Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н. Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н. Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К НАУЧНО-ИССЛЕДОВАТЕЛЬСКОЙ РАБОТЕ

НА ТЕМУ:

«Классификация методов сериализации данных»

Студент ИУ7-76Б (Группа) _____ (Подпись, дата) Д. В. Варин (И. О. Фамилия)

Руководитель _____ (Подпись, дата) Д. А. Кузнецов (И. О. Фамилия)

2022 г.

РЕФЕРАТ

Расчетно-пояснительная записка 32 с., 1 рис., 5 табл., 19 источн., 1 прил.

Объектом исследования является изучение методов сериализации данных. Целью данной работы является классификация методов сериализации данных. В ходе работы были классифицированы форматы сериализации данных.

Результаты работы могут быть применены при принятии решения в выборе метода сериализации данных.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

По данным за 2020 год в России интернет пользователями являются 118 миллионов человек. Всего на 2020 год в России проживало 145.9 миллионов человек, это значит, что пользуется интернетом 81 % от всего населения.

Основными устройствами являются компьютеры и мобильные устройства – смартфоны. На них приходится 86 % и 91 % соответственно [1].

Большую часть времени пользователи используют браузер. В современном вебе подавляющее большинство сайтов являются веб приложениями. Поэтому пользователи являются клиентами, которые взаимодействуют с сервером при помощи браузера.

На серверной стороне приложения используют протоколы, позволяющие взаимодействовать между собой.

Один из используемых протоколов взаимодействия - RPC.

RPC - это удаленный вызов процедур. Реализация протокола включает в себя два компонента: сетевой протокол для обмена данными по сети - транспорт и язык сериализации.

Реализации RPC в качестве сетевого протокола используют TCP, UDP или HTTP. В качестве формата сериализации используют JSON или XML.

Цель работы — проанализировать существующие методы сериализации данных.

Для достижения поставленной цели потребуется:

— Провести обзор предметной области и описать ее термины.

— Описать анализируемые форматы сериализации данных.

— Выявить критерии сравнения и сравнить форматы.

1 Обзор RPC

1.1 Удаленный вызов процедур

RPC – это процесс, при котором программа на одной машине вызывает выполнение процедуры на другой, которая находится в другом адресном пространстве [2].

Для доставки (или получения) данных реализации RPC в качестве транспорта используют UDP [3], TCP [4] или HTTP [5] протоколы.

Отсутствие конкретного протокола связано с тем, что RPC не строго специфицированный протокол. Реализации отличаются между собой, но сам принцип остается прежним.

На рисунке 1.1 представлены компоненты взаимодействия клиента и сервера при RPC вызове.

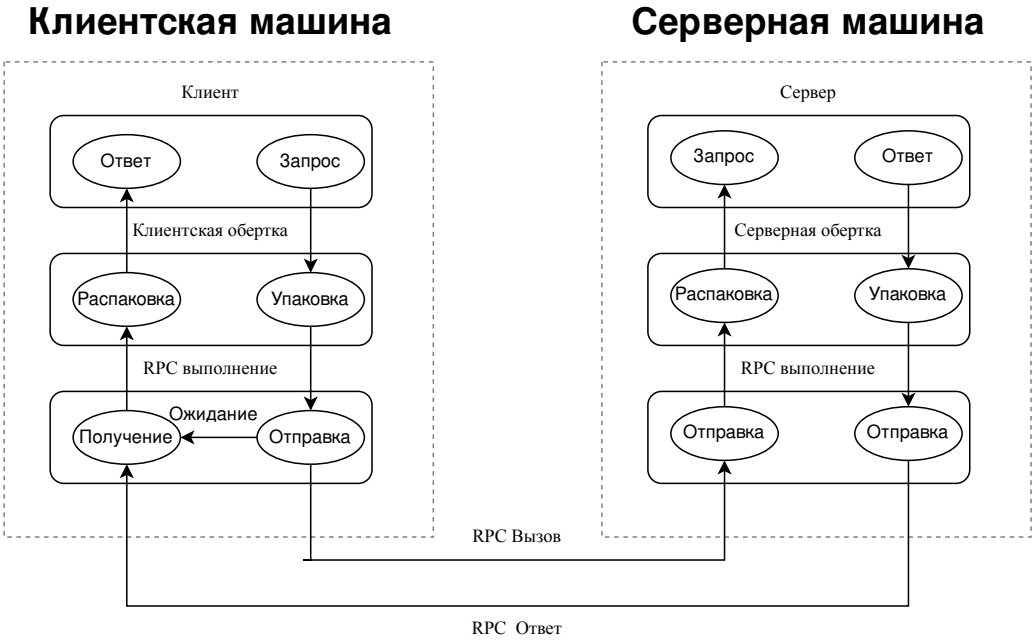


Рисунок 1.1 – Процесс взаимодействия клиента и сервера при RPC

С точки зрения удаленного вызова вызывающая программа известна как клиент, а вызываемая ею как сервер.

Рассмотрим, как выполняется RPC вызов [6].

1. Клиент вызывает процедуру-обертку клиента (stub), передавая параметры обычным способом.

2. Клиентская обертка находится в собственном адресном пространстве и упаковывает параметры в стандартный формат, копируя каждый параметр в сообщение.

3. Клиентская обертка упаковывает параметры в сообщение и передает на транспортный уровень, который отправляет его на удаленный сервер.

4. На сервере транспортный уровень передает сообщение серверной обертке, которая демаршалирует (распаковывает) параметры и вызывает нужную серверную процедуру, используя обычный механизм вызова процедур.

5. Когда серверная процедура завершается, она возвращается к серверной обертке (например, через обычный вызов процедуры return), которая упорядочивает возвращаемые значения в сообщение. Затем обертка сервера передает сообщение транспортному уровню.

6. Транспортный уровень отправляет сообщение с результатом обратно на клиентский транспортный уровень, который возвращает сообщение клиентской обертке.

7. Клиентская обертка демаршалирует возвращаемые параметры, и выполнение возвращается к вызывающей стороне.

1.2 Маршалинг

Во время RPC вызова клиент маршалирует сообщение, сервер демаршалирует. На этом этапе параметры, которые передаются в процедуру для передачи в другое адресное пространство, упаковываются в сообщение.

Маршалинг и сериализация – близкие понятия, которые имеют несколько отличий [7].

Маршалинг описывает процесс передачи объекта от клиента к серверу.

Во время маршалинга сохраняется информация о пользовательских типах данных и сериализованные данные.

Сериализация касается только перевод объекта в поток байтов в двоичном формате.

Таким образом, хотя формально сериализация является частью маршалинга, в дальнейшем эти термины будут использоваться в работе, как синонимы.

2 Форматы сериализации

2.1 Форматы сериализации данных и критерии сравнения

В RPC используется несколько форматов сериализации данных. Самые популярные построены на базе JSON [8]. В работе предлагается сравнить некоторые из них:

— JSON – сериализатор, основанный на JSON по описанной в RFC-7159 спецификации [9];

— JSON+GZIP – сериализатор JSON с сжатием GZIP [10];

— JSON+ZSTD – сериализатор JSON с сжатием Zstandard [11];

— CBOR – бинарный сериализатор на основе JSON [12];

— BSON – сериализатор на основе BSON (Binary JavaScript Object Notation) [13];

— MessagePack – бинарный сериализатор данных на основе JSON [14];

— Protobuf – бинарный сериализатор данных от Google [15].

Для сравнения форматов следует выделить критерии.

Такими являются:

— максимальная вложенность;

— нумерация полей;

— эффективность компрессии данных (размер сериализованных DTO [16]);

— latency сериализации/десериализации (время, затрачиваемое на сериализацию);

— возможность работы из различных языков (Go [17], Python [18]);

— поддержка версионности/эволюционирования структуры данных.

Следует использовать при сравнении структуры разного размера. Также протестировать структуры с большой вложенностью, чтобы выявить ограничения форматов.

В таблице 2.1 описаны постоянные критерии форматов сериализации.

Таблица 2.1 – Сравнительная таблица форматов

Формат	**Бинарный**	**Макс. вложенность**	**Нумерация полей**	**Версионность**	**Языки**
BSON	Да	65535	Нет	Да	Go,Python
CBOR	Да	Размер стека	Нет	Да	Go,Python
JSON	Нет	10000	Нет	Да	Go,Python
MessagePack	Да	Размер стека	Нет	Да	Go,Python
Protobuf	Да	100*	Да	Нет	Go,Python

По описанным характеристикам серьезных отличий у форматов нет. Поэтому следует сравнить непостоянные характеристики.

Для этого необходимо собрать тестовые данные и реализовать бенчмарки, позволяющие замерить:

— размер сериализованных данных через пропускную способность;

— время, затрачиваемое на сериализацию;

2.2 Бенчмарки для форматов сериализации

Технические характеристики

Исследование проводилось с использованием одного компьютера. Его технические характеристики:

— процессор: Apple M1 Pro;

— память: 32 Гб;

— операционная система: macOS Monterey [19] 12.4.

Анализируемые метрики

Для каждого формата сериализации есть два бенчмарка: для сериализации и десериализации.

Бенчмарки реализованы на языках программирования Go и Python3.

Таблицы ниже состоят из четырех столбцов.

1. Название формата.
2. NS/op – это среднее время выполнения каждого вызова функции в наносекундах.
3. MB/s – это пропускная способность в мегабайтах (скорость обработки).
4. B/op – это число байт, выделяемых за операцию.

Данные для бенчмарков

Для бенчмарков было взято два вида данных. Для упрощения они именуются как данные типа А и типа Б.

Данные типа А – массив JSON весом 1.7 МБ, в каждом элементе массива используются строковые значения.

Данные типа Б – массив JSON весом 120 КБ, в каждом элементе массива превалируют числовые значения: целые числа и числа с плавающей точкой.

2.2.1 Используемые формулы

NS/op рассчитывается по формуле:

$$\frac{T}{N}, \tag{2.1}$$

где T – время выполнения функции в наносекундах, а N – количество итераций.

Чем меньше NS/op, тем быстрее проходит сериализация/десериализация данных (тем более эффективен формат по времени).

MB/s рассчитывается, как:

$$\frac{B \cdot N}{T \cdot C}, \tag{2.2}$$

где B - выделяемое память в байтах на одной итерации, N – общее число итераций, T – время в секундах, C - константа, равная $1e-6$.

Чем больше MB/s, тем больше данных за единицу времени способен обрабатывать формат, что позволяет быстрее обрабатывать объемы сериализуемых данных и повышает эффективность.

B/op рассчитывается, как:

$$\frac{M_b}{N}, \tag{2.3}$$

где M_b – общее число выделенных байт, за все итерации бенчмарка, N - общее число итераций.

Чем меньше B/op, тем меньше памяти требуется выделения памяти в приложении при сериализации/десериализация (тем более эффективен формат по памяти).

2.2.2 Golang

Таблица 2.2 – Бенчмарки на golang с тестовыми данными A

Формат	**Ns/op**	**MB/s**	**B/op**
Сериализация			
JSON	1 947 565	551.08	1 109 587
JSON+GZIP	7 505 140	6.56	1 228 276
JSON+ZSTD	2 072 994	24.83	1 178 614
CBOR	3 773 309	209.15	376 789
BSON	6 719 164	182.56	2 599 949
MessagePack	2 334 210	363.70	2 097 938
Protobuf	962 667	508.97	491 522
Десериализация			
JSON	15 610 498	110.25	4 385 190
JSON+GZIP	16 417 442	3.45	1 001 751
JSON+ZSTD	15 325 713	3.65	1 220 576
CBOR	112 696	15097.51	1 705 460
BSON	10 002 288	122.65	4 582 385
MessagePack	4 367 620	194.38	730 293
Protobuf	1 963 471	249.54	2 525 047

По времени обработки и пропускной способности лучше всего справляется Protobuf, по количеству выделяемой памяти – CBOR.

Таблица 2.3 – Бенчмарки на golang с тестовыми данными Б

Формат	**Ns/op**	**MB/s**	**B/op**
Сериализация			
JSON	199 762	461.77	125 136
JSON+GZIP	1 199 464	76.90	130 031
JSON+ZSTD	247 166	373.21	126 482
CBOR	78 878	1008.31	98 403
BSON	218 095	515.15	119 181
MessagePack	102 063	1092.18	261 500
Protobuf	103 150	666.70	73 728
Десериализация			
JSON	686 517	146.93	217 785
JSON+GZIP	1 011 915	23.04	165 771
JSON+ZSTD	780 738	29.78	393 457
CBOR	10 321	9773.44	108 017
BSON	476 252	235.91	184 688
MessagePack	93 540	1191.69	48
Protobuf	58 568	1174.20	152 241

По всем показателям лучше всего себя показал CBOR.

2.2.3 Python3

Таблица 2.4 – Бенчмарки на python3 с тестовыми данными A

Формат	**Ns/op**	**MB/s**	**B/op**
Сериализация			
JSON	3 194	73.85	428 131
JSON+GZIP	439 313	0.1	875 000
JSON+ZSTD	9 603	6.1	1 137 784 615
CBOR	120 484	0.35	472 527
BSON	13 260 222	2.1	4 978 888 889
MessagePack	13 373	18.2	179 253 012
Protobuf	59 520	0.78	367 717
Десериализация			
JSON	171 805	0.12	15 625
JSON+GZIP	154 063	0.49	527 972
JSON+ZSTD	66 668	2.12	436 615
CBOR	574 250	17.3	743 483 333
BSON	15 681 333	1.1	7 436 666 667
MessagePack	207 438	0.37	701 786
Protobuf	650	22.3	12 631

На наборе данных типа A Protobuf показал себя лучше всего в скорости обработки и требуемой памяти, в пропускной способности первое место занял JSON.

Таблица 2.5 – Бенчмарки на python3 с тестовыми данными Б

Формат	**Ns/op**	**MB/s**	**B/op**
Сериализация			
JSON	12	26,56	569
JSON+GZIP	10 538	0,02	24 312
JSON+ZSTD	229	1,68	1 183
CBOR	2 913	0,01	181
BSON	7 351	0,05	38 274
MessagePack	4 547	1,33	24 312
Protobuf	104	2,27	1 859
Десериализация			
JSON	97	1,84	1 400
JSON+GZIP	373	0,32	771
JSON+ZSTD	118	3,36	1 221
CBOR	7 202	0,07	7 967
BSON	2 548	0,01	2 951
MessagePack	18 814	0,05	7 792
Protobuf	6	70,65	397

Во втором типе данных лидерство по всем характеристикам занимает Protobuf.

2.2.4 Выводы

В результате исследования были реализованы бенчмарки на двух ЯП: Golang, Python. Проведено тестирование на основе двух наборов данных: набор А – текстовые данные, набор Б – текстовые и числовые.

Результаты тестирования можно интерпретировать так:

На обоих ЯП лучше всего себя показал формат Protobuf. На втором месте CBOR.

Для наборов данных, которые похожи на используемые в тестировании, Protobuf будет самым эффективным форматом сериализации. Однако, его отличие от всех остальных форматов заключается в том, что для него необходимо описывать схему, по которой уже генерируется код, используемый для передачи данных по сети. Это повышает сложность работы с Protobuf, но повышает производительность.

В качестве альтернативы следует использовать CBOR, который чуть менее производительный, но более легкий в разработке формат сериализации.

ЗАКЛЮЧЕНИЕ

В ходе выполнения данной работы были выполнены следующие задачи:

— Проведен обзор предметной области и описаны ее термины.

— Классифицированы форматы сериализации данных.

— Выявлены критерии сравнения.

— Проведено сравнение форматов.

Все поставленные задачи были решены. Цель данной работы была достигнута.

Исследование показало, что для наборов данных, состоящих из чисел и строк, лучше всего с сериализацией справляется Protobuf. Но с ним сложно работать из-за наличия схемы, в которой нужно описывать все передаваемые структуры и затем генерировать код, который нужно интегрировать в приложение. В качестве альтернативы следует использовать CBOR, который чуть менее производительный, но более легкий в разработке формат сериализации.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. DataReportal - DIGITAL 2020: THE RUSSIAN FEDERATION [Электронный ресурс]. — URL: https://datareportal.com/reports/digital-2020-russian-federation (дата обр. 18.11.2022).
2. Remote Procedure Call [Электронный ресурс]. — URL: https://www.ibm.com/docs/en/aix/7.1?topic=concepts-remote-procedure-call (дата обр. 18.11.2022).
3. User Datagram Protocol [Электронный ресурс]. — URL: https://www.rfc-editor.org/rfc/rfc768 (дата обр. 18.11.2022).
4. Transmission Control Protocol [Электронный ресурс]. — URL: https://www.rfc-editor.org/rfc/rfc793 (дата обр. 18.11.2022).
5. Hypertext Transfer Protocol – HTTP/1.1 [Электронный ресурс]. — URL: https://www.rfc-editor.org/rfc/rfc2616 (дата обр. 18.11.2022).
6. RPC Flow [Электронный ресурс]. — URL: https://www.w3.org/History/1992/nfs_dxcern_mirror/rpc/doc/Introduction/HowItWorks.html (дата обр. 18.11.2022).
7. Schema for Representing Java(tm) Objects in an LDAP Directory [Электронный ресурс]. — URL: https://datatracker.ietf.org/doc/html/rfc2713#section-2.3 (дата обр. 18.11.2022).
8. The JavaScript Object Notation (JSON) Data Interchange Format [Электронный ресурс]. — URL: https://www.rfc-editor.org/rfc/rfc8259 (дата обр. 18.11.2022).
9. JSON [Электронный ресурс]. — URL: https://www.json.org/json-en.html (дата обр. 08.12.2022).
10. Gzip is a file format and a software application used for file compression and decompression [Электронный ресурс]. — URL: https://www.gzip.org/ (дата обр. 08.12.2022).
11. Zstandard - Fast real-time compression algorithm [Электронный ресурс]. — URL: https://github.com/facebook/zstd (дата обр. 08.12.2022).
12. Concise Binary Object Representation [Электронный ресурс]. — URL: https://cbor.io/ (дата обр. 08.12.2022).

13. BSON is a binary-encoded serialization of JSON-like documents [Электронный ресурс]. — URL: https://bsonspec.org/ (дата обр. 08.12.2022).

14. MessagePack is an efficient binary serialization format [Электронный ресурс]. — URL: https://msgpack.org/ (дата обр. 08.12.2022).

15. Protocol Buffers - Google's data interchange format [Электронный ресурс]. — URL: https://developers.google.com/protocol-buffers (дата обр. 08.12.2022).

16. Data Transfer Object [Электронный ресурс]. — URL: https://martinfowler.com/eaaCatalog/dataTransferObject.html (дата обр. 18.11.2022).

17. The Go Programming Language [Электронный ресурс]. — URL: https://go.dev/ (дата обр. 09.12.2022).

18. Python Programming Language [Электронный ресурс]. — URL: https://www.python.org/ (дата обр. 09.12.2022).

19. macOS Monterey [Электронный ресурс]. — URL: https://www.apple.com/ru/macos/monterey/ (дата обр. 09.12.2022).

ПРИЛОЖЕНИЕ А

Презентация

Классификация методов сериализации данных

Студент: Варин Д. В., ИУ7-76Б
Научный руководитель: Кузнецов Д. А.

Москва, 2022 г.

Цель и задачи работы

Цель работы: проанализировать существующие методы сериализации данных

Задачи работы:

1. Провести обзор предметной области.
2. Описать термины предметной области.
3. Описать анализируемые форматы сериализации данных.
4. Выявить критерии сравнения.
5. Сравнить форматы.

Предметная область

RPC - это процесс, при котором программа на одной машине вызывает выполнение процедуры на другой.

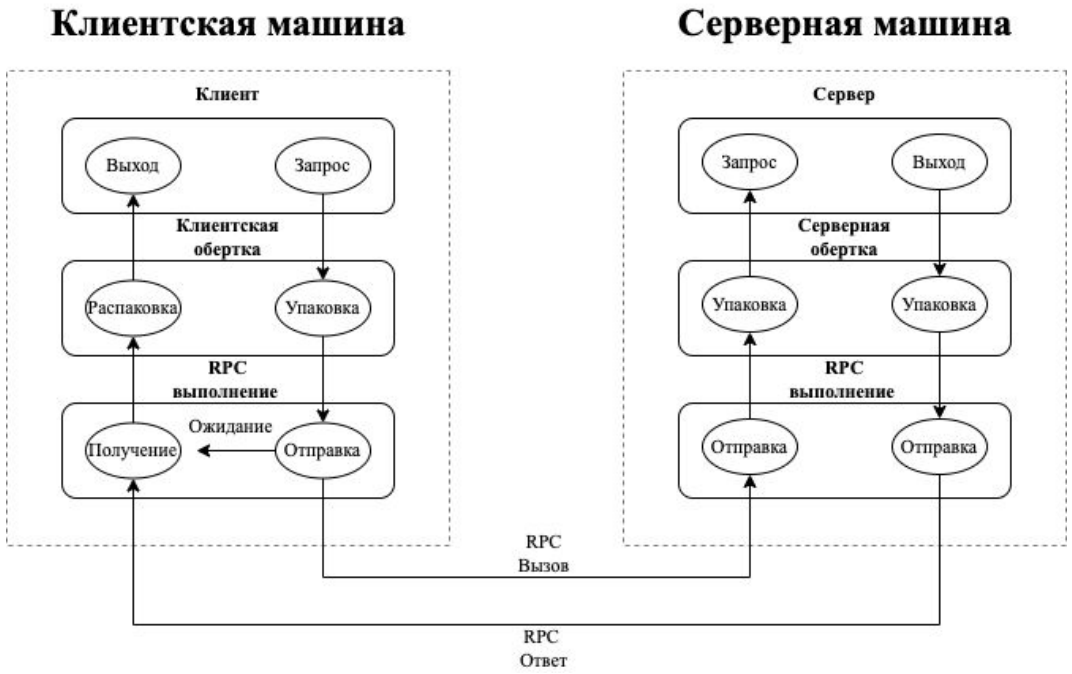
Обычно, процесс включает в себя два компонента:

— сетевой протокол для обмена данными по сети (транспорт);
— язык сериализации.

Реализации RPC в качестве транспорта используют TCP, UDP или HTTP.
В качестве формата данных используют форматы, основанные на JSON или XML.

RPC вызов

1. Клиент вызывает процедуру-обертку.
2. Обертка упаковывает параметры, копируя их в сообщение и передает их на сервер.
3. На сервере сообщение распаковывается и вызывается процедура.
4. Когда процедура завершается, она возвращается к обертке, которая упаковывает возвращаемые значения в сообщение.
5. Транспортный уровень отправляет сообщение с результатом обратно клиенту.
6. Клиент распаковывает сообщение, и возвращает результат.



Форматы сериализации

В RPC используется множество форматов сериализации данных.
Самые популярные построены на базе JSON.
В работе сравниваются:

- **JSON** - сериализатор, основанный на JSON по описанной в RFC-7159 спецификации;
- **JSON+GZIP** - сериализатор JSON с сжатием GZIP;
- **JSON+ZSTD** - сериализатор JSON с сжатием Zstandard;
- **CBOR** - бинарный сериализатор на основе JSON;
- **BSON** - сериализатор на основе BSON (Binary JavaScript Object Notation);
- **MessagePack** - бинарный сериализатор данных на основе JSON;
- **Protobuf** - бинарный сериализатор данных от Google.

Критерии сравнения

Для сравнения форматов следует выделить критерии.

Такими являются:

— максимальная вложенность;
— нумерация полей;
— эффективность компрессии данных (размер сериализованных данных);
— время, затрачиваемое на сериализацию;
— возможность работы из различных языков (Go, Python);
— поддержка версионности/эволюционирования структуры данных.

Сравнительная таблица постоянных характеристик

Формат	Бинарный	Макс. вложенность	Нумерация полей	Версионность	Языки
BSON	Да	65535	Нет	Да	Go,Python
CBOR	Да	Размер стека	Нет	Да	Go,Python
JSON	Нет	10000	Нет	Да	Go,Python
MessagePack	Да	Размер стека	Нет	Да	Go,Python
Protobuf	Да	100*	Да	Нет	Go,Python

Исследование непостоянных характеристик

Технические характеристики:

— Процессор: Apple M1 Pro.
— Память: 32 Гб.
— Операционная система: macOS Monterey 12.4.

Измеряемые характеристики:

— Название формата.
— **NS/op** - среднее время выполнения каждого вызова функции в наносекундах (чем меньше, тем лучше);
— **MB/s** - пропускная способность в мегабайтах (скорость обработки, чем больше, тем лучше);
— **B/op** - это число байт, выделяемых за операцию (чем меньше, тем лучше).

Сравнение непостоянных параметров

Golang

Строки

Формат	Ns/op	MB/s	B/op
Сериализация			
JSON	1 947 565	551.08	1 109 587
JSON+GZIP	7 505 140	6.56	1 228 276
JSON+ZSTD	2 072 994	24.83	1 178 614
CBOR	3 773 309	209.15	376 789
BSON	6 719 164	182.56	2 599 949
MessagePack	2 334 210	363.70	2 097 938
Protobuf	962 667	508.97	491 522
Десериализация			
JSON	15 610 498	110.25	4 385 190
JSON+GZIP	16 417 442	3.45	1 001 751
JSON+ZSTD	15 325 713	3.65	1 220 576
CBOR	112 696	15097.51	1 705 460
BSON	10 002 288	122.65	4 582 385
MessagePack	4 367 620	194.38	730 293
Protobuf	1 963 471	249.54	2 525 047

По времени обработки и пропускной способности лучше всего справляется Protobuf, по количеству выделяемой памяти - CBOR.

Числа и строки

Формат	Ns/op	MB/s	B/op
Сериализация			
JSON	199 762	461.77	125 136
JSON+GZIP	1 199 464	76.90	130 031
JSON+ZSTD	247 166	373.21	126 482
CBOR	78 878	1008.31	98 403
BSON	218 095	515.15	119 181
MessagePack	102 063	1092.18	261 500
Protobuf	103 150	666.70	73 728
Десериализация			
JSON	686 517	146.93	217 785
JSON+GZIP	1 011 915	23.04	165 771
JSON+ZSTD	780 738	29.78	393 457
CBOR	10 321	9773.44	108 017
BSON	476 252	235.91	184 688
MessagePack	93 540	1191.69	48
Protobuf	58 568	1174.20	152 241

По всем показателям лучше всего себя показал CBOR.

Сравнение непостоянных параметров

Python3

Строки

Формат	Ns/op	MB/s	B/op
Сериализация			
JSON	3 194	73.85	428 131
JSON+GZIP	439 313	0.1	875 000
JSON+ZSTD	9 603	6.1	1 137 784 615
CBOR	120 484	0.35	472 527
BSON	13 260 222	2.1	4 978 888 889
MessagePack	13 373	18.2	179 253 012
Protobuf	59 520	0.78	367 717
Десериализация			
JSON	171 805	0.12	15 625
JSON+GZIP	154 063	0.49	527 972
JSON+ZSTD	66 668	2.12	436 615
CBOR	574 250	17.3	743 483 333
BSON	15 681 333	1.1	7 436 666 667
MessagePack	207 438	0.37	701 786
Protobuf	650	22.3	12 631

Protobuf показал себя лучше всего в скорости обработки и требуемой памяти, в пропускной способности первое место занял JSON.

Числа и строки

Формат	Ns/op	MB/s	B/op
Сериализация			
JSON	12	26,56	569
JSON+GZIP	10 538	0,02	24 312
JSON+ZSTD	229	1,68	1 183
CBOR	2 913	0,01	181
BSON	7 351	0,05	38 274
MessagePack	4 547	1,33	24 312
Protobuf	104	2,27	1 859
Десериализация			
JSON	97	1,84	1 400
JSON+GZIP	373	0,32	771
JSON+ZSTD	118	3,36	1 221
CBOR	7 202	0,07	7 967
BSON	2 548	0,01	2 951
MessagePack	18 814	0,05	7 792
Protobuf	6	70,65	397

По всем параметрам лидерство занимает Protobuf.

Выводы

В ходе выполнения данной работы были выполнены следующие задачи

1. Проведен обзор предметной области и описаны ее термины.
2. Классифицированы форматы сериализации данных.
3. Выявлены критерии сравнения.
4. Проведено сравнение форматов.

Все поставленные задачи были решены. Цель данной работы была достигнута.

Исследование показало, что для наборов данных, состоящих из чисел и строк, лучше всего с сериализацией справляется protobuf. Но наличие схемы усложняет работу с ним.
В качестве альтернативы следует использовать CBOR, который чуть менее производительный, но более легкий в разработке формат сериализации.