Memory Usage Interfaces for Serverless Functions

Yonghao Zou (Student)
EPFL
yonghao.zou@epfl.ch

David Hua (Student)
University of Waterloo
david.hua@uwaterloo.ca

George Candea
EPFL
george.candea@epfl.ch

FaaS platforms like AWS Lambda, Microsoft Azure functions, and Google Cloud Functions have become popular for building serverless applications. They offer a pay-as-you-go model, automatic scaling, and high availability. However, they also suffer from various limitations. For example, they cannot accurately predict the memory usage of a function because they lack knowledge of their memory requirements. As a result, functions may consume more or less memory than expected, which would cause memory wasting that increases the cost to vendors or lead to performance issues such as inefficient memory usage and even out-of-memory errors, which would disappoint users.

On the other side, the cost of memory is a significant part of the total cost of running serverless functions. For example, AWS Lambda charges up to $0.06 per GB-hour of memory used. Users still have to worry about the low-level details of memory usage to calculate and optimize the cost of running functions. They need to choose the right amount of memory for a function to balance the cost and performance. However, the memory usage of a function is not directly exposed to users, and they have to estimate the memory usage of a function based on profiling or testing.

For example, OFC [1], an opportunistic caching system for FaaS platforms, uses machine learning to predict the memory usage of functions to identify any overbooked memory and use it to cache data and improve the performance of functions. However, the machine learning model OFC uses is based on the memory usage of functions collected from profiling, which might not be accurate and may lead to performance issues. It also needs to be updated frequently to adapt to different and changing workloads.

To ease the pain of worrying about low-level memory usage details, a cost calculator [2] was proposed to simulate the execution in FaaS and calculate possible costs based on the simulated memory usage. However, the cost calculator is based on profiling and a community-contributed dataset, which may not be accurate and up-to-date.

To solve these problems, inspired by performance interfaces [3, 4], we propose *memory usage interfaces* as a way to describe the memory usage of functions. We are developing a tool called MUX (**M**emory **U**sage e**X**tractor) that can automatically extract memory usage interfaces from a serverless function's source code using symbolic execution. MUX outputs memory usage interfaces in both a human-readable and machine-readable format, e.g., an annotation on a Python function or a header file for a C function with additional keywords.

```python
def matmul(n):
    A = np.random.rand(n, n)
    B = np.random.rand(n, n)
    start = time()
    C = np.matmul(A, B)
    latency = time() - start
    printf("Latency: %f" % latency)
    return C
```

For example, the above code snippet shows a Python function retrived from a FaaS benchmark that returns a matrix calculated by two random matrices. The memory usage of the matrices in `matmul` is a function of n: $3 \times sizeof(float) \times n^2$. The `A` and `B` matrices can be freed after the function exits. There are also two integers for calculating the lantency, each one consuming 28 bytes in Python. So, the memory usage interfaces of the function would be extracted as follows:

```python
@mu.consumed(3 * sizeof(float) * n * n + 56)
@mu.freed(2 * sizeof(float) * n * n + 56)
def matmul(n):
```

Memory usage interfaces can be used in two ways. First, they can be used by the FaaS platform to predict the memory usage of a function and enforce proper memory limits, which can prevent the function from consuming more memory than expected. We provide meta-programming APIs for platform developers to query the memory usage, including the consumed memory and the freed memory, of a function for now. Second, they can be used by the users to estimate the memory usage of a function and choose the right amount of memory to allocate to balance cost and performance. We are currently working on the implementation of MUX and evaluating it on real-world serverless functions. We also plan to apply similar techniques to other domains, such as deep learning models, far memory systems, and distributed systems.

References

[1] D. Mvondo, M. Bacou, K. Nguetchouang, L. Ngale, S. Pouget, J. Kouam, R. Lachaize, J. Hwang, T. Wood, D. Hagimont, N. De Palma, B. Batchakui, and A. Tchana, "OFC: an opportunistic caching system for FaaS platforms," in *Proceedings of the 2021 European Conference on Computer Systems (EuroSys)*, 2021.

[2] J. Spillner, "Resource Management for Cloud Functions with Memory Tracing, Profiling and Autotuning," in *Proceedings of the 2020 International Workshop on Serverless Computing*, 2021.

[3] R. Iyer, K. Argyraki, and G. Candea, "Performance Interfaces for Network Functions," in *Proceedings of the 2022 USENIX Symposium on Networked Systems Design and Implementation (OSDI)*, 2022.

[4] J. Ma, R. Iyer, S. Kashani, M. Emami, T. Bourgeat, and G. Candea, "Performance interfaces for hardware accelerators," in *Proceedings of the 2024 USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2024.

Memory-Usage Interfaces and Use Cases

EPFL

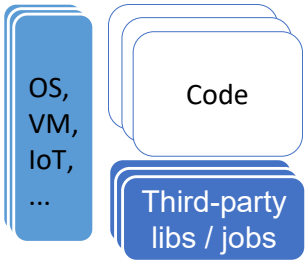

Yonghao Zou

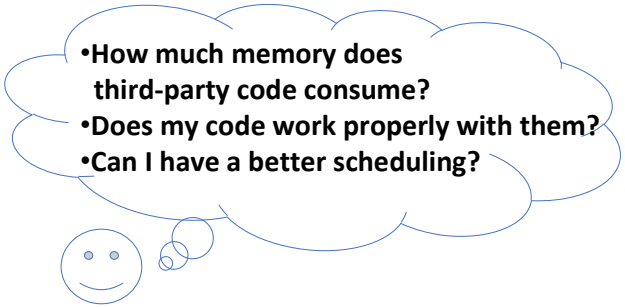

David Hua


George Candea

DSLab

There is no formal syntax for expressing memory usage
- Developers must implement and check memory usage carefully
- Third-party vendors cannot write formal memory usage documents
- Systems cannot predict memory usage of programs before they run





There are interfaces describing how to call the function properly, like the declaration with the function's parameters. Can we have memory-usage interfaces?

Functional interface

```
void *malloc(size);

void free(void *ptr);
```

Memory-usage interface

```
[alloc size]
void *malloc(size);
[free resource_of(ptr)]
void free(void *ptr);
```

```
int large_mem_func(n) {
 size = sizeof(..) * n;
 b = malloc(size, ..); ...
}
```

```
[alloc sizeof(..) * n]
int large_mem_func(n);
```

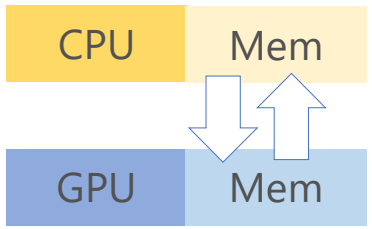
```
def matmul (n):
  A = np.random.rand(n, n)
  B = np.random.rand (n, n)
  C = np. matmul (A, B)
```

```
@consumed(3*sizeof(..)*n*n)
@freed(2*sizeof(..)*n*n)
def matmul(n);
```

With the memory usage interface
- Developers have a clear picture of memory usage for their code
- Third-party vendors can write formal memory usage documents
- System designers can utilize useful information to enrich system functionalities

Use cases

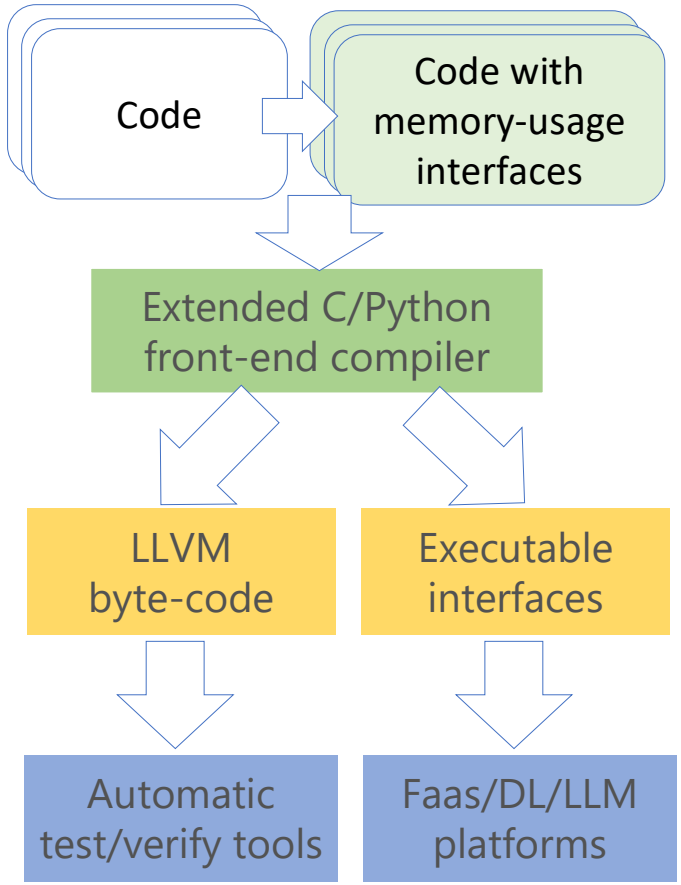
Serverless Functions scheduling



DL/LLM memory allocation

```
[alloc n]
int alloc_page(n);
Verifying...
Error: usage n is
not equal to n + 1
```

Verification



Want to work on something related? Talk to us!