DOI 10.37882/2223-2966.2023.07.10

# ЭРГОНОМИЧНОСТЬ ВЗАИМОДЕЙСТВИЯ КОМПОНЕНТОВ КЛИЕНТ-СЕРВЕРНОЙ АРХИТЕКТУРЫ ИНФОРМАЦИОННОЙ СИСТЕМЫ

## ERGONOMICS OF THE INTERACTION BETWEEN THE COMPONENTS OF THE CLIENT-SERVER ARCHITECTURE OF THE INFORMATION SYSTEM

***B. Goryachkin***
***R. Baibarin***

*Summary*: The article is devoted to the analysis of the efficiency of interaction between the components of the client-server architecture. The objects of research are the formats of the textual representation of transmitted messages. Their characteristics of transformation from a language structure object to a string and vice versa are analyzed. Four formats of text representation are parsed: JSON, XML, YAML, TOML. The research was conducted for two languages: Python and JavaScript. Exponentially weighted estimation (EVO) was used as a cleanup method. An integral criterion for evaluating different formats of textual representation was developed to reduce a multi-criteria problem to a single-criteria one. As a result of the research, the advantages of the JSON format in the speed of serialization and deserialization are shown.

*Keywords*. Serialization, deserialization, JSON, XML, YAML, TOML, EVO, client-server architecture, message enlargement ratio, ergonomics.

***Горячкин Борис Сергеевич***
*кандидат технических наук, доцент;*
*Московский государственный технический университет им. Н.Э. Баумана*
*bsgor@mail.ru*
***Байбарин Роман Григорьевич***
*Московский государственный технический университет им. Н.Э. Баумана*
*rbaybarin@icloud.com*

*Аннотация*. Статья посвящена анализу эффективности взаимодействия компонентов клиент-серверной архитектуры. Объектами исследования являются форматы текстового представления передаваемых сообщений. Анализируются их характеристики преобразования из объекта структуры языка в строку и наоборот. Анализируются четыре формата текстового представления: JSON, XML, YAML, TOML. Исследование проводилось для двух языков: Python и JavaScript. В качестве метода очистки от выбросов использовалось экспоненциально взвешенное оценивание (ЭВО). Был разработан интегральный критерий оценивания разных форматов текстового представления для сведения многокритериальной задачи к однокритериальной. В результате проведенных исследований показаны преимущества формата JSON в скорости сериализации и десериализации.

*Ключевые слова*: сериализация, десериализация, JSON, XML, YAML, TOML, ЭВО, клиент-серверная архитектура, коэффициент увеличения сообщения, эргономичность.

## Введение

Современные тенденции развития информационных систем таковы, что все больше и больше разработчиков и поставщиков программного обеспечения (ПО) внедряют процессы серверной обработки данных для своих систем. Причин такого вектора развития много: обеспечение безопасности бизнес-процессов, сбор данных для Big Data аналитики [1], обеспечение взаимодействия пользователей в системах типа «чат» [2], увеличение популярности облачных решений и многое другое. При этом проблемы доступности серверного ПО и эффективности взаимодействия компонентов являются основополагающими в вопросах удобства работы с приложением. При этом под эффективностью взаимодействия понимается метрика, характеризующая время получения ответа клиентом на выполненный им запрос.

Задача повышения эффективности взаимодействия решается на разных этапах проектирования, разработки и эксплуатации системы. Примерами методов решения может быть разработка своего сетевого протокола [3] для повышения скорости общения, выполнение операции анализа кода программного обеспечения, профилирования, выявления узких мест [4], интеграция горизонтального масштабирования, сервисов балансировки [5] и менеджеров очередей [6] и другие решения.

## Постановка задачи

В данной работе рассматривается и оценивается эффективность взаимодействия на уровне тела передаваемых данных в ходе сетевого взаимодействия. Предметом анализа будет строковое представление передаваемых данных, а также программные средства, позволяющие преобразовать произвольно структурированную информацию в строку (сериализаторы) и обратно (десериализаторы). При этом в качестве формата представления данных используются общепринятые форматы (JSON, XML, YAML, TOML), а программное обеспечение для сериализации и десериализации — наиболее популярные библиотеки для каждого из языков.

Стоит отметить, что, ввиду рассмотрения общей задачи, в качестве передаваемых данных рассмотрены объекты произвольной структуры. Это означает, что система заранее не имеет схему данных (к примеру: вида XSD схемы для XML документов [7]). Данное условие особен-

но критично для десериализации, так как система определяет структуру получаемых данных прямо по ходу выполнения операции.

В рамках анализа предполагается оценить следующие величины:

1. Время, затраченное на преобразование структуры языка в строку (время сериализации). [с/байт]
2. Время, затраченное на преобразование строки в структуру языка (время десериализации). [с/байт]
3. Коэффициент увеличения сообщения. Является отношением суммарного количества информации, затрачиваемой на хранение структуры языка в памяти, и суммарной длины информации, затрачиваемой на хранение сериализованной строки в памяти. [безразмерная]

Ввиду того, что операции сериализации и десериализации выполняются методами конкретных библиотек языка, скорости выполнения данных операций напрямую зависят от внутренней его архитектуры. С целью минимизировать влияние характеристик конкретного языка на анализ, были рассмотрены несколько языков программирования, в частности Python и JavaScript.

## Описание форматов представления

Рассматриваемые форматы являются наиболее популярными и широко используемыми в разных сферах IT, таких как сетевое взаимодействие, конфигурационные файлы, CI/CD, описание развертывания систем, интерфейсов и многое другое. Ниже приведено описание форматов текстового представления информации.

### JSON

JSON (JavaScript Object Notation) — формат, реализующий неструктурированное текстовое представление структурированных данных, основанное на принципе пар ключ-значение и упорядоченных списках. Хотя JSON начал свое распространение с JavaScript, он поддерживается в большинстве языков, либо изначально, либо с помощью специальных библиотек. Обычно Json используется для обмена информацией между веб-клиентами и веб-сервером. Формат JSON был разработан Дугласом Крокфордом в 1999 году [8].

### XML

XML (eXtensible Markup Language) — это расширяемый язык разметки. Рекомендован Консорциумом Всемирной паутины (W3C) в 1998 году [9]. Спецификация XML описывает XML-документы и частично описывает поведение XML-процессоров (программ, читающих XML-документы и обеспечивающих доступ к их содержимому). XML разрабатывался как язык с простым формальным синтаксисом, удобный для создания и обработки документов как программами, так и человеком, с акцентом на использование в Интернете. Язык называется расширяемым, поскольку он не фиксирует разметку, используемую в документах: разработчик волен создать разметку в соответствии с потребностями к конкретной области, будучи ограниченным лишь синтаксическими правилами языка.

### YAML

YAML («Yet Another Markup Language») — «дружественный» формат сериализации данных, концептуально близкий к языкам разметки, но ориентированный на удобство ввода-вывода типичных структур данных многих языков программирования [10]. Первая версия формата опубликована в 2001 году.

Язык похож на XML и JSON, но использует более минималистичный синтаксис при сохранении аналогичных возможностей. YAML обычно применяют для создания конфигурационных файлов в программах типа «Инфраструктура как код» (Iac), или для управления контейнерами в работе DevOps.

Таблица 1.

Примеры форматов данных

| | |
|---|---|
| Исходная структура | Пользователь<br>Поля:<br>Фамилия: Петров<br>Имя: Иван<br>Отчество: Иванович<br>Возраст: 25 |
| JSON | {“Фамилия”:“Петров”,“Имя”:“Иван”,“Отчество”:“Иванович”,”<br>Возраст”:25} |
| YAML | Фамилия: Петров<br>Имя: Иван<br>Отчество: Иванович<br>Возраст: 25 |
| XML | <root><Фамилия>Петров</Фамилия><Имя>Иван</<br>Имя><Отчество>Иванович</Отчество><Возраст>25</<br>Возраст></root> |
| TOML | [root]<br>Фамилия = “Петров”<br>Имя = “Иван”<br>Отчество = “Иванович”<br>Возраст = +25 |

### TOML

TOML — формат конфигурационных файлов, спроектированный для обеспечения человекочитаемости, с одной стороны, и однозначного преобразования в ассоциативный массив, с другой. Спецификация языка открыта и дополняется сообществом. Название «TOML» является акронимом «Tom’s Obvious, Minimal Language» (Очевидный язык Тома) [11], имея в виду своего создателя, Tom Preston-Werner.

Для наглядности в качестве примеров рассмотрим простую структуру (пользователь) и ее представление в разных форматах (табл. 1).

## Критические моменты преобразования форматов

Ввиду произвольности структуры сериализуемой информации существуют ряд критических моментов, которые влияют на метрики. Таковыми являются:

1. *Вложенность структуры*. Рассматриваемые форматы по-разному оформляют в текстовом представлении вложенность одной структуры в другую. При этом для описания вложенности разные форматы используют разное количество дополнительных символов. Стоит также отметить, что некоторые форматы добавляют дополнительные символы для каждого поля вложенной структуры (например: YAML).
2. *Длина ключей и значений*. Для примера рассмотрим структуру, состоящую из одного строкового поля, длина которого стремится к бесконечности.

Таблица 2.

Статистические параметры операции сериализации

| ЯП | Тип формата | Среднее (сек) | СКО (сек) | Асимметрия | Эксцесс | Медиана (сек) |
|---|---|---|---|---|---|---|
| python | json | $4{,}46*10^{-8}$ | $2{,}48*10^{-8}$ | 0,2662761 | 0 | $4{,}36*10^{-8}$ |
| python | yaml | $5{,}27*10^{-6}$ | $2{,}28*10^{-8}$ | –1,1427154 | –0,3175087 | $5{,}81*10^{-6}$ |
| python | toml | $3{,}54*10^{-7}$ | $1{,}82*10^{-7}$ | –1,0265058 | 0 | $3{,}67*10^{-7}$ |
| python | xml | $7{,}48*10^{-6}$ | $3{,}67*10^{-6}$ | –1,1754568 | –0,4155718 | $8{,}51*10^{-6}$ |
| javascript | json | $1{,}77*10^{-8}$ | $1{,}09*10^{-8}$ | 1,3019915 | 0 | $1{,}58*10^{-8}$ |
| javascript | yaml | $1{,}35*10^{-7}$ | $6{,}03*10^{-8}$ | –0,9120042 | 0 | $1{,}38*10^{-7}$ |
| javascript | toml | $1{,}54*10^{-7}$ | $8{,}37*10^{-8}$ | –0,2923781 | 0 | $1{,}56*10^{-7}$ |
| javascript | xml | $2{,}41*10^{-7}$ | $1{,}34*10^{-7}$ | 1,9222012 | 0 | $2{,}05*10^{-7}$ |

Таблица 3.

Статистические параметры операции десериализации

| ЯП | Тип формата | Среднее (сек) | СКО (сек) | Асимметрия | Эксцесс | Медиана (сек) |
|---|---|---|---|---|---|---|
| python | json | $3{,}40*10^{-8}$ | $1{,}88*10^{-8}$ | –0,04564203 | 0 | $3{,}25*10^{-8}$ |
| python | yaml | $1{,}07*10^{-5}$ | $5{,}29*10^{-6}$ | –0,98539081 | –0,086417 | $1{,}16*10^{-5}$ |
| python | toml | $1{,}78*10^{-6}$ | $1{,}13*10^{-6}$ | 1,37085632 | 1,294440 | $1{,}56*10^{-6}$ |
| python | xml | $6{,}24*10^{-7}$ | $3{,}12*10^{-7}$ | –0,84338833 | 0 | 6,71* |
| javascript | json | $2{,}00*10^{-8}$ | $1{,}20*10^{-8}$ | 0,45666336 | 0 | $1{,}87*10^{-8}$ |
| javascript | yaml | $1{,}27*10^{-7}$ | $6{,}29*10^{-8}$ | –0,70030589 | 0 | $1{,}42*10^{-7}$ |
| javascript | toml | $3{,}01*10^{-7}$ | $1{,}93*10^{-7}$ | 0,26328552 | 0 | $2{,}87*10^{-7}$ |
| javascript | xml | $3{,}46*10^{-7}$ | $1{,}73*10^{-7}$ | –0,50608841 | 0 | $3{,}64*10^{-7}$ |

Таблица 4.

Статистические параметры операции коэффициента увеличения

| ЯП | Тип формата | Среднее (сек) | СКО (сек) | Асимметрия | Эксцесс | Медиана (сек) |
|---|---|---|---|---|---|---|
| python | json | 1,883 | 0,491 | 0,113 | 0,431 | 1,898 |
| python | yaml | 2,518 | 1,761 | 3,220 | 1,951 | 1,849 |
| python | toml | 3,327 | 2,928 | 3,365 | 2,021 | 2,132 |
| python | xml | 2,248 | 0,680 | 0,331 | 0,467 | 2,305 |
| javascript | json | 1,676 | 0,406 | 1,222 | 0,862 | 1,657 |
| javascript | yaml | 2,714 | 1,992 | 2,983 | 1,927 | 1,926 |
| javascript | toml | 4,502 | 4,881 | 3,674 | 2,101 | 2,389 |
| javascript | xml | 2,264 | 0,704 | 0,822 | 0,622 | 2,310 |

В таком случае коэффициент увеличения будет стремиться к 1 ввиду того, что накладные расходы на описание структуры полей перестанут играть важную роль в строковом представлении.

3. *Поля структуры типа «массив»*. Рассматриваемые форматы по-разному оформляют подобные конструкции в текстовом представлении. Это может создать больше накладных расходов для конкретного формата.
4. *Поля структуры типа «число с плавающей точкой»*. Подобные данные при переводе в строку могут иметь непредсказуемые последствия с точки зрения занимаемой памяти. К примеру: имеем число 1.1 (в памяти 4 байта, как переменная типа double). При переводе в строку имеем "1.1" (3 символа), что (ввиду известного набора используемых символов) соответствует 3 байтам в кодировке UTF-8 (необходимая память уменьшилась). Рассмотрим другой пример: имеем число 1.2345678912345 (в памяти 4 байта, как переменная типа double). При переводе в строку имеем "1.2345678912345" (15 символов), что соответствует 15 байт в кодировке UTF-8 (необходимая для хранения память увеличилась более, чем в 3 раза).

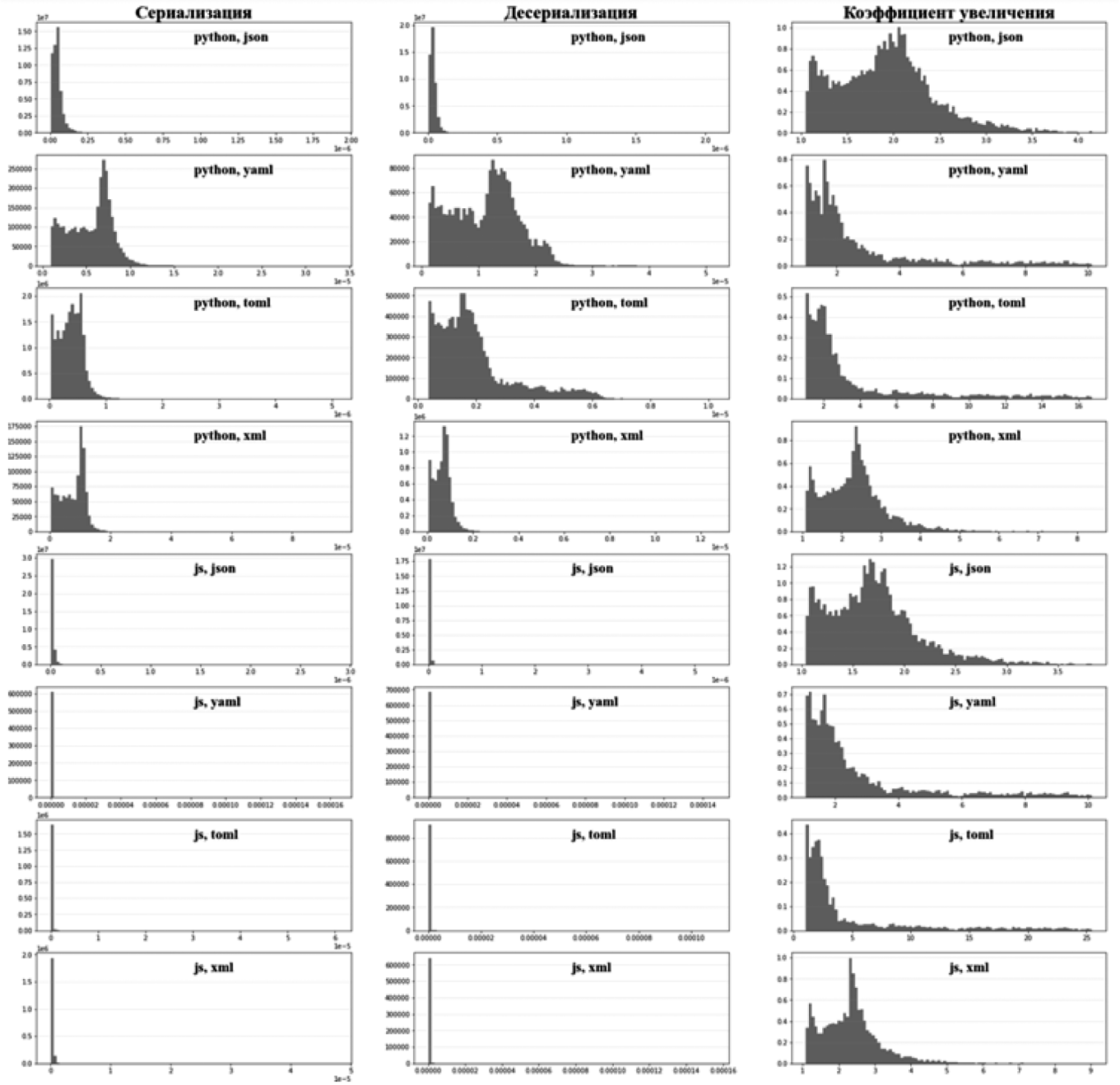


Рис. 1. Распределения сериализации, десериализации и коэффициента увеличения в зависимости от языка

### Анализ эргономичности взаимодействия компонентов клиент-серверной архитектуры

Эргономичность в упомянутом контексте не является синонимом эффективности взаимодействия, представляется более обобщенным критерием и позволяет сделать обоснованный вывод о приоритетном выборе форматов. В качестве начального набора структур был сгенерирован набор из 8400 структур. Каждая структура была объектом эксперимента 10 раз, в результате был получен набор замеров для анализа. Данная операция была проведена для двух языков программирования: Python и JavaScript. В табл. 2–4 приведены первичные статистические характеристики сериализации.

Языки программирования Python и JavaScript были выбраны ввиду того, что оба языка являются интерпретируемыми и поддерживают динамическую типизацию. Данный факт значительно упрощает процедуру десериализации.

Следует заметить, что значения медиан сильно отличаются от значений оценки математического ожидания (МО) (среднего значения). Это может свидетельствовать

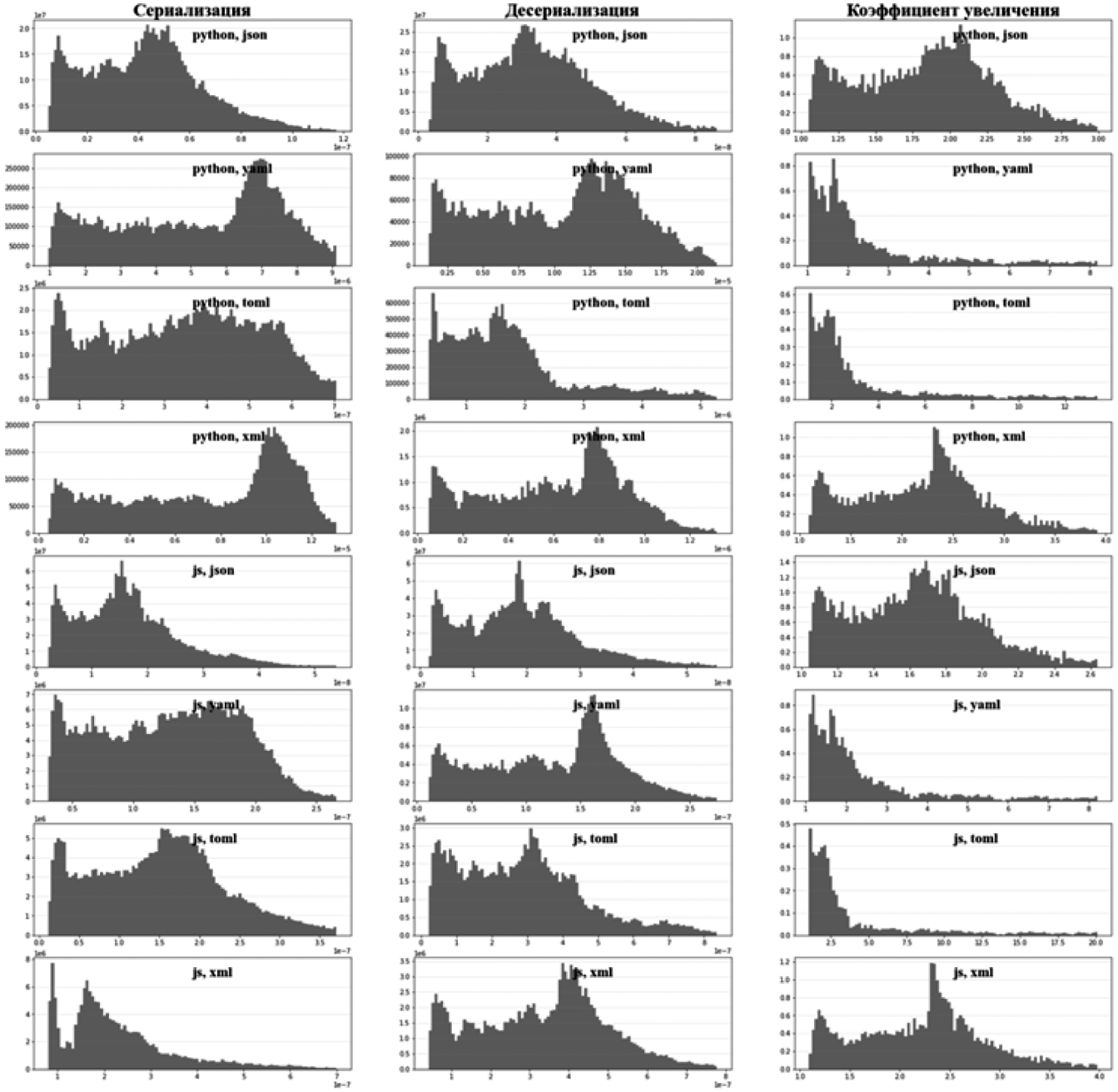


Рис. 2. Распределения сериализации, десериализации и коэффициента увеличения в зависимости от языка после эмпирической очистки

о смещении оценки МО (наличии в данных выбросов). На рисунке 1 видна общая тенденция данных.

Из рис. 1 видно, что у метрик наблюдается большой «хвост». От данных выбросов следует избавиться для продолжения анализа. Для этого проведем эмпирическую очистку путем отсечения данных после 96 перцентиля (данный перцентиль выбран эмпирически) по каждой метрике. На рис. 2 получим соответствующий результат.

Данная операция позволяет наблюдать основные формы распределений. Оценим результат очистки путем сравнения характеристик на примере времени сериализации языка JavaScript, формата JSON. В табл. 5 приведены статистические характеристики предварительно очищенных данных.

Заметим уменьшение наличия в данных выбросов (сближение медианы и среднего), уменьшение разброса данных (уменьшение СКО), а также выпрямление Гауссовой кривой (уменьшение асимметрии). Тем не менее, данная операция привела лишь к незначительному улучшению, ввиду своей простоты. Для более точной очистки набора от выбросов, а также исключения человеческого фактора, был использован алгоритм экспоненциально взвешенного оценивания (далее ЭВО). Данный метод основывается на взвешивании Хьюбера экспоненциальных измерений, что позволяет создать механизм исключения аномалий из анализа. Результаты анализа приведены в табл. 6.

Полученные в результате ЭВО оценки можем считать устойчивыми к выбросам.

Для комплексной оценки результатов ЭВО был разработан интегральный критерий — время, за которое по каналу связи с пропускной способностью $V_{ks}$ байт/с будет передан 1 байт полезной информации.

Пусть $T_s$ — случайная величина, характеризующая скорость сериализации группы замеров (определенного языка и формата сообщений), $T_d$ — случайная величина, характеризующая скорость десериализации, $K$ — случайная величина, характеризующая увеличение сообщения группы замеров. В таком случае объем информации, передаваемый по каналу связи равен $a \times K$, где $a$ — объем передаваемой полезной информации. В таком случае имеем формулу оценки интегрального критерия:

$$T = a * \left( T_s + \frac{K}{V_{ks}} + T_d * K \right), \quad (1)$$

где: $V_{ks}$ — скорость передачи данных по каналу связи (КС) (байт/с),

$a$ — количество передаваемой полезной информации (байт),

$T_s$ — время, затраченное на сереализацию 1 байта информации (с/байт),

$T_d$ — время, затраченное на десереализацию 1 байта информации (с/байт),

$K$ — увеличение сообщения.

Таблица 5.

Сравнение статистических показателей до и после очистки

| Тип | Среднее | СКО | Асимметрия | Эксцесс | Медиана |
|---|---|---|---|---|---|
| После очистки | $1{,}73*10^{-8}$ | $1{,}02*10^{-8}$ | 0,77399 | 0 | $1{,}57*10^{-8}$ |
| До очистки | $1{,}77*10^{-8}$ | $1{,}09*10^{-8}$ | 1,30199 | 0 | $1{,}58*10^{-8}$ |

Таблица 6.

Результаты ЭВО

| | | Сериализация, с/байт | | Десериализация, с/байт | | Увеличение сообщения | |
|---|---|---|---|---|---|---|---|
| ЯП | Тип | Среднее | СКО | Среднее | СКО | Среднее | СКО |
| python | json | $4{,}14*10^{-8}$ | $1{,}73*10^{-8}$ | $3{,}12*10^{-8}$ | $1{,}35*10^{-8}$ | 1,898979121 | 0,346021275 |
| python | yaml | $6{,}99*10^{-6}$ | $5{,}28*10^{-7}$ | $1{,}18*10^{-5}$ | $4{,}52*10^{-6}$ | 1,571260914 | 0,337084183 |
| python | toml | $3{,}74*10^{-7}$ | $1{,}66*10^{-7}$ | $1{,}39*10^{-6}$ | $5{,}59*10^{-7}$ | 1,764754711 | 0,442542795 |
| python | xml | $1{,}05*10^{-5}$ | $8{,}04*10^{-7}$ | $7{,}62*10^{-7}$ | $1{,}65*10^{-7}$ | 2,370007034 | 0,294617121 |
| javascript | json | $1{,}46*10^{-8}$ | $5{,}71*10^{-9}$ | $1{,}77*10^{-8}$ | $7{,}47*10^{-9}$ | 1,633367029 | 0,263202166 |
| javascript | yaml | $1{,}38*10^{-7}$ | $5{,}51*10^{-8}$ | $1{,}62*10^{-7}$ | $9{,}48*10^{-9}$ | 1,580975685 | 0,330321323 |
| javascript | toml | $1{,}53*10^{-7}$ | $6{,}36*10^{-8}$ | $2{,}54*10^{-7}$ | $1{,}34*10^{-7}$ | 1,878057338 | 0,513501418 |
| javascript | xml | $1{,}81*10^{-7}$ | $5{,}24*10^{-8}$ | $3{,}68*10^{-7}$ | $1{,}20*10^{-7}$ | 2,375869799 | 0,255453062 |

Таблица 7.

Интегральный критерий в зависимости от пропускной способности КС

| | | Пропускная способность КС | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | 1b/c | | 1Kb/c | | 1Mb/c | | 1Gb/c | |
| ЯП | Тип | Среднее | СКО | Среднее | СКО | Среднее | СКО | Среднее | СКО |
| python | json | 1,898 | 0,1197 | 0,0018 | $1{,}20*10^{-7}$ | $2{,}01*10^{-6}$ | $1{,}59*10^{-13}$ | $1{,}17*10^{-7}$ | $3{,}64*10^{-15}$ |
| python | yaml | 1,571 | 0,1136 | 0,0015 | $1{,}21*10^{-7}$ | $2{,}08*10^{-5}$ | $1{,}68*10^{-10}$ | $1{,}93*10^{-5}$ | $1{,}60*10^{-10}$ |
| python | toml | 1,764 | 0,1958 | 0,0017 | $1{,}97*10^{-7}$ | $4{,}31*10^{-6}$ | $4{,}10*10^{-12}$ | $2{,}55*10^{-6}$ | $2{,}60*10^{-12}$ |
| python | xml | 2,370 | 0,0867 | 0,0023 | $8{,}81*10^{-8}$ | $1{,}32*10^{-5}$ | $1{,}14*10^{-11}$ | $1{,}09*10^{-5}$ | $9{,}95*10^{-12}$ |
| javascript | json | 1,633 | 0,0692 | 0,0016 | $6{,}93*10^{-8}$ | $1{,}68*10^{-6}$ | $8{,}22*10^{-14}$ | $5{,}21*10^{-8}$ | $9{,}48*10^{-16}$ |
| javascript | yaml | 1,580 | 0,1091 | 0,0015 | $1{,}09*10^{-8}$ | $1{,}88*10^{-6}$ | $2{,}57*10^{-13}$ | $2{,}98*10^{-7}$ | $4{,}12*10^{-14}$ |
| Javascript | toml | 1,878 | 0,2636 | 0,0018 | $2{,}64*10^{-7}$ | $2{,}46*10^{-6}$ | $8{,}37*10^{-13}$ | $5{,}83*10^{-7}$ | $2{,}07*10^{-13}$ |
| Javascript | xml | 2,375 | 0,0651 | 0,0023 | $6{,}53*10^{-8}$ | $3{,}57*10^{-6}$ | $3{,}68*10^{-13}$ | $1{,}19*10^{-6}$ | $1{,}51*10^{-13}$ |

Изменение $a$ приведет к пропорциональному увеличению интегрального критерия. Положим $a = 1$. Рассмотрим интегральный критерий в зависимости от пропускной способности канала передачи данных. В табл. 7 приведены результаты.

## Анализ полученных результатов

На рисунке 3 и 4 приведена визуализация результатов ЭВО в части коэффициента увеличения сообщения. Для удобства вертикальными пунктирами отмечены математические ожидания соответствующих распределений.

Заметим, что величины несколько отличаются ввиду разных параметров формирования строкового представления в каждой конкретной библиотеке языка. При этом можно заметить общие тенденции для обоих языков:

1. YAML является наиболее предпочтительным форматом для хранения информации в силу наименьшего МО коэффициента увеличения сообщения.
2. XML является наименее предпочтительным выбором для хранения информации в силу наибольшего МО коэффициента увеличения сообщения.

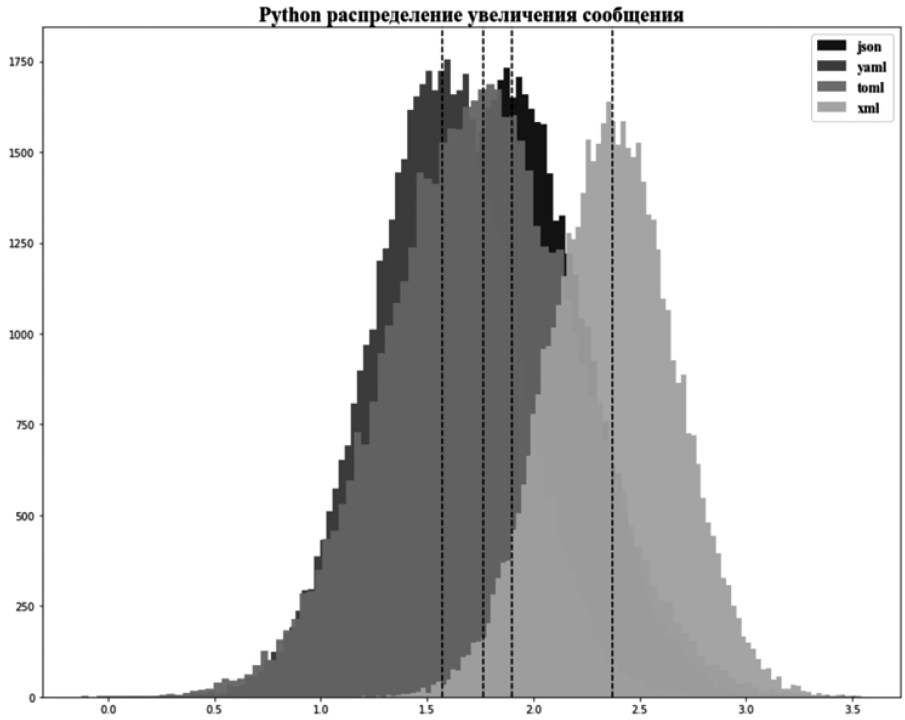


Рис. 3. Распределения увеличений сообщения в зависимости от формата (Python)

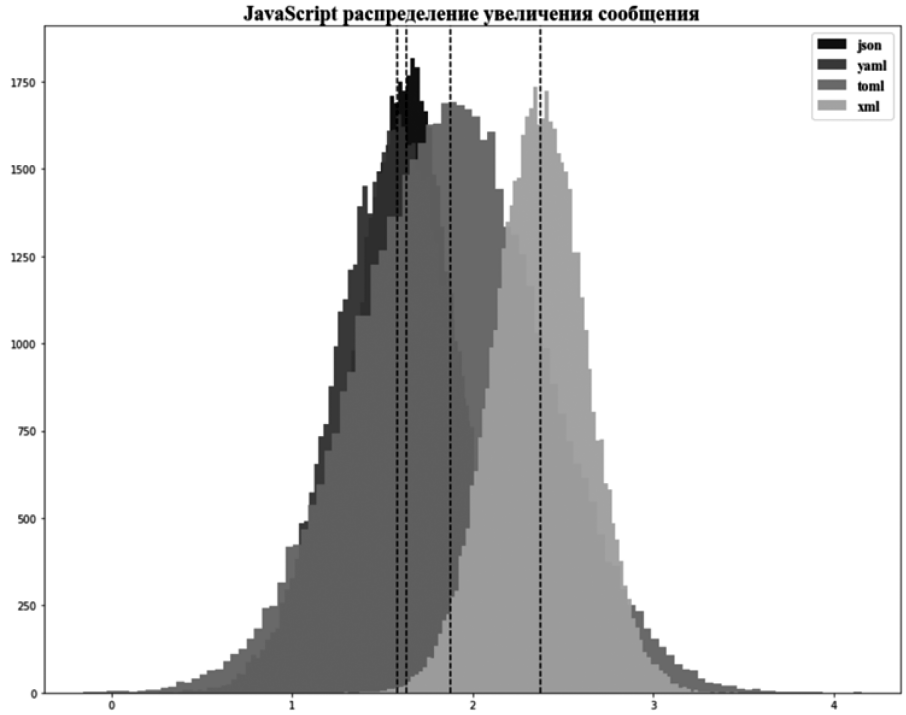


Рис. 4. Распределения увеличений сообщения в зависимости от формата (JavaScript)

Рассмотрим результаты оценки интегрального критерия. Ввиду небольшой выборки точек в таблице 7. Рассмотрим диаграмму зависимости математического ожидания интегрального критерия от пропускной способности канала связи (байт/с). Для удобства восприятия диаграммы приведены в логарифмическом масштабе (рис. 5).

Нетрудно заметить, что для пропускной способности величиной примерно до $10^4$ байт/с наиболее предпочтительным является вариант формата YAML. При значениях пропускной способности выше наиболее предпочтительным вариантом становится JSON. Очевидно, данное явление получается ввиду уменьшения влияния в сумме интегрального критерия времени передачи данных через канал и, как следствие, уменьшение влияния коэффициента увеличения сообщения на итоговый результат.

Обратим внимание, что для языка JavaScript на всех этапах наименее предпочтительным форматом передачи данных является XML. Также на данной диаграмме следует обратить внимание на первое серьезное отличие показателей разных языков, а именно тот факт, что наименее предпочтительным форматом передачи данных для Python при высокой пропускной способности является YAML. Данный факт можно объяснить только архитектурой языка, так как реализация библиотеки на JavaScript подобных результатов не дает.

## Заключение

Анализ показывает обоснованность современной методологии REST [12] в использовании JSON в качестве основного формата передачи данных в рамках сетевого взаимодействия. При высокой пропускной способности канала связи наиболее эффективным форматом является JSON, причем JSON эффективнее остальных форматов как минимум на порядок, что является существенным преимуществом.

Наиболее эффективным форматом для хранения данных является YAML. Это объяснимо его простым синтаксисом, отсутствием лишних символов для описания полей (таких как, например: кавычки), простого описания вложенности. Тем не менее YAML оказался наименее эффективным форматом для передачи данных для языка Python.

Наименее эффективным форматом для хранения информации является XML. Это является ожидаемым результатом, ввиду неэффективной архитектуры текстового представления данных. Как минимум, следует обратить внимание на дублирование наименований ключей. Данный формат также уступает JSON в эффективности при передаче. Тем не менее XML является основным форматом передачи данных в методологии SOAP [13], что свидетельствует о его конкурентоспособности.

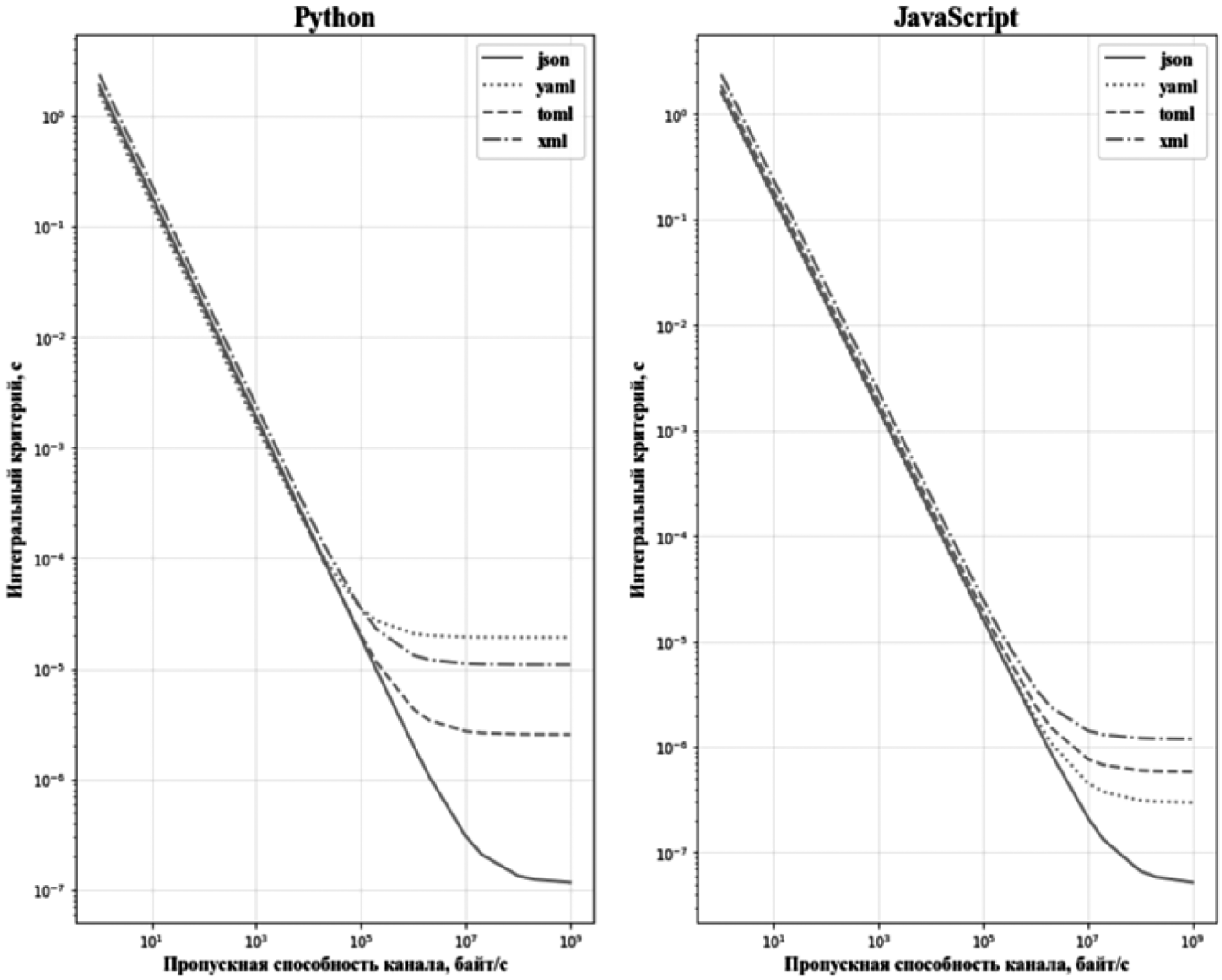


Рис. 5. Диаграмма зависимости интегрального критерия от пропускной способности канала

ЛИТЕРАТУРА

1. Сухобоков, А.А. Влияние инструментария Big Data на развитие научных дисциплин, связанных с моделированием / А.А. Сухобоков, Д.С. Лахвич // Наука и образование: научное издание МГТУ им. Н.Э. Баумана. — 2015. — № 3. — С. 207–240. — DOI 10.7463/0315.0761354.
2. Создание социальной сети на языке Golang / А.Н. Аксенов, Р.Г. Байбарин, А.Р. Васильев, Н.В. Тюлькина // Естественные и технические науки. — 2020. — № 5(143). — С. 88–94.
3. Пишем свой протокол поверх UDP (официальный блок компании VK Group) [Электронный ресурс] // vk.com URL: https://vk.com/@scienceandlive-pishem-svoi-protokol-poverh-udp
4. Профилирование и оптимизация программ на Go (официальный блог компании Badoo) [Электронный ресурс] // habr.com URL: https://habr.com/ru/company/badoo/blog/301990/
5. Документация Nginx [Электронный ресурс] // nginx.org URL: https://nginx.org/ru/docs/ (дата обращения 20.09.2019)
6. Centrifugo — 3.5 миллионов оборотов в минуту [Электронный ресурс] // habr.com URL: https://habr.com/ru/post/326236/
7. Виноградов, В.И. Постреляционные модели данных и языки запросов / В.И. Виноградов, М.В. Виноградова. — Москва: Московский государственный технический университет им. Н.Э. Баумана, 2017. — 100 с. — ISBN 978-5-7038-4283-6.
8. Введение в JSON [Электронный ресурс] // json.org URL: https://www.json.org/json-ru.html
9. XML для начинающих [Электронный ресурс] // support.microsoft.com URL: https://support.microsoft.com/ru-ru/office/xml-для-начинающих-a87d234d-4c2e-4409-9cbc-45e4eb857d44
10. Формат YAML [Электронный ресурс] // symphony.ru URL: https://symfony.ru/doc/current/components/yaml/yaml_format.html
11. Официальный портал TOML [Электронный ресурс] // toml.io URL: https://toml.io/en/
12. Архитектура REST [Электронный ресурс] // habr.com URL: https://habr.com/ru/post/38730/
13. SOAP [Электронный ресурс] // ibm.com URL: https://www.ibm.com/docs/ru/rsas/7.5.0?topic=standards-soap