FlexIT: Not just another JSON Parser

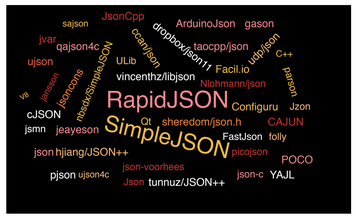
Nipun Jindal, Pranay Kumar

Adobe Inc.

cppcon the c++ conference
SEPTEMBER 15-20 2019
Aurora, Colorado, USA

Problem

JSON is the de facto for transmitting data, so picking the right library is critical.
Requirements vary in terms of the maximum message size handling, parsing times, parsing memory, in-memory usage, access times, with that comes the tradeoff of each library. None of the solutions seem to be optimized for low end memory devices.



Related Work / Motivation

We were motivated to tackle the problem of high memory usage throughout the application cycle even when the deserialized JSON is not in use, considering we were tackling mobile devices. Also, we wanted to have improved access times.
Current solutions like ProtoBuf, FlexBuff provide strong typed solutions, which requires schema deductions & generation investment each time JSON changes. A ready to go library which could help a developer bridge those was something we solved.

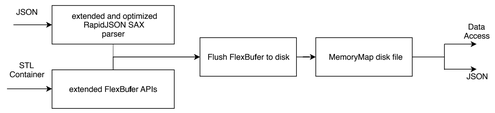
Solution

FlexIT is a highly customizable library to support JSON parsing and provide containers based on Google's FlexBuff which will be memory mapped just using FlexIT. Without developer learning curve, you gain low memory and access time with JSON.

Pipeline and APIs

FlexIT uses a modified RapidJson to construct flexbuffer, highly aligned/packed binary structure & provides random data access. This is flushed to disk and memory mapped using FlexIT API(s) thus utilizing OS mmap-ing, memory friendly-ness, and still be efficient. It is a simple and direct way available to reduce memory usage with better access times.

Flow Diagram



Deserialization

```
using namespace flexbuffers;
using namespace flexit;

if (auto reference = JsonToFlexBuffer("/tmp/.memoryMappedFile", test_string_valid)) {
    if (reference->IsMap()) {
        auto lookUpMap = reference->AsMap();
        log(lookUpMap["item3"].AsString().str());
    }
}
```

```
[TestLoadFromFile] An Item
Program ended with exit code: 1
```

Serialization

```
std::string jsonEncodedString;
if (FlexBufferToJson(reference, jsonEncodedString)) {
    log(jsonEncodedString);
}
```

```
[TestLoadFromFile] { "": [  ], "item2": { "a": 1, "b": 2, "c": 3 }, "item3": "An Item", "item4": "Another
   Item", "item5": "An Item", "item6": "Another Item" }
Program ended with exit code: 1
```

Results

In our benchmarks, FlexIT proved to be better than RapidJson in terms of data access, serialization, deserialization and enables a way to reduce memory usage. That's what you always wanted from a JSON Parser, right?
Currently, we are working on further optimizing FlexIT to support loading partial memory pages into memory, since it is aware of the offset of required key/value pair. This would further reduce the memory usage during access.

Current Solutions:

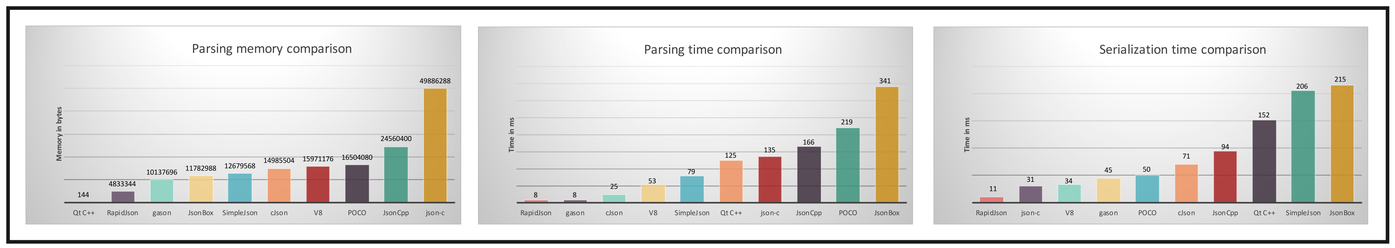


Fig: Performance comparison across available libraries. RapidJSON does fair across the board.
*source: https://github.com/miloyip/nativejson-benchmark

Our Solution:

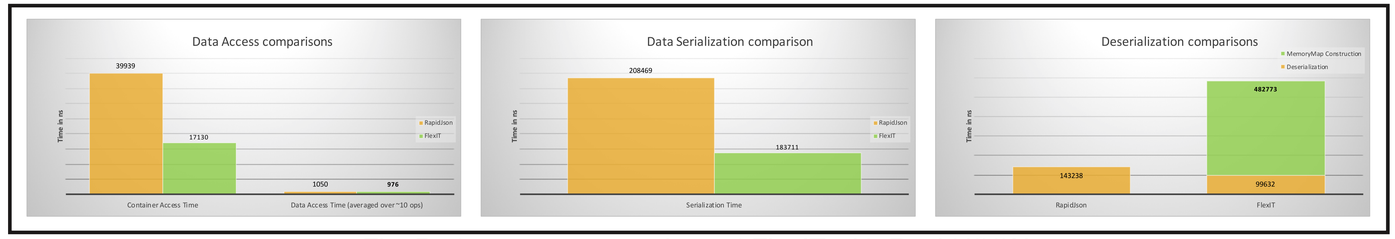


Fig: Performance comparison of FlexIT with RapidJSON.
*MacBook Pro (Retina, 15-inch, Mid 2015) Processor 2.5 GHz Intel Core i7 Memory 16 GB 1600 MHz DDR3

References

1. Rapidjson performance benchmarking and comparison: https://github.com/miloyip/nativejson-benchmark
2. Flexbuffers https://google.github.io/flatbuffers/flexbuffers.html