МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
имени М. В. ЛОМОНОСОВА

Факультет вычислительной математики и кибернетики

ПРОГРАММНЫЕ СИСТЕМЫ И ИНСТРУМЕНТЫ

Тематический сборник

№ 20

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

МАКС ПРЕСС

Москва — 2020

УДК 519.6+517.958
ББК 22.19
П75

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

Программные системы и инструменты: Тематический сборник / П75 Под ред. Р. Л. Смелянского. — Москва : Издательский отдел факультета ВМК МГУ имени М. В. Ломоносова (лицензия ИД № 05899 от 24.09. 2001 г.); МАКС Пресс, 2020. — № 20. — 188 с.

ISBN 978-5-89407-618-8 (ВМК МГУ имени М. В. Ломоносова)
ISBN 978-5-317-06545-4 (МАКС Пресс)

Данный выпуск сборника составлен по материалам работ студентов и аспирантов, получивших рекомендации научных семинаров кафедры автоматизации систем вычислительных комплексов, кафедры алгоритмических языков, а также стендовых докладов международной конференции “Современные сетевые технологии”. Редколлегия сборника продолжает традицию его издания в память о первом руководителе кафедры АСВК Л. Н. Королёве. В предлагаемом читателю тематическом сборнике публикуются статьи, посвященные проблемам современных компьютерных сетей, методам управления облачными вычислениями, инструментальным средствам, обрабатывающим текст на русском языке и методологическим проблемам преподавания программирования.

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

Ключевые слова: информационно-телекоммуникационные технологии, программно-конфигурируемые сети, OpenFlow-коммутатор, централизованный контроллер, облачные вычисления, виртуальные ресурсы, виртуализация сетевых функций, облачная платформа, сетевое процессорное устройство, холодный старт, прогноз популярности, кэширование, бессерверные вычисления, платформа для медицинских исследований, формальная семантика, сетевое кодирование, поля Галуа, трит, эластичные вычисления, легковесная виртуализация, обработка текста, информационно-ориентированные сети, реконфигурация, Интернет вещей, эволюционные алгоритмы, нейросетевые вычисления, генетическое программирование.

УДК 519.6+517.958
ББК 22.19

ISBN 978-5-89407-618-8
ISBN 978-5-317-06545-4

Оглавление

От редколлегии **5**

1 *Kukushkin D.I., Antonenko V.A.* **Daddy: Data Dependancy Graphs in Serverless Computations** **6**

2 *Zaitseva O.A., Antonenko V.A.* **CLEVER: Cold Start for Serverless Applications** **21**

3 *Аникевич Ю.В., Бахмуров А.Г., Чайчиц Д.А.* **Разработка и реализация платформы для сбора и обработки физиологических данных о человеке** **36**

4 *Баула В.Г.* **Формальная семантика в обучении программированию** **49**

5 *Ержанов Ж.К., Смелянский Р.Л.* **Сетевое кодирование на основе** $GF(3^m)$ **59**

6 *Иванов И.В., Антоненко В.А.* **Разработка архитектуры эластичного приложения в среде легковесной виртуализации** **68**

7 *Кузьмин Я.К., Волканов Д.Ю., Скобцова Ю.А.* **Исследование применимости алгоритмов обработки пакетов с сохранением состояния в архитектуре сетевого процессорного устройства RuNPU** **82**

8 *Машонский И.Д., Большакова Е.И.* **Инструментальные средства извлечения терминов из текстов: разработка компонентов для русского языка** **94**

9 *Миков А.И., Миков А.А.* **Устойчивость к возмущениям масштабируемых мобильных информационно-ориентированных сетей** **106**

10 *Никифоров Н.И., Волканов Д.Ю., Скобцова Ю.А.* **Анализ и исследование структур данных для поиска в таблицах классификации в сетевом процессорном устройстве с архитектурой RuNPU** **118**

11 *Шапошников В.А., Писковский В.О.* **Постановка задачи и краткий обзор решений для проблемы управления процессами согласованной реконфигурации ПКС** **133**

12 *Шмитов Н.О., Пашков В.Н.* **Метод управления устройствами интернета вещей и обменом данных между ними в программно-конфигурируемых сетях** **142**

13 *Королев Л.Н.* **Об эволюционных алгоритмах, нейросетевых вычислениях, генетическом программировании – математические проблемы** **156**

Аннотации **174**

О третьей международной научной конференции "Современные сетевые технологии"(Monetec-2020) **180**

Школа по сетевым и облачным технологиям Monetec-2020 **182**

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 научных семинаров кафедры Автоматизации Систем Вычислительных Комплексов, которую Л.Н. Королев создал и бессменно руководил, кафедры Алгоритмических Языков, а также стендовых докладов международной конференции “Современные сетевые технологии”.

Также в этом сборнике представлена одна из работ Льва Николаевича Королёва «Об эволюционных алгоритмах, нейросетевых вычислениях, генетическом программировании – математические проблемы».

Редколлегия

Kukushkin D.I., Antonenko V.A.

Daddy: Data Dependancy Graphs in Serverless Computations

1. Introduction

Serverless computing are computations performed with the help of containerized functions on a remote server. Functions placed in a container are called *serverless functions.*

Function as a Service (FaaS) is a model of commercial use of serverless computing, in which the servers where serverless functions are executed are maintained and configured by the service provider. A user gets an opportunity to run serverless functions written by him on remote hardware and pays for the resources spent on the execution of the functions launched by him.

Serverless computing allows to structure and flexibly scale the program, breaking it into separate functions. However, the task of combining serverless functions into a program carries some difficulties [1]. In particular, serverless trilemma [2] describes the main problems appearing when implementing composite serverless functions, such as *double payment*, *black box restriction*, *substitution principle* and *necessity to use client to execute composite functions.*

In this work, we investigate the problem of reducing data exchanges during execution of a composite serverless function. We suppose this will reduce composite serverless function execution time. By a composite serverless function we mean a function that calls other serverless functions in the user-specified order. No changes are made to the text of the called functions, thus the principle of the black box is not violated. When organizing the execution of a serverless function by traditional methods, data exchanges are conducted outside the service provider hardware. In addition, due to the need to maintain an intermediate result, their number increases at least twice. This leads to unnecessary exchanges and delays in their execution, since the network speed inside a computing cluster and from cluster to user is different. Moreover, there is the problem of excessive data transfer when performing group data exchanges in parallel serverless computing [1, 3, 4]. For example, the "shuffle"template, requires K^2 more operations when using serverless functions compared to virtual machines. However, it is possible to reduce the number of operations by several times by combining data exchanges into groups [4].

2. Related works

In this section, we discuss various works that deal with the problems appearing in systems allowing to execute composite functions.

AWS Step Functions [5] is an Amazon service allowing to describe workflows using the state machine and execute them. The state machine consists of a number of steps, each step receives input data, invokes a serverless function and returns the result. The Json format is used to describe the state machine and input data. The system supports parallel execution of steps and conditions.

As was mentioned in the Introduction **the serverless trilemma** [2] describes problems that face most of the systems implementing composite functions. This work investigates these problems and present a solution to avoid these limitations. The solution is based on the use of OpenWhisk triggers and the Kafka message queue service.

In **Formal Foundations of Serverless Computing** [7] arguments are given for the importance of usage of composite functions. The authors have developed a language that allows to describe compound functions mathematically. Based on this language and the OpenWhisk system, a prototype allowing to implement conditional structures and loops is built. The experimental studies prove the expressiveness of this language.

3. Our solution

Our main idea is to present a composite function as a workflow and to describe this workflow by a dependency graph. The dependency graph is then passed to the scheduler, which invokes the necessary functions with the corresponding input data. By *task scheduler* we mean a system that controls and manages the distribution of tasks between computing nodes in order to optimize certain parameters such as execution time. Works specializing in linear algebra calculations [8, 9] use dependency graph in the way similar to the described above. We suppose that with the help of dependency graph we can reduce the number of costly data exchanges by moving those interaction to the providers network thus excluding the user from them.

Workflow is a sequence of operations that processes a data set. There is a *data dependency* [10] between operation B and operation A if operation A modifies or creates the data used by operation B. In other words B *depends* on A. This dependency requires operation A to be performed before operation B.

Data dependency graph is a directed acyclic graph in which the nodes correspond to the instructions while the edges reflect the dependencies

between the instructions in the direction of the dependent instruction (see figure 1).

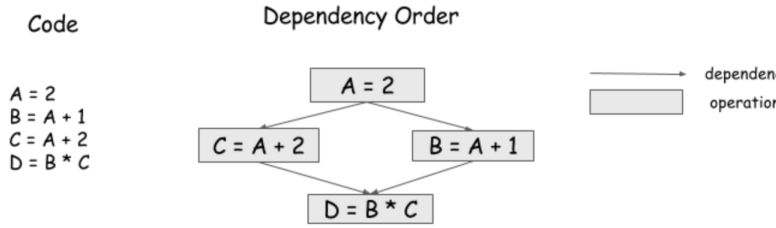

Figure 1. Data dependency graph

4. Implemetation description

We chose OpenFaaS with provider based on Kubernetes as the platform for implementing the prototype. OpenFaaS is a popular open source serverless computing platform. Meanwhile Kubernetes provides flexibility in setting up resource allocation, thanks to the ability to extent API with custom resource definition(CRD).

This section describes OpenFaas tools used to invoke serverless functions.

4.1 OpenFaaS

OpenFaas consists of modules:

1. **UI** - client application, for example faas-cli, sending commands to invoke, delete and deploy serverless functions
2. **Gateway** - a backend application, processing UI commands and also controlling resosurces.
3. **Faas-provider** - a backend application, creating, updating and deleting function images. It also maintains environment necessary for their invocation.
4. **Prometheus, AlertManager** - maintains or scales up the number of function images ready to be invoked.
5. **NATS** - process queues of async functions.

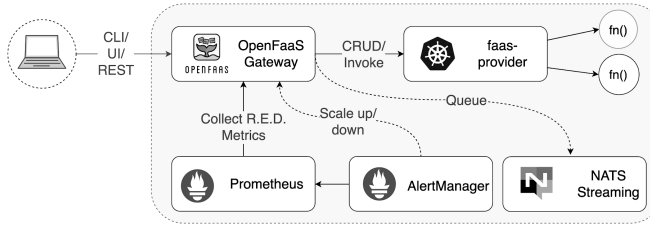

Figure 2. OpenFaaS

4.2 Function invocation

In this subsection we discuss the interaction of OpenFaaS modules when a serverless function is invoked with the help of faas-cli. The following actions take place:

- Faas-cli application receives input data, and then forms and sends a http request to the gateway module.
- The server part of the gateway module receives a request and redirects it to the function invocation handler.
- The handler generates a new http request using the **forwardRequest()** method, and with the help of the client part of the gateway module, depending on the "direct functions-setting, it either calls the function directly or passes the request to the faas-provider module.
- If the request was passed to the faas-provider module, then this module generates another http request and invokes the function with the help of it.
- After the function has been executed, the result of its operation or an error returns to the user via a http response series.

The "direct functions"setting is enabled by default, this means that the gateway module calls functions directly. The creators of OpenFaaS do not recommend turning it off, as otherwise extra http requests will be made.

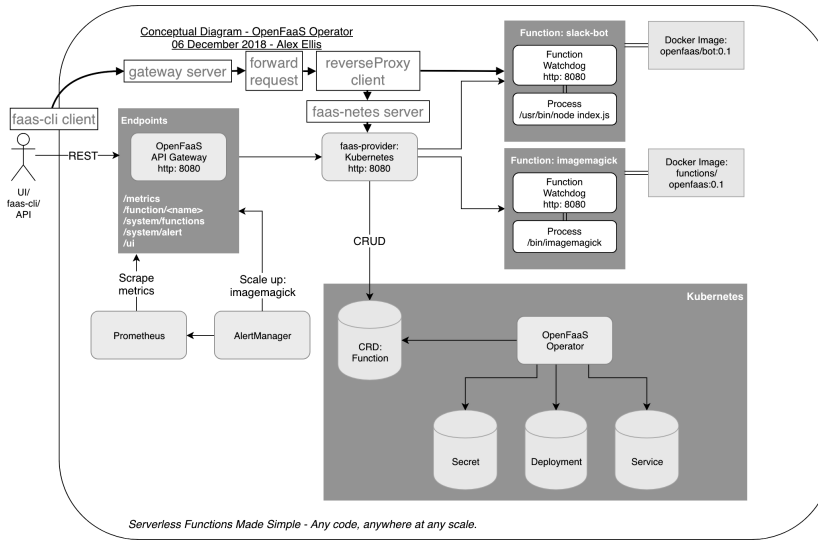

Figure 3. Serverless function invokation in OpenFaaS

4.3 Our contribution

Our prototype is implemented in gateway module and allows to execute serverless functions consisting of stages. Each stage receives a string as input, and passes its result to the next one. The user can insert additional data into the input string before the start of each stage. A stage consists of one or more serverless functions performed in parallel. All functions within a single stage receive the same input data, and their output data is combined into the result of the stage.

We use headers of an http request to pass a dependency graph and input data to a composite function. When such function is invoked in the faas-cli application, it is necessary to indicate that the function is composite with the help of the flag: `--header Workflow=YES`. The dependency graph is passed with the following flag: `--headerGraph=GraphNodes`. Instead of `GraphNodes`, there should be a string containing the names of functions. These functions must have been already deployed. Their names are separated from each other with the symbol `"_"`for sequential execution and with `"`for parallel. Additional parameters are sent via a string before the function name and `":"`. For example, the construction

`_parameters:func-name` means that when the function "func-name"is called, in addition to the main parameters, it will receive the string "parameters". The stage consisting of the function that a user invokes is always executed first, its result is passed to the next stage. An example of a workflow call starting with the *sleep* function is shown in figure 4. The figure 5 shows how it works. The following functions are used in the example:

- sleep - waits a fixed timeout. It is described later in section 5.1.
- text - just returns the input string. It is also described later in section 5.1.
- nodeinfo - is a function from OpenFaaS store that shows system configuration.
- figlet - is a function from OpenFaaS store that converts input data into ASCII graphics.

```
faas-cli invoke sleep --header Workflow=YES --header Graph=nodeinfo_test,text,text,text_figlet
```

Figure 4. Workflow example

This example works as follows: the *sleep* function waits for a specified period of time, *nodeinfo* displays information about the system, *text* duplicates this information 4 times, and then *figlet* converts the result to ASCII graphics.

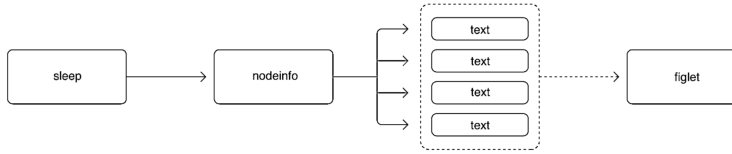

Figure 5. Workflow example

4.4 Composite function processing

The dependency graph is processed in **forwardRequest()** function. This function creates and executes a http request that invokes a serverless function. This function checks the "Workflow"flag. If it is present, it ejects the dependency graph from the request header and processes it. For invocation of each serverless function that is a part of a composite function a new http request is created. A **startFunction()** which invokes such a function and receives its result is called in a separate thread for every serverless function in the parallel block. After the dependency graph has been processed, the result is returned to the user.

5. Experimental Research

The purpose of the experimental research is to evaluate the ability of the prototype to execute composite functions and compare the results with traditional methods of execution of composite functions. We divide our research into two parts: performance tests and a real task test. We used Gaussian elimination transforming square matrix to reduced row echelon form as a real task test. It was chosen because it is a computational task and it allows us to show the practical applicability of the prototype. At the same time this program includes stages with parallel functions, allowing us to evaluate how the prototype performs in non trivial tasks.

5.1 Setup

Workflows that were to be processed by standard tools were handled as follows: the functions were called with the help of faas-cli application in a shell script, their output was transferred to the input of the next function through UNIX pipelines.

The prototype received dependency graph describing the same workflow and the same input data.

The execution time of the workflows was measured with **time** utility.

The following serverless functions took part in the evaluation:

1. **nodeinfo** - a function from OpenFaaS store that shows system configuration.

2. **text** - this function receives a large text file as an input and simply returns it. Thus it allows to compare how the big data transfers affects the speed of processing a request with the help of standard tools and the prototype. We used following file sizes: 1KB, 10KB, 1MB, 10MB.

3. **sleep** - this function receives an empty string as an input, waits for a specified period of time, and returns an empty string. It allows to compare how a long execution time affects the speed of a request processed by standard tools and by the prototype. In our evaluation, various time intervals were used: 0.1s, 0.5s, 1s.

We used square matrices of sizes from 3 to 80 in the evaluation of the Gaussian elimination implementation. The matrices elements were natural numbers in the range [0, 9]. The program representing traditional tools was written in Go and used http requests to invoke serverless functions. Implementation with the help of the prototype was the same as

in other experiments. The Gaussian elimination itself was implemented by three serverless functions:

1. **find** - searches for the first row, if any, with a nonzero element in the given column starting from the given row. If such a row is found, this row is swapped with the starting one. Returns the modified matrix.

2. **substr** - nullifies a given element of a row by subtracting another row from it. Returns a row with a nullified element and the number of this row in matrix.

3. **toMatrix** - receives matrix rows and their numbers in the matrix and restores the matrix from these data.

5.2 Results and Analysis

Function text Since the prototype allows to execute a workflow without unnecessary data exchanges with the user, when large input data is involved it should show better performance compared to processing the same workflow by standard tools(here and further by standard tools we mean OpenFaaS system without our prototype). The difference in execution time should increase with the increase in the number of functions and thus data exchanges in the workflow and processed data size.

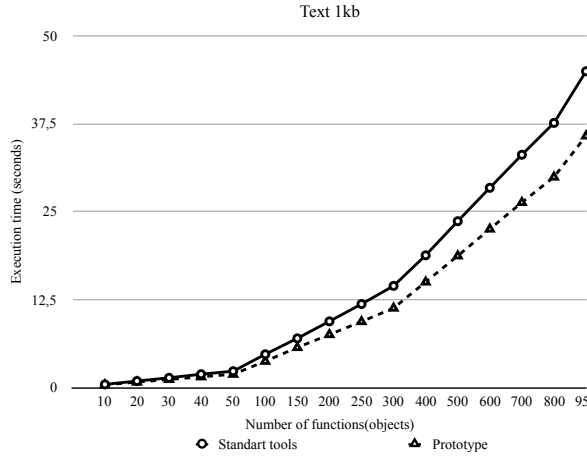

Figure 6. Function text with 1kb input

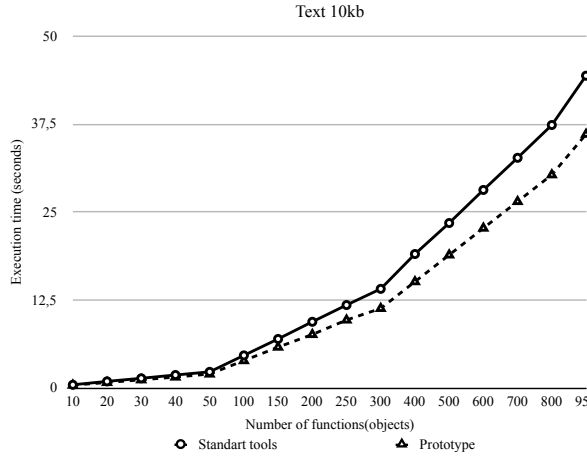

Figure 7. Function text with 10kb input

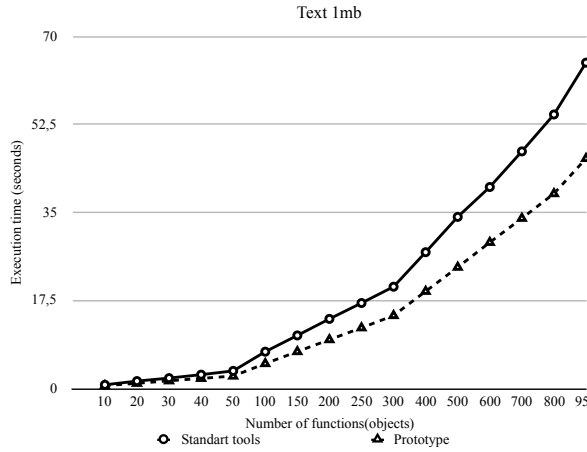

Figure 8. Function text with 1mb input

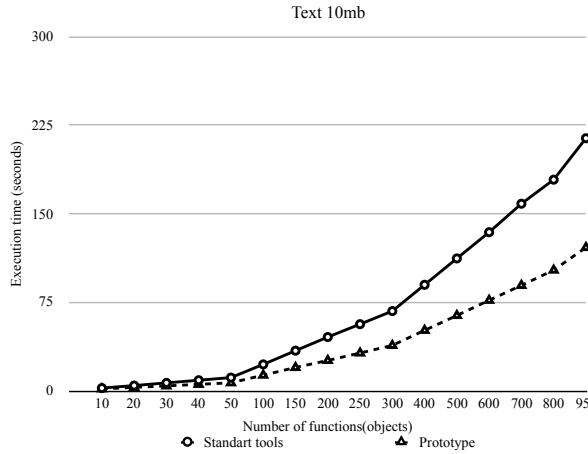

Figure 9. Function text with 10mb input

Function sleep The prototype allows to reduce the overhead of excessive data exchanges between the user and the server, but it does not allow to optimize the execution time of the workflow using data dependencies. Thus processing a workflow consisting of functions that require long computations and operate on small amounts of data the prototype should show the results similar to the results shown by standard tools. The difference in execution time should decrease with increasing time to calculate one function, and rise with increasing the number of functions in the workflow.

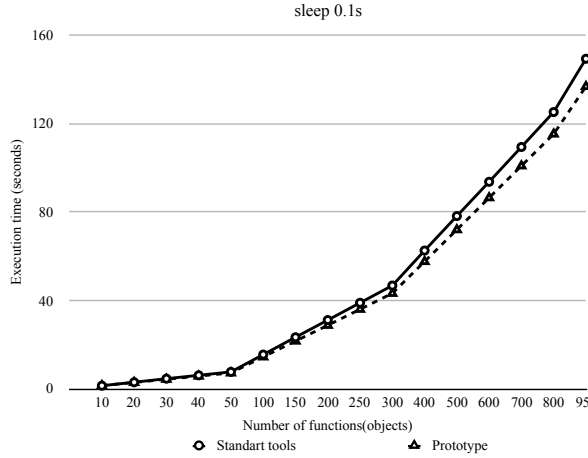

Figure 10. Function sleep waiting for 0.1s

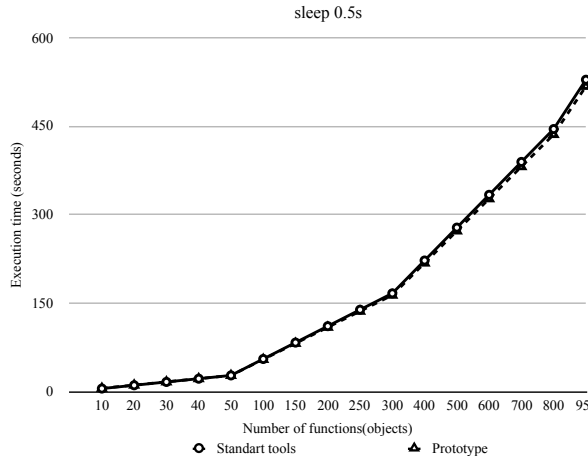

Figure 11. Function sleep waiting for 0.5s

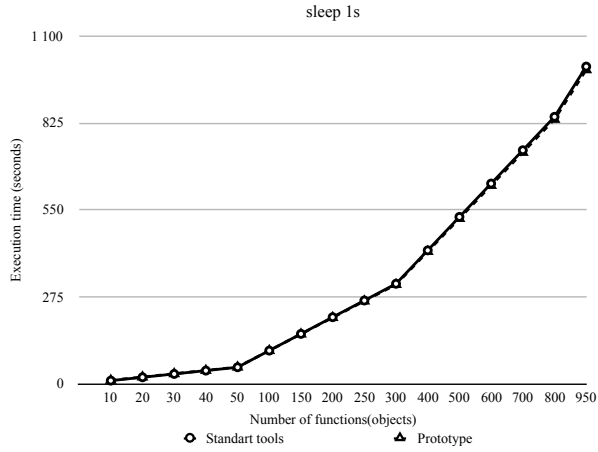

Figure 12. Function sleep waiting for 1s

Function nodeinfo Function nodeinfo evaluation shows that in some cases the prototype can significantly decrease the runtime compared to standard tools.

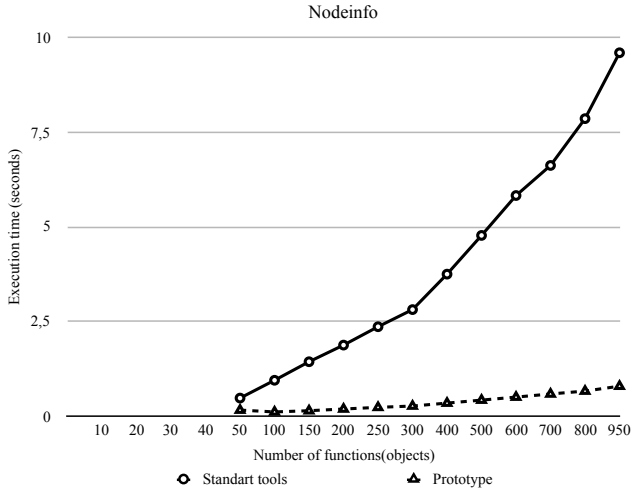

Figure 13. Function nodeinfo

Gaussian elimination To simplify the perception, the experimental results were united into groups of 7 elements by size, the graph shows the average value in each group. The results confirm the possibility of using this approach in the implementation of composite functions performing non-trivial tasks. In addition, the prototype turns out to show the better performance than standard tools the bigger amount of data is processed.

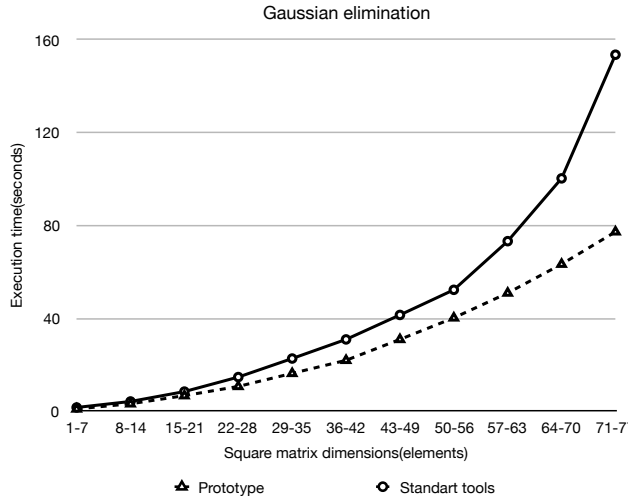

Figure 14. Gaussian elimination

6. Conclusion and Further research

The prototype allows to reduce the overheads connected with transferring large amounts of data between functions, and at the same time shows similar to standard tools performance if the data transfers are relatively small(kilobytes).

Due to transferring all data exchanges between functions in the workflow to service provider infrastructure, the prototype should also reduce overheads connected with low quality connection between the user and the service provider. Such experimental studies were not carried out as part of this work, and may be carried out in the future. We can reduce the overheads of excessive data exchanges by transferring these exchanges to the service provider's network. Workflows constructed of serverless functions and described by dependency graph open up new possibilities in the use of serverless computing. This allows users to rely on the service provider to organize workflow execution for them.

The current prototype appears to have several shortcomings:

- it does not support cycles when building workflows.
- it does not support nested stages.
- it does not use data dependencies to optimize the order of function invocations, and thus cutting down workflow execution time.

The refinement of these deficiencies may form the basis of future research.

References

1. J. Hellerstein, J. Faleiro, J. Gonzalez, J. Schleier-Smith, V. Sreekanti, A. Tumanov, C. Wu *Serverless computing: One step forward, twosteps back* 12 2018.
2. I. Baldini, P. Cheng, S. Fink, N. Mitchell, V. Muthusamy, R. Rabbah, P. Suter, O. Tardieu *The serverless trilemma: function compositionfor serverless computing* 10 2017, pp. 89–103.
3. A. Akhter, M. Fragkoulis, A. Katsifodimos *Stateful functions as a service in action* Proceedings of the VLDB Endowment, vol. 12, pp.1890–1893, 08 2019.
4. E. Jonas, J. Schleier-Smith, V. Sreekanti, C.-C. Tsai, A. Khandelwal, Q. Pu, V. Shankar, J. Carreira, K. Krauth, N. Yadwadkar, J. Gonzalez, R. Popa, I. Stoica, D. Patterson *Cloud programming simplified: A berkeley view on serverless computing* 02 2019.
5. Amazon `https://aws.amazon.com/ru/step-functions/`, `accessed: 2020-07-13`
6. Inkscape `https://inkscape.org/ru/`
7. A. Jangda, D. Pinckney, Y. Brun, A. Guha, *Formal foundations of serverless computing* Proceedings of the ACM on Programming Languages, vol. 3, pp. 1–26, 10 2019.
8. V. Shankar, K. Krauth, Q. Pu, E. Jonas, S. Venkataraman, I. Stoica, B. Recht, J. Ragan-Kelley, *numpywren: serverless linear algebra* 10 2018.

9. Z. Qian, X. Chen, N. Kang, M. Chen, Y. Yu, T. Moscibroda, Z. Zhang *Madlinq: Large-scale distributed matrix computation for the cloud* EuroSys 2012: 7th ACMEuropean Conference on Computer Systems, Berne, Switzerland. ACM, April 2012, euroSys Bes tPaper Award https://www.microsoft.com/en-us/research/publication/madlinq-large-scale-distributed-matrix-computation-for-the-cloud/

10. T. M. Austin, G. S. Sohi *Dynamic dependency analysis of ordinary programs* Proceedings the 19th Annual International Symposium on Computer Architecture, 1992, pp. 342–351

Zaitseva O.A., Antonenko V.A.

CLEVER: Cold Start for Serverless Applications[1]

Introduction

Serverless computing [1] refers to the concept of creating and running applications without the need for server management. This application describes a more detailed deployment model in which applications, combined into one or more functions, are uploaded to the platform and then executed, scaled, and billed by exact requirements. The serverless computing concept allows you not to waste time and resources on server preparation, maintenance, updates, scaling, and capacity planning. Instead, all of these tasks and capabilities are handled by a serverless platform and are completely separate from developers and IT departments.

Two key features of serverless computing [2]:

1. Cost-payment occurs only on the fact that is, you do not need to pay for downtime. Since servers and their use are not part of the serverless computing model, you need to pay only for those computing resources that are allocated directly for the execution of the function code, and not for busy servers.

2. Elasticity-scaling from zero to "infinity". Since developers do not have control over the servers on which their code is running, and do not know the number of servers on which their code is running the scaling decisions are left to the cloud service providers.

Modern serverless platforms that use Function-as-a-Service (FaaS), where the unit of calculation is a function that executes in response to triggers, such as events or HTTP requests, place functions in virtual containers to isolate processes and optimize delivery resources. A container is a separate namespace in the underlying operating system, with its limitation of the central processor, memory and storage, and without access to the rest of the system. When a function is called, the platform creates a new container, fills the container with files and application dependencies, performs the application function, and exits after a short period of inactivity.

[1]This work is partially supported by the Russian Foundation for Basic Research under grant 19-07-01112.

In this article, we study how the caching popularity prediction approach can be applied to FaaS. We consider functions as content that needs to be cached to avoid the **cold start** delays that occur when a function instance starts, usually with new code from a state without an available runtime. We solve the problem of which functions to leave in the **hot start** mode, that is, the launch of the function instance from the expanded state, when all the necessary infrastructure is already loaded and there is no cold start delay, which is equivalent to **caching the function**, and which one to transfer to the cold start, which is equivalent to deleting functions from the cache. The popularity of functions in this paper will be measured by the number of function calls.

We use the traditional methods of caching video content and based on content popularity caching without a learning period on network with caching support from article [3]. That article presents basic caching algorithms such as LRU, LFU, etc. An algorithm based on content popularity caching without a learning period is presented by PopCaching algorithm. The description of that algorithm will be presented in Section III.

In the article, we present the results of an experimental study in which we compare the operation of the modified PopCaching algorithm with traditional caching policies, with the Beladi algorithm and with the execution of functions without delay. The studies were carried out with different restrictions on s - cache capacity, that is, with the restriction on the number of functions that can be called from the hot start state of the function, and with different patterns of user behavior. We compare the following criteria: `hit_rate` - the percentage of hits in the cache, so the percentage of function calls without delay and `duration` - the total duration of the functions in one experiment.

Next, in Section 1, we will present the most famous caching methods, in Section 2 we formulate a formal statement of the problem, and in Section 3 we describe the developed cold start function. An experimental study was carried out for the developed system, the description of which is presented in Section 4. Section 5 comments on the results and in Section 6 we outline some future work.

1. Related Work

In this section, we will consider traditional methods of caching video content and caching based on the popularity of content without a learning period on networks with caching support from the article [3].

We compare algorithms with 3 criteria: Serverless Platform Applicability to have opportunity to deploy an algorithm on the

platform, Lack of learning period to simplify our solution, and Optional caching for functions not to store function before its running. Results of comparing in Table 1.

Algorithm	Serverless Platform Applicability	Lack of learning period	Optional caching for functions
LRU	+	+	-
LFU	+	+	-
FIFO	+	+	-
LRFU	+	+	-
PopCaching	+	+	+
MIN	-	?	-

Table 1: Comparison of Algorithms

Least recently used [4] (LRU) is a traditional caching scheme. LRU stores data about the time of the last access to the function, and if there is not enough memory, it replaces the function with the longest downtime, that is, the time when the function is not called, with the new function called. The main disadvantage of LRU about the problem under consideration is the use of recentness as a metric of choice because memory can be wasted on unpopular functions just because they were recently requested. LRU does not meet criterion 3.

Least frequently used [5] (LFU) is a caching scheme based on the frequency of requests as an indicator of the popularity of functions. When requesting a function that is not already in the cache, the function with the least number of requests is forced out of the cache. The disadvantage of LFU - cache pollution - is a phenomenon in which a previously cached unpopular function that received a large number of historical calls remains in the cache. LFU does not meet criterion 3.

First in first out [3] (FIFO) is a caching scheme that stores functions by the order of calls. When calling a function that is not in the cache, the earliest stored function is forced out, which means that we must first cache the function and then execute it, which does not satisfy criterion 3.

Least recently frequently used [6] (LRFU) is a caching scheme based on LRU and LFU. The coefficients for the recent functions and the frequency of their call are determined in advance. Does not satisfy criterion 3.

PopCaching [7] is a caching scheme that eliminates the need for a training phase. When a function is called, its popularity is determined by the context vector. The main difference from basic caching schemes is

that the function is not necessarily cached. This scheme suffers from low productivity at the initial stage, because the forecasting method does not have previous knowledge about popularity models, but gradually studies them. This caching scheme meets all the criteria.

Belady Algorithm (MIN) [8] is an optimal algorithm. Not applicable in real systems, as it requires information about future launches.

2. Formal problem statement

Following [7], we introduce the following notation:

- $\varsigma = \{1, 2, \ldots, C\}$– a set of functions that can be requested by the user.
- $s < C$ – the capacity of the cache. **The number of functions** that can be supported to run without delay. For the simplicity of the problem, in this paper, we assume that all functions are the same size, so we are talking about the number of functions.
- Cache $= [func_1, func_2, \ldots, func_s]$ – variety of functions in the cache, that is, available to run without delay. $func_i$, where $1 \leq i \leq s$, contains: `name` – function name and `estimate` – function popularity rating.
- `req` – a request to execute a function, which is represented by a context vector with the following values: `name` – function name, `hot` – last hot start time in seconds, `cold` – last cold start time in seconds, `last_count` – number of function calls in the last 100 calls, `last_count2` – number of function calls in the last 10000 calls.

`last_count` and `last_count2` are constants for working of the PopCaching algorithm. These values were chosen as appropriate constants for short-term and long-term periods.

In this paper, we are given ς, s, and incoming `req` queries.

It is necessary: Keep Cache up to date. The relevance of the cache is understood as the requirement to update it after each request for the function. Initialize the execution of functions by incoming `req` requests.

3. Implementation of a system for minimizing a cold start function

3.1 OpenFaaS

To develop a system for minimizing the cold start of a function, OpenFaaS serverless open-source platform was chosen, which has several positive qualities, according to [9]:

- A wide range of supported programming languages;
- OpenFaaS is a complete all-in-one framework: you do not need to configure manually a lot of custom components;
- OpenFaaS has Prometheus [10] as an integrated extensible monitoring solution;
- Simple setup process;
- Native support for Docker Swarm [10].

OpenFaaS [11] is an open-source serverless platform for Docker and Kubernetes [10]. OpenFaaS uses the OpenFaaS CLI to develop and deploy functions. The developer provides only the function and the handler, and the CLI processes the function and places it in the Docker container. The container contains a web server that acts as an entry point for function calls. Gateway API provides an external interface for functions, collects metrics and controls scaling. The OpenFaaS API Gateway proxy balances events and uses resource monitoring to decide on scaling the number of containers. Integration with the cloud platform may use existing container monitoring (Prometheus) or CPU-based scaling (Kubernetes) but leaves a serverless platform with less control over instance location.

By default, OpenFaaS disables the ability to scale to zero instances of the function. But this platform offers the faas-idler tool, which can be additionally configured for your project.

A Prometheus [12] is a monitoring system with the ability to collect open-source metrics. The Prometheus instance is running in the Gateway container that is deployed when starting the OpenFaaS framework.

The faas-idler module (Figure 1) scales OpenFaaS functions to zero instances after a period of inactivity that the user sets. faas-idler is implemented as a controller that regularly polls Prometheus metrics and scales function instances based on the inactivity condition [13].

For the system under development, we will change work of the faas-idler module by the algorithms selected in the review. In the traditional scheme shown on Figure 1, Prometheus metrics are used to decide to scale functions to 0 instances after a period of inactivity. By default this period equals 5 minutes. In the article we focus on function invocation and function cold start and hot start mode managing. We use faas-idler module for changing a function start mode. When a function is requested, we run one of the selected algorithms to decide whether to store the function in cache. According to an algorithm decision we send an http-request to the OpenFaaS platform where faas-idler module processes and scales functions to 0 or 1 instance if needed. After scaling we send an http-request to the OpenFaaS platform for running the function. In this case, Prometheus metrics are only used to collect performance metrics.

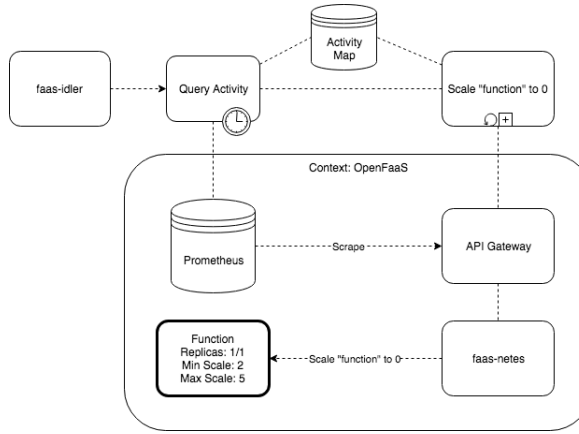

Figure 1. Architecture of OpenFaaS [14]

3.2 Software implementation

The software implementation of the system for minimizing the cold start time of a function is as an application for OpenFaaS, which makes a decision on scaling functions and sends HTTP-requests to the OpenFaaS platform to execute them.

The workflow of the application is as follows:

1. A request is received to execute the function.
2. By the name of the function, this function is searched in the cache of functions ready for execution without delay.

3. If the function is not found, the application decides on the extrusion and caching of functions with the subsequent initialization of their scaling and the cache update is initialized by the decision made.

4. Sending an HTTP-request to execute a function.

5. Collect metrics using Prometheus from the serverless OpenFaaS platform and update them with cached functions.

This work scheme is presented in Figure 2.

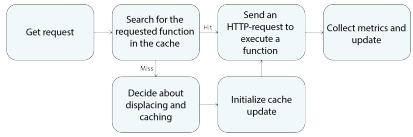

Figure 2

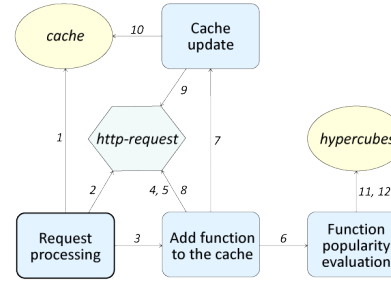

Figure 3

3.3 Applying the PopCaching Algorithm

The PopCaching algorithm consists of 3 main stages:

1. Update

2. Request processing

3. Learning

1. The Update module initiates updating information on the number of function launches for the metrics `last_count` and `last_count2`.

2. The Request Processing module presentes in Figure 3 works as follows:

   (a) Search the cache for functions that are ready to be executed without the delay of the requested function.

   (b) If the function is in the cache, sending an HTTP-request to execute the function.

(c) If the function is not found in the cache, call the Add Function to the Cache.

`Add Function to the Cache`:

(d) Sending an HTTP-request to scale the function.

(e) Sending an HTTP-request to execute a function.

(f) Call function popularity evaluation.

(g) If the revealed popularity of the function is higher than the popularity of the most unpopular function in the cache, call the Cache update function.

(h) Otherwise, sending an HTTP-request to scale the requested function to zero.

`Cache update`:

(i) Removing the most unpopular function from the cache by sending an HTTP-request to scale the function to zero.

(j) Paste the requested function into the cache.

`Function popularity evaluation`:

(k) Find the hypercube to which the function belongs. Hypercube is a combination of functions with adjusted parameters.

(l) The popularity assessment of the hypercube.

Each P_i hypercube has 2 variables: $N(P_i)$ – the number of requests in P_i; $M(P_i)$ – the sum of the identified future call frequency to function calls.

$\widetilde{M}(P_i)$ – the predicted future popularity is calculated based on the variables of the hypercube:

$$\widetilde{M}(P_i) = \frac{M(P_i)}{N(P_i)} \tag{1}$$

In this paper, to modify the PopCaching algorithm in terms of popularity, we do not evaluate the popularity of functions whose execution difference between the last function calls with and without delay is less than 0.1 second. We assume that this is inessential for FaaS and we can neglect it. To implement this approach, we skip the step of the popularity evaluation of such functions, considering their popularity to be a constant equal to 0 to be always executed from a cold start state.

3. The Learning module initiates updating information in the hypercube:

(a) The definition of the hypercube P_i to which the requested function belongs is by the parameters of the hypercube.

(b) Updating P_i hypercube variables:

(c) If the number of requests has reached the threshold value, break it into smaller hypercubes:
If $N(P_i) \geq \zeta(l_i)$, where
l_i is a hypercube level, numbering from zero;
z_1, z_2 are user coefficients;
$\zeta(l_i)$ is a threshold value:

$$\zeta(l_i) = z_1 * 2^{z_2 * l_i} \tag{2}$$

Update hypercube variables:

(d) The number of requests is increased by one:

$$N(P_i) = N(P_i) + 1 \tag{3}$$

(e) The sum of the revealed future request frequency increases by the request frequency of the requested function:

$$M(P_i) = M(P_i) + M(req) \tag{4}$$

Splitting into smaller hypercubes:

(f) The partition of the hypercube into 2^d hypercubes P_j, where d is the number of parameters of the context vector.

(g) Each new hypercube P_j has a level one more than the parent:

$$l_j = l_i + 1 \tag{5}$$

(h) Inheritance by each hypercube of the values of the variables of the parent hypercube:

$$N(P_j) = N(P_i) \qquad (6), M(P_j) = M(P_i) (7)$$

The PopCaching algorithm learns relationships between function call frequency and future function call prediction. The algorithm makes the right decisions about cache replacements for maximising cache hit rate. The PopCaching algorithm does not predict future popularity for each function. It learns popularity for access patterns and searches similar access patterns for different functions. A future popularity prediction procedure does not need a learning period nor a priori knowledge of popularity distribution and it can be used from the start.

4. Experimental research

4.1 Methodology

System testing is carried out using a function request generator with various parameters. To meet the objectives of the study, each study changes one of the parameters: $s, 5 \leq s \leq 40$, in increments of 5, where s is the "capacity" of the cache, that is, the number of functions that can be launched without delaying a cold start; `probabilities` is a list with the given function start frequency to regulate user behavior.

Probabilities are generated in such a way that the following possible patterns of user behavior are investigated: 1) Calling functions with uniform distribution when no function is called more often than another. 2) For longer functions, the probability of a call is higher. 3) For shorter functions, the probability of a call is higher. 4) The probability of calling functions does not depend on their duration. Among functions, both longer functions and shorter functions are equally likely to be equally likely. 5) Simulate a change in user preferences by changing the probability of a function call after N calls, where N is predefined.

Since the total duration can vary greatly depending on the order of calling the functions and calling certain functions, the system is tested in parallel for all the algorithms given in the implementation. Functions are launched separately without limitation to determine the minimum possible run time in this study. At the end of each study, the values of the criteria for comparison are calculated.

For research, we took the standard OpenFaaS functions [11] and the generated functions with a timeout to simulate long-term functions.

4.2 Results

Figures 3-9 below show the results of an experimental study in the form of diagrams. In the diagram legends, PopCaching is an algorithm without modifications, PopCaching2 is an algorithm with a modification that we made in this paper, MIN is an optimal algorithm, Hot is a script for launching functions in which all functions are supported in a hot start state, that is, they are launched without delay, shows the minimum possible total operating time of the functions with the generated set of requests for launching the functions.

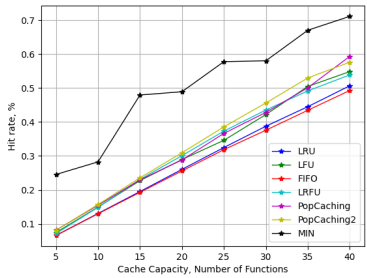

Figure 4

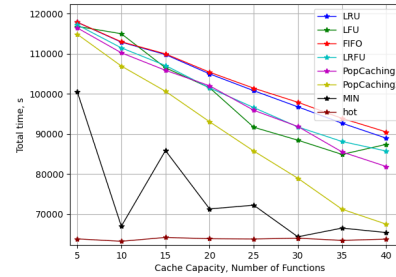

Figure 5

Figures 4-5 show the results of an experimental study when generating requests to launch functions with a higher probability of calling functions with a longer duration. Figure 4 shows that the modified algorithm has a larger number of hits compared to other algorithms. Figure 5 demonstrates how, depending on the size of the cache, the algorithm developed in this paper approaches the value of the optimal algorithm.

Figures 6-7 show the results of an experimental study when generating requests to launch functions with a higher probability of calling functions with a shorter duration. Figure 6 shows that with such query generation, the algorithm developed in this paper has the least cache hits. At the same time, Figure 7 shows that the total execution time of functions is better than that of the optimal algorithm even with a cache capacity of 25 functions out of 100. With a cache capacity of < 25 functions, the algorithm shows the closest results to the optimal cache algorithm. With a cache capacity of ≥ 25 functions, the algorithm better than optimal caching algorithm because it does not need to cache function before running it.

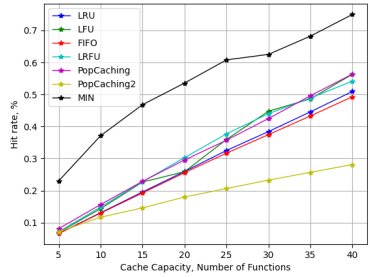

Figure 6

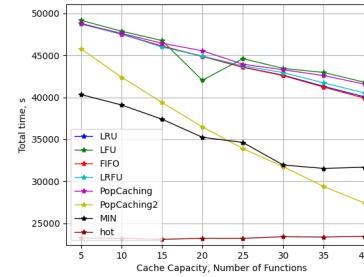

Figure 7

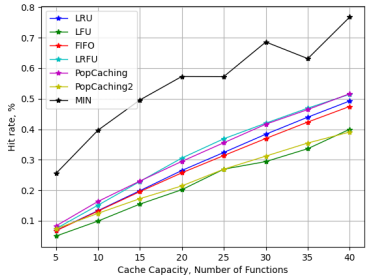

Figure 8

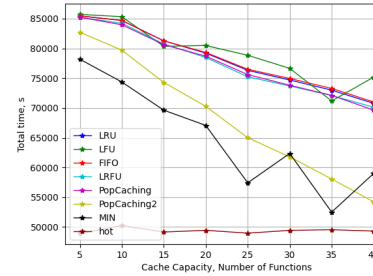

Figure 9

Figures 8-9 show the results of an experimental study when generating requests to launch functions with simulated changes in user behavior. To simulate changes in user behavior during the experimental study, the probabilities of requests to call functions changed. Although the developed algorithm shows a lower hit percentage compared to other algorithms, as can be seen in Figure 8, by the value of the total execution time of the functions, it is closest to the value of the total execution time of the functions when applying the optimal caching algorithm.

Figure 10 shows the minimum possible time for executing functions, when all functions are started from the hot start state, relative to the total execution time of functions called using the modified PopCaching algorithm, in percent. The numbers indicate different scenarios of work by the patterns of user behavior given in subsection A) Methodology for conducting experimental research. Figure 10 shows the dependence of the relative execution time of functions when they are called with or without delay on the cache capacity. With a cache capacity of ≥ 25 functions, the relative amount of execution time is $> 68\%$, and with a cache capacity of 40 functions, the relative amount is $> 85\%$ for all the scenarios considered.

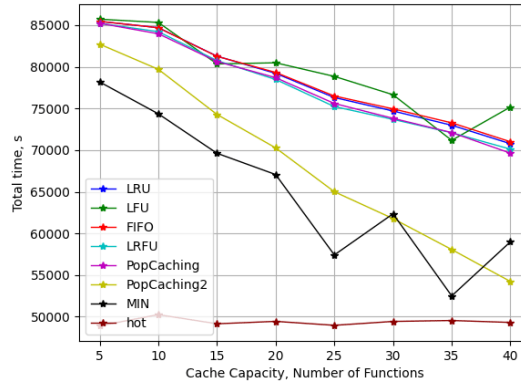

Figure 10. Relative PopCaching Runtime Dependence and Maintaining All Hot Start Functions on Cache Capacity

4.3 Analysis

The study showed that in most cases the developed algorithm has a lower percentage of a hit in the cache than traditional caching schemes. Moreover, if we compare the total operating time of the functions, the developed algorithm shows the closest result to the value of the total execution time of the functions called using the optimal caching algorithm. Since the optimal caching algorithm is not applicable in real systems because of the need to know future requests, the developed algorithm shows the best result in terms of the operation time of functions.

The study also revealed that depending on the user's behavior with a cache capacity of ≥ 25 functions, the developed algorithm may have a total time of functions less than the total time of functions called using the optimal caching algorithm. With this limitation on cache capacity in the worst-case scenario, the runtime of functions called using the developed algorithm does not exceed the runtime of functions called using the optimal caching algorithm by more than 20%.

When comparing the total execution time of functions called using the developed algorithm with the minimum time when all functions are called without delaying the cold start of the function, depending on the cache capacity limit, the value of the developed algorithm exceeds the execution time value without the cache limit by no more than 50% with the minimum cache limit examined. With a limit on the cache capacity of ≥ 25 functions, the execution time of functions called using the developed

algorithm exceeds the execution time of functions without delay by no more than 32%.

5. Conclusion

In this article, we introduced a system for minimizing the cold start of a function in serverless computing, based on the caching algorithm selected during the review, in which we understood cached functions as functions in the state of a hot start of a function that we can call without delaying the cold start of a function. During an experimental study, it was found that the optimal limit on the cache capacity, that is, a limit on the number of functions that can find functions in the hot start state, is at least 25 functions in the presence of 100 possible functions, i.e. 25% of the total. Also, during the experimental study, it was found that the developed algorithm shows the closest values of the total execution of functions to the value of the optimal caching algorithm and the value of the minimum possible time for starting the functions in comparison with traditional caching policies and the basic algorithm without modifications.

6. Future Work

Future interests for the research are the research of the possibility of reducing the delay of a cold start function by maintaining part of the infrastructure to perform functions and the research of the possibility of modifying the algorithm developed in this paper to work with different types of delay of a cold start function.

References

1. CNCF Serverless Working Group *CNCF WG-Serverless Whitepaper v1.0* 2018.

2. P. Castro et al. *The server is dead, long live the server: Rise of Serverless Computing, Overview of Current State and Future Trends in Research and Industry* 2019.

3. H. S. Goian et al. *Popularity-based video caching techniques for cache-enabled networks: a survey* IEEE Access., vol. 7, pp. 27699-27719, 2019.

4. M. Ahmed et al. *Analyzing the performance of LRU caches under non-stationary traffic patterns* 2013.

5. A. Jaleel et al. *High performance cache replacement using re-reference interval prediction (RRIP)* ACM SIGARCH Computer Architecture News, vol. 38, № 3, pp. 60-71, 2010.

6. D. Lee et al. *LRFU: A spectrum of policies that subsumes the least recently used and least frequently used policies* IEEE transactions on Computers, № 12, pp. 1352-1361, 2001.

7. S. Li et al. *Popularity-Driven Content Caching* IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications, pp. 1-9, 2016.

8. L. A. Belady *A study of replacement algorithms for a virtual-storage computer* IBM Systems journal, vol. 5, № 2, pp. 78-101, 1966.

9. F. Gand et al. *Serverless container cluster management for lightweight edge clouds* 10th Intl Conf on Cloud Computing and Services Science CLOSER, 2020.

10. S. K. Mohanty et al. *An evaluation of open source serverless computing frameworks* CloudCom, pp. 115-120, 2018.

11. A. Ellis *OpenFaaS - Serverless Functions Made Simple* `https://github.com/openfaas/faas`

12. *Prometheus* The Linux Foundation, 2014-2020 `https://prometheus.io/`

13. A. Ellis *Add feature: scale from zero to 1 replicas 685* `https://github.com/openfaas/faas/pull/685`

14. *faas-idler* `https://github.com/openfaas-incubator/faas-idler`

Аникевич Ю.В., Бахмуров А.Г., Чайчиц Д.А.

Разработка и реализация платформы для сбора и обработки физиологических данных о человеке

Введение

С каждым днем все известнее становится такое направление в здравоохранении, как (IoMT, Internet of Medical Things). Данная формулировка обозначает концепцию сети, которая позволяет объединять устройства и приборы, способные отслеживать состояние человеческого организма, окружающую его среду и влиять на профилактический, лечебный и реабилитационный процессы.

На сегодняшний день проблема мониторинга состояния человека вне медицинских учреждений чрезвычайно актуальна. Данное направление существует с целью получения детальной информации об организме человека, на основании которой можно принимать обоснованные решения, предотвращающие возникновение и развитие заболеваний. Современная концепция мониторинга предлагает проводить непрерывный контроль за состоянием человека, и, с помощью оценки диагностических показателей, выявлять случаи отклонения их от нормы.

Персональные мониторные устройства способны выявить нарушения в работе систем организма на более ранней стадии, а также могут послужить дополнительным источником информации для специалистов. Также, такая система позволяет оперативно предупредить критическое состояние пациента, что даст возможность вовремя провести необходимые мероприятия для его стабилизации.

1. Анализ предметной области и актуальность работы

К настоящему времени сформирован обширный рынок портативных носимых устройств и систем для регистрации физиологических параметров человека (например, умные часы, фитнес-браслеты). Для большинства из них производитель предлагает свои проприетарные решения для обработки и хранения собираемых данных. Также известны приложения, например, Apple Health, Google Fit, которые работают с датчиками разных производителей, но также являются

проприетарными и предусмартривают хранение данных на серверах производителя. Следовательно, из них невозможно получить необработанные данные, и персональные данные пользователей хранятся “на стороне”. Актуально создание платформы сбора и обработки медицинской телеметрии с открытым исходным кодом, так как оно позволит:

- реализовывать собственные методы обработки телеметрии, как в исследовательских целях, так и для обучения студентов, например, по специальности “медицинская кибернетика”;
- устранить необходимость хранения персональных данных на зарубежных серверах;
- устранить зависимость от ошибок сторонних разработчиков; так, например, была описана ошибка в Google Fit;
- создать задел для отечественного промышленного средства сбора и обработки медицинской телеметрии.

Однако в большинстве своем они не являются open-source проектами, и данные из них получить либо невозможно, либо они выдаются уже в обработанном виде. Таким образом, создание открытой платформы сбора и обработки физиологических данных является актуальной задачей на сегодняшний день.

2. Архитектура платформы

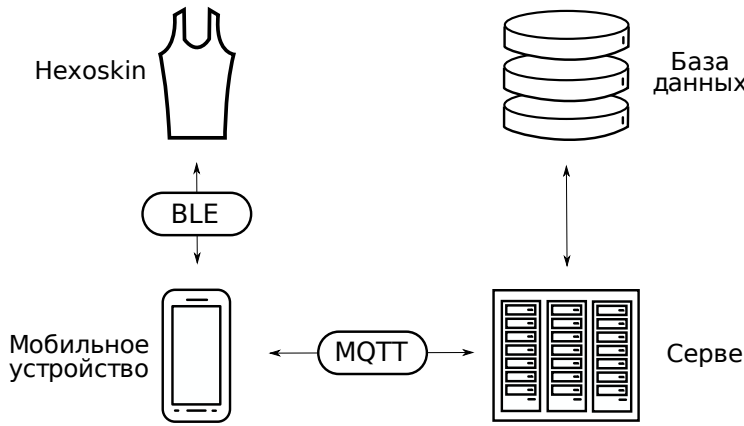

Рис. 1: Архитектура платформы

Архитектура платформы показана на рисунке 1:

1. Физиологические показатели пользователя считываются с помощью датчиков умной одежды Hexoskin, после чего отправляются на мобильный телефон на операционной системе Android по протоколу Bluetooth Low Energy (BLE).

2. Мобильный телефон принимает считанные показатели и отправляет их на сервер по протоколу MQTT.

3. Сервер записывает полученные данные в базу данных, откуда их можно взять для медицинской обработки.

3. Обоснование выбора компонентов

В качестве носимого устройства выступает умный жилет Hexoskin. Это устройство позволяет считывать с человека следующие показатели:

- частоту сердцебиения;
- вариабельность сердечного ритма;
- число шагов;

- частота дыхания;
- VO2 max;
- объем легких;
- уровень активности;
- число шагов в минуту(каденция);

Выбор обусловлен как широкими возможностями жилета по сбору данных, так и широким его применением для медицинских исследований. Жилет подключается к смартфону посредством протокола Bluetooth Low Energy.

3.1 Выбор СУБД для хранения телеметрии на сервере

Данные, получаемые с носимого устройства, представляют собой временные ряды. Под временным рядом понимается совокупность значений какого-либо показателя, собранного в различные моменты времени. Измерения, составляющие временной ряд, упорядочены по временной шкале, что показывает историю изменения данных во времени и может быть очень полезным для медицинской диагностики.

Для работы с временными рядами существуют специализированные СУБД, которые более удобны для пользователя, и обеспечивают большую скорость записи и более высокую производительность запросов, несмотря на большой объем данных, которые они организуют. В некоторых случаях СУБД временных рядов выполняют те же функции, что и обычные СУБД, однако попытка использовать реляционную базу данных для данных временных рядов приведет к снижению производительности.

Запросы к базе данных временных рядов аналогичны запросам в других типах баз данных, но вместо поиска по значениям пользователи могут осуществлять поиск по прошедшему периоду времени, диапазону дат или конкретному моменту времени, когда произошло событие. В рамках данной работы для хранения телеметрии на сервере были рассмотрены системы управления базами данных, оптимизированные для хранения и обработки временных рядов.

Сценарий использования БД и обусловленные им требования к СУБД

Для выбора СУБД рассматривались такие критерии, как открытость исходного кода, долговременное хранение, масштабируемость

и наименьшая гранулярность.

Под масштабируемостью понимается возможность увеличить объем хранилища или производительность за счет добавления дополнительных узлов.

Долговременное хранение необходимо для больших объемов данных, поскольку не все СУБД могут хранить все значения в полном разрешении в течение длительного периода времени. Старые данные могут удаляться или храниться в агрегированном состоянии (например, хранение среднего значения за минуту вместо значений для каждой миллисекунды).

Гранулярность описывает наименьшее возможное расстояние между двумя временными метками. При вставке данных в базу данных степень детализации входных данных может быть выше, чем степень детализации хранилища, для которой СУБД гарантирует безопасное хранение. Некоторые СУБД принимают данные с меньшей степенью детализации, чем они могут хранить, что приводит к агрегированию или потере данных.

База данных представляет собой набор таблиц, где каждая таблица отведена под конкретного пользователя. В таблицы будет производиться запись семи измерений, в соответствии параметрами, получаемыми с датчиков Hexoskin.

Согласно информации о датчиках Hexoskin, самая высокая частота получения данных имеет значение 256 Гц. Строго говоря, для выбора СУБД важно, чтобы она имела возможность сохранять данные с разрешением в 4 мс. Предполагаем, что гранулярность принимает значения 1 мс, 10 мс, 100 мс и т.д. Таким образом, необходимая гранулярность имеет значение 1 мс.

Необходимость в долговременном хранении обусловлена возможностью хранить данные мониторинга за длительный промежуток времени, что позволяет получить более полную картину о состоянии человека.

Высоким приоритетом при выборе обладает такой критерий, как время выполнения запросов, то есть важно, чтобы чтение из базы данных происходило достаточно быстро. Более низким – время записи в базу данных и степень сжатия записанных данных на диске.

Выбор СУБД для обзора и результаты обзора

Для обзора были взяты СУБД временных рядов, которые занимают первые десять позиций в рейтинге DB-Engines СУБД временных рядов. DB-Engines [10] - это независимый веб-сайт, который ранжирует системы управления базами данных на основе популярности в поисковых системах, упоминаний в социальных сетях, количества

предложений о работе и частоты технических обсуждений. Также в список обозреваемых СУБД отдельно была включена ClickHouse, разрабатываемая компанией Яндекс, в виду хороших результатов, показанных в статье [9], где сравнивается производительность СУБД ClickHouse, InfluxDB и TimescaleDB. Таким образом, в список обозреваемых систем управления базами данных были включены СУБД InfluxDB, Kdb+, Prometheus, Graphite, RRDtool, Druid, OpenTSDB, TimescaleDB, FaunaDB, KairosDB, ClickHouse. Одним из критериев выбора СУБД является открытость исходного кода, поэтому далее такие решения, как Kdb+ и FaunaDB не рассматривались. Результаты сравнения по оставшимся критериям приведены в таблице 1. По результатам сравнения, всем перечисленным критериям удовлетворяют InfluxDB, ClickHouse и Druid.

	Масштабируемость	Долговременное хранение	Наименьшая гранулярность для хранения	Наименьшая гранулярность для безопасного хранения
InfluxDB	+	+	1 мс	1 мс
Prometheus	+	-	1 мс	1 мс
Graphite	+	+	1000 мс	1000 мс
RRDtool	-	-	1000 мс	1000 мс
Druid	+	+	1 мс	1 мс
OpenTSDB	+	-	1 мс	>1 мс (1000 мс)
TimescaleDB	-	+	1 мс	1 мс
KairosDB	+	-	1 мс	1 мс
ClickHouse	+	+	1 мс	1 мс

Таблица 1: Сравнение СУБД

Из статьи [14] можно сделать вывод, что Druid больше подходит для сценариев, когда имеется большой кластер с большим количеством таблиц и данных. Таблицы и наборы данных периодически появляются в кластере, анализируются и удаляются из него в отличие от ClickHouse, где таблицы и данные находятся в кластере перманентно. Поэтому Druid был исключен из дальнейшего сравнения.

Экспериментальное сравнение СУБД

Для сравнения по производительности СУБД InfluxDB v1.8.0 и ClickHouse v19.3.3 были выбраны следующие критерии: время загрузки данных, степень сжатия данных на диске и время выполнения запросов.

InfluxDB [8] - это нереляционная система управления базами данных временных рядов, разработанная компанией InfluxData, которая имеет открытый исходный код с дополнительными компонентами с закрытыми исходными ко- дами. Она написана на языке программирования Go и оптимизирована для обработки временных рядов. Поддерживает SQL-подобный язык запросов. Время является самой важной концепцией в InfluxDB. Столбец времени включен в каждую базу данных InfluxDB и содержит дискретные временные метки, которые связаны с конкретными данными.

ClickHouse [7] – это аналитическая колоночная база данных с открытым исходным кодом, разработанная компанией Яндекс для OLAP сценариев работы. Язык запросов ClickHouse представляет собой диалект SQL.

Колоночные базы данных хранят записи в блоках, сгруппированных по столбцам, а не по строкам. Поскольку такая БД не загружает данные для столбцов, которых нет в запросе, она тратит меньше времени на чтение данных при выполнении запросов. Потому колоночные базы данных могут вычислять и возвращать результаты для определенных рабочих нагрузок, таких как OLAP, намного быстрее, чем традиционные строчные системы.

Сервер тестового стенда имел следующую конфигурацию:

- Ubuntu 18.04.3
- Intel(R) Xeon(R) CPU E5-2697 v4 @ 2.30GHz
- 3,9 Gb RAM
- L2 cache: 4096K
- L3 cache: 16384K

Для генерации тестового набора данных была воспроизведена непрерывная запись данных с самой высокой частотой из всех датчиков Hexoskin. Временной промежуток генерации составил около двух недель. В конечном итоге, набор данных состоит из 398.458.880 строк и содержит 16 показателей. Общий объем тестовых данных составил 55 Гигабайт.

Для сравнения выбранных СУБД по времени выполнения запросов было использовано 8 типов запросов, характерных для работы

с временными рядами. Предположим, что $v_1, v2, ..., vn$ – значения, соответствующие датчикам $1, 2, ..., n$. Также в нижеприведенных запросах func() представляет собой функцию агрегации, такую как avg, min и т.д.

1. Точечный запрос
   SELECT v1 ... FROM data WHERE time = ?
2. Запрос временного диапазона
   SELECT v1 ... FROM data WHERE time >? AND time <?
3. Запрос с фильтром значений
   SELECT v1 ... FROM data WHERE v op ? (op: <,>,=)
4. Запрос временного диапазона с фильтром значений
   SELECT v1 ... FROM data WHERE time > ? AND time < ? AND v1 op ? (op: <,>,=)
5. Запрос с лимитом строк
   SELECT v1 ... FROM data LIMIT ?
6. Запрос с функцией агрегации
   SELECT func(v) FROM data
7. Запрос временного диапазона с функцией агрегации
   SELECT func(v) FROM data WHERE time > ? AND time < ?
8. Агрегация + group by
   SELECT func(v) FROM data GROUP BY time(?)

Результаты сравнения СУБД

Загрузка данных

Для загрузки тестового набора данных в базы данных был использован интерфейс командной строки и утилита time для измерения времени загрузки. Загрузка данных в ClickHouse заняла 712 секунд, а в InfluxDB – 2367 секунд, что в 3,3 медленнее, чем время загрузки данных в ClickHouse.

Степень сжатия данных

Пространство, занимаемое тестовым набором данных в случае ClickHouse, составляет 4.7 Гб, а в случае InfluxDB - 3.1 Гб. Таким образом, коэффициент сжатия равен 1:12 для ClickHouse и 1:18 для InfluxDB.

Время выполнения запросов

Для измерения производительности выполнения запросов было использовано 8 типов запросов, приведенных выше. Каждый запрос

был выполнен 5 раз, после чего для него было вычислено среднее значение. Среднее время ответа на запросы для ClickHouse составляет около 3,5 секунд, а для InfluxDB – около 107 секунд. В среднем, время выполнения запроса в ClickHouse в 77 раз быстрее, чем в InfluxDB. По двум из трех критериев ClickHouse показал лучшие результаты. Время загрузки тестового набора в ClickHouse заняло в 3,3 раза меньше времени, чем в InfluxDB и также выполнение тестового набора запросов в среднем вышло в 77 раз быстрее, чем в InfluxDB. Таким образом, по результатам сравнения для использования в реализации была выбрана СУБД ClickHouse.

3.2 Выбор инструмента для промежуточного хранения данных на мобильном устройстве

В связи с возможными разрывами в подключении к сети Интернет немедленная отправка данных на сервер после получения их с жилета не является хорошей стратегией, так как велик риск потери данных. Поэтому после того, как данные с носимого устройства будут получены, необходимо их сохранять до момента отправки. В ОС Android на выбор доступны следующие механизмы хранения данных: внутреннее хранилище, внешнее хранилище, SharedPreferences и база данных. Сравним эти механизмы.

Внутреннее хранилище

Каждое приложение Android имеет свой собственный внутренний каталог хранения, взаимодействующий с ним, в котором приложение может хранить текстовые и двоичные файлы. Файлы внутри этого каталога недоступны для пользователя или других приложений, установленных на устройстве пользователя. Они также автоматически удаляются, когда пользователь удаляет приложение. Таким образом, это обычный текстовый файл, в который можно записать данные и откуда их потом можно прочесть. Для чтения конкретного значения придется искать его в файле вручную. Объем вмещаемой информации будет ограничен объемом свободной памяти во внутреннем хранилище смартфона.

Внешнее хранилище

Поскольку внутреннее хранилище устройств Android обычно фиксировано и часто довольно ограничено, некоторые устройства Android поддерживают внешние носители данных, такие как съем-

ные microSD-карты. Это все еще будет файл, но в который можно будет записать гораздо больший объем данных.

SharedPreferences

Средства SharedPreferences позволяют приложениям сохранять настройки или другие данные в виде пар ключ-значение. Эти данные сохраняются даже когда пользователь закрывает приложение. Android хранит SharedPreferences в виде XML файла. SharedPreferences зависят от приложения, то есть сохраненные данные будут удалены из Android устройства после удаления приложения или после очистки данных приложения. Это также является файлом, но поиск и запись в нем потребуют меньше времени.

База данных

Если использовать базу данных, нужно быть уверенным, что каждое устройство будет её поддерживать. В этом случае в качестве единственного кандидата на выбор остается СУБД SQLite, так как она входит в состав ОС Android и уже есть у каждого в смартфоне. Размер индексируемой памяти для этой СУБД равен 140 Тб, а реальный размер будет ограничен размером внутренней памяти.

Исходя из полученных результатов оптимальным вариантом является использование СУБД SQLite. Данные, которые будут в ней храниться структурированы, что позволит производить предобработку показаний с помощью SQL запросов в будущем. Кроме того, при принудительном завершении приложения (или выключении смартфона), возможна потеря данных при работе с файлом, а СУБД позволяет восстановить данные при перезапуске благодаря журналу.

4. Описание работы мобильного приложения

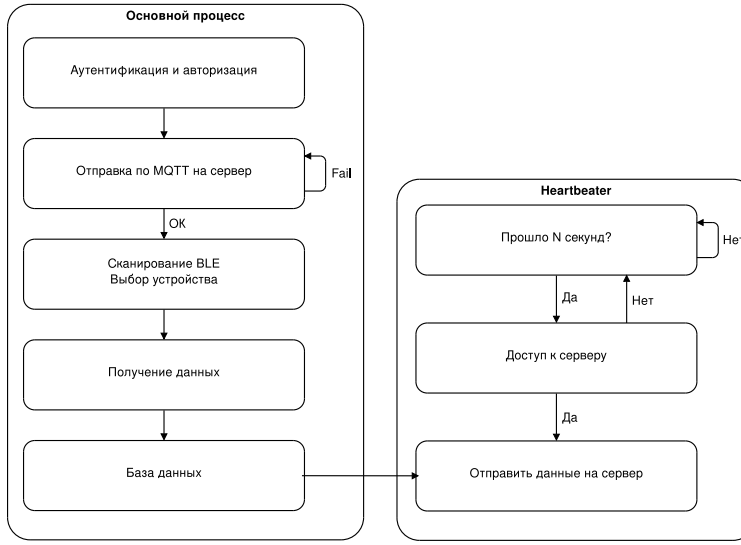

Рис. 2: Схема работы мобильного приложения

Схема работы мобильного приложения показана на рисунке 2. После запуска приложения перед пользователем открывается окно входа в систему. Получив логин и пароль, приложение использует их для создания соединения по MQTTS, после чего в случае ответа от сервера о правильности данных, брокер отправляет подтверждение устройству. В случае ошибки пользователю будет предложено ввести данные заново. После успешного входа приложение переходит в работы с носимым устройством.

Пользователь имеет возможность выбирать, к какому из доступных носимых устройств подключаться. После выбора такового приложение устанавливает соединение и начинает приём данных. Параллельно с этим начинает жизнь второй процесс - Heartbeater. Из название следует его роль: периодически с некоторым интервалом он оживает и проверяет есть ли соединение с сервером и возможность принятия данных. В случае наличия он отправляет содержимое базы данных и очищает последнюю. Если же нет возможности что-то отправить, то процесс снова засыпает до следующего “удара”.

Заключение

После реаализации базовой функциональности системы, проведения серии тестов получилось наладить передачу данных с умного устройства на сервер с промежуточным хранением в базе данных на смартфоне. В ходе экспериментального исследование было выяснено, что расход заряда батареи при использовании радиоинтерфейса телефона увеличивается при росте частоты отправки данных на сервер.

Также в рамках настоящей работы был проведён обзор СУБД временных рядов, на основании которого было проведено дальнейшнее сравнение по производительности двух выбранных СУБД, в результате которого была выбрана СУБД для использования в реализации.

В ближайшей перспективе продолжения работ - экспериментальное исследование программной реализации, измерение экономии заряда аккумулятора. В более отдалённой перспективе - изучение возможности масштабирования решения с помощью облачной платформы (в частности, запуска нескольких экземпляров сервера) для поддержки большого числа пользователей.

Благодарности

Авторы благодарят декана медико-биологического факультета РНИМУ им. Н.И. Пирогова Е. Б. Прохорчука, зам. декана А.А. Лагунина и научного сотрудника ФИАН Д.В. Малахова за предложение по совместной работе и материальное обеспечение - предоставление жилетки Hexoskin. Также авторы выражают благодарность сотруднику компании Яндекс А.А. Любицкому за технические консультации по выбору и использованию ПО.

Литература

1. Balasubramanian N., Balasubramanian A., Venkataramani A. *Energy Consumption in Mobile Phones: A Measurement Study and Implications for Network Applications* //Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement. – 2009.

2. AndroidDevelopersCommunity `https://developer.android.com`

3. Hexoskin SDK https://bitbucket.org/carre/hexoskin-smart-demo/src/master/hexoskin-smart-android

4. EclispeMQTTAndroid https://www.eclipse.org/paho

5. Хранение данных в Android https://code.tutsplus.com/ru/tutorials/android-from-scratch-how-to-store-application-data-locally-cms-26853

6. Authentication with Username and Password - MQTT Security Fundamentals https://www.hivemq.com/blog/mqtt-security-fundamentals-authentication-username-password

7. ClickHouse Documentation. https://clickhouse.tech/docs/ru/

8. InfluxDB Documentation. https://docs.influxdata.com/influxdb/

9. ClickHouse Crushing Time Series.
   https://www.altinity.com/blog/clickhouse-for-time-series

10. DB-Engines https://db-engines.com/en/

11. Rui Liu, Jun Yuan, Benchmark Time Series Database with IoTDB-Benchmark for IoT Scenarios, 2019 https://arxiv.org/pdf/1901.08304.pdf

12. Bader A., Kopp O., Falkenthal O., Survey and Comparison of Open Source Time Series Databases, 2017 https://www.researchgate.net/publication/315838456_Survey_and_Comparison_of_Open_Source_Time_Series_Databases

13. Naqvi S.N. Yfantidou S., Time Series Databases and InfluxDB, 2017 https://cs.ulb.ac.be/public/_media/teaching/influxdb_2017.pdf

14. Comparison of the Open Source OLAP Systems for Big Data: ClickHouse, Druid, and Pinot
    https://leventov.medium.com/comparison-of-the-open-source-olap-systems-for-big-data-clickhouse-druid-and-pinot-8e04-2a5ed1c7

Баула В.Г.

Формальная семантика в обучении программированию

1. Введение

> Программирование начиналось как искусство; даже сейчас большинство учится ему, наблюдая, как работают другие (например, преподаватель или более опытный коллега), постигая приемы и мало задумываясь над принципами, которые лежат в их основе. Однако в результате научных исследований последнего десятилетия найдены некоторые полезные теоретические положения и общие принципы, так что наступает время, когда можно начинать учить принципам и их осознанному применению.
>
> Дэвид Грис «Наука программирования» [1]

Подготовка специалистов в области разработки программного обеспечения является сложным многоэтапным процессом. Имеет большое значение и обеспечение хорошей математической подготовки и достаточно высокого общенаучного уровня знаний. Но, конечно, главным всё же является обучение в области собственно программирования.

Как среди преподавателей по программированию, так и среди самих обучающихся бытует мнение, что первом этапе обучения необходимо иметь как можно больше практики в написании различных простых программ Такая практика приводит к тому, что по определённым шаблонам пишется большое количество упрощённых, сугубо учебных программ, имеющих весьма малое отношение к настоящему программному обеспечению. Предполагается, что лишь на следующем этапе студенты должны вовлекаться в полноценные программные проекты. Этот подход к обучению представляется не совсем правильным, так как игнорирование важных этапов в написании программного обеспечения в начальный период обучения приводит к существенным пробелам в профессиональном образовании студентов. Здесь представляется целесообразным не забывать о старом полезном принципе «лучше меньше, да лучше».

Настоящая работа призвана объяснить важность с самого начала производить строгую спецификация программы и задавать формальную правильность этой программы. Приводятся простые примеры на языке Free Pascal [2].

2. Осознание и спецификация задачи

Традиционно при разработке программного обеспечения выделяют несколько стадий или этапов. Прежде всего, следует заметить, что при этом собственно написание текста программы (он же часто называется исходным или программным кодом) всегда явно или неявно предваряется некоторыми начальными этапами. Принято считать, что разработка и реализация программного обеспечения должны содержать следующие этапы.

Сначала производится осознание задачи. Это исключительно важный этап, он предполагает ясное понимание того, что является входными данными задачи, и что она должна выдать в качестве своего результата. Преподавателям многократно приходилось сталкиваться с фактом, что обучаемый решил не ту или не совсем ту задачу, которая перед ним ставилась. Ошибка на данном начальном этапе обходится очень дорого в смысле затрат времени и нервов, а часто и конечного результата, поскольку неверное решение не всегда возможно быстро исправить так, чтобы получилось верное.

В качестве примера рассмотрим следующую простую задачу. Необходимо ввести из стандартного входного потока целое число в целочисленную переменную x и вывести сумму всех цифр в десятичной записи этого числа. Часто приводится следующее решение этой задачи на Паскале.

```
var x,sum: integer;
begin
  read(x); sum:=0;
  repeat
    sum:=sum+x mod 10; x:=x div 10
  until x=0;
  writeln(x)
end.
```

В этом решении не осознано, что вводимое число может быть отрицательным. При этом в большинстве конкретных реализаций языка Паскаль выражение x mod 10 будет отрицательным (именно так вычисляют остаток целочисленного деления большинство ЭВМ), а

цифры в нашей десятичной системе счисления по определению имеют положительные значения. В результате получиться отрицательная сумма цифр, что лишено математического смысла.

Для исправления этой ошибки можно попытаться перед циклом вычислить абсолютную величину числа `x:=abs(x)`, это может решить проблему, но не всегда полностью. Здесь нужно осознать, что для некоторых целых чисел могут не существовать их абсолютные величины, именно так и обстоит дело в наиболее распространенной на ЭВМ целочисленной системе счисления с так называемым дополнительным кодом. Таким образом, при правильном осознании задачи для верного решения абсолютную величину надо брать от значения каждой цифры числа `sum:=sum+abs(x mod 10)`.

Осознанию поставленной задачи в значительной степени мешают особенности компьютерной арифметики. Та математика, которая реализована в ЭВМ, называется дискретной математикой и значительно отличается от традиционной, привычной для нас со школы математикой. Главное отличие заключается в том, что как для целых, так и для вещественных величин в ЭВМ существует только конечное число допустимых значений. Таким образом, в компьютере существуют минимальное и максимальное целые и вещественные, а вне этих диапазонов чисел нет.

При выполнении арифметических операций (сложения, вычитания и т.д.) с такими величинами, правильный результат получается только в том случае, если этот результат (сумма, разность и т.д.) не выходит за допустимый диапазон представимых величин соответствующего типа (целого, вещественного и т.д.). А что же будет, если, скажем, результат сложения выходит за допустимый диапазон? В описании стандарта Паскаля сказано, что в этом случае результат операции не определён. Это, конечно, не означает, что он принимает случайное значение. Просто, когда в стандарте языка говорится, что результат некоторого действие не определён, это означает, что в каждой конкретной реализации этого языка в данном случае должно быть чётко сказано, что будет делаться. Например, в языке Free Pascal при выходе целочисленного значения за допустимый диапазон, результат будет зависеть от того, в каком режиме будет работать исполнитель (в каком режиме будет выполняться компиляция программы).

Все режимы компилятора задаются комментариями особого вида, интересующий нас режим задаётся директивами `{$R+}` (контроль выхода за допустимый диапазон включён) и `{$R-}` (контроль выключен). При включённом контроле в случае выхода целочисленного значения за допустимый диапазон происходит аварийное заверше-

ние выполнения программы с выдачей соответствующей диагностики. При выключенном контроле результат операции будет неправильным, но выполнение программы продолжится. Например, при выполнении программы

```
var x: integer; {$R-}
begin
  x:=30000; x:=x+x; writeln('x=',x)
end.
```

будет выдан ответ `x=-5536`.

Как уже говорилось, дискретная математика, в которой работает ЭВМ, существенно отличается от обычной. В частности, если коммутативный закон по сложению и умножению выполняется, т.е. всегда `a+b=b+a`, ассоциативный и дистрибутивный законы не выполняются, т.е. в общем случае `(a+b)+c` $\neq$ `a+(b+c)` и `a*(b+c)` $\neq$ `a*b+a*c`. Легко понять, что это происходит из-за того, что результат любой операции может выйти за допустимый диапазон. Кроме того, все действия с вещественными величинами производятся приближённо, т.е. результат операции округляется до ближайшего представимого вещественного значения.

Для того, чтобы более наглядно представить себе отличие дискретной математики от обычной, рассмотрим на вещественной прямой решение уравнения `x+A=A`. В обычной математике это уравнение имеет единственное решение x=0, а вот в дискретной могут существовать и отличные от нуля решения. Например, для типа `real` в языке Free Pascal для `A`=10^{10} это будет корень `x=0.156`, для `A`=10^{15} - корень `x=1024.0`, а для `A`=10^{20} - уже корень `x`=10^6. Так происходит потому, что после прибавления относительно маленького корня `x` к большой константе `A` результат округляется снова в `A`. Легко понять, что и все числа, меньше чем `x`, тоже будут корнями этого уравнение.

К сожалению, во многих учебниках по программированию на такие «не нужные тонкости» вообще не обращают внимания, что приводит в реальных программах к семантическим ошибкам, которые трудно выявляются при отладке.

Далее должен следовать важный этап спецификации задачи. На этом этапе требуется рассмотреть все случаи входных данных, для которых необходимо решить задачу, и выделить те из них, для которых решение имеет какую-либо специфику. Для всех таких особых случаев следует строго определить, какие данные программа получает в качестве входных и что она должна выдать в результате своей работы.

В качестве иллюстрации важности спецификации рассмотрим следующую простую задачу для реализации на Паскале. Пусть необходимо ввести одно целое число в переменную x оператором `read(x)`, затем присвоит переменной `y` значение `x+1` и вывести полученное значение `y`. Ясно, что чаще всего для этого на Паскале будет написана такая программа.

```
var x,y: integer;
begin
  read(x); y:=x+1; Writeln(y)
end.
```

3. Формальная правильность программы

А теперь зададимся естественным вопросом, когда эта программа будет давать правильный ответ? Ясно, что для того, чтобы эта программа дала правильный ответ, необходимо наложить определённые ограничения на её входные данные. Эти ограничения можно записать в виде некоторого логического условия от входных данных, в теории программирования это логическое условие называется предусловием программы [1]. Какое же предусловие будет в нашем случае?

Во-первых, необходимо потребовать, чтобы в стандартном входном потоке данных `input`, к которому обращается оператор `read`, находилась правильная лексема целого числа, причём после преобразования её во внутреннее представление, это целое число принадлежало к диапазону представимых в нашей Паскаль-машине целых чисел. Обозначим этот диапазон буквой `Z`, по аналогии с обозначением в математике множества всех целых чисел, тогда это требование предусловия можно записать как $\texttt{x} \in \texttt{Z}$.

Далее, надо потребовать, чтобы величина `x` была меньшей, чем самое большое представимое в Паскале целое число, которое, как известно, имеет стандартное имя `MaxInt`. Теперь можно написать предусловие нашей программы в виде

$$\texttt{Pred} = \{\ \texttt{x} \in\ \texttt{Z}\ \wedge\ \texttt{x<MaxInt}\ \}$$

Когда предусловие истинно, то программа должна выдать правильный ответ, что можно записать в виде логического постусловия программы.

$$\texttt{Post} = \{\texttt{y} \in\ \texttt{Z}\ \wedge\ \texttt{y=x+1}\ \}$$

[1] Более точно оно называется слабейшим предусловием [3]

Дадим теперь определение того, когда (не имеющая синтаксических ошибок) программа является (семантически) правильной. Сначала поймём, что правильность или неправильность программы зависит от её предусловия. Например, приведённая выше программа для предусловия

```
Pred = { x ∈  Z  }
```

будет неправильной, так как для введённого x=MaxInt правильного ответа не получиться, т.е. постусловие не будет выполняться. Предусловие, сам алгоритм и постусловие носят название триада Хоара [2].

Итак, будем называть программу правильной в некотором предусловии, если при истинности этого предусловия программа завершится и обязательно обеспечит истинность нужного нам постусловия.[3] Здесь можно провести аналогию с правильностью работы завода: при соответствии входного сырья заданным стандартам, правильно работающий завод должен обеспечит выпуск продукции, тоже удовлетворяющей заданным стандартам, иначе завод работает неправильно.

В том случае, если предусловие не выполнено, программа, вообще говоря, не обязана обеспечить правильность постусловия. Другими словами, в этом случае программа может зациклиться, аварийно остановиться (ошибка времени выполнения – Run Time Error, по-русски АВОСТ), или же выдать неверный результат. По аналогии с заводом, при получении им плохого сырья он не обязан производить хорошую продукцию, а может выдать брак или повести себя и совсем нехорошо (например, взорваться ☹).

Как Вы догадываетесь, такое неправильное поведение программы не обрадует заказчика, который поручил программисту написать эту программу. Исходя из этого, с точки зрения нашего заказчика, лучшим предусловием программы было бы предусловие

```
Pred = True
```

Такое предусловие следует известному принципу хорошего обслуживания «Клиент всегда прав». В этом предусловии написанная выше программа, конечно же, будет неправильной. Разберёмся сначала, а какое постусловие должно быть у нашей программы в таком, тождественно истинном, предусловии. Естественно, оно тоже должно быть тождественно истинным, т.к. все входные данные по опреде-

[2] Используются элементы аксиоматической семантики Т.Хоара. Рассматривается применение предусловий и постусловий только ко всей программе целиком, а не к составляющим её операторам [1, 7].

[3] В научной литературе в этом случае говорят о полной правильности программы. Частичная правильность подразумевает, что если программа завершится, то постусловие будет выполнено.

лению «хорошие». Можно, например, для этого случая предложить следующее постусловие

$$\texttt{Post} = \{\ \texttt{x} \in \texttt{Z}\ \wedge\ \texttt{x<MaxInt} \Rightarrow\ \texttt{y} \in\ \texttt{Z}\ \wedge\ \texttt{y=x+1} \vee \texttt{x} \notin \texttt{Z}\ \Rightarrow \texttt{«Плохое x»} \vee \texttt{x} \in \texttt{Z}\ \wedge\ \texttt{x=MaxInt} \Rightarrow \texttt{«Большое x»}\ \}$$

Как Вы можете убедиться, данное постусловие является тождественно истинным, так как для любого значения, введённого из входного потока, одно из трёх логических слагаемых обязательно будет истинным. Как можно догадаться, написать программу для решения задачи в тождественно истинном предусловии будет совсем не так просто. В то же время, очевидно, что «настоящие» программы должны быть написаны именно в таком предусловии (как уже говорилось, «Пользователь всегда прав»). У хороших программ не должно быть зацикливаний, выдачи непонятных диагностик и «синего экрана смерти».

В качестве еще одного примера рассмотрим такую задачу. Необходимо ввести 100 вещественных чисел и вывести их сумму. Обычно студенты осознают, что для решения этой задачи не надо описывать в программе массив для хранения вводимых чисел и предлагают такой вариант программы для решения задачи (эта же программа приведена и в большинстве учебников по программированию):

```
const N=100;
var i: integer; x,sum: real;
begin
  sum:=0;
  for i:=1 to N do begin
    read(x); sum:=sum+x
  end;
  Writeln(sum)
end.
```

Когда (т.е. в каком предусловии) эта программа является правильной? На первый взгляд, можно предложит такое предусловие для этой задачи, в котором написанная программа будет правильной:

$$\texttt{Pred} = \ \{\ \forall \texttt{x} \in \texttt{1..N} \mid x_i \in \texttt{R}\ \wedge \sum_{i=1}^{N} x_i \in \texttt{R}\ \}$$

Необходимо осознать, что в таком предусловии написанная программа неправильная. Действительно, в процессе суммирования ча-

стичная сумма может выйти за допустимый диапазон представимых целых чисел, и правильного ответа не получиться. При выполненном предусловии это может случиться, например, если сначала вводятся в основном положительные числа, а затем отрицательные, так что общая сумма принадлежит Z, но не будет получена программой!

На самом деле написанная выше программа верна в таком предусловии:

$$\texttt{Pred} = \ \{\ \forall \texttt{x} \in \texttt{1..N} \mid x_i \in \texttt{R} \ \wedge \ \forall j \le N \ \sum_{i=1}^{j} x_i \in \texttt{R}\ \}$$

Другими словами, допустимой должна быть любая частичная сумма. Это предусловие, конечно, для заказчика не очень хорошее, так как многие представимые суммы программой не будут получены. Как же, однако, написать такую программу в хорошем первоначальном предусловии? Необходимо понять, что без использования массива, в котором будут храниться все введённые числа, при этом уже не обойтись.[4]

Новая программа сначала должна будет ввести все числа в некоторый массив, а затем упорядочить его по не убыванию. После этого надо организовать «встречное» суммирование, двигаясь одновременно от начала к концу и от конца к началу массива. Ниже приведена возможная программа (она не будет выдавать диагностику при выходе конечной суммы за допустимый диапазон).

```
const N=100;
type Mas=array[1..N] of integer;
var sum,i,j,temp: integer; x: Mas;
begin
  for i:=1 to N do Read(x[I]);
{ простейшее упорядочивание массива по неубыванию }
  for i:=1 to N-1 do for j:=i+1 to N do
    if x[j]<x[i] then begin
      temp:=x[i]; x[i]:=x[j]; x[j]:=temp
    end;
{ Встречное суммирование }
  sum:=0; i:=1; j:=N;
  while i<=j do
```

[4] Если вводимых чисел не очень много, и они помещаются в массив, то использования для хранения этих чисел других структур данных (файлов, списков и т.д.) не рассматривается, как крайне неэффективное.

```
    if sum>0 then begin sum:=sum+x[i]; inc(i) end
             else begin sum:=sum+x[j]; dec(j) end;
  Writeln(sum)
end.
```

Большинство языков программирования не позволяют записывать в программе предусловие и постусловие иначе, чем в виде комментария. В некоторых языках, однако, можно использовать специальные логические утверждения, истинность которых должна проверяться во время счета программы. Например, можно упомянуть язык Эйфель [4], в котором предусловие и постусловие являются директивами для соответствующего исполнителя и могут учитываться при выполнении программы. Из достаточно широко используемых языков можно назвать Free Pascal и Python [5] с директивами-утверждениями `assert` и язык C++ [6] с директивами `expect` и `ensures`.

Приведённые примеры должны подчеркнуть важность правильной спецификации задачи. Можно сказать, что сутью спецификация является четкая постановка задачи. В некоторых книгах по программированию говорится, что необходимо построить математическую модель решаемой задачи, однако очевидно, что это относится в основном к физическим и техническим задачам.

Литература

1. Грис Д. Наука программирования. – М., Мир, 1984, 416 с.
2. Кетков Ю.Л., Кетков А.Ю. Свободное программное обеспечение. Free Pascal для студентов и школьников, БХВ-Петербург, 2011, 372 с.
3. Дейкстра Э. Дисциплина программирования. – М., Мир, 1978, 275 с.
4. G.J. Myers, Software Reliability: Principles and Practices, New York, John Wiley, 1976, p. 275.
5. Россум Г., Дрейк Ф.Л.Дж., Откидач Д.С. Язык программирования Python. – 2001, 454 с.
6. International Standard ISO/IEC 14882:2017(E) – Programming Language C++.

7. Хоар Т. Взаимодействующие последовательные процессы (Communicating Sequential Processes). – М., Мир, 1989, 264 с.

Ержанов Ж.К., Смелянский Р.Л.

Сетевое кодирование на основе $GF(3^m)$

Введение

Настоящая статья является постановочной, т.е. в ней мы постарались сформулировать новый подход к технологии сетевого кодирования на основе поля Галуа $GF(3^m)$ и сформулировать основные задачи, связанные с его применением. Основными достоинствами предложенного подхода являются отсутствие дублирования кадров и асинхронной доставки необходимой информации получателю, что снижает влияние блокировок на выходе коммутатора на его пропускную способность. Предложенный подход основан на сжатии данных без потерь, путем перехода к новому, более "компактному" алфавиту. В отличие от линейного сетевого кодирования, где несколько кадров одинаковой длины преобразуются в один пакет такой же длины, в рассмотренном методе, несколько пакетов путем сжатия преобразуются в меньшее количество кадров, но при этом кадры не дублируются.

1. Сетевое кодирование

Сетевое кодирование, Network Coding, является относительно новой технологией в построении компьютерных сетей. Оно используется для увеличения пропускной способности сети, её эффективности и масштабируемости. Суть этой технологии заключается в том, что вместо пересылки традиционных кадров на L2, узлы сети агрегируют несколько кадров в один, без увеличения размера агрегированного кадра. Промежуточные коммутаторы по нескольким полученным агрегированным пакетам однозначно восстанавливают исходные кадры [1]. Одним из наиболее показательных примеров применения сетевого кодирования является сеть типа "бабочка" [2]. Пусть два узла пытаются передать кадры A и B двум конечным узлам (см. рис. 1). Они передают по одному кадру конечным узлам (A слева и B справа), а также передают по одному кадру в один промежуточный узел. Вместо поочередной отправки кадров, и потери времени, промежуточный узел применяет к входным кадрам побитовую операцию исключающего или (XOR), и затем отправляет получившийся агрегированный кадр $A + B$. Узлы получатели, имея информацию об одном кадре и информацию об агрегированном кадре, могут однозначно восстановить второй кадр.

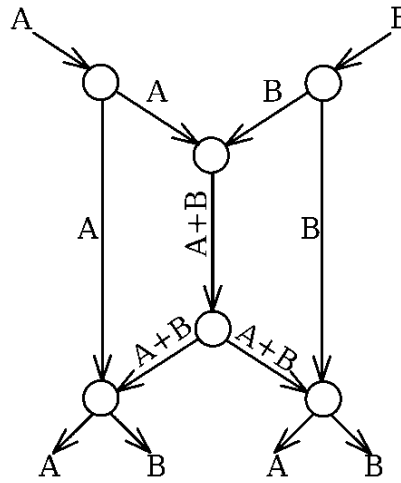

Рис. 1 Пример

Такой подход к агрегированию кадров (побитовое применение операции XOR) получил название линейное сетевое кодирование. Этот метод достаточно хорошо изучен и имеет некоторое количество преимуществ и недостатков. Достоинствами техники, предлагаемой в этом подходе, являются повышение пропускной способности, защита от атак типа Wiretapping, использование естественных свойств беспроводных технологий, а также сокращение задержки на выходе коммутатора.

Недостатками такой техники сетевого кодирования являются дополнительная вычислительная нагрузка на коммутаторы сети, большая уязвимость перед скомпрометированными коммутаторами, сложность внедрения в существующую архитектуру, а также проблемы дублирования и задержки из-за асинхронной передачи кадров в сети. В линейном сетевом кодировании, для успешного восстановления исходного кадра, получатель должен иметь информацию об этом кадре “внутри” нескольких других кадров. Иными словами, в сетевом линейном кодировании неизбежно дублирование (одновременное нахождение внутри сети) одних и тех же кадров в том или ином виде. В добавок к этому, кадры будут приходить в приемник асинхронно, разными путями, а значит устройство будет ждать, пока не получит все нужную для восстановления информацию.

2. Поле Галуа порядка q^n

Полем называется множество F, с заданными в нем замкнутыми (результатом применения операции к любым двум элементам F также будет элемент F) операциями "умножения" $(*)$ и "сложения" $(+)$. Причем это множество вместе с операцией $+$ является абелевой группой (которая называется аддитивной группой поля), а множество $F/\{0\}$, где 0 – нейтральный элемент в аддитивной группе, вместе с операцией $*$ является коммутативной группой (которая называется мультипликативной группой поля). Нейтральный элемент в аддитивной группе называется нулевым, а в мультипликативной – единичным. В добавок к этому, выполняется свойство дистрибутивности умножения относительно сложения:

$$(a+b)*c = (a*c)+(b*c), \forall a,b,c \in F$$

Поле, имеющее конечное число элементов, называется полем Галуа, а количество элементов в этом поле называется его порядком. Поля Галуа имеют порядок вида q^n, где q – простое число, а n – натуральное число (всюду далее q – простое, а n – натуральное). Поле Галуа порядка q^n обозначается как $GF(q^n)$. Такие поля обладают тем свойством, что мультипликативная группа поля является циклической (все элементы группы являются степенями некоторого элемента группы), а все различные поля одинакового порядка равны с точностью до изоморфизма [3].

Удобно элементы полей Галуа $GF(q^n)$ представлять как многочлены степени $n-1$ с коэффициентами из множества $\{0,1,\ldots,q-1\}$. В таком представлении, нулевым элементом является многочлен 0, а единичным – многочлен 1. Сложение и произведение определены, как обычные сложение и умножение многочленов по модулю некоторого неприводимого (неразложимый на неконстантные многочлены) многочлена степени n. При этом сложение и умножение элементов при одинаковых степенях производится по модулю q. Все поля Галуа одинакового порядка равны с точностью до изоморфизма, а значит выбор примитивного многочлена не имеет столь высокой роли в представлении элементов [3]. Несмотря на это, есть некоторые многочлены, называемые примитивными, по модулю которых, операция умножения производится гораздо быстрее [3].

Поля Галуа находят большое применение в телекоммуникации. Их используют для построения кодов, исправляющих и обнаруживающих ошибки. В основе их применения, для построения корректирующих кодов лежит следующая идея: в исходное сообщение вносится некоторая избыточность в виде дополнительных символов, что-

бы при передаче сообщения можно было обнаружить, возникли ли ошибки. Эту избыточность вводят, чтобы все правильные кодослова (исходная последовательность символов вместе с избыточными символами) находились друг от друга на определенном расстоянии, в смысле Хэмминга. Если кодослово отстоит от правильных кодослов на ненулевое расстояние, то это признак ошибки. В определенных случаях (конструкциях кодослов) – это расстояние позволяет определить ошибочные символы в кодослове и то, на какие символы их надо заменить [4].

Простейшим примером кода, обнаруживающего единичную ошибку в двоичном сообщении, является добавление бита четности. То есть, если количество единичных бит в исходном сообщении является нечетным, то значением бита четности выставляется единица, иначе ноль. Таким образом, в передаваемом сообщении количество единиц всегда кратно двум. Пусть, к примеру, в одном, некотором месте передаваемого сообщения ноль поменялся на единицу, или наоборот. Тогда суммарное количество единиц в сообщении перестанет быть четным, а значит при передаче произошла ошибка.

Простейшим примером кода, исправляющего единичную ошибку, может послужить добавление двух копий исходного сообщения. То есть вместо сообщения *abc* будет передано сообщение *abcabcabc*. Таким образом, если в отправленном сообщении на какой-нибудь позиции возникнет ошибка, то посмотрев на соответствующие позиции в двух копиях можно будет восстановить исходное сообщение.

Наиболее популярными кодами, исправляющими и обнаруживающими ошибки, являются так называемые циклические коды. Такие коды строятся на основе полей Галуа, благодаря свойству цикличности мультипликативной группы поля порядка q^n. За счет тесной связи с конечными полями, процесс кодирования и декодирования производится относительно легко, а количество избыточной информации, для некоторых видов кодирования, достигает теоретического минимума [5].

3. Троичная система счисления

В данной работе предлагается использовать сжатие данных, основанное на переходе к другому, более компактному алфавиту. В информационных технологиях, для представления данных традиционно используется двоичная запись. Любое число, оказываясь в памяти компьютера, приводится к записи в двоичной системе счисления. Такая запись имеет ряд достоинств; в их числе простота реализации (1 – есть напряжение, 0 – нет напряжения), а также простота

осуществления арифметических операций. Но оказывается, что по части "компактности", как будет показано ниже, двоичная система уступает троичной.

Для оценки той или иной системы счисления в качестве основы для конструирования вычислительной машины имеет значение то, что обычно называют экономичностью системы. Под этим понимается тот запас чисел, которые можно записать в данной системе с помощью определенного количества знаков [6].

Например, можно представить числа от 0 до 999 используя 30 знаков в десятичной системе счисления (цифры $0-9$ на каждую из 3-х позиций). Таким образом, в десятичной системе счисления, имея 30 знаков, можно представить 1000 различных чисел. В двоичной системе, используя те же 30 знаков, можно представить $2^{15} = 32768$ различных чисел. В троичной же системе, используя тот же запас знаков, можно представить уже $3^{10} = 59049$ различных чисел. Получается, что для 30 знаков, двоичная система экономичнее десятичной, а троичная система, в свою очередь, экономичнее двоичной.

В общем случае такое отношение сохраняется для любого количества знаков. Троичная система доказано [6] является не только экономичнее двоичной и десятичной системы счисления, но и любой другой системы счисления. Этот факт делает троичную систему счисления очень выгодной для механизмов сжатия сообщений, где для хранения чисел используется тот алфавит, в котором запись числа наиболее компактна. Таким алфавитом является троичная система счисления.

4. Предлагаемый метод

Приведенные выше рассуждения можно применить к системам передачи данных. А именно, чем эффективнее (в указанном выше смысле) система кодирования чисел, тем больше данных можно передать по каналу, при одной и той же скорости пакетизации (скорость выталкивания очередного символа в среду передачи данных).

Если мы рассмотрим известные способы организации коммутаторов, то увидим, что наиболее эффективным является коммутатор с буферизацией на выходе (минимальная задержка при загрузке до 90%). Нет блокировок кадров, но высокие требования к объему и скорости буферной памяти [4].

Предположим, что на одном и том же выходе коммутатора появилось n кадров длины l бит. Соберем все кадры в буфере в битовую матрицу $n \times l$. Каждый столбец этой матрицы можно однозначно представить, как элемент поля $GF(2^n)$. Действительно, всего воз-

можно 2^n различных комбинаций битовых столбцов высоты n, что равно порядку поля $GF(2^n)$. Далее, построим инъективное отображение из $GF(2^n)$ в $GF(3^m)$. Свойство инъективности влечет, что из любого полученного в результате преобразования значения можно будет однозначно восстановить прообраз. Очевидно, что при таком отображении, порядок $GF(3^m)$ должен быть больше, чем порядок $GF(2^n)$, а значит $m \geq log_3 2^n = n * log_3 2$.

Заметим, что нижняя оценка для m является не целым числом, а потому минимальным возможным значением для m является ближайшее сверху целое. В таком случае образуется некоторый "избыток", так как не все элементы $GF(3^m)$ имеют прообраз в $GF(2^n)$. Этот избыток может использоваться для передачи дополнительных сообщений, для исправления или обнаружения ошибок, для передачи управляющей информации и т. д.. В этой статье предлагается применять кодирование, исправляющее ошибки, и, соответственно, технику Forward Error correction (FEC).

Рассмотрим пример. Пусть $n = 8$, получим, что порядок $GF(2^n)$ равен 256. Тогда $m \geq 8 * log_3 2 \approx 5.05$. Если взять $m = 6$, то порядок $GF(3^m)$ равен 729, что дает избыток в, примерно, 2.85 раз. Чтобы преобразовать столбцы исходной матрицы в элементы $GF(2^n)$, воспользуемся представлением элементов $GF(2^n)$ и $GF(3^m)$ в виде многочленов. Можно положить, что верхний элемент столбца соответствует коэффициенту при старшей степени $n-1 = 7$ многочлена, второй сверху элемент соответствует коэффициенту при степени $n-2 = 6$, и т. д. Аналогично, после преобразования $GF(2^n) \to GF(3^m)$ многочлены, представляющие элементы $GF(3^m)$, можно представить в виде столбцов, в которых верхний элемент соответствует старшей степени многочлена, второй сверху элемент соответствует коэффициенту при степени $m-2 = 4$, и т. д. Таким образом, входная матрица размера $8 \times l$ преобразована в матрицу размера $6 \times l$. При одинаковой скорости пакетизации, получим выигрыш в скорости освобождения буфера $8/6 \approx 1.33$ раза.

Приведем пример того, как можно использовать избыточность. В предыдущем примере избыточность была более чем в 2 раза, а 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.

Заметим также, что в предыдущем примере m было выбрано равным 6. Если выбрать m равным 7, то избыточность будет в 3 раза больше, чем при m равном 6: $3^7/2^8 \approx 8.54$. При этом все еще имеется выигрыш в скорости освобождения буфера, хоть и меньше. В этом

случае, с каждым столбцом в добавок к биту данных может быть передан еще 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 m зависит от числа n и от внешних факторов.

Еще одним важным замечанием является предположение об одинаковой длине всех кадров. Это предположение не всегда выполняется, что может вызвать проблему. Решением может быть приведение всех кадров к одинаковой длине, заполняя все недостающие позиции, например, нулями. При этом, для получения оригинального размера кадра будет использоваться информация о длине из заголовка кадра или дополнительное, закодированное в избытке сообщение.

На основе сказанного, предлагаемый метод можно сформулировать следующим образом:

- В зависимости от внешних факторов и информации о коммутаторах в сети, выбираются значения для n и m
- В буфере выходного порта коммутатора собирается n кадров
- Кадры приводятся к одинаковой длине l
- Эти кадры преобразовываются в матрицу размера $n \times l$
- Для каждого столбца размера n производится преобразование вида $GF(2^n) \rightarrow GF(3^m)$ в столбец размера m, помимо исходного сообщения включающий дополнительное сообщение
- Полученная матрица размера $m \times l$ отправляется на физический уровень
- Устройство, принявшее полученную матрицу, применяет обратное преобразование $GF(3^m) \rightarrow GF(2^n)$ и однозначно восстанавливает исходные n пакетов.

5. Заключение

В работе был сформулирован новый подход, в рамках которого реализуется сжатие данных без потерь. Предложенный подход основан на технологии сетевого кодирования, а именно используется идея преобразования кадров в промежуточных узлах сети. Данный подход имеет ряд достоинств, которые стоит отдельно отметить. Во-первых, за счет перехода в новый алфавит, уменьшается число кадров, циркулирующих внутри сети. Во-вторых, уменьшаются затраты на повторную отправку кадров, 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 [7]. В добавок к этому, на устройства внутри сети будет налагаться дополнительная вычислительная нагрузка. Также недостатком остается сложность внедрения такого подхода в уже существующую инфраструктуру.

Предложенный метод оставляет несколько открытых проблем:

- Значения n и m 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.

- Нужно, сконструировать преобразование $GF(2^n) \to GF(3^m)$ таким образом, чтобы вычислительная сложность была минимальной.

- В работе предлагается использовать кодирование с исправлением ошибок, но большинство современных методов кодирования с исправлением ошибок предназначены для двоичных систем. Требуется исследовать методы кодирования в троичных системах и сравнить их.

- В представленном примере избыточность используется не полностью. Из 2.85 единиц избытка было использовано лишь 2 единицы в виде дополнительного бита. Возникает вопрос: существуют ли преобразования $GF(2^n) \to GF(3^m)$, использующие всю избыточность?

Литература

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

2. Ahlswede R. et al. *Network information flow.* IEEE Transactions on information theory. – 2000. – Т. 46. – №. 4. – С. 1204-1216.

3. Журавлев Ю. И., Флеров Ю. А., Вялый М. Н. *Дискретный анализ. Основы высшей алгебры.* М.: МЗ Пресс. – 2007.

4. Смелянский Р. Л. *Компьютерные сети. (в 2т.* М.: Академия. – 2011. - 304+240с.

5. Власов Е. Г. *Конечные поля в телекоммуникационных приложениях. Теория и применение FEC, CRC и M-последовательностей.* 2016.

6. Фомин С. В. *Системы счисления.* 1987.

7. Ильин В. А. *Математический анализ* Рипол Классик, 1979.

8. Xuebing W., Na Q., Yantao L. A. *Secure Network Coding System Against Wiretap Attacks* 2019 34rd Youth Academic Annual Conference of Chinese Association of Automation (YAC). – IEEE, 2019. – С. 62-67.

Иванов И.В., Антоненко В.А.

Разработка архитектуры эластичного приложения в среде легковесной виртуализации

Введение

В последнее десятилетие облачные вычисления набирают все большую популярность и привлекают к себе большое внимание как промышленных, так и научных кругов. Основным фактором, стимулирующим использование облачных платформ, является их способность предоставлять ресурсы в соответствии с потребностями клиента. Поэтому одним из основополагающих свойств облачных вычислений является эластичность, подразумевающая динамическую регулировку выделенных ресурсов для соответствия изменениям в рабочей нагрузки [1].

Для дальнейшего понимания работы введем несколько определений:

- Stateless - состояние приложения, не предполагающее обмен данными между его экземплярами [2].
- Stateful - состояние приложения, предполагающее необходимость обмена данными с другими частями приложения или его экземплярами [2].
- Горизонтальное масштабирование - представляет собой добавление или удаление экземпляров вычислительных ресурсов, связанных с приложением [3].
- Вертикальное масштабирование - подразумевает увеличение или уменьшение характеристик вычислительных ресурсов, таких как количество ядер процессора, количество памяти, пропускной способности сети [3].
- Миграция - перенос экземпляра приложения между частями инфраструктуры системы [4].
- Реплики - экземпляры приложения, запущенные на случай ошибки в работе экземпляра, реализующего работу приложения на постоянной основе [5].

- Проактивный подход к масштабированию - прогнозирует потребность в ресурсах в ближайшем будущем на основе уже имеющихся данных о работе приложения и заранее выделяет или освобождает ресурсы [3].

- Реактивный подход к масштабированию - предполагает использование пороговых значений для использования ресурсов, если приложение использует количество ресурсов превышающее или недостающее заданного порога, то число экземпляров приложения увеличивается или уменьшается соответственно [3].

Облачные провайдеры обычно используют подходы на основе виртуализации [6] для реализации инфраструктуры. Виртуализация позволяет запускать несколько операционных систем и несколько приложений на одном сервере одновременно. Она создает абстрактный уровень, убирающий необходимость во взаимодействии как с аппаратной, так и с программной рабочими средами. Облачные вычисления, в своей основе использующие виртуализацию, позволяют быстро развертывать необходимые для работы приложения ресурсы и управлять ими. Развертывание осуществляется посредством виртуальных машин(ВМ) и контейнеров. Виртуализация обычно реализуется с помощью гипервизоров. Виртуализация при помощи гипервизора позволяет нескольким операционным системам совместно использовать один аппаратный хост таким образом, чтобы каждая операционная система имела свои независимые ресурсы(наиболее известные гипервизоры: VMware ESX, KVM).

Виртуализация на основе контейнеров или легковесная вирутализация [8], является еще одним подходом к виртуализации, она основывается на концепции контейнера [7], в котором группа процессов по-прежнему имеет свою собственную выделенную систему, но на самом деле она работает в специально изолированной среде. Все контейнеры работают поверх одного и того же ядра. Далее в работе под виртуализацией будем понимать именно легковесный подход.

В научной литературе дано уже довольно много определений эластичности [9], [10], [11], в данной работе, определим эластичность как способность системы добавлять и удалять ресурсы (такие как ядра процессора, память, экземпляры виртуальных машин и контейнеров) динамически, для адаптации к изменению нагрузки в режиме реального времени. Существует два вида эластичности: горизонтальная и вертикальная. Горизонтальная эластичность представляет собой добавление или удаление экземпляров вычислительных ресурсов, связанных с приложением. Вертикальная эластичность подразумевает увеличение или уменьшение характеристик вычислительных

ресурсов, таких как время процессора, количество ядрер, количество памяти и пропускной способности сети. Основным отличием понятия эластичности от автоматического масштабирования является необходимость работы в реальном времени: эластичная система должна адаптировать инфраструктуру к изменяющейся нагрузке за определенный промежуток времени.

Согласно [12], эффекты масштабируемости видны пользователю с помощью значения пропускной способности. Эластичность же, а именно действия по изменению размера выделенных ресурсов, может быть невидимой для пользователя из-за скорости перевыделения ресурсов и частоты проверки соответствия предоставленных ресурсов. Влияние перевыделения на показатели производительности (время отклика, задержку) можно представить как количественную оценку эластичности. Время отклика системы (скорость ее реакции) очень важно, так как эластичность подразумевает адаптацию системы за определенный промежуток времени, называемый временем реакции. Время реакции - это интервал между моментом, когда было инициировано событие перевыделения и моментом когда модификация инфраструктуры была завершена.

В данной работе рассмотрены преимущества эластичности для разных архитектур приложений, развернутых на основе контейнерной виртуализации. На основе выявленных преимуществ разработка архитектуры приложения, позволяющей для определенного вида задач в полной мере использовать эластичное масштабирование ресурсов.

1. Эластичность контейнеров

Несмотря на то, что контейнеры набирают все большую популярность среди облачных провайдеров, на данный момент мало работ посвященных эластичности контейнеров. Существующие решения используют различные политики и методы, преследуют разные цели, имеют разные конфигурации и архитектуры. Приведем примеры, как механизмы, использующиеся для виртуальных машин, могут применяться к контейнерам.

В статье [14] предлагается проект системы, используемой для разработки и автоматического развертывания микроуслуг. В ней отслеживается количество запросов и загрузка памяти, когда они достигают определенного порога, контейнеры масштабируются. Для горизонтального масштабирования используются репликации контейнеров.

В [13] авторы предлагают архитектуру системы управления, ко-

торая динамически настраивает виртуальные машины и контейнеры. В этой работе контейнеры и виртуальные машины можно масштабировать как вертикально (изменяя количество вычислительных ресурсов, доступные для каждого экземпляра), так и горизонтально (изменяя количество экземпляров) в соответствии с целевой функцией, минимизирующей затраты вычислительных ресурсов.

В исследовании [15] предлагается структура под названием MultiBox. MultiBox - это средство для создания и миграции контейнеров. MultiBox использует механизм cgroups Linux для создания и переноса контейнеров. Контейнеры MultiBox поддерживают приложения как с сохранением состояния, так и без сохранения состояния.

DoCloud [3] - гибкая облачная платформа на основе Docker. Она позволяет добавлять или удалять контейнеры Docker для удовлетворения требований к ресурсам приложения. В DoCloud предлагается гибридный контроллер, использующий проактивный и реактивный подходы масштабирования. На данный момент почти все стратегии поддержания эластичности используют реактивный подход, основанный на предварительно определенных пороговых значениях. DoCloud использует метод динамического изменения пороговых значений, на основе прогноза, использующего предварительные данные работы приложения. Прогноз формируется гибридным контроллером, использующим реактивный и проактивный подходы совместно.

В статье Baresi [16] предлагается метод горизонтального и вертикального автоматического масштабирования с использованием контроллера обратной связи с дискретным временем. Подобная структура позволяет конфигурировать инфраструктуру и производить адаптацию платформы для различных приложений. В данной системе точно определены требования к запускаемым приложениям и метаданные используемые ими, чтобы система могла работать корректно.

В Таблице 1 приведены рассмотренные системы и три основных критерия, определяющих их преимущества. А именно:

- Подход масштабирования - реактивный или проактивный, очевидно что система обладающая возможностью заранее подготавливать ресурсы для приложения обеспечивает наиболее эффективное их предоставление.

- Тип масштабируемого приложения - под данным критерием будем понимать возможность системы обеспечивать эластичную работу как приложений, не подразумевающих обмен данными между экземплярами, так и приложений с необходимостью передачи данных между его частями или экземплярами (stateless

и stateful соответственно). Кроме того, для stateful приложений пользователю необходимо описать зависимости между частями приложения, для того чтобы провайдер, предоставляющий инфраструктуру, имел представление о том, как их масштабировать.

- Метод поддержания эластичности - метод взаимодействия системы с экземплярами приложения, это может быть горизонтальное масштабирование, вертикальное масштабирование и миграция контейнеров.

Проект	Подход масштабирования	Тип приложения	Метод поддержания эластичности
Kukade et. al. [14]	реактивный	stateless	горизонтальный
Hoenisch et. al. [13]	реактивный	stateless	горизонтальный вертикальный
Hadley et. al. [15]	реактивный	stateless stateful	миграция
DoCloud [3]	проактивный реактивный	stateless	горизонтальный вертикальный
Baresi et. al. [16]	реактивный	stateless	горизонтальный вертикальный

Таблица 1: Сравнение проектов по критериям.

Из вышеприведенных статей можно заключить, что хотя контейнеры можно масштабировать и горизонтально, и вертикально, для реализации всех механизмов, используемых для виртуальных машин, необходимы некоторые модификации. Например, в реактивных подходах продолжительность времени отклика - это период времени, оставленный для того, чтобы дать системе возможность достичь стабильного состояния после каждого решения о масштабировании, поскольку контейнеры очень быстро адаптируются к требованию рабочей нагрузки, продолжительность времени отклика должна быть небольшой по сравнению с ВМ. Из рассмотренных систем, лишь в DoCloud [3] используется проактивный подход для масштабирования контейнеров на основе прогнозирования ARMA (модель авторегрессии — скользящего среднего [17]). Также, для stateful приложений требуется отдельное описание пользователем зависимости между его частями, так как на данный момент нет возможности автоматически определять такие части. Дальнейшее исследование проводилось с опорой на сформулированные критерии эластичности: поддержка приложения с stateful состоянием, возможность масштабироваться, мигрировать и проактивно предсказывать нагрузку.

2. Эластичное приложение

В статье G. Toffetti [18] о самоуправляемом приложении описывается архитектура приложения, удовлетворяющего всем критериям эластичности, рассмотрим ее подробнее. Из проведенного в предыдущей главе обзора ясно, что самой большой проблемой является масштабирование stateful приложений (в которых состояние одного экземпляра зависит от другого) по горизонтали. Примером такого приложения может являться любое веб-приложение с базой данных. Если для работы с запросами мы можем развернуть необходимое количество контейнеров (которое позволяют нам ресурсы инфраструктуры) с обработчиками, то для базы данных подобный способ не сработает. Во-первых, копировать большой объем данных почти всегда невыгодно для пользователя, во-вторых, сильно возрастает сложность поддержания согласованного состояния базы данных параллельно работающими копиями [19]. На основе статьи [18] разработаем архитектуру приложения, способного самостоятельно масштабировать свои экземпляры и протестируем его работу в связке с системой DoCloud.

В статье [18] основной идеей является распределенная архитектура высокого уровня, которую можно использовать для реализации самоуправляемых облачных приложений. Идея состоит в том, что, как и надежные сервисы в облаке с использованием распределенных алгоритмов и компонентов, так и функции управления могут быть реализованы как устойчивые распределенные приложения. Авторы статьи используют современные распределенные хранилища (если конкретнее, то Etcd [20]) для хранения текущего состояния каждого приложения, кроме того использование распределенных хранилищ позволяет облегчить реализацию алгоритма консенсуса для выбора лидера и назначения функций управления узлам кластера. Таким образом, приложения переходят в состояние stateless и, если какой-либо из узлов управления выходит из строя, узел с тем же состоянием может быть перезапущен сразу же. Более конкретно, любые функциональные возможности управления (например, логика автоматического масштабирования) могут быть развернуты в качестве компонента stateless приложения, что позволяет приложению становится самоуправляемым. Если приложение или ВМ, на которой оно развернуто, выйдет из строя, функциональность самоуправления перезапустит его, и из распределенного хранилища оно получит последнее сохраненное состояние.

Функциональными возможностями, важными для эластичного самоуправляемого приложения являются обнаружение и настройка

компоенентов, мониторинг, управление экземплярами и автоматическое масштабирование. Чтобы реализовать эти возможности, обсудим требования, необходимые для согласованной работы системы.

Оркестрация. Предоставление пользователю возможности развертывания приложений в контейнерах есть по сути IaaS услуга (инфраструктура как сервис). Данный подход предлагает интерфейсы для развертывания и распределения ресурсов (вычислительных, хранения и сетевых), которые пользователь может использовать, когда ему это необходимо. Однако функциональность предоставления ресурсов по требованию была бы ограничена, если бы этот процесс нельзя было автоматизировать. Обычно приложения используют все виды ресурсов для реализации своей функциональности. Чтобы автоматизировать развертывания необходимых для них ресурсов необходимы специальные инструменты (Kubernetes, Openstack). Подобные инструменты по описанию приложения развертывают необходимый набор ресурсов. Автоматизация развертывания ресурсов называется оркестрацией [21]. На каждый запрос на запуск приложения в облачной инфраструктуре оркестратор запускает и настраивает необходимые ресурсы. В предложенной концепции автоматизация развертывания достигается за счет распределения логики оркестрации между экземплярами оркестровщика внутри каждого экземпляра приложения, таким образом каждый экземпляр имеет собственное представление о состоянии развернутого приложения и его компонентов.

Отдельно стоит обсудить проблему **конфигурации компонентов** и их зависимостей. Основная сложность заключается в динамически изменяющейся инфраструктуре облачных вычислений, на которой развернуты приложения. Контейнеры предоставляются динамически и их точки доступа (сетевые параметры: IP-адреса, порты) известны только после выделения ресурсов и запуска компонентов. Следовательно, невозможно заранее получить доступ к различным службам контейнера. Для решения этой проблемы в работе используется ранее представленное Etcd, с помощью встроенного в него алгоритма согласования (к примеру Raft15) обеспечивается отказоустойчивость и согласованность хранилища с данными точек доступа экземпляра приложения. Таким образом части приложения могут обнаруживать друг друга используя глобальную службу Etcd. Например, если внутри приложения будет развернут компонент балансировки нагрузки между его экземплярами, с помощью запроса к Etcd он будет способен получить данные о всех экземплярах приложения и переконфигурировать при надобности политику распределения нагрузки по данным между ними.

Для реализации функциональности автоматического масштабирования необходима функция **мониторинга** компонентов приложения, она также может быть предоставлена через глобальную службу Etcd. Обоснованием использования Etcd для мониторинга также может служить необходимость хранения информации не в частях приложения, что переводит его в состояние stateless. Основываясь на данных мониторинга, компонент автоматического масштабирования управляет горизонтальной масштабируемостью экземпляров приложения, поэтому основная сложность работы данного модуля - решить сколько необходимо развернуть экземпляров для обработки запросов поступающих приложению.

Кроме того, для обеспечения эластичного состояния приложения, модуль должен реагировать за определенный интервал времени на происходящие события, требующие увеличения или уменьшения количества экземпляров приложения. Обеспечение отклика модуля за определенное количество времени, так же достигается с помощью Etcd, которая способна передать его состояние на другой узел при сбое и предоставить данные мониторинга.

Целью эластичного приложения является поддержание состояния, максимального соответствующего текущему пользовательскому спросу, оно должно уметь масштабироваться и самовосставнавливаться за определенный промежуток времени. Таким образом, оно должно соответствовать следующей архитектуре, представленной на Рис.1

Также стоит упомянуть, что разработка приложения на основе эластичной архитектуры может позволить пользователю эффективнее использовать вычислительные ресурсы, что сэкономит его средства [18]. К тому же использование описываемой архитектуры позволит провайдеру автоматически определять зависимые части в stateful приложениях.

3. Экспериментальное исследование

Для экспериментального исследования авторы статьи [18] рассматривают веб-приложение Zurmo [22], ориентирующееся на управление взаимоотношения с клиентами(CRM). Возьмем для простоты исследования это же приложение, измененное авторами.

Zurmo (Рис. 2) - это PHP-приложение, использующее шаблон MVC [23] (добавлен фронтенд-контроллер, отвечающий за принятие/обработку входящих HTTP-запросов). Использует Apache веб-сервер, MySQL в качестве хранилища данных, Memcached - для кэширования данных. Это довольно типичное монолитное 3-уровневое

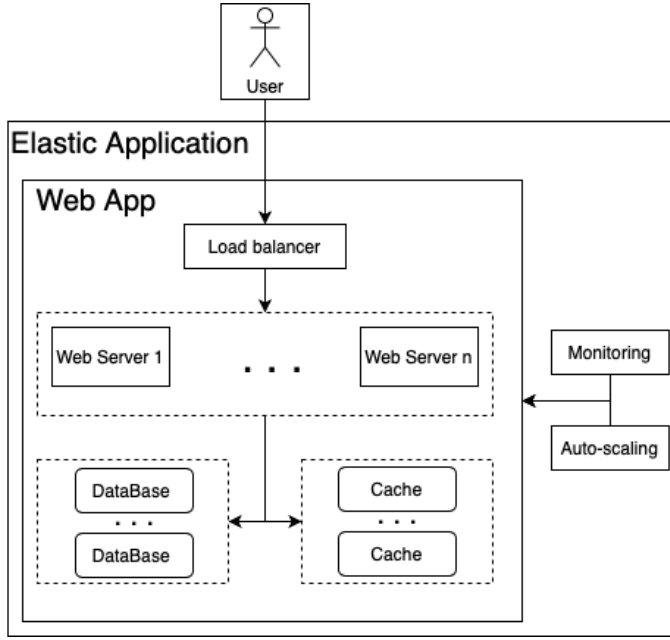

Рис. 1: Архитектура эластичного приложения, содержащая модули, необходимые для масштабирования и самовосстановления.

приложение с дополнительным уровнем кэширования. В качестве платформы для запуска приложения был выбран Kubernetes с встроенной в него Etcd (подробнее в экспериментальном исследовании).

Для достижения эластичного состояния приложения, будем использовать микросервисную архитектуру, позволяющую создать наиболее отказоустойчивую версию приложения [24]. Заимствуя идею из статьи [18] развернем части Zurmo в контейнерах: Apache, Memcached, Mysql). Эластичное состояние веб-части приложения достигается за счет балансировщика нагрузки, так как при масштабировании для корректной работы приложения нам лишь необходимо балансировать запросы между обработчиками. При этом будем сохранять состояние приложения и в кэш и в базу данных, таким образом если произойдет какой-либо сбой, мы сможем восстановить приложение и из кэша и из базы данных. Memcached изначально умеет масштабироваться горизонтально.

Автоматическое масштабирование. Для автоматического масштабирования будем использовать механизм Kubernetes - Horizontal Pod Autoscaler [25].

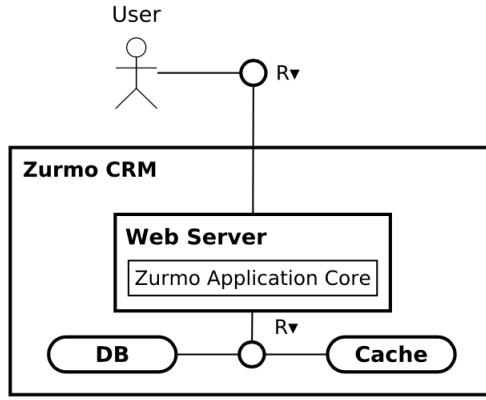

Рис. 2: Архитектура Zurmo.

Экспериментальное исследование проводилось на тестовом стенде, развернутом с помощью следующих технологий:

- CentOS - дистрибутив операционной системы Linux
- Docker
- Kubernetes(с Etcd)
- Zurmo

Эксперименты были направлены на демонстрацию того, что предлагаемая архитектура совместно с предложенными проектами соответствуют требованиям эластичности:

- отказоустойчивость
- способность горизонтально масштабироваться в сторону как увеличения так и уменьшения количества экземпляров

Кроме того, в экспериментах планировалось найти время отклика и определить его верхнюю границу, реализовав тем самым требование времени отклика системы. Эксперименты проводились с помощью искусственных отказов частей инфраструктуры приложения. Для тестирования автоматического масштабирования на части инфраструктуры с помощью Python-скриптов отправлялись GET-запросы.

Результаты экспериментов показали, что контейнеры с частями приложения успешно восставнавливаются после как внутренних сбоев (вызванных изменением кода), так и сбоев вызванных падением контейнера.

В таблице показаны время в секундах перезапуска разных типов контейнеров после их искусственного отключения.

Модуль	Попытка 1	Попытка 2	Попытка 3
Web server	15.243	22.847	20.602
Cache	38.190	40.928	46.723
DataBase	71.692	68.204	62.419

Таблица 2: Время перезапуска частей приложения.

При тестировании функциональности автоматического масштабирования создавались новые веб-части приложения. Время их создания примерно совпадает с средним временем перезапуска аналогичных частей приложения при искусственном отказе контейнеров. Время перезапуска частей приложения, связанных с хранением данных и поддержанием их актуального состояния для многих экземпляров перезапускаются дольше, так как происходит копирование данных и проверка их актуального состояния.

Проведенное экспериментальное исследование доказало, что рассматриваемое приложение отказоустойчиво и способно масштабироваться, а также удовлетворяет понятию эластичности и сформулированным критериям. Кроме того для каждой из частей приложения определено время перезапуска контейнеров. При этом говорить о каком-то унифицированном времени эластичности приложения (времени перезапуска контейнеров) не приходится, так как оно сильно зависит от роли, выполняемой конкретным контейнером.

4. Заключение

В данной работе дано определение эластичности, сформулированы его критерии. Проведен обзор существующих решений, поддерживающих эластичное состояние инфраструктуры. На основе самоуправляемого приложения из статьи [18] представлена архитектура эластичного приложения, удовлетворяющая сформулированным критериям эластичности. В основе ее реализации лежат механизмы распределенного хранения данных (в данной работе выбран Etcd), что позволяет преобразовать состояние приложения из stateful в stateless и последующе проводить горизонтальное масштабирование

всех его частей. В качестве будующей работы предлагается реализовать применение проактивного подхода предсказания необходимости масштабирования из платформы DoCloud [3].

Литература

1. Barnawi A. et al. *The views, measurements and challenges of elasticity in the cloud: A review* //Computer Communications. – 2020. – Т. 154. – С. 111-117.

2. Sang B. et al. *PLASMA: programmable elasticity for stateful cloud computing applications* //Proceedings of the Fifteenth European Conference on Computer Systems. – 2020. – С. 1-15.

3. C. Kan, *"DoCloud: An elastic cloud platform for Web applications based on Docker,"* in Proc. 18th Int. Conf. Adv. Commun. Technol., Jan. 2016, pp. 478–483.

4. Puliafito C. et al. *Companion fog computing: Supporting things mobility through container migration at the edge* //2018 IEEE International Conference on Smart Computing (SMARTCOMP). – IEEE, 2018. – С. 97-105.

5. *Swarm V. Fleet V. Kubernetes V. Mesos.* [Online]. Available: http://radar.oreilly.com/2015/10/swarm-v-fleet-v-kubernetes-v-mesos.html

6. Popek Gerald J., Goldberg Robert P. *Formal requirements for virtualizable third generation architectures* // Communications of the ACM. Vol. 17. 1974.

7. C. Pahl, *"Containerization and the PaaS cloud,"* IEEE Cloud Comput., vol. 2, no. 3, pp. 24–31, May/Jun. 2015.

8. Bernstein D. *Containers and cloud: From lxc to docker to kubernetes* //IEEE Cloud Computing. – 2014. – Т. 1. – №. 3. – С. 81-84.

9. L. Badger, T. Grance, R. Patt-Corner, and J. Voas, *"Draft cloud computing synopsis and recommendations,"* NIST Special Publication, vol. 800, 2011, Art. no. 146.

10. E. F. Coutinho, F. R. de Carvalho Sousa, P. A. L. Rego, D. G. Gomes, and J. N. de Souza, *"Elasticity in cloud computing: A survey,"* Ann. Telecommun.—Ann. des Telecommun., vol. 70, no. 7, pp. 289–309, 2015. [Online]. Available: http://dx.doi.org/10.1007/s12243-014- 0450-7

11. N. R. Herbst, S. Kounev, and R. Reussner, *"Elasticity in cloud computing: What it is, and what it is not,"* in Proc. 10th Int. Conf. Autonomic Comput., 2013, pp. 23–27.

12. M. Kuperberg, N. R. Herbst, J. G. von Kistowski, and R. Reussner, *"Defining and quantifying elasticity of resources in cloud computing and scalable platforms,"* Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany, vol. 16, 2011, http://digbib. ubka.uni-karlsruhe.de/volltexte/1000023476

13. P. Hoenisch, I. Weber, S. Schulte, L. Zhu, and A. Fekete, *"Fourfold auto-scaling on a contemporary deployment platform using docker containers,"* in Service-Oriented Computing. Berlin, Ger- many: Springer, 2015, pp. 316–323.

14. P. P. Kukade and G. Kale, *"Auto-scaling of micro-services using containerization,"* Int. J. Sci. Res., vol. 4, pp. 1960–1963, 2013.

15. J. Hadley, Y. El Khatib, G. Blair, and U. Roedig, *MultiBox: Lightweight Containers for Vendor-Independent Multi-Cloud Deployments.* Berlin, Germany: Springer Verlag, 2015, pp. 79–90.

16. L. Baresi, S. Guinea, A. Leva, and G. Quattrocchi, *"A discretetime feedback controller for containerized cloud applications,"* in Proc. 24th ACM SIGSOFT Int. Symp. Found. Softw. Eng., 2016, pp. 217–228.

17. Roy N.cite41, Dubey A., Gokhale A. *Efficient autoscaling in the cloud using predictive models for workload forecasting* //2011 IEEE 4th International Conference on Cloud Computing. – IEEE, 2011. – С. 500-507.

18. Toffetti G. et al. *Self-managing cloud-native applications: Design, implementation, and experience* //Future Generation Computer Systems. – 2017. – Т. 72. – С. 165-179.

19. Springer G. M., Ricker C. E., Pope N. G. *System and method for providing horizontal scaling of stateful applications* : заяв. пат. 13707094 США. – 2014.

20. *https://github.com/coreos/etcd, accessed 22 November 2020*

21. Liu C. et al. *Cloud resource orchestration: A data-centric approach* //Proceedings of the biennial Conference on Innovative Data Systems Research (CIDR). – 2011. – С. 1-8.

22. *https://github.com/zurmo/Zurmo, accessed 22 November 2020*

23. Cui W. et al. *The research of PHP development framework based on MVC pattern* //2009 Fourth International Conference on Computer Sciences and Convergence Information Technology. – IEEE, 2009. – С. 947-949.

24. Balalaie A., Heydarnoori A., Jamshidi P. *Microservices architecture enables devops: Migration to a cloud-native architecture* //Ieee Software. – 2016. – Т. 33. – №. 3. – С. 42-52.

25. *https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/, accessed 22 November 2020*

Кузьмин Я.К., Волканов Д.Ю., Скобцова Ю.А.

Исследование применимости алгоритмов обработки пакетов с сохранением состояния в архитектуре сетевого процессорного устройства RuNPU[1]

Введение

Основным функциональным элементом коммутатора является сетевое процессорное устройство (СПУ). СПУ — это специализированная интегральная схема, предназначенная для обработки сетевых пакетов.

В работе рассматривается программируемое СПУ RuNPU. Программируемые СПУ позволяют как загружать новые программы обработки пакетов, так и определять новые протоколы передачи данных [10]. Обработка пакета в СПУ происходит в соответствии с программой обработки пакетов.

Среди алгоритмов обработки пакетов существует класс алгоритмов, называемых алгоритмами обработки пакетов с хранением состояния. Состояние алгоритма обработки пакетов — это набор изменяемых переменных, значения которых сохраняются при переходе к обработке следующего пакета. Алгоритмы обработки пакетов, требующие хранения состояния, применяются в сетях центров обработки данных и в сетях телекоммуникационных провайдеров [2]. Примерами являются следующие алгоритмы: алгоритм балансировки транспортных потоков [3], алгоритм port-knocking [1], алгоритм быстрого восстановления потока пакетов после отключения канала [3]. Главной особенностью алгоритма обработки пакетов с хранением состояния является возможность введения зависимости процесса обработки пакета от свойств пакетов, обработанных данным СПУ ранее.

Рассмотрим алгоритм port-knocking. Он предназначен для открытия порта в межсетевом экране. Для того чтобы поток пакетов с определённого хоста был пропущен, требуется, чтобы он начинался с последовательности пакетов, имеющих определённые номера портов. До получения этой последовательности все пакеты, полученные с данного хоста, будут сброшены. Для этого для каждого обрабатываемого потока требуется хранить номер порта, ожидаемого в сле-

[1]Работа выполнена при частичной поддержке РФФИ, грант № 19-07-01076.

дующем пакете данного потока, и указание, должен ли этот пакет быть передан далее или сброшен. В процессе работы СПУ эта информация обновляется на основе обработанных пакетов.

В настоящее время активно развиваются программно конфигурируемые сети (ПКС) [11]. Основой технологии ПКС является размещение управляющих функций сети на отдельном сервере, называемом контроллером, и перенос функций управления сетью из сетевых устройств в контроллер. При таком подходе контроллер выполняет настройку сетевых устройств (определяет алгоритм обработки пакетов и параметры алгоритма), а сетевые устройства (коммутаторы) выполняют обработку и передачу сетевых пакетов на основе данных, полученных от контроллера.

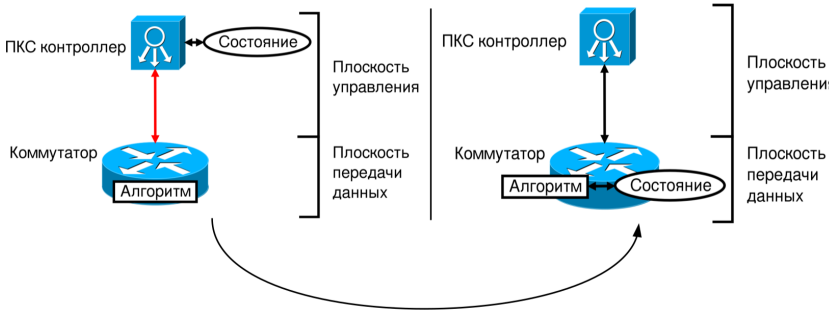

Рисунок 1. Проблема размещения состояния на контроллере.

В ПКС контроллер определяет используемый алгоритм обработки пакетов. Но для алгоритмов обработки пакетов с хранением состояния требуется обеспечить хранение состояния алгоритма. Если СПУ не имеет механизма хранения состояния алгоритма обработки пакетов, то задача хранения состояния возлагается на контроллер ПКС. Но состояние алгоритма обработки пакетов с хранением состояния изменяется в зависимости от пакетов, проходящих через СПУ. Если состояние будет хранить контроллер, то при обработке каждого пакета сетевое устройство будет обращаться к контроллеру для обновления состояния, что приведёт к снижению производительности сетевого устройства, вызовет сброс пакетов, замедление и нарушение работы сетевых сервисов (Рис. 1). При перемещении состояния алгоритма обработки пакетов в СПУ необходимость частых обращений к контроллеру для обновления состояния пропадает. Рассмотрим это утверждение в контексте алгоритма port-knocking. При хранении состояния на контроллере запрос к нему будет происходить каждый раз при получении пакета, относящегося к начальной последовательности.

Структура статьи следующая: раздел 1 описывает архитектуру СПУ RuNPU, 2 — постановку задачи, 3 — существующие подходы к хранению состояния алгоритма обработки пакетов. В разделе 4 предлагаются модификации архитектуры СПУ, разделы 5 и 6 посвящены описанию имитационной модели СПУ и экспериментального исследования.

1. Описание архитектуры СПУ RuNPU

Рассмотрим процесс обработки пакета данным СПУ (Рис. 2). После получения пакет попадает во входную очередь, формируются его метаданные. Метаданные представляют собой набор полей, содержащих информацию о пакете: размер пакета, размер заголовка, временную метку, номер входного порта и маску выходных портов.

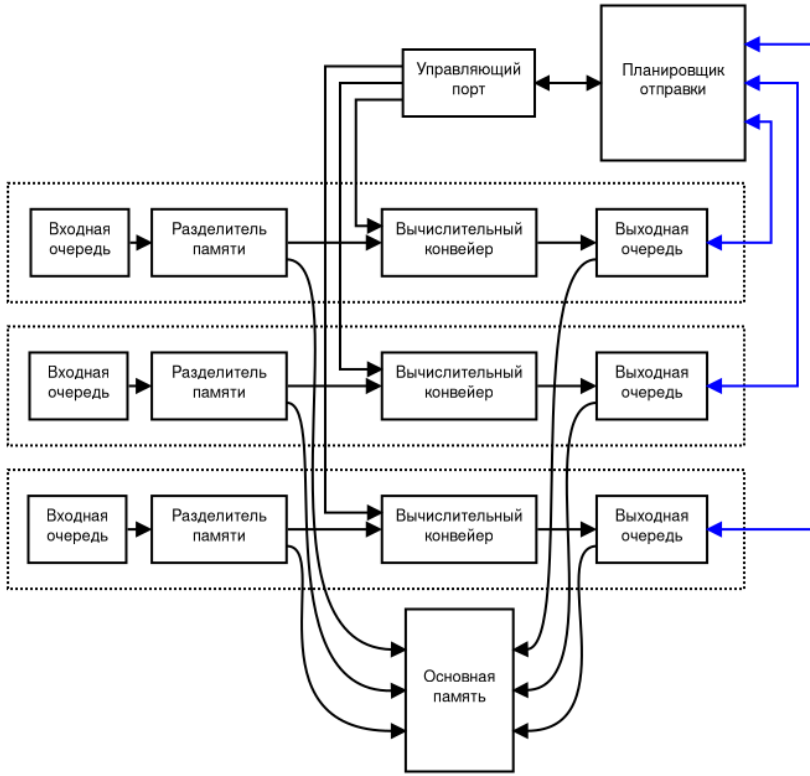

Рисунок 2. Структурная схема СПУ.

После извлечения из очереди пакет обрабатывается разделителем памяти: тело пакета записывается в основную память СПУ, а заголовок и метаданные направляются в вычислительный конвейер, соответствующий порту. Важно отметить, что вычислительные конвейеры изолированы друг от друга и работают параллельно. После завершения обработки заголовка конвейером заголовок помещается в выходную очередь заголовков, а метаданные отправляются в планировщик отправки, управляющий порядком отправки пакетов.

В ходе обработки конвейером заголовок пакета и метаданные последовательно проходят по стадиям конвейера (Рис. 3). Каждая стадия конвейера содержит RISC ядро и устройство памяти для хранения микропрограммы, заголовка обрабатываемого пакета и метаданных. Стадия имеет ограничение в 250 тактов на обработку заголовка.

Таблицы классификации в данной архитектуре представлены в виде деревьев поиска, представленных в коде стадии конвейера в виде инструкций. После завершения обработки пакета стадией конвейера заголовок пакета и метаданные передаются на следующую стадию, а вместо них загружаются данные заголовка и метаданные следующего пакета. При этом никакие данные, оставшиеся от обработки предыдущего пакета, не сохраняются в памяти стадии и не могут быть использованы при обработке следующего пакета.

В силу того, что в данной архитектуре СПУ не предусмотрены области памяти для хранения состояния, данная архитектура СПУ не позволяет применять алгоритмы обработки пакетов с хранением состояния.

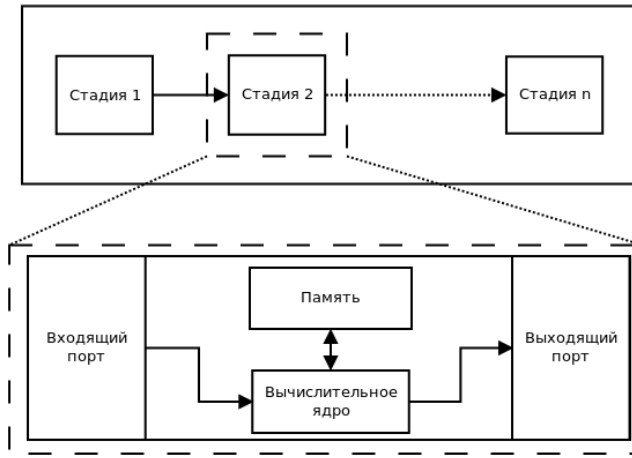

Рисунок 3. Схема конвейера СПУ.

2. Постановка задачи

В рамках исследования рассматривалась следующая постановка задачи: дана архитектура процессора RuNPU. Необходимо предложить модификации исходной архитектуры СПУ, обеспечивающие поддержку механизма хранения состояния алгоритма обработки пакетов без синхронизации состояния между портами СПУ. В силу ограничений рассматриваемой архитектуры к модификациям архитектуры предъявляются следующие требования:

1. Время обработки заголовка пакета не должно превышать 250 тактов.

2. Для каждой стадии каждого конвейера должен храниться отдельный набор переменных состояния.

В качестве целевого алгоритма рассматривается алгоритм port-knocking [1], требующий хранения состояния, но не требующий синхронизации состояния между портами СПУ.

3. Анализ существующих подходов к хранению состояния алгоритма обработки пакетов

3.1 Рассматриваемые подходы

В данном разделе рассмотрены как аппаратные, так и программные методы хранения состояния алгоритма обработки пакетов. Данный подход позволяет рассмотреть как особенности аппаратных методов хранения состояния алгоритма обработки пакетов, так и программные абстракции, используемые для представления состояния алгоритма обработки пакетов.

3.2 Аппаратные подходы

Были рассмотрены следующие подходы:

1. Хранение переменных состояния в специальных устройствах памяти, размещённых на стадиях конвейера [5].

2. Архитектура СПУ на основе атомов [8].

Данные подходы предполагают использование набора изолированных вычислительных ядер, расположенных на стадиях конвейера. Состояние алгоритма обработки пакетов представляет собой набор переменных, хранящихся в устройствах памяти на стадиях конвейера. Устройства памяти не связаны между собой. Первый подход предполагает завершение обработки заголовка пакета на стадии конвейера за фиксированное число тактов. Второй подход предполагает обработку заголовка на стадии конвейера за один такт за счёт использования массива вычислительных ядер, обрабатывающих заголовок параллельно.

3.3 Программные подходы

Рассмотрены следующие подходы:

1. Конечный автомат (реализация с двумя таблицами) [3, 7].
2. Конечный автомат (реализация с тремя таблицами) [9].

В данных подходах механизм представлен в виде конечного автомата. Состоянием алгоритма обработки пакетов является текущее состояние автомата, переход между состояниями осуществляется на основе характеристик обрабатываемых пакетов.

Данные подходы допускают возможность хранения отдельных переменных состояния для различных групп пакетов (например, транспортных потоков). Для этого предлагается использовать систему таблиц. Одна из таблиц системы, называемая таблицей состояний, используется для хранения текущего состояния каждой группы пакетов. Для изменения состояния используется таблица переходов. Данная таблица содержит правила, на основании которых по текущему состоянию и свойствам обрабатываемого пакета определяется новое состояние для данной группы пакетов. В зависимости от реализации, может применяться таблица действий, определяющая набор действий над пакетом в зависимости от его свойств и текущего состояния.

Аппаратная поддержка данного механизма может быть реализована аналогично реализации аппаратной поддержки для таблиц классификации. При этом для хранения таблицы состояний, для которой должна быть обеспечена возможность её изменения в ходе нормальной работы СПУ, может быть применено отдельное устройство памяти, находящееся на стадии конвейера.

Таким образом, целесообразно рассмотреть следующий подход: размещение на каждой стадии конвейера устройств памяти для хранения переменных состояния алгоритма обработки пакетов. Система

хранения состояния алгоритма обработки пакетов представляет собой конечный автомат, представленный в виде системы таблиц.

4. Модификации архитектуры RuNPU

4.1 Модификации стадии конвейера

Для хранения состояния на каждую стадию каждого конвейера добавляются устройства памяти [5] (Рис. 4). Оно располагается в том же адресном пространстве, что и основная память стадии конвейера. Устройства памяти, размещённые на разных стадиях, не имеют связи между собой. Также на каждую стадию конвейера добавляется блок Tree controller, его роль будет описана далее.

Для хранения переменного количества переменных состояния используется таблица состояний [3]. Примером подобной ситуации является хранение состояния для каждого потока, обрабатываемого данным СПУ.

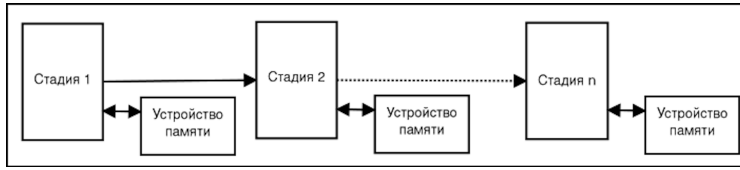

Рисунок 4. Схема конвейеров модифицированной архитектуры СПУ.

4.2 Представление таблицы состояний

Для представления таблиц классификации в данном СПУ применяются бинарные деревья поиска. Они представлены в виде микрокода и хранятся в основной памяти стадий конвейера. Из-за данной особенности СПУ не имеет возможности обновить данные таблицы самостоятельно, так как он получает обновления микрокода стадий конвейера через управляющий порт, соединённый с центральным процессором коммутатора.

Но к таблице состояний подобный подход неприменим, так как в ней хранятся текущие состояния потоков, так что обновления данных в ней происходят достаточно часто. Например, при открытии порта с использованием алгоритма port-knocking обновления происходят при обработке каждого пакета.

Предлагается следующий подход: таблица состояний представлена в виде бинарного дерева поиска, хранимого в устройстве памяти

для хранения состояния. Данное дерево поиска представлено в виде записей. Каждая запись содержит ключ, значение и адреса правого и левого поддеревьев. Для работы с таблицей в данном представлении используется блок Tree controller. Для управления этим блоком вводятся следующие инструкции:

1. **stsearch** — поиск по ключу, находящемуся в регистре процессорного ядра. Значение записывается в регистр, найденная запись помечается как текущая. Если запись не найдена, создаётся новая запись. В этом случае результатом, записанным в регистр, является ноль.

2. **stset** — запись нового значения в текущую запись. Новое значение считывается из регистра.

3. **stdel** — удаление текущей записи.

4.3 Использование алгоритмов обработки пакетов без хранения состояния

В том случае, если используемый алгоритм обработки пакетов не требует хранения состояния, он может быть выполнен на данной архитектуре без использования устройств памяти для хранения состояния. В этом случае работа данной архитектуры СПУ аналогична работе исходной архитектуры СПУ.

5. Описание разработанного программного средства

5.1 Имитационная модель СПУ

За основу взято программное средство, представляющее собой имитационную модель конвейера исходной архитектуры СПУ на языке программирования Python, позволяющее настраивать количество вычислительных ядер, оценивать энергопотребление и площадь кристалла [6]. В ходе работы имитационная модель выводит следующую информацию:

1. Среднее количество тактов на обработку пакета.

2. Пропускная способность.

3. Среднее энергопотребление.

В ходе модификации были добавлены следующие модули:

1. **State memory**. Данный модуль имитирует работу устройства памяти для хранения состояния.

2. **Tree controller**. Данный модуль имитирует работу блока Tree controller. В ходе его работы происходит расчёт количества тактов, потраченных на работу с деревом поиска.

Для конфигурирования имитационной модели были добавлены следующие параметры:

1. Размер памяти, используемой для хранения кода, заголовка и метаданных. Размер памяти для хранения состояния определяется как разность размера памяти стадии и размера памяти, используемой для хранения кода.

2. Описания инструкций stsearch, stset, stdel: их коды и описания параметров.

Также в ходе модификации был добавлен вывод максимального количества тактов, поторебовавшихся на обработку одного пакета.

6. Экспериментальное исследование

6.1 Цель экспериментального исследования

Целью экспериментального исследования является оценка количества тактов, требуемых для обработки пакета модифицированным СПУ в зависимости от количества потоков пакетов, обрабатываемых СПУ. В силу того, что в данном СПУ на обработку пакета выделяется фиксированное количество тактов, необходимо исследовать количество тактов, необходимых для обработки пакета алгоритмом, хранящим состояние каждого потока.

Помимо этого в ходе экспериментального исследования оценивается пропускная способность и энергопотребление СПУ.

6.2 Методика исследования

Экспериментальное исследование проводится для алгоритма port-knocking [1]. Для проведения исследования написана программа на языке ассемблера, реализующая алгоритм port-knocking.

Генерация экспериментального трафика происходила следующим образом: была записана TCP сессия, в ходе которой было передано

12 пакетов. Далее к данной последовательности были добавлены 4 пакета с номерами портов, соответствующими программе, реализующей алгоритм port-knocking. Далее на базе полученной последовательности пакетов было сгенерировано требуемое количество потоков с другими номерами портов. Полученные последовательности пакетов были объединены в одну последовательность путём поочерёдного добавления к итоговой последовательности пакетов из сгенерированных последовательностей.

В ходе экспериментального исследования производится запуск имитационной модели с различным количеством потоков в обрабатываемых пакетах. Размер памяти для хранения состояния составляет 32 КБ на стадию. Это значение даёт ограничение в 2000 состояний потоков, хранимых стадией в один момент времени.

6.3 Результаты исследования

Таблица 1. Результаты экспериментального исследования.

Количество потоков	Среднее число тактов на пакет	Максимальное число тактов	Энергопотребление, мВт	Пропускная способность, млн. пакетов/с
500	32	50	10814	30.4
1000	35	56	10812	28.5
1500	36	65	10812	27.2
2000	35	81	10812	27.9

Были проведены запуски имитационной модели для 500, 1000, 1500 и 2000 потоков. Результаты представлены в таблице 1.

Для рассмотренных случаев количество тактов, потребовавшихся для обработки пакета, не превышает максимально допустимого, равного 250 тактам. Это показывает применимость предложенного подхода. Среднее энергопотребление составляет 10,8 Вт, что совпадает с энергопотреблением исходной архитектуры СПУ. Средняя пропускная способность составляет 28.5 миллионов пакетов в секунду. При размере пакета больше 64 байт скорость передачи данных составит более 10 Гбит/с.

Заключение

В данной статье рассмотрена задача исследования применимости алгоритмов обработки пакетов с хранением состояния в ПКС, показана необходимость хранения состояния в СПУ. Предложены модификации архитектуры СПУ RuNPU, позволяющие применять алгоритмы обработки пакетов с хранением состояния. Проведено экспериментальное исследование модифицированной архитектуры, показавшее применимость предложенного подхода.

В силу отсутствия механизма синхронизации между устройствами памяти разработанная архитектура не позволяет применять алгоритмы обработки пакетов, требующие доступ к одним и тем же переменным состояния для обработки пакетов, полученных с разных портов. Таким образом, возможны дальнейшие исследования с целью разработки подобного механизма, что позволит расширить спектр алгоритмов обработки пакетов, реализуемых на данной архитектуре. Также в дальнейшем можно провести исследование для конкретных протоколов с хранением состояния, например, MPTCP [4].

Литература

1. Bianchi G. et al. *OpenState: Programming Platform-independent Stateful Openflow Applications Inside the Switch* // ACM SIGCOMM Computer Communication Review. - 2014. - Т. 44. - № 2. - С. 44–51.

2. Bifulco R., Retvari G. *A Survey on the Programmable Data Plane: Abstractions, Architectures and Open Problems* // International Conference on High Perfomance Switching and Routing (HPSR). – IEEE, 2018. – С. 1–7.

3. Cascone C. et al. *Traffic Management Applications for Stateful SDN Data Plane* // 2015 Fourth European Workshop on Software Defined Networks. - IEEE, 2015. - С. 85–90.

4. Ford A. et al. *TCP Extensions for Multipath Operation with Multiple Addresses draft-ietf-mptcp-rfc6824bis* // IETF RFC-6824. - 2016.

5. Kaushalram A. S., Budiu M., Kim C. *Data-plane Stateful Processing Units in Packet Processing Pipelines*: заяв. пат. 14864088 США. - 2017.

6. Markoborodov A., Skobtsova Y., Volkanov D. *Representation of the OpenFlow Switch Flow Table* // 2020 International Scientific and Technical Conference Modern Computer Network Technologies (MoNeTeC). - IEEE, 2020.

7. Pontarelli S. et al. *Stateful Openflow: Hardware Proof of Concept* // 2015 IEEE 16th International Conference on High Performance Switching and Routing (HPSR). - IEEE, 2015. - С. 1–8.

8. Sivaraman A. et al. *Packet Transactions: High-level Programming for Line-rate Switches* // Proceedings of the 2016 ACM SIGCOMM Conference. - ACM, 2016. - С. 15–28.

9. Zhu S. et al. *Sdpa: Enhancing Stateful Forwarding for Software-defined Networking* // 2015 IEEE 23rd International Conference on Network Protocols (ICNP). - IEEE, 2015. - С. 323–333.

10. Беззубцев С. О., Васин В. В., Волканов Д. Ю. и др. *Об одном подходе к построению сетевого процессорного устройства* // Моделирование и анализ информационных систем. - 2019. - Т. 26, № 1. – С. 39–62.

11. Смелянский Р. Л. *Программно-конфигурируемые сети* // Открытые системы, № 9, 2012, С. 15–26.

Машонский И.Д., Большакова Е.И.

Инструментальные средства извлечения терминов из текстов: разработка компонентов для русского языка

Введение

Быстрое развитие современных наук (компьютерных наук, медицины, биологии и др.) способствует непрерывному появлению новых терминов, употребляемых в текстах. Терминами считаются слова и словосочетания, обозначающие понятия предметной области (например, "нейронные сети", "стволовые клетки").

Термины используются исследователями как средство поиска научных статей в определенной области, но поиск усложняется для статей, описывающих результаты передовых исследований, в которых авторы вводят новые термины: эти термины могут быть малоизвестны научному сообществу. Из-за больших объемов текстовой информации, представленной в цифровых библиотеках, актуальной задачей является автоматическое обнаружение новых терминов, также важна задача систематизации накопленных знаний через построение пополняемых терминологических словарей и глоссариев. Для решения указанных задач необходимы методы и программные инструменты, позволяющие автоматически извлекать термины из текстов на естественных языках.

Задача автоматического извлечения терминов из текстов (automatic term extraction, ATE) изучается достаточно давно, предложено и изучено много методов ATE (см., например, [6, 8, 11]), а в последние годы появился ряд соответствующих программных инструментов, в большинстве своем созданных для англоязычных текстов [1, 5, 7–10, 12–14]. Реализованные в этих инструментах методы не учитывают особенности русскоязычных терминов (в первую очередь, морфологические) и поэтому могут показывать невысокую эффективность применительно к текстам на русском языке. В то же время известно сравнительно немного аналогичных работ для русскоязычных текстов (в частности, [9, 15–17]), в целом они пока не решают проблему языковой адаптации методов ATE.

В настоящей работе дается краткий обзор методов, применяемых для извлечения терминов из текстов, а также программных инструментов, в которых реализованы эти методы. На основе проведенного

нами анализа возможностей этих инструментов было предложено решение задачи автоматического извлечения терминов из текстов на русском языке — в виде дополнительных программных компонентов, реализованных на базе библиотеки atr4s [1]. Выбор этой библиотеки был обусловлен тем, что она превосходит другие инструменты по ряду критериев: функциональности, конфигурируемости, гибкости и расширяемости. Библиотека atr4s поддерживает многие методы ATE, а также имеет модульную структуру, что позволяет легко встраивать дополнительные компоненты, расширяющие ее базовые возможности. В работе кратко описываются архитектура и функции этой библиотеки и рассматриваются разработанные нами компоненты, которые расширяют ее возможности в части работы с русскоязычными текстами, включая распознавание терминологических вариантов, что впервые реализуется для русского языка.

1. Методы извлечения терминов

В работах по автоматическому извлечению терминов из текстов выделяют два основных вида признаков, с помощью которых определяется, является ли слово или словосочетание термином предметной области: лингвистические и статистические [11].

К лингвистическим признакам, в первую очередь, относятся грамматические образцы терминов, фиксирующие части речи слов термина и, реже, другие их морфологические характеристики. Так, терминами русского и английского языков чаще всего являются словосочетания, состоящие из двух существительных или прилагательного и существительного ("биосинтез белка", "небесное тело", "gene pool", "bynomial tree"). Лингвистические признаки не позволяют определять с приемлемой точностью, является ли слово или словосочетание термином, поэтому обычно они используются для выявления терминов-кандидатов, которые далее ранжируются с помощью значений статистических признаков (мер): кандидаты с наибольшими значениями мер обычно относят к терминам предметной области.

Кратко охарактеризуем наиболее эффективные и часто используемые статистические признаки. Наиболее простая в вычислительном плане мера — частота встречаемости слова или словосочетания в тексте, она опирается на предположение о том, что наиболее важные понятия предметной области имеют высокую частоту появления в текстах.

TF-IDF — известная из информационного поиска статистическая мера, которая учитывает появление терминов-кандидатов в текстовых документах некоторой коллекции: чем чаще термин-кандидат

появляется в рассматриваемом документе при низкой частоте употребления в других документах, тем выше значение TF-IDF и тем выше вероятность того, что термин-кандидат является термином в этом документе.

C-value [6] — мера для многословных терминов, учитывающая частоту встречаемости термина-кандидата в тексте, количество слов в нем и частоту его вложенности в другие термины-кандидаты. Чем больше частота встречаемости термина-кандидата в тексте и меньше его частота встречаемости в составе других терминов-кандидатов, тем больше значение C-value и, соответственно, вероятность того, что термин-кандидат является термином.

Различные меры ассоциации, такие как MI, T-score, Log-Likelihood ratio и др. (см., например, [11, 16]) — оценивают степень связи пары слов в составе двухсловного термина-кандидата. Значения таких мер зависят от соотношения частот встречаемости в тексте этих слов вместе и по отдельности. Некоторые меры (например, MI) могут выявлять термины, редко встречающиеся в текстах.

Ряд статистических мер вычисляются с помощью внешних текстовых ресурсов, в частности, в работе [1] используются тексты Wikipedia и учитываются частота встречаемости термина-кандидата среди заголовков ее статей, частота вхождения его в статьи той же тематики, что и исходный текст, и вероятность того, что термин-кандидат может входить в название гиперссылки на другую статью.

Обычно процесс извлечения терминов из текста включает следующие этапы:

1. Предобработка и лингвистический анализ текста: производится разбиение на токены, определяются части речи отдельных слов, извлекаются словосочетания.

2. Определение терминов-кандидатов на основе характерных для терминов грамматических образцов (правил, шаблонов).

3. Вычисление статистических признаков для отобранных терминов-кандидатов.

4. Ранжирование терминов-кандидатов по значениям статистических признаков (элементы из верхней части списка отранжированных кандидатов считаются терминами).

При решении задачи ATE следует учитывать также вариантность терминов [4]: аббревиатуры и терминологические синонимы, соответствующие одному и тому же понятию предметной области (например, "ИТ" и "информационные технологии", "алгебра логики"

и "булева алгебра"). Если не учитывать подобные терминологические варианты, то результаты вычисления статистических признаков будут искажены, что может негативно сказаться на результатах извлечения терминов из текстов.

2. Программные инструменты извлечения терминов

При рассмотрении программных инструментов для извлечения терминов из текстов [1, 5, 7–10, 12–14] в первую очередь нами учитывались следующие критерии: функциональность, конфигурируемость, расширяемость, гибкость.

Под функциональностью мы понимаем набор используемых лингвистических и статистических признаков терминов. Этот критерий важен из-за того, что невозможно выделить какой-либо один признак или их совокупность, который бы равно эффективно работал для любого текста или коллекции текстов произвольной предметной области [14]. Чем больше признаков поддерживает программный инструмент, тем шире его возможности по обработке текстов различных предметных областей. Общим для практически всех рассмотренных инструментов является применение грамматических образцов терминов, записанных обычно в виде регулярных выражений, как, например, образец *(JJ | NN)+ NN* в системе FlexiTerm [12], или в виде регулярных грамматик, как в TermoPL [7].

Что касается статистических признаков, то по их набору программные инструменты существенно отличаются. В большинстве инструментов используется частота встречаемости, TF-IDF, C-value. Система atr4s [1] использует меры на текстах Wikipedia, а TermExtractor [2, 9] и atr4s применяют методы тематического моделирования (LDA, PLSA). В инструментах JATE [14] и atr4s реализован большой набор дополнительных признаков (Basic, Relevance, RAKE — подробнее см. [1])

В нескольких системах ATE решается проблема вариантности терминов, но только для английского и французского языков. Так, FlexiTerm [12] реализует метод bag-of-words (мешок слов), при котором многословный термин-кандидат рассматривается как множество слов без учета порядка следования их в тексте. За счет этого, к примеру, термины-кандидаты вида "offshore wind turbine" и "wind offshore turbine" будут считаться одним и тем же термином (соответствовать одному понятию).

Конфигурируемость программного инструмента дает возмож-

ность более точно настроить его под конкретную задачу извлечения терминов, а расширяемость позволяет дополнительно встраивать в инструмент новые эффективные методы извлечения терминов. С этой точки зрения можно выделить системы atr4s [1], JATE [14] и TermRaider [8]: они имеют модульную структуру и предоставляют множество конфигурационных параметров для более тонкой настройки методов извлечения терминов, включая максимальное количество слов в термине, набор грамматических образцов терминов-кандидатов, минимально допустимую частоту встречаемости термина-кандидата в тексте (или текстовой коллекции), а также формат входных и выходных данных.

Большинство программных инструментов для ATE принимают на вход обычный текст (plain text), а на выходе выдают список извлеченных из него терминов. Некоторые системы допускают также документы форматов DOC (TermRaider[8]), PDF (Termine [6], TermRaider), HTML (Termine, TermRaider, FlexiTerm, fivefilters [5]).

Заметим, что по удобству применения в различном вычислительном окружении (гибкости) JATE и TermRaider уступают системе atr4s, так как реализованы на базе платформ Apache Solr и GATE соответственно, в то время как atr4s не является частью какой-либо инструментальной системы и может использоваться как самостоятельный инструмент извлечения терминов. Atr4s реализована как библиотека: это означает, что её компоненты можно встраивать в состав более сложных программных систем.

По совокупности рассмотренных критериев, среди рассмотренных систем ATE (ориентированных на английский язык), для построения системы ATE для текстов на русском языке, наиболее подходящей базой является библиотека atr4s [1] — расширяемая конфигурируемая библиотека на языке Scala, в которой реализована большая часть признаков для извлечения терминов из текстов.

3. Функции и архитектура библиотеки atr4s

Библиотека atr4s [1] имеет модульную структуру и включает следующие модули на языке Scala, которые применяются последовательно, формируя в совокупности систему извлечения терминов из англоязычных текстов:

- Модуль предобработки текста;
- Модуль извлечения терминов-кандидатов;

- Модуль ранжирования терминов-кандидатов.

Каждый модуль состоит из набора более мелких компонентов, выполняющих определенную задачу: к примеру, компонент, вычисляющий значение некоторого статистического признака в составе модуля ранжирования терминов-кандидатов.

Модуль предобработки текста выполняет чтение текста из заданной директории, разбивает текст на токены и для каждого слова выполняет определение части речи и лемматизацию. Встроено две реализации модуля предобработки текста для английского языка: на базе Apache OpenNLP (https://opennlp.apache.org) и на базе EmoryNLP (https://emorynlp.github.io/nlp4j).

Модуль извлечения терминов-кандидатов решает задачу распознавания в предобработанных текстах терминов-кандидатов. Для этого на первом этапе производится группировка токенов, полученных на этапе предобработки, в N-граммы. На втором этапе из полученного списка N-грамм удаляются те, которые либо содержат слова с длиной менее заданного значения, либо не удовлетворяющие заданному грамматическому образцу, либо содержат стоп-слова (предлоги, союзы, частицы, междометия, не входящие, как правило, в состав терминов), либо имеют частоту встречаемости в тексте ниже заданного порога. Полученные в итоге N-граммы считаются терминами-кандидатами. Конфигурационными параметрами модуля являются размеры N-грамм, порог частоты встречаемости, список стоп-слов, грамматические образцы терминов.

Модуль ранжирования терминов-кандидатов выполняет вычисление значений статистических признаков и упорядочивает термины-кандидаты на основе вычисленных значений. В библиотеке atr4s реализовано большое число статистических признаков, наиболее часто применяемых для ATE (C-value, Weirdness и др.), а также признаки на базе статей Wikipedia (Key Concept Relatedness) и на основе тематического моделирования (Novel Topic Model), всего 21 признак. Вычисляться может как значение только одного признака, так и значение нескольких (в последнем случае для принятия итогового решения о принадлежности терминов-кандидатов к терминам применяется алгоритм голосования или внешний метод машинного обучения).

Структура библиотеки дает возможность легко заменять реализацию отдельных её компонентов в составе модулей системы извлечения терминов. Выбор конкретной реализации компонентов определяется конфигурацией, которая задается в JSON-файле с названиями модулей, составляющих систему и с указанием используемых в них компонентов и их параметров. Этот JSON-файл передается на

вход системе извлечения терминов при её запуске. Например, для модуля предобработки текста в конфигурации можно указать, какой компонент из имеющихся использовать для определения частей речи (на базе OpenNLP или EmoryNLP).

Однако в библиотеке не реализованы средства распознавания вариантности терминов. Кроме того, в грамматических образцах может указываться только часть речи слов, а для извлечения русскоязычных терминов необходимо учитывать и другие морфологические характеристики (например, падеж, род и число).

4. Расширение библиотеки atr4s для обработки русскоязычных текстов

Для извлечения терминов из текстов на русском языке были разработаны на языке Scala и встроены в библиотеку atr4s следующие компоненты (программный код доступен на сайте https://github.com/ivan-mashonskiy/atr4s):

- Компонент предобработки текста, выполняющий его токенизацию, лемматизацию, определение части речи и других морфологических характеристик слов (падеж, род, число) на основе морфологического анализатора Mystem.
- Компонент, выявляющий термины-кандидаты среди словосочетаний предобработанного текста, с помощью грамматических образцов, ориентированных на русский язык (учитывающих нужные морфологические характеристики слов).
- Компонент, выполняющий распознавание вариантов русскоязычных терминов (заданных грамматических образцов).

Каждый из реализованных компонентов может быть указан в конфигурации системы извлечения терминов и использован вместо стандартных компонентов модулей atr4s. На Рисунке 1 показана общая архитектура системы ATE на базе atr4s и внесенные в неё изменения.

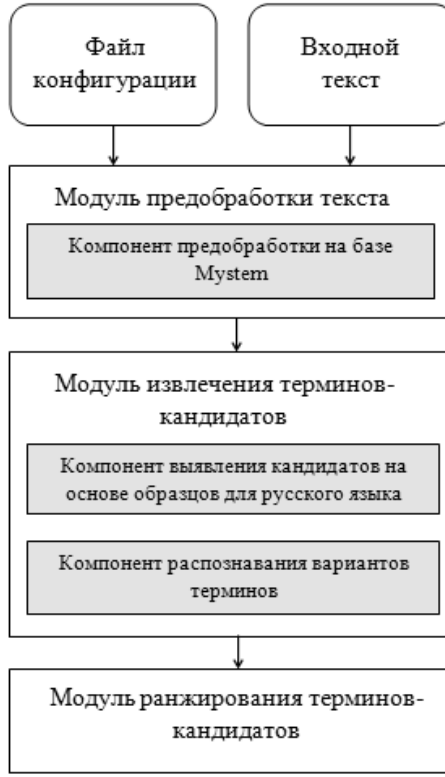

Рисунок 1. Общая архитектура системы извлечения терминов на базе atr4s

Компонент предобработки текста на базе Mystem

Mystem — свободно доступный модуль от Яндекс, выполняющий токенизацию входного текста на русском языке и морфологический анализ каждого полученного токена: лемматизацию, определение части речи и других морфологических характеристик. За счет этого в реализованном компоненте системы ATE поддерживается извлечение таких характеристик, как падеж, род и число для тех частей речи, которые имеют эти характеристики.

Для входного текстового документа реализованный компонент возвращает последовательность входящих в него слов, для каждого слова указывается результат лемматизации и морфологические теги (часть речи, падеж, род, число).

Так как рассматриваемый компонент предобработки базируется на Mystem, то для него доступны следующие опции конфигурации:

- кодировка ввода-вывода (cp866, cp1251, koi8-r, utf-8);
- возможность включения/отключения контекстного снятия омонимии;
- формат вывода (text, xml, json).

Компонент выявления терминов-кандидатов

В качестве параметра этот компонент принимает файл с набором грамматических образцов терминов и последовательно проверяет словосочетания из предобработанного текста на соответствие этим образцам. Словосочетание, удовлетворяющее хотя бы одному из заданных образцов, включается в итоговый список терминов-кандидатов.

В разработанных для компонента грамматических образцах учитываются морфологические характеристики (теги) слов (падеж, род, число), для записи образцов терминов была введена следующая нотация. Каждый образец состоит из трех частей, разделенных символом | . Первая часть содержит последовательность тегов частей речи слов термина, для различения одинаковых слов используются индексы. Например, последовательность тегов для термина "большой адронный коллайдер" будет описана как *A1 A2 S1* (*A* — прилагательное, *S* — существительное). Вторая часть опциональна (если она не нужна в образце, то заменяется символом #), в ней через запятую указываются падежи, которые должны иметь отдельные слова в составе словосочетания. Например, для термина "метод градиентного спуска", который удовлетворяет последовательности тегов частей речи *S1 A1 S2*, вторая часть образца имеет вид *A1=gen, S2=gen* (*gen* — родительный падеж). Третья часть образца также опциональна (в случае ее отсутствия ставится символ #), и в ней указываются слова словосочетания, которые должны быть грамматически согласованы (т.е. они должны иметь одинаковые морфологические теги падежа, рода, числа, при их наличии). Для термина "большой адронный коллайдер" третья часть соответствующего образца будет такой: *A1=A2=S1*, а весь образец имеет вид *A1 A2 S1*| *#* | *A1=A2=S1*.

В данный момент набор грамматических образцов включает 13 образцов, соответствующих наиболее частым типам русскоязычных терминов: однословных, двухсловных и трехсловных. Для конкретного приложения ATE этот набор может быть легко расширен.

Компонент распознавания вариантов терминов

Этот программный компонент также встроен в состав модуля извлечения терминов-кандидатов, он выявляет среди сформированного множества кандидатов словосочетания, потенциально обозначающие одно и то же понятие. Это позволяет на последующем этапе ранжирования терминов-кандидатов проводить более точный подсчет (пересчет) значений их статистических признаков и их уточненное ранжирование для дальнейшего извлечения терминов как топовых элементов отранжированного списка кандидатов.

В настоящий момент реализовано два способа определения вариантов терминов. Первый предназначен для определения вариантов многословных терминов на основе метода *bag-of-words*: каждое словосочетание представляется в виде множества лемм, и термины-кандидаты считаются одним и тем же термином в случае равенства этих множеств. Например,термины-кандидаты "равномерное прямолинейное движение" и "прямолинейное равномерное движение" будут считаться вариантами одного и того же термина (соответствовать одному понятию).

Второй способ реализован для однословных терминов и выявляет так называемые морфологические варианты терминов, различающихся несколькими синонимичными суффиксами, например: "сегментация" и "сегментирование", "итеративный" и "итерационный". Для этого применяется недавно реализованная программная модель морфемного разбора слов [3], позволяющая для заданного слова получить набор входящих в него морфем с указанием их класса (префикс, корень, суффикс, окончание).

Тестирование новых компонентов

Для тестирования реализованных компонентов была собрана коллекция русскоязычных текстов из разных источников: Habr, Киберленинка, Russia Today и сборники научных статей — всего 89 документов. Размер документов в собранной коллекции варьируется от 3327 до 6078 слов. В качестве статистического признака для ранжирования терминов-кандидатов был выбран признак C-Value. Эксперименты проводились на компьютере со следующими параметрами: Intel Core i7-7500U (4 CPU) 2.7GHz, 16GB RAM.

Для оценки производительности было выполнено 10 тестовых запусков системы извлечения русскоязычных терминов на всей коллекции документов и произведены замеры времени работы для каждого запуска. Усредненное значение времени работы на всей коллекции составляет ∼6.48 мин (∼4.369 с на 1 документ), что говорит о доста-

точно высокой производительности.

Что касается наборов извлеченных в ходе экспериментов терминов, то их экспертная оценка (специалистами в предметной области обработанных текстов) показала приемлемые результаты, сопоставимые с результатами для английского языка.

Заключение

Описанные в работе программные компоненты для извлечения терминов из русскоязычных текстов, реализованные на языке Scala, были встроены в библиотеку atr4s, их работоспособность проверена в составе полной системы ATE, на текстах разных тематик. С помощью библиотеки atr4s, расширенной этими компонентами, можно путем настройки конфигурационных параметров построить систему извлечения терминов для конкретной коллекции текстов, а также использовать модули библиотеки в составе более сложной системы обработки текстов на русском языке, где процесс извлечения терминов является лишь одним из этапов решения более крупной задачи.

В рамках дальнейшей работы предполагается реализовать возможность подключения к библиотеке еще одного морфоанализатора для русского языка (поскольку они отличаются многоми своими свойствами), а также доработать компонент распознавания морфологических вариантов для многословных терминов.

Литература

1. Astrakhantsev N. *ATR4S: toolkit with state-of-the-art automatic terms recognition methods in scala.* Language Resources and Evaluation. — 2018. — T. 52. — №. 3. — pp. 853–872.

2. Bolshakova E., Loukachevitch N., Nokel M. *Topic models can improve domain term extraction.* Advances in Information Retrieval. LNCS, № 7814 — Springer, 2013. — pp. 684–687.

3. Bolshakova E., Sapin A. *Bi-LSTM Model for Morpheme Segmentation of Russian Words.* Proceedings of the Conference on Artificial Intelligence and Natural Language, AINL 2019. CCIS, № 1119. Springer, Cham, 2019, pp. 151–160.

4. Cram D., Daille B. *Terminology extraction with term variant detection.* Proceedings of ACL-2016. System Demonstrations. — 2016. — pp. 13–18.

5. FiveFilters terminology extraction tool `http://fivefilters.org/term-extraction/`

6. Frantzi K., Ananiadou S., Mima H. *Automatic recognition of multi-word terms:. the c-value/nc-value method.* International journal on digital libraries. — 2000. — Т. 3. — №. 2. — pp. 115–130.

7. Marciniak M., Mykowiecka A., Rychlik P. *TermoPL — a flexible tool for terminology extraction.* Proc. of the Int. Conf. on Language Resources and Evaluation (LREC'16). — 2016. — pp. 2278–2284.

8. Maynard D., Li Y., Peters W. *NLP Techniques for Term Extraction and Ontology Population.* Ontology Learning and Population: Bridging the Gap between Text and Knowledge. — 2008 — pp.107.

9. Nokel M., TermExtractor tool
   `https://bitbucket.org/Meister17/term-extraction`

10. Oliver A., Vàzquez M. *TBXTools: a free, fast and flexible tool for automatic terminology extraction.* Proc. of the Int. Conf. Recent Advances in NLP. — 2015 — pp.473–479.

11. Pazienza M. T., Pennacchiotti M., Zanzotto F. M. *Terminology extraction: an analysis of linguistic and statistical approaches.* Knowledge mining. — Springer, Berlin, 2005 — pp. 255–279.

12. Spasić I. et al. *FlexiTerm: a flexible term recognition method.* Journal of biomedical semantics. — 2013. — V. 4, №.1. — pp. 27.

13. Topia terminology extraction tool `https://pypi.org/project/topia.termextract/`

14. Zhang Z., Gao J., Ciravegna F. *Jate 2.0: Java automatic term extraction with apache solr.* Proc. of the Int. Conf. on Language Resources and Evaluation (LREC'16). — 2016 — pp. 2262–2269.

15. Большакова Е.И., Лукашевич Н.В., Нокель М.А. *Извлечение однословных терминов из текстовых коллекций на основе методов машинного обучения.* Информационные технологии, № 7 — 2013. — С. 31–37.

16. Захаров В. П., Хохлова М.В. *Автоматическое выявление терминологических словосочетаний.* Структурная и прикладная лингвистика. Вып. 10. СПб., 2014 — С. 182–200

17. Лукашевич Н.В., Логачев Ю.М. *Комбинирование признаков для автоматического извлечения терминов.* Вычислительные методы и программирование, разд. 2 — 2010. — С. 108–116.

Миков А.И., Миков А.А

Устойчивость к возмущениям масштабируемых мобильных информационно-ориентированных сетей

Введение

Мобильные беспроводные сенсорные сети широко используются для мониторинга в гражданской, промышленной, сельскохозяйственной, военной и других областях. Для обеспечения надлежащего покрытия зоны мониторинга разработаны алгоритмы оптимизации покрытия зоны мониторинга [1] с учетом закономерностей движения узлов сети. Разработка схемы исходного расположения и перемещения узлов позволяет улучшить эффект охвата в зоне наблюдения.

Беспилотные летательные аппараты (БПЛА) представляют собой новую технологию, которая потенциально может быть использована в промышленности и различных сферах жизни человека для предоставления широкого спектра приложений и услуг. Более того, благодаря контролируемой мобильности БПЛА и регулируемой высоте их можно считать подходящим кандидатом для повышения производительности и преодоления ограничений наземных сетей.

В связи с растущими возможностями БПЛА и требованиями к мониторингу удаленных районов, наблюдение с дронов становится все более популярным. В случае стихийного бедствия они могут быстро просканировать большую пораженную территорию и ускорить процесс поиска, чтобы спасти больше человеческих жизней [2].

Во многих случаях для покрытия большой территории с целью мониторинга, сбора и обработки информации в непрерывном режиме важно не только покрытие территории активными датчиками, но и совместная работа датчиков, которая, в свою очередь, требует управления «стаей» [3] дронов.

Потеря связности сети может произойти из-за воздействия на БПЛА внешних возмущений, таких как изменение направления и силы ветра, «воздушные ямы» и т. д. При этом члены «стаи» могут пролетать большие расстояния и оказываться вне зоны уверенного приема радиосигналов от других дронов. Для управления отдельными узлами необходимо своевременно прогнозировать возможную потерю связности и предлагать алгоритмы предсказания разделения графа мобильной сети на отдельные компоненты.

Одним из сигналов о предстоящей потере связности может стать появление подграфов – мостов внутри графа мобильной сети. Для анализа мостов и других частей графа сети могут использоваться распределенные алгоритмы [4]. Тем не менее, даже при корректировке в определенных пределах перемещений узлов мобильной сети, нельзя избежать постепенного ухудшения ее характеристик QoS. Момент времени, когда QoS достигает критического значения, называется временем жизни (lifetime) сети. Разрабатываются различные методы [5] для увеличения времени жизни сети. В нашей работе мы рассмотрим проблему увеличения времени жизни сети в контексте увеличения математического ожидания времени связности динамического графа.

1. Масштабируемая конфигурация мобильной сети

Мобильные сети описываемого класса предназначены для мониторинга (патрулирования) областей S круглой формы с радиусом R_S на поверхности Земли, а также для дистанционного управления устройствами, расположенными в этих областях. Предполагается, что стационарная наземная инфраструктура либо отсутствует (безлюдная территория), либо разрушена в результате стихийных бедствий. Мобильная сеть основана на системе («стае») из n беспилотных летательных аппаратов (дронов) планерного типа, движущихся по круговым траекториям относительно точки, расположенной над центром области. У каждого дрона есть оборудование для наблюдения за наземными объектами и/или для управления наземными объектами. Зона покрытия одного дрона представляет собой окружность радиуса $R_Z < R_S$. Каждый дрон имеет приемопередающее оборудование для связи с другими дронами. Отношение сигнал/шум таково, что радиосигнал уверенно принимается на расстоянии не более $r < R_S$. Координаты дронов в любой момент времени определяют некоторую геометрическую конфигурацию, на основе которой с учетом значения r можно построить геометрический граф [6]и соответствующий ему обычный неориентированный граф. Если в данный момент времени обычный граф связен, то система дронов может выполнять свою функцию как информационно-ориентированная сеть: гарантировать, что каждый БПЛА получает полную «картину» района патрулирования, реализовывать алгоритмы обработки полученных данных, и управлять всеми наземными устройствами. Из этого условия вытекают два требования:

1. геометрическая конфигурация и радиус R_Z должны быть такими, чтобы обеспечить полное покрытие области S;

2. геометрическая конфигурация и радиус r должны соответствовать связному геометрическому графу.

В процессе мониторинга изменяется геометрическая конфигурация сети, поскольку на БПЛА действуют внешние силы (движение воздушных масс и т. д.). Соответственно меняется и геометрический граф, и, возможно, меняется и обыкновенный граф. В некоторые промежутки времени граф будет несвязным, и мобильная сеть теряет полноценные функции, снижается QoS.

Чтобы предотвратить потерю связности можно использовать методы прогнозирования, основанные на вычислении различных характеристик графа. Полученная информация используется для корректировки положения отдельных дронов и, таким образом, увеличения периода связности графа сети.

В следующих разделах представлена математическая модель, предназначенная для построения начальной масштабируемой конфигурации сети мобильного патрулирования области, описаны случайные процессы, моделирующие возмущения, алгоритмы управления для узлов мобильной сети и результаты анализа эффективности алгоритмов управления.

2. Математическая модель

2.1 Базовая начальная конфигурация

Модель мобильной компьютерной сети состоит из динамического геометрического графа $G(t)$ и математической модели области пространства, в которой перемещаются узлы сети. Динамический граф, в свою очередь, определяется исходным геометрическим графом (или многомерным распределением вероятностей геометрических конфигураций – расположением узлов в пространстве) и моделью мобильности.

В отличие от моделей случайного блуждания, в которых узлы перемещаются независимо, мы рассматриваем модели движения узлов, объединенных одной целью. Цель, полностью или частично, может предопределять траектории узлов. В нашей задаче рассматривается круговое движение, полярные координаты узла (ρ_i, φ_i) определяются как $(\rho_i, \varphi_i = \omega t + \psi_i)$, т.е. радиус не зависит от времени, но может варьироваться для разных узлов, а угол φ_i увеличивается с постоянной угловой скоростью. Начальные значения углов могут быть

разными для разных узлов сети. Движение узлов подчиняется ограничениям скорости, $0 \leq v_{min} \leq v_i \leq v_{max}$, ограничениям ускорения, $a_{min} \leq a_i \leq a_{max}$, ограничениям углов поворота, шага, вектору смещения (вертикальные перемещения исключены) и т.д. Существуют взаимные ограничения: минимальное расстояние между узлами.

Мы предполагаем, что все перемещения происходят в одной плоскости, поэтому для удобства вместо зоны наблюдения и зон покрытия на Земле мы можем рассматривать в математической модели их проекции на эту плоскость. Также поставим дополнительное условие - все узлы сети находятся в области S. Зона покрытия $Z_i(t)$ относится к i-му узлу. Зона, полностью лежащая в области S, имеет форму круга радиуса R_Z . Целью перемещения узлов компьютерной сети является обеспечение в любой момент времени полного покрытия области, $\cup\, Z_i(t) \supseteq S$, и связи между всеми узлами.

В нашем исследовании в качестве начальной геометрической конфигурации динамического графа используется экстремальная геометрическая конфигурация [7], основанная на гексагональной упаковке кругов радиуса δ.

Базовая конфигурация отличается наиболее плотной упаковкой, т.е. минимальным количеством сетевых узлов на единицу площади. Но она не обеспечивает полного покрытия территории. Поэтому устанавливаем радиус покрытия одного узла $R_Z = (2/\sqrt{3})\delta$, где δ – радиус окружностей гексагональной упаковки, сохраняя при этом координаты узлов (центров окружностей). В этом случае некоторые пары зон наблюдения Z_i пересекаются, но такая система узлов сети обеспечивает покрытие зоны наблюдения S.

2.2 Связность начальных конфигураций

Базовая конфигурация оптимальна по плотности, но не полностью соответствует ограничению, введенному для области круглой формы. Отдельные узлы базовой конфигурации могут уйти за границу области S, либо (в случае их удаления) покрытие области будет неполным. В этом плане реальная начальная конфигурация отличается от базовой сдвигом координат некоторых узлов сети. На рис. 1 показан пример такой конфигурации [7]. Центры пронумерованных окружностей радиуса δ соответствуют узлам. Дополнительные окружности, обозначенные пунктирными линиями, имеют радиус $R_Z = (2/\sqrt{3})\delta$ и демонстрируют минимальные зоны покрытия узлов (на рисунке они показаны только для узлов 4, 12, 20, 21 в качестве примеров). Точка между кругами 1, 2 и 3 – это центр области S, обозначенной большим кругом. Узлы 1–12 имеют те же координаты, что и в базовой конфигурации. Узлы 13, 14, 15 смещены

относительно базовых положений к центру области, так что центры соответствующих окружностей лежат на границе области S. Узлы 16–21 также имеют измененные полярные координаты. В этом случае они попадают в область и вместе с другими узлами обеспечивают полное покрытие.

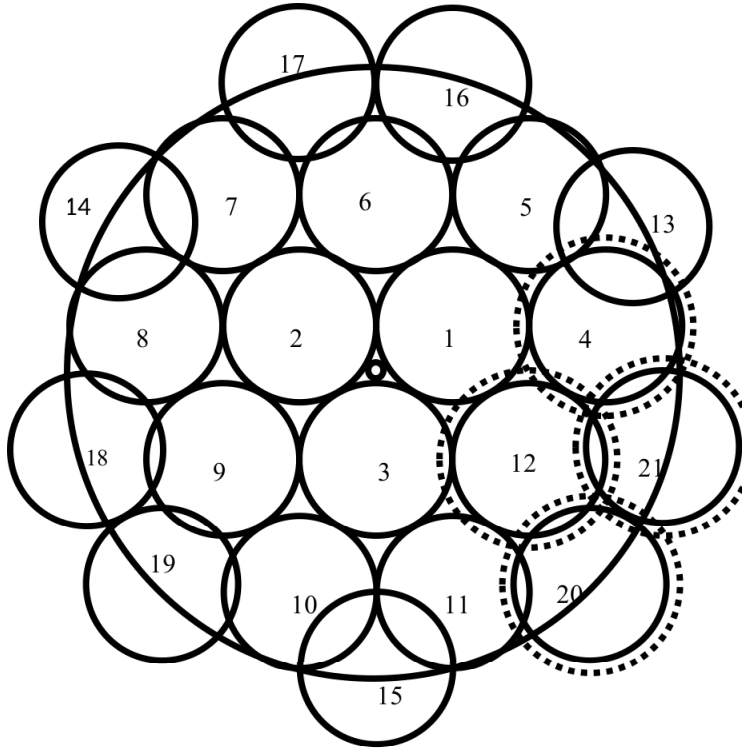

Рисунок 1. Пример начальной конфигурации

Рассмотрим проблему связности беспроводной компьютерной сети. Каждый узел, кроме радиуса покрытия R_Z, также характеризуется радиусом r зоны надежного приема / передачи сигнала. При увеличении радиуса от 0 до $r \approx 1{,}5\delta$ все узлы сети изолированы друг от друга. В интервале от $r \approx 1{,}5\delta$ до $r \approx 1{,}9\delta$ граф сети имеет 6 ребер между парами ближайших узлов. Со значения $r \approx 1{,}9\delta$ количество ребер увеличивается до 18, они образуют цикл периферийных узлов, но центральные узлы остаются изолированными. При сохранении положения узлов для обеспечения связи между соседними узлами в этой конфигурации (для связности графа) требуется выполнение неравенства $r > 2\delta$, т.е. $r > R_Z$.

Поскольку отношение максимального и минимального радиусов траекторий дрона составляет $4/(2/\sqrt{3}) = 2/\sqrt{3} \approx 3{,}46$, то при соотношении $v_{max}/v_{min} > 3{,}46$ вращательное движение этой геометрической конфигурации может быть обеспечено с помощью сохранение формы и, следовательно, может быть обеспечено постоянное наблюдение за территорией круга радиуса 4δ с помощью 21 БПЛА, оснащенного оборудованием для наблюдения за подстилающей поверхностью внутри круга радиусом $(2/\sqrt{3})\delta$ и оборудованием для передачи информации между БПЛА на расстояние не менее 2δ.

2.3 Движение конфигурации

В идеальной ситуации патрулирования области конфигурация дронов вращается с постоянной угловой скоростью. При этом взаимное расположение БПЛА не меняется.

Из-за изменения возможно искажение конфигурации системы траектории относительно проектной некоторого узла. В большинстве случаев покрытие области S будет нарушено (при $R_Z = (2/\sqrt{3})\delta$). Связность компьютерной сети более устойчива: основная часть графа сети - это триангуляция области, а смещение вершины отдаляет ее от одних вершин, но приближает ее к другим. Таким образом, с исчезновением некоторых ребер, возможно, возникают новые ребра.

Рассмотрим случайные возмущения (например, ветер переменной силы и направления), которые приводят к отклонениям траекторий дронов от теоретических. Все возмущения представляют собой случайные векторы сил $\mathbf{F}_j(t)$, независимые для разных узлов, не превышающие по модулю величины b.

Процесс изменения переменной «связность графа» (связен/несвязен $= 1/0$) представляет собой кусочно-постоянный случайный процесс $\sigma(t)$. Не очевидно, что этот процесс является стационарным: постоянные отклонения могут привести к разрушению исходной конфигурации. Действительно, моделирование показало, что со временем количество ребер в динамическом графе постепенно уменьшается и, наконец, в какой-то момент граф становится несвязным.

3. Управление динамическим гиперграфом

В [7] авторы предложили алгоритм управления связностью динамического графа мобильной беспроводной компьютерной сети, ис-

пользующий прогнозы на основе появления мостов в графе.

Изменения траекторий отдельных БПЛА, вызванные возмущениями, не влияют на достижение целей всей системы БПЛА до тех пор, пока из системы не «выпадет» отдельный дрон. Например, можно значительно (но на одну и ту же величину) изменить угловые скорости всех устройств, сохраняя геометрическую конфигурацию. Таким образом, в системе автоматического управления (САУ) БПЛА нет необходимости поддерживать устойчивость траекторий. Достаточно поддерживать только связность графа. Таким образом, задача синтеза САУ из непрерывной области переходит в топологическую.

Задача САУ – оценить «расстояние» данного связного графа сети до «ближайшего» несвязного графа, локализовать те вершины динамического графа, в области которых граф может распасться на компоненты, и – переход к геометрии – скорректировать положения соответствующих БЛА в пространстве. В качестве расстояния мы можем использовать инвариант графа, такой как реберная связность $\lambda(G(t))$. Однако частые расчеты реберной связности создают значительную нагрузку на вычислительные мощности БПЛА, а небольшая частота вычислений приводит к чистой задержке в контуре управления, что снижает качественные характеристики САУ. Анализ случайных процессов изменения связности сети для рассматриваемых в работе условий показывает, что после момента $t = 0$ количество q ребер динамического графа постепенно уменьшается. Потеря связности происходит при $q \approx 10r/\delta$ (анализ проводился для $2{,}1\delta \leq 10r \leq 3{,}0\delta$ и $0{,}005 \leq b \leq 0{,}02$). Поэтому в алгоритме управления [7] количество ребер было выбрано как предварительный индикатор приближения динамического графа к состоянию несвязности. Затем отыскивались мосты для управления движением узлов. Далее корректировалось движение вершин графа, образующих мост, в результате чего они начинали сходиться.

3.1 Алгоритм управления, основанный на гиперребрах

Здесь мы рассмотрим новый алгоритм управления узлами мобильной мониторинговой сети в условиях действия возмущений. Этот алгоритм основан на вычислении степеней вершин графа сети. Поскольку мы имеем дело с геометрическими графами, для данной вершины соседи (смежные вершины) являются географически ближайшими к ней вершинами. В этих условиях гиперграф – более подходящая абстракция. Гиперребро, инцидентное узлу, включает этот узел и всех его географических соседей (находящихся на расстоянии

меньше r). В динамике гиперребра происходят изменения: отдельные вершины, входящие в его состав, меняются – исключаются или, наоборот, включаются. Если степень некоторого ребра становится равной единице, то это означает, что вершина геометрического гиперграфа потеряла всех своих соседей и стала изолированной. Когда степень гиперребра становится равна двум, мы отмечаем возможное наличие моста в графе. Периодически проверяя значения степеней гиперребер, можно заметить их критическое уменьшение во времени.

Более подробно алгоритм состоит из следующих шагов.

1. Инициализация: формирование списка D, содержащего степени вершин $\deg(i)$ исходной конфигурации; здесь i – номер вершины в графе. Степени рассчитываются на основе значений параметров модели R_S, r, δ, m, где m – масштаб конфигурации. Начальное формирование списка Q текущих степеней вершин, $Q = D$. Генерация пустого списка различий L. Задание начального периода времени τ тестирования динамического графа.

2. Основной цикл: тело цикла выполняется с периодом τ.

   (a) Тест: Обновление списка Q текущих степеней вершин. Если $Q(i) < D(i)$, то перейти (с ожиданием) к следующей итерации цикла.

   (b) Сигнал предупреждения: если $Q(i) < D(i)$ для любого i, то обновить список различий L.

   (c) Цикл по списку L: для каждой вершины j из списка L определить отсутствующих «бывших» соседей по начальной конфигурации. Вычислить вектор смещения V_j от вершины j в направлении центра масс многоугольника, образованного вершиной j и ее бывшими соседями. Начать (или продолжить) перемещение вершины j в направлении вектора V_j с учетом ограничений на величину ускорения.

3.2 Результаты моделирования

Для оценки эффективности предложенного алгоритма управления мобильной сетью было проведено имитационное моделирование. Стохастические возмущения были наложены на детерминированный процесс вращения роя БПЛА, как описано ранее. Возмущения для каждого узла были индивидуальными и независимыми, но подчинялись одним и тем же вероятностным законам. В отличие от часто используемых методов [8], мы использовали более реалистичные псевдослучайные процессы.

При моделировании движения мобильной беспроводной сети случайные процессы $\mathbf{F}_i(t)$ представляются псевдослучайными последовательностями, полученными из рекуррентных соотношений $u_i = a_1 u_{i-1} + a_2 u_{i-2} + a_3 u_{i-3} + \xi_i$ (для угла) и $v_i = b_1 v_{i-1} + b_2 v_{i-2} + b_3 v_{i-3} + \zeta_j$ (для модуля скорости) с независимыми (псевдослучайными) величинами ξ_i , ζ_i и нулевыми начальными значениями.

В каждом сеансе моделирования было проведено 3000 независимых экспериментов для оценки среднего времени периода связности мобильной сети. Каждый эксперимент начинался с начальной конфигурации сети (рис. 1) и заканчивался в момент потери связности. На основании этих данных было оценено математическое ожидание продолжительности периода связности. Сеансы моделирования проводились для различных значений относительного радиуса (r/δ) и различных уровней (b) возмущения движения.

На рис. 2 и 3 показаны три графика зависимости математического ожидания длительности периода связности графа мобильной сети от относительного радиуса зоны надежного приема / передачи радиосигналов узлом сети.

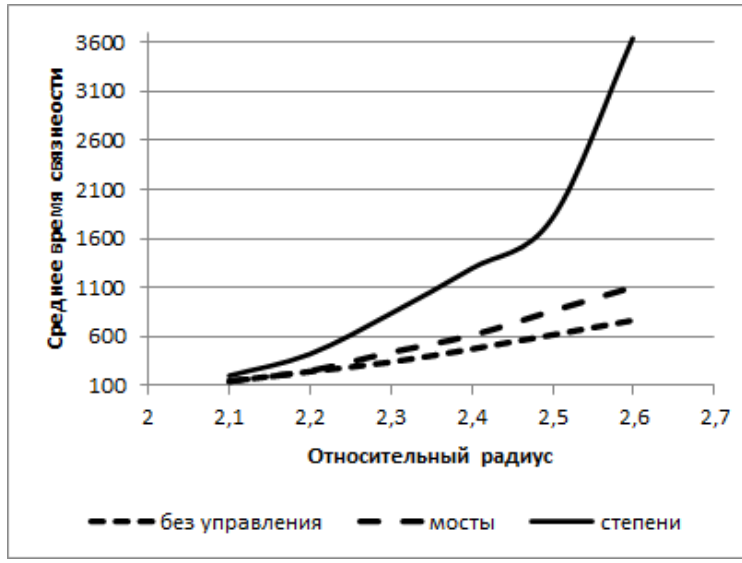

Рисунок 2. Среднее время связности для $b = 0.01$

График с пометкой «без управления» получен в условиях помех движению, но без каких-либо компенсирующих эффектов. Обозначение «мосты» означает, что использовался алгоритм прогнозирования, основанный на вычислении общего количества ребер и поиске мостов. Обозначение «степени» означает, что был использован

алгоритм, предложенный в данной статье. Хорошо видно, что при данном уровне возмущений эффективность нового алгоритма прогнозирования и управления значительно выше, чем эффективность алгоритма, основанного на поиске мостов.

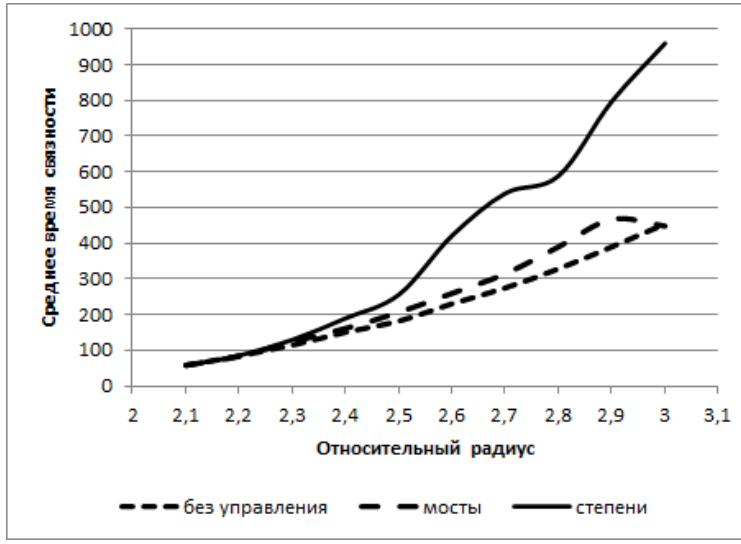

Рисунок 3. Среднее время связности для $b = 0.02$

Рисунки 2 и 3 показывают, что по мере увеличения возмущения различие характеристик между полетом без управления и полетом с использованием двух разных методов управления становится небольшой. Это связано с тем, что при использовании метода «мостов» управляющая программа не успевает регулировать движение, а при использовании метода «степеней вершин» постоянных значений управления движением становится недостаточно для регулирования полета. Однако при малых значениях возмущения управление с использованием степеней вершин показывает более высокую эффективность по сравнению с методом мостов. Следовательно, чтобы алгоритм управления с использованием степеней вершин был более эффективным, необходимо на этапе управления зафиксировать разницу между идеальной траекторией полета и фактической, провести соответствующие расчеты и выбрать постоянные значения управления полетом, необходимые для конкретной ситуации.

Заключение

В статье представлена модель мобильной информационно-ориентированной сети для патрулирования области и алгоритм автоматического управления сетью, обеспечивающий ее работоспособность при наличии возмущений движения. Геометрическая структура сети основана на теории экстремальной упаковки, которая обеспечивает использование минимального количества узлов, расположенных на беспилотных летательных аппаратах. Этот подход также характеризуется масштабируемостью создаваемых мобильных сетей. Алгоритм управления реализует прогнозную математическую модель, основанную на вычислении характеристик динамических геометрических графов. Основная задача алгоритма управления - поддерживать связность сети максимально возможное время при наличии внешних воздействий, разрушающих структуру. В качестве индикаторов для прогнозирования потери связности динамического графа сети используются степени вершин графа. Сравнение проводится с ранее разработанным авторами алгоритмом управления по другому показателю - появлению мостов в графе. Алгоритм сочетает в себе глобальную (по всей сети) оценку топологии с локальными управляющими действиями по коррекции положения (по отношению к одному или нескольким соседним узлам одновременно). С помощью методов моделирования проведена сравнительная оценка эффективности предложенного алгоритма управления. Показано, что новый алгоритм поддерживает связность сети значительно дольше при умеренных возмущениях.

Благодарности. Исследование выполнено при финансовой поддержке РФФИ и Администрации Краснодарского края в рамках научного проекта № 19-47-230003.

Литература

1. Li Q., Liu N. *Monitoring area coverage optimization algorithm based on nodes perceptual mathematical model in wireless sensor networks.* M.: Computer Communications, vol. 155, 1 April 2020, pp. 227-234.

2. Mishra B., Garg D., Narang P., Mishra V. *Drone-surveillance for search and rescue in natural disaster.* M.: Computer Communications, vol. 156, 15 April 2020, pp.1-10.

3. Nguyen M.T. *Distributed compressive and collaborative sensing data collection in mobile sensor network.* M.: Internet of Things, 9 (2020) Article 100156.

4. Akram V.K. *An asynchronous distributed algorithm for minimum s-t cut detection in wireless multi-hop network.* M.: Ad Hoc Networks, vol. 101, 15 April 2020, Article 102092.

5. Zhang X., Lu X., Zhang X. *Mobile wireless sensor network lifetime maximization by using evolutionary computing methods.* M.: Ad Hoc Networks, vol. 101, 15 April 2020, Article 102094.

6. Penrose M. *Random geometric graph.* M.: Oxford Studies in Probability. Oxford University Press, 200.

7. Миков А.И., Миков А.А. *Управление динамическими графами мобильных компьютерных сетей при случайных возмущениях.* М.: Информатизация и связь, 2, 2020, с.136-142, doi 10.34219/2078-8320-2020-11-2-136-142.

8. Silva R.T., Colletti R.R., Pimentel C., de Morae R.M. *BETA random waypoint mobility model for wireless network simulation.* M.: Ad Hoc Networks, vol. 48, 15 September 2016, pp. 93-100.

Никифоров Н.И., Волканов Д.Ю., Скобцова Ю.А.

Анализ и исследование структур данных для поиска в таблицах классификации в сетевом процессорном устройстве с архитектурой RuNPU [1]

Введение

В настоящее время активно развивается технология программно-конфигурируемых сетей [1], в которых требуются высокопроизводительные коммутаторы [2]. Возникает задача разработки программируемого сетевого процессора, являющегося основным функциональным элементом коммутаторов. Сетевое процессорное устройство (СПУ) представляет из себя интегральную микросхему, специализированную для обработки сетевых пакетов, которая выполняет следующие функции: получение пакета с физического порта, выделение заголовка, классификация пакета по его заголовку, принятие решения о дальнейшем пути следования пакета, отправка пакета на физический порт [3]. В настоящее время активно ведётся разработка программируемых СПУ. Под программируемым СПУ понимается интегральная микросхема, которая позволяет менять программу обработки пакетов и набор различаемых полей заголовков. Программируемый СПУ позволяет быстро подстраиваться под новые протоколы, и использовать коммутатор в ПКС сетях [4].

Исходя из функций СПУ, целесообразно рассматривать архитектуру, основанную на наборе конвейеров, которая позволяет с фиксированной задержкой обрабатывать каждый пакет. Конвейер в сетевом процессоре состоит из вычислительных блоков. В данной работе рассматривался этап классификации пакетов, который выполняется на конвейере СПУ. Под классификацией понимается процесс идентификации сетевого пакета по его признакам, определяемыми текущим протоколом. Таблица классификации – набор правил, содержащих в себе признаки, по которым идентифицируется группа пакетов, и действия, которые СПУ выполняет над данной группой пакетов. Таким образом, для выполнения классификации СПУ должно включать в себя ассоциативное устройство. Для реализации этого устройства естественным будет использование ассоциативной памяти. Однако

[1]Работа выполнена при частичной поддержке РФФИ, грант № 19-07-01076

единственный контроллер ассоциативной памяти будет являться узким местом, так как к нему должны иметь доступ все стадии всех конвейеров. Соответственно, возникает потребность усложнения архитектуры, например, путём добавления в неё нескольких контроллеров ассоциативной памяти. Чтобы избежать сложной организации памяти, в архитектуре можно отказаться от использования ассоциативной памяти. В таком случае одно из решений — совместить память команд и данных, и разместить память на кристалле сетевого процессора. Таким образом, возникает задача разработки структур данных для поиска в таблицах классификации в СПУ без выделенного ассоциативного устройства.

Раздел 1 содержит описание предметной области. В разделе 2 формулируются ограничения на реализуемую структуру данных. Раздел 3 содержит описание рассматриваемой архитектуры сетевого процессора. В разделе 4 проводится обзор предлагаемых структур данных. Раздел 6 посвящён экспериментальному исследованию и его результатам. В заключении кратко описаны основные результаты работы и возможные направления дальнейших исследований.

1. Введение в предметную область

Рассмотрим подробнее архитектуру ПКС коммутатора, он состоит из двух главных компонентов: центрального процессорного устройства (ЦПУ) и СПУ [4]. ЦПУ отвечает за получение сигналов с ПКС контроллера и обработку их. В данной работе нас будет интересовать функция ЦПУ, отвечающая за трансляцию таблиц классификации в программу обработки пакетов на языке ассемблера (микрокод). В работе используется описание структур данных в терминах ЦПУ, которые затем по определённым правилам транслируются в микрокод СПУ. Так ссылка в описании структур данных транслируется в переход на метку, которая является указателем на определённую инструкцию микрокода. Это связано с отсутствием адресуемой памяти, в которой могли бы хранится данные. Под адресуемой памятью понимается память, в которой с данными можно выполнять действия чтения и записи по указателю. Вместо этого микрокод, загружаемый на СПУ, содержит в себе все необходимые данные для классификации пакетов. Будем говорить, что микрокод, полученный путём трансляции структуры данных, построенной для заданной таблицы классификации, реализует структуру данных.

2. Постановка задачи

Необходимо разработать структуру данных для поиска в таблицах классификации в рамках архитектуры сетевого процессора без выделенного ассоциативного устройства.
Разрабатываемая структура данных должна удовлетворять следующим требованиям.

1. Структура данных должна транслироваться в микрокод сетевого процессора;

2. Обработка одного пакета на СПУ с загруженным микрокодом, реализующим структуру данных, не должна превышать 250 тактов СПУ;

3. Объём памяти, занимаемый микрокодом, реализующим структуру данных, содержащую не более чем 32000 битовых строк длиной до 64 бит, не должен превышать 2 МБ;

4. Структура данных должна быть универсальна, а именно должна позволять выполнять поиск любых битовых строк длиной до 64 бит.

3. Архитектура сетевого процессора

3.1 Общее описание архитектуры

В рассматриваемом сетевом процессорном устройстве (RuNPU) [4] используется конвейерная архитектура, каждый конвейер состоит из 10 вычислительных блоков.

Каждый вычислительный блок имеет доступ к участку памяти, объёмом 200 Кб, в котором располагаются микрокод программы обработки таблиц классификации. Таким образом, на один вычислительный конвейер отводится 2 Мб памяти. Схематично архитектура изображена на рисунке 1. Одной из особенностей рассматриваемой архитектуры является наличие ограничения на 25 тактов, которые один пакет может обрабатываться на одном вычислительном блоке СПУ. Данное ограничение обусловлено требованием к производительности сетевого процессора, а именно требованием фиксированного времени обработки одного пакета на сетевом процессоре.

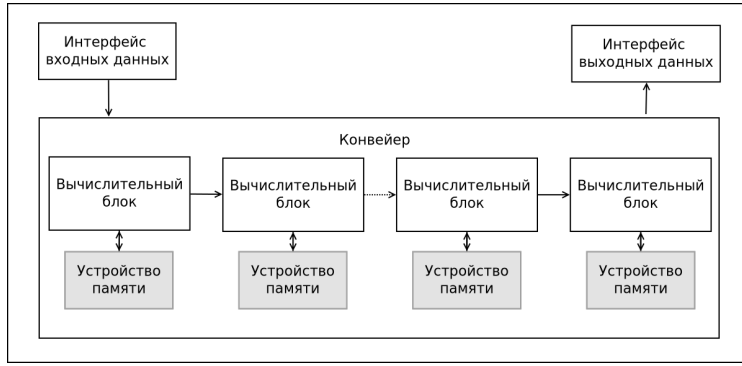

Рис. 1: Архитектура рассматриваемого сетевого процессора

3.2 Язык ассемблера сетевого процессора

Для описания программ обработки сетевых пакетов в рассматриваемой архитектуре сетевого процессора используется язык ассемблера. В рассматриваемом языке присутствуют следующие классы инструкций:

- Инструкции для работы с основным регистром;
- Инструкции арифметических операций;
- Инструкции битовых операций;
- Инструкции для условного и безусловного перехода на метку;
- Инструкция записи в регистр выходного порта.

4. Обзор структур данных

4.1 Критерии обзора

В силу особенностей архитектуры сетевого процессора, а именно отсутствия адресуемой памяти, которая требуется для не древовидных структур данных будут рассматриваться только древовидные структуры данных.

При выборе структур данных использовались следующие критерии:

- **асимптотическая сложность поиска** – позволяет оценить использование ресурсов сетевого процессора для поиска в структуре данных;
- **универсальность структуры данных** – используемая структура данных должна поддерживать поиск произвольных битовых строк, не превосходящих по длине 64 бита;
- **количество вершин**, которое необходимо посетить для поиска, в случае хранения битовых строк длиной не более W;
- **оценка объёма памяти**, занимаемого структурой данных – рассматривается объём памяти, занимаемый структурой данных, для 32000 вхождений префиксов IPv4.

4.2 Бинарное однобитное дерево

Наиболее распространённая [6] структура данных для поиска по наиболее длинному префиксу [3]. Представляет из себя дерево, которое строится по битовым строкам, для каждого бита используется отдельная вершина. Поиск осуществляется спуском в глубину по битам обрабатываемой битовой строки. Поиск продолжается до тех пор, пока пустая вершина не будет достигнута, а результатом является последний встретившийся префикс.

- **Асимптотическая сложность поиска** для этой структуры данных соответствует $O(W)$;
- **Универсальность структуры данных** – рассматриваемое дерево может быть использовано для любых данных, представимых в битовом виде;
- **Количество вершин**, которое необходимо посетить для поиска не превосходит W;
- **Оценка объёма памяти**, занимаемого структурой данных – 39 Мбайт.

4.3 Бинарное сжатое дерево

Данное дерево является оптимизацией бинарного однобитного дерева [7]. В нём отсутствуют проходные вершины, вершины с одним листом, которые удаляются после построения бинарного однобитного дерева. Благодаря этому, бинарное сжатое дерево занимает меньше памяти. Поиск осуществляется спуском в глубину по битам обрабатываемой битовой строки. Поиск продолжается до тех пор, пока

пустая вершина не будет достигнута, а результатом является последний встретившийся префикс.

- **Асимптотическая сложность поиска** для этой структуры данных соответствует $O(W)$;
- **Универсальность структуры данных** – рассматриваемое дерево может быть использовано для любых данных, представимых в битовом виде;
- **Количество вершин**, которое необходимо посетить для поиска не превосходит W;
- **Оценка объёма памяти**, занимаемого структурой данных – 2 Мбайт.

4.4 АВЛ дререво

АВЛ дерево относится к классу самобалансирующихся деревьев. В данных деревьях могут использоваться только скалярные ключи. Был разработан алгоритм представляющий префиксы как скалярные величины [6]. Данный алгоритм состоит из следующих шагов [8]:

- Если длина двух префиксов совпадает, то они сравниваются как две скалярные величины;
- Если длина первого префикса q меньше длины префикса p, первые $len(q) - 1$ бит обоих префиксов сравниваются как скалярные величины.
- **Асимптотическая сложность поиска** для этой структуры данных соответствует $O(1 + log_2(N))$,, где N количество префиксов в таблице;
- **Универсальность структуры данных** – рассматриваемое дерево может быть использовано для любых данных, представимых в битовом виде;
- **Количество вершин**, которое необходимо посетить для поиска не превосходит $1 + log_2(N)$, где N количество префиксов в таблице;
- **Оценка объёма памяти**, занимаемого структурой данных – 1.5 Мбайт, так как в структуре данных хранятся только вершины, в которых содержатся префиксы.

4.5 Сравнение структур данных

Для сравнения структур данных рассмотрим Таблицу 1 сравнения по критериям: асимптотическая сложность, универсальность структуры данных, количество вершин, которое необходимо посетить для поиска и оценка памяти занимаемой структурой данных. В таблице используются следующие обозначения: W − максимальная длина префикса, N − количество префиксов в структуре данных, L − количество дочерних вершин, − глубина дерева.

Название	**Сложность поиска**	**Универсальность**	**Количество вершин**	**Память, Мбайт**
Двоичное однобитное дерево	$O(W)$	да	W	39
Двоичное сжатое дерево	$O(W)$	да	W	2
АВЛ дерево	$O(1 + \log_2 N)$	да	$\log_2 N$	1.5

Таблица 1: Сравнение структур данных;

У каждой рассмотренной структуры данных есть свои достоинства и недостатки, рассмотрим их:

1. **Двоичное однобитное дерево** − данная структура проста в реализации, но занимает много памяти и поиск требует прохождения W вершин.

2. **Двоичное сжатое дерево** − занимает меньше памяти, чем двоичное однобитное дерево, но поиск требует прохождения W вершин. Соответственно использование данного дерева предпочтительнее, чем двоичного однобитного дерева.

3. **АВЛ дерево** − занимает меньше всего памяти, и при этом, поиск требует прохождения малого количества вершин, которое зависит от количества битовых строк, которые хранятся в структуре данных, а не от конкретного префикса.

Таким образом на основе обзора для дальнейшей реализации были выбраны две структуры данных: АВЛ дерево с алгоритмом представления префиксов как скалярных величин и бинарное сжатое дерево.

5. Алгоритмы построения рассматриваемых структур данных

5.1 Обозначения

В описании алгоритмов использовалась структура данных *Node*, которая содержит в себе следующие поля:

- *Node.left* — ссылка на левого потомка вершины;
- *Node.right* — ссылка на правого потомка вершины;
- *Node.rule* — действие, соответствующее префиксу данной вершины;
- *Node.prefix* — префикс в текущей вершине;
- *Node.bit* — значащий бит в текущей вершин;

И структура *prefix*, которая содержит в себе следующие поля:

- *prefix.bit_string* — битовая строка, задающая префикс;
- *prefix.length* — длина префикса. При поиске точного совпадения длина префикса равна длине битовой строки;

5.2 Алгоритм построения бинарного однобитного дерева

Для добавления вершин в структуру данных определим процедуру *Add* (Рис. 2).
Входные данные: текущая вершина *Node*, добавляемый префикс *prefix*, правило для текущего префикса *rule*, и текущий бит *bit*.
Выходные данные: обновлённая текущая вершина, а именно объект структуры *Node*.

```
1  procedure Add(Node, prefix, rule, bit)
2      if Node is empty then
3          Node = new Node()
4          Node.bit = bit
5      endif
6      if prefix.length == bit then
7          Node.prefix = prefix
8          Node.rule = rule
9          Node.bit = bit
10         return Node
11     endif
12     if prefix.bit_string[bit] == 1 then
13         Node.left = Add(Node.left, prefix, rule,
       bit + 1)
14     else
15         Node.right = Add(Node.right, prefix, rule
       , bit + 1)
16     endif
17     return Node
```

Рис. 2: Процедура добавления вершины в бинарное однобитное дерево.

Таким образом, для построения бинарного однобитного дерева необходимо последовательно добавить все правила из таблицы классификации, используя процедуру *Add*.

5.3 Алгоритм построения бинарного сжатого дерева

Для построения бинарного сжатого дерева, необходимо построить бинарное однобитное дерево, затем удалить все проходные вершины. Проходной вершиной называется вершина, у которой только один лист. Количество удалённых вершин записывается в следующую вершину дерева. Рассмотрим подробнее процедуру удаления проходной вершины.

Рассмотрим процедуру удаления *Remove* проходных вершин из бинарного однобитного дерева, для получения бинарного сжатого дерева.

Входные данные: текущая вершина *Node*.

Выходные данные: обновлённая текущая вершина *Node*. Алгоритм удаления не важных вершин может быть представлен в виде псевдокода процедуры *Remove* (Рис. 3)

```
1  procedure Remove(Node):
2      if Node.prefix is empty then
3          if Node.left is empty and Node.right is
       not empty then
4              return Node.right
5          endif
6          if Node.left is not empty and Node.right
       is empty then
7              return Node.left
8          endif
9      else
10         return Node
11     endif
```

Рис. 3: Процедура удаления проходных вершин.

5.4 АВЛ дерево

алгоритм сравнения префиксов как скалярных величин

Данный алгоритм может быть описан процедурой *PrefixCompare* (Рис. 4).
Входные данные: префикс *prefix1* и префикс *prefix2*.
Выходные данные: *True*, если первый префикс больше второго, иначе *False*.

```
1  procedure PrefixCompare(prefix1, prefix2):
2      if prefix1.length == prefix2.length then
3          return Int(prefix1.bit_string) > Int(prefix2.bit_string)
4      else
5          minimal_length = Min(prefix1.length, prefix2.length) - 1
6          return Int(prefix1.bit_string[0:minimal_length]) > Int(prefix2.
       bit_string[0:minimal_length])
7      endif
```

Рис. 4: Процедура сравнения префиксов как скалярных величин.

Алгоритм построения

Рассмотрим алгоритм построения АВЛ дерева, а именно добавления в него новых вершин. Данный алгоритм может быть описан процедурой *Add* (Рис. 5). **Входные данные:**

- *Node* — объект структуры вершины АВЛ дерева.
- *prefix* — текущий префикс.
- *rule* — правило для текущего префикса.

Выходные данные: обновлённая текущая вершина *Node*.
Также в процедуре *Add* используется процедура *BalanceNode*, которая выполняет операцию балансировки АВЛ дерева.

```
1  procedure Add(Node, prefix, rule):
2      if Node.prefix is empty then
3          return new Node(prefix, rule, depth=1)
4      endif
5
6      if prefixCompare(prefix, Node.prefix) then
7          Node.right = Add(Node.right, prefix, rule)
8      else:
9          Node.left = Add(Node.left, prefix, rule)
10     endif
11
12     return BalanceNode(Node)
```

Рис. 5: Процедура добавления вершины в АВЛ дерево.

Данная процедура позволяет построить АВЛ дерево, последовательно добавляя правила из таблицы.

6. Экспериментальное исследование

6.1 Описание программной реализации

Для проведения дальнейшего экспериментального исследования была разработана программная реализация на языке python. Программная реализация содержит следующие модули:

- Модуль загрузки таблиц классификации из файла.
- Модуль построения АВЛ-дерева.
- Модуль построения Бинарного сжатого дерева.
- Модуль построения Бинарного однобитного дерева.
- Модуль преобразования построенного дерева в язык ассемблера.

Схема взаимодействия модулей представлена на рисунке 6.

6.2 Методика экспериментального исследования

Экспериментальное исследование было проведено на имитационной модели СПУ. Разработанные структуры данных сравнивались

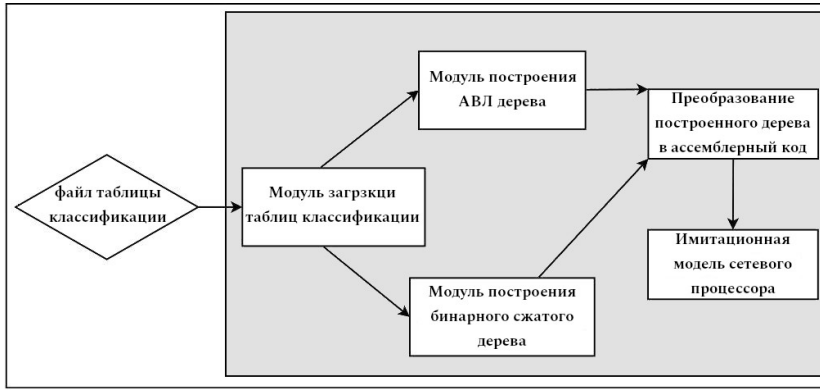

Рис. 6: Схема программной реализации

по следующим параметрам: количество циклов процессора затраченных на классификацию одного пакета, и размер итоговой структуры данных. Экспериментальное исследование проводилось для таблиц классификации различных размеров, а именно:

- 1000,
- 8000,
- 16000,
- 32000,
- 64000.

И битовые строки различной длины: 32, 48 бит. В качестве исходных данных брались таблицы содержащие правила для IPv4 адресов российского сектора сектора.

Объём памяти занимаемый структурой данных будет рассчитываться по формуле $M = instruction\_size * N$, где $instruction\_size =$ 128 бит, так как в рассматриваемой архитектуре сетевого процессора максимальная длина инструкции равна 128 бит. N - количество инструкций в полученной программе на языке ассемблера.

6.3 Результаты экспериментального исследования

На первой диаграмме (Рис. 7) представлено сравнение реализованных структур данных по среднему количеству инструкций на

один пакет. Видно, что АВЛ дерево показывает лучший результат из всех реализованных структур данных.

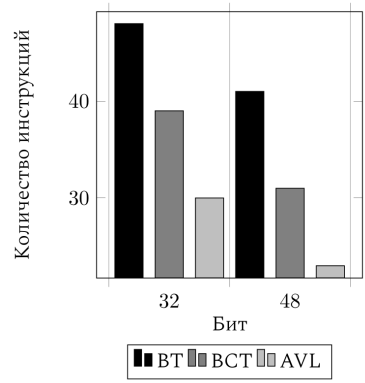

Рис. 7: Сравнение количества инструкций, затраченных на обработку пакета.

На второй диаграмме (Рис. 8) представлено сравнение реализованных структур данных по максимальному объёму памяти занимаемому структурой данных. Видно, что только АВЛ дерево удовлетворяет ограничением представленным в пункте 2, и занимает значительно меньше памяти, чем остальные структуры данных.

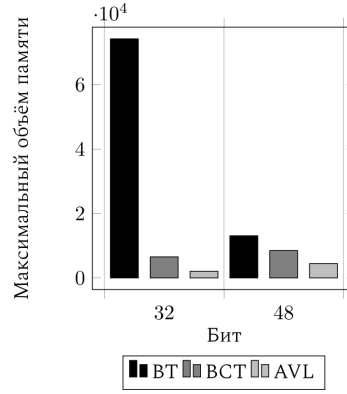

Рис. 8: Сравнение максимального объёма памяти.

7. Заключение

В данной статье рассмотрена проблема классификации пакетов в СПУ без отдельного ассоциативного устройства. Были разработаны структуры данных и алгоритмы трансляции в микрокод для СПУ. Была проведена оценка разработанных структур данных на имитационной модели СПУ. По результатам оценки был сделан вывод, что только АВЛ дерево удовлетворяет ограничениям СПУ. В качестве направления дальнейших исследований можно указать: оптимизацию по памяти реализованных структур данных, а также возможность изменений в архитектуре СПУ, для применения других структур данных.

Литература

1. Смелянский Р.Л. *Программно-конфигурируемые сети*: Открытые системы, 2012. № 9, с. 15–26.

2. Bifulco, Roberto and Rétvári, Gábor. *A survey on the programmable data plane: Abstractions, architectures, and open problems*: 2018 IEEE 19th International Conference on High Performance Switching and Routing (HPSR). 1–7.

3. Chao, H Jonathan and Liu, Bin. *High performance switches and routers*: John Wiley & Sons - 2007

4. Беззубцев С.О., Васин В.В., Волканов Д.Ю., Жайлаулова Ш.Р., Мирошник В.А., Скобцова Ю.А., Смелянский Р.Л. *Об одном подходе к построению сетевого процессорного устройства*: Моделирование и анализ информационных систем, 2019. № 26, 1, с. 39–62.

5. Mun, Ju Hyoung and Lim, Hyesook and Yim, Changhoon. *Binary search on prefix lengths for IP address lookup* IEEE Communications Letters, 2006. 10, 6, 492–494.

6. Behdadfar, Mohammad and Saidi, Hossein and Alaei, Hamid and Samari, Babak. *Scalar prefix search: A new route lookup algorithm for next generation internet*: IEEE INFOCOM, 2009. 2509–2517.

7. Ruiz-Sánchez, Miguel Á and Biersack, Ernst W and Dabbous, Walid. *Survey and taxonomy of IP address lookup algorithms*: IEEE network, 2001. 15, 2, 8–23.

8. Behdadfar, Mohammad and Saidi, Hossein and Hashemi, Massoud Reza and Lin, Ying-Dar. *Coded and scalar prefix trees: Prefix matching using the novel idea of double relation chains*: ETRI Journal, 2011. 33, 3, 344–354.

9. Berger, Michael. *IP lookup with low memory requirement and fast update*: Workshop on High Performance Switching and Routing, 2003, HPSR. 287–291.

10. Philippe Biondi. *Scapy python library*: https://scapy.net - 2019

11. Le, Hoang and Prasanna, Viktor K. *Scalable tree-based architectures for IPv4/v6 lookup using prefix partitioning*: IEEE Transactions on Computes, 2011. 61, 7, 1026–1039.

12. Cheung, Gene and McCanne, Steven. *Optimal routing table design for IP address lookups under memory constraints*: IEEE INFOCOM, 1999. 3, 1437–1444.

Шапошников В.А., Писковский В.О.

Постановка задачи и краткий обзор решений для проблемы управления процессами согласованной реконфигурации ПКС

Введение

Вследствие развития информационных технологий прослеживается постоянный рост сетевого трафика, что может привести к проблемам, не позволяющим эффективно использовать потенциал развития. Одним из решений данной проблемы является технология Программно-Конфигурируемых сетей (ПКС). ПКС — это концепция построения сети, в котором контур управления сетью разделён с контуром передачи данных. Это достигается с помощью переноса функции управления в приложения, работающего на отдельном сервере (контроллере). Коммутаторы связаны с контроллером выделенным каналом управления, по которому передаются специальные сообщения, описанные в спецификациях протокола OpenFlow.

Коммутатор OpenFlow содержит таблицы классификации потоков и таблицы групп. Таблицы определяют порядок обработки и пересылки пакетов. Контроллер, используя OpenFlow протокол, может добавлять, обновлять и удалять правила и таким образом задавать правила обработки пакетов. Пакет, поступивший во входной буфер одного из портов, подключенных к каналу передачи данных, обрабатывается правилами коммутации, содержащихся в таблицах коммутатора. После этого пакет либо поступает в выходной буфер одного из портов для отправления по каналу передачи данных или по каналу управления на контроллер для анализа, либо сбрасывается.

1. Аномалии реконфигурации

Реконфигурация сети - изменение содержимого таблиц коммутации пакетов в сетевых коммутаторах. Процедура реконфигурации может потребоваться для множества причин: поддержание и восстановление движения пакетов в случае отключения аппаратуры, оптимизации маршрутов в сети и так далее. Однако бесконтрольный процесс реконфигурации сети может привести к ослаблению безопасности сети и возникновению нежелательных ситуаций (аномалий) [1].

Это связано с тем, что задачи политики безопасности действуют лишь для общего поведения сети после того, как сеть уже сконфигурирована, в то время как некорректные состояния могут возникать именно в процессе её реконфигурации, изменения записей контроллером в таблицах коммутации в ответ на запросы, обрабатываемые контроллером, по маршрутизации потоков. При этом разные запросы могут приводить к возникновению конфликтов, то есть внесению противоречащих друг другу изменений, при конфигурации коммутаторов. Предлагаемый выход из ситуации – устранение предполагаемых конфликтов путем применения технологии упорядочивания (сериализации) планов выполнения запросов, успешно применяемой в работе транзакционных систем обработки информации.

2. Решения проблемы реконфигурации ПКС

В рамках работы проведён краткий обзор и сравнительный анализ существующих решений проблемы реконфигурации ПКС по следующим критериям:

1. Требования для непротиворечивой реконфигурации сети [2]:
    - **Согласованность пакетов**: Множество правил, согласно которому обрабатывается отдельный пакет в сети, должно быть или полностью старым, или полностью новым;
    - **Отсутствие противоречий в целом**: новое множество правил, вычисляемых контроллером, должны быть непротиворечивым. Это означает, что итоговая реконфигурация не содержит аномалий.
    - **Согласованность потоков**: Все пакеты в одном потоке должны быть обработаны с помощью одной конфигурации в сети (строгая согласованность) или начало потока должно быть обработано одной конфигурацией сети, а завершение потока должен - другой конфигурацией (слабая согласованность)
2. **Масштабируемость**: насколько данное решение может быть применимо при увеличении:
    - Количества элементов сети
    - Конкурирующих запросов на её реконфигурацию

3. **Требовательность к ресурсам**. Работа сети во многом зависит от требований, предъявляемых решением к вычислительным ресурсам сетевых устройств, коммутаторов, как-то объём буферной памяти, наличие и объём ассоциативной тернарной памяти, производительность и прочее. Для целей нашего обзора сформулируем следующие критерии к сетевым устройствам:

   - тип, объём и производительность памяти,
   - вычислительная производительность,
   - пропускная способность устройства, как обобщающая характеристика.

   Кроме того, большое влияние на работу сети оказывает возможность динамического построения маршрутов для приоритетных потоков, их демультиплексирования на подпотоки с последующей сборкой. Таким образом, способность решения работать с большим количеством сетевых устройств также влияет на качество работы сети.

2.1 Хранение нескольких конфигураций

Данное решение является фактически продолжением работы, основывающейся на тегирование пакета [4]. Основная идея заключается в использовании дополнительного поля в пакете для хранения номера его версии, а также загрузки нескольких конфигураций на контроллеры. В таком случае мы имеем возможность одновременно обрабатывать несколько конфигураций.

Общий порядок работы данного метода следующий:

1. Загрузка дополнительных таблиц коммутации.
2. Обновление коммутаторов. Теперь в сети присутствуют две версии пакетов в сети, старые и новые. Новые пакеты хранят противоположный бит в заголовке, если старые хранят - 0, то новые - 1, и наоборот.
3. Дождаться окончания действия предыдущих пакетов.
4. Удаление старых таблиц коммутации.

Непротиворечивая реконфигурация сети. Непротиворечивость в рамках согласованности пакетов достигается в силу того, что после загрузки новых наборов правил в каждый коммутатор, старые пакеты обрабатываются старой конфигураций, а новые - новой. В таком случае выполняется и согласованность в плане пакетов

и в плане потоков. Итоговое отсутствие противоречий никак не проверяется.

Масштабируемость. В таблице правил каждого коммутатора должны присутствовать данные двух конфигураций, т.е. доступная память коммутатора, в таком случае, делится пополам. Это может привести к большим проблемам, как и в случае увеличение количества элементов сети, так и в случае конкурирующих запросов на её реконфигурацию. Также это может привести к ограничению функциональности, так как используемый бит распознавания версии потока фактически частично уменьшает функционал (он может занимать VLAN тэг или метку MPLS).

Требовательность к ресурсам. Данное решение является эффективным в рамках небольшой сети, когда число наборов правил в коммутаторах невелико. После загрузки таблиц нет никаких задержек по обработке пакета (в зависимости от бита применяется соответствующие правила).

2.2 Верификатор

Поведение сети описывается своими политиками. Описать поведение сети можно анализируя программы контроллера, т.н. верификация потоков управления, или анализируя установленные правила коммутации, верификация потоков данных. Для решения задачи анализа программ контроллера приходится рассматривать все потенциальные состояния сети, в которые может привести множество приложений контроллера, что в условиях отсутствия общего стандарта программирования на уровне NBI (North Bound Interface) практически сложно реализуемая задача. Анализ потоков данных - более достижимая задача. К таким средствам анализа потоков данных относится и средство VerMont [3], разработанный в Центре прикладных компьютерных сетей. VerMont решает задачу верификации (проверки) в оперативном режиме отправляемых с контроллера команд реконфигурации сети на предмет непротиворечия с политиками безопасности поведения сети. Средство становится в разрыв между контроллером и коммутатором, моделирует результат выполнения команды реконфигурации и в зависимости от результата соответствия принятым в сети политикам пропускает команду или блокирует её выполнение.

Непротиворечивая реконфигурация сети. Данное решение подразумевает, что перед внесением реконфигурационных изменений идёт предварительная проверка на непротиворечивость данной операции реконфигурации. В данном случае итоговая реконфигурация будет применена в случае, если итоговая реконфигурация бу-

дет полностью непротиворечивой. Требование покрывает согласованность потоков и пакетов в целом.

Требовательность к ресурсам и масштабируемость. VerMont спроектирован для работы в сетях, состоящих из не более чем 30 коммутаторов и обслуживающих не более 1000 потоков с интенсивностью обработки не более 1 OF команды секунду. Показатели количества коммутаторов и потоков могут быть увеличены на порядок за счет снижения интенсивности обработки OF команд на эту же величину.

2.3 Использование метода сериализации

Развиваемый в работе метод базируется на решении аналогичных проблемах, возникающих в системах управления баз данных [1]. В данном случае, проводятся аналогии понятий присущим SDN сетям и СУБД. Основная идея заключается в применении теории упорядочивания в работе транзакционных баз данных для решения проблем реконфигурации и построения плана выполнения конкурирующих запросов. Во время реконфигурации ведётся планирование запроса с целью избежать возникновение правил на коммутаторах, противоречащих политикам поведения сети.

Непротиворечивая реконфигурация сети. Использование подхода сериализации (упорядочивания) в плана запросов ведёт к обеспечению непротиворечивой работе сети при условии отсутствия аномалий, аналогичных для работы реляционных СУБД [1].

Требовательность к ресурсам и масштабируемость. Предлагаемый подход отвечает условиям требований к ресурсам и масштабируемости de-facto, как доказавший свою применимость на практике. В статье рассматривается адаптированный подход теории упорядочивания к построению непротиворечивого плана реконфигурации сети, количественная оценка качества его работы и требуемых ресурсов.

3. Использование метода сериализации транзакций при реконфигурации сети

Предлагаемый метод сериализации состоит в развитии подхода, сформулированного в [2], и адаптации алгоритмов, разработанных в рамках теории упорядочивания (Serializability Theory) для обеспечения эффективного функционирования транзакционных информационных систем. Суть метода – в построении глобально непротиворечи-

вого плана выполнения конкурентных запросов на реконфигурацию ПКС сети в целом.

Как ранее упоминалось, в случае ПКС аналогом транзакции является реконфигурация, состоящая из изменения таблиц маршрутизации в коммутаторах. Записи реляционной базы данных, или просто данные, которые подвержены аномалиям и являются неотъемлемой частью транзакций, логически соотносятся записям в таблице маршрутизации в ПКС сетях. Проследим, как методы работы с реляционными базами данных можно применить при реконфигурации ПКС сетей.

Транзакции реляционных баз данных, соответствующие изменению правил в таблицах маршрутизации, не могут происходить мгновенно. Мы должны обратить внимание на время доступа к объекту, время внесения изменений, время отклика о результате операций. В случае же ПКС-сетей, мы также обязаны дополнительно обратить внимание на скорость, латентность, загруженность каналов управления, а также – тип коммутатора (виртуальный или физический).

В работе реляционных баз данных нередки появления так называемых аномалий. Существует несколько типов аномалий: Dirty Write, Dirty Read, Lost Update, Fuzzy Read, Phantom, Read Skew, Write Skew. В статье [1], рассмотрены сути аномалий и их аналогии для процесса реконфигурации.

Понятия согласованности и синхронизации, приведенные ниже, позволят ввести количественные критерии управления процессом реконфигурации сети и оценки рисков нарушения политик поведения сети в случае неполной согласованности.

3.1 Понятие реконфигурации

Назовём процедурой реконфигурации вектора $P^i = ((O_1^i, e_1^i), ..., (O_n^i, e_n^i))$, где O_k^i - операция; e_k^i - объекты реконфигурации k-й операции в i-й процедуре.

Это означает, что на нектором шаге k некой процедуры реконфигурации P были последовательно выполнены операции $O_1 - O_n$ над, соответсвенно, объектами $e_1 - e_n$. В данном определении под объектами понимаются сущности графа сети (хосты, коммутаторы, марштуизаторы и т.д.). Операции же - это команды по реконфигурации сети.

3.2 Понятие согласованности

Согласованность в базах данных — согласованность данных друг с другом, целостность данных, а также внутренняя непротиворечи-

вость является одним из требований ACID. Это означает, что в результате транзакций не возникает аномалий. Введём данное понятие для терминов ПКС.

Рассмотрим $P = ((O_1, e_1), ..., (O_n, e_n))$;

Введём функцию $time(O_j, e_j) = (t^j_{st}, t^j_{en})$, где t^j_{st} - время начала, а t^j_{en} - время завершения процедуры реконфигурации j.

Также введём понятие: TTL^p_t. Которое охарактеризовывает значение TTL для пакета p в момент времени t. TTL^p_{Max} охарактеризовывает максимально возможное значение TTL перед сбросом пакета p.

Тогда объекты (O_j, e_j) и (O_k, e_k) являются согласованными, если:

$$(e_j \cap e_k) \quad and \quad ((time(O_j, e_j) \cap (time(O_k, e_k))) < \varepsilon) = \varnothing$$

и дополнительно выполнено:

$$\forall p_j \in e_j : TTL^{p_j}_{t^j_{en}} < TTL^{p_j}_{Max}, \quad \forall p_i \in e_i : TTL^{p_i}_{t^i_{en}} < TTL^{p_i}_{Max}$$

В случае если $\varepsilon = 0$, то это полная согласованность, никаких рисков нет. При росте ε возрастает вероятность появление конфликта. Требование к TTL означает, что реконфигурация должна быть своевременной. В сети должны остаться пакеты, для которых данная реконфигурация предназначалась.

Подробнее рассмотрим данное требование. Данное требование обеспечивается тем, что объекты реконфигурации, операции над некоторыми объектами, при взаимодействии в процедуре не могут приводить к аномалиям. Это может произойти в случае, если две процедуры выполняются на одних и тех же участках сети в одно и то же время. Т.е. для того, чтобы не возникало аномалий надо потребовать, чтобы выполнимость некоторых операций над одними и теме же участками сети происходили в разное время (в таком случае $\varepsilon = 0$). Но в таком случае, мы получаем упорядоченное выполнение операций, что может привести к низкой скорости обработки запросов. Увеличение ε означает, что может существовать сегмент времени, в котором на одних и тех же объектах работают в одно время некоторые объекты реконфигурации, приведёт к понижению безопасности сети, но повысит скорость обработки.

3.3 Понятие синхронизации

Аналогично понятию согласованности введем понятие синхронизации для процессов реконфигурации сети.

Рассмотрим $P = ((O_1, e_1), ..., (O_n, e_n))$;

Процедура реконфигурации называется синхронизированной, если для любых двух шагов, порядок выполнения которых не задан, выполняется:

Рассмотрим

$$P = (..., (O_i, e_i), ..., (O_j, e_j), ...) = (..., (O_j, e_j), ..., (O_i, e_i), ...) = P';$$

Это означает, что итоговый результат выполнения каждой из процедур одинаков. Понятие синхронизации – дополнительное требование целостности правил маршрутизации и их соответствия политикам поведения сети. В случае, если оно не выполнено, это может означать, что некоторые операции выполняются в разных адресных пространствах и могут привести в результате к возникшим аномалиям. Данный пример рассмотрен в статье [1].

4. Заключение

В данной работе представлено описание постановки задачи и краткий обзор существующих решений методов проблемы реконфигурации ПКС. В результате анализа выбран подход к решению на основе метода сериализации, предложены определения согласованности и синхронизации процедур рекофигурации.

В дальнейшем планируется дополнить обзор, сформулировать и доказать достаточное условие полного согласования запросов, провести программную реализацию в качестве приложения для контроллера RunOS.

Литература

1. Грушо А. А, Забежайло М. И, Зацаринный А. А, Писковский В. О *Безопасная автоматическая реконфигурация облачных вычислительных сред.* Системы и средства информ., 26:3 (2016), 83–92.

2. Грушо А. А, Терёхина И. Ю *Анализ непротиворечивости реконфигурации программно-конфигурируемой сети.* Системы и средства информ., 27:3 (2017), 12–22

3. V. Altukhov; V. Podymov; V. Zakharov; E. Chemeritskiy *VERMONT - A toolset for checking SDN packet forwarding policies on-line.* Published in: 2014 International Science

and Technology Conference (Modern Networking Technologies) (MoNeTeC)

4. Reitblatt M, Foster N, Rexford J, Schlesinger C, Walker D. Abstractions for network update. *Abstractions for network update* ACM SIGCOMM 2012 Conference on Applications, Technologies, Archi 2012

5. Hong, C. Y., S. Kandula, R. Mahajan, M. Zhang, V. Gill, M. Nanduri, and R. Wattenhofer. *Achieving high utilization with software-drivenWAN* ACM SIGCOMM 2013 Comp. Com. 43(4):15-26.

6. Eswaran P., Gray J., Lorie R., Traiger I. *The notions of consistency and predicate locks in a database system* Commun. ACM, 1976. Vol. 19. No. 11. P. 624-633.

7. C. J. Anderson, N. Foster, A. Guha, J.-B. Jeannin, D. Kozen, C. Schlesinger, D. Walker *NetKAT: Semantic foundations for networks* Acm sigplan notices. — 2014 — Т. 49, № 1 — С. 113—126.

Шмитов Н.О., Пашков В.Н.

Метод управления устройствами интернета вещей и обменом данных между ними в программно-конфигурируемых сетях

Введение

Интернет вещей (IoT) – это технология, которая объединяет устройства в компьютерную сеть и позволяет им собирать, анализировать, обрабатывать и передавать данные другим объектам с помощью программного обеспечения, приложений или технических устройств.

В настоящее время концепция Интернета Вещей находится на пике своей популярности. Это связано с увеличением количества устройств, взаимодействующих не только с пользователем, но и друг с другом [1].

Современный IoT состоит из разрозненных сетей. С развитием технологии, сети будут соединяться друг с другом и получать больше возможностей в области безопасности, аналитики и управления.

Это развитие ставит новые задачи перед разработчиками программного обеспечения для вычислительных и сетевых инфраструктур. Количество подключенных объектов в интернете вещей исчисляется миллиардами, а управление и контроль – сложная задача для большой распределенной сети, а с ней неразрывно связаны проблемы контроля и обеспечения безопасности.

Программно-конфигурируемые сети (Software-Defined Networks, SDN) как раз и являются одной из ключевых технологий, с помощью которой удастся масштабировать управление и реализовать необходимый контроль. В основе технологии SDN лежит концепция разделения контура передачи и управления данными, позволяющая осуществлять программное управление контуром передачи данных, которое может быть физически или логически отделено от аппаратных коммутаторов и маршрутизаторов. SDN обеспечивает гибкость и программируемость в сети IoT, не беспокоясь о базовой архитектуре существующих устройств.

Помимо этого, с помощью SDN контроллера можно решить проблемы балансировки нагрузки в сети и обеспечить защиту от некоторых видов уязвимостей, таких как:

1. Уязвимость к DDoS атакам.

2. Уязвимость к DoS атакам.

3. Уязвимость к атакам типа man-in-the-middle.

Реализация этих угроз в традиционной сети может приводить к ряду последствий, например, таких как: медленная работа или полная недоступность, нарушение обязательств по договорам обслуживания (Service-level agreements, SLA), неполученные прибыли, судебные издержки и так далее.

В связи с чем очевидна актуальность разработки методов управления устройствами интернета вещей и трафиком между ними в программно-конфигурируемых сетях. В данной статье предлагается интегрировать устройства IoT в программно-конфигурируемые сети (SDN) для управления обменом данными между ними с помощью SDN контроллера.

Целью данной работы является разработка и исследование методов управления устройствами интернета вещей и управления обменом данными между ними в программно-конфигурируемых сетях.

Для достижения поставленной цели необходимо решить следующие задачи:

1. Разработать метод, обеспечивающий обнаружение и классификацию IoT устройств в ПКС сети, аутентификацию, дистанционную активацию и деактивацию устройств, конфигурацию, управление конфигурацией сети и трафиком IoT устройств в ней.

2. Разработать приложение управления IoT устройствами и трафиком между ними для контроллера RUNOS 2.0.

3. Разработать методику тестирования разработанного приложения. Создать стенд, эмулирующий взаимодействие IoT устройств посредством ПКС сети.

4. Провести экспериментальное исследование и проанализировать полученные результаты.

1. Задача управления устройствами Интернета вещей и трафиком между ними в программно-конфигурируемых сетях

Пусть имеется программно-конфигурируемая сеть с фиксированной топологией под управлением контроллера. Управление сетью

осуществляется по протоколу OpenFlow не ниже версии 1.3 [2]. Сеть задана графом G (S, E, H) – коммутаторы, каналы связи, хосты. Пусть некоторое множество хостов является устройствами IoT. Необходимо найти конфигурацию сети, удовлетворяющую требованиям:

- Возможность распознавания IoT трафика
- Возможность классификации IoT устройств
- Изолированность (минимизация случаев внедрения устройств третьей стороны)

2. Обзор существующих методов интеграции устройств Интернета вещей в программно-конфигурируемые сети

2.1 Критерии обзора

В рамках работы был проведен обзор и сравнительный анализ существующих методов интеграции устройств Интернета вещей в программно-конфигурируемые сети по следующим критериям:

1. Управление и контроль.
2. Распознавание трафика устройств Интернета вещей — каким образом распознается трафик Интернета вещей.
3. Классификация устройств Интернета вещей — каким образом решается задача классификации устройств Интернета вещей для последующей задачи отслеживания нехарактерного поведения для каждого класса устройств.
4. Изолированность — реализация механизма для исключения случая внедрения устройств третьей стороны.

Стоит заметить, что ввиду различных факторов, таких как сложность практической реализации и отсутствие удобных программных средств для разработчиков, решения, включающего в себя реализации всех 4-х пунктов на данный момент не существует, поэтому рассмотренные ниже методы покрывают описанные критерии лишь частично, в лучшем случае три пункта из четырех.

2.2 Метод на основе использования контроллера умной среды (SEC)

Авторы метода [3] предлагают, опираясь на принципы SDN, создать контроллер среды (Smart Environmental Controller), который создает разделение между уровнем инфраструктуры, уровнем управления и уровнем приложения. Физические устройства представлены как виртуальные объекты, в то время как сетевой контроллер обеспечивает управляемое взаимодействие между устройствами, благодаря динамическому программированию домашнего шлюза на основе Open vSwitch с помощью OpenFlow. Выступая в качестве программного посредника между приложениями и плоскостью данных, SEC получает от пользователя простые команды для обмена данными между группами устройств. Таким образом, все указанные устройства результирующей сетевой конфигурации могут обмениваться данными, как если бы они находились в одной и той же физической локальной сети.

Управление и контроль. В рамках данного метода контроль обеспечивается централизованно с помощью рассматриваемого контроллера.

Распознавание трафика IoT устройств. В рамках данного метода не рассматривается ситуация, что в сети могут находиться устройства, не являющиеся устройствами Интернета вещей, поэтому метод распознавания IoT трафика отсутствует.

Классификация устройств Интернета вещей. Задача классификации не ставится.

Изолированность. Авторы предлагают использовать механизм авторизации, что является избыточным.

К достоинствам метода можно отнести то, что пользователь имеет доступ к своим устройствам из любой точки мира, а также изолированность получившейся сети и возможность постоянного мониторинга через контроллер. Данный метод является наиболее близким к описываемой далее реализации.

2.3 Метод распознавания устройств Интернета вещей на основе анализа сетевого трафика

В работе [4] авторы на основе собственного набора данных, собранного за несколько дней работы реальных физических устройств Интренета вещей, решают задачу классификации этих устройств с применением нескольких методов машинного обучения (Random Forest, XGBoost, GBM).

Управление и контроль. В рамках данного метода, контроль обеспечивается с использованием одноплатного компьютера Raspberry Pi. Однако, единственная задача, которая возлагается на него — это сбор данных с устройств Интернета вещей, что не решает задачи управления.

Распознавание трафика IoT устройств. В рамках данного метода не рассматривается ситуация, что в сети могут находится устройства, не являющиеся устройствами Интернета вещей, поэтому метод распознавания IoT трафика отсутствует.

Классификация устройств Интернета вещей. В этой статье авторы продемонстрировали, что устройство IoT можно точно идентифицировать на основе характеристик генерируемого им сетевого трафика. Их метод позволяет классифицировать устройства Интернета вещей, в том числе по маркам и моделям, с точностью 99,3%. Однако и используемый набор данных и исходный код не раскрываются авторами. Также в статье решается задача классификации для 9-ти классов, что является недостаточным для утверждения о надежности метода.

Изолированность. Авторы не поднимают вопрос изолированности.

К достоинствам работы хочется отнести крепкую теоретическую базу и реализацию алгоритмов с использованием неформального языка, но для проверки данного метода потребуется проводить эксперименты с реальными датчиками на протяжении продолжительного времени для создания собственного набора данных.

2.4 Метод использования контейнеров для управления устройствами Интернета вещей.

В данной работе авторы [5] описали подход, который обеспечивает новый способ построения цепочек сервисных функций, в которых компоненты можно вставлять и удалять во время работы цепочки. Этот подход особенно подходит для сценариев Интернета вещей, где ограниченная вычислительная мощность шлюза может потребовать динамический выбор компонентов с более высоким приоритетом для работы.

Управление и контроль. Именно реализация управления компонентами Интернета вещей является наиболее интересной темой, затронутой в данной работе. Компоненты работают в контейнерах на Docker v1.12. Это позволяет создать сеть Docker, контейнеры которой подключены к виртуальному коммутатору, управляемому контроллером SDN.

Распознавание трафика IoT устройств. Авторы не берут во внимание случай, когда в сети могут находится устройства, не являющиеся устройствами Интернета вещей, поэтому метод распознавания IoT трафика не предусмотрен.

Классификация устройств Интернета вещей. Задача классификации не ставится.

Изолированность. Авторы предлагают изолировать друг от друга группы устройств, подключенные к одному шлюзу. Контроллер SDN создает разделенные сети, однако каждая из этих сетей по-прежнему может быть подвержена воздействию злоумышленников.

К достоинствам метода можно отнести то, что разработанный авторами подход поддерживается двумя технологиями, программно-конфигурируемой сетью (SDN) и виртуализацией на основе контейнеров, которые обеспечивают ряд преимуществ с точки зрения гибкости, легкости программирования и универсальности.

3. Разработка метода

В рамках взаимодействия SDN-контроллера с IoT устройствами необходимо не только распознавать трафик этих устройств, но также обеспечить обнаружение и классификацию IoT устройств, аутентификацию, дистанционную активацию и деактивацию устройств, конфигурацию, управление конфигурацией сети и IoT трафиком в ней.

Для этого предлагается реализовать приложение для контроллера RUNOS 2.0 [6], и сеть mininet [7], куда подключить устройства IoT. Mininet – это эмулятор компьютерных сетей, включающих в себя хосты, коммутаторы и OpenFlow-контроллеры.

В приложении предлагается распознавать трафик от IoT-устройств, благодаря анализу содержимого приходящих пакетов, в которых можно легко определить прикладные протоколы IoT, зная структура пакетов того или иного протокола.

Для классификации IoT устройства после распознавания трафика предлагается внедрить один из методов машинного обучения.

Задача обеспечения изолированности может быть обеспечена путем реализации механизма авторизации, однако в рамках данной работы предлагается реализовать механизм аутентификации, используя цифровые сертификаты стандарта X.509.

Управление и контроль. Для реализации контроля в рамках сети предлагается разработать экспериментальный стенд. Для этого

необходимо связать между собой контроллер RUNOS 2.0, топологию mininet и симуляторы IoT устройств. В качестве основы предлагается использовать Cooja Simulator [8], так как это единственное популярное существующее на сегодняшний день решение для моделирования TCP трафика IoT и совместимое с эмулятором mininet, который, в свою очередь, совместим с контроллером RUNOS 2.0. Для Cooja уже существуют различные реализованные генераторы IoT трафика, однако при желании можно написать собственную реализацию на языке Java или C. В частности, предлагается использовать популярный пакет симуляторов SDN-WISE [9], в котором есть тестовые генераторы случайного траффика, именуемые далее motes и имитирующие траффик устройств общего назначения, а также архитектуру SDN-WISE.

Cooja – это межуровневый симулятор беспроводной сенсорной сети на основе Java, распространяемый вместе с Contiki. Он позволяет моделировать различные уровни от физического до прикладного уровня, а также позволяет эмулировать аппаратные средства набора сенсорных узлов.

Contiki — это операционная система для сетевых систем с ограниченным объемом памяти, ориентированная на беспроводные устройства Интернета вещей с низким энергопотреблением. Существующие варианты использования Contiki включают системы уличного освещения, звуковой мониторинг для «умных городов», радиационный мониторинг и сигнализацию. Это программное обеспечение с открытым исходным кодом, выпущенное по лицензии BSD.

Распознавание трафика IoT устройств. После анализа существующих IoT протоколов прикладного уровня была приведена следующая сравнительная характеристика. В таблице 1 приведено сравнение описанных выше протоколов по:

1. Используемому транспортному протоколу
2. QoS
3. Архитектуре
4. Используемому протоколу безопасности

В таблице 2 приведена сравнительная характеристика IoT протоколов по выполняемым операциям и целям, которым они служат.

Из полученных результатов можно сделать вывод, что существенные преимущества протокола CoAP резко отличают от других протоколов Интернета вещей, например, в отличие от MQTT, CoAP

Протокол	Трансп.	QoS	Архитектура	Безопасность
DDS	TCP/ UDP	Экстенсивная	Публикация/ Подписка	TLS/DTLS/ DDS
COAP	UDP	Маркировка	Запрос/Ответ	DTLS
MQTT	TCP	3 уровня	Публикация/ Подписка	TLS/SSL
XMPP	TCP	-	Запрос/Ответ Публикация/ Подписка	TLS/SSL
STOMP	TCP	-	Запрос/Ответ	TLS/SSL
AMQP	TCP	3 уровня	Запрос/Ответ Публикация/ Подписка	TLS/SSL

Таблица 1: Основные различия IoT протоколов

имеет небольшое количество сообщений, простые алгоритмы их обработки и поддержку многоадресной передачи. Протокол CoAP ориентирован на приложения M2M и обеспечивает взаимодействие сетей IoT с Интернетом через прокси-сервер CoAP. Также мы знаем структуру заголовка пакета CoAP. На основе анализа заголовков приходящих пакетов можно распознать этот протокол и понять, что трафик идет от IoT устройства. Поэтому разрабатываемое приложение будет анализировать пакеты, переданные с использованием данного протокола.

Классификация устройств Интернета вещей. Для реализации данной задачи предлагается протестировать несколько методов машинного обучения: random forest (RF), дерево решений (DT), SVM (с ядром rbf), k-ближайших соседей (KNN), искусственные нейронные сети (ANN) и Наивный байесовский классификатор (GNB). Для описания поведения сети были выбраны такие характеристики, как размер первых N пакетов, отправленных и полученных, и соответствующее время между прибытиями.

Изолированность. Как упоминалось ранее, предлагается использовать протокол X.509 (IETF RFC 5280), который обеспечивает наиболее безопасную аутентификацию и основан на цепочке сертификатов модели доверия.

Инфраструктура открытых ключей (PKI) состоит из древовидной структуры серверов и устройств, которые поддерживают список доверенных корневых сертификатов. Каждый сертификат содержит открытый ключ устройства и подписан закрытым ключом CA (Certificate Authority). Уникальный отпечаток обеспечивает идентифи-

Протокол	Назначение	Операции
DDS	Для сетей, нуждающихся в балансировке нагрузки	Процедуры получения и отправки данных
COAP	Для сети с ограниченными ресурсами, низким энергопотреблением	Процедуры записи и получения необходимых специфических параметров
MQTT	Для загруженных сетей со многими устройствами и посредником	Процедуры публикации / подписки
XMPP	Для адресации в небольшой персональной сети	Процедуры запроса информации, получения данных, установки новых значений
STOMP	Для сети, в которой возможно использование нескольких комбинаций разных протоколов, для которой требуется простой протокол для отправки сообщений через брокера	Процедуры публикации / подписки Транзакции
AMQP	Для балансировки нагрузки	Процедуры очередей и балансировки нагрузки

Таблица 2

фикацию, которая может быть подтверждена с помощью алгоритма шифрования, такого как RSA.

Цифровые сертификаты обычно образуют цепочку сертификатов, в которой каждый сертификат подписан закрытым ключом другого доверенного сертификата, и эта цепочка должна возвращаться к глобально надежному корневому сертификату. Эта схема устанавливает делегированную цепочку доверия от доверенного корневого центра сертификации (ЦС) до конечного «листового» сертификата объекта, установленного на устройстве через каждый промежуточный ЦС.

4. Реализация

4.1 Экспериментальный стенд

К сожалению, урезанные возможности Contiki по сравнению с “чистой” Ubuntu не позволили запустить контроллер Runos 2.0. В документации к Runos, можно заметить, что требуется Ubuntu 18.04 и выше. Contiki же разработана на базе Ubuntu 14.04. После обновления Contiki, по аналогии, с обновлением классической Ubuntu до версии 18.04, контроллер также не удалось запустить.

В связи с невозможностью создания стенда в рамках операционной системы Contiki было решено импортировать Cooja в Ubuntu 18.04, а позднее в Ubuntu 20.

Как было упомянуто выше, симулятор Cooja реализован на языке Java. Конкретизируя, Java 7, которая не поддерживается на современных Ubuntu, поэтому в рамках данной работы симулятор был полностью адаптирован под современные операционные системы.

Также, оригинальный симулятор Cooja совместим только с контроллерами на языке Java, в частности на контроллер Onos. Этот момент был исправлен в разработанной новой версии Cooja. В текущей версии контроллер подключается по IP при запуске симулятора. Этого удалось достигнуть путем перепрограммирования интерфейса подключения контроллера.

От оригинального Cooja в новую версию была перенесена полная совместимость с топологиями mininet, поэтому создание экспериментального стенда Runos + mininet + IoT теперь стало возможным.

На рис. 1 изображена архитектура Cooja. В итоговой версии был переписан Cooja Control Plugin, обеспечивающий взаимодействие между симуляторами и библиотекой Cooja control Library, что обеспечило возможность работы на Ubuntu 18.04 и выше. Были переписаны симуляторы (motes), написанные на Java версии 7 и младше, так как иначе они некорректно работали на операционных системах с версией Java старше. Был переписан интерфейс взаимодействия с контроллером, позволяющий подключаться к контроллеру по IP-адресу. Это было сделано для того, чтобы можно было подключить любой контроллер, а не только контроллеры, реализованные на языке Java. Были исправлены и полностью переписаны классы, отвечающие за отрисовку XML объектов, а также функции сохранения конфигурации модели и восстановления из файлов. Помимо этого, для корректной работы симулятора Cooja на виртуальную машину были перенесены все необходимые утилиты операционной системы Contiki, и обновлены в местах, где программный код устарел.

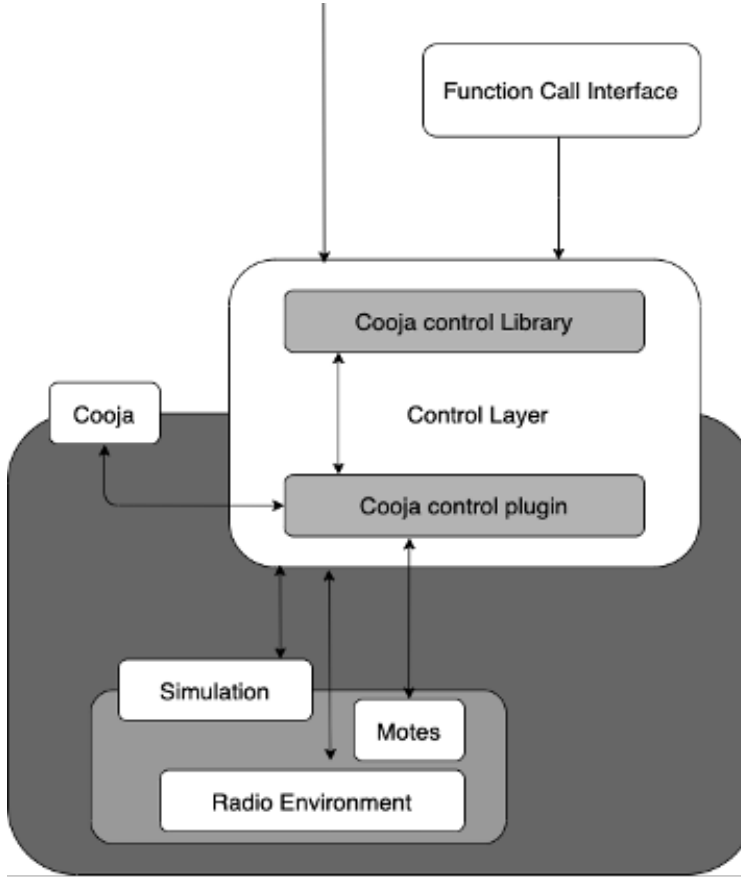

Рисунок 1. Архитектура Cooja

4.2 Задача классификации

Каждый двунаправленный поток между IoT устройством и портом назначения описывается вектором признаков, состоящим из следующих элементов:

- размер первых N отправленных пакетов;
- размер первых N принятых пакетов;
- время между прибытиями N - 1 пакетов среди первых N отправленных пакетов;

- время между прибытиями N - 1 пакетов среди первых N полученных пакетов.

Все векторы признаков должны иметь одинаковый размер. Если двунаправленный поток содержит менее N пакетов, остальные значения полей устанавливаются в ноль. Значение N устанавливается опытным путем. Чем больше устройств мы должны классифицировать, тем больше должно быть N, чтобы у классификатора было достаточно информации, чтобы точно различать двунаправленные потоки, генерируемые разными устройствами. Между тем, N должно быть как можно меньше из-за проблем, связанных с производительностью.

Большинство рассматриваемых алгоритмов классификации имеют параметры регуляризации, также называемые гипер-параметрами, которые необходимо настроить. На этапе обучения набор проверки, состоящий из 25% обучающего набора, используется для точной настройки гипер-параметров. После того, как лучшие параметры будут найдены, классификатор переобучается на всей обучающей выборке. Производительность различных алгоритмов измеряется на тестовом наборе. Используемые метрики: accuracy, precision, recall и оценка F1. Точность классификатора — это доля правильно классифицированных потоков. В рамках текущей реализации рассматривается задача 4-х классной классификации. Четыре класса: device1, device2, device3 и device4.

5. Экспериментальное исследование

В рамках экспериментального исследования необходимо проверить работоспособность разработанного стенда, а также провести сравнительный анализ различных метрик машинного обучения для методов задачи классификации с целью выявления наиболее оптимального подхода.

Экспериментальный стенд состоит из:

- Контроллер Runos 2.0 с приложением learning-switch (коммутаторы узнают MAC-адреса узлов в сети по адресам хостов с которых приходят пакеты)
- Топологии Mininet
- Адаптированная среда Cooja для симуляции трафика IoT по протоколу CoAP

Экспериментальный запуск проводился на топологии с тремя симуляторами IoT: датчик движения, видеокамера и веб-трафик.

В результате проведенных экспериментов была доказана полная работоспособность спроектированного стенда. Была успешна развернута архитектура SDN-WISE в симуляторе Cooja с обновленным ядром.

Эксперимент также показал, что спроектированный стенд работает не только с симуляторами, идущими вместе с SDN-WISE, но и с любыми симуляторами, реализованными на языке C.

RUNOS был собран с приложением learning-switch, все хосты в реализованной топологии пинговались и отправляли пакеты на свитчи.

Стенд готов для реализации приложения с возможностью управления траффиком.

Задача классификации. Результаты полученных метрик приведены в таблице 3. Общая точность классификатора 99,9% была достигнута классификацией методом случайного леса (RF). Необходимо провести больше экспериментов, чтобы оценить этот метод на больших наборах данных, а именно следует опробовать в сетях с большим количестве различных устройств Интернета вещей, при этом различных не только по типу, но и по характеру генерируемого трафика (частота, объем и т.д.).

	accuracy	Micro-av. precision	Micro-av. recall	Micro-av. F1 score
RF	.999	.999	.999	.999
DT	.995	.995	.995	.995
SVM	.993	.993	.993	.993
KNN	.989	.989	.989	.989
ANN	.986	.986	.986	.986
GNB	.919	.919	.919	.919

Таблица 3: Результаты экспериментального исследования

Заключение

В рамках данной работы:

1. Разработан метод, обеспечивающий обнаружение, классификацию и аутентификацию IoT устройств в SDN сети, управление конфигурацией сети и трафиком IoT устройств в ней.

2. Разработан стенд, эмулирующий взаимодействие IoT устройств посредством SDN сети.

3. Приведена программная реализация метода классификации IoT устройств в SDN сети.

4. Разработана методика тестирования экспериментального стенда.

5. Разработана методика тестирования классификатора IoT устройств.

6. Проведено экспериментальное исследование.

В будущем необходимо также реализовать механизмы дистанционной активации и деактивации устройств IoT и разработать приложение для контроллера RUNOS 2.0., которое будет использовать функционал разработанных методов.

Литература

1. Смелянский Р. Л. *Программно-конфигурируемые сети. Открытые системы.* М.: СУБД, №9, 2012.

2. Open Networking Foundation. OpenFlow Switch Specification. Version 1.3.0 `https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-spec-v1.3.0.pdf`

3. Nicolas Le Sauze, Mathieu Boussard *SDN in LANs: Programming the Network to Secure IoT Traffic.* M.: Nokia Bell Labs, 2016.

4. Yair Meidan, Michael Bohadana, Asaf Shabtai *ProfilIoT: A Machine Learning Approach for IoT Device Identification Based on Network Traffic Analysis* M.: SAC '17: Proceedings of the Symposium on Applied Computing, 2017.

5. Roberto Morabito, Nicklas Beijar *A Framework based on SDN and Containers for Dynamic Service Chains on IoT Gateways* M.: HotConNet '17:, 2017.

6. RUNOS 2.0 `https://github.com/ARCCN/runos`

7. Mininet `http://mininet.org/`

8. Cooja Simulator `https://github.com/contiki-os/contiki/wiki/An-Introduction-to-Cooja`

9. SDN-WISE `https://sdnwiselab.github.io`

Королев Л. Н.

Об эволюционных алгоритмах, нейросетевых вычислениях, генетическом программировании – математические проблемы

Начиная с 80-х годов прошлого столетия получило развитие направление научных и прикладных исследований, получившее название в смысловом русском переводе – «обучение вычислительных машин». В англоязычной литературе это направление относят к Computer Science или к IT-технологиям и называют “Machine Learning” (ML).

Цель статьи — попытаться разобраться с теми допущениями, ограничениями и математическими проблемами, которые возникают в этой области исследований, основанной главным образом на эвристиках и на гипотезах, в той или иной степени адекватных предметной области, в рамках которой решаются задачи методами ML.

В это направление исследований и применений входят эволюционные и генетические алгоритмы, генетическое программирование, нейросетевые вычисления (нейрокомпьютинг), клеточные автоматы.

Появление обобщающего термина Machine Leaning связывают с именем K. Samuel, опубликовавшим в 1963 году статью [1] о некоторых проблемах обучения машины на примере игры в шахматы.

1. О нейронных сетях.

С формальной точки зрения нейронную сеть можно представить как частный случай параметризованного преобразователя, на вход которого поступает вектор значений, и на выходе которого появляется другой вектор, возможно из другого векторного пространства.

Такого типа преобразование в обычной математической символике можно записать, как систему из m функций вида:

$$\begin{aligned} Y_1 &= F_1(x_1, x_2, \ldots, x_n, \omega_{11}, \omega_{21}, \ldots, \omega_{k1}) \\ Y_2 &= F_2(x_1, x_2, \ldots, x_n, \omega_{12}, \omega_{22}, \ldots, \omega_{k2}) \\ &\ldots \\ Y_m &= F_m(x_1, x_2, \ldots, x_n, \omega_{1m}, \omega_{2m}, \ldots, \omega_{km}) \end{aligned} \qquad (*)$$

Визуализировать эту систему функций * можно в виде схемы Рис. 1, которая очень похожа по виду на однослойную нейронную сеть.

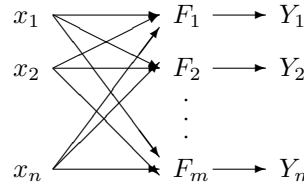

Рис. 1

В отличие от преобразователя общей формы в нейронной сети все функции F_i имеют одинаковый вид, например, такой простейший:

$$F_i = \begin{cases} 1, & \text{если } \sum\limits_{1}^{n}(x_i\omega_i) > v, \\ 0, & \text{если } \sum\limits_{1}^{n}(x_i\omega_i) \leqslant v. \end{cases}$$

Существенно отметить, что значения всех функций в системе (*) могут вычисляться одновременно. Это означает, что преобразованиям этого типа присущ внутренний, естественный параллелизм.

Следует также сделать следующее важное замечание. Рис. 1 можно трактовать как представление однослойной параллельно работающей вычислительной сети. На практике чаще всего используются многослойные сети по той причине, что при использовании только одного слоя при вычислении результирующих значений функций F_i теряется информация о параметров сети, связанных с другими функциями.

Впервые работы по формальному анализу математических моделей нейроподобных сетей относятся к 1943 г., имеется в виду работа Мак Коллока и Вальтера Питтса [2] "Логическое исчисление идей, относящихся к нервной активности". На русский язык эта работа переведена в 1956 г.

Фундаментальный анализ математических моделей нейронных сетей был проведен рядом известных специалистов в области математической логики, включая С. К. Клини, М. Минского и С. Пейперта и ряда других.

Важным шагом в развитии идей нейрокомпьютинга явилось появление модели персептрона Розенблатта [3] (1962), которая с успехом была использована для решения некоторых задач распознавания образов.

Модель персептрона Розенблатта была подвергнута всестороннему математическому анализу в работе Минского М. и Пейперта

С. [4], которые доказали невозможность использования персептрона для решения некоторых простых задач распознавания геометрических объектов, в частности, невозможность распознавать геометрически подобные образы.

Затем были предложены многочисленные усовершенствованные модели нейронных сетей, которые снимали ряд ограничений персептрона и расширяли возможный круг решаемых задач. Однако мощностей существовавшей на тот период вычислительной техники было явно недостаточно для решения многих реальных задач распознавания и классификации.

Анализируя первый этап развития исследований нейронных сетей, можно вполне определенно сказать, что эта область относилась к математике, к исследованиям некоторого класса математических моделей, отражающих, в том числе и в терминологии, физиологические реалии.

Сложность и необычность этих исследований заключалась в том, что нейронные сети по своей сути являются моделями параллельного процессирования, в которых все узлы (нейроны) работают одновременно и одновременно обмениваются информацией между собой.

Замечательным свойством математических моделей нейронных сетей, используемых для вычислений, является принцип их «обучения» и «самообучения», их настройки на решение определенного класса задач.

Процесс «обучения» и «самообучения» состоит в подборе параметров $\omega_1, \omega_2, \ldots, \omega_k$ так, чтобы «выходное» векторное пространство обладало предопределенными свойствами.

В ряде задач множество выходных значений Y_i должно обозначать номер класса, к которому отнесен входной вектор, в других задачах это множество представляет собой сжатую информацию, в третьих – аппроксимированное значение некоторой функции и т. п.

Как уже отмечалось, идеи генетического программирования (Genetic Programming) возникли в конце 50-х годов прошлого столетия [5]. Большую роль в развитии этих идей сыграл профессор J.Koza, опубликовавший в 1992 г. первую монографию, посвященную многим аспектам этого нового направления в автоматизации программирования [13], [14].

Прежде чем говорить о генетическом программировании следует ввести в круг идей, связанных с эволюционными алгоритмами вообще и с генетическими алгоритмами в частности.

2. Эволюционные алгоритмы

Эволюционные алгоритмы можно отнести к известному методу «проб и ошибок», связанному с решением переборных задач, возникающих при поисках правильных решений или экстремальных значений, в случае отсутствия знания или теории о поведении изучаемого объекта, процесса или функции.

Способ решения таких плохо формализованных, плохо определенных задач – это направленный перебор.

В непрерывной математике широко используется метод градиентного спуска для поиска экстремумов функций, в дискретной математике используется метод ветвей и границ, позволяющей отсечь большое число вариантов, которые заведомо не приведут к нахождению правильного решения.

В том и в другом случае отыскания экстремума направленным перебором необходимы сведения о характеристиках функции или функционала, экстремум которых надлежит отыскать. При решении практических задач эти сведения можно получать в ходе решения.

Пусть нам задано конечное множество из N уникальных элементов. Уникальность элементов означает, что каждый из них имеет свое присущее только ему имя и обладает некоторым набором приданных ему характеристик. Каждому элементу этого множества может быть приписан свой порядковый номер. Если в нашем множестве N элементов, то мы можем присвоить им порядковые номера $N!$ различными способами. Пусть с каждой нумерацией связано некоторое значение и существует алгоритм вычисления этого значения. Функцию, зависящую от способа нумерации элементов множества, мы будем называть функцией, заданной на перестановках или просто функцией перестановок: $F(p)$.

В качестве имен элементов можно выбрать номера от 1 до N и назвать эту последовательность начальной перестановкой $p_1 = (1, 2, 3, \ldots, N)$. Как известно, любую другую перестановку можно получить последовательной переменой мест двух соседних элементов перестановок, соответственно, все перестановки можно перенумеровать от 1 до $M = N!$

Многие задачи сводятся к поиску экстремумов функций перестановок. Классическим примером функции, заданной на перестановках, является задача коммивояжера. Если перенумеровать все города, которые должен обойти коммивояжер, если заданы расстояния между любой парой городов, то перестановку можно считать порядком обхода городов, а сумму пройденного расстояния можно считать значением функции на выбранной перестановке.

При небольшом числе городов эту задачу можно решить прямым перебором. Но если N большое, например 300, то, по формуле Стирлинга ($n! \approx \left(\frac{n}{e}\right)^n \sqrt{2\pi n}$) $300! > 10^{600}$.

С таким числом переборов не справится даже очень мощный компьютер!

Другим примером задачи, сводящейся к поиску экстремума функции на перестановках, является хорошо известная задача о рюкзаке.

Её формулировка в простейшем случае сводится к следующему. Дано N предметов, каждый из которых обладает своим весом и стоимостью. Рюкзак может быть нагружен только до определенного веса. Следует из имеющихся N предметов положить в рюкзак наиболее ценные, не превысив допустимый вес. Здесь с каждым предметом связано уже два значения – вес и стоимость (ценность).

И ту, и другую задачу можно попытаться решать, выбирая одну перестановку за другой наугад, в надежде случайно наткнуться на удачную перестановку, доставляющую искомый экстремум функции или приближающую к экстремуму.

Здравый смысл подсказывает, что подобные задачи нужно решать направленным перебором, отбрасывая сразу недопустимые варианты последовательности обхода городов, или последовательности загрузки рюкзака. Но для этого, необходимо найти законы, определяющие правила построения «хороших» перестановок.

Для этого можно поступить так. Выбрать наугад несколько перестановок, подсчитать для всех соответствующие значения функции, внимательно проанализировать их состав, и выработать стратегию формирования новых подстановок, приближающих значения функции к искомому экстремуму.

Если, например, мы ищем максимум функции, то нам следует пробовать выбирать для дальнейшего исследования перестановки, чем-то похожие на изначально случайно выбранные перестановки с большими значениями функций от них. Однако если ориентироваться только на «хорошие» перестановки, то можно оказаться в локальном максимуме. Чтобы этого не случилось надо продолжать пробовать случайные перестановки в надежде не оказаться в локальном экстремуме.

Формально это означает, что для получения новых перестановок на основе набора выбранных ранее перестановок нам необходима операция генерации похожих перестановок и операция, удаляющая нас от «хороших» перестановок. Эти две операции для некоторых задач могут быть формально определены, как некие алгебраические операции над перестановками (См.[15], [16])

Природа «выдумала» свой метод отыскания правильных решений, обеспечивающих улучшение приспособленности особей живой природы к среде обитания, вообще говоря, не требующий априорных знаний о функции или функционале.

Эволюционная теория Дарвина представляет собой гипотезу, хорошо согласующуюся с фактами, которая объясняет механизмы, появления устойчивых популяций организмов (видов), наилучшим образом приспособленных к обитанию и размножению. Это механизмы «скрещивания», «мутации» и «селекции» (отбора особей для воспроизведения новых популяций вида и продолжения рода).

Современная генетика раскрыла детали механизмов скрещивания и мутации, происходящих на молекулярном уровне в цепочке генов или геноме. Геном особи содержит всю необходимую информацию для формирования конкретной особи, и предопределят ее приспособленность к жизни в среде обитания.

Естественно возникло желание применить механизмы естественного отбора для решения задач поиска наилучших решений в тех областях, где отсутствуют априорные знания об объекте, экстремальное значение которого необходимо отыскивать.

В математике давно известны методы, позволяющие находить локальные экстремумы функций. Первый шаг численных методов поиска экстремума, как правило, состоит в том, что на некотором отрезке определения функции, «разбрасываются» случайным образом несколько точек, в которых вычисляются значения функции. Следующие шаги состоят в том, что выбираются точки с наибольшим (с наименьшим) значением функции и вблизи от них отыскивается новые «перспективные» точки для дальнейшего анализа с использованием, например, идей градиентного спуска. Выбор перспективных точек можно назвать селекцией, получение новых точек можно назвать процессом генерации новых поколений точек. Подобные численные методы по своей сути являются итерационными, и каждый шаг итерации состоит в селекции объектов и получении новой их генерации для дальнейшего анализа. Алгоритмы такого типа называются, следуя Дарвину, эволюционными. В настоящее время трудно назвать автора, который ввел этот нематематический термин в численные методы.

Возникло множество эволюционных алгоритмов, использующих в разных видах механизмы типа естественного отбора.

Для обозначения некоторых проблем, возникающих при попытках моделирования природных процессов, рассмотрим простой пример задачи поиска глобального минимума некоторой функции одного переменного.

Пусть мы не знаем о функции ничего, кроме того, что нам доступен алгоритм, позволяющей вычислить ее значение в любой произвольно выбранной точке. Имея в виду компьютерную обработку, мы будем предполагать, что поиск глобального экстремума производится для функции, заданной только в конечном множестве дискретных точек и, следовательно, на конечном сегменте.

Это сразу же отсекает возможности точного применения хорошо развитых математических методов поиска экстремумов функций, заданных своими дифференциальными свойствами.

Для поиска глобального максимума так заданных функций часто используется эволюционный алгоритм, получивший название генетического алгоритма [10, 11, 12], который состоит в следующем:

- Возьмем наугад несколько n значений $x_i^{(1)}$, принадлежащих области определения. Вычислим (отыщем) значения $y_i^{(1)} = f(x_i^{(1)})$ во всех этих точках, а далее будем рассуждать, и действовать так:
- Расположим $x_i^{(1)}$ в порядке возрастания соответствующих значений $y_i^{(1)}$;
- будем считать, что значения $x_i^{(1)}$, соответствующие большим значениям $f(x_i^{(1)})$, находятся ближе к искомому максимуму нашей функции, нежели другие и будем считать их «хорошими» (*отметим тут же, что это предположение есть гипотеза, основанная только на интуиции*);
- далее, мы сформируем новую последовательность $x_j^{(2)}$ той же длины n, но уже не совсем случайным образом, а так, чтобы часть новых $x_j^{(2)}$ находились бы поблизости от «хороших» точек $x_j^{(2)}$, а другая часть, во избежание попадания в локальный максимум, охватывала бы другие, «далекие» области определения функции, а часть была бы отброшена;
- получив новую последовательность $x_j^{(2)}$, которая в сложившейся терминологии называется «популяцией», мы будем генерировать таким же образом все новые и новые популяции.

Этот процесс построения все новых и новых популяций $x_j^{(k)}$ будем продолжать в надежде, что на каком то шаге мы найдем точку, в которой функция $f(x)$ достигнет своего максимума. Чем может быть подтверждено, что наша надежда оправдалась, что мы действительно на некой итерации получили значение координаты искомого минимума?

Например, тем, что новые итерации не дают уже улучшения результатов, или на основании того, что с точки зрения эксперта вся область определения функции, экстремум которой мы ищем, достаточно «плотно» обследована. Эти и многие другие критерии завершения процесса итераций генетического алгоритма, очевидно, не имеют ничего общего с формальными строгими математическими доказательствами правильности полученных результатов.

В генетических алгоритмах используется терминология, заимствованная из эволюционной теории Дарвина, которая перекочевала в чисто математические исследования, не имеющие ничего общего с биологией.

В нашем примере последовательности $x_j^{(k)} = \left\{x_{j1}^{(k)}, x_{j2}^{(k)}, \ldots, x_{jn}^{(k)}\right\}$ называются популяциями, сами x_i называются *геномами*, порождающими значения функции, а сама функция называется функцией *выживаемости* или *функцией качества* (fitness function).

Выше мы говорили о функциях, заданных на перестановках, в этом смысле геном есть перестановка, популяция – набор исследуемых перестановок, функция качества – перестановочная функция.

Способ получения точек новой популяции на основе анализа точек (геномов) предыдущей популяции, именуются операциями и *скрещивания и мутации*. Выбор «хороших» точек и отбрасывание плохих можно назвать *селекцией*.

Обращаясь к нашему простому примеру, важно заметить следующее. Если мы будем каждый раз выбирать для получения новой популяции только хорошие геномы предыдущей популяции, велика вероятность «зациклиться» около точки локального максимума и не найти популяцию, содержащую глобальный максимум. Во избежании такого явления живая природа придумала процесс мутации – случайного изменения какого-то элемента генома, приводящего к изменчивости отдельной особи либо в лучшую, либо в худшую сторону. В соответствии с этим механизмом изменчивости в генетических алгоритмах используют операцию *мутации*, которая обеспечивает, как правило, выход из зацикливания около «хороших» точек. Более сложную операцию «скрещивания» можно интерпретировать, как попытку сформировать новую особь, смешением признаков двух других особей (родителей).

После популярного изложения идеи генетического алгоритма можно перейти к более формальному изложению механизмов основных операций, используемых в вычислениях.

Общую ситуацию, в которой действует генетический алгоритм в нашей простой задаче отыскания максимума функции, следует представлять следующим образом.

Имеется «черный ящик», устройства которого мы не знаем, но который, получая на вход некоторые данные, на выходе выдает некоторые значения, которые можно сравнивать между собой, ранжируя их по величине или значимости.

На вход могут поступать характеристики некоторых объектов, которые мы будем называть геномом особи. Геном должен быть закодирован некоторой последовательностью символов. С вычислительной точки зрения любой код для компьютера представляет собой структурированную последовательность двоичных разрядов. Структурированную в том смысле, что вся двоичная последовательность разбита на поля (подпоследовательности), несущие некоторую смысловую нагрузку, определяемую постановкой задачи.

Двоичные элементы такой последовательности часто называют *генами*, а смысловые поля последовательности иногда называют хромосомами. Сразу же отметим, что в формальной постановке задач, решаемых генетическими алгоритмами, эти термины не связаны с исходным их биологическим смыслом.

В общем случае можно считать, что на вход «черного ящика» поступают данные в двоичном коде, на выходе мы получаем закодированные в двоичном виде результаты, с помощью которых можно упорядочить геномы популяции, генерируемой на каждом шаге работы алгоритма.

Операция *скрещивания* формально состоит в том, что из популяции выбираются геномы двух «родителей» и из них формируются геномы нескольких «потомков». По некоторым правилам, напоминающим перекрестное опыление: для потомков часть хромосом (блоков генов) берется от одного родителя, часть хромосом берется от другого, они объединяются, но так чтобы их общее число, а иногда и место расположения, при этом оставалось таким же, как и у родителей. В классических генетических алгоритмах общая длина двоичного генома у всех особей всех популяций одинакова.

Одноместная операция *мутации* состоит в том, что у *случайно* выбранного представителя популяции, *случайно* меняется один двоичный разряд (ген) на противоположный по значению и так получают значение нового представителя. Слово «случайно» означает, что этот процесс производится с некоторой заранее заданной вероятностью.

Операция селекции состоит в выборе геномов, полученных в результате скрещиваний и мутаций, для формирования новой популяции для следующего шага итерации генетического алгоритма. При организации селекции также используется вероятностный подход, т. е. задаются вероятности выбора новых геномов и оставления в по-

пуляции старых особей.

Перечисленные выше вероятности являются параметрами генетического алгоритма, и от их выбора в значительной степени зависит качество работы самого алгоритма, успех в решении поставленной задачи. Входными параметрами генетического алгоритма являются также размер генома, заданный размер популяции и число шагов итерационного процесса, необходимых для получения приемлемого решения.

В 1975 году Holland [5] доказал теорему, известную под названием теоремы схем (schemata theorem).

В очень грубом приближении, содержательно «теорема схем» утверждает, что с увеличением числа итерационных шагов генетического алгоритма, вероятность приближения к экстремуму функции качества возрастает.

Теорема исходит из того, что геном представляет собой двоичную последовательность, у которой каждый ген (двоичный разряд) может изменяться независимо от других в результате действий операций скрещивания и мутации. Теорема утверждает, что от поколения к поколению в результате операции селекции, которая оставляет в следующем поколении в основном геномы с «хорошим» значением функции качества, растет (по вероятности) число разрядов в геноме, не меняющихся от поколения к поколению.

Поясним утверждение теоремы на примере. Пусть геном представляет собой последовательность n двоичных разрядов. Рассмотрим популяцию некоторого поколения, состоящую из M особей, обладающих самыми хорошими значениями функций качества. Каждому разряду генома поставим в соответствие частоту повторяемости его значения. Например, значение «0» генома с номером i повторяется в популяции раз, значение «1» этого гена повторяется $M - k$ раз. Если $k \approx M - k$, то будем считать такой ген «случайным» или неинформативным и обозначим его неопределенное значение символом «*», если же $k \gg M - k$, то будем считать, что вероятность закрепления за геном некоторого значения пропорциональна частоте его появления в популяции. Это позволяет нам построить «схему» генома, отражающую степень закрепления определенных значений генов в популяции. Эта схема будет выглядеть, например, так:

0	0	*	*	1	0	1	1	*	0

Рис. 2

Это означает, что в популяции гены, помеченные *, достаточно случайным образом меняют свое значение от особи к особи, случайно меняя значение 0 на 1 или, наоборот. Значения, помеченные в этом

наборе 0 или 1, говорят о том, что в популяции соответствующие гены принимают у большинства особей означенные значения.

При некоторых предположениях о вероятностях мутаций, скрещиваний и правилах селекции, вероятность появления неопределенностей типа * постепенно убывает от популяции к популяции, символы * постепенно исчезают из схем.

Геном, с заданными значениями всех его разрядов, можно интерпретировать как одну вершину n-мерного куба. Множество вершин, удовлетворяющих схеме, в которой закреплены значения лишь за несколькими разрядами, представляет собой несколько вершин, образующих аналог гиперплоскости в обычном n-мерном пространстве.

Если исследуемый геном представляет собой последовательность двоичных разрядов длины n, то пространство всех возможных двоичных последовательностей такой длины есть 2^n. Например, при $n = 64$ число $2^n \approx 10^{18}$. Прямой перебор для решения задачи отыскания экстремума «функции выживаемости» потребует огромного числа вычислений. Практические эксперименты на задаче о рюкзаке показывают, что для достижения заданной цели требуется приблизительно около 200 итераций при размере популяций порядка 40 геномов в каждой. Тем самым потребуется вычислить $200 * 40 \approx 10^4$. Ускорение, даваемое генетическим алгоритмом по сравнению с прямым перебором, равно 10^{12}, Работа ускоряется в миллиарды раз.

При некоторых искусственных предположениях математически точно доказывается сходимость генетического алгоритма к точному решению поставленной задачи.

Для некоторых реальных задач, как уже говорилось, точное доказательство того, что алгоритм выдал нам точное (или близкое к точному) решение трудно получить, и это составляет один из недостатков генетических вычислений.

3. О генетическом программировании

В модели с «черным ящиком» возможна и другая постановка задачи. Пусть мы знаем реакцию черного ящика на входные данные, которые мы также будем называть геномами. Нас интересует, по какой программе работает этот аппарат, выдавая значения некоторой вычисленной им функции.

Задача генетического программирования как раз сводится к конструированию программы, выполняющей аналогичное черному ящику преобразование данных с использованием идей генетических алгоритмов. Речь идет о конструировании реальной вычислительной

программы, которую можно задать вычислительной машине для ее реального исполнения, то есть о программе, закодированной на одном из формальных алгоритмических языков, доступных для понимания компьютером.

Формальный механизм генетического программирования состоит в следующем.

Модель любой программы представляется в виде графа или списка в смысле алгоритмического языка программирования логических задач – ЛИСП.

Для простоты дальнейшего изложения будем рассматривать программу как визуально заданный граф.

Любое вычисление арифметического выражения можно интерпретировать как поток вычислений, определяемый соответствующим графом. Например, рассмотрим простое выражение:

$$y = ((a + b) * c + (a - d) * x) * r$$

Это выражение можно представить графом потока данных и операций над ними (Рис. 3):

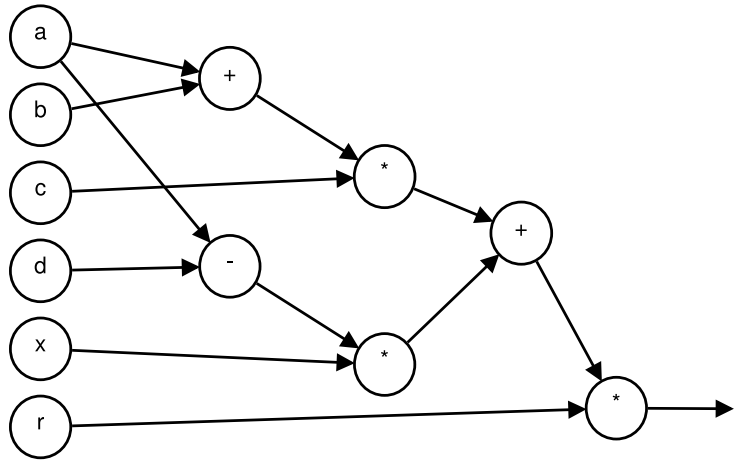

Рис. 3

Последовательность вычислений, выраженных этим графом, может быть таковой

$$a \Rightarrow b \Rightarrow + \Rightarrow c \Rightarrow * \Rightarrow a \Rightarrow d \Rightarrow - \Rightarrow x \Rightarrow * \Rightarrow + \Rightarrow r \Rightarrow *.$$

Эту последовательность символов, отображающих граф вычислений, мы будем считать геномом, точнее набором хромосом, несущих на себе следующую семантическую нагрузку:

Коды букв обозначают переменные и константы, коды символов операций – набор допустимых в компьютере арифметических действий. Код символа $\Rightarrow$ обозначает стрелку в графе программы.

Теперь, соблюдая некоторые очевидные правила, случайно выберем последовательность этих трех типов символов, которые образуют программу, которая что-то может вычислять. Подставим в эту случайно выбранную программу входные значения, воспринимаемые нашим черным ящиком. Наверное, результат вычислений по такой случайно сформированной программе будет значительно отличаться от ожидаемого значения. Ошибку в вычислениях мы будем считать значением функции качества генетического алгоритма, конструирующего нужную машинную программу.

Тем самым мы определили вид генома, функцию качества и для реализации генетического алгоритма нам недостает разумно определить операции скрещивания, мутации, селекции, размер популяции и критерии окончания итерационного процесса по выбору правильной программы, которая минимизирует ошибку в вычислении отклика черного ящика на входные данные.

Перейдем к более формальной постановке задачи генетического программирования, для простого примера, связанного с конструированием программы вычисления функции одного переменного. Пусть нам вместо черного ящика задана конечная таблица пар чисел (x_i, y_i), где x_i обозначает аргумент, а y_i значение функции от этого аргумента. Будем искать программу, представленную древовидным графом, которая будет вычислять по значению x_i значения z_i минимально отличающимися в совокупности от y_i , то есть будем искать программу, минимизирующую функцию

$$F(p) = \sum_{1}^{n} |y_i - z_i|.$$

Аргументом функции F является программа-геном , о которых речь шла выше. Определим размер популяции программ, равным, например, 10, и случайным образом построим 10 графов-программ с небольшим числом узлов и связей между ними.

Для каждого графа i вычислим значение функции $F(p_i)$ и упорядочим p_i по значениям функции $F(p_i)$.

Далее мы будем поступать так, как диктует классический генетический алгоритм для получения популяции программ на следующем шаге. Однако операцию скрещивания мы будем проводить своеобразным способом, а именно:

Выберем два графа из популяции, в каждом из них выделим какие то ветви дерева и поменяем их местами, например, так, как по-

казано на рисунке 4.

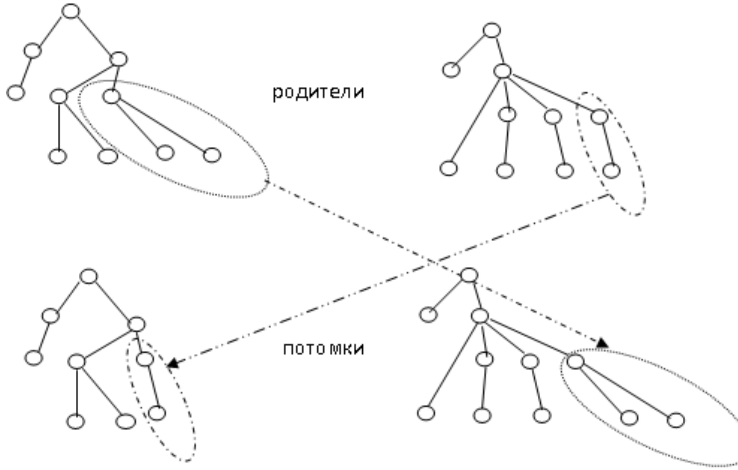

Рис. 4

Из двух родителей мы получаем двух потомков, которые могу оказаться в следующей популяции, если алгоритм селекции это выполнит.

Операция мутации графа состоит в том, что к дереву либо добавляется один узел, либо убирается какая-либо ветвь или узел. При выполнении операций мутации и скрещивания соблюдается правила сохранения правильного синтаксиса новых графов, которые составят новую популяцию.

В отличие от обычного генетического алгоритма геномы алгоритма могут быть разной длины с разным числом узлов и связей между ними. Это создает дополнительные сложности в организации алгоритма конструирования хороших программ.

Представление программ в виде деревьев и других графовых структур наиболее часто распространено в практике использования генетического программирования (ГП). Однако развитая теория ГП позволяет различным образом интерпретировать понятие программы. Вместо машинных программ, используя ГП, можно конструировать электрические сети с заданными характеристиками, функции алгебры логики, в принципе любые автоматические устройства, представляющие собой композицию элементарных функциональных компонент, связанных в единую систему.

На основе использования идей генетического программирования сконструированы и даже запатентованы [10], некоторые автоматические радиотехнические устройства!

Программу в ГП в общем случае следует понимать как план действий для достижения поставленной цели с наилучшими параметрами получающегося результата, т. е. план достижения экстремального значения некоторого функционала.

В формальной постановке программа представляет собой некоторую структуру, состоящую из узлов и связей между ними. Такая структура типа дерева семантически должна допускать следующую интерпретацию:

Каждый узел должен уметь выполнять некоторые действия; корневой узел доставляет решение поставленной задачи. Чтобы он мог выполнить назначенное ему действие должны быть выполнены задачи подмножеством узлов, с которыми он непосредственно связан. Это касается каждого узла, который может выполнить свою работу, если выполнили свою работы непосредственно связанные с ним узлы следующего по иерархии уровня. Листья дерева узлов представляют собой терминальные узлы, с которых начинается выполнение работ, и от которых уже нет связей с другими узлами более низкого уровня. В смысле машинных программ терминальные узлы представляют собой исходные данные программы, либо процедуры, не требующие входных параметров.

Основными элементами любой генетической программы являются терминалы и функции. При использовании ГП для решения оптимизационной задачи в какой-либо предметной области следует выбрать алфавит терминальных символов и множество используемых функций. В области конструирования машинных программ в качестве набора функций, как правило, выбирают стандартный набор арифметических операций, элементарных функций, операторов ветвлений и циклов. В качестве терминальных символов – идентификаторы начальных данных и результатов промежуточных вычислений. Последние иногда играют роль функций обращения к памяти по записи и чтению. Набор функций может включать и более сложные элементы, такие как используемые в данной предметной области готовые библиотечные программы.

Количество практически полезных результатов применения идей моделирования на компьютерах реальных процессов, с целью получения практически полезных результатов непрерывно растет.

Процессы происходящие в природе на разных уровнях, начиная с биомалекулярного и кончая мокропроцессами развития живой материи, дают богатую пищу для создания все новых и новых алгоритмов, подражающих природным процессам. Достаточно привести название некоторых алгоритмов: «Муравьиный алгоритм», воспро-

изводящий логику поведения муравьев по отысканию кратчайшего пути к пище, алгоритм «птичьей стаи», решающий некоторые навигационные задачи и т. пр.

Но в этой сфере исследований и приложений слишком много эвристик, адекватных здравому смыслу, но не получивших точных математических доказательств правильности полученных решений. Совпадение правдоподобия и правильности полученных решений обосновывается экспериментом на некоторых базах данных, принятых в качестве мировых стандартов, на которых проверяется эффективность и точность многочисленных эвристик по распознаванию образов, классификации, кластеризации и прогнозирования.

В нашей стране развитием идей нейровычислений, генетического программирования и эволюционных вычислений занимаются в ряде академических институтов и университетах. За последние 3-4 года на эту тему опубликовано более 50 научных статей в журналах и сборниках.

Тем не менее, приходится констатировать, что до «наведения математического порядка» в этих исследованиях еще очень далеко. Причина может быть в том, что все эвристики этого рода имеют дело с дискретными последовательностями и конечными множествами, для которых не существует таких важных понятий, как бесконечность непрерывность, предел, т. е. не существует тех «китов», на которых держится фундамент классической математики с ее грандиозными успехами в точном решении многих и многих задач.

Компьютерные вычисления по самой своей сути являются конечными и дискретными. По-видимому, настало время создания и развития классической компьютерной математики, учитывающей эти особенности вычислительной техники!

В этом смысле очень важными представляются исследования, выполняемые школой академика Журавлева Ю. И. по математическим проблемам прогнозирования и алгебраическому подходу к анализу этого круга задач [8], [15], [16].

4. Заключение

Алгоритмы, описанные выше, обладают важной особенностью — они хорошо распараллеливаются в силу своей конструкции. Однако это распараллеливание в большинстве случаев нейросетевых вычислений «мелкозернистое», т. е. сложность вычислений в каждом узле вычислительной сети очень мала по сравнению с возможностями процессорных узлов современных кластерных установок, реализованных на многоядерных микропроцессорах с гигафлопной про-

изводительностью. Во многих задачах, связанных с генетическими алгоритмами и генетическим программированием вычисление функции качества требует больших вычислительных ресурсов по памяти и быстродействию и распараллеливание в этом случае вполне оправдано и рационально.

Одной из проблем использования эволюционных алгоритмов, требующих предварительного обучения, состоит в том, что этот итерационный процесс плохо распараллеливается, т. к. он по своему существу является последовательным. Разработка параллельных алгоритмов обучения являет собой один из примеров, в которых используются эвристики, достоверность которых, как правило, не доказывается. Суть параллельных алгоритмов обучения сводится к тому, что обучающая выборка делится на подмножества и каждому узлу предлагается для независимого обучения свое подмножество общей выборки. Очевидно, что чем меньше выборка тем меньше точность настройки. В принципе каждый узел может обучить нейросеть посвоему. Проблема согласования результатов «независимых» обучений не имеет четко обоснованных решений.

Тем не менее, современные мультипроцессоры с массовым параллелизмом, занимающие ведущие позиции в TOP-500, предоставляют возможности решения практически важных задач на основе использования принципов обучения машин – ML.

Литература

1. K. Samuel. - Some studies in machine leaning using the game of checkers. Computers and Thought // Mc Craw – Hill, New York. 1963.

2. Mc Culloch V. S., Pitts W. H. A logical calculus of ideas immanent in nervous activity // Bull. Math. Biophysics, 1943. - Vol. - 2. – Pp548-558

3. Розенблатт Ф. Принципы нейродинамики. // «Наука», М., 1965.

4. Минский М. и Пейперт С. - Персептроны. // «Наука», М., 1971.

5. Holland J. Adaptation in natural and artificial systems. // MIT press, Cambridge MA, 1992

6. W. Bonzhaf, P. Nordin, R. E. Keller, F.D. Francone. Genetic Programming – An Introduction // Verlag fur digitale Technologie CmbH, Heidelberg, 1998.

7. С. Осовский. Нейронные сети для обработки информации. // М.: «Финансы и статистика», 2002.

8. Ю. И. Журавлев. Непараметрические задачи распознавания образов. // Ж. Кибернетика, N6, Москва, 1976.

9. Ту, Дж., Гонсалес, Р. – Принципы распознавания образов: Пер. с англ. // М.: Мир, 1978.

10. Srinivas M, Patnaik L. Genetic Algorithms, a survey. // J. Computer, v. 27, № 6, 28-43, IEEE press. June 1994.

11. Ribeiro J, and al. Genetic Algorithms. Programming Environments. // Computer, v. 27, № 6, 28-43, 17-26, IEEE press, June, 1994.

12. Forrest S. Genetic Algorithm: Principles of Natural Selection Applied to Computation. // J. Sciense. V. 261, 872-878, August 1993.

13. John Koza, Genetic Programming: On the Programming of Computers by Means of Natural Selection // MIT Press 1992.

14. John Koza, Genetic Programming II: Automatic Discovery of Reusable Programs // MIT Press,1994.

15. Чехович Ю.В. Применение алгебраического подхода к задачам выделения трендов. // Матем. методы распознавания образов (ММРО-10). Докл. 10-й Всероссийск. конф.ВЦ РАН. М.: АЛЕВ-В, 2001. С.315-316.

16. Рудаков К.В., Чехович Ю.В. О проблеме синтеза обучающих алгоритмов выделения трендов (алгебраический подход). // Прикладная математика и информатика.# 8. М.: Изд. отдел ф–та ВМиК МГУ, 2001. С. 97–113.

Аннотации

Kukushkin D.I., Antonenko V.A. Daddy: Data Dependancy Graphs in Serverless Computations // Программные системы и инструменты. Тематический сборник № 20, М.: Изд-во факультета ВМК МГУ, 2020.

This paper explores the possibility of using a data dependency graph to reduce the overhead of data exchanges when invoking composite serverless functions. This work describes a prototype allowing to move data exchanges between parts of a composite serverless function to the infrastructure of the service provider. Experimental evaluations show prototype's possibility to reduce the overhead of such data exchanges.

Ил.: 14 рис., Библиогр.: 10.

Zaitseva O.A., Antonenko V.A. CLEVER: Cold Start for Serverless Applications // Программные системы и инструменты. Тематический сборник № 20, М.: Изд-во факультета ВМК МГУ, 2020.

Serverless computing - the cloud paradigm for deploying applications and services represents an evolution in cloud application development, programming models, abstractions, and platforms. Cloud providers provide separate containers to perform on-demand functions. The container environment often must be reinitialized, which leads to reduced productivity and increased overhead. This is called a cold start. In this paper, we consider functions as content that needs to be cached to reduce execution delay. Given the dynamic nature of user behavior, the integration of popularity forecasting in caching is of paramount importance for the better use of serverless computing and user satisfaction.

Ил.: 10 рис., 1 табл. Библиогр.: 14.

Аникевич Ю.В., Бахмуров А.Г., Чайчиц Д.А. Разработка и реализация платформы для сбора и обработки физиологических данных о человеке // Программные системы и инструменты. Тематический сборник № 20, М.: Изд-во факультета ВМК МГУ, 2020.

В работе предлагается открытая платформа для разработки широкого класса описанных выше систем. Описывается архитектура системы, структура базы данных, необходимое прикладное ПО на мобильных телефонах.

Ил.: 2 рис., 1 табл., Библиогр.: 14.

Баула В.Г. Формальная семантика в обучении программированию // Программные системы и инструменты. Тематический сборник № 20, М.: Изд-во факультета ВМК МГУ, 2020.

Статья посвящена некоторым аспектам преподавания программирования. При обучении программированию зачастую не обращают внимания на область значений на которых программа выдаёт корректный ответ. В работе на нескольких довольно простых примерах показывается, что в зависимости от предусловий задача может реализовываться по разному. В статье показывается, что важно правильно специфицировать задачу. Для спецификация программы используются элементы аксиоматической семантики Т. Хоара.

Библиогр.: 7.

Ержанов Ж.К., Смелянский Р.Л. Сетевое кодирование на основе $GF(3^m)$ // Программные системы и инструменты. Тематический сборник № 20, М.: Изд-во факультета ВМК МГУ, 2020.

В данной статье формулируется новый подход к сетевому кодированию, основанный на сжатии данных без потерь, путем перехода в другой алфавит; в ней перечисляются преимущества и недостатки линейного сетевого кодирования, а также приводится свойство экономичности троичной системы счисления. На основе приведенных рассуждений предлагается новый подход к организации систем передач данных, с использованием сетевого кодирования. В конце статьи приводятся открытые проблемы, связанные с предлагаемым подходом.

Ил.: 1 рис., Библиогр.: 8.

Иванов И.В., Антоненко В.А. Разработка архитектуры эластичного приложения в среде легковесной виртуализации // Программные системы и инструменты. Тематический сборник № 20, М.: Изд-во факультета ВМК МГУ, 2020.

В данной работе проведен обзор существующих решений, поддерживающих эластичное состояние инфраструктуры, представлена архитектура эластичного приложения, удовлетворяющая сформулированным критериям эластичности. В основе ее реализации лежат механизмы распределенного хранения данных, что позволяет преоб-

разовать состояние приложения из stateful в stateless и последующе проводить горизонтальное масштабирование всех его частей.

Ил.: 2 рис., 2 табл. Библиогр.: 25.

Кузьмин Я.К., Волканов Д.Ю., Скобцова Ю.А. Исследование применимости алгоритмов обработки пакетов с сохранением состояния в архитектуре сетевого процессорного устройства RuNPU // Программные системы и инструменты. Тематический сборник № 20, М.: Изд-во факультета ВМК МГУ, 2020.

В данной работе предлагаются модификации архитектуры сетевого процессорного устройства, позволяющие применять алгоритмы обработки пакетов с хранением состояния. Проводится экспериментальное исследование модифицированной архитектуры, направленное на измерение характеристик данной архитектуры. В результате экспериментального исследования показана применимость данного подхода, оценены энергопотребление и пропускная способность СПУ.

Ил.: 4 рис., 1 табл., Библиогр.: 11.

Машонский И.Д., Большакова Е.И. Инструментальные средства извлечения терминов из текстов: разработка компонентов для русского языка // Программные системы и инструменты. Тематический сборник № 20, М.: Изд-во факультета ВМК МГУ, 2020.

В статье дается краткий обзор методов, применяемых для автоматического извлечения терминов из текстов на естественном языке, а также программных инструментов, в которых реализованы эти методы. На основе проведенного анализа возможностей этих инструментов предлагается решение задачи автоматического извлечения терминов из текстов на русском языке – в виде дополнительных программных компонентов, реализованных на базе библиотеки atr4, описываются ключевые особенности разработанных компонентов.

Ил.: 1 рис., Библиогр.: 17.

Миков А.И., Миков А.А. Устойчивость к возмущениям масштабируемых мобильныхинформационно-ориентированных сетей // Программные системы и инструменты. Тематический сборник № 20, М.: Изд-во факультета ВМК МГУ, 2020.

В статье представлена модель мобильной информационно- ориентированной сети для патрулирования области и алгоритм ав- томатического управления сетью, обеспечивающий ее работоспособность при наличии возмущений движения. Геометрическая структура сети основана на теории экстремальной упаковки, которая обеспечивает использование минимального количества узлов, расположенных на беспилотных летательных аппаратах. Этот подход также

характеризуется масштабируемостью создаваемых мобильных сетей. Алгоритм управления реализует прогнозную математическую модель, основанную на вычислении характеристик динамических геометрических графов. Основная задача алгоритма управления - поддерживать связность сети максимально возможное время при наличии внешних воздействий, разрушающих структуру. В качестве индикаторов для прогнозирования потери связности динамического графа сети используются степени вершин графа. Сравнение проводится с ранее разработанным авторами алгоритмом управления по другому показателю - появлению мостов в графе. Алгоритм сочетает в себе глобальную (по всей сети) оценку топологии с локальными управляющими действиями по коррекции положения (по отношению к одному или нескольким соседним узлам одновременно). С помощью методов моделирования проведена сравнительная оценка эффективности предложенного алгоритма управления. Показано, что новый алгоритм поддерживает связность сети значительно дольше при умеренных возмущениях.

Ил.: 3 рис., Библиогр.: 8.

Никифоров Н.И., Волканов Д.Ю., Скобцова Ю.А. Анализ и исследование структур данных для поиска в таблицах классификации в сетевом процессорном устройстве с архитектурой RuNPU // Программные системы и инструменты. Тематический сборник № 20, М.: Изд-во факультета ВМК МГУ, 2020.

В данной работе рассматривается проблема классификации пакетов в рамках архитектуры сетевого процессора, без выделенного ассоциативного устройства TCAM памяти. Для этапа классификации требуется реализация структур данных для хранения таблиц классификации. В статье рассмотрены структуры данных с учётом ограничений рассматриваемой архитектуры. Экспериментальное исследование, проведённое на имитационной модели сетевого процессора, показало, что использование адаптированного АВЛ дерева позволяет сократить количество тактов сетевого процессора на обработку одного пакета и уменьшить использование памяти вычислительных блоков конвейера.

Ил.: 7 рис., 1 таб., Библиогр.: 12.

Шапошников В.А., Писковский В.О. Постановка задачи и краткий обзор решений для проблемы управления процессами согласованной реконфигурации ПКС // Программные системы и инструменты. Тематический сборник № 20, М.: Изд-во факультета ВМК МГУ, 2020.

Логикой работы программно-конфигурируемой сети управляют запущенные на контроллере приложения. Результат работы приложений зависит от окружения, наличия других приложений, также выполняемых на контроллере и влияющих на работу сети. Таким образом, возникает проблема корректного управления работой сети при поступлении конкурирующих запросов на её реконфигурацию. В статье представлено описание постановки задачи и краткий обзор существующих решений проблемы. В результате анализа предложен подход к решению на основе теорииупорядочивания (сериализации), предложены определения согласованности и синхронизации процедур реконфигурации.

Ил.: Библиогр.: 7.

Шмитов Н.О., Пашков В.Н. Устойчивость к возмущениям масштабируемых мобильныхинформационно-ориентированных сетей // Программные системы и инструменты. Тематический сборник № 20, М.: Изд-во факультета ВМК МГУ, 2020.

С развитием Интернета вещей становятся и более актуальными его проблемы. Количество устройств растет, как и разнообразие возможных атак на них, а управление и контроль становятся нетривиальной задачей для распределенной сети, поэтому в этой работе предлагается использовать программно-конфигурируемые сети (ПКС) вместо традиционных. Переход к ПКС позволяет рассмотреть совершенно иные методы управления устройствами интернета вещей и управления обменом данными между ними. В данной статье приведен сравнительный анализ существующих методов управления устройствами интернета вещей в ПКС, предложен и частично реализован, собственный метод, позволяющий классифицировать устройства интернета вещей для последующей задачи распознавания нехарактерного поведения для каждого класса устройств, а также реализован собственный экспериментальный стенд, позволяющий проводить эксперименты с ПКС контроллером.

Ил.: 1 рис., 3 табл., Библиогр.: 9.

Королев Л.Н. Об эволюционных алгоритмах, нейросетевых вычислениях, генетическом программировании – математические проблемы // Программные системы и инструменты. Тематический сборник № 20, М.: Изд-во факультета ВМК МГУ, 2020.

В статье в общих чертах рассматриваются некоторые проблемы, связанные с применением эволюционных и генетических алгоритмов, генетического программирования, нейросетевых вычислений, для решения прикладных задач, сводящихся к анализу функций, заданных на перестановках. Отмечается их естественный параллелизм

и возможность реализации этих методов на современных вычислительных машинах.

Ил.: 4 рис., Библиогр.: 16.

О третьей международной научной конференции "Современные сетевые технологии"(Monetec-2020)

Третья международная научно-техническая конференция «Современные сетевые технологии» (Monetec-2020) прошла в онлайн-формате 27 - 29 октября 2020 года и собрала представителей международного научного сообщества, исследовательских подразделений корпораций, стартапов, промышленности и бизнеса, институтов развития и органов государственной власти для обсуждения перспективных и актуальных технологий в сфере компьютерных сетей, виртуализации сетевых ресурсов и облачных вычислений, использования методов искусственного интеллекта.

Сети передачи данных являются основой современной цивилизации. Области телекоммуникации вбирают в себя и постоянно порождают все новые и новые технологии, которые открывают новые возможности, повышают качество сервиса, безопасности в современных сетях. Технологии программного управления в сетях, виртуализации сервисов, периферийные облачные вычисления, стали ключевыми элементами построения современных сетей передачи данных и информационных инфраструктур в целом. В настоящее время в мире (и в России, в частности) начато их применение на практике. Однако творческая мысль не останавливается на достигнутом.

Сегодня мы уже говорим о Реконфигурируемых по Требованию Сетях (Intent Based Network), Информационно- Ориентированных Сетях (Information Centric Network). Контент-Ориентированных Сетях (Content Centric Network). Возникает много новых проблем и направлений для исследований.

На конференции выступили с пленарными докладами ряд зарубежных и отечественных ученых. Тематикой их выступлений были перспективные направления развития современных сетей передачи данных и их приложений. Также сосотоялись секционные доклады, стендовая сессия для студентов, индустриальная секция для представителей промышленности. В рамках конференции прошло несколько школ по сетевым технологиям и применению отечественных решений.

Тематика конференции:

1. Архитектура и протоколы для автоматизации управления сетями и оркестрации в облачных инфраструктурах.

2. Инновационные технологии для облаков, интернета, компьютерных сетей.

3. Новые парадигмы в организации и функционирования сетей, например, Intent Based Network, Information Centric Network, Content Centric Network.

4. Подходы, методы и средства для управления качеством сервиса и распределения ресурсов в сетях и облачных средах.

5. Применение больших данных и машинного обучения для повышения эффективности функционирования сетей, облачных платформ и управления ими.

6. Проблемы безопасности сетей и облачных платформ.

7. Применение сетевых облачных технологий к обработке больших данных, Интернета вещей, сетей беспроводной и мобильной связи.

8. Вопросы архитектуры программно-управляемого телекоммуникационного оборудования.

Приглашаем Вас к участию в конференции Monetec-2022.

Школа по сетевым и облачным технологиям Monetec-2020

22-26 октября 2020 года в рамках третьей международной конференции «Современные сетевые технологии» (Monetec-2020) под руководством Пашкова В.Н. силами преподавателей ВМК МГУ имени М.В. Ломоносова, СПбГУТимени проф. Бонч-Бруевича и МТУСИ была проведена Школа по сетевым и облачным технологиям из 6 научно-практических семинаров. Основной целевой аудиторией семинаров Школы являлись бакалавры 3-4 курсов, магистры, аспиранты, преподаватели, научные сотрудники, программисты, системные администраторы, ИТ специалисты и ИТ менеджеры. В рамках семинаров проводилисьпрактические мастер-классы по быстрому погружению и освоению технологий программно-конфигурируемых сетей, технологийблокчейн и смарт-контрактов,технологий виртуализации сетевых функций,мастер-классы по основам работы с docker-контейнерами и с облачной платформой OpenStack. В Школе Monetec-2020 прошли обучение более 50 молодых специалистов. По окончании Школы все участники получили сертификаты. Ниже приведены аннотации прошедших семинаров Школы Monetec-2020. Подробные программы семинаров доступны на странице Школы по сетевым и облачным технологиям: http://monetec.ru/seminars.

1. Практический семинар по SDN технологиям

В рамках семинара освещаются основы концепции программно-конфигурируемых сетей (ПКС, SDN), архитектура SDN сетей и их основные компоненты (протокол OpenFlow, SDN/OpenFlow коммутаторы, SDN контроллер), преимущества использования в сравнении с традиционными сетями. В практической части семинара проводится мастер класс по разработке приложения для SDN контроллера

RUNOS.

Преподаватель: Е.П. Степанов, МГУ имени М.В. Ломоносова

2. Практический семинар по NFV технологиям

В рамках семинара освещаются основы концепции виртуализации сетевых функций (NFV), архитектура и особенности облачных платформ, облачных сервисов. В практической части семинара проводится мастер класс по разработке TOSCA-шаблонов для облачных сервисов под управлением MANO-платформы C2.

Преподаватель: В.А. Антоненко, МГУ имени М.В. Ломоносова

3. Практический семинар по проведению лабораторных работ и практикумов в распределенной облачной платформе на базе OpenStack

В рамках семинара освещаются основы архитектуры облачной платформы OpenStack, основные компоненты и их назначение, аппаратные и программные требования для создания и развертывания распределенной лаборатории на OpenStack, функциональные возможности лаборатории, освещаются процедуры создания новой лабораторной работы и процедуры их проведения. В практической части семинара проводится мастер класс по разработке лабораторной работы, образов виртуальных машин и проведению занятий в распределенной лаборатории.

Преподаватель: В.Н. Пашков, МГУ имени М.В. Ломоносова

4. Практический семинар по применению технологий распределенного реестра и разработке смарт контрактов

В рамках семинара освещаются основы разработки с использованием технологии блокчейн, их преимущества и недостатки, архитектура приложений, построенных на базе или с применением технологии блокчейн. В практической части семинара проводится мастер класс по разработке простейших смарт-контрактов на программном языке Solidity.

Преподаватели: К.Э. Есалов, СПбГУТ им. М.А.Бонч-Бруевича, А.В. Помогалова, СПбГУТ им. М.А.Бонч-Бруевича

5. Практический семинар по применению технологий контейнеризации docker при разработке изолированных приложений

В рамках семинара освещаются основы концепции контейнеризации, их преимущества и недостатки, архитектура взаимодействия

нескольких docker контейнеров. В практической части семинара проводится мастер класс по разработке изолированного приложения в рамках docker контейнера.

Преподаватели: К.Э. Есалов, СПбГУТ им. М.А.Бонч-Бруевича, А.А. Швидкий, СПбГУТ им. М.А.Бонч-Бруевича

6. Практический семинар по анализу производительности мультисервисных узлов доступа

В рамках семинара освещаются основные понятия, относящиеся к организации процесса совместного обслуживания мультимедийного трафика в действующих и перспективных сетях связи. Излагаются особенности моделирования и анализа процедур распределения ресурса передачи информации, в которых учитывается зависимость потоков заявок от загрузки сети, наличие ограниченного доступа, резервирования, динамического распределения ресурса и т.д. Для всех перечисленных моделей построены эффективные алгоритмы оценки основных показателей качества обслуживания поступающих заявок. В практической части семинара проводится мастер-класс по использованию моделей для организации эффективного использования ресурса при совместном обслуживании беспроводными узлами доступа мультимедийного трафика сервисов реального времени и трафика IoT.

Преподаватель: С.Н. Степанов, МТУСИ

Software systems and tools: Tematic collection / Ed. by R. L. Smelyansky.— Moscow: Publishing Department of the Faculty of Computational Mathematics and Cybernetics (license ID № 05899 from 24.09.2001); MAKS Press, 2020.— № 20.— 188 p.

ISBN 978-5-89407-618-8 (CMC MSU)

ISBN 978-5-317-06545-4 (MAKS Press)

Theese proceedings consists of student's works, who have received the recommendation of the departments Automation of Computer Systems and Algorithmic Languages, as well as poster presentations of the international conference «Modern network technologies-2020». This edition continues the tradition of publishing in memory of L. N. Korolev. The proceedings contains articles devoted to creating information computing infrastructure for scientific research, problems of modern computer networks, methods and tools for organizing and managing cloud computing, tools for processing text in Russian and methodological problems of teaching programming.

These papers will be of interest to students, graduate students and professionals in the development of application software systems.

Keywords: information telecommunication technology, software-defined networking, OpenFlow switch, centralized controller, cloud computing, virtual resources, network functions virtualization, cloud platform, network processor unit, cold start, popularity prediction, caching, serverless applications, medical environment, formal semantics, network coding, Galois fields, trit, elastic computing, lightweight virtualization, word processing, information-oriented networks, reconfiguration, Internet of Things, evolutionary algorithms, neural network computing, genetic programming.

Научное издание

ПРОГРАММНЫЕ СИСТЕМЫ И ИНСТРУМЕНТЫ

Тематический сборник

№ 20

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

Издательский отдел
Факультета вычислительной математики и кибернетики МГУ
имени М.В. Ломоносова
Лицензия ИД N 05899 от 24.09.01 г.

119992, ГСП-2, Москва, Ленинские горы,
МГУ имени М.В. Ломоносова,
2-й учебный корпус

Издательство «МАКС Пресс»
Главный редактор: *Е. М. Бугачева*

Напечатано с готового оригинал-макета
Подписано в печать 30.12.2020 г.
Формат 60x90 1/16. Усл.печ.л. 11,75.
Тираж 50 экз. Заказ 215.

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»
115201, г. Москва, ул. Котляковская, д. 3, стр. 13.