SimFaaS: A Performance Simulator for Serverless Computing Platforms

Nima Mahmoudi  
nmahmoud@ualberta.ca

Hamzeh Khazaei  
hkh@yorku.ca
Introduction

System Description

The Design of SimFaaS

Sample Use Cases

Experimental Validation

Conclusion
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

Upload your code to AWS Lambda or write code in Lambda's code editor

Set up your code to trigger from other AWS services, HTTP endpoints, or in-app activity

AWS Lambda
Lambda runs your code only when triggered, using only the compute resources needed

Just pay for the compute time you use

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 be 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:
- Server Created
- Last Request
- Expiration Threshold $T_{exp}$
- Server Expired
- $\lambda$
- $1/\mu$
- $C_{req,m,i}$ Requests

Scaling Out:
- Capacity = $m$
- Blocking due to lack of idle functions instances
- $P_{B,m}$
- Blocking due to reaching maximum concurrency
- Instance added to warm pool
- Increase capacity to $m+1$
- Warm pool queue

$\lambda$
The Design of SimFaaS
Sample Use Cases
Steady State Analysis

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arrival Rate</td>
<td>0.9 req/s</td>
</tr>
<tr>
<td>Warm Service Time</td>
<td>1.991 s</td>
</tr>
<tr>
<td>Cold Service Time</td>
<td>2.244 s</td>
</tr>
<tr>
<td>Expiration Threshold</td>
<td>10 min</td>
</tr>
<tr>
<td>Simulation Time</td>
<td>$10^6$ s</td>
</tr>
<tr>
<td>Skip Initial Time</td>
<td>100 s</td>
</tr>
<tr>
<td>Cold Start Probability</td>
<td>0.14 %</td>
</tr>
<tr>
<td>Rejection Probability</td>
<td>0 %</td>
</tr>
<tr>
<td>Average Instance Lifespan</td>
<td>6307.7389 s</td>
</tr>
<tr>
<td>Average Server Count</td>
<td>7.6795</td>
</tr>
<tr>
<td>Average Running Servers</td>
<td>1.7902</td>
</tr>
<tr>
<td>Average Idle Count</td>
<td>5.8893</td>
</tr>
</tbody>
</table>
```python
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

![Graphs showing the probability of cold starts and average instance count as functions of arrival rate. The graphs compare simulation results with experiment results.]
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