Lightweight Preemptible Functions

Sol Boucher[†], Anuj Kalia[†], David G. Andersen[†], Michael Kaminsky[‡]
[†]Carnegie Mellon University, [‡]Intel Labs

DEFINITION: **A *lightweight preemptible function* is an unmodified function invoked with bounded execution time.**

GOAL: **Maintain low system-wide tail latency even when using code not trusted for timely completion.**

Use cases: Pause/cancel function calls

Timed function calls:

- Ensure reliable return from libraries with weak timing constraints (e.g., image decoders)
- Preempt straggler nodes in cluster fanout

Userland scheduling:

- Implement preemptive threading or coroutines
- Share a process with user code (e.g., FaaS) while maintaining control of the CPU core

Features: Preemption and cancellation

Building a library for timed function calls that provides:

- Preemption on 10s of microseconds timescales
- Resource tracking for cleanup upon cancellation
- An API that avoids abnormal control flow (e.g., repeated or absent function returns)

Invoke `lambda()` with a timeout using our API:

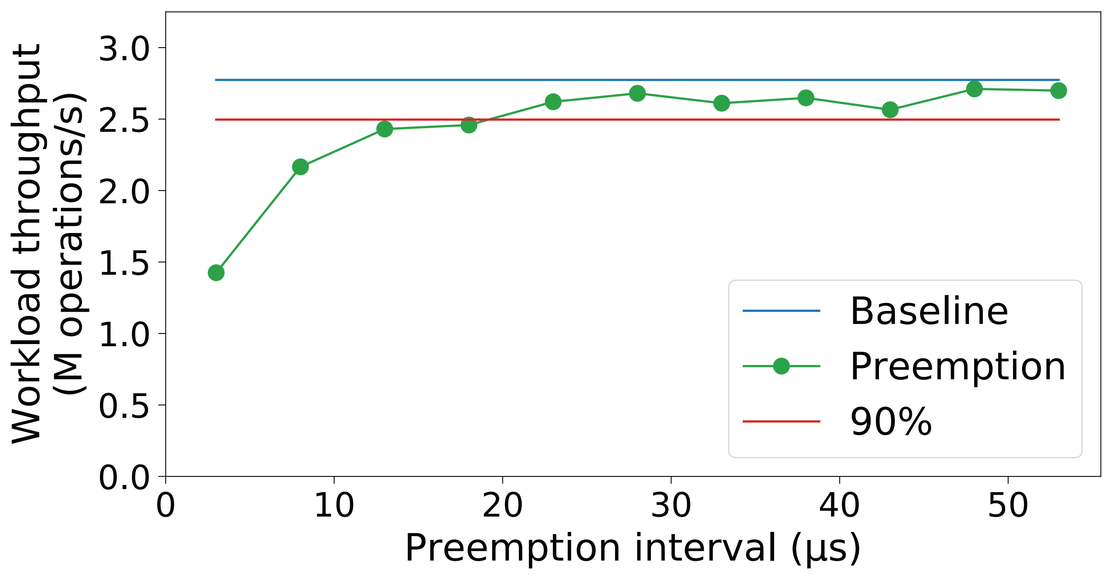
```
res = launch(lambda, TIMEOUT);
if(is_completion(res)) {
  // lambda() completed within the allotted time.
  print("Return value: " + unpack(res));
} else {
  // lambda() timed out; give it a bit more time.
  res = resume(res, TIMEOUT);
}
```

Implementing preemption and checkpoint/restore

Preemption: Use POSIX timers

Subscribe to SIGALRM to periodically regain CPU control.

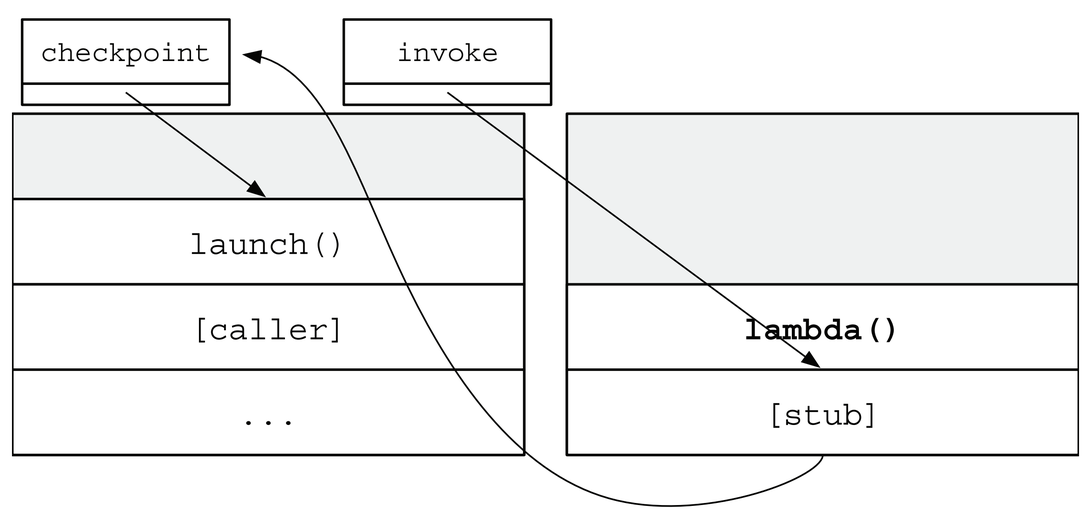
Achieve 90% of baseline SHA-256 throughput by 20µs.



Checkpoint/restore: Use POSIX contexts

Run a timed function on its own stack to support resumption.

Checkpointing and restoring takes about 3µs.



Challenge: Global state

Client code:

- Should not use shared state mutated by a preemptible function until that function completes

Library code:

Domain	Solution
C library functions	Interposing
Third-party libraries	Dynamic call rerouting

Dynamic call rerouting:

- Replace the global offset table during timed functions to retarget dynamic calls against a different GNU linker/loader namespace.
- Generating a 100-entry GOT takes ~60µs (installing takes 1µs)

Related publication

Sol Boucher, Anuj Kalia, David G. Andersen, and Michael Kaminsky. *Putting the "micro" back in microservices (short paper)*. USENIX ATC '18, Boston, MA, USA, 2018.

Carnegie Mellon University