоагентный алгоритм обучения с подкреплением распределяет задания по вычислительным устройствам. Каждый эпизод состоит из 500 итераций, каждой итерации соответствует один временной шаг,

введенный в математической постановке (см. раздел 1). Однако после каждого эпизода многоагентный алгоритм обучения с подкреплением должен работать все лучше и лучше, так как он каждый раз не обучается заново, а продолжает обучение с предыдущего эпизода. Такой принцип обучения и проверки качества алгоритмов обучения с подкреплением считается общепринятым, детально однакомиться с ним можно в работе [17].

Как определить, что алгоритм обучился и сошелся к *стационарному состоянию*? Рассмотрим временное окно, длительностью 100 итераций. Стационарным состоянием будем называть такое состояние, в котором последние 100 итераций не изменялись одно из следующих значений: значение целевой функции, количество выполненных в срок заданий или количество просроченных заданий.

Конфигурация РВС определяется следующими параметрами: общее количество ВУ, количество периферийных ВУ, функция распределения времен поступления приходящих заданий. Одна конфигурация РВС отличается от другой, если хотя бы один из перечисленных параметров отличается. На каждой конфигурации алгоритм обучения с подкреплением запускался по 5 раз, результаты работы алгоритма усреднялись, а именно: значение целевой функции, количество выполненных в срок заданий, количество просроченных заданий.

Параметры различных конфигураций РВС, используемых в экспериментальном исследовании, представлены в Таблицах 2-3:

Название параметра	**Значение**
Кол-во итераций в эпизоде	500
Общее кол-во ВУ	10
Кол-во периферийных ВУ	5, 7, 9
Распределение времени поступления заданий	Равномерное, Пуассона

Таблица 2: Параметры конфигурация для экспериментов с 10 ВУ.

При обучении алгоритма *PPO* использовалась функция награды $\mathfrak{R}$ (см. раздел 3, формула 3). Однако в процессе обучения алгоритма значение целевой функции $\mathfrak{F}$ (см. раздел 1, формула 1) также вычислялось. Так как в конечном счете нас интересует именно значение целевой функции, на рисунках с результатами запусков алгоритма на различных конфигурациях РВС представлена зависимость значений именно целевой функции от количества эпизодов обучения.

Название параметра	Значение
Кол-во итераций в эпизоде	500
Общее кол-во ВУ	15
Кол-во периферийных ВУ	5, 7, 9 11, 13, 15
Распределение времени поступления заданий	Равномерное, Пуассона

Таблица 3: Параметры конфигурация для экспериментов с 15 ВУ.

На каждой конфигурации выбранный многоагентный алгоритм обучения с подкреплением *PPO* работал минимум 1 час до тех пор, пока не сходился к стационарному состоянию. Как уже упоминалось ранее каждый эпизод состоит из 500 итераций, поэтому количество итераций до достижения стационарного состояния можно оценить, как $500 * количество\_эпизодов$.

Детальное описание входных данных вместе с примерами данных, которые использовались в экспериментальном исследовании, публично доступны на Github [16]:

Характеристики используемого оборудования для проведения экспериментального исследования:

- ОЗУ — 16 ГБ;
- ЦП — Intel(R) Core(TM) i7 CPU 2.9GHz, 4 ядра;
- ГП — Intel UHD Graphics 630.

7.3 Результаты экспериментов

Исследование влияния общего количества агентов и количества периферийных агентов на скорость сходимости алгоритма

Рис. 2-5 показывают результаты экспериментального исследования алгоритма обучения с подкреплением *PPO*, запущенного на различных конфигурациях РВС. Каждый рисунок состоит из двух частей: зависимость количества просроченных заданий от количества эпизодов обучения и зависимость значения целевой функции от количества эпизодов обучения. По Рис. 2-5 можно сделать следующие выводы:

1. Чем больше общее количество агентов, тем дольше алгоритм сходится к стационарному состоянию. Для 15 ВУ для

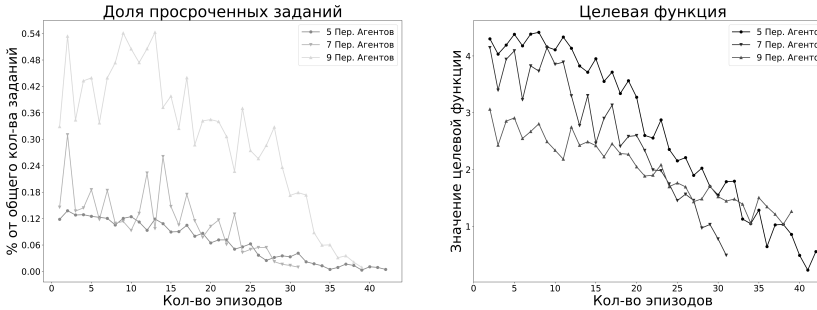

Рис. 2: Распределение Пуассона, общее количество агентов 10, количество периферийных агентов 5, 7, 9.

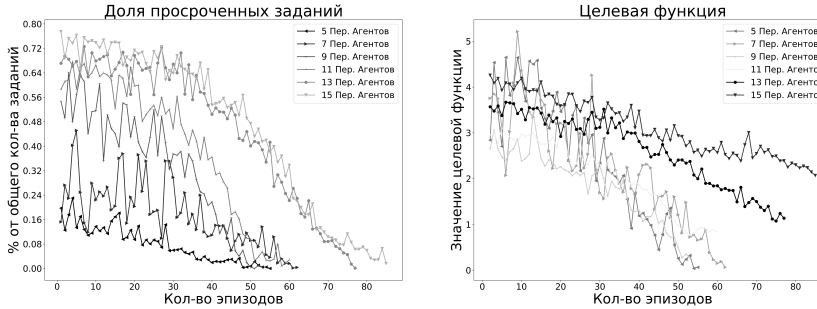

Рис. 3: Распределение Пуассона, общее количество агентов 15, количество периферийных агентов 5, 7, 9, 11, 13, 15.

сходимости необходимо в два раза больше эпизодов. Такое поведение можно объяснить тем, что чем больше агентов, тем больше перемещений между ВУ совершают задания, и в результате этого большее количество заданий не могут завершиться в срок.

2. Чем больше количество периферийных агентов, тем дольше алгоритм сходится к стационарному состоянию. Для конфигурации РВС с 10 агентами это верно для любого количества периферийных агентов. Для конфигурации РВС с 15 агентами это верно не всегда. В проведенных экспериментах для конфигураций с 7 и 9 периферийными агентами наблюдается самая медленная сходимость. Возможно это выброс и необходимо совершить больше повторных запусков алгоритма. Это предмет для дальнейшего

изучения.

3. Алгоритм всегда сходится к стационарному состоянию независимо от общего количества агентов и количества периферийных агентов.

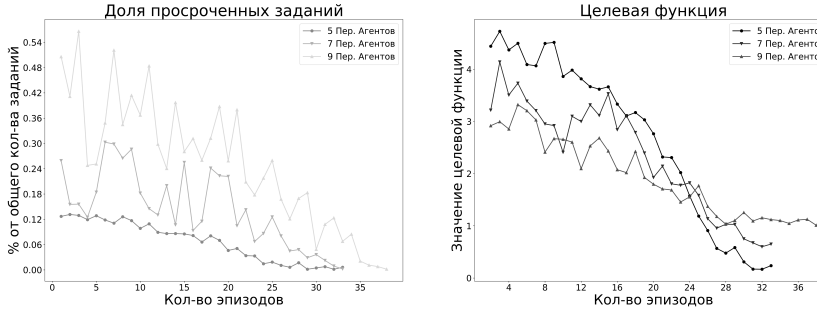

Рис. 4: Равномерное распределение, общее количество агентов 10, количество периферийных агентов 5, 7, 9.

Исследование влияния распределения времени поступления задания в РВС на скорость сходимости алгоритма

На Рис. 2-5 показаны графики исследуемых значений в зависимости от выбранной функции распределения времени поступления задачи к периферийным агентам РВС. Согласно зависимостям, представленным на Рис. 2-5, можно сделать выводы, что для всех рассмотренных распределений верно:

1. Значение целевой функции уменьшается (с разной скоростью для разных распределений) и стабилизируется во время обучения;

2. Количество просроченных заданий уменьшается (с разной скоростью для разных распределений) в процессе обучения.

Эксперименты показали, что при различных конфигурациях РВС (Рис. 2-3) и различных функциях распределения времени поступления заданий (Рис. 4-5), в процессе обучения разработанного алгоритма уменьшается значение целевой функции, количество просроченных заданий. С ростом общего количества агентов пространство действий алгоритма и пространство состояний среды алгоритма обучения с подкреплением

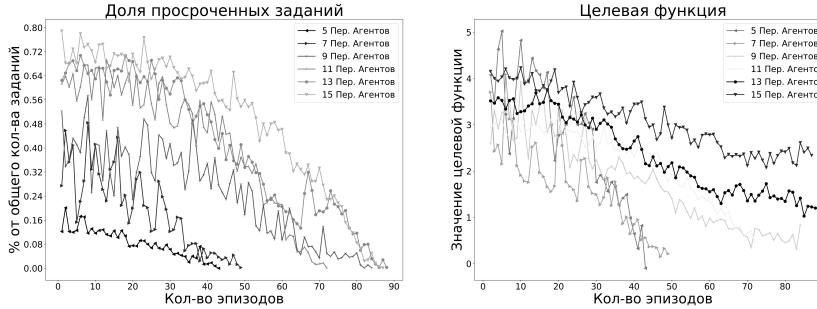

Рис. 5: Равномерное распределение, общее количество агентов 15, количество периферийных агентов 5, 7, 9, 11, 13, 15.

увеличивается, что затрудняет выбор оптимальных политик, но даже в этом случае наблюдается сходимость исследуемых значений.

Результаты эксперментов и реализованный алгоритм публично доступны на GitHub [16].

Заключение

В результате проведенного исследования задача балансировки нагрузки в гетерогенной распределенной вычислительной системе представлена как стохастическая игра. Задача сводится к однокритериальной с помощью целевой функции (см. раздел 1, формула 1), которая минимизирует затраты ресурсов (вычислительные ресурсы, длину очереди) вычислительных узлов вычислительной системы, а также уменьшает количество просроченных заданий, что в свою очередь увеличивает количество выполненных в срок заданий. Разработан многоагентный алгоритм обучения с подкреплением для рассматриваемой задачи и реализована его программная реализация. Проведено экспериментальное исследование предложенного алгоритма *MARL* с различными конфигурациями РВС и различными функциями распределения времени поступления заданий в РВС. Экспериментальное исследование показало эффективность применения реализованного алгоритма к рассматриваемой задаче.

В этой статье был рассмотрен общий случай распределенной гетерогенной вычислительной системы без привязки к конкретной предметной области. Для простоты изложения и демонстрации применимости предлагаемого алгоритма балансировки нагрузки на основе алгоритма *PPO* были использованы синтетические данные,

сгенерированные с использованием двух различных функций распределения времени прибытия заданий в РВС: Пуассоновское распределение и равномерное распределение.

Возможные направления будущей работы заключаются в следующем:

- Сравнить между собой различные многоагентные алгоритмы обучения с подкреплением. Провести сравнение с классическими методами оптимизации;
- Использовать реальные данные в экспериментальной исследовании;
- Провести эксперимент на масштабируемость разработанного алгоритма: рассмотреть РВС с большим количеством агентов (более 100 агентов);
- Рассмотреть модифицированную математическую постановку задачи. В целевую функцию добавить поддержку дополнительных ресурсов вычислительных узлов (ОЗУ, дисковая память). Учитывать время, затрачиваемое на передачу локальной информации о состоянии агентов. Добавить зависимости между заданиями.

Литература

1. Zhang K. et al., *Mobile-edge computing for vehicular networks: A promising network paradigm with predictive off-loading* // IEEE Vehicular Technology Magazine, 2017, Vol. 12, no. 2, pp. 36-44.
2. Abbas N. et al., *Mobile edge computing: A survey* // IEEE Internet of Things Journal, 2017, Vol. 5, no. 1, pp. 450-465.
3. Andriotis C. P., Papakonstantinou K. G., *Managing engineering systems with large state and action spaces through deep reinforcement learning* // Reliability Engineering & System Safety, 2019, Vol. 191, p. 106483.
4. Hameed M. S. A., Schwung A., *Reinforcement learning on job shop scheduling problems using graph networks* // arXiv preprint arXiv:2009.03836, 2020.

5. Milan S. T. et al., *Nature inspired meta-heuristic algorithms for solving the load-balancing problem in cloud environments* // Computers & Operations Research, 2019, Vol. 110, pp. 159-187.

6. Luong N. C. et al., *Applications of deep reinforcement learning in communications and networking: A survey* // IEEE Communications Surveys & Tutorials, 2019, Vol. 21, no. 4, pp. 3133-3174.

7. Zhang Q. et al., *Task offloading and resource scheduling in hybrid edge-cloud networks* // IEEE Access, 2021, Vol. 9. pp. 85350-85366.

8. Pan S. et al., *Dependency-aware computation offloading in mobile edge computing: A reinforcement learning approach* // IEEE Access, 2019, Vol. 7. pp. 134742-134753.

9. Nguyen D. et al., *Cooperative Task Offloading and Block Mining in Blockchain-based Edge Computing with Multi-agent Deep Reinforcement Learning* // IEEE Transactions on Mobile Computing, 2021.

10. Xu Y., Feriani A., Hossain E., *Decentralized Multi-Agent Reinforcement Learning for Task Offloading Under Uncertainty* // arXiv preprint arXiv:2107.08114, 2021.

11. Cao Z. et al., *Multiagent deep reinforcement learning for joint multichannel access and task offloading of mobile-edge computing in industry 4.0* // IEEE Internet of Things Journal, 2020, Vol. 7, no. 7, pp. 6201-6213.

12. Seid A. M. et al., *Multi-agent DRL for task offloading and resource allocation in multi-UAV enabled IoT edge network* // IEEE Transactions on Network and Service Management, 2021, Vol. 18, no. 4, pp. 4531-4547.

13. Schulman J. et al., *Proximal policy optimization algorithms* // arXiv preprint arXiv:1707.06347, 2017.

14. Lillicrap T. P. et al., *Continuous control with deep reinforcement learning* // arXiv preprint arXiv:1509.02971, 2015.

15. Grondman I. et al., *A survey of actor-critic reinforcement learning: Standard and natural policy gradients* // IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews), 2012, Vol. 42, no. 6, pp. 1291-1307.

16. *MARL Load Balancing Environment* [Онлайн] URL: https://github.com/alexey-kaz/customenv [Дата последнего обращения: 07-Июн-2022].

17. *RLlib: Industry-Grade Reinforcement Learning with TensorFlow and PyTorch* [Онлайн] URL: https://github.com/ray-project/ray/tree/master/rllib [Дата последнего обращения: 14-Май-2022].

18. *Python pandas* [Онлайн] URL: https://pandas.pydata.org [Дата последнего обращения: 29-Апр-2022].

19. *Python Matplotlib* [Онлайн] URL: https://matplotlib.org [Дата последнего обращения: 29-Апр-2022].

Коваленко А. П., Сальников А. Н.

Разработка метода трехмерной визуализации задержек в коммуникационной среде кластера

Введение

В современном мире многие значимые и вычислительно сложные задачи решаются не на отдельных компьютерах, а на вычислительных кластерах. В общем случае, вычислительный кластер — это набор компьютеров (вычислительных узлов), объединенных некоторой коммуникационной сетью. Каждый вычислительный узел имеет несколько многоядерных процессоров, свою оперативную память и работает под управлением своей операционной системы. Наиболее распространенным является использование однородных кластеров, то есть таких, где все узлы одинаковы по своей архитектуре и производительности.

Узлы взаимодействуют через коммуникационную среду кластера, которая представляет из себя высокоскоростную сеть. Однако даже в такой сети присутствуют задержки при передаче данных, а задержка — это та характеристика коммуникационной среды, которая сильно влияет на время решения задачи.

Анализ задержек в сети крайне важен для более полного понимания процессов, происходящих в кластере при решении той или иной задачи, а также для выявления возможных ошибок и проблем.

Значения задержек можно измерить, но для того, чтобы интерпретировать полученные численные значения, нужно перевести их в вид, более подходящий для зрительного наблюдения и анализа. То есть нужно построить изображение топологии кластера с указанием значений задержек на линках. Задача визуализации состоит в том, чтобы представить граф в понятном для человека виде, подчеркивающем его структуру и топологические свойства, в том числе удалённость вершин друг от друга в зависимости от величины задержки.

Под линками (или дугами) будем понимать физические связи между вычислительными узлами. Топологией называется конфигурация графа, вершинам которого соответствуют конечные узлы сети и коммуникационное оборудование, а рёбрам – линки.

1. Постановка задачи

Данная работа использует в качестве входных данных сведения о задержках в сети полученные с помощью программы network-tests2, принципы работы которой описаны в статье [1].

Предположим, что на многопроцессорной системе (кластере) было произведено предварительное тестирование коммуникационной среды. Для взаимодействия вычислительных узлов использовался стандарт MPI. В результате этого тестирования получены значения задержек при передаче данных в дискретном трехмерном пространстве размерности $N \times N \times M$, где N – множество процессоров/узлов вычислительной системы, а M – множество значений длин передаваемых сообщений. Результаты в таком виде предоставляет программный инструмент network-tests2.

Полученные данные можно преобразовать таким образом, чтобы получить матрицу задержек, с помощью которой в дальнейшем производится анализ топологии и визуализация.

Неориентированный граф в трехмерном пространстве изображается следующим образом: каждой вершине ставится в соответствие точка, а ребро представляется в виде отрезка, соединяющего две соответствующие вершины-точки. Таким образом, задача отображения графа сводится к расчету трехмерных координат точек. Данная формулировка соответствует классической задаче многомерного шкалирования, то есть задаче отображения многомерных данных в пространство малой размерности. Для решения задачи вводится функция стресса — мера погрешности отображения расстояния между точками и ищется ее минимум. Координаты точек, соответствующие минимуму функции стресса принимаются, как итоговые координаты вершин графа. Такой метод используется для визуализации в приложении Network Viewer 2 (основано на результатах network-tests2) и в этой работе.

Целью работы является усовершенствование метода многомерного шкалирования, и его интеграция в приложение Network Viewer 2. Это позволяет более наглядно отобразить топологию коммуникационной среды кластера и степень загруженности линков.

2. Исследование существующих методов и построение решения задачи.

Для визуализации больших графов часто используются методы, основанные на физических аналогиях. Для построения укладки строится специальная модель, в которой вершины и ребра графа соответствуют «реальным» физическим взаимодействующим объектам. Для этой системы вводится функция энергии таким образом, что конфигурации с меньшим уровнем энергии соответствуют лучшим укладкам. При этом задача поиска лучшей укладки графа сводится к поиску минимума энергии системы.

2.1 Пружинная модель

Одна из основополагающих работ в изучении визуализации графов — статья [2], в которой описывается пружинная модель визуализации ненаправленных графов. Вершины графа рассматриваются как тела, соединенные пружинами. Идея метода — минимизация энергии этой системы тел, в точке минимума достигается наилучшая укладка графа. Функция энергии (функция стресса):

$$E = \sum_{i<j}^{n} \omega_{i,j}(||x_i - x_j|| - d_{i,j})^2$$

Здесь x_i, x_j – векторы координат вершин i и j соответственно, $||x_i - x_j||$ – евклидово расстояние между вершинами i и j, $d_{i,j}$ - «идеальное» заданное расстояние между вершинами. Значение $\omega_{i,j}$ моделирует коэффициент упругости пружины и играет роль нормировочной константы. Обычно $\omega_{i,j}$ выражают формулой $\omega_{i,j} = d_{i,j}^{-\alpha}$. Kamada и Kawai выбрали $\alpha = 2$, а Cohen в статье [3] использует $\alpha = 0$ и $\alpha = 1$. Поиск оптимальных координат осуществляется методом Ньютона-Рафсона [4].

2.2 Мажоризация функции стресса

Развивая идею многомерного шкалирования и пружинную модель Kamada-Kawai, был предложен подход, который получил название мажоризация функции стресса (stress majorization) [5].

Мажоризация дает ряд преимуществ по сравнению с локальными процессами, такими как метод Ньютона-Рафсона и градиентный спуск. В том числе гарантируется монотонное убывание функции стресса, устойчивость к локальным минимумам

и более короткое время работы. Мажоризация также подходит для работы с разреженными моделями. Далее рассмотрим, в чем заключается суть метода.

Преобразуем функцию стресса. Возьмем $\omega_{i,j} = d_{i,j}^{-2}$, объекты запишем в виде матрицы X размерности $n \times k$, где n – число объектов, k – размерность пространства, X_i, X_j – строки матрицы, соответствующие координатам объектов i и j.

$$stress(X) = \sum_{i<j}^{n} \omega_{i,j} d_{i,j}^2 + \sum_{i<j}^{n} \omega_{i,j} ||X_i - X_j||^2 - 2 \sum_{i<j}^{n} \delta_{i,j} ||X_i - X_j||,$$

где $\delta_{i,j} = \omega_{i,j} d_{i,j}$, $i, j = 1, ..., n$.

Эту функцию можно оценить сверху выпуклой функцией $F^Z(X)$ (подробное обоснование приведено в [5]):

$$F^Z(X) = \sum_{i<j}^{n} \omega_{i,j} d_{i,j}^2 + Tr(X^T L^\omega X) - 2Tr(X^T L^Z Z)$$

где Z – произвольная константная матрица размерности $n \times k$, $tr(A)$ – след матрицы , матрица L^ω задана как

$$L_{i,j}^{\omega} = \begin{cases} -\omega_{i,j}, i \neq j \\ \sum\limits_{m \neq i}^{n} \omega_{i,m}, i = j \end{cases}$$

Матрица L^Z размерности $n \times n$ имеет вид:

$$L_{i,j}^{Z} = \begin{cases} -\delta_{i,j} inv(||Z_i - Z_j||), i \neq j \\ -\sum\limits_{j \neq i} L_{i,j}^{Z}, i = j \end{cases}$$

Здесь $inv(x) = \frac{1}{x}$, если $x \neq 0$, и 0 иначе.

В итоге получаем оценку:

$$stress(X) \leq F^Z(X)$$

Равенство достигается при $Z = X$. Функция $F^Z(X)$ выпуклая, а значит, имеет единственный оптимум и ее минимизация гарантирует быструю сходимость [5].

2.3 Описание итогового алгоритма

Для того, чтобы найти оптимум, введем итерационный процесс таким образом, чтобы значение функции стресса на каждой итерации уменьшалось. Рассматриваем функцию $F^{X(t)}(X) = stress(X(t))$. Значения каждой компоненты $X(t+1)$ получаем, решая систему

$$L^{\omega} X(t+1)^{(a)} = L^{X(t)} X(t)^{(a)}, a = 1, ..., k$$

Напомним, что здесь k – размерность пространства вершин. Итерационный процесс завершается, когда выполнено

$$\frac{stress(X(t)) - stress(X(t+1))}{stress(X(t))} < \varepsilon,$$

здесь $\varepsilon = 10^{-10}$.

Чтобы упростить вычисления и не решать систему каждый раз, воспользуемся методом, предложенным Kamada и Kawai [2]. Зафиксируем положения всех вершин, кроме одной i-ой вершины. Тогда новые координаты этой вершины вычисляются по формуле:

$$X_i^{(a)} = \frac{\sum_{i \neq j} \omega_{i,j} (X_j^{(a)} + d_{ij}(X_i^{(a)} - X_j^{(a)}) inv(||X_i - X_j||))}{\sum_{i \neq j} \omega_{i,j}}, a = 1, ..., k$$

Таким образом, можно итерироваться по всем вершинам и на каждой итерации последовательно менять k координат вершины i. На каждой итерации гарантируется уменьшение функции стресса.

3. Экспериментальное исследование полученного решения

Приложение Network Viewer 2 написано на языке C++ с использованием фреймворка Qt и графической библиотеки Qwt. Исходный код находится в репозитории `https://github.com/NastyaKov01/network-tests2`.

На вход приложению подаются файлы в формате NetCDF с расширением .nc или текстовые файлы. Начальные положения вершин задаются случайным образом.

Для оценки полученных результатов была введена метрика качества, которая оценивает соответствие длин ребер полученного графа тем расстояниям между вершинами, которые были заданы изначально.

Для каждого ребра вычисляется коэффициент соответствия длины по следующей формуле:

$$dev_{i,j} = \frac{abs(||X_i - X_j|| - d_{i,j})}{d_{i,j}},$$

где X_i, X_j – векторы координат вершин i и j соответственно, $||X_i - X_j||$ – евклидово расстояние между вершинами i и j, $d_{i,j}$ - «идеальное» заданное расстояние между вершинами, $abs(x)$ – функция, вычисляющая модуль аргумента. Чем меньше это значение для каждого ребра, тем лучше оно аппроксимирует изначальное расстояние между вершинами.

Тогда метрика качества имеет вид:

$$rate(X) = \sum_{i,j}(dev_{i,j} \geq s),$$

то есть вычисляется количество ребер, длина которых отличается от изначально заданной не меньше, чем в s раз. Чем меньше значение введенной метрики, тем лучше полученная модель аппроксимирует исходную топологию. В проведенных экспериментах рассматривается $s = 0.5$ и $s = 0.98$. В дальнейшем будем называть эту метрику погрешностью визуализации топологии.

3.1 Результаты экспериментов

Была проведена серия экспериментов с данными задержек, измеренных на суперкомпьютерах JUROPA, Ломоносов, Ломоносов-2.

Были измерены средние показатели задержек, среднеквадратичное отклонение, медианы и минимальные значения задержек. Считаем эти значения изначально заданными эталонными длинами ребер между вершинами. Для каждого вида данных посчитано количество ребер (в процентах от общего количества), итоговые длины которых отличаются от изначально заданных не менее, чем на 50% и не менее, чем на 98%.

В таблицах ниже приведены результаты измерений. В первых двух столбцах указаны погрешности визуализации при использовании исходного алгоритма, в третьем и четвертом столбцах указаны показатели для усовершенствованного алгоритма. Чем меньше значение в ячейке, тем лучше полученная модель аппроксимирует исходную топологию.

	Погрешность не менее 50% (изначально)	Погрешность не менее 98% (изначально)	Погрешность не менее 50% (итоговая)	Погрешность не менее 98% (итоговая)
Средние значения задержек	60.38%	1.24%	7.23%	0%
Средние отклонения значений задержек	73.07%	23.36%	12.8%	1.39%
Медианы значений задержек	58.53%	1.79%	10.62%	0%
Мин. значения задержек	63.23%	0.94%	14%	2.53%

Таблица 1: Измерение качества визуализации задержек в сети суперкомпьютера jurora (64 узла, 2016 ребер)

	Погрешность не менее 50% (изначально)	Погрешность не менее 98% (изначально)	Погрешность не менее 50% (итоговая)	Погрешность не менее 98% (итоговая)
Средние значения задержек	65.51%	1.34%	10.7%	1.57%
Средние отклонения значений задержек	62.34%	1.24%	11.25%	0%
Медианы значений задержек	60.56%	1.43%	10.26%	0%
Мин. значения задержек	63.99%	1.46%	13.43%	1.75%

Таблица 2: Измерение качества визуализации задержек в сети суперкомпьютера jurora (128 узлов, 8128 ребер)

	Погрешность не менее 50% (изначально)	Погрешность не менее 98% (изначально)	Погрешность не менее 50% (итоговая)	Погрешность не менее 98% (итоговая)
Средние значения задержек	64.54%	16.46%	23%	13.37%
Средние отклонения значений задержек	69.69%	17.72%	23.35%	13.54%
Медианы значений задержек	40.3%	6.14%	9.65%	0%
Мин. значения задержек	65.77%	13.47%	19.44%	6.05%

Таблица 3: Измерение качества визуализации задержек в сети суперкомпьютера Ломоносов-2 (118 узлов, 6903 ребра)

	Погрешность не менее 50% (изначально)	Погрешность не менее 98% (изначально)	Погрешность не менее 50% (итоговая)	Погрешность не менее 98% (итоговая)
Средние значения задержек	97.1%	69.38%	64.67%	49.97%
Средние отклонения значений задержек	97.09%	71.49%	68.82%	53.03%
Медианы значений задержек	36.34%	21.01%	33.67%	24.26%
Мин. значения задержек	58.8%	11.73%	15.69%	3.34%

Таблица 4: Измерение качества визуализации задержек в сети суперкомпьютера Ломоносов (500 узлов, 124640 ребер)

Видно, что показатели усовершенствованного алгоритма гораздо лучше исходного. С увеличением числа вершин и ребер погрешность также растет, но все еще существенно ниже

погрешности оригинального алгоритма.

Мажоризация не только повысила качество визуализации, но и позволила сократить время работы приложения. Если раньше для построения графа, состоящего из 500 вершин, требовалось несколько сотен и даже тысяч итераций, то после модификации алгоритма это число сократилось до нескольких десятков.

Заключение

В рамках данной работы были рассмотрены существующие методы визуализации графов и был предложен алгоритм визуализации задержек в коммуникационной среде кластера.

Имеющееся приложение Network Viewer 2 было переписано с использованием мажоризации функции стресса. Итоговый алгоритм работает значительно быстрее исходного и снижает погрешность построения длин ребер в графе.

Была проведена серия экспериментов с данными задержек, измеренных на суперкомпьютерах JUROPA, Ломоносов, Ломоносов-2. На основании результатов этих экспериментов был проведен сравнительный анализ исходного и полученного алгоритмов.

Итоговое приложение может быть использовано для визуализации задержек в коммуникационной среде кластера, выявления и анализа возможных ошибок и проблем в сети.

Литература

1. Банников П. С., Сальников А. Н. *Выявление топологии коммуникационной среды вычислительного кластера по результатам нагрузочного тестирования.* Вестник Московского университета. Серия 15: Вычислительная математика и кибернетика, 2014, с. 43–52.

2. Tomihisa Kamada, Satoru Kawai *An algorithm for drawing general undirected graphs.* Information Processing Letters, 1989, p. 7–15.

3. Cohen J. D. *Drawing Graphs to Convey Proximity: an Incremental Arrangement Method.* ACM Transactions on Computer-Human Interaction, 1997, p. 197–229.

4. Ньютон И. *Математические работы / Пер. с латинского, вводная статья и комментарии Д.Д. Мордухай-Болтовского.* Москва-Ленинград: ОНТИ, 1937.

5. Gansner E. R., Koren Y., North S. *Graph Drawing by Stress Majorization.* International Symposium on Graph Drawing, 2004, p. 239–250.

Порывай М.В., Чупахин А.А.

Исследование алгоритмов решения задачи нерегулярного размещения плоских геометрических фигур

Введение

Во многих областях промышленности, таких, как, например, сталелитейная, кожевенная, текстильная, часто возникает задача вырезания из больших фрагментов материала заданных фигур произвольной формы. При этом группировать места размещения контуров фигур для раскроя фрагмента материала требуется максимально плотно, чтобы минимизировать остаток. Так, экономия 1% материала с каждого исходного фрагмента может предотвратить потерю предприятием в сумме нескольких миллионов долларов США [1].

Задачи такого типа называются в литературе задачами раскроя, или задачами упаковки. Они могут варьироваться по параметрам входных данных, таких, как, например, форма и размеры фрагментов материала и размещаемых фигур, ограничения на полезный остаток.

В данной работе рассматривается двумерная нерегулярная задача раскроя. Нерегулярность означает, что фигуры могут быть не только прямоугольниками, но и иметь достаточно сложную форму. Двумерная задача — это означает, что фигуры заданы на двумерной плоскости.

Задача раскроя является NP-трудной [2], и на практике нахождение ее точного решения требует много как временных, так и вычислительных ресурсов. Поэтому при ее решении часто применяются приближенные (эвристические) алгоритмы, которые позволяют за приемлемое время получать пусть не оптимальный, но хороший результат.

Известно большое количество классических эвристических алгоритмов. Среди них можно выделить, например, класс природных алгоритмов, основанных на естественных природных процессах. Также на основе процесса отжига металлов в конце прошлого века был разработан алгоритм имитации отжига. Эти алгоритмы хорошо зарекомендовали себя при решении таких NP-трудных задач, как построение расписания выполнения работ без прерываний, построение оптимального маршрута, распознавание структуры белков, финансовое прогнозирование.

В данной работе задача раскроя решается с помощью алгоритма имитации отжига и жадного алгоритма.

Результатом проведенного исследования является создание программной библиотеки, содержащей реализации данных алгоритмов, решающих задачу раскроя в заданной постановке, оформленных в едином стиле, с единой иерархией классов.

1. Постановка задачи

Обозначения

1. $Q = \{q_i, i = 1, \ldots, M, \ldots\}$ — бесконечное множество одинаковых листов прямоугольной формы, стороны каждого листа равны a, b, $a \geq b$ (для определенности).

2. $S = \{s_i, i = 1, \ldots, n\}$ — конечное множество фигур (выпуклых многоугольников без отверстий); каждый многоугольник s_i задан последовательностью своих вершин $s_{ij}, j = 1, \ldots, k_i$ (вершина $s_{ij} = (x_{ij}, y_{ij})$ — точка в первой четверти плоскости OXY, начало координат O помещено в левый нижний угол листа, координатные оси OX и OY направлены вдоль смежных сторон листа соответственно горизонтально вправо и вертикально наверх).

3. M — количество листов, использованных для размещения заданного набора фигур S.

4. d_i — выпуклая оболочка для всех фигур, размещенных на листе q_i.

5. e_i — минимальный прямоугольник со сторонами, параллельными координатным осям (сторонам листа), в который можно вписать все фигуры, размещенные на листе q_i.

6. $sq(x)$ — площадь объекта x, где $x \in S \cup Q \cup \{d_i, i = 1, \ldots, M\} \cup \{e_i, i = 1, \ldots, M\}$.

7. Если фигура s_j помещена на лист q_i, будем писать $s_j \in q_i$.

8. FF_a (мера незанятости листов) — средняя по всем листам

площадь части листа, не занятой фигурами.

$$FF_a = \frac{\sum\limits_{i=1}^{M-1} \{(1 - \sum\limits_{s_j \in q_i} sq(s_j))/sq(q_i)\} + (1 - \sum\limits_{s_j \in t_M} sq(s_j))/sq(e_M)}{M} \quad (1)$$

9. FF_b (мера неплотности размещения) — средняя по всем листам площадь части выпуклой оболочки всех фигур на листе, не занятой фигурами.

$$FF_b = \frac{\sum\limits_{i=1}^{M} (1 - \sum\limits_{s_j \in q_i} sq(s_j))/sq(d_i)}{M} \quad (2)$$

Дано

1. Q, a, b.
2. S, s_{ij} $(i = 1, \ldots, n; j = 1, \ldots, k_i)$

Ограничения

1. $\forall i \in \{1, \ldots, n\}$: в s_i нет отверстий.
2. $\forall i \in \{1, \ldots, n\}$: s_i может быть повернута на углы $\alpha = 90t$ $(t \in \{0, 1, 2, 3\})$ градусов с помощью следующего преобразования:
   $\forall j \in \{1, \ldots, k_i\}$:
   $(x_{ij}, y_{ij}) \to (x_{ij} \cdot \cos\alpha - y_{ij} \cdot \sin\alpha, x_{ij} \cdot \sin\alpha + y_{ij} \cdot \cos\alpha)$
3. $\forall i \in \{1, \ldots, n\}$: s_i может быть зеркально отражена с помощью следующего преобразования:
   $\forall j \in \{1, \ldots, k_i\} : (x_{ij}, y_{ij}) \to (x_{ij}, -y_{ij})$

Найти

Необходимо найти размещение фигур на листах, удовлетворяющее требованию максимизации специально введенной функции свертки:

$$FF = 1 - (A \cdot FF_a + B \cdot FF_b) \to max \quad (3)$$

Замечания

1. Если трансформация фигуры приводит к тому, что координаты некоторых вершин становятся отрицательными, то происходит сдвиг фигуры (всех ее координат) в первую четверть (на самом деле, важно лишь относительное положение координат по отношению друг к другу, т.е. после трансформации форма фигуры должна сохраняться, первая четверть выбрана для удобства).

2. Значения коэффициентов A и B можно подбирать экспериментальным путем, но в данной работе они рассматриваются равными 0.5, что приводит к нормализации значений функции свертки на отрезок $[0, 1]$.

3. Задача поставлена как задача максимизации для лучшей интерпретируемости — удобно, когда значение функции свертки определяет качество, а не неоптимальность полученного размещения в процентах.

4. Задача многокритериальной оптимизации сведена к задаче однокритериальной оптимизации.

2. Обзор предметной области

Задача раскроя является NP-трудной, а значит для ее решения, особенно при больших размерностях входных данных, необходимо использовать приближенные алгоритмы, в частности, эвристические алгоритмы. В рамках обзора были рассмотрены как точные алгоритмы: смешанно-целочисленное линейное программирование, метод ветвей и границ [4] — так и эвристические алгоритмы: природные (эволюционные [5], роя частиц [6], муравьиный [7]), поиск с запретами [8], имитация отжига [9] и др. [10].

В ходе выполнения обзора предметной области было выяснено, что математические постановки задачи раскроя могут сильно различаться по параметрам исходных данных, ограничениям, оптимизирующему функционалу. Например, область размещения может быть конечной, бесконечной, полубесконечной полосой, или же разделенной на дискретные подобласти (листы). Фигуры могут быть прямоугольниками, многоугольниками (выпуклыми/невыпуклыми), иметь произвольную форму, с отверстиями и без, с возможностью поворота и отражения и без. Остаток области (если она конечная) может не приниматься во

внимание, либо на его размер/форму могут накладываться дополнительные ограничения. Тем не менее, некоторые подзадачи, рассматриваемые при решении задачи раскроя в различных постановках, могут быть полезны при решении задачи рассматриваемой в данной работе.

Точные подходы, такие как методы смешанно-целочисленного линейного программирования в комбинации с методом ветвей и границ, не масштабируемы на большое число размещаемых фигур при решении задачи в общем случае: верхняя граница не превышает нескольких десятков фигур. У эволюционного алгоритма, муравьиного алгоритма и алгоритма роя частиц большое число параметров, подбор оптимальных значений которых крайне трудоемкий.

Первым для реализации был выбран алгоритм имитации отжига, имеющий всего два основных параметра. Также было решено разработать детерминированный жадный алгоритм для последующего сравнения результатов работы алгоритмов.

Для размещения фигур с непересечением был выбран метод годографа функции плотного размещения, известный в иностранной литературе как *NFP (no-fit polygon)* [3]. Допустим, имеется зафиксированная фигура из заданного множества. Положение данной фигуры на конкретном листе уже не будет меняться. Также имеется незафиксированная фигура, которую необходимо разместить на этот же лист. У незафиксированной фигуры выбирается опорная точка — любая граничная точка этой фигуры. NFP — это многоугольник, который получается в результате скольжения незафиксированный фигуры по зафиксированной [4]. Различные положения опорной точки во время скольжения определяют координаты многоугольника. Скольжение незафиксированной фигуры достигается только операцией сдвига, повороты запрещены.

Стоит обратить внимание на то, что задача раскроя исследуется уже довольно продолжительное время. Тем не менее, это не является проблемой, потому что суть однотипных алгоритмов, применяемых при решении задачи раскроя во всех статьях примерно одинакова, но в более новых работах суть алгоритмов часто отходит на второй план, уступая место дополнительным незначительным усовершенствованиям. В процессе обзора, однако, было важно получить именно идеи по применению классических алгоритмов в задачах упаковки. Более тонкие моменты, связанные с реализацией, будут рассмотрены в продолжении исследования, когда реализация основных частей алгоритмов будет завершена, в

то время как в данной статье описываются результаты только первого этапа исследования.

3. Описание разработанных алгоритмов

3.1 Жадный алгоритм

Жадные алгоритмы — класс алгоритмов, в которых результат формируется последовательно с помощью жадного критерия, применяемого к определенному параметру (например, к функции, значение которой в задаче необходимо максимизировать или минимизировать): на каждом шаге происходит перебор вариантов для этого шага и выбирается вариант, который дает лучшее значение требуемого параметра на данном шаге.

Жадный алгоритм для решения задачи раскроя в данной работе реализован следующим образом:

```
1  Отсортировать фигуры в порядке убывания площади или
   в случайном порядке (задается при запуске);
2  while (Количество оставшихся фигур > 0):
3      Взять следующую фигуру, объявить ее текущей;
4      Установить значения максимума функции
       свертки равным -1;
5      for (Все преобразования):
6          Применить преобразование к копии текущей фигуры;
7          for (Текущие листы):
8              Вычислить значение функции свертки
               для самого нижнего левого возможного
               размещения фигуры с текущем
               преобразованием на конкретном листе;
9              Если оно больше текущего максимума,
               обновить максимум;
10     if (Никуда разместить не удалось):
11         Взять новый лист;
12         Разместить фигуру в левый нижний угол
           нового листа;
13     else:
14         Применить изменение с максимальным
           значением функции свертки;
15     Убрать фигуру из списка неразмещенных;
```

Сортировка фигур означает, что мы фиксируем порядок, в котором будем брать фигуры из заданного множества для

размещения их на листах.

3.2 Алгоритм имитации отжига

Алгоритм основывается на имитации физического процесса, который происходит при кристаллизации вещества, в том числе при отжиге металлов. Предполагается, что атомы уже выстроились в кристаллическую решётку, но еще допустимы переходы отдельных атомов из одной ячейки в другую. Предполагается, что процесс протекает при постепенно понижающейся температуре. Также определена функции энергии системы, и устойчивая решетка соответствует минимуму энергии. Переход атома из одной ячейки в другую происходит точно, если это понижает энергию системы, и с некоторой вероятностью, если не понижает, причем эта вероятность уменьшается с понижением температуры.

Алгоритм имитации отжига для решения задачи раскроя в данной работе реализован следующим образом:

```
1  Сгенерировать изначальное размещение фигур (сортировка
   случайная или по убыванию площади; задается
   при запуске; отличается от жадного алгоритма тем, что
   к фигурам не применяются операции поворота и отражения,
   и каждый раз фигуры размещаются в самое нижнее
   левое допустимое место только на последнем из листов
   (‘‘новый’’ лист становится доступным тогда, когда
   фигуру нельзя никак разместить на предыдущем листе));
2  Задать начальное значение температуры
   и шаг ее уменьшения;
3  Вычислить значение энергии системы
   E = 100000 * (1 - FF) после изначального размещения;
4  while (температура > 0):
5      Выбрать два случайных листа;
6      Выбрать по одной случайной фигуре
       (со случайной комбинацией поворота и отражения)
       с данных листов;
7      Поменять местами фигуры : переместить
       одну из этих фигур на лист, где изначально
       была расположена другая фигура, и наоборот
       (пробовать размещать фигуры на листе
       в самом нижнем левом доступном месте);
8      if (Удалось разместить две фигуры):
9          Вычислить разность значений энергии системы
           после изменения и до него;
```

```
10          if ((разность < 0) or (случайное число
            из [0, 1] < exp(разность / температура))):
11              Применить изменение;
12              Обновить значение энергии системы;
13              Температура -= шаг;
14          else:
15              Температура -= шаг * 0.1;
16      else:
17          Температура -= шаг * 0.1;
```

Константа 100000 в формуле энергии системы введена потому, что иначе разность энергии системы всегда будет слишком маленьким числом (значения целевой функции FF (3) лежат на отрезке $[0, 1]$, из-за чего степень у экспоненты стремиться к 1, а значит изменения будут применяться всегда и потеряется смысл применения алгоритма). Изменение температуры на десятую часть шага в случае неуспешности попытки размещения сделано для того, чтобы уменьшать температуру, но не растрачивать слишком много шагов алгоритма на действия, ничего качественно не изменяющие.

4. Описание программного модуля

Для решения задачи раскроя в предложенной в данной работе постановке была реализована библиотека [11] на языке программирования *Python*, поскольку это позволяет сделать разработку быстрее, а код — более легким для понимания по сравнению с более сложными языками, например, C++.

При разработке были использованы следующие библиотеки: *NumPy*, *MatPlotLib*, *Shapely*, *Scikit-Geometry*, *Numba*, *Pydantic*, *PyClipper*.

Точки (вершины) в программе представлены кортежами, многоугольники представлены кортежами их вершин, следующих в порядке обхода фигуры по контуру. Набор фигур, которые должны быть размещены, изначально задается словарем *shape (фигура): number (количество)*. Каждая из фигур изначально задается конкретными координатами на плоскости, но фактическое размещение каждой фигуры может подвергаться преобразованиям и учитывать смещения (перемещения) относительно начала координат для исходных координат фигуры. Поэтому в программе каждая фигура представлена в виде кортежа из трех элементов: фигуры в начальных координатах, координат преобразованной

фигуры в начальных координатах, смещения относительно начала координат. Каждый из листов размещения фигур представляет собой список таких трехэлементных кортежей, соответствующих фигурам.

Основным классом библиотеки является класс *Packing*, поля которого: *bin_size* — кортеж, содержащий ширину и длину листов, *bins* — листы из предыдущего абзаца, *polygons* — фигуры из предыдущего абзаца, без преобразований и смещений, *remaining* — словарь оставшихся фигур и их количеств, *coeffs* — кортеж коэффициентов функции свертки.

Для каждого из алгоритмов реализован свой класс: *SA* — алгоритм имитации отжига, *Greedy* — жадный алгоритм.

В классе *Greedy* есть следующие поля: *packing* — элемент класса *Packing*, карта раскроя которого строится алгоритмом, *sort* — используемая начальная сортировка. Методы класса *Greedy*: *greedy()* — сортирует список фигур и подает фигуры по одной в метод *greedy_step()*, где происходит размещение текущей фигуры.

В классе *SA* есть следующие поля: *packing* — элемент класса *Packing*, карта раскроя которого строится алгоритмом, *sort* — используемая начальная сортировка. Методы класса *SA*: *simulated_annealing()* — сортирует фигуры, генерирует начальное размещение и содержит основной цикл алгоритма, внутри данного метода вызывается метод *make_a_swap_move()*, в котором выбираются случайным образом листы и фигуры для перемещения и вызывается метод *nest_polygon_to_a_bin()*, производящий перемещение.

Реализована процедура визуализации полученной карты раскроя с помощью функций *plot_packing()* и *plot_bin()*, использующая модуль *Matplotlib*.

Входные данные в программу подаются с помощью текстового файла специального формата (описание можно посмотреть в файле *README* в [11]), выходной файл содержит текстовый вывод поля *bins* экземпляра класса *Packing*.

5. Экспериментальные исследования

В данном разделе приведено экспериментальное исследование разработанных алгоритмов, сравнение результатов их работы.

5.1 Цели экспериментального исследования

1. Необходимо подобрать число запусков алгоритма имитации отжига на каждом наборе входных данных для получения статистически значимого результата работы алгоритма. Подбор оптимального числа запусков алгоритма с применением метода проверки статистических гипотез в данной работе не рассматривается.

2. Необходимо провести сравнение алгоритма имитации отжига и жадного алгоритма по критериям: качество получаемого решения, время работы алгоритма.

5.2 Методика проведения исследования

Описание входных данных

Было проведено исследование стабильности работы реализованного алгоритма имитации отжига. Для этого было сгенерировано 4 набора фигур: наборы из 100, 200, 300, 400 фигур. В каждом из наборов было поровну следующих видов фигур: равносторонний треугольник со стороной, равной 10; равнобедренный прямоугольный треугольник с катетом, равным 10; равнобедренная трапеция с основаниями, равными 9 и 3, и высотой 5; параллелограмм со сторонами, равными 5, $5 \cdot \sqrt{5}$, и углом между ними, равным $\arctan(2)$; “звезда”, составленная из последовательно соединенных точек $(0, 3)$, $(2, 4)$, $(3, 6)$, $(4, 4)$, $(6, 3)$, $(4, 2)$, $(3, 0)$, $(2, 2)$. Использовались листы размером 30 на 40.

Описание вычислительной среды для проведения экспериментов

В качестве вычислительной среды использовалась ЭВМ со следующими характеристиками:

1. ЦП: AMD Ryzen 7 4700U;

2. Общее количество ядер: 8 x 2 GHz;

3. ОЗУ: 16 ГБ;

4. Объем дискового хранилища (SSD): 512 ГБ;

5. ОС: Ubuntu 20.04.4 LTS.

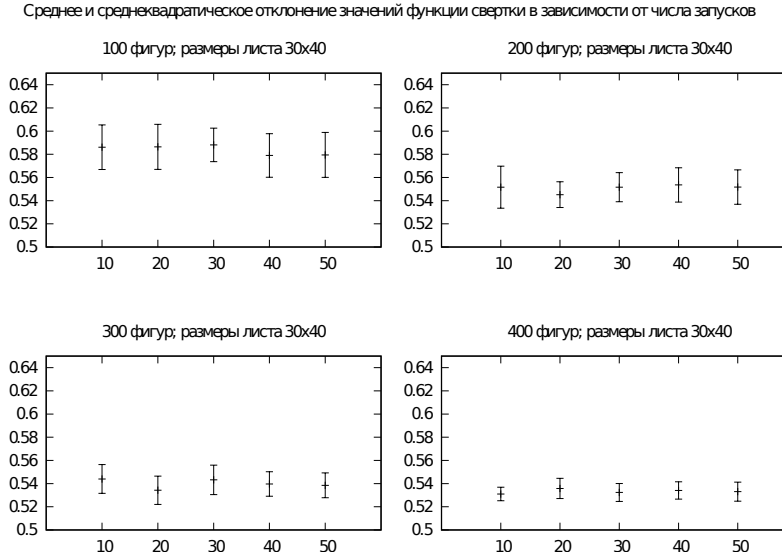

Рисунок 1. Стабильность алгоритма имитации отжига.

Подбор числа запусков

На каждом из четырех наборов фигур было произведено последовательно 10, 20, 30, 40 и 50 запусков алгоритма имитации отжига со случайной сортировкой фигур, с начальной температурой, равной количеству фигур в каждом наборе, и шагом уменьшения температуры, равным 1 для всех наборов. Результат эксперимента представлен на Рис. 1. На оси абсцисс на графиках отражено количество запусков, на оси ординат — значение функции свертки.

С учетом того, что с увеличением числа запусков среднее и стандартный разброс остаются практически неизменными, можно запускать алгоритм имитации отжига на каждом из наборов входных данных по 10 раз.

5.3 Сравнение алгоритмов

Было проведено сравнение реализованных алгоритмов по качеству получаемого решения и по времени работы. Для получения входных данных было сгенерировано 80 наборов фигур. В каждом из наборов были следующие виды фигур: равносторонний треугольник со стороной, равной 10; равнобедренная трапеция с основаниями, равными 9 и 3, и высотой 3; прямоугольник со сторонами, равными 4 и 5. В первом наборе было по 5 фигур каждого вида, во втором — по 10, и т. д. (с шагом 5). Таким образом, в последнем наборе было по 400 фигур каждого вида. Использовались листы размером 30 на 40. Выбор именно таких фигур обоснован тем, что в начале исследования есть смысл исследовать алгоритмы и их реализации на достаточно простых фигурах. Это может помочь найти какие-то концептуальные ошибки, в то время как проведение экспериментального исследования на более сложных фигурах относитс к следующему этапу исследования.

Каждый из алгоритмов был запущен с обеими вариантами начальной сортировки фигур: в случайном порядке и в порядке убывания их площади. Результаты эксперимента представлены на Рис. 2 и Рис. 3. Алгоритм имитации отжига на каждом из наборов входных данных был запущен по 10 раз. На графиках закрашена область между минимальным и максимальным значениями на каждом из 10 запусков; ломаная внутри области соединяет средние значения по 10 запускам.

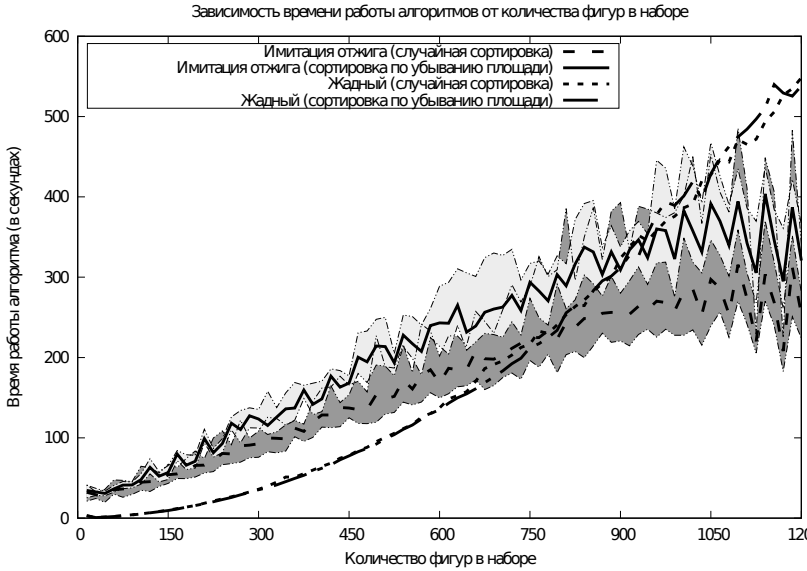

Рисунок 2. Сравнение времени работы алгоритмов.

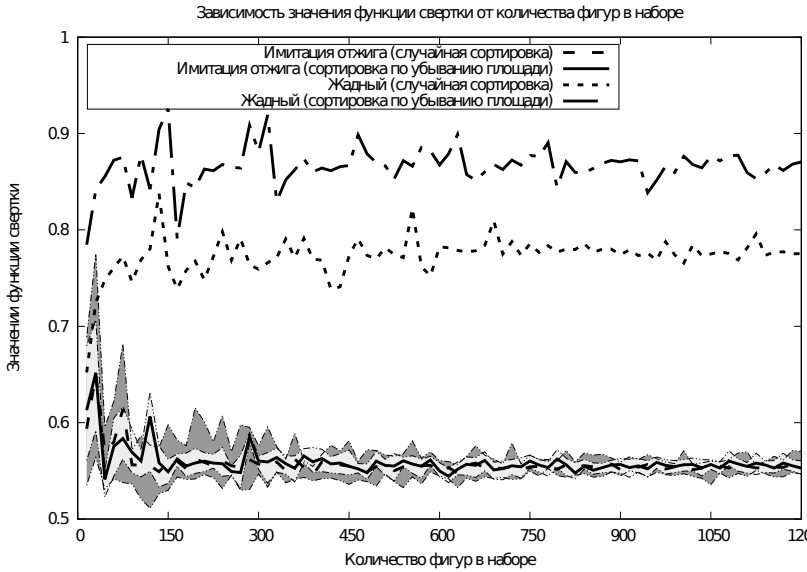

Рисунок 3. Сравнение качества результатов работы алгоритмов.

5.4 Выводы из исследования

1. Жадный алгоритм дает более качественный результат, но при этом время его работы при большом количестве фигур больше, чем у алгоритма имитации отжига из-за того, что в алгоритме присутствует ограниченный перебор: на каждой итерации рассматривается размещение фигуры на всех текущих листах со всеми возможными преобразованиями (комбинациями поворотов и отражений) (см. Раздел 3.1).

2. Алгоритм имитации отжига на данный момент дает более плохой результат. В дальнейшем планируется модифицировать метод перехода к новой карте раскроя и функцию энергии системы, а также определить дополнительные критерии останова.

3. Стратегия сортировки фигур по убыванию площади показала себя намного лучше, чем стратегия случайной сортировки.

Заключение

В работе рассматривались двумерная нерегулярная задача раскроя и способы ее решения. Был проведен аналитический обзор существующих алгоритмов решения нерегулярной двумерной задачи раскроя, разработана математическая постановка задачи. Также были разработаны и реализованы на языке программирования *Python* жадный алгоритм и алгоритм имитации отжига, решающие задачу в предложенной постановке. Проведены экспериментальное исследование свойств разработанных алгоритмов и сравнение результатов их работы на основе разработанной методики.

В продолжении данного исследования планируется провести доработку алгоритма имитации отжига, реализовать другие алгоритмы из рассмотренных в обзоре. Также можно расширить обзор предметной области с помощью рассмотрения более новых статей, представляющих модификации классических алгоритмов, а также, возможно, новые подходы к решению задачи раскроя. Планируется также переписать критичные ко времени выполнения фрагменты программных реализаций алгоритмов на язык программирования *C++*.

Литература

1. *CEPI Key Statistics Report (2012).* `https://web.archive.org/cepi.org/node/16197` [Дата обращения: 21.11.2021]

2. Garey M. R., Johnson D. S. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman and Company, San Francisco, 1979

3. Burke E. K., Hellier R. S. R., Kendall G., Whitwell G. *Complete and robust no-fit polygon generation for the irregular stock cutting problem.* European Journal of Operational Research, 2007, vol. 179, no. 1, pp. 27–49

4. Fischetti M., Luzzi I. *Mixed-Integer Programming Models for the Nesting Problem.* Journal of Heuristic, 2009, vol. 15, no. 3, pp. 201–226

5. Singh G., Lavista Ferres H. *2D Packing problem with irregular shapes.* `https://usnd.to/LF30` [Дата обращения: 27.10.2021]

6. Shalaby M., Kashkoush M. *A Particle Swarm Optimization Algorithm for a 2-D Irregular Strip Packing Problem.* American Journal of Operations Research, 2013, vol. 3, pp. 268–278

7. Верхотуров М. А. *Задача нерегулярного раскроя фигурных заготовок: оптимизация размещения и пути режущего инструмента.* Вестник Уфимского государственного авиационного технического университета, 2007, vol. 9, no. 2, pp. 106–118

8. Blazewicz J., Hawryluk P., Walkowiak R. *Using a tabu search approach for solving the two-dimensional irregular cutting problem.* Annals of Operations Research, 1993, vol. 41, pp. 313–325

9. Dagli C. H., Hajakbari A. *Simulated annealing approach for solving stock cutting problem.* Proceedings of the IEEE International Conference on Systems, Man and Cybernetics, 1990

10. Lopez E., Ochoa G., Terashima-Marin H., Burke E. *An effective heuristic for the two-dimensional irregular bin packing problem.* Annals of Operations Research, 2013, vol. 206, pp. 241-264

11. Poryvai M. *2D irregular stock cutting problem multi-algorithm solution.* `https://github.com/maxporyvay/2D-irregular-stock-cutting-problem-multi-algorithm-solution` [Дата обращения: 12.06.2022]

Шибаев П. П., Чупахин А. А.

Применение фильтра Колмогорова-Винера и нейронных сетей с управляемым рекуррентными блоками для решения задачи Wi-Fi-сканирования

Введение

Wi-Fi-сканирование основано на использовании Wi-Fi-устройств для распознавания активности людей путём определения отклонений и особенностей в различных характеристиках Wi-Fi-сигнала. Данная технология особенно актуальна в преддверии нового стандарта Wi-Fi, в фокусе которого находится активное развитие Wi-Fi сетей для интернета вещей. Технология Wi-Fi-сканирования может иметь применения в сфере транспорта, безопасности домохозяйств, здравоохранении и других сферах жизни. Целью исследования является разработка алгоритма определения присутствия человека в помещении на основе данных об уровне принимаемого сигнала — Received signal strength indicator information (RSSI).

Для достижения указанной цели должны быть решены следующие задачи:

1. разработать математическую постановку решаемой задачи;
2. провести аналитический обзор методов определения присутствия человека по данным RSSI;
3. разработать и собрать экспериментальный стенд для сбора данных RSSI;
4. произвести серию экспериментов по сбору данных;
5. разработать алгоритмы, решающие поставленную задачу, на основе фильтра Колмогорова-Винера и нейронной сети на управляемых рекуррентных блоках;
6. произвести оценку качества разработанных алгоритмов;
7. сравнить качество разработанных алгоритмов с существующими аналогами.

1. Постановка задачи

1.1 Содержательная постановка задачи

В рамках системы из двух Wi-Fi-устройств: приёмника и передатчика (точки доступа) — определить факт присутствия человека между этими устройствами. Определение факта присутствия происходит на основании анализа данных RSSI. В качестве дополнительного условия принимается отсутствие значительных возмущений внешней среды (передвижения габаритных объектов, резких изменений влажности и температуры). Данные условия необходимы для наличия стабильного электромагнитного фона в помещении.

1.2 Математическая постановка задачи

Исходные данные задачи:

1. Вектор тренировочных данных RSSI $r_{train} = [r_1, r_2, r_3, \ldots, r_n]$, каждый элемент вектора представляет собой полную мощность принимаемого приёмником сигнала — RSSI. Мощность сигнала измеряется приёмником по логарифмической шкале, представляет собой целое число и измеряется в дБм (dBm, децибел относительно 1 милливатта);

2. Вектор меток $y_{train} = [y_1, y_2, \ldots, y_n]$, где значение метки 1 соответствует присутствию человека, 0 — отсутствию. i-ое значение в векторе y_{train} соответствует i-ому значению в векторе r_{train};

3. Вектор тестовых данных RSSI, на которых проверяется точность алгоритма: $r_{test} = [r_1, r_2, r_3, \ldots, r_m]$;

4. Вектор меток для тестовых данных $y_{test} = [y_1, y_2, \ldots, y_m]$;

Для определения присутствия человека предлагается использовать алгоритм $A(r)$, входом которого является значение сигнала RSSI — r, выходом либо 1 — присутствие человека, либо 0 — отсутствие человека. Алгоритм $A(r)$ должен обучаться на данных r_{train} и y_{train}.

Качество алгоритма $A(r)$ будем определять следующим образом:

$$Accuracy(A(r)) = \frac{\sum_{i=1}^{m} 1[A(r_{test}[i]) = y_{test}[i]]}{m} \quad (1)$$

Таким образом, решается классическая задача классификации, с метрикой качества (1). Данную метрику качества в литературе называют точностью, или *accuracy* (по англ.).

Необходимо разработать алгоритм $A'(r)$, максимизирующий точность согласно формуле (1).

2. Обзор предметной области

2.1 Критерии обзора

Для проведения аналитического обзора подходов к решению поставленной задачи были выбраны следующие критерии:

1. класс алгоритма: например, статистические алгоритм или алгоритм машинного обучения;
2. наличие специального метода борьбы с шумом в рассматриваемом алгоритме;
3. точность рассматриваемого алгоритма;
4. тип помещения, в котором производился эксперимент.

2.2 Обзор различных подходов к решению задачи определения присутствия человека с помощью данных RSSI

Наиболее частый подход к решению рассматриваемой задачи основан на статистическом подходе. Однако возможно и применение машинного обучения. В Таблице 1 представлен результат сравнения рассмотренных методов по выбранным критериям обзора.

При дальнейшем выборе наилучшего алгоритма для решения поставленной задачи предлагается рассмотреть новые подходы из обеих категорий. Где:

- ФК — фильтр Калмана;
- МО — математическое ожидание;
- Д — дисперсия;
- ВСС — взвешенное скользящее среднее;
- МС — Марковская сеть;
- Лаб. — лабораторное помещение.

Статья	Тип алгоритма	Борьба с шумом	Точность	Помещение
[1]	ФК	Есть	0.95	Лаб.
[2]	Оценка МО и Д	Нет	≥ 0.9	Лаб.
[3]	ВСС	Есть	~ 1	Лаб.
[4]	МС	Есть	0.86	Офис

Таблица 1: Результаты аналитического обзора

3. Реализованный алгоритм определения присутствия человека в помещении на основе фильтра Колмогорова-Винера и фильтра Гампеля

В качестве базового алгоритма решающего рассматриваемую задачу предлагается рассмотреть алгоритм, который состоит из последовательного применения фильтра Колмогорова-Винера, который стоит отнести к группе статистических подходов, и фильтра Гампеля.

Фильтр Колмогорова-Винера является более простым аналогом фильтра Калмана, который успешно используется для борьбы с шумом в сигналах. В нашей реализации [5] фильтр имеет два метода: fit и $transform$. В метод fit передаются данные RSSI в период отсутствия человека для определения уровня шума. В метод $transform$ передаются произвольные данные RSSI, над которым производится операция очистки от шума.

Фильтра Гампеля используется для обнаружений аномалий в очищенных от шума данных.

Таким образом, на основе фильтра Колмогорова-Винера со вспомогательным подходом, который часто называется фильтром Гампеля, можно построить алгоритм для решения рассматриваемой задачи детектирования присутствия человека.

4. Реализованный алгоритм определения присутствия человека в помещении на основе рекуррентой нейронной сети

4.1 Предобработка данных

Для фильтра Колмогорова-Винера предобработка данных не производится. Однако для данных, которые поступают в нейронную сеть, и на вход алгоритмам бустинга, которые используются для сравнения с реализованными алгоритмами, выполняется предобработка данных RSSI. В процессе предобработки применяется техника “скользящего окна”. Кратко опишем данную технику. Пусть w — это размер окна для необработанных данных RSSI: $r_1, ..., r_n$. Составляется матрица $M = [[r_1, ...r_w], [r_2, ..., r_{w+1}], ..., [r_{n-w}, ..., r_n]]$, которая подаётся на вход алгоритмам. Матрица M разделяется построчно на две части: M_{train} (71.5% первых строк матрицы M) и M_{test} (28.5% оставшихся строк матрицы M). M_{train} используется для обучения алгоритмов, M_{test} — для определения качества алгоритма (см. формулу (1)).

4.2 Нейронная сеть на основе рекуррентных управляемых блоков

Рекуррентные нейронные сети (Recurrent Neural Networks, RNN) успешно используются для обработки временных последовательных данных. В частности, хорошо показали себя на данных RSSI сети с долгой краткосрочной памятью (Long short-term memory, LSTM). Альтернативой подходу с LSTM являются архитектуры на основе рекуррентных управляемых блоков (Gated recurrent units, GRU).

В нашей реализации нейронная сеть пытается определить присутствие человека в данный момент времени по 500 предыдущим замерам RSSI. На Рисунке 1 представлена архитектура разработанной нейронной сети. Первый слой — GRU (Gated Recurrent Unit — управляемый рекуррентный блок), принимает на вход вектор размерности 500×1 (это значение было выбрано эмпирически после нескольких запусков). Это значение соответствует оптимальному размеру скользящего окна при предварительной обработке данных (см. Раздел 4.1). В первом слое используется функция активации гиперболического тангенса — $tanh$. Второй полносвязный слой принимает на вход вектор размерности 200×1, используется функция активации

RELU. Третий полносвязный слой принимает на вход вектор размерности 50×1, на выходе получается значение метки от 0 до 1.

Рис. 1: Архитектура разработанной нейронной сети

5. Экспериментальное исследование разработанных алгоритмов

В данном разделе приведено экспериментальное исследование разработанных алгоритмов.

5.1 Цели экспериментального исследования

- исследовать точность реализованных алгоритмов (см. Раздел 3 и Раздел 4.1);
- сравнение точности реализованных алгоритмов с алгоритмами бустинга (используются реализации на языке *Python* из библиотеки $Scikit - learn$): градиентный бустинг ($GradientBoosting$), случайный лес ($RandomForest$), адаптивный бустинг ($AdaBoost$).

5.2 Описание стенда для экспериментального исследования

На практике цель состоит в том, чтобы определить присутствие человека с помощью системы из двух устройств Wi-Fi (приемное устройство — смартфон и точка доступа Wi-Fi). Присутствие определяется на основе показателей RSSI. В качестве дополнительного условия принимается отсутствие значительных возмущений окружающей среды (перемещение габаритных объектов, резкие изменения влажности и температуры). Эксперимент проводился с использованием стенда, состоящего из двух устройств Wi-Fi: мобильного телефона Samsung A7 2018 и

точки доступа TP-Link TL-MR3020 с прошивкой Open WRT операционной системы 19.07.19. Точка доступа была подключена через Ethernet-кабель к ноутбуку. Был написан сценарий на bash, который запускался на точке доступа Wi-Fi и периодически отправлял значения RSSI на ноутбук. Эксперименты проводились в двух помещениях: жилом и офисном. Расстояние в жилом помещении между устройствами составляет 3 м, а в офисном помещении — 5 м. В обоих случаях устройства находились на высоте около 1.5 м.

5.3 Методика экспериментального исследования

Проводились эксперименты двух типов. Эксперимент первого типа состоит из следующих этапов:

- человек отсутствует в помещении в течение 200 с;
- человек возвращается в помещение;
- в первые 50 с человек пересекает линию между устройствами (примерно 5 секунд на пересечение), а также находится сбоку;
- в последующие 150 с человек удаляется из помещения.

Запрос данных RSSI происходит каждую секунду. В рамках этого эксперимента демонстрируется эффективность фильтра Колмогорова-Винера в конкретном помещении.

Эксперимент второго типа состоит из следующих этапов:

- человек выходит из помещения на 5 минут;
- человек возвращается обратно и в течение 5 минут ходит по помещению;
- цикл повторяется.

Эксперимент длится 2100 секунд. Данные RSSI также запрашиваются каждую секунду. Цель этого эксперимента — собрать набор данных для оценки рассмотренных алгоритмов. В результате экспериментов второго типа собран набор данных RSSI с 2100 точками для обоих помещений. Эксперименты проводились в двух помещениях с различным электромагнитным фоном: в жилом и офисном помещениях.

Если необходимо, то производится предварительная обработка данных (см. Раздел 4.1). Для подбора гиперпараметров алгоритмов машинного обучения производится K-fold кросс-валидация на датасете M_{train}, который разделяется на 5 частей ($K = 5$).

5.4 Результаты экспериментального исследования

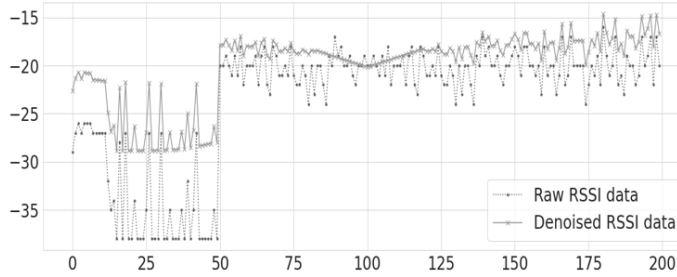

Рис. 2: Результаты выполнения измерений RSSI в жилом помещении

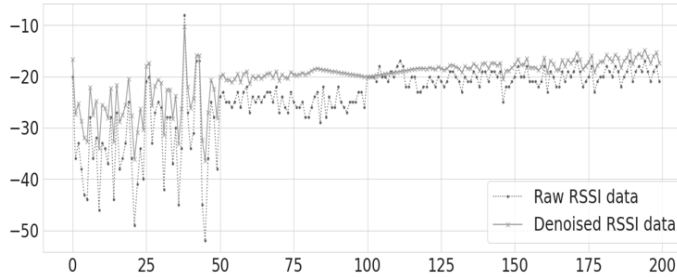

Рис. 3: Результаты выполнения измерений RSSI в офисном помещении

На Рисунке 2 и Рисунке 3 представлены результаты экспериментов первого типа. Представлены данные с показаниями RSSI, которые получены в результате измерений в офисном и жилом помещениях. Представлены данные как до (RSSI до обработки), так и после применения фильтра Колмогорова-Винера (RSSI после обработки), который использовался для удаления шумов. В Таблицах 2-5 используются следующие обозначения:

1. ГБ — градиентный бустинг;
2. СЛ — случайный лес;

3. АБ — адаптивный бустинг;

4. КВГ — применение фильтр Колмогора-Винера, после фильтра Гампеля;

5. НС — нейронная сеть с управляемыми рекуррентными блоками.

Алгоритм	**Точность**	**TPR**	**TNR**
ГБ	0.99	0.99	0.997
СЛ	0.99	0.99	0.994
АБ	0.98	0.98	0.997
КВГ	0.90	1.00	0.83
НС	0.98	0.97	0.99

Таблица 2: Сравнение алгоритмов — жилое помещение

Алгоритм	**Точность**	**TPR**	**TNR**
ГБ	0.99	0.98	0.99
СЛ	0.98	0.98	0.99
АБ	0.97	0.95	0.99
КВГ	0.94	1.0	0.90
НС	0.97	0.97	0.98

Таблица 3: Сравнение алгоритмов — офисное помещение

В Таблице 2 и Таблице 3 представлены результаты экспериментов второго типа: сравнение качества работы алгоритмов в жилом и офисном помещениях. Для сравнения алгоритмов используются следующие метрики качества (все метрики принимают значения от 0 до 1, чем больше значение, тем выше качество):

1. Точность (accuracy) — отношение верно классифицированных точек к общему количеству точек (см. формулу (1));

2. True Positive Rate (TPR) — отношение верных положительных срабатываний к общему числу точек в датасете;

3. True Negative Rate (TNR) — отношение верных отрицательных срабатываний к общему числу точек в датасете.

Алгоритм	Точность	TPR	TNR
ГБ	0.985	0.99	0.97
СЛ	0.99	0.99	0.985
АБ	0.98	0.99	0.96
КВГ	0.75	0.97	0.46
НС	0.86	0.83	0.95

Таблица 4: Сравнение алгоритмов при обучении на данных жилого помещения и проверке на данных офисного

Алгоритм	Точность	TPR	TNR
ГБ	0.88	0.82	0.96
СЛ	0.76	0.99	0.44
АБ	0.76	0.94	0.44
КВГ	0.72	0.94	0.44
НС	0.85	0.81	0.96

Таблица 5: Сравнение алгоритмов при обучении на данных офисного помещения и проверке на данных жилого

Также интересно исследовать вопрос обобщающей способности и “переносимости” рассматриваемых алгоритмов. Для оценки была произведена следующая процедура. В первом случае алгоритм был обучен на данных из жилого помещения, а его качество проверено на данных из офисного. Во втором случае для обучения были использованы данные из жилого помещения, а для проверки его качества — данные из офисного. В обоих случаях использовались данные для экспериментов второго типа. Результаты представлены в Таблице 4 и Таблице 5.

Заключение

Результаты экспериментального исследования показали, что алгоритмы машинного обучения являются более точными, чем алгоритм, основанный на фильтре Колмогорова-Винера, который, кроме того, требует дополнительной предварительной настройки определения уровня шума в помещении. Алгоритмы машинного обучения не требуют специальной настройки уровня шума. Архитектура нейронной сети нуждается в доработке для того,

чтобы превысить точность, достигнутую с помощью градиентного бустинга. Результаты экспериментального исследования и код реализованных алгоритмов находятся в открытом доступе [5].

Wi-Fi сканирование — это очень перспективная область исследований, поскольку аппаратное обеспечение Wi-Fi постоянно совершенствуется. Возможными направлениями будущей работы являются следующие:

1. Разработать новые алгоритмы для обработки данных не только об уровне принимаемого сигнала (RSSI), но и данных с информацией о состоянии канала (Channel State Information, или CSI);
2. Разработать алгоритм для решения проблемы обнаружения человека на основе значений RSSI и CSI для нескольких точек доступа Wi-Fi;
3. Разработать методы интеграции датчиков Wi-Fi в существующие решения Интернета вещей.

Литература

1. Wang Haijing, Zhang Fangfang, Zhang Wenli *Human detection through RSSI processing with packet dropout in wireless sensor network* Journal of Sensors. — 2020. — Vol. 2020. — P. 1–9
2. Sigg Stephan, Blanke Ulf, Troster Gerhard *The telepathic phone: Frictionless Activity Recognition from WIFI-RSSI* 2014 IEEE International Conference on Pervasive Computing and Communications (PerCom). — 2014
3. Santiprapan Phonsit, Sengchuai Kiattisak, Jindapetch Nattha, Saito Hiroshi, Booranawong Apide *Development of an adaptive device-free human detection system for residential lighting load control* Computers & Electrical Engineering. — 2021. — Vol. 93. — P. 107-233
4. Xu Chenren, Firner Bernhard, Moore Robert S., Zhang Yanyong, Trappe Wade, Howard Richard, Zhang Feixiong, An Ning *SCPL* Proceedings of the 12th international conference on Information processing in sensor networks - IPSN '13. — 2013.
5. Датасеты и исходные коды

   `https://github.com/shibaeff/RSSISensing`

Королёв Л. Н.

Формальный анализ выборок

Классификация и кластеризация [1] элементов некоторых множеств, в которых тем или иным способом закодированы свойства реальных объектов предметной области исследований, является важным этапом в решении многих практических задач обработки данных.

Классификацию в общем случае можно определить как задачу построения функции, разбивающей множество на подмножества — на классы близких по каким-то признакам элементов.

Мы будем рассматривать задачу классификации в обычной постановке: задано множество V, каждый элемент которого представляет набор (вектор) признаков, характеризующий данный элемент множества. Известно, что множество V содержит конечное число подмножеств $K1, K2, \ldots, Km$, называемых классами. Заданы также конечные выборки элементов, принадлежащих этим кассам: $E_{k1} \subset K1 \subset V, E_{k2} \subset K2 \subset V, \ldots, E_{km} \subset Km \subset V$. Задача состоит в том, чтобы конструктивно построить функцию, которая каждому элементу множества V, о котором заранее неизвестна его принадлежность к классу, ставит в соответствие имя (или номер) класса, к которому этот элемент принадлежит.

Без дополнительных условий, касающихся объёмов выборок, характеристик свойств множества V и структуры классов, такая неформальная постановка задачи классификации не может дать общих подходов к построению функций, разделяющих множество на классы.

В статье рассматриваются оценки вычислительной сложности некоторых переборных алгоритмов формального анализа выборок, требующих минимального привлечения априорных сведений о структурах классов.

Рассмотрим предельно простой пример, демонстрирующий возможность построения булевой функции, разделяющей множество двоичных слов на классы на основе формального анализа выборок.

Пусть исходное множество V представляет собой множество двоичных слов (векторов) заданной длины, например, $n = 6$. Пусть нам заданы конечные выборки слов, принадлежащих двум классам $K1$ и $K2$:

$$E_{k1} \subset K1 \subset V = \{010111, 111001, 101101, 010001\},$$

$$E_{k2} \subset K2 \subset V = \{111100, 100010, 010110, 101010\}.$$

Рассмотрение содержимого этих двух выборок показывает, что все элементы выборки из класса $K1$ являются нечётными двоичными числами, а элементы выборки из $K2$ — чётными. Это, в свою очередь, позволяет нам высказать интуитивно обоснованное предположение, что весь класс $K1$ состоит только из нечётных чисел, а весь класс $K2$ — только из чётных. Алгоритм решения задачи классификации в этом конкретном случае, если верно наше предположение, очевиден: проверяется младший разряд слова, если он равен 1, то слово относится к классу $K1$, в противном случае — к классу $K2$. Это позволяет построить булеву функцию, разделяющую всё множество V на два непересекающихся класса $K1$ и $K2$. Она совершенно проста: $F(x_1, x_2, x_3, x_4, x_5, x_6) = x_6$.

Заметим, что нашу интуицию можно подтвердить формальными выкладками, пользуясь началами теории вероятности.

Подход к вероятностной оценки выборок, генерируемых в классическом генетическом алгоритме, названных там популяциями, был рассмотрен Холландом [2].

В предположении, что выборки, принадлежащие некоторому классу, сделаны случайно, вероятность того, что в последних разрядах всех элементов выборки E_{K1} случайно оказались все единицы, а в последних разрядах E_{K2} случайно оказались все нули, мала, а при больших объёмах выборок — приближается к нулю.

Пусть в множестве V задано строгое подмножество $E_K \subset V$ и пусть известно, что все его элементы принадлежат одному и только одному классу (другому подмножеству) $K \supset E_K$.

Утверждение о принадлежности элементов выборки только одному классу интуитивно означает, что все элементы выборки в чём-то *близки* между собой. Общая задача классификации состоит в поиске функции, которая причисляла бы любой вектор из V к классу векторов, *близких* к данной выборке. Такого рода постановки встречаются во многих практических задачах кластеризации. По-видимому, найти универсальный алгоритм решения поставленной задачи, пригодный для любого случая, невозможно.

Тем не менее, иногда возможно получить критерии оценки принадлежности элемента к классу, используя только анализ выборок. Для практического решения такой задачи должно быть формально определено понятие близости в каждом конкретном случае.

Рассуждения эксперта, умеющего классифицировать объекты некоторой предметной области, могут быть представлены в виде логического выражения от переменных, которые являются

признаками объекта.

В самом деле, эксперт обычно рассуждает примерно так: «Если у элемента ε признак $\alpha_i < 5$ и при этом признак $\alpha_j \leqslant 0$, а признак $\alpha_k = 0$, то этот элемент относится к классу , в противном случае он относится к другому классу ... и т. д.». Подобного рода рассуждения хорошо формализуются, преобразуясь в некоторое логическое выражение, что позволяет полностью или частично расшифровать способ рассуждений эксперта.

В качестве одного из возможных определений близости, на основе которого для многих реальных задач правомочно и достаточно просто строить алгоритмы классификации, будем использовать следующее: если в некотором множестве для каждого его элемента определено подмножество, которое мы назовём подмножеством близких ему элементов, то такое множество мы будем называть пространством с введённым отношением близости.

Такое определение эквивалентно тому, что любое множество, разделённое любым способом на непересекающиеся подмножества — классы, можно считать пространством с введённым понятием близости. Любую конечнозначную функцию, определённую над элементами множества, значения которой мы будем толковать как метки классов, также можно считать функцией, вводящей понятие близости.

Следует различать отношение близости и понятие расстояния. Расстояние, отвечающее аксиомам классической метрики, легко превращается в пространство с введённым отношением близости. Для каждого элемента множества близкими ему элементами можно считать, например, все элементы, отстоящее от него на расстоянии, меньшее некоторого заданного порога. Отношение близости в этом случае не будет транзитивно.

Отношение $\eta(x, y)$ между двумя элементами множества, принимающее значение 0 или 1, мы и будем считать отношением или функцией близости.

Далее мы будем рассматривать множества, состоящие только из слов признаков одинаковой длины над двоичным алфавитом, как наиболее универсальным. При обработке на вычислительной машине математических моделей реальных объектов и процессов все присущие им признаки закодированы тем или иным способом такими двоичными словами-векторами. Во многих задачах классификации и распознавания образов признаки объектов задаются величинами, т. е. числовыми значениями, но и в этом случае в памяти вычислительной машины они выглядят как структурированная последовательность двоичных значений

ограниченной длины. При обработке реальных задач длина их может достигать нескольких сотен и даже миллионов бит (в задачах анализа изображений). Это следует учитывать при выборе методов обработки подобного типа данных.

Замечание 1, важное для дальнейших рассмотрений.

Множество двоичных слов одинаковой длины — конечно, их полный, упорядоченный в лексикографическом порядке список можно мыслить себе как вход в таблицу истинности (ТИ) некоторой булевой функции, принимающей единичные значения в строках, которые представляют элементы, относящиеся к определённому классу, тем самым класс (или подмножество) индуцирует единственную булеву функцию.

По аналогии с таблицей истинности (ТИ), задающих булеву функцию, введём понятие таблицы меток классов ТМК, которая каждому слову из множества V ставит в соответствие идентификатор класса, к которому это слово принадлежит, например, целое число, обозначающее номер класса.

Каковы бы ни были по сложности реальные критерии отнесения объектов к классу, теоретически эти критерии могут быть реализованы соответствующей расстановкой единиц в ТИ. Если исходное множество разделено на несколько непересекающихся классов, то будет индуцировано несколько булевых функций, которые можно объединить в одну таблицу меток классов.

Многие задачи классификации могут быть сведены к задаче поиска такой расстановки меток классов, которая была бы адекватна ходу рассуждений эксперта, умеющего относить реальные объекты к тому или иному классу.

Если априори ход рассуждений эксперта нам неизвестен, а известны только выборки из классов, тогда эта задача сводится к попыткам расшифровать закодированные в выборках сведения об особенностях классов и характеристик их различий, то есть к расстановке меток в ТМК.

Вернувшись к рассмотренному ранее простому примеру, основываясь на ранее высказанной гипотезе, что все векторы, имеющие 1 в младшем разряде, следует отнести к классу $K1 \subset V$, мы в соответствующих строках ТМК поставим 1 и тем самым определим булеву функцию, выделяющую класс $K1$ элементов, близких по этому критерию. В данном примере векторами *близкими* к выборке E_K будут считаться только векторы, помеченные 1 в таблице (их в данном примере окажется ровно половина конечного множества V).

Любой выборке E_x, можно поставить в соответствие булеву

функцию, принимающую значения 1 в строках таблицы истинности, совпадающих по-битно с элементами выборки E_x. Размечая произвольным образом остальные строки таблицы истинности, мы будем получать различные функции, определяющие классы, в которые входит исходная выборка. Если длина выборки равна k_x, то число различных функций, удовлетворяющих требованию сохранения исходной выборки в составе класса, будет равно: $2^{(2^n - k_x)}$.

Замечание 2.

Мы всегда будем исходить из предположения, что класс не исчерпывается элементами выборки, что выборка составляет некоторую, как правило, небольшую часть класса.

Булева функция, проверяющая принадлежность любого элемента множества V к классу, характеризуемому выборкой EK, в своей таблице истинности должна обязательно содержать единицы в строках, соответствующих элементам этой выборки, и обязана содержать нули в строках таблицы, соответствующих элементам выборок из всех других классов. Следовательно задание выборок, принадлежащих разным классам, позволяет нам частично заполнить таблицы истинности булевых функций, проверяющих принадлежность элементов к классам, из которых взяты выборки.

Число неопределённых строк ТИ конструируемых булевых функций реально оказывается очень большим.

На практике длина двоичного вектора признаков может исчисляться сотнями разрядов. Например, при $n = 100$ число строк таблицы меток классов есть 2^{100}, размеры выборок могут содержать сотни элементов, например $1000 \approx 2^{10}$. Тогда пространство неопределённых строк таблицы, в котором следует найти подходящее размещение меток классов для построения булевых функций, отличающих принадлежность любого вектора к одному классу, составит порядка 2^{90}.

Таблица меток классов (ТМК), если метки классов представлены числами, можно рассматривать как задание некоторой функции табличным методом. Выборки классов, как отмечалось, заполняют только небольшую часть этой таблицы. Попытаемся найти алгоритм, позволяющий дополнять таблицу значений номерами классов, т. е. расширять выборки элементами множества V так, чтобы комплекс расширенных выборок не имел пересечений и был, в некотором смысле, логически обоснованным.

Рассмотренные выше соображения позволяют сформулировать условия, в рамках которых будет рассмотрен алгоритм дополнения ТМК.

Дано пространство V слов $\nu = (x_1, x_2, \ldots, x_n)$ одинаковой длины n над двоичным алфавитом.

- Даны m выборок — $E_1, E_2, \ldots, E_m$, принадлежащих m различным классам: $K_1, K_2, \ldots, K_m$.
- Заданы длины этих выборок — $k_1, k_2, \ldots, k_m$.
- Мощности классов неизвестны, но всегда предполагается, что число элементов в классе превосходит длину выборки, взятой из этого класса.
- Выборки, принадлежащие некоторому классу, формируются случайным образом.

Мы попытаемся найти такой набор переменных $X_{\alpha_1}, X_{\alpha_2}, \ldots, X_{\alpha_k}$, $k < n$, на котором система логических булевых функций $B_1, B_2, \ldots, B_m$, каждая связанная со своим классом, позволяла бы для множества слов, не только принадлежащих выборкам, определять их принадлежность к тому или иному классу. При этом требуется так сконструировать эту систему функций, чтобы минимизировать множество слов, для которых классификация невыполнима.

Конструируемая система булевых функций $B_1, B_2, \ldots, B_m$ должна обладать следующими свойствами: при подстановке любого слова $\nu = (x_1, x_2, \ldots, x_n) \subset V$ во все функции только одна из них должна принять значение единицы, и её номер будет считаться номером класса принадлежности слова. Если при подстановке некоторого слова несколько функций примут одинаковые значения, то такое слово не может быть отнесено ни к какому из классов.

Учитывая определение функции близости, данное выше, введём понятие логически близких элементов по заданной последовательности позиций переменных в векторе признаков.

Пусть нам заданы несколько номеров позиций $\alpha_1, \alpha_2, \ldots, \alpha_k$, $k < n$ в n-разрядных словах исходного множества. Два слова, совпадающих по значениям переменных в этих позициях, мы будем считать логически близкими по данным позициям или частично близкими.

То есть частично близкими будут вектора x и y если: $\eta(x = \{X_{\alpha_1}, X_{\alpha_2}, \ldots, X_{\alpha_k}\}, y = \{Y_{\alpha_1}, Y_{\alpha_2}, \ldots, Y_{\alpha_k}\}) = 1$, только, если $X_{\alpha_i} = Y_{\alpha_i}$ для всех i.

Тем самым последовательность позиций $\alpha_1, \alpha_2, \ldots, \alpha_k$, $k < n$ для каждого слова строит подмножество частично логически близких ему элементов множества V.

Идея предлагаемого далее алгоритма состоит в отыскании такой последовательности $\alpha_1, \alpha_2, \ldots, \alpha_k$, $k < n$, которая удовлетворяла бы требованиям отсутствия пересечений подмножеств, индуцированных элементами выборок различных классов. Это позволит максимально расширить исходные выборки новыми логически близкими элементами. В ряде случаев при этом удастся разбить всё множество на непересекающиеся классы.

Это эквивалентно попытке найти в n-мерном кубе такие гиперплоскости, проекции элементов расширенных выборок на которые были бы компактны, т. е. по-существу попытаться построить систему гиперплоскостей, разделяющих классы [4].

Предлагается следующий переборный алгоритм, удовлетворяющий выше перечисленным требованиям, который состоит в расширении выборок «логически близкими элементами».

Рассмотрим схему предлагаемого алгоритма.

Выполним следующую процедуру.

Произвольно выберем несколько номеров позиций $\alpha_1, \alpha_2, \ldots, \alpha_k$, $k < n$ в n-разрядных словах исходного множества переменных и упорядоченный набор этих позиций обозначим P.

Возьмём j-й элемент выборки из i-го класса и в таблице истинности булевой функции B_i, индуцированной выборкой $E_i \subset K_i$, проставим 1 единицы во всех строках, совпадающих с j-тым элементом в разрядах с номерами $\alpha_1, \alpha_2, \ldots, \alpha_k$. Перебирая один за другим элементы выборки E_i, мы будем добавлять единицы в таблицу истинности B_i и через k_i шагов завершим построение некоторой булевой функции, которая исходную ТИ дополняет единицами, в строках логически близких к элементам выборки по набору $P = \{\alpha_1, \alpha_2, \ldots, \alpha_k\}$. Мы эту функцию назовём функцией расширения исходной выборки по набору переменных и обозначим B_i^P.

Проделав такого рода построения над булевыми функциями из других классов с фиксированным набором , и убедившись в том, что эти расширения не пересекаются, мы можем сделать заключение, что выбранный нами частичный набор переменных, может служить для разделения некоторого подмножества исходного множества V на классы.

ДНФ булевой функции по переменным $X_{\alpha_1}, X_{\alpha_2}, \ldots, X_{\alpha_k}$, связанной с исходной выборкой E_i, мы будем строить как конъюнкцию мономов вида:

$$\bigwedge_{i=1}^{k} x_{\alpha_i}^{\delta_{\alpha_i}^j},$$

соответствующих значениям разрядов с номерами $\alpha_1, \alpha_2, \ldots, \alpha_k$ во всех словах рассматриваемой выборки, где $\delta^j_{\alpha_i}$ степень переменного α_i, в j-той строке исходной выборки.

Итак, задача сводится к отысканию такого набора переменных $X_{\alpha_1}, X_{\alpha_2}, \ldots, X_{\alpha_k}$, который генерировал бы систему булевых функций $B_1^P, B_2^P, \ldots, B_m^P$ с непересекающимися наборами единиц в ТИ. Существуют или нет такие наборы — зависит от состава выборок. Известны примеры, показывающие, что таких функций теоретически построить невозможно.

Верны следующие утверждения.

Если найдены такие номера переменных $\alpha_1, \alpha_2, \ldots, \alpha_k$, что мономы ДНФ функций $B_1, B_2, \ldots, B_m$, построенные по ним, не имеют пересечений и число мономов в каждой функции одинаково и равно r, то заполнение исходной ТМК суммарным количеством меток классов, равно $r2^{n-k+1}$. Отсюда следует, что чем меньше k тем меньше остаётся неопределённых строк в общей таблице меток классов.

Проверка принадлежности слова к некоторому классу будет состоять в подстановке этого слова во все $B_1^P, B_2^P, \ldots, B_m^P$ и выяснении, какая из функций приняла единичное значение. Если ни одна из функций не приняла единичного значения, то принадлежность такого слова к классу не будет определена. Вычислительная сложность такой проверки складывается из поиска строки в ТИ, содержание которой совпадает в заданных разрядах с тестируемым словом, т. е. вычислительная сложность эквивалентна сложности проверки на отсутствие пересечений мономов функций $B_1^P, B_2^P, \ldots, B_m^P$ при анализе пригодности выбранной частичной последовательности переменных $\alpha_1, \alpha_2, \ldots, \alpha_k$.

Рассмотрим пример:

Пусть нам заданы две выборки из двух разных классов следующего содержимого: выборка из первого класса:

(*******110, *******011, *******101, *******110, *******101, *******011);

выборка из второго класса:

(*******100, *******010, *******001, *******010, *******001, *******100).

Символ «*» означает, что конкретное расположение 0 и 1 в этих разрядах в данном примере нам безразлично.

Эти выборки подсказывают гипотезу, что к первому классу относятся все слова, в последних трёх разрядах которых всегда две 1, а второй класс характерен тем, что в последних трёх разрядах

принадлежащих ему слов встречается только по одной 1.

Булевы функции $B_1 = X_2X_1\overline{X}_0 \vee \overline{X}_2X_1X_0 \vee X_2\overline{X}_1X_0$ и $B_2 = X_2\overline{X}_1\overline{X}_0 \vee \overline{X}_2X_1\overline{X}_0 \vee \overline{X}_2\overline{X}_1X_0$ расширяют исходные выборки классов, при этом обеспечивается отсутствие пересечений расширенных классов. В этом примере нам удалось найти две булевых функции, не имеющие совпадающих мономов. Малая вероятность такого случайного события делает правомочной высказанную гипотезу.

Приведённые выше соображения служат обоснованием алгоритма, состоящего в переборе всех возможных частичных булевых функций и проверке того, удовлетворяет ли выбранные функции условиям разделения на непересекающиеся классы.

Схема предлагаемого переборного алгоритма такова:

- перебираем комбинации переменных $X_{\alpha_1}, X_{\alpha_2}, \ldots, X_{\alpha_k}$, $k < n$, начиная с $k = 1$ и для каждой комбинации пытаемся строить систему частичных булевых функций $B_1^P, B_2^P, \ldots, B_m^P$, где $P = \{\alpha_1, \alpha_2, \ldots, \alpha_k\}$, удовлетворяющих требованиям расширения выборок без их пересечений;

- если комбинация не генерирует подходящей системы функций — она отвергается и производится переход к пробам другой комбинации переменных;

- если ни одна из всевозможных комбинаций переменных, при условии $k < n$, не удовлетворяет нашим требованиям, то делается заключение о невозможности построить для этих комбинаций расширяющих функций.

Анализ выбранной комбинации переменных $X_{\alpha_1}, X_{\alpha_2}, \ldots, X_{\alpha_k}$, $k < n$ (столбцов переменных исходной ТИ с номерами $\alpha_1, \alpha_2, \ldots, \alpha_k$) на некотором этапе перебора сводится к следующей последовательности шагов.

1. для каждой выборки E_i выбираем j-тый элемент, принадлежащий рассматриваемому классу, строим ДНФ частичной функции из мономов $x_{\alpha_1}^{\delta_{\alpha_1}^j}, x_{\alpha_2}^{\delta_{\alpha_2}^j}, \ldots, x_{\alpha_k}^{\delta_{\alpha_k}^j}$ для всех $j = 1, k_i$ (k_i — длина выборки).

2. Проверяем полученные мономы на совпадение с аналогично полученными мономами из выборок других классов. Если произошло совпадение одного из мономов с одним из мономов, сконструированного из выборок других классов, то выбранная комбинация переменных с номерами $\alpha_1, \alpha_2, \ldots, \alpha_k$

исключается из рассмотрения, т.к. она не позволят построить на её основе системы функций, обеспечивающих отсутствие пересечений с другими классами, в этом случае переходим к п. 3. Если пересечений мономов не обнаружено, то так построенные частичные булевы функции объявляем пригодными для расширения выборки рассматриваемого класса и на этом заканчиваем перебор.

3. Переходим к выбору новой комбинации номеров строк и снова к п.1.

Теперь можно оценить вычислительную сложность такой переборной задачи.

Максимальное количество различных наборов позиций переменных длины $k < n$ (n) (число слов длины n над двоичным алфавитом, кроме слов из всех единиц и всех нулей) $\phi = 2^n - 2$.

С каждым набором позиций с номерами $\alpha_1, \alpha_2, \ldots, \alpha_k$ необходимо выполнить сравнения с соответствующими подстроками элементов всех других выборок. Максимальное число таких сравнений, следовательно, не превышает $\sum_1^m K_i \sum_{r=1, r\neq i}^m k_r$.

Описанный алгоритм более прозрачно выглядит при поиске расширения выборок для двух классов, к которому сводится разделение на несколько классов.

Рассмотрим более подробно этот алгоритм в ситуации, когда заданы две выборки E_{K1} и E_{K2} из двух непересекающихся классов $K1$ и $K2$, при чём $K1 \cup K2 = V$.

Пусть размер выборки E_{K1} есть m_{K1}, а размер выборки E_{K2} есть m_{K2}.

Цикл по $j = 1, m_{K1}$:

Строим ДНФ частичной функции, состоящей из одной переменной $_i$ для выборки E_{K1}. Различных мономов в такой ДНФ будет не более 2. Если в этой ДНФ оказался только один моном, то следует проверить, не совпадает ли он с мономом ДНФ, построенным таким же образом для выборки E_{K2}. Если совпадений не произошло, то полученные ДНФ могут служить в качестве функций разделителей классов (См. пример 1). Число проб по i в этом случае равно n — числу букв в слове.

Таким же образом строим ДНФ частичных функций, состоящих из j переменных $X_{\alpha_1}, X_{\alpha_2}, \ldots, X_{\alpha_j}$ для выборки E_{K1} и выборки E_{K2}. Таких ДНФ будет построено и проверено на отсутствие совпадений не более C_n^j. Если среди построенных функций не нашлось приемлемых, то переходим к проверке частичных функций для $j + 1$. Соответственно, общее число проверяемых функций и проверок их на пригодность может составить 2^n.

Практически алгоритм состоит в следующем:

Выбирается подпоследовательность номеров переменных $\alpha_1, \alpha_2, \ldots, \alpha_r$, $r < n$;

Из выборки E_{K1} для каждого её элемента j выделяется соответствующая подстрока значений $X^j_{\alpha_1}, X^j_{\alpha_2}, \ldots, X^j_{\alpha_j}$;

Выбранная подстрока значений сравнивается со всеми аналогично полученными подстроками выборки E_{K2};

Если произошло совпадение хотя бы с одной из подстрок выборки, то весь набор позиций переменных $\alpha_1, \alpha_2, \ldots, \alpha_r$ отметается и в дальнейших проверках не используется;

Если совпадений не произошло, переходим к анализу подстроки элемента выборки E_{K1} с номером $j + 1$.

Если для всех j совпадений не произошло, то набор позиций объявляется пригодным для расширения выборки E_{K1} и построения булевой функции, выделяющей класс $K1$.

Любое слово множества V, подслово которого совпадает с одним из подсловом выборки E_{K1} объявляется принадлежащим классу $K1$.

Над выборкой E_{K2} производится такая же процедура поиск подпоследовательности позиций переменных, на которой достигается эффект отсутствия пересечений с подсловами выборки E_{K1}.

Подпоследовательности «удачных» позиций могут быть таковыми, что не каждое слово множества V найдёт свой класс.

Максимальное число сравнений $s(p)$, которое может понадобиться при осуществлении этого алгоритма над заданной последовательностью позиций равно $r_1^* r_2^*$, где r_1^* и r_2^* — число различных частичных подслов в выборках E_{K1} и E_{K2} инициированных подпоследовательностью $\alpha_1, \alpha_2, \ldots, \alpha_r$, $r < n$. При этом всегда $r_i^* \leqslant 2^r$, и $s() \leqslant 2^{r+1}$.

Общее число наборов $\alpha_1, \alpha_2, \ldots, \alpha_r$, $r < n$ для r от 1 до $n - 1$ есть $2^n - 2$, соответственно максимальное число сравнений не превосходит $\sum_1^{n-1} 2^{r+1} = 2^n - 4$. При этом следует учесть, что сравнения оканчиваются как только произошло первое совпадение подслова с элементом чужой выборки поэтому реально число сравнений оказывается значительно меньшим чем данная величина.

Современный суперкомпьютер, обладающий производительностью в $10^{15} \approx (2^{10})^{15} = 2^{150}$, может справиться с задачей анализа выборок 150-ти разрядных слов.

Изложенный выше алгоритм позволяет строить расширенную таблицу меток классов, которую можно рассматривать как табличное задание некоторой целочисленной функции.

Поскольку нейронные сети являются универсальным инструментом аппроксимации функций [4], можно надеяться, что их применение позволит построить функцию-классификатор, основанную на использовании приведённого алгоритма, основанного на введённом понятии логической близости, как средства предварительной обработки выборок.

Введём понятие структурной близости элементов множества векторов признаков, построенных на основе двоичного алфавита. Это определение близости легко трансформируется на множества, состоящие из слов, построенных над любым конечным алфавитом.

Шаблоном [7] мы назовём двоичную последовательность длины n, где n число букв в элементах рассматриваемого множества V. Подсловом данного слова по шаблону мы назовём последовательность букв рассматриваемого слова, выбранных из позиций, соответствующих единицам шаблона.

Пусть исходное слово $s = (x_1, x_2, \ldots, x_n)$, пусть единицы шаблона расположены в разрядах с номерами $\alpha_1, \alpha_2, \ldots, \alpha_r$, тогда подслово, выбранное по данному шаблону будет представлять собой последовательность, букв длины r вида: $X_{\alpha_1}, X_{\alpha_2}, \ldots, X_{\alpha_r}$.

Например: пусть n равно 7, пусть задан шаблон 0100111 и двоичное слово 1100110.

Подсловом данного слова по этому шаблону будет следующая последовательность букв исходного слова 1110.

Длинной шаблона мы будем считать число единиц в нём. Соответственно, длиной подслова мы будем считать число выбранных по шаблону букв.

Определим степень близости между двумя словами как максимальную длину полностью совпадающих их подслов.

Это означает следующее. Для того чтобы определить степень близости двух слов $s_1 = (x_1, x_2, \ldots, x_n)$ и $s_2 = (y_1, y_2, \ldots, y_n)$ нам надо найти шаблон для s_1 и другой шаблон для s_2, одинаковые по длине, такие, чтобы подслова, выбранные по этим шаблонам, совпадали и были бы максимальными по длине. Такое определение близости эквивалентно тому, что нам следует найти минимальное число вычёркиваний букв из s_1 и s_2 так, чтобы полученные подслова были бы одинаковы.

При таком общем определении степени близости её нахождение сводится к переборной задаче опробования всех возможных шаблонов как для слова s_1 так и для слова s_2. Порядок вычислительной сложности при этом оценивается величиной 2^{n+1} [5].

Во многих практических задачах классификации степень

структурной близости регулируется условиями, налагаемыми на шаблоны. Если, например, потребовать одинаковости шаблонов для s_1 и s_2, то степень близости будет величиной противоположной расстоянию по Шеннону.

Например: для двух слов 1001101 и 1110010 максимальное совпадающее подслово будет выглядеть так: $1 * * * * * *$, т. е. близость величина противоположная расстоянию по Шеннону (число совпадающих букв).

Можно потребовать, чтобы шаблоны различались между собой циклическим сдвигом. В задачах поиска совпадающих участков буквенных последовательностей такое требование вполне оправдано.

Если при этом рассматривать шаблоны, состоящие из одной серии единиц, то это подойдёт к задаче поиска вхождений заданных слов.

Выбрав критерии определения структурной близости [6] можно на этом основании применить описанный выше алгоритм «расширения выборок» близкими элементами для дополнения таблицы меток классов. По вычислительной сложности эти алгоритмы будут близки.

Литература

1. Ту, Дж., Гонсалес, Р. *Принципы распознавания образов: Пер. с англ.* М.: Мир, 1978.

2. Holland J. *Adaptation in natural and artificial systems.* MIT press, Cambridge MA, 1992

3. Forrest S. *Genetic Algorithm: Principles of Natural Selection Applied to Computation.* J. Sciense. V. 261, 872-878, August 1993.

4. С. Осовский. *Нейронные сети для обработки информации.* М.: «Финансы и статистика», 2002.

5. Сачков В. Н. *Комбинаторные методы дискретной математики.* М., 1977.

6. Иванов С.А. *Методы поиска закономерностей в символьных последовательностях.* Программные системы и инструменты. Тематический сборник № 8, М.: Изд-во факультета ВМиК МГУ, 2007.

7. Magnusson M.S. *Discovering Hidden Time Patterns in Behavior: T-Patterns and their Detection.* Behavior Research Methods, Instruments and Computers. 2002. 32. N 1. P. 93-100.

Аннотации

Pluzhnikova D.R., Antonenko V.A. BORIS: Prototype of BGP routing system based on blockchain technology // Программные системы и инструменты. Тематический сборник № 22, М.: Изд-во факультета ВМК МГУ, 2022.

The BGP protocol, which is used for global routing, is insecure because it does not have authentication mechanisms and authentication of the update source. In this regard, many BGP security solutions have been developed, but each of them depends on a central trusted node. This work aims to develop and implement a prototype of the dynamic routing protocol BGP based on blockchain technology, which allows you to build a secure decentralized system. The article presents an experimental study of the prototype, which shows that, despite the increase in operating time, the prototype increases the security of the BGP protocol.

Ил.: 12 рис., Библиогр.: 13.

Абрамов А.В., Чупахин А.А. Построение однопроцессорного расписания с минимизацией пикового использования ресурса при помощи муравьиного алгоритма // Программные системы и инструменты. Тематический сборник № 22, М.: Изд-во факультета ВМК МГУ, 2022.

В данной работе рассматривается задача составления расписания с минимизацией пикового потребления ресурса. В работе проведён обзор предметной области, разработан муравьиный алгоритм (МА) для решения поставленной задачи, рассмотрены две модификации алгоритма (локальный поиск и подход Ant-Q, основанный на обучении с подкреплением), а также проведено сравнение реализованного МА с алгоритмом имитации отжига (ИО).

Ил.: 2 рис., 5 табл. Библиогр.: 10.

Бахмуров А.Г. Опыт преподавания курса "Имитационное

моделирование в исследовании и разработке вычислительных систем"и планы по развитию курса // Программные системы и инструменты. Тематический сборник № 22, М.: Изд-во факультета ВМК МГУ, 2022.

Курс читается на кафедре АСВК факультета ВМК для студентов 3 курса, с 2014 г., в 5 семестре. В статье обоснована необходимость создания курса, кратко изложено и обосновано его содержание. Материалы курса размещены на вебстранице https://asvk.cs.msu.su/education/simulation.

Библиогр.: 8.

Бодров А.О., Бахмуров А.Г. Масштабирование приложения с помощью средств docker swarm и HAproxy // Программные системы и инструменты. Тематический сборник № 22, М.: Изд-во факультета ВМК МГУ, 2022.

Приложения для сбора и обработки данных, например, в сетях Интернета вещей, могут собирать данные с большого количества (сотни, тысячи) клиентов одновременно. Во избежание роста задержек на обработку данных и их потерь, необходима масштабируемость приложения, т.е. увеличение его производительности (при росте числа клиентов) пропорционально дополнительным ресурсам (оборудованию), причём автоматическое. В настоящей статье авторы рассмотрели масштабирование на примере разрабатываемого на факультете ВМК МГУ сервиса сбора медицинских данных. Однако, описанный подход и архитектура могут быть применены и для других приложений.

Ил.: 8 рис. Библиогр.: 13.

Гуров С.И., Смелянский Р.Л., Ержанов Ж.Р. Сетевое кодирование в троичных полях // Программные системы и инструменты. Тематический сборник № 22, М.: Изд-во факультета ВМК МГУ, 2022.

В статье предложен новый метод сетевого кодирования, на основе перехода от двоичного к троичному представлению данных. Предложенный метод позволяет сократить задержки при передаче данных, поскольку отпадает необходимость агрегировать один и тот же кадр с несколькими другими кадрами в разные пакеты. Также становится не нужным обеспечивать асинхронную доставку двух или более агрегированных кадров в одну и ту же точку сети для выделения исходного кадра. Кроме того, предложенный метод устраняет влияние блокировок на выходе коммутатора на его

пропускную способность, позволяя сократить повторную передачу данных из-за ошибок передачи.

Ил.: 1 табл. Библиогр.: 10.

Казантаев А.Д, Чупахин А.А. Алгоритм балансировки нагрузки в распределенной вычислительной системе с применением мультиагентного обучения с подкреплением // Программные системы и инструменты. Тематический сборник № 22, М.: Изд-во факультета ВМК МГУ, 2022.

В данной работе приведены результаты исследования механизмов балансировки нагрузки в неоднородной распределенной вычислительной системе (РВС) и предложена формулировка задачи балансировки в терминах стохастической игры. Неоднородная РВС рассматривается как мультиагентная система, где каждому вычислительному устройству (ВУ) в РВС можно сопоставить обучающего агента. Исходя из информации о загруженности используемых ресурсов (процессор, память) на всех ВУ, каждый агент принимает решение: выполнять задание локально или отправить его выполняться к другому агенту, чтобы равномерно распределить задания по всем ВУ системы. Для решения стохастической игры разработана мультиагентная система обучения с подкреплением с использованием алгоритма *PPO* (*Proximal Policy Optimization*). Результаты экспериментов демонстрируют, что предложенный алгоритм применим для решения сформулированной задачи и является устойчивым относительно различных сценариев изменений конфигурации вычислительной системы.

Ил.: 5 рис., 3 табл. Библиогр.: 19.

Коваленко А.П., Сальников А.Н. Разработка метода трехмерной визуализации задержек в коммуникационной среде кластера // Программные системы и инструменты. Тематический сборник № 22, М.: Изд-во факультета ВМК МГУ, 2022.

В современном мире многие значимые и вычислительно сложные задачи решаются не на отдельных компьютерах, а на вычислительных кластерах. В рамках данной работы были рассмотрены существующие методы визуализации графов и был предложен алгоритм визуализации задержек в коммуникационной среде кластера.

Ил.: 4 табл. Библиогр.: 5.

Порывай М.В., Чупахин А.А. Исследование алгоритмов решения задачи нерегулярного размещения плоских

геометрических фигур // Программные системы и инструменты. Тематический сборник № 22, М.: Изд-во факультета ВМК МГУ, 2022.

В работе рассматривались двумерная нерегулярная задача раскроя и способы ее решения. Был проведен аналитический обзор существующих алгоритмов решения нерегулярной двумерной задачи раскроя, разработана математическая постановка задачи. Также были разработаны и реализованы на языке программирования *Python* жадный алгоритм и алгоритм имитации отжига, решающие задачу в предложенной постановке. Проведены экспериментальное исследование свойств разработанных алгоритмов и сравнение результатов их работы на основе разработанной методики.

Ил.: 3 рис., Библиогр.: 11.

Шибаев П.П., Чупахин А.А. Применение фильтра Колмогорова-Винера и нейронных сетей с управляемым рекуррентными блоками для решения задачи Wi-Fi-сканирования // Программные системы и инструменты. Тематический сборник № 22, М.: Изд-во факультета ВМК МГУ, 2022.

Wi-Fi-сканирование основано на использовании Wi-Fi-устройств для распознавания активности людей путём определения отклонений и особенностей в различных характеристиках Wi-Fi-сигнала. Данная технология особенно актуальна в преддверии нового стандарта Wi-Fi, в фокусе которого находится активное развитие Wi-Fi сетей для интернета вещей. В работе было показано, что алгоритмы машинного обучения являются более точными, чем алгоритм, основанный на фильтре Колмогорова-Винера для решения задачи Wi-Fi-сканирования.

Ил.: 3 рис., 5 табл. Библиогр.: 5.

Королёв Л.Н. Формальный анализ выборок // Программные системы и инструменты. Тематический сборник № 22, М.: Изд-во факультета ВМК МГУ, 2022.

Классификация и кластеризация элементов некоторых множеств, в которых тем или иным способом закодированы свойства реальных объектов предметной области исследований, является важным этапом в решении многих практических задач обработки данных. В статье рассматриваются оценки вычислительной сложности некоторых переборных алгоритмов формального анализа выборок, требующих минимального привлечения априорных сведений о структурах классов.

Библиогр.: 7.

О четвёртой международной научной конференции "Современные сетевые технологии"(Monetec-2022)

Четвёртая международная научно-техническая конференция «Современные сетевые технологии» (Monetec-2022) прошла 27-28 октября 2022 года в МТУСИ (Москва). Международная научно-техническая конференция «Modern Network Technologies, MoNeTec-2022» собрала представителей международного и российского научного сообщества, исследовательских подразделений компаний для обсуждения перспективных и актуальных технологий в сфере компьютерных сетей, виртуализации сетевых сервисов и облачных вычислений.

В качестве приглашённых докладчиков выступили:

- Руслан Леонидович Смелянский, (МГУ, Россия) - «Network Powered by Computing»
- Юрий Гугель (Национальный исследовательский центр "Курчатовский институт Россия) - «The Russian Segment (RU-VRF) in WLCG Infrastructure: High Performance Computing Network»
- Владимир Воеводин (МГУ, Россия) – «Supercomputer Systems

and Structural Features of Algorithms»

- Дмитрий Лаконцев, (Центр компетенций НТИ на базе Сколтеха, Россия) - «The Challenges of Building Algorithm Stack for a 5G Open RAN Base Station»
- Александр Лазарев (МГУ, Россия) – «A Metric Approach to Solving Problems in the Scheduling Theory»
- Ричард Брукс (Clemson University, США) - «DLT Design and Robustness Verification Based on Random Graph Theory»
- Прадибан Катхиравелу (Emory University, USA) - "Uniform Interfaces for Healthcare Data Access Federation"

Также было заслушано более 20 секционных докладов по основным направлениям:

- Data Communication Infrastructure;
- QoS control in Data Communication;
- Cloud Computing;
- Optimization Methods, Tools, and Technologies in Cloud Computing;
- Network Function Virtualization and Services;
- Edge Computing;
- 5/6G Wireless Technologies, Applications and Services;
- Future Networking.

Итоги конференции подвели на церемонии закрытия и вручения наград авторам лучших работ. Жюри были отмечены исследования:

- 1 место — «Алгоритмы топологии наводнения для компьютерных сетей», Михаил Шуплецов, Дмитрий Романов, Евгений Степанов.
- 2 место — «Управляемые сети для решения проблемы транспортных потоков», Елена Софронова и Асхат Дивеев.
- 3 место — «Многоцелевое моделирование вычисления плоскости маршрутизации на основе отжига для различных требований», Максим Приходько, Longfei Dai, Degao Zheng, Zhaoyu Jiang, Zhibo Hu, Yi Zhang, Yanmiao Wang.

Приглашаем Вас к участию в конференции Monetec-2024.

УДК 519.6+517.958
ББК 22.19
П75

Software systems and tools: Thematic collection / Edited by Smelianskii
П75 R.L. – M: Publishing Department of the Faculty of CMC Lomonosov Moscow State University (license ID №05899 from 24.09.2001y.;MAKS Press, 2022, №22. - 160 p.

ISBN 978-5-89407-629-4 (CMC MSU)
ISBN 978-5-317-06895-0 (MAKS Press)
https://doi.org/10.29003/m3106.978-5-89407-610-2

This volume contains student works that were recommended for publication by the scientific seminar of Automation of Computer Systems chair. The editorial board continues the publishing tradition in memory of L.N. Korolev. The proposed thematic collection contains papers on the actual problems of computer networks, methods of computations scheduling and load balancing, discrete optimization problems and algorithms, methods of information encoding and visualization.

These papers are of interest to students, graduate students and professionals in the development of applied software systems.

Ключевые слова: Информационно-телекоммуникационные технологии, машинное обучение, нейронные сети, блокчейн, программно-конфигурируемые сети, сетевое кодирование, планирование вычислений, балансировка нагрузки, облачные вычисления, эволюционные алгоритмы, имитационное моделирование, сбор медицинской информации, Интернет вещей, мультиагентные системы, визуализация сетевой среды, анализ выборок..

Научное издание

ПРОГРАММНЫЕ СИСТЕМЫ И ИНСТРУМЕНТЫ

Тематический сборник

№ 22

Под общей редакцией чл.-корр. РАН,
профессора Р. Л. Смелянского

Издательство «МАКС Пресс»
Главный редактор: *Е. М. Бугачева*

Напечатано с готового оригинал-макета
Подписано в печать 20.10.2022 г.
Формат 60x90 1/16. Усл.печ.л. 10.
Тираж 40 экз. Заказ 196.

Издательство ООО “МАКС Пресс”
Лицензия ИД N 00510 от 01.12.99 г.
119992, ГСП-2, Москва, Ленинские горы, МГУ им. М.В. Ломоносова,
2-й учебный корпус, 527 к.
Тел. 8(495)939-3890/91. Тел./Факс 8(495)939-3891.

Отпечатано в полном соответствии с качеством
предоставленных материалов в ООО «Фотоэксперт»
109316, г. Москва, Волгоградский проспект, д. 42,
корп. 5, эт. 1, пом. I, ком. 6.3-23Н