Oleg Stotsky

@e_boy_coding

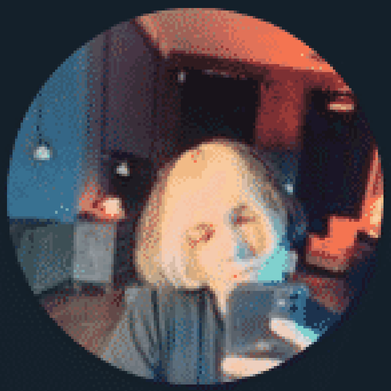
I’ve worked on numerous Highload Golang application that served more than 100 000+ requests per second. Here are 3 patterns that I’ve encountered the most in this code bases:

8:00 AM · Feb 17, 2023

Oleg Stotsky
@e_boy_coding

1\ 1.Reducing allocations.
Reducing the average allocation per request to zero is a key goal in highly optimized Go code.
Allocations can be costly in terms of CPU cycles during garbage collection and can increase latency of requests due to the STW pause of the Garbage Collector.

8:00 AM · Feb 17, 2023



Oleg Stotsky
@e_boy_coding

2\ To achieve this, stack and memory pools can be used. When possible, values should be allocated on the stack. For example, when creating something with a literal value (e.g. x := 5), it will be allocated on the stack.

8:00 AM · Feb 17, 2023

Oleg Stotsky
@e_boy_coding

3\ This value can then be passed down the stack when making function calls (e.g. foo(&x)) and will remain on the stack. Values escape to the heap when they are passed "up" the stack (i.e. returned from functions) or passed to a different goroutine.

8:00 AM · Feb 17, 2023

Oleg Stotsky
@e_boy_coding

4\ In cases where the size of the value is not known in advance, it must be allocated on the heap. Examples of values that always live on the heap are strings, slices, and interfaces. In this case, the sync package's Pool data structure can be used to cache the allocated value,

8:00 AM · Feb 17, 2023

Oleg Stotsky
@e_boy_coding

5\ so that it can be reused the next time an allocation is needed. For more information on buffer pools, please refer to buff.ly/3Igocug . For more on escape analysis, see

Go: The Idea Behind Sync.Pool
medium.com

8:00 AM · Feb 17, 2023

Oleg Stotsky
@e_boy_coding

6\ 2.Use concrete types instead of interfaces. Interfaces are great for code reuse, but they can be quite expensive and can cost you a lot of CPU cycles when used in high frequency operations. When you are dealing with lots of requests, you should always use concrete types.

8:01 AM · Feb 17, 2023

Oleg Stotsky
@e_boy_coding

7\ When using concrete types, you will be able to avoid type assertions and type switches, which are usually expensive operations. You can also take advantage of inlining, which can significantly reduce the total number of CPU cycles required to process the request.

8:01 AM · Feb 17, 2023

Oleg Stotsky
@e_boy_coding

8\ 3.Optimize the database access - Database access is often ne of the most expensive operations when dealing with large number of requests. To optimize the database access, you should make sure that you are using indexes and your queries are efficent.

8:01 AM · Feb 17, 2023

Oleg Stotsky
@e_boy_coding

9\You should also consider caching the frequently accessed data in order to reduce the number of trips to the database. You can use in-memory caches like Redis, Memcached or even something like sync Map to store the frequently accessed data.

8:01 AM · Feb 17, 2023