DOI 10.37882/2223-2966.2024.4-2.09

АНАЛИЗ ЭФФЕКТИВНОСТИ ВЗАИМОДЕЙСТВИЯ КОМПОНЕНТОВ КЛИЕНТ-СЕРВЕРНОЙ АРХИТЕКТУРЫ ИНФОРМАЦИОННОЙ СИСТЕМЫ

ANALYZING THE EFFICIENCY OF INTERACTION BETWEEN COMPONENTS OF CLIENT-SERVER ARCHITECTURE OF INFORMATION SYSTEM

B. Goryachkin
A. Kanev
S. Poghosyan

Summary. The article analyzes the efficiency of interaction of client-server architecture components. The main focus is on the formats of textual representation of transmitted messages. The processes of transformation of these messages between language structure and string representation are investigated. The analysis considers three popular formats: JSON, YAML and TOML, in the context of three programming languages: Go, Python and JavaScript. The performance of the formats is evaluated using an integral criterion that allows comparing different aspects and reducing a multi-criteria problem to a single-criteria one. The research results demonstrate the advantages of JSON in the speed of serialization and deserialization processes.

Keywords: serialization, deserialization, JSON, YAML, TOML, client-server architecture, ergonomics.

Горячкин Борис Сергеевич
кандидат технических наук, доцент, Московский государственный технический университет им. Н.Э. Баумана
bsgor@mail.ru
Канев Антон Игоревич
старший преподаватель, Московский государственный технический университет им. Н.Э. Баумана
aikanev@bmstu.ru
Погосян Сос Левонович
Московский государственный технический университет им. Н.Э. Баумана
pogosyan777.sp@gmail.com

Аннотация. Статья освещает анализ эффективности взаимодействия компонентов клиент-серверной архитектуры. Основной фокус сделан на форматы текстового представления передаваемых сообщений. Исследуются процессы преобразования этих сообщений между структурой языка и строковым представлением. В рамках анализа рассматриваются три популярных формата: JSON, YAML и TOML, в контексте трех языков программирования: Go, Python и JavaScript. Осуществляется оценка эффективности форматов с помощью интегрального критерия, позволяющего сравнить различные аспекты и свести многокритериальную задачу к однокритериальной. В результате исследований демонстрируются преимущества JSON в скорости процессов сериализации и десериализации.

Ключевые слова: сериализация, десериализация, JSON, YAML, TOML, клиент-серверная архитектура, эргономичность.

Введение

Реалии развития информационных технологий на сегодняшний день таковы, что сайтами и интернет-приложениями могут одновременно пользоваться сотни и даже миллионы человек. Все они обращаются к одному ресурсу, который должен выдержать большое количество запросов, корректно их обрабатывать и возвращать ответы за, относительно, малое количество времени. Для обеспечения этих требований все больше поставщиков программного обеспечения (ПО) внедряют серверную обработку данных. Благодаря клиент-серверной [1] архитектуре эта цель становится достигнутой.

На сегодняшний день практически все сайты и интернет-ресурсы построены на клиент-серверной архитектуре. Также ее используют десктопные приложения, которые передают данные по сети. Стоит отметить, что на ровне с доступностью, на перевод обработки данных на сторону сервера повлияли также и другие факторы, такие как: обеспечение безопасности бизнес-процессов, сбор данных для Big Data аналитики [2], увеличение популярности облачных решений и многое другое.

Вместе с этим, проблемы доступности серверного ПО и эффективности взаимодействия его компонентов становятся ключевыми вопросами, влияющими на удобство работы с приложением. Под эффективностью взаимодействия понимается метрика, оценивающая время, необходимое для получения ответа клиентом на выполненный им запрос.

Решение задачи по повышению эффективности взаимодействия требует комплексного подхода на разных этапах проектирования, разработки и эксплуатации системы. Примерами методов решения могут быть разработка собственного сетевого протокола для оптимизации скорости обмена данными, проведение анализа кода программного обеспечения, профилирование и выявление узких мест, интеграция горизонтального масштабирования [3], использование сервисов балансировки [4] и менеджеров очередей [5], а также другие технические решения.

Описание языков программирования

Языки программирования — это формальные языки для создания компьютерных программ. Программирование постоянно развивается, а с ним и языки программирования, которые используются разработчиками.

- Go (Golang)

Go, или Golang, — это компилируемый, строго типизированный язык программирования, созданный для решения проблем масштабируемости и эффективности [6]. Язык Go был представлен в 2009 году корпорацией Google. Его полное название — Golang — производное от «Google language». Он легок в изучении, быстр и надежен, что делает его популярным для создания высокопроизводительных систем, таких как облачные сервисы и сетевые приложения.

- Python

Python — интерпретируемый динамический язык программирования общего назначения с простым синтаксисом, что делает его идеальным для начинающих. Разработчики используют Python, потому что он эффективен, прост в изучении и работает на разных платформах. Python востребован в разработке веб-приложений, научных исследований, машинного обучения и искусственного интеллекта. Благодаря богатой экосистеме библиотек и инструментов, Python продолжает удерживать позиции лидера [7].

- Node JS

Node.js — это среда выполнения кода JavaScript вне браузера [8], которая позволяет писать серверный код для веб-страниц и веб-приложений, а также для программ командной строки.

Node.js — не отдельный язык программирования, а платформа для использования JavaScript на стороне сервера. С помощью платформы можно работать с файлами, сетью, базами данных и другими системными ресурсами на сервере.

Если говорить о языке, то как для фронтенда, так и для бэкенда используется один и тот же JavaScript, который является интерпретируемым, динамическим языком программирования. Разница только в наборе API, которые используют фронтендеры и бэкендеры. Браузерный JavaScript использует Web API, которые обеспечивают доступ к DOM и пользовательскому интерфейсу страниц и веб-приложений. Серверный JavaScript использует API, обеспечивающие доступ к файловой системе приложений, HTTP-запросам, потокам.

Описание форматов представления данных

Эти форматы являются весьма распространенными и широко применяемыми в различных областях информационных технологий, таких как сетевое взаимодействие, конфигурационные файлы, непрерывная интеграция и развертывание (CI/CD), описание системных развёртываний, интерфейсов и многие другие. Ниже приведены описания форматов текстового представления информации.

- JSON

JSON (JavaScript Object Notation) — текстовый формат обмена данными, основанный на JavaScript [9]. Но при этом формат независим от JS и может использоваться в любом языке программирования. Чаще всего используется в REST API для обмена между клиентской и серверной частью приложений, в API и для хранения вспомогательных данных в простом текстовом формате, который позволяет разрабатывать приложения и данные для них параллельно. Этот формат также удобно передавать по сети в виде запросов/ответов к API.

- YAML

YAML («Yet Another Markup Language») — это язык для сериализации данных, который отличается простым синтаксисом и позволяет хранить сложноорганизованные данные в компактном и читаемом формате [10].

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.

- TOML

TOML — это минимальный формат файла конфигурации. Целью TOML является простота чтения, однозначное сопоставление со словарями и простота анализа различных структур данных. TOML имеет спецификацию с открытым исходным кодом [11], которая получила поддержку сообщества. TOML поддерживается многими языками программирования, такими как C, C#, Dart, Elixir, Erlang, Go, Java, PHP, Python, Ruby, Swift и т. д. Файлы TOML в основном состоят из пар ключ/значение, разделов/таблиц, комментариев и должны быть допустимым документом Unicode в кодировке UTF-8. TOML поддерживает типы данных String, Integer, Float, Boolean, Datetime, Array и Table (хэш-таблица/словарь).

Для наглядности рассмотрим простую структуру и ее представление в разных форматах (таблица 1).

Таблица 1.
Различное представление простой структуры данных

Формат данных	Структура данных
Исходная структура	Поля Имя Фамилия: Иван Иванов Год рождения: 1949 Год смерти: 1982 Профессия: актер, режиссер

JSON	{
	«name»: «John Belushi»,
	«birth_year»: 1949,
	«death_year»: 1982,
	«profession»:»actor,soundtrack,writer»
	}

Yaml	birth_year: 1949 death_year: 1982 name: John Belushi profession: actor,soundtrack,writer
Toml	birth_year = 1949 death_year = 1982 name = «John Belushi» profession = «actor,soundtrack,writer»

Критические моменты преобразования форматов

Из-за разнообразия структуры сериализуемой информации возникает ряд критических моментов, оказывающих влияние на метрики. Эти моменты включают:

1. Сложность вложенности структуры. Различные форматы сериализации используют разные способы обозначения вложенных структур. Некоторые форматы требуют больше дополнительных символов для описания вложенности, а некоторые добавляют символы для каждого поля вложенной структуры (как, например, YAML).
2. Размер ключей и значений. Если размер полей в структуре увеличивается, то это может привести к увеличению объема информации, необходимой для их представления в текстовом виде. Однако, для больших значений, увеличение размера может не иметь существенного влияния на общий объем данных, так как накладные расходы на описание структуры могут стать незначительными по сравнению с объемом данных.
3. Обработка массивов. Различные форматы могут представлять массивы по-разному, что может влиять на объем информации и накладные расходы для конкретного формата.
4. Обработка чисел с плавающей точкой. При сериализации чисел с плавающей точкой могут возникать неожиданные затраты памяти из-за различий в их текстовом представлении. Например, для чисел с большим количеством знаков после запятой может потребоваться больше памяти для их хранения в текстовом формате, чем для чисел с меньшим количеством знаков.

Анализ библиотек языков программирования

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 Go, Python и JavaScript, наравне с этим фактором также рассмотрены сторонние библиотеки.

Стоит отметить, что JSON на текущий момент является одним из самых распространенных форматов данных для обмена сообщениями между клиентом и сервером. Именно по этой причине в статье рассматриваются не только стандартные библиотеки выбранных языков программирования, но и сторонние наиболее популярные реализации.

Библиотеки языков программирования — представляют из себя готовый набор функций и объектов. Обычно такие наборы объединены назначением или сферой использования и предназначены для ускорения и упрощения работы.

- Golang

В Go попытались объединить скорость, характерную для C-подобных языков, и лёгкость разработки, характерную для Python. Его задумывали как универсальный, и с этой ролью он справляется, но лучше всего Go показывает себя в разработке серверных приложений: парсеров, сложных вычислительных систем, многопоточных приложений. Используется в первую очередь в бэкенде, то есть в разработке логики приложений и сайтов.

Главная особенность Go — в его минимализме. В то время как другие языки меняются, обрастая новыми функциями, Go изначально был создан, чтобы идеально решать поставленную перед ним задачу, поэтому чаще меняется только в сторону улучшения уже существующих инструментов, а не добавления новых.

Хорошим подтверждением тому, что у Go хороший стандартный инструментарий для работы — является стандартный пакет для работы с сообщениями в формате JSON — encoding/json [12]. У нее хорошая скорость сериализации и десериализации данных, которой хватает в большинстве реальных задач, а также высокая надежность и стабильность. Многие большие компании не отказываются от его использования в производственных задачах.

Несмотря на свои преимущества, иногда, стандартного пакета языка бывает недостаточно, особенно, если речь идет о высоконагруженных серверных приложениях, где каждая миллисекунда играет значение. В таких случаях обращаются к внешним пакетам, одним из которых является easyjson.

Пакет для Go — easyjson [13] предоставляет быстрый и простой способ сериализации и десериализации структуру Go в JSON и обратно на основе рефлексии. По производительности easyjson несколько раз превосходит стандартный пакет encoding/json.

Основным преимуществом easyjson является сокращение количества аллокаций памяти за счет генерирации кода, который минимизирует создание временных объектов. Это способствует уменьшению накладных расходов и повышению производительности приложения.

- Python

Python обладает множеством преимуществ, таких как простота в изучении, выразительность, обширная экосистема библиотек и фреймворков. Однако, скорость выполнения не является его сильной стороной. Именно по этой причине для python написано множество библиотек на других, более быстрых, языках программирования, таких как C, C++, Rust и т.д.

Данная проблема не обошла стороной и стандартную библиотеку для работы с форматом JSON. По этой причине большинство разработчиков предпочитают использовать внешние библиотеки.

Одним из таких решений является orjson [14], которая написана на языке программирования Rust. На данный момент эта библиотека является одной из наиболее быстрых и позволяет избавится от эффекта «бутылочного горлышка» в вопросах сериализации и десериализации данных в высоконагруженных приложениях.

- NodeJS (JavaScript)

Node JS — это среда выполнения JavaScript кода, построенная на движке Chrome V8. В начале развития JavaScript предназначался только для создания интерактивных сайтов, так как он являлся специализированным браузерным языком программирования. С программной платформой Node.js появилась возможность создавать серверные и даже десктопные приложения.

Стоит отметить, что JavaScript оказал огромное влияние на формат JSON (JavaScript Object Notation). Фактически, JSON произошёл от JavaScript, и его синтаксис тесно связан с синтаксисом объектов данного языка программирования. Не в последнюю очередь за счет этого сериализация и десериализация сообщений в формате JSON стандартными средствами языка являются достаточно быстрыми.

Несмотря на хорошие показатели стандартной библиотеки в некоторых задачах может потребоваться ускорение работы с JSON. Для таких целей была разработана библиотека typia. Она написана на чистом TypeScript и в несколько раз быстрее стандартной библиотеки.

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 данных было сгенерировано 100.000 структур. Для каждого языка и каждого формата было сделано 10 прогонов с предварительным разогревом и взята медиана для каждого из них для сериализации и десериализации соответственно 1, 100.000, 1.000.000 документов. Это было сделано для того, чтобы избежать возможных флуктуаций, возникающих под влиянием других запущенных процессов. Также перед каждым измерением производился

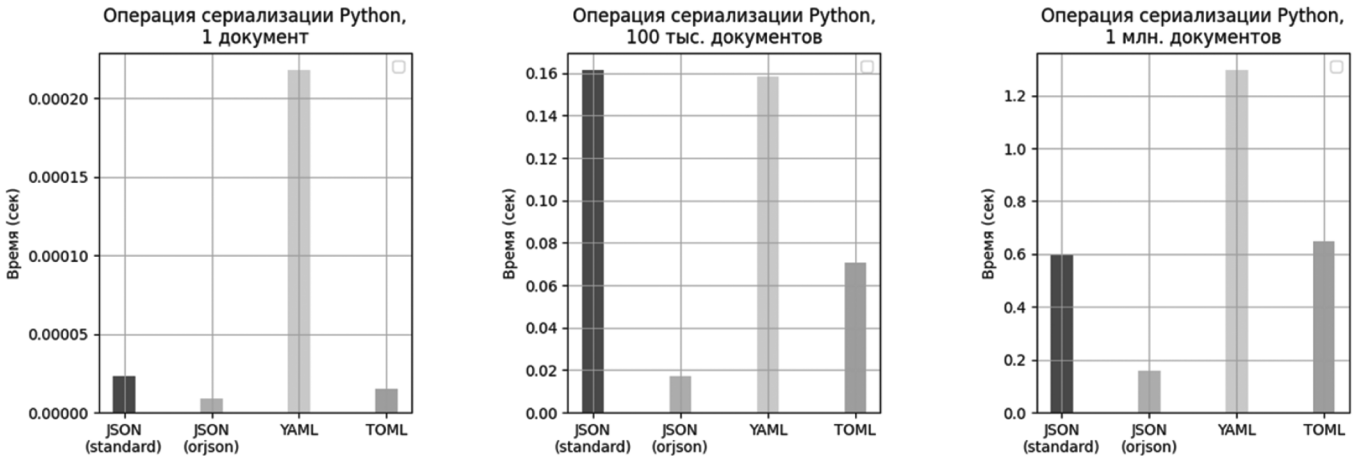

Рис. 1. Статистические параметры сериализации Python

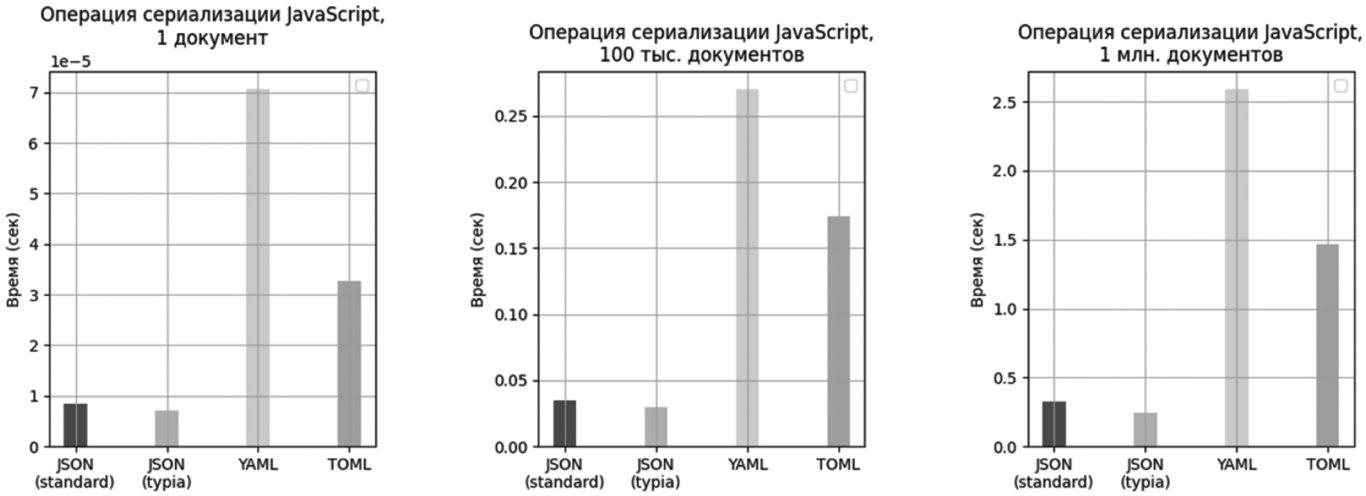

Рис. 2. Статистические параметры сериализации JavaScript

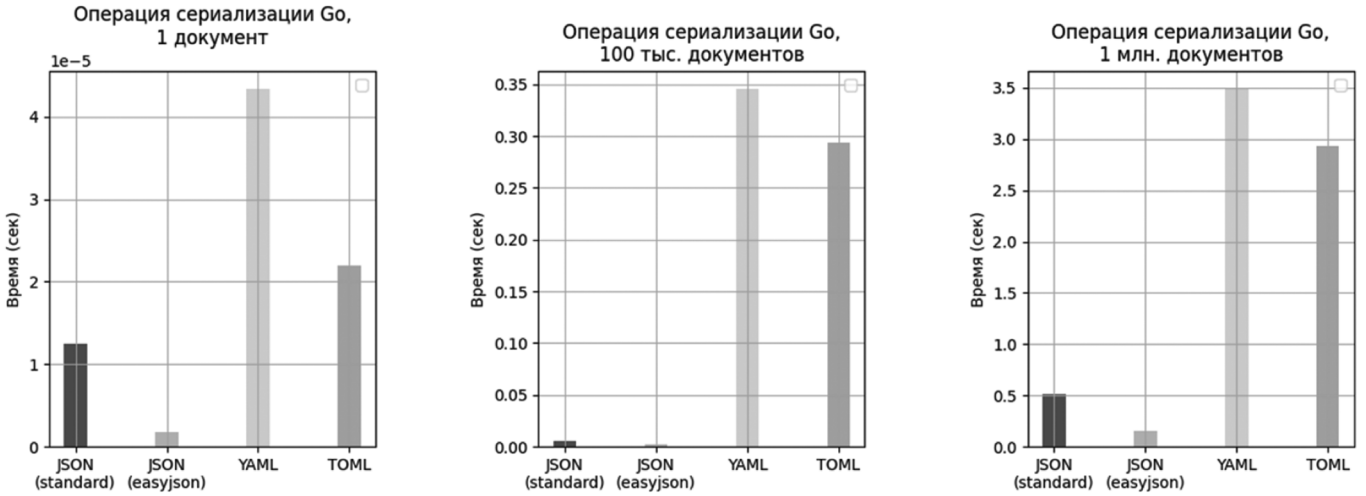

Рис. 3. Статистические параметры сериализации Go

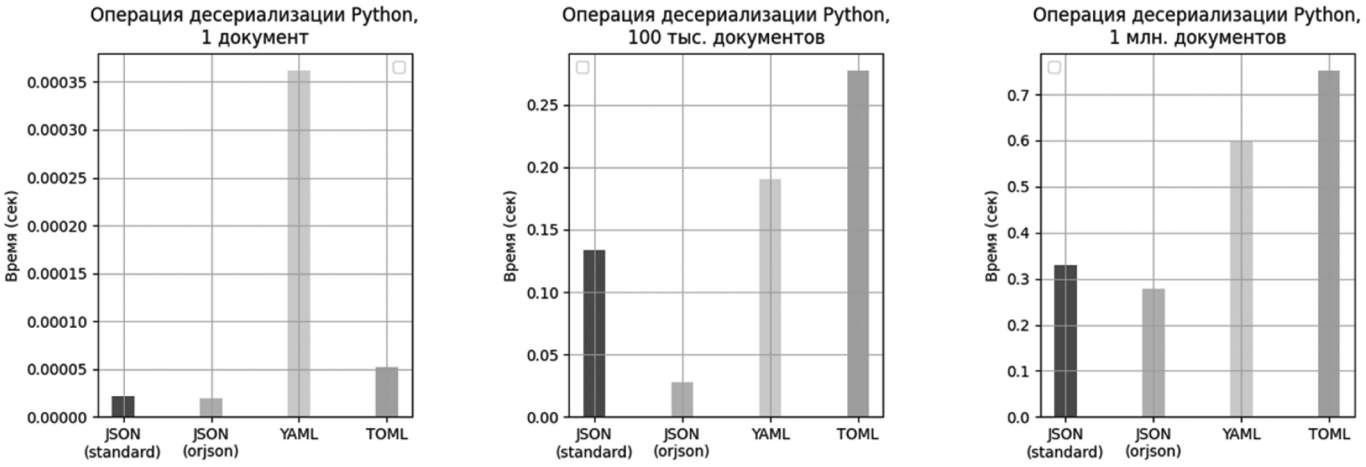

Рис. 4. Статистические параметры десериализации Python

небольшой прогрев (1–5 предварительных вызовов целевой функции). На рис. 1–6 приведены первичные статистические характеристики экспериментов.

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

Впротивовесданнымязыкамбылвзяткомпилируемый и строго типизированный язык программирования Go.

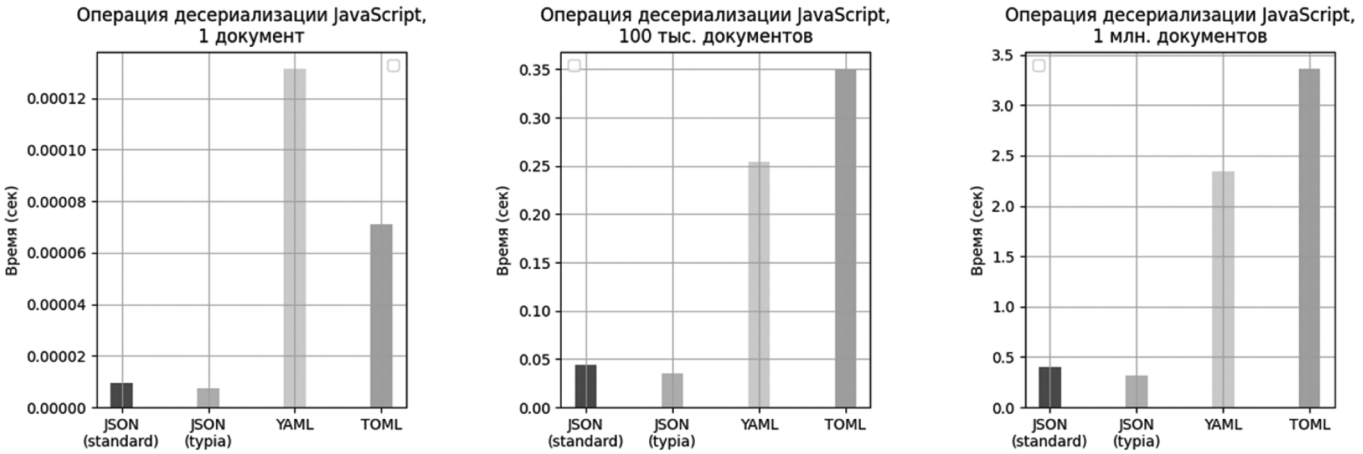

Рис. 5. Статистические параметры десериализации JavaScript

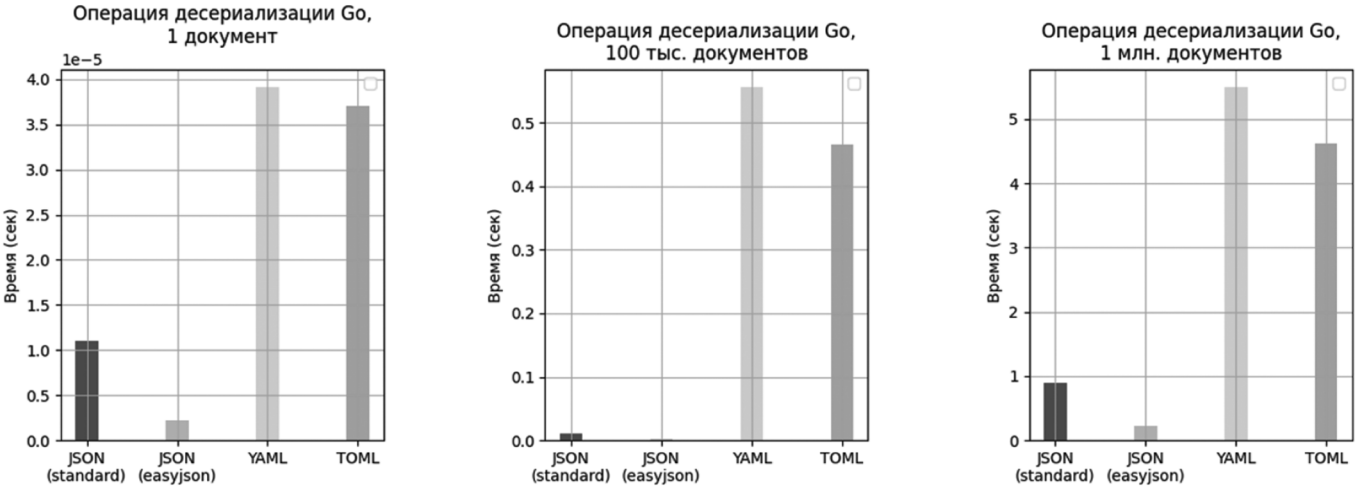

Рис. 6. Статистические параметры десериализации Go

Из рис. 1–6 можно отметить, что в независимости от языка программирования наиболее быстрым является формат JSON. В случае же TOML и YAML разница неочевидна. Для более комплексной же оценки был разработан интегральный критерий, который будет рассмотрен далее.

Еще раз следует отметить, что для работы с форматом JSON у всех языков программирования из данного перечня есть стандартные библиотеки, что нельзя сказать про форматы TOML и YAML.

На измерения также могут влиять (как в худшую, так и в лучшую сторону) типы данных, используемые в документах разных форматов. Для более точных результатов имеет смысл производить измерения на шаблоне, близком к тому, что будет использоваться на практике.

Интегральный критерий

Для комплексной оценки применяется интегральный критерий [15] — 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 $a * K$, где a — объем передаваемой полезной информации. В таком случае имеем формулу оценки интегрального критерия:

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

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

a — количество передаваемой полезной информации (байт),

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

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

K — увеличение сообщения.

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

Таблица 1.

Интегральный критерий в зависимости от пропускной способности КС

ЯП	Тип	Пропускная способность КС			
		1 b/c	1 Kb/c	1 Mb/c	1 Gb/c
		Среднее	Среднее	Среднее	Среднее
Go	JSON	1,278	0,0011	$1{,}49*10^{-6}$	$4{,}91*10^{-8}$
Go	YAML	2,1	0,0022	$2{,}86*10^{-6}$	$6{,}43*10^{-7}$
Go	TOML	1,862	0,0016	$4{,}01*10^{-6}$	$2{,}87*10^{-6}$
Python	JSON	1,562	0,0021	$2{,}01*10^{-6}$	$1{,}17*10^{-7}$
Python	YAML	1,571	0,0015	$2{,}08*10^{-5}$	$1{,}93*10^{-5}$
Python	TOML	1,764	0,0017	$4{,}31*10^{-6}$	$2{,}55*10^{-6}$
JS	JSON	1,633	0,0016	$1{,}68*10^{-6}$	$5{,}21*10^{-8}$
JS	YAML	1,58	0,0015	$1{,}88*10^{-6}$	$2{,}98*10^{-7}$
JS	TOML	1,878	0,0018	$2{,}46*10^{-6}$	$5{,}83*10^{-7}$

Анализ полученных результатов

Рассмотрим результаты оценки интегрального критерия. Ввиду небольшой выборки точек в таблице 1. Рассмотрим диаграмму зависимости математического ожидания интегрального критерия от пропускной способности канала связи (байт/с). Для удобства восприятия диаграммы приведены в логарифмическом масштабе (рис. 7).

Нетрудно заметить, что для пропускной способности величиной примерно до 10^4 байт/с наиболее для языков Python и JavaScript предпочтительным является вариант формата YAML, при этом для языка Go — TOML. При значениях пропускной способности выше наиболее предпочтительным вариантом для всех рассмотренных языков программирования 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 Go на всех этапах наименее предпочтительным форматом передачи данных является YAML. Данный факт можно объяснить только архитектурой языка, а также самой библиотеки, так как реализация библиотеки на остальных языках таких результатов не дает.

Заключение

Как показывает анализ, использование JSON в качестве основного формата передачи данных в REST API

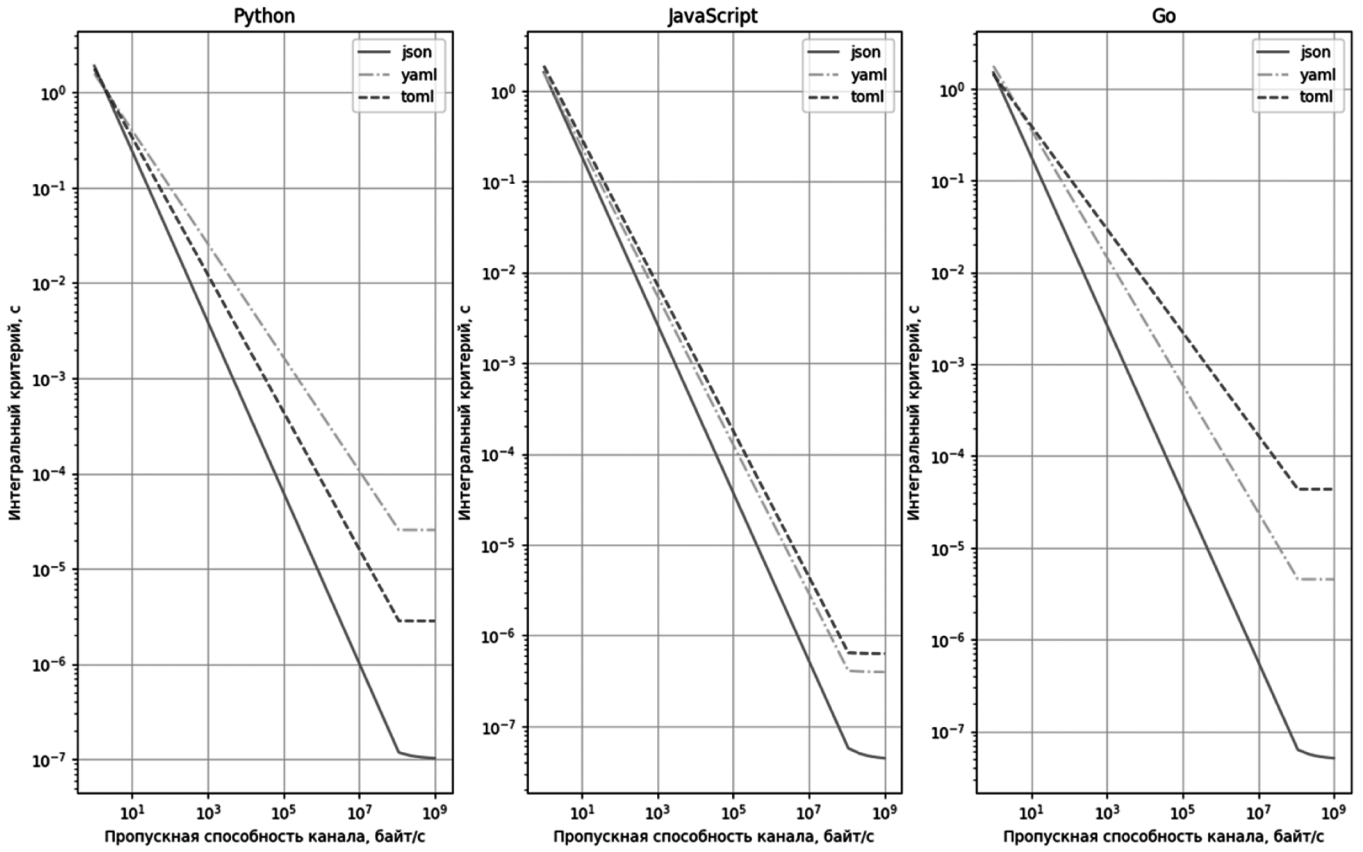

Рис. 7. Диаграмма зависимости интегрального критерия от пропускной способности

является обоснованным выбором. JSON демонстрирует наилучшую эффективность при высокой пропускной способности канала связи, превосходя другие форматы как минимум на порядок. На ровне с этим у JSON есть неоспоримое преимущество — он легко читается и воспринимается людьми, что упрощает отладку и разработку.

В то время как YAML является оптимальным форматом для хранения данных благодаря своей простоте и лаконичности, он оказывается наименее эффективным для передачи данных в Python и Go.

ЛИТЕРАТУРА

1. Клиент-серверная архитектура [электронный ресурс] // URL: https://ru.hexlet.io/courses/internet-fundamentals/lessons/client-server/theory_unit (дата обращения: 5.03.2024).
2. 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.
3. Масштабирование баз данных [электронный ресурс] // URL: https://simpleone.ru/blog/masshtabirovanie-baz-dannyh/ (дата обращения: 5.03.2024)
4. Документация Nginx [электронный ресурс] // URL: https://nginx.org/ru/docs/ (дата обращения: 7.03.2024)
5. Официальный ресурс apache kafka [электронный ресурс] // URL: https://kafka.apache.org/documentation
6. Практики применения языка программирования Go [электронный ресурс] // URL: https://go.dev/solutions/case-studies (дата обращения: 7.03.2024)
7. Индекс языков программирования TIOBE [электронный ресурс] // URL: https://www.tiobe.com/tiobe-index/ (дата обращения: 7.03.2024)
8. Официальный ресурс Node JS [электронный ресурс] // URL: https://nodejs.org/en/about (дата обращения: 7.03.2024)
9. Спецификация формата JSON [электронный ресурс] // URL: https://www.json.org/ (дата обращения: 7.03.2024)
10. Спецификация формата YAML [электронный ресурс] // URL: https://yaml.org/ (дата обращения: 7.03.2024)
11. Спецификация формата TOML [электронный ресурс] // URL: https://toml.io/en/ (дата обращения: 7.03.2024)
12. Документация стандартного пакета для работы с JSON в Go [электронный ресурс] // URL: https://pkg.go.dev/encoding/json (дата обращения: 7.03.2024)
13. Официальный ресурс пакета easyjson [электронный ресурс] // URL: https://github.com/mailru/easyjson (дата обращения: 7.03.2024)
14. Официальный ресурс пакета orjson [электронный ресурс] // URL: https://github.com/ijl/orjson (дата обращения: 7.03.2024)
15. Анализ эффективности взаимодействия компонентов клиент-серверной архитектуры информационного ресурса / Горячкин Б.С., Байбарин Р.Г. // Современная наука: актуальные проблемы теории и практики. Серия: Естественные и технические науки 2023 — № 7. — С. 48–50 DOI: 10.37882/2223-2982.2023.07.10