Measuring Context Switches in Serverless Environments

Elaine Yao
elainedv111@gmail.com

1 Introduction

Serverless computing is a new type of cloud application model. Unlike traditional cloud computing, users only need to upload the function code into the cloud and the resources needed are scheduled by the provider. However, user's functions can't be executed continuously due to limited resource. Providers want to invoke more functions at the same hardware to achieve the low resource cost. When resources like memory or CPU is limited, a running function may be interrupted to let another function being executed. Thus, the function execution time(CPU wall-clock time) is larger than the actual execution time, which is presented in previous studies. [7] For users, who are charged with the function execution time, this phenomenon makes them charged more than they should be. One of the main extra time is spent in context switch [7].

Context switch [3] refers to the situation when the operating system interrupts the current execution and switch it off to another task. In Linux systems, this happens due to multithreading/processing or timer interruption. There are various studies [2, 4–6] proposing benchmarks for measuring the context switch time in Linux systems. However, these benchmarks can't be directly used due to the programming languages used are not supported in current cloud environment. Also, there are other factors that may influence the number of the context switch and the context switch time in serverless environment, as the latter supports configurations like the memory allocated.

Therefore, measuring the context switch time can reduce the user's cost in the future by reporting the extra time to the cloud provider. The characteristics of the context switch in the cloud are different from traditional Linux systems and also the current benchmarks can't be used directly. In this work, we aim to answer the following research questions:

1. How to measure the context switch time accurately in a serverless environment?

2. What's the factors that may influence the number and time of the context switch in a serverless environment?

2 Methodology

2.1 Challenges

Measuring the context switch time in serverless functions has several challenges:

C1 Characteristics of context switch in serverless environments Context switch is triggered differently in serverless functions compared to in traditional operating systems as it's scheduled by the cloud provider. And the service provider has the tendency to trigger more functions in the same hardware to get more profits. Thus, context switch may happen not only due to performance factors like multi-threading/processing, but also provider's profit considerations.

C2 Benchmark accuracy As no such benchmark is provided so far, we have to analyze the factors in our own benchmark that may lead to the potential variation to golden value, in order to reason about the accuracy of the measurement.

2.2 Factors in serverless environment

To tackle *C1*, we design the following experiments to analyze the factors influencing context switch in serverless environments. We measure the execution time elapse between two adjacent lines of code repeatedly under different configurations in serverless functions. Shahrad *et al.* [7] shows that the execution time in serverless functions is influenced by function invocation frequency, memory size and function execution time. Thus, these configurations will include the function invocation rate, memory allocated. Theoretically, the measured time elapse should be 0 as there is no execution between these two lines. However, our initial experiments show that this is neither 0 nor a fixed value, indicating that interruption happens between these two lines and the time elapse changes with different configurations. Although the interruption is not limited to context switch but also page

faults handling or container management, it can still provide insights in the influence of different configurations on context switch time in serverless environments.

2.3 Combine various benchmarks

To tackle *C2*, we first run our own designed benchmarks on local Linux system and compare the results with published benchmarks [1, 2, 5, 6], ensuring the correctness of the proposed ones. Then we deploy the new benchmarks in serverless functions and compare the results. If one of the benchmarks is consistently higher or lower than others, then we'll explore the reasons and this can help us improve our accuracy.

The previous benchmarks in context switching on Linux system mainly consider Our initial study to *C1* finds that context switch in serverless environments will also be influenced by invocation rate and memory allocation.

Currently, the new benchmarks are designed as the prior benchmarks with different configurations. The prior benchmarks we are choosing are:

1. Pingpong pipes [1, 6]: Two threads or processes are created. One thread/process writes the data to another process through a pipe, which induces a context switch. After the data is read, the read thread/process transfers the data to write thread/process and this induces another context switch. The total execution time equals the read and write operation time adds two context switch time. In contrast, a single process is created. It pass the data through a pipe to it self. Here the total execution time only consists of the read and write opration time. By substracting these two time and dividing the result with 2, we get the context switch time.

2. Condition var [1]: A shared variable between two threads is used and is protected by mutex to prevent it from being modified by two threads at the same time. A signal is passed between two threads to tell which thread has the access to write the variable. And by passing the signals, the context will switch from one thread to the other. The time for the signal changes is the context switch time.

3. Lmbench [5]: A ring of pipes is created and a token is passed from process to process with thses pipes. Also, the time of passing the token in a single process with this ring of pipes is measured. Substraction of these two time is the total context switch time of a number of context switches.

Next, we'll implement these functions with different invocation rate and memory allocation setting in the cloud, to measure the influence of configurations on the context switch time.

3 Experiments

3.1 Experiment setup

We implement the benchmarks on Google Cloud, which is a mainstream cloud platform to provide serverless functions.

Benchmark	Pingpong	Condition Var	Lmbench	Pipe(python)
Time(/us)	1.96	2.29	1.78	33.6

Table 1: Context switch time by different benchmarks

For local measurements, we use an Ubuntu 20.04-64bit machine with Intel(R) Core(TM) i7-8750H CPU @ 2.20GHz processor and 12GB RAM. We plan to do experiments in more types of machines and operating systems to show the generality of the benchmark.

3.2 Measuring local context switches

All the previous benchmarks [1, 4–6] are implemented in C language. However, currently Google Cloud only supports functions written in Node.js, Go, Java, Python and Ruby. Considering that Python is one of the most popular languages, rewriting the prior benchmarks in Python and examining its correctness in local PC is important.

The context switch time measured by the first three benchmarks in local PC is usually among 2us from Table 1. In the first row, pipe, condition var and lmbench refers to methods listed in Section 2 respectively. However, when it's rewritten by Python, the measured time is 10 times larger than previous results. The reason might be that extra execution is added to the time measurement when translating the language. What's more, the C programs execute faster as compiled programs while Python executes slower due its interpreted programs, and these extra execution time is more obvious when using python. Therefore, we plan to measure the time variation induced by Python on local computers and scale the measured time in cloud accordingly. Table 1 shows that the context switch time measured by the pipe method written in Python is about 17.14 times larger than measuring in C programs. In our further experiment in cloud, we'll take the python measured time divided by 17.14 as the context switch time.

3.3 Measuring context switches in the cloud

We create functions with different memory allocated and measure the context switch time with the Pingpong pipes method. For each memory configuration we run 10 times and get the average of the calculated time. We also notion that the variation induced by Python language has already been considered and the value shown in Table 2 is processed. We notice that if the allocate memory is 128GB, the context switch time is remarkably higher than the other configurations. And we plan to look deeper into this in the future.

Memory(/MB)	128	256	512
Time(/us)	81.11	33.96	36.17

Table 2: Context switch time in functions with different memories

4 Future work

We plan to look at the following problems in the future.

1. Implement condition variable and lmbench in python and deploy the function in the cloud. We aim to measure the context switches in different ways in order to improve its accuracy.
2. Explore the factors that influence the number and the time of the context switch in serverless environment.

5 Related Work

A few studies [2, 4–6] have proposed benchmarks for measuring context switch in Linux system. Ousterhout *et al.* [6] proposes using two pipes to ping pong data between two processes. McVoy *et al.* [5] reduces the cost of system calls for context switch by designing a ring of pipes, and they also consider the influence of data size and cache size on the context switch time [4]. Suo *et al.* [8] measures the context switch on the edge in AI workloads. None of them considers the context switch in a serverless environment.

References

[1] Measuring context switching and memory overheads for linux threads. https://eli.thegreenplace.net/2018/measuring-context-switching-and-memory-overheads-for-linux-threads/. Accessed: 2022-2-25.

[2] Francis M. David, Jeffrey C. Carlyle, and Roy H. Campbell. Context switch overheads for linux on arm platforms. In *Proceedings of the 2007 Workshop on Experimental Computer Science*, ExpCS '07, page 3–es, New York, NY, USA, 2007. Association for Computing Machinery.

[3] David Kalinsky. Context switch. *Embedded Systems Programming*, 14(1):94–105, 2001.

[4] Chuanpeng Li, Chen Ding, and Kai Shen. Quantifying the cost of context switch. In *Proceedings of the 2007 Workshop on Experimental Computer Science*, ExpCS '07, page 2–es, New York, NY, USA, 2007. Association for Computing Machinery.

[5] Larry W. McVoy and Carl Staelin. lmbench: Portable tools for performance analysis. In *USENIX Annual Technical Conference*, 1996.

[6] John K. Ousterhout. Why aren't operating systems getting faster as fast as hardware? In *USENIX Summer*, 1990.

[7] Mohammad Shahrad, Jonathan Balkind, and David Wentzlaff. Architectural implications of function-as-a-service computing. *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2019.

[8] Kun Suo, Yong Shi, Chih-Cheng Hung, and Patrick Bobbie. *Quantifying Context Switch Overhead of Artificial Intelligence Workloads on the Cloud and Edges*, page 1182–1189. Association for Computing Machinery, New York, NY, USA, 2021.