УДК 004.738

ИСПОЛЬЗОВАНИЕ ПРОТОКОЛА ПЕРЕДАЧИ ДАННЫХ «PROTOCOL BUFFERS»

Филиппов С.С., студент гр. ИТб–152, IV курс
Научный руководитель: Алексеева Г. А., старший преподаватель
Кузбасский государственный технический университет
имени Т.Ф. Горбачева.
г. Кемерово

При разработке веб-приложения с архитектурой клиент-сервер неизбежно возникает проблема обмена данными между его частями. В настоящие время существует большое количество разнообразных форматов передачи данных. Решение проблемы выбора формата передачи данных, может оказать значительное влияние на производительность приложения в целом. От формата передачи данных зависит время отклика, объем передаваемой информации по каналу связи, расширяемость и портируемость.

Рассмотрим влияние формата передачи данных на примере приложения для мобильной станции технического обслуживания (МСТО).

Для решения поставленной задачи была разработана схема передачи, обработки и хранения данных (рисунок 1).

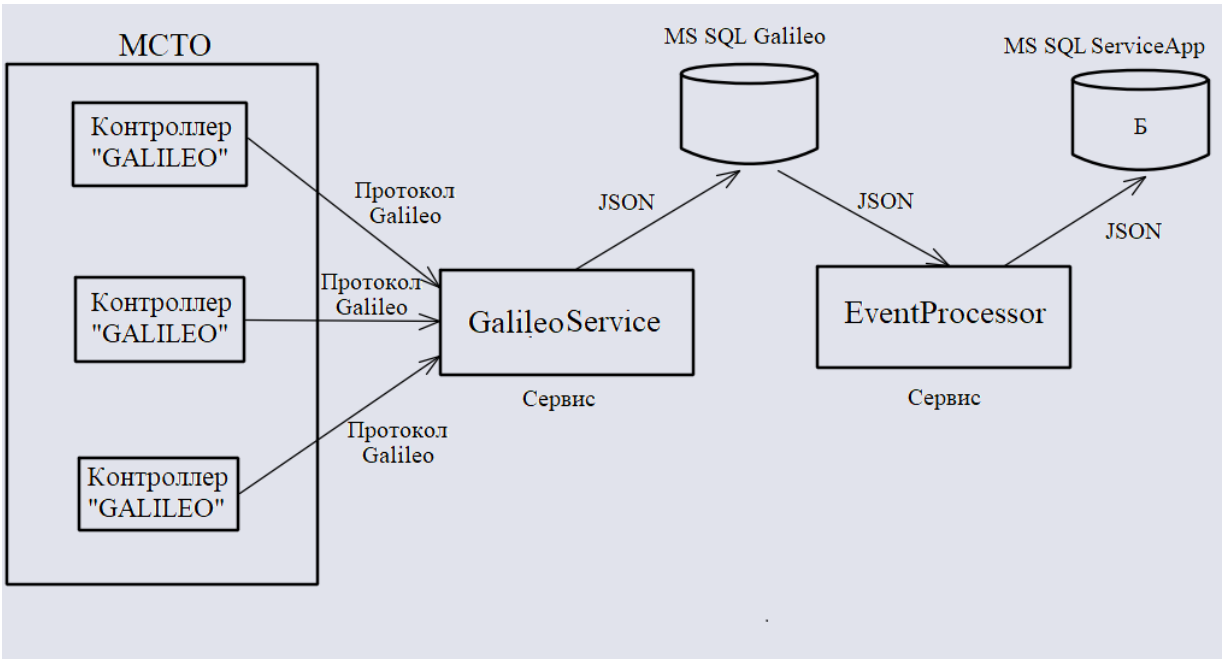

Рисунок 1

Как видно из схемы, на каждом МСТО стоят 3 контролера «GALILEO», они собирают данные о состоянии станции и отправляют их на сервер стандартным протоколом galileo [1]. Протокол galileo – это массив из данных:

название ключа – значение. Сервис «GalileoService» обрабатывает пришедшие данные и сериализует их в формат JSON и сохраняет в таблице MS SQL Galileo.

В свою очередь сервис «EventProcessor» обращается к таблице Galileo и получает все строки, находящиеся в ней и десериализует их. Стоит учесть тот факт, что пришедшие данные являются показанием приборов, и чтобы и их преобразовать в удобный для пользователя вид, используется таблица тарировки. После преобразования данных, они привязываются к МСТО и сохраняются в основную базу MS SQL ServiceApp.

При детальном рассмотрении схемы можно заметить, что сохранение данных в таблице Galileo, дает только лишнюю нагрузку на сервер, так как передача данных от сервиса к сервису проще, но это только на первый взгляд. Такая схема передачи, позволяет регулировать время обработки независимо от «GalileoService», что дает значительный прирос к производительности.

Первые тесты показали, что данная схема имеет свои изъяны:

1. Увеличение объема основной базы, что влечет за собой увеличение времени запроса-ответа.
2. Заполнение объема жёсткого диска сервера и как следствие остановка всего приложения.

Для решения этих проблем была разработана новая схема передачи данных, представленная на рисунке 2.

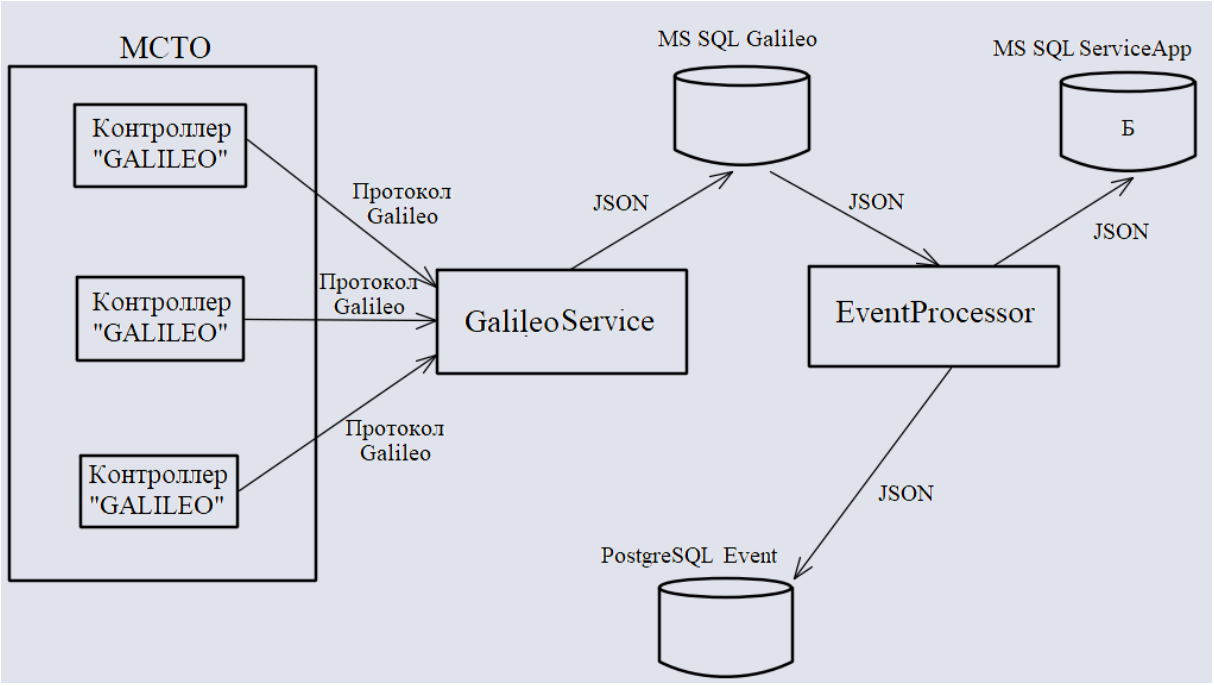

Рисунок 2

Особенностью данной схемы является хранение в основной БД только мета данных о состоянии МСТО. Все значения контролеров хранятся в БД PostgreSQL Event. Принятые меры позволили не увеличивать объем основной БД.

Для решения проблемы заполнения объема жёсткого диска, используется свойство БД PostgreSQL – секционирование таблиц, которое позволяет разбить большую по объему таблицу на несколько мелких. Благодаря чему появляется возможность, удалять полностью таблицу, что гораздо проще, чем удаление отдельных ее строк.

Новая схема позволила работать приложению полноценно, но все так же требовала вмешательства разработчика, так как БД PostgreSQL Event занимала все свободное пространство на сервере (в среднем за 20 дней) и приходилось удалять старые таблицы вручную, что приводило к потере данных.

При анализе проблемы было принято решение изменить формат передачи данных.

Рассмотрим протокол формата передачи данных, который использовался в предыдущих вариантах. Это протокол JSON.

JSON – текстовый формат обмена данными, основанный на JavaScript [2]. Как и многие другие текстовые форматы, JSON легко читается людьми. Выбор JSON в первых схемах был обоснован тем, что работать с этим форматом данных просто. Библиотека Newtonsoft для C# позволяет без труда сериализовать и десериализовать данные в формат JSON.

Достоинства JSON:

– читаемость человеком;
– сериализация без определенной схемы;
– поддержка браузеров;
– расширяемость.

В качестве недостатков JSON выделим:

– поддерживает только один тип чисел;
– не имеет определенной структуры.

Рассмотрев существующие форматы передачи данных, лучшим вариантом для решения этой проблемы стала замена JSON на бинарный, Protocol Buffers.

Protocol Buffers (ProtoBuf) – протокол сериализации (передачи) структурированных данных, предложенный Google как эффективная бинарная альтернатива текстовому формату [3, 4].

ProtoBuf обладает следующими достоинствами:

– занимает меньший объем памяти по сравнению со строковым типом;
– сериализация осуществляется по определенной схеме;
– передача данных осуществляется быстрее, чем при использовании строкового формата.
– позволяет создавать классы, которые в дальнейшем легче использовать программно.

Из минусов ProtoBuf отметим только, что он не доступен для чтения человеком.

Финальная схема передачи, обработки и хранения данных показана на рисунке 3.

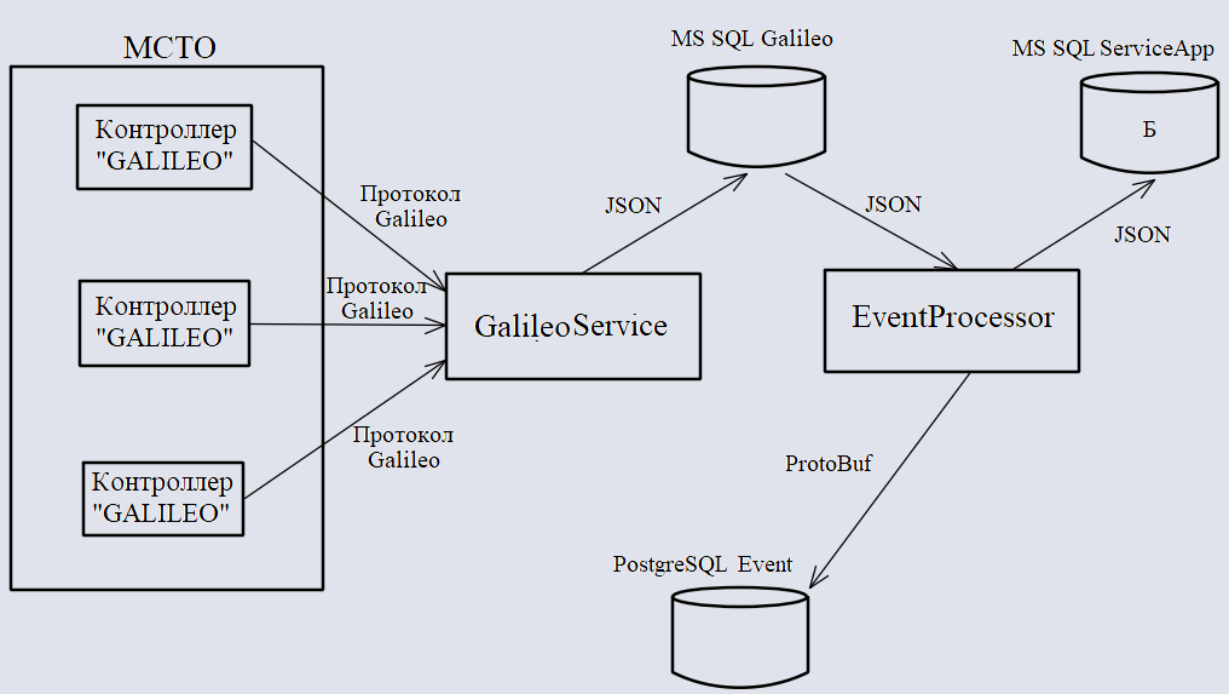

Рисунок 3

Использование Protocol Buffers на языке C# реализуется подключением библиотеки ProtoBuf-net. Для сериализации данных необходим отдельный класс – ProtoContract, в котором определяется формат сообщений (рисунок 4).

```
[ProtoContract]
public class EventData
{
    [ProtoMember(1)] public float? RealParameterName { get; set; }
    [ProtoMember(2)] public float? RealFloatParameter2{ get; set; }
    [ProtoMember(3)] public float? param3 { get; set; }
    [ProtoMember(4)] public float? param4 { get; set; }
    [ProtoMember(5)] public float? param5 { get; set; }
    [ProtoMember(6)] public float? param6 { get; set; }
    [ProtoMember(7)] public float? param7 { get; set; }
    [ProtoMember(8)] public float? param8 { get; set; }
    [ProtoMember(9)] public float? param9 { get; set; }
    [ProtoMember(10)] public float? param10 { get; set; }
    [ProtoMember(11)] public float? param11 { get; set; }
    [ProtoMember(12)] public float? param12 { get; set; }
    [ProtoMember(13)] public float? param13 { get; set; }
    [ProtoMember(14)] public float? param14 { get; set; }
    [ProtoMember(15)] public float? param15 { get; set; }
    [ProtoMember(16)] public float? param16 { get; set; }
    [ProtoMember(17)] public float? param17 { get; set; }
    [ProtoMember(18)] public float? param18 { get; set; }
    [ProtoMember(19)] public float? param19 { get; set; }
    [ProtoMember(20)] public float? param20 { get; set; }
}
}
```

Рисунок 4

В отличие от JSON, ProtoBuf позволяет сократить объем данных, за счет замены имени параметра на номер ProtoMember и изменения строкового типа данных на битовый.

Для сравнения двух видов протоколов передачи данных, проведем их анализ скорости сериализации и объёма данных [2, 3, 5]. Для этого анализа была написана простая программа, которая сравнивает вес и скорость обработки (рисунок 5).

```
static void Main(string[] args)
{
    UserContext _db = new UserContext(); // подключение DbContext;
    var InputData = _db.Galileos.ToList(); // Получение всех данных таблицы.

    foreach (var item in InputData) // Обход всех входнфх параметров
    {
        //Тест для JSON
        var json = JsonConvert.SerializeObject(item, Formatting.None, // сериализация входных параметров в JSON
                                    new JsonSerializerSettings
                                    {
                                        NullValueHandling = NullValueHandling.Ignore
                                    });
        _db.EventJSONs.Add(new EventJSON() //Добавление даххы в контекст
        {
            Date = json
        });

        //Тест для ProtoBuf
        using (var ms = new MemoryStream())
        {
            Serializer.Serialize(ms, item); // Сериализация данных в ProtoBuf
            byte[] protoItem = ms.ToArray();
            _db.EventProtoBufs.Add(new EventProtoBuffTest // добавление данных в контекст
            {
                Date = protoItem
            });
        }
        _db.SaveChanges(); // сохранение изменений
```

Рисунок 5

Анализ форматов передачи данных показал, таблица Galileo весом в 1,5 ГБ была сериализирована в JSON таблицу весом 1,3 ГБ, тогда как после сериализация в ProtoBuf она заняла всего 60 МБ. Данный тест показал, что ProtoBuf даже с тестовой выборкой занимает в 8 раз меньше объема на жёстком диске сервера.

Таким образом, можно сделать вывод, что бинарный формат данных ProtoBuf занимает объем данных в 10-15 раз меньше чем строковый, тем самый выигрывает у него в скорости обработки данных. Следовательно, ProtoBuf незаменим в тех случаях, когда требуется хранить большой объем данных, без обращения к этим данным разработчиком.

Список литературы:

1. Galileosky [Электронный ресурс]. – https://7gis.ru/
2. Json.Net Popular high-performance JSON framework for .NET [Электронный ресурс]. – https://www.newtonsoft.com/json

3. Protocol Buffer Basics C# | Protocol Buffers | Google Developers [Электронный ресурс]. – https://developers.google.com/protocol-buffers/docs/csharptutorial

4. Использование Protocol Buffers на платформе .Net (Часть 1) / Хабр [Электронный ресурс]. – https://m.habr.com/ru/post/119503/

5. Использование Protocol Buffers на платформе .Net (Часть 2) / Хабр [Электронный ресурс]. – https://m.habr.com/ru/post/119510/