Temporal Performance Modeling of Serverless Computing Platforms

Nima Mahmoudi
nmahmoud@ualberta.ca
University of Alberta

Hamzeh Khazaei
hkh@yorku.ca
York Université University

N. Mahmoudi and H. Khazaei, “Temporal Performance Modelling of Serverless Computing Platforms,” in Proceedings of the 6th International Workshop on Serverless Computing, 2020, p. 1–6., doi: 10.10.1145/3429880.3430092.

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

Performant and Available Computing Systems (PACS) Lab

TOC

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

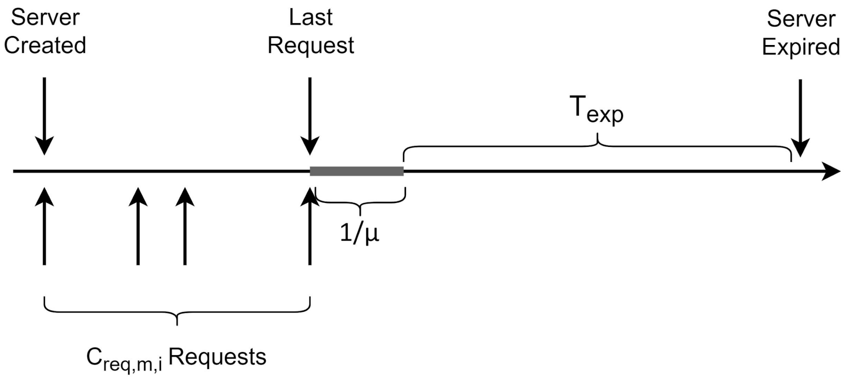
Function States, Cold Starts, and Warm Starts

- 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

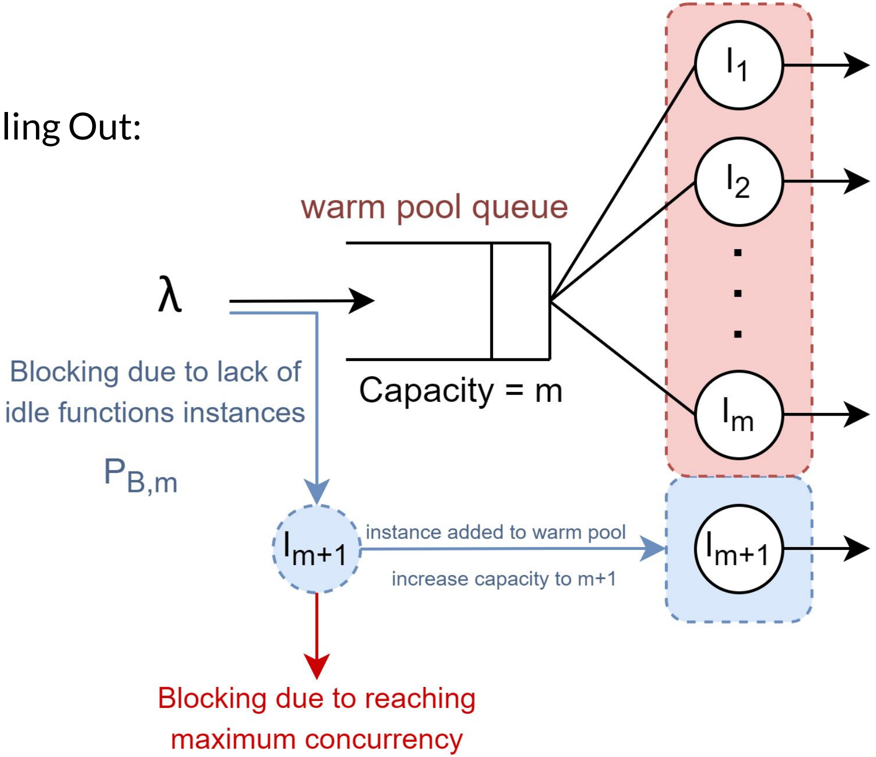
Autoscaling

Expiration Threshold

Scaling In:



Scaling Out:



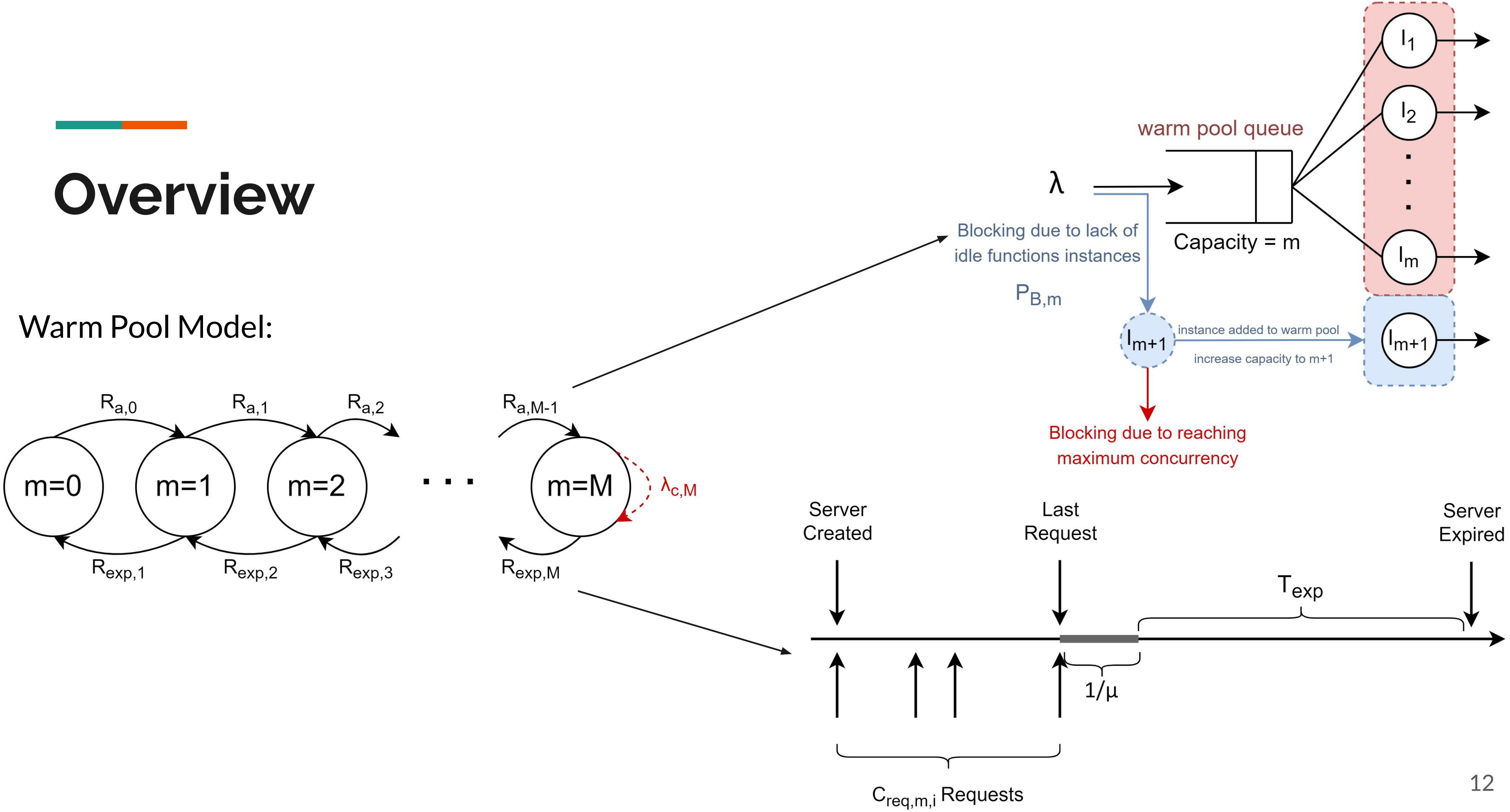
Other Important Characteristics

- **Initialization Time:** The amount of time 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:



- 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.

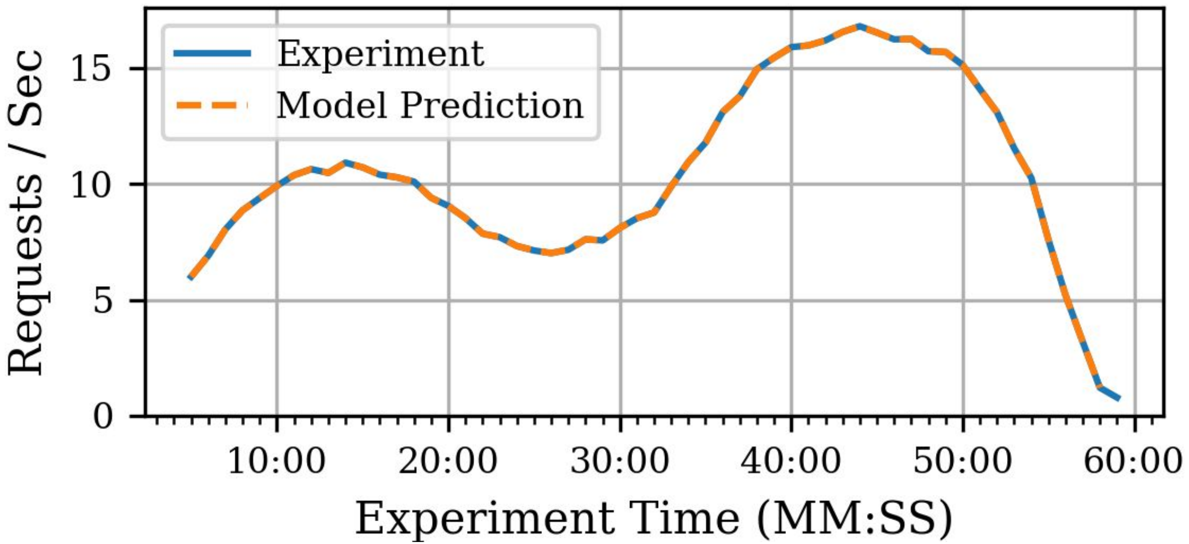
- For each state, we can also calculate:
  - **Probability of Rejection:** Probability of being blocked when reaching maximum concurrency.
  - **Probability of Cold Start:** Probability of being blocked in other states.
  - **Average Response Time:** $RT_{avg} = RT_w(1 - P_B) + RT_c P_{cld}$
  - **Mean 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

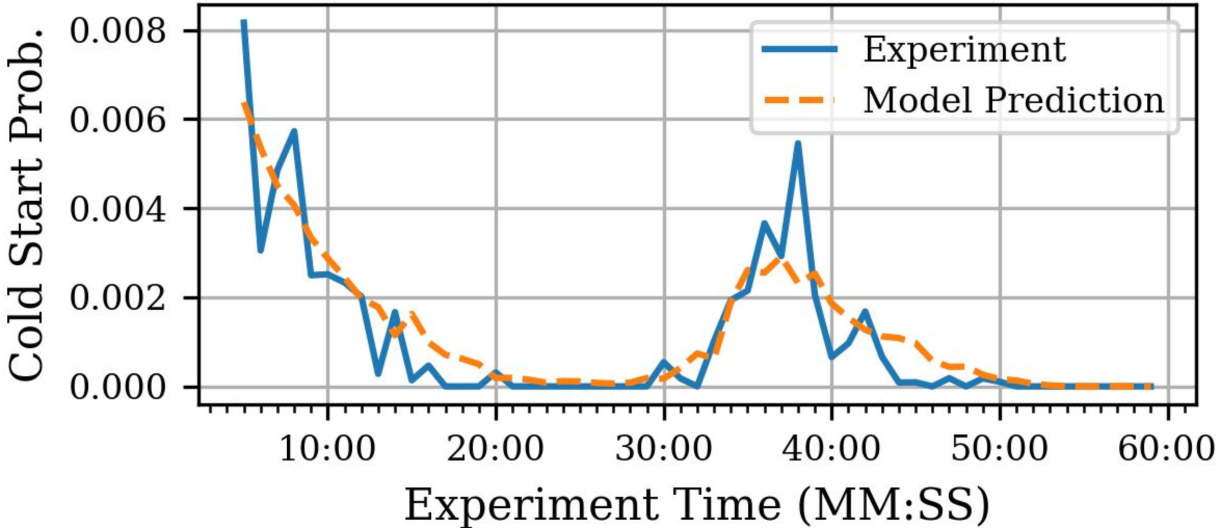
Experimental Setup

- 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

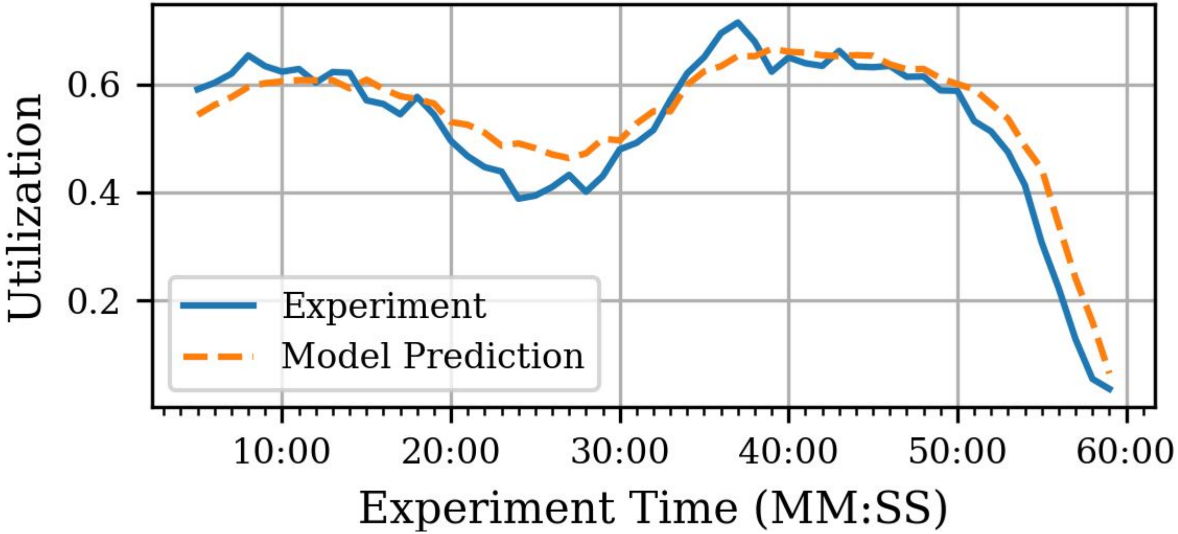
Sample Workload



Experimental Results



Experimental Results (2)



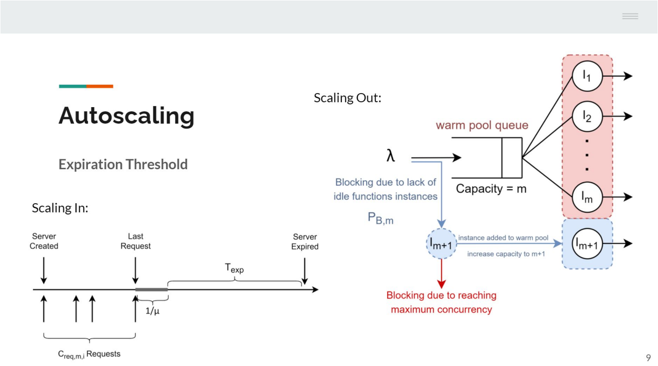
Conclusion

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



→ to state $(,j)$

↓ from state $(i,)$

$$
\begin{bmatrix}
-R_{a,0} & R_{a,0} & 0 & \cdots & & 0 \\
R_{exp,1} & -(R_{exp,1}+R_{a,1}) & R_{a,1} & 0 & & \vdots \\
0 & \cdots & -(R_{exp,2}+R_{a,2}) & 0 & \cdots & \vdots \\
\vdots & & & \ddots & & \\
 & & R_{exp,M-1} & -(R_{exp,M-1}+R_{a,M-1}) & R_{a,M-1} & \\
0 & & \cdots & 0 & R_{exp,M} & -R_{exp,M}
\end{bmatrix}
$$