Temporal Performance Modeling of Serverless Computing Platforms

Nima Mahmoudi  
nmahmoud@ualberta.ca

Hamzeh Khazaei  
hkh@yorku.ca


https://www.serverlesscomputing.org/wosc6/#p1
TOC

Introduction
System Description
Analytical Modelling
Experimental Validation
Conclusion
Introduction
Serverless Computing

- Runtime operation and management done by the provider
  - Reduces the overhead for the software owner
  - Provisioning
  - Scaling resources
- Software is developed by writing functions
  - Well-defined interface
  - Functions deployed separately
Serverless Computing

Image source: https://aws.amazon.com/lambda/
The Need for a Performance Model

- No previous work has been done for performance modelling of Serverless Computing platforms
- Accurate performance modelling 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
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

Expiration Threshold

Scaling In:

- Server Created
- Last Request
- Server Expired

$\lambda$  
Blocking due to lack of idle functions instances

$P_{B,m}$

$C_{req,m,i}$ Requests

$\frac{1}{\mu}$

$T_{exp}$

Scaling Out:

- Warm pool queue
- Capacity $= m$

Blocking due to reaching maximum concurrency
Other Important Characteristics

- **Initialization Time**: The amount of time an instance spends in the initializing state.
- **Response Time**: No queuing, so it is equal to service time. It remains stable throughout time for cold and warm start requests.
- **Maximum Concurrency Level**: Maximum number of instances that can be in the state `running` in parallel.
- **Request Routing**: To facilitate scaling in, requests are routed to recently created instances first.
Analytical Modelling
Overview

Warm Pool Model:

- \( m = 0 \)
- \( m = 1 \)
- \( m = 2 \)
- \( \cdots \)
- \( m = M \)

1. \( R_{a,0} \) to \( R_{a,1} \) to \( R_{a,2} \) to \( \cdots \) to \( R_{a,M-1} \) to \( R_{a,M} \)

- Blocking due to lack of idle functions instances
  - \( P_{B,m} \)

- Capacity = \( m \)

- Instance added to warm pool
  - Increase capacity to \( m + 1 \)

- Blocking due to reaching maximum concurrency

Server Created

Last Request

Server Expired

- \( \frac{1}{\mu} \)

- \( \lambda \)

- \( T_{exp} \)

- \( C_{req,m_j} \) Requests
• **Cold Start Rate**
  ○ The system behaves like an Erlang Loss System (M/G/m/m).
  ○ The blocked requests are either rejected (reached maximum concurrency level) or cause a cold start (and thus the creation of a new instance)

• **Arrival Rate for Each Instance**
  ○ Requests blocked by instance n are either processed by instance n+1 or blocked by it.
  ○ The difference between blocked rates gives us individual arrival rates.

• **Server Expiration Rate**
  ○ Can be calculated knowing individual arrival rates and expiration threshold.

• **Warm Pool Model**
  ○ Each state represents the number of instances in the warm pool.
For each state, we can also calculate:

- **Probability of Rejection**: Probability of being blocked when reaching maximum concurrency.
- **Probability of Cold Start**: Probability of being blocked in other states.
- **Average Response Time**: 
  \[ RT_{avg} = RT_w (1 - P_B) + RT_c P_{clid} \]
- **Mean Number of Instances in Warm Pool**:
  - Running
  - Idle
- **Utilization**: Ratio of instances in *running* state over all instances.

All predictions can be found in a time-bounded fashion (e.g., answers the question “what happens to my QoS in the next 5 minutes?”)
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

- Predictions are made 5 minutes into the future
- Assumed oracle request pattern prediction
Sample Workload
Experimental Results

![Experimental Results](image-url)
Experimental Results (2)

![Graph showing experimental results over time]

- **Utilization**
  - **Experiment**
  - **Model Prediction**

**Experiment Time (MM:SS)**

- 10:00
- 20:00
- 30:00
- 40:00
- 50:00
- 60:00
Conclusion
Conclusion

- Accurate and tractable analytical performance model
- Ability to predict important performance/cost related metrics
- Can predict QoS
- Can benefit serverless providers
  - Ability to predict QoS under different loads on deployment time
  - Can be used in the management to prevent performance degradation
- Could be useful to application developers
  - Predict how their system will perform in the immediate future
  - Help them optimize their memory configuration to occur minimal cost that satisfies performance requirements throughout their daily request patterns
- Can be used in the management systems to warm-up instances to prevent SLA violations
Summary

Website:  nima-dev.com
Twitter:  @nima_mahmoudi

Experiment
Model Prediction

Requests / Sec

0 5 10 15

10:00 20:00 30:00 40:00 50:00 60:00

Experiment Time (MM:SS)

Cold Start Prob.

0.000 0.002 0.004 0.006 0.008

10:00 20:00 30:00 40:00 50:00 60:00

Experiment Time (MM:SS)

to state (i,j)

\[
\begin{pmatrix}
-R_{a,0} & R_{a,0} & 0 & \ldots & 0 \\
\exp_{1} & -(\exp_{1} + R_{a,1}) & R_{a,1} & 0 & \ldots \\
0 & \ldots & -(\exp_{2} + R_{a,2}) & 0 & \ldots \\
\ldots & \ldots & \ldots & \ldots & \ldots \\
0 & \ldots & R_{a,M-1} & -(\exp_{M-1} + R_{a,M-1}) & R_{a,M-1} \\
0 & \ldots & 0 & R_{a,M} & -R_{a,M}
\end{pmatrix}
\]

from state (i,)

\[
\begin{pmatrix}
-R_{a,0} & R_{a,0} & 0 & \ldots & 0 \\
\exp_{1} & -(\exp_{1} + R_{a,1}) & R_{a,1} & 0 & \ldots \\
0 & \ldots & -(\exp_{2} + R_{a,2}) & 0 & \ldots \\
\ldots & \ldots & \ldots & \ldots & \ldots \\
0 & \ldots & R_{a,M-1} & -(\exp_{M-1} + R_{a,M-1}) & R_{a,M-1} \\
0 & \ldots & 0 & R_{a,M} & -R_{a,M}
\end{pmatrix}
\]