Performance Guide

Sevki

Sat Apr 14 20:30:31 CES 2018

Abstract

This is a guide handy guide for improving your applications performance.

What causes slow performance?

Slow performance can be caused by a variety of things from connection speeds to data encoding.

Data Serialization

Assumptions: We are going to assume we already know what the message looks like, and we'll only look in to how to decode a string, which should illustrate the performance drawbacks of parsing data structures.

Let's take a very simple message and see it's faximile over the wire; here is what it would look like in everyone's favourite data markup language:

Figure 1: JSON

This is what it would look like in a `protocol buffers` by far the most popular binary serialization format

So how would we decode the a string in a JSON message? (this is a super simplified, rough estimation)

rougly this is what this code is doing:

- first byte we read is a `"` so we'll be recieving a string literal in next char, we'll make a note of that by pushing the address to the `stringStack`
- we read the next char and increment the `bytesRead` counter
- when we hit another `"`, we pop the address from the stack

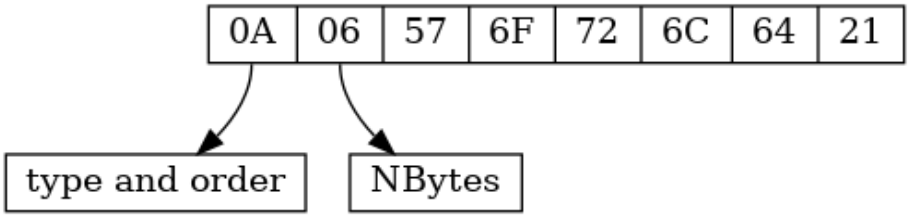

Figure 2: Protocol Buffers

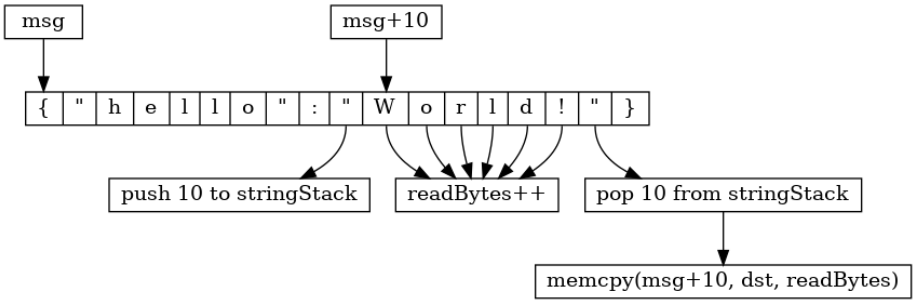

Figure 3: JSON Decoding

- `memcpy` the string

All we are doing here is reading a **VERY** simple `ascii` string and copying it from the network buffer, this example doesn't include how to corner cases like, malformed json, handling of unicode characters, \ escape sequences and so on, as you start handling those cases the code becomes a lot more costly.

In contrast here is what decoding `protocol buffers` look like;

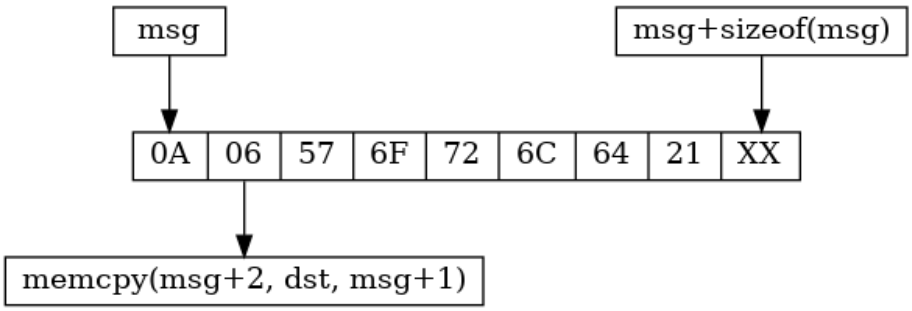

Figure 4: Protocol Buffers Decoding

- `memcpy` by passing the 2nd byte as the length argument

`read(2)`

Let's assume, we have a proxy that takes a message and dispatches it to another port on the same machine. We'll modify our message so we know which port to forward our message: `{"port": 80,"hello": "World!"}`.

The message it self is contrived but the reality isn't that further off from a real world scenario, change the `port` to `host` and that's roughly how `http` requests are routed.

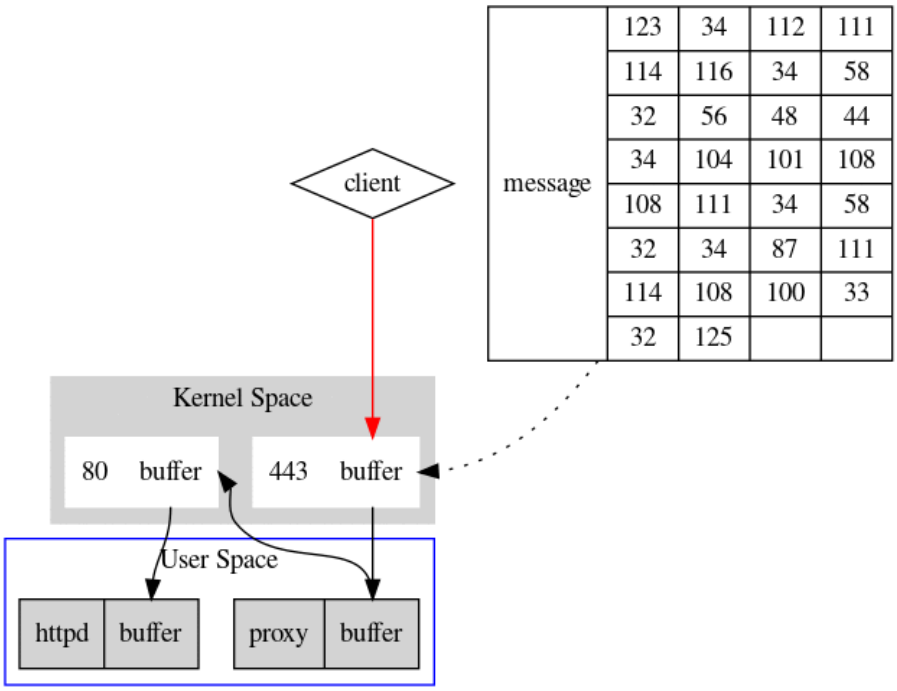

Figure 5: calls

The animation above has two distinct domains called `user space` to `kernel space`. The reason why that distinction is emphasized in the animation is everytime you `read(2)` or `write(2)` the ammount of bytes you've processed will cross from one domain to the other.

`sendfile(2)`

Let's look at the example below; first bit is the order and type, second byte `50` is the port number. After reading just 2 bytes, we can lookup which file descriptor corresponds to which port and use `sendfile` to proxy the rest of the message to

the approprite `fd` without having to read the entire message. That saves the hassle of not having to copy the same message from `kernelspace` to `userspace` and back again.

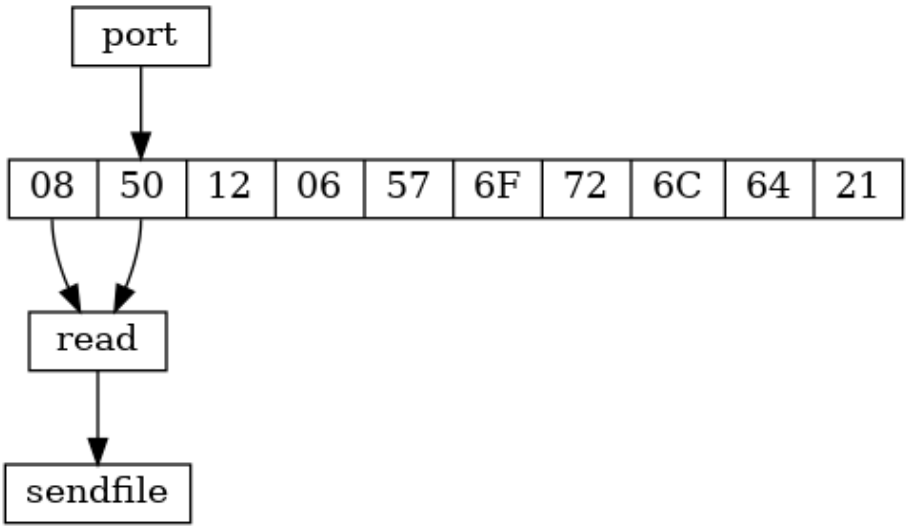

Figure 6: Protocol Buffers Sendfile

Wrapping up

Binary serialized data gives us constant read times `O(1)`, where as delimited formats like `json`, `xml` or `csv` you'd have to read either the entire message or a substantial portion of it.

As `HTTP/2` spec puts it

> Binary protocols are more efficient to parse, more compact "on the wire", and most importantly, they are much less error-prone, compared to textual protocols like HTTP/1.x, because they often have a number of affordances to "help" with things like whitespace handling, capitalization, line endings, blank lines and so on.

	Pros	Cons
Binary (pb,capnproto)	• `O(1)` read time	• needs schema definition to read
Delimited (json, xml)	• self describing	• `O(n)` read time

Slow connection

let's put some context around the Binary Serialization vs Delimited Serialization overhead; these are approximate timing for various operations on a typical PC (source:[1] originally:[2])

Here are some tools to simulate network latency issues - http://blogofsomeguy.com/a/2018-03-11/lossylink.html

[1] Latency Numbers Every Developer Should Know
[2] Teach Yourself Programming in Ten Years