ИСПОЛЬЗОВАНИЕ ОСОБЕННОСТЕЙ ПРОТОКОЛА СЕРИАЛИЗАЦИИ PROTOCOL BUFFERS ДЛЯ ОПТИМИЗАЦИИ ВЫЧИСЛИТЕЛЬНЫХ РЕСУРСОВ

Козлов В. А., Рогов М.Г.

Белорусский государственный университет информатики и радиоэлектроники, г. Минск, Республика Беларусь

В данной работе рассматривается способ уменьшения затрат вычислительных ресурсов в задачах, требующих фильтрации или выборочной десериализации protobuf-сообщений, содержащих большое количество данных.

В настоящее время большой популярностью пользуется микросервисная архитектура построения серверного программного обеспечения. В связи с необходимостью передачи больших объёмов данных между микросервисами всё большее количество компаний выбирают протокол Protocol Buffers для эффективного обмена информацией. В связи с этим возникают задачи, в которых десериализация больших сообщений занимает значительную часть времени ответа сервиса.

Цель данной работы – рассмотреть алгоритмы сериализации и десериализации, используемые в данном протоколе, с целью оптимизации ресурсов в некоторых прикладных задачах.

Структура сериализуемых данных описывается в файле с расширением .proto. На рисунке 1 представлен пример [1] объявления protobuf-сообщения, описывающего человека (поля имя, идентификационный номер, адрес электронной почты).

```
message Person {
  optional string name = 1;
  optional int32 id = 2;
  optional string email = 3;
}
```

Рисунок 1 – Пример protobuf-сообщения

После описания формата данных, созданный файл компилируется protoc-компилятором. В результате компиляции генерируется файл с описанием структуры данных в выбранном языке программирования. На рисунке 2 представлен пример работы со сгенерированным классом на языке C++.

```
// C++ code
Person john;
fstream input(argv[1],
    ios::in | ios::binary);
john.ParseFromIstream(&input);
id = john.id();
name = john.name();
email = john.email();
```

Рисунок 2 – Работа со сгенерированным классом на языке C++

Рассмотрим верхнеуровнево алгоритм сериализации сообщения [2]. В общем виде формат представляет собой закодированную последовательность полей, состоящих из ключа и значения. В качестве ключа выступает номер, определённый для каждого поля сообщения в proto-файле. Перед каждым полем указываются совместно закодированные номер поля в формате varint и тип поля. Если в качестве типа указана строка, вложенное сообщение, повторяющееся сообщение или набор байт, то следом идёт размер данных в формате varint [3]. Далее идёт значение, соответствующее полю (данные).

Тот факт, что в сериализованном сообщении для каждого поля известно кол-во занимаемых им байт, позволяет игнорировать при десериализации поля, которых больше нет в сообщении (например, были удалены в новой версии ПО). Также заметим, что алгоритм сериализации никак не использует названия полей и название сообщения. Эти два замечания позволяют сделать следующий вывод. Пусть имеется два protobuf-сообщения. Для простоты восприятия назовём их TestProto и TestProtoThin. Пусть TestProtoThin представляет из себя подмножество полей TestProto, сохраняя их типы данных и номера. Тогда сериализованное сообщение TestProto будет корректно десериализовано в TestProtoThin, при этом лишние поля при десериализации будут проигнорированы.

Рассмотрим следующую прикладную задачу. В сервис поступает большое количество сериализованных сообщений типа TestProto, которое, в свою очередь, имеет очень большое количество полей, и даже в сериализованном виде занимает большое количество памяти. В соответствии с бизнес-логикой, сервис должен производить некоторые манипуляции над приходящими сообщениями, но лишь над теми, которые подходят под некое условие: например, TestProto имеет поле timestamp, и манипуляции стоит производить только над сообщениями, у которых timestamp больше определённого значения. Получаем ситуацию, в которой часть сообщений десериализуется только ради проверки одного поля.

Используя замечание выше, можно произвести следующую оптимизацию: создать сообщение TestProtoThin, имеющее лишь одно поле timestamp. В коде обработки приходящее сообщение десериализуется сначала в TestProtoThin, тем самым все тяжеловесные поля пропускаются и их десериализация не происходит. При соответствии timestamp-а условиям, сообщение десериализуется полностью в TestProto, чтобы все поля стали доступны.

Таким образом, если на практике данные таковы, что большая часть приходящих сообщений не подходит под критерии, данная оптимизация позволит существенно снизить затраты на обработку лишних данных.

Список использованных источников:

1. Protocol Buffers Documentation [Электронный ресурс]. – Режим доступа: https://protobuf.dev/programming-guides/encoding/. – Дата доступа: 10.04.2024.

2. Protocol Buffers [Электронный ресурс]. – Режим доступа: https://ru.wikipedia.org/wiki/Protocol_Buffers. – Дата доступа: 11.04.2024.

3. Разбор Protobuf в Visual Studio под C++ [Электронный ресурс]. – Режим доступа: https://habr.com/ru/articles/645505/. – Дата доступа: 11.04.2024.