Start learning Stickie after finishing profiling and optimization of programs last week and this week (week 5 and week 6)

I will focus on StackImpact next week

**Overview**

StackImpact is a performance profiling and monitoring service for production Go applications. It gives developers continuous visibility with line-of-code precision into application performance, such as CPU, memory and I/O hot spots as well execution bottlenecks, allowing developers to optimize applications and troubleshoot issues before they impact customers. Learn more at [stackimpact.com](https://stackimpact.com/).

**Features**

* Automatic hot spot profiling for CPU, memory allocations, network, system calls and lock contention.
* Automatic bottleneck tracing for HTTP handlers and HTTP clients.
* Error and panic monitoring.
* Health monitoring including CPU, memory, garbage collection and other runtime metrics.
* Alerts on hot spot anomalies.
* Multiple account users for team collaboration.

Let’s play with it.

agent := stackimpact.NewAgent();

agent.Start(stackimpact.Options{

AgentKey: "d7400ebf4d95f2b0f49c25f829336044c4ff0cb7",

AppName: "MyGoApp",

})

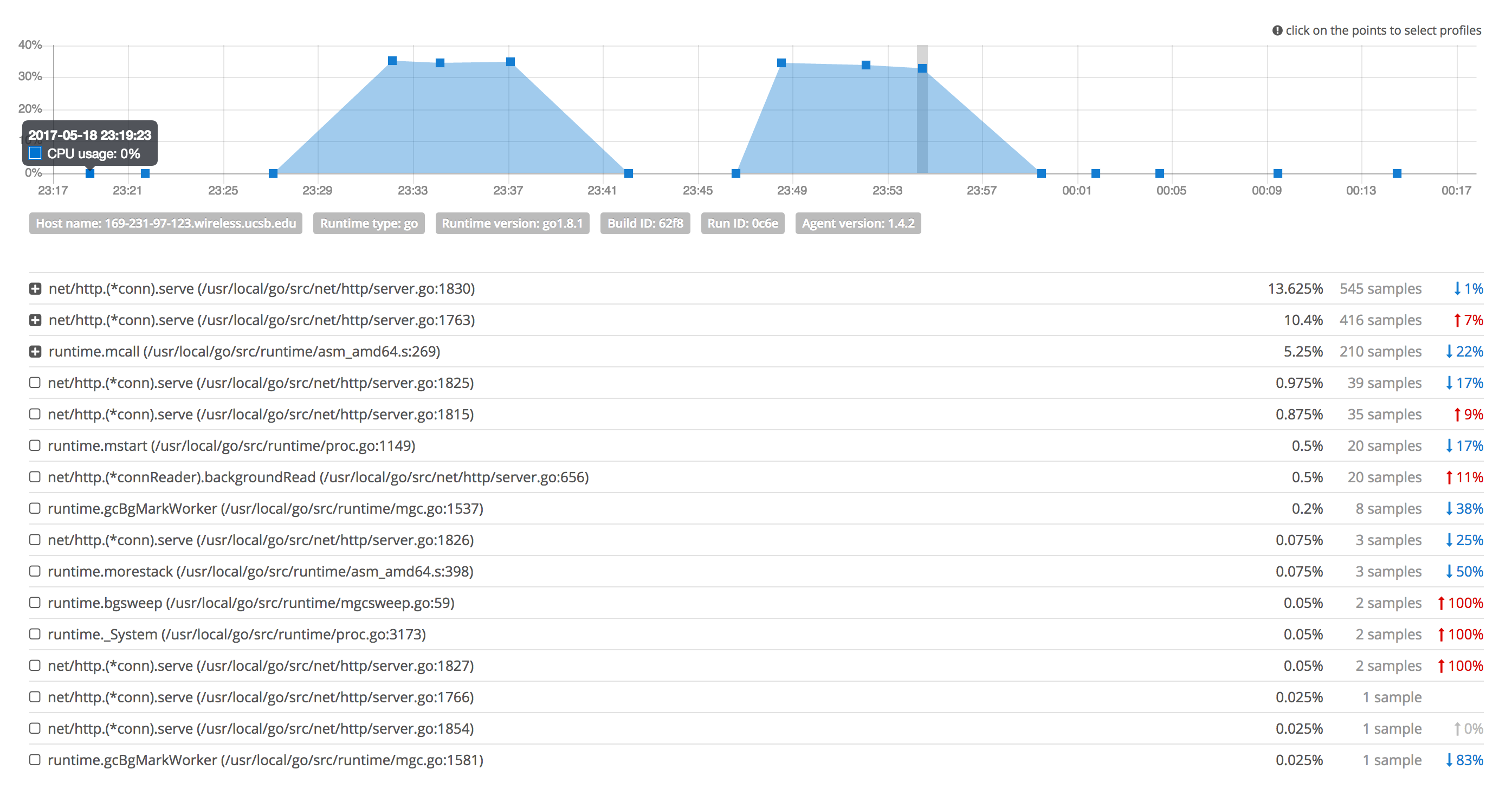
After creating the account, we get this AgentKey in our configuration. Using stackImpact is easy, according to the official website, by simply add these lines of code in the code. Lets start with the basic http request used in the second week: showing helloworld in browser.

The code is in testServer.go. Good news is that when we add this key to the program, it can be seen on the screen. Since there is nothing in the code but just one request handler, there is nothing shown in the dashboard.

Let’s try out previous code in week5.

One conclusion we can draw:

StackImpact is not a good tool for profiling small size code (maybe it’s better for profiling App as well as other programs that requires dynamic profiling). However, it is a great tool to find out the result when we use the pprof.



We can see that after doing some modification(using pprof), the CPU profile dashboard shows that the program is indeed doing a better performance than the previous version(from 23:49 -> 23:52 -> 23:54, the program stop after the line I marked out in the picture so it drop down immediately).

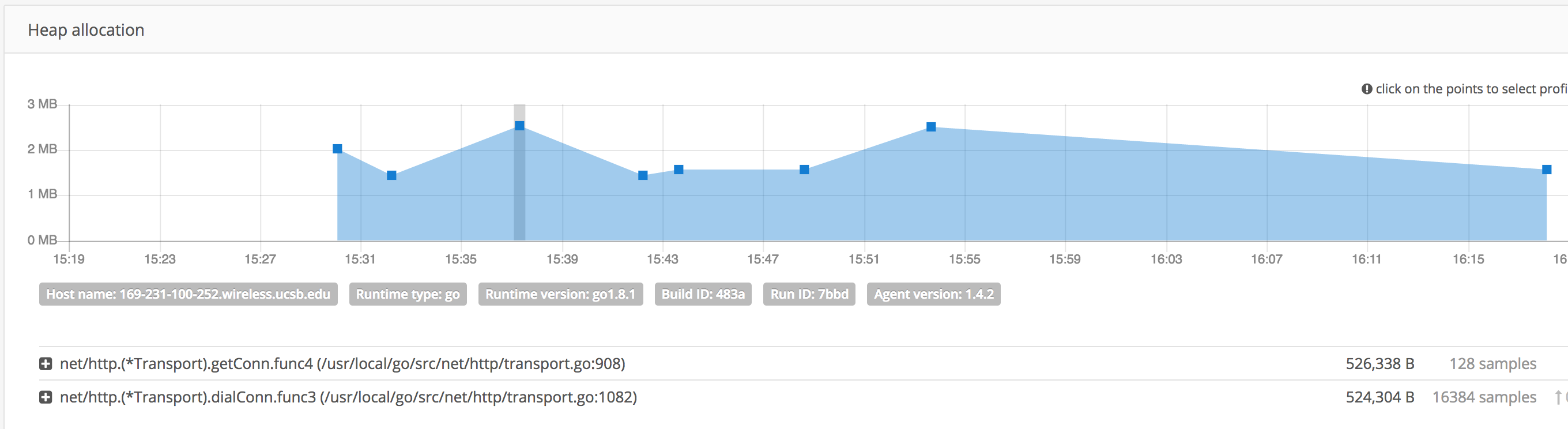
After discussed with professor and researched stackImpact for a week, I believe that stackImpact is not a better tool for profiling Go program than pprof. However, we can combine this two tools together to make profiling easier for both programmer and reader.

Let me demonstrate and prove my idea with a easy way:

I write a really basic http request program, I use:

go-wrk -d 300 http://localhost:8080/hello

to run it for 5 mins, and then check the dashboard on stackImpact:



We can see that in memory profiling dashboard, it shows that we use the func3 and func4 (that are all http library). The func3 include handshake, verify, loadSystem etc. The advantage of using stackImpact is that it can show the improvement or retrogression of the programs, as well as the whole hierarchy structure of the program. It gives a clear structure of how the program is called and run. However, it didn’t show the program I wrote (for example in main.go). As a result, even if I know there might have some latency in my program that need to improve, there is no way for me to find which one is the line I need to improve, since there may have lots of functions that call these libraries. As I shown in the previous pictures, all the function that listed on the dashboard are all library functions. The name of the function is totally different than the name shown in pprof. As a result, it is a good tool to see the overall performance, but it’s not a good tool to actually do the profiling

Using pprof, however, is different and controllable. For example, after running the naïve program, we can see the top10 list and after checking the svg file, we found out that getStatsTags

slows the program down. We go deeper and check what getStatesTags do, and find out that it calls os.Hostname(), and we then realized that it creates a hostname every time it executed. We then cache this hostname since it’s same for one request. We store the hostname as a global variable and the performance of the program is indeed better after the cache.

Conclusion for week7:

Based on all the above experiment and research, we should use stackImpect as a tool to check the overall performance, but use pprof to acutally profile and improve the program. We used couple days to research and realized that stackImpect is powerful(maybe), but is not a helpful tool for our project. We can use it as a “monitor” though.

TODO:

Start writing the report and ppt for the presentation.

Need more profiling data, especially using pprof.

Keep doing the go profile, but should not focus on stackImpect anymore.