SimFaaS: A Performance Simulator for Serverless Computing Platforms

Nima Mahmoudi

nmahmoud@ualberta.ca

UNIVERSITY OF ALBERTA

Hamzeh Khazaei

hkh@yorku.ca

YORK UNIVERSITÉ UNIVERSITY

N. Mahmoudi and H. Khazaei, "SimFaaS: A Simulator for Serverless Computing Platforms," in *The 11th IEEE International Conference on Cloud Computing and Services Science (CLOSER 2021)*.

Performant and Available Computing Systems (PACS) Lab

TOC

Introduction

Serverless Computing

- A cloud-native deployment model
  - Developers build and run application on the cloud
  - Pay for resources used instead of provisioned
  - Event-driven execution model
- Runtime operation and management done by the provider
  - Overhead reduction
  - Provisioning
  - Scaling resources
- Software is developed by writing functions
  - Well-defined interface
  - Functions deployed separately

Typical Developer Workflow

Image source: https://aws.amazon.com/lambda/

The Need for a Performance Simulator

- No previous work has been done that accurately captures the unique dynamics of modern serverless computing platforms
- Accurate performance simulation can beneficial in many ways:
  - Ensure the Quality of Service (QoS)
  - Improve performance metrics
  - Predict/optimize infrastructure cost
  - Move from best-effort to performance guarantees
- It can benefit both serverless provider and application developer
- Allows performance prediction in absence of analytical performance models

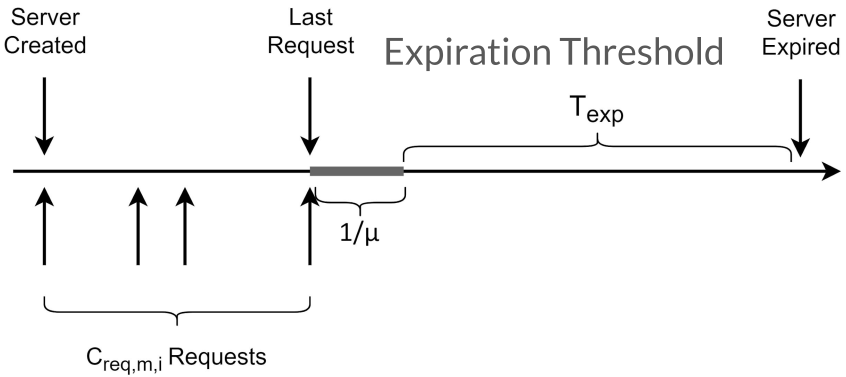
System Description

Function States, Cold Starts, and Warm Starts

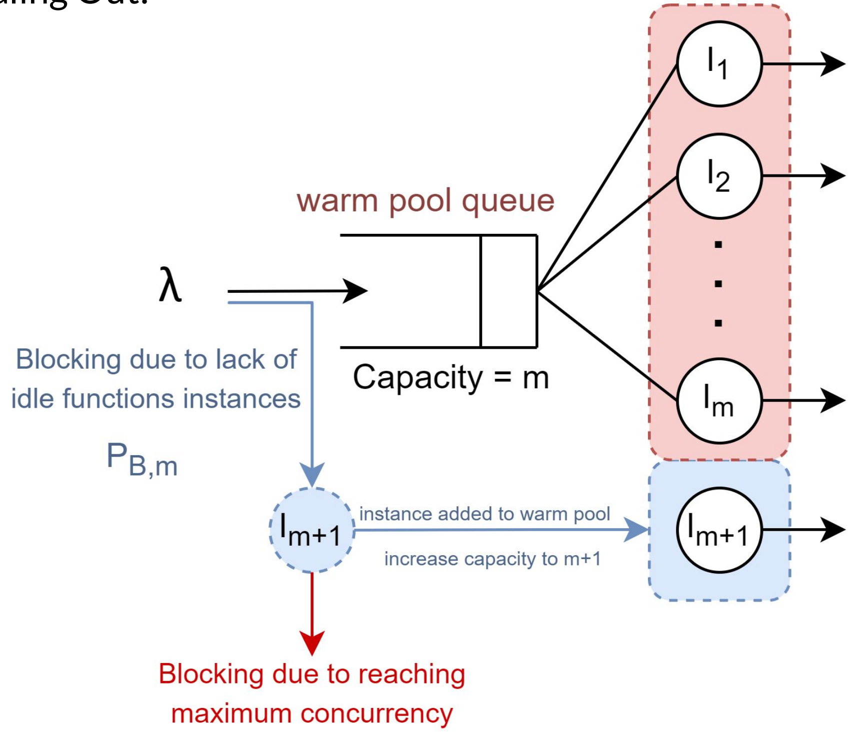
- Function States:
  - **Initializing**: Performing initialization tasks to prepare the function for incoming requests. Includes infrastructure initialization and application initialization.
  - **Running**: Running the tasks required to process a request.
  - **Idle**: Provisioned instance that is not running any workloads. The instances in this state are not billed.
- Cold Start Requests:
  - A request that needs to go through initialization steps due to lack of provisioned capacity.
- Warm Start Requests:
  - Only includes request processing time since idle instance was available

Autoscaling

Scaling In:

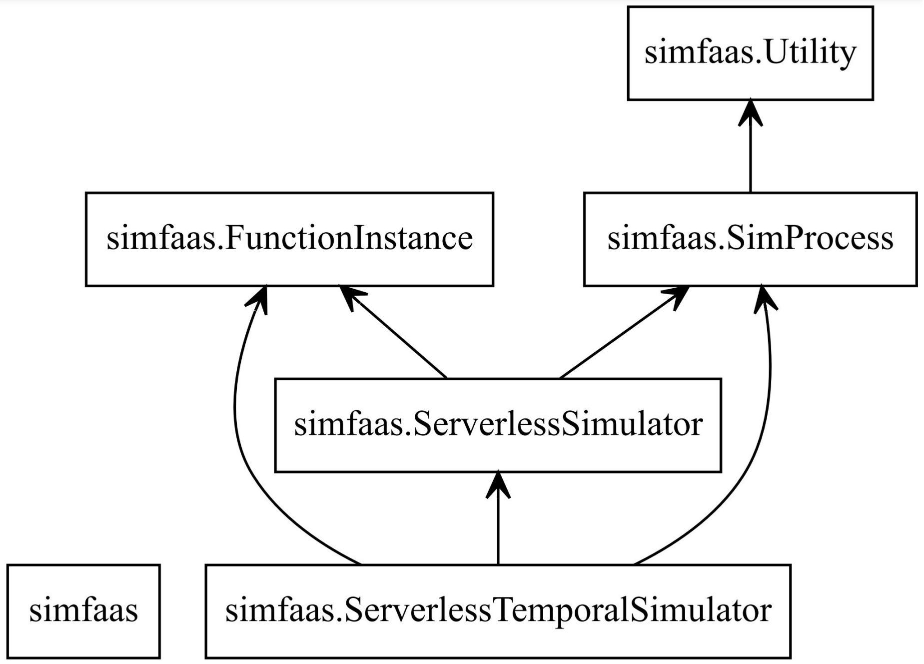


Scaling Out:

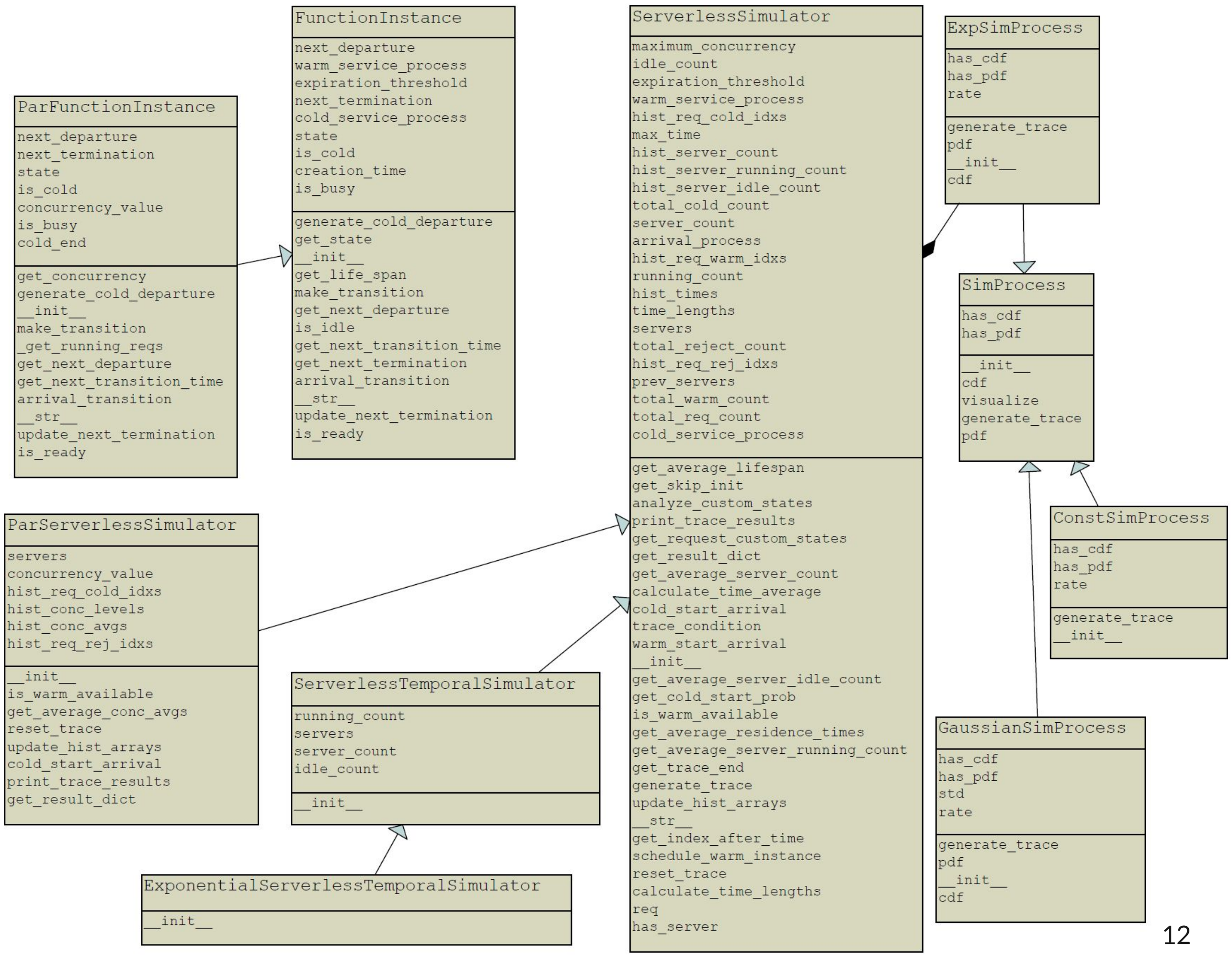


The Design of SimFaaS

SimFaaS Package Diagram



SimFaaS UML Class Diagram



Sample Use Cases

Steady State Analysis

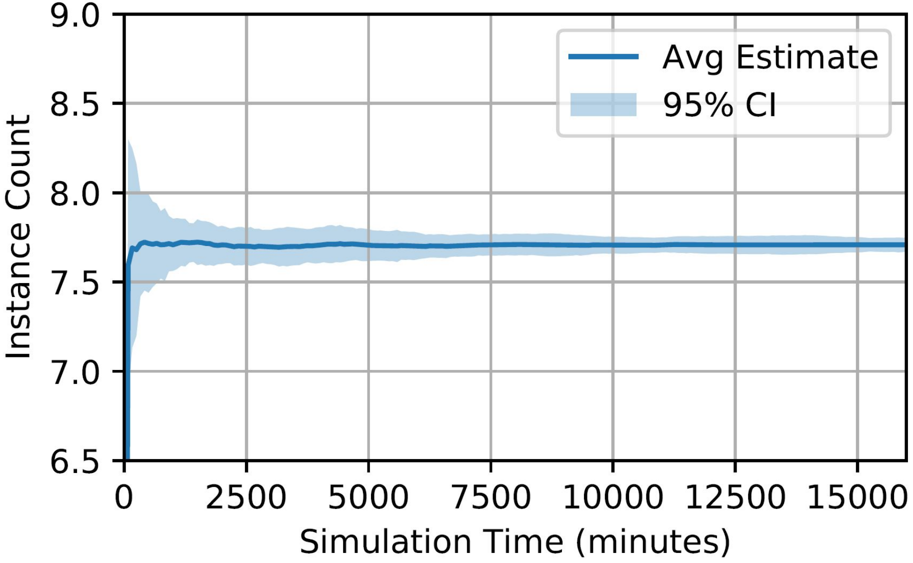
Parameter	Value
Arrival Rate	0.9 req/s
Warm Service Time	1.991 s
Cold Service Time	2.244 s
Expiration Threshold	10 min
Simulation Time	10^6 s
Skip Initial Time	100 s
*Cold Start Probability	0.14 %
*Rejection Probability	0 %
*Average Instance Lifespan	6307.7389 s
*Average Server Count	7.6795
*Average Running Servers	1.7902
*Average Idle Count	5.8893

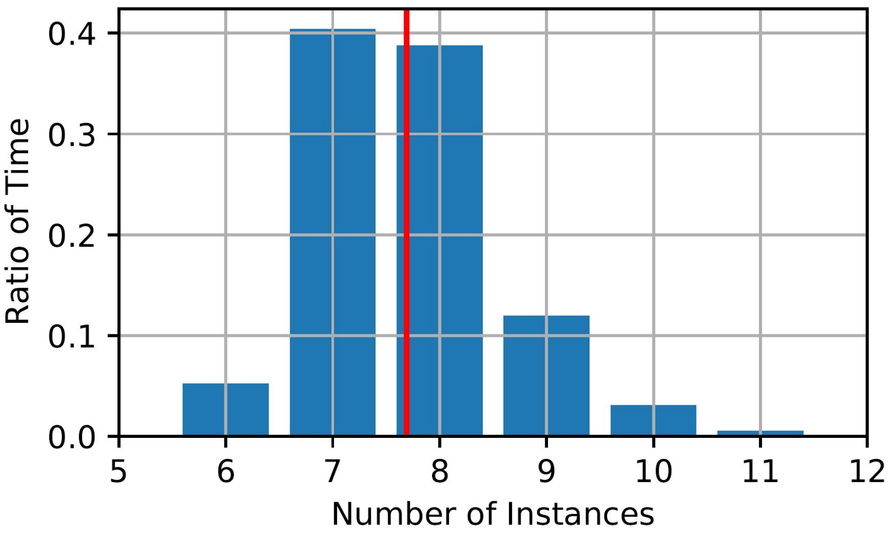
Sample Code - Instance Count

```
sim = Sim(arrival_rate=0.9,
          warm_service_rate=1/1.991,
          cold_service_rate=1/2.244,
          expiration_threshold=600,
          max_time=1e6)
sim.generate_trace(debug_print=False,
                   progress=False)
sim.print_trace_results()
```

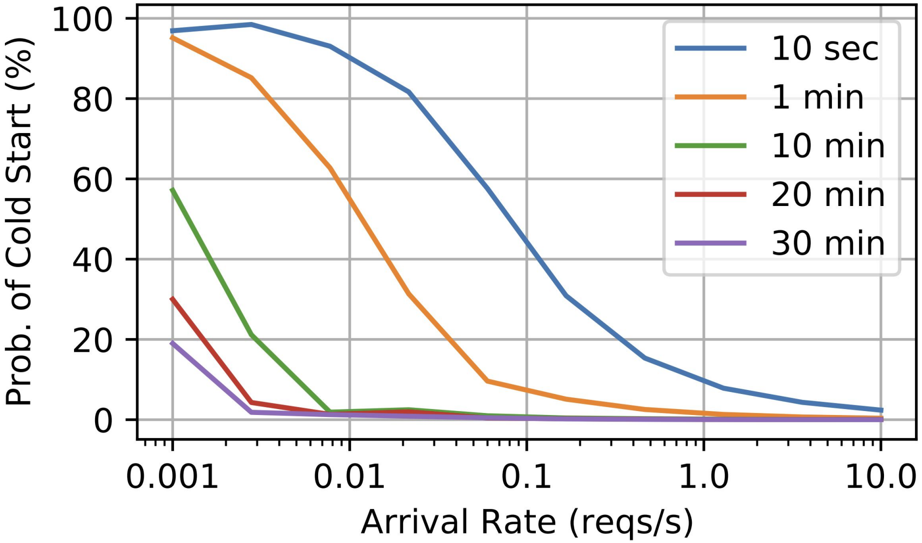
```
unq_vals, val_times =
    sim.calculate_time_average(
        sim.hist_server_count,
        skip_init_time=100
    )
plt.bar(unq_vals, val_times)
plt.grid(True)
plt.axvline(
    x=sim.get_average_server_count(),
    c='r'
)
```

Instance Count Estimation





What-If Analysis: Adaptive Expiration Threshold

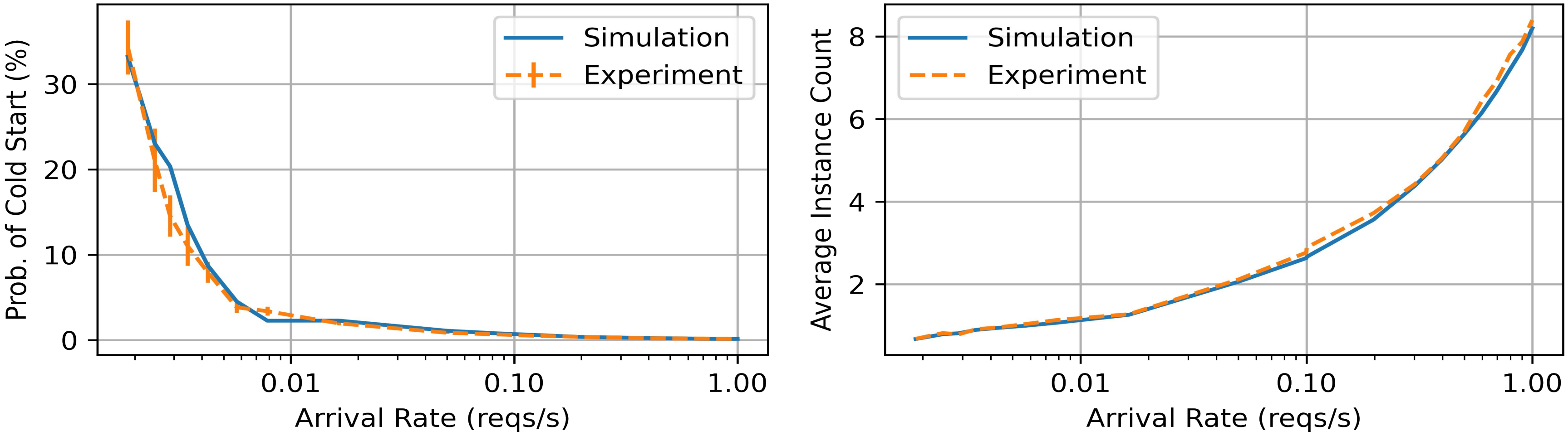


Experimental Validation

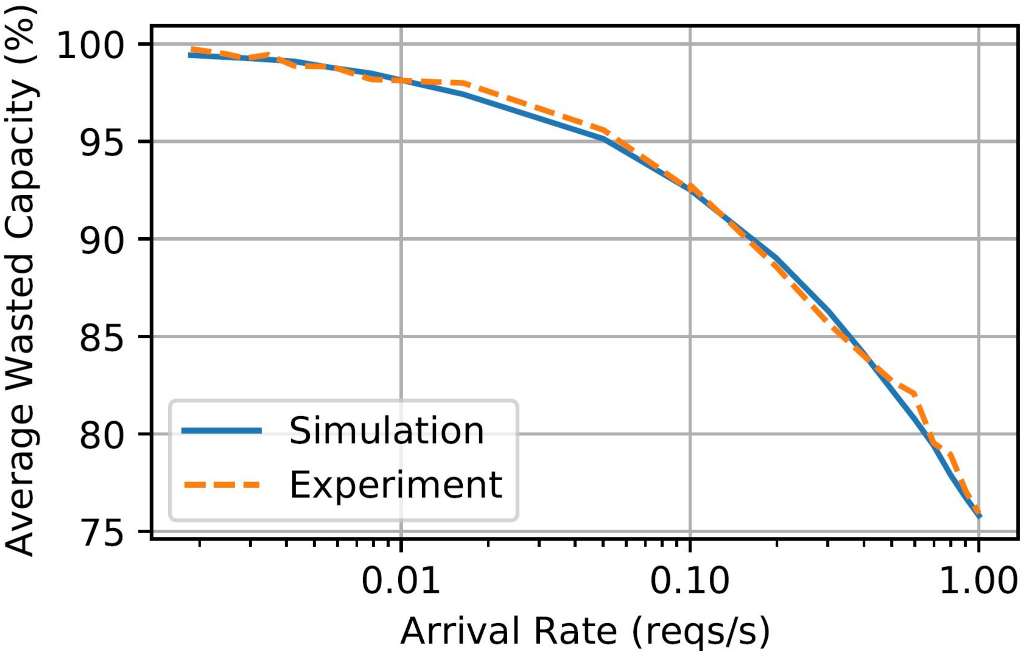
Experimental Setup

- Experiments done on AWS Lambda
  - Python 3.6 runtime with 128MB of RAM on us-east-1 region
  - A mixture of CPU and IO intensive tasks
- Client was a virtual machine on Compute Canada Arbutus
  - 8 vCPUs, 16GB of RAM, 1000Mbps connectivity, single-digit milliseconds latency to AWS servers
  - Python with in-house workload generation tool *pacswg*
  - Official *boto3* library for API communication
  - Communicated directly with Lambda API, no intermediary interfaces like API Gateway

Experimental Validation



Experimental Validation



Conclusion

Conclusion

- Accurate and extendable performance simulator
- Ability to predict important performance/cost related metrics
- Can predict QoS
- Can benefit serverless providers
- Could be useful to application developers
  - Predict how their system will react under different loads
  - Help them optimize their memory configuration to occur minimal cost that satisfies performance requirements
- Making the *expiration threshold* adaptive would allow better cost-performance tradeoff

Thank you

Website: research.nima-dev.com
Twitter: @nima_mahmoudi