
Wonseok Shin\textsuperscript{1}, Wook-Hee Kim\textsuperscript{2}, Changwoo Min\textsuperscript{3}

\textsuperscript{1}SK Telecom, \textsuperscript{2}Konkuk University, \textsuperscript{3}Virginia Tech
Here is Serverless Computing!

- Serverless computing is now mainstream in the cloud era
- Amazon Lambda, Microsoft Azure, Google Cloud, IBM Cloud Functions provide their serverless computing models
- Developers **do not need to effort** into administration
- It offers significant **scalability** for the resource provisioning
- The serverless introduces **pay-as-you-go**
What is Serverless Computing?

- Application can consist of small serverless functions
- Serverless computing is Function-as-a-Service (FaaS)
What is Serverless Computing?

- Inside the serverless, each function runs **in its own sandbox**
- **Cloud operator** manages functions flexibly for users
- In function, most used languages and runtime (90%) is **interpreter language**

![Diagram showing the components of serverless computing including Hypervisor, Operating System, and Server, with a pie chart showing the languages used with Python being the most used at 52.9%]

*serverless benchmark report AWS lambda 2020*
Characteristics of Serverless Computing

- **On-demand execution**
  - The resources for functions are created on-demand when invoked

- **Shorter execution time**
  - Functions on the serverless run in a short span

- Cloud operators aim to **consolidate a large number of serverless functions** in a few machines to utilize server’s resources more efficiently
Problems

(1) Long startup time penalty
   ● Long booting time of VM, OS, container and runtime
   ● 1st request = cold start, subsequent requests = warm start (can be expired)
   ● In some cases, startup time > execution time of function

(2) Unpredictability in Just-In-Time (JIT) compilation of interpreter language
   ● Short execution time of the function is not suitable for JIT compilation

(3) Memory is bottleneck in the consolidation
   ● Cloud operators retain the functions in memory for a while waiting

(4) Security challenge
   ● High consolidated serverless environment needs robust isolation
Our approach

- Low-latency start up time
  - Snapshot

- Fast execution time
  - Post-JIT
  - Snapshot

- Memory efficiency
  - Sharing
  - Post-JIT
  - Snapshot

- Robust isolation
  - Sharing
  - VM-level
  - Post-JIT
  - Snapshot
Our approach: VM-level post-JIT snapshot

**Installation phase**: create a memory snapshot with JIT compilation

**Invocation phase**: load the memory snapshot on a sandbox as a new function

**Installation phase**

- User code
  - JIT Compilation
  - Machine code

**Invocation phase**

- Create memory snapshot
- Load memory snapshot
Challenges

● Installation phase
  ○ How can we manipulate JIT compilation?
  ○ Do we know when creating a snapshot?

● Invocation phase
  ○ How we can reduce a memory footprint?
  ○ How to solve the problem of duplicated IP and MAC using the same snapshot?
  ○ How can we pass multiple users’ arguments to the same running state of a memory snapshot?
Our solution: FIREWORKS

We suggest FIREWORKS, a way to solve the challenges and realize a post-JIT snapshot.
How can we manipulate JIT compilation?

Use source code annotation

- Modern highly-optimized language runtimes already support **annotation to trigger JIT** at the program loading time.
- We re-purpose this feature to create a post-JIT VM-level snapshot

```
function optfn() {
    try {
        %OptimizeFunctionOnNextCall(main);
    }
    catch (e) {
        console.log(e);
    }
}
```

Node.js

```
@jit(cache=True)
def f(x, y):
    return x + y
```

Python
Do we know when to create a snapshot?

Place a trigger of making a snapshot in **user code**

- **Only a function knows** that the code is optimized and ready to run, rather than the environment that manages it

```python
import package

@jit(cache=True)
def main(params):
    Snapshot_request()
```
How we can reduce memory footprint?

**Redefine** Running a function as loading a memory snapshot file

- Sandbox #1 runs by loading the memory snapshot file
- Sandbox #2 can share the memory in a copy-on-write manner
How to solve the problem of duplication IP and MAC using the same snapshot?

Take advantage of MMDS, iptables NAT and network namespace

- MicroVM metadata service (MMDS) inserts unique information of the microVM
- **iptables NAT** map the same ip to different exposed IP
- conflict for the same device. ⇒ own network namespace.

![Diagram showing network setup and solving IP duplication problem]
How can we pass multiple users’ arguments to the same running state of a memory snapshot?

Create a repository that can be delivered between users and functions in the sandbox, and add logic to check it.
FIREWORKS

High isolation level

VM-level

Post-JIT

Fast execution

Snapshot

Low-latency start up

Memory efficiency

Code annotation

Argument passing

Request snapshot from a function

Multiple sandboxes from one memory snapshot
Implementation & Evaluation Methodology

Implementation

- Firecracker v0.24.0
- 3,000 lines of Bash code for microVM manager, Invoker, Code annotator
- 500 lines of Node.js, Python for Snapshotter, Parameter passer
- 40 lines of C++ for Node.js V8 Source
- 17,480 lines of Node.js, Python borrowed code with modification from FaasDom and SeverlessBench Benchmark

Evaluation Methodology

- **Faasdom Benchmark**\(^1\) represents basic performance in serverless
- **ServerlessBench**\(^2\) measures the real-world serverless application performance

\(^1\) [DEBS20] Faasdom: a benchmark suite for serverless computing
\(^2\) [SoCC20] Characterizing serverless platforms with serverlessbench.
How much can FIREWORKS reduce *start-up time and function execution time* of various serverless applications?

How effective are FIREWORKS’s *design choices (VM-level snapshot, post-JIT snapshot)* in improving performance and saving memory usage?

How much memory can Fireworks save by *sharing memory snapshots* across sandboxes?
Microbenchmark : Python

faas-fact

faas-matrix-mult

faas-diskio

faas-netlatency
Memory Usage

- FIREWORKS can make **565 sandboxes**, and Firecracker can make **337 sandboxes** without the swap memory.
- It allows FIREWORKS to consolidate **167%** more sandbox than Firecracker.
● Seeking a **safe, efficient, high performance** serverless framework continues.

● **FIREWORKS** can get all three:

- VM-level
- Post-JIT
- Snapshot

  - Safe
  - High performance
  - Efficiency

● We **designed, implemented, and evaluated new serverless framework** by using VM-level snapshots and JIT-based snapshots
  - 20 time shorter (cold) startup time, 7 times lower memory footprint
  - The achievements give a **guidance** to utilize **JIT** and **Snapshot** in the serverless computing.