FaaSRank: Learning to Schedule Functions for Serverless Platforms

Hanfei Yu[1], Athirai A. Irissappane[1], Hao Wang[2], Wes J. Lloyd[1]

September 29, 2021

University of Washington Tacoma[1], Louisiana State University[2]

Outline

> Background
> Design
> Implementation
> Evaluation
> Conclusion

Background

> Serverless Computing & FaaS

> Scheduling & Load Balancing

> Reinforcement Learning

Traditional Load Balancing

> **Web Service Load Balancing**

> **Classic Algorithms**
- Round-robin
  - > distributes requests to servers in rotation
- Least-connections
  - > distributes requests to the server with the least number of active connections
- Greedy
  - > sends requests to the same server until filling capacity
- Hashing
  - > sends requests to servers based on unique hash values
- …

FaaS vs Traditional Scheduling

> **Common**
  – Distribute web/function requests to servers

> **Differences?**

Traditional Web Service	FaaS
Fixed deployment	Freeze-thaw life cycle
Static resource management	Dynamic resource provisioning

> **Existing schedulers in FaaS** ⇨ **Classic!**
  – AWS Lambda: Greedy
  – Apache OpenWhisk: Hashing

How do we incorporate server states to improve scheduling outcomes for FaaS?

Server Assessment

> **A static fitness function**
  – Scores are used to characterize fitness of attributes
> **Select a server with the highest score**
> **Schedule the next function request to the selected server**

Server	CPU Score	Memory Score	Disk Score	Network Score	Infrastructure Score	Load Score	Overall Score
#1	0.2	0.1	0.05	0.05	0.1	0.1	0.6
#2	0.15	0.05	0.05	0.05	0.05	0.15	0.5
#3	0.3	0.1	0.05	0.05	0.1	0.2	**0.8**

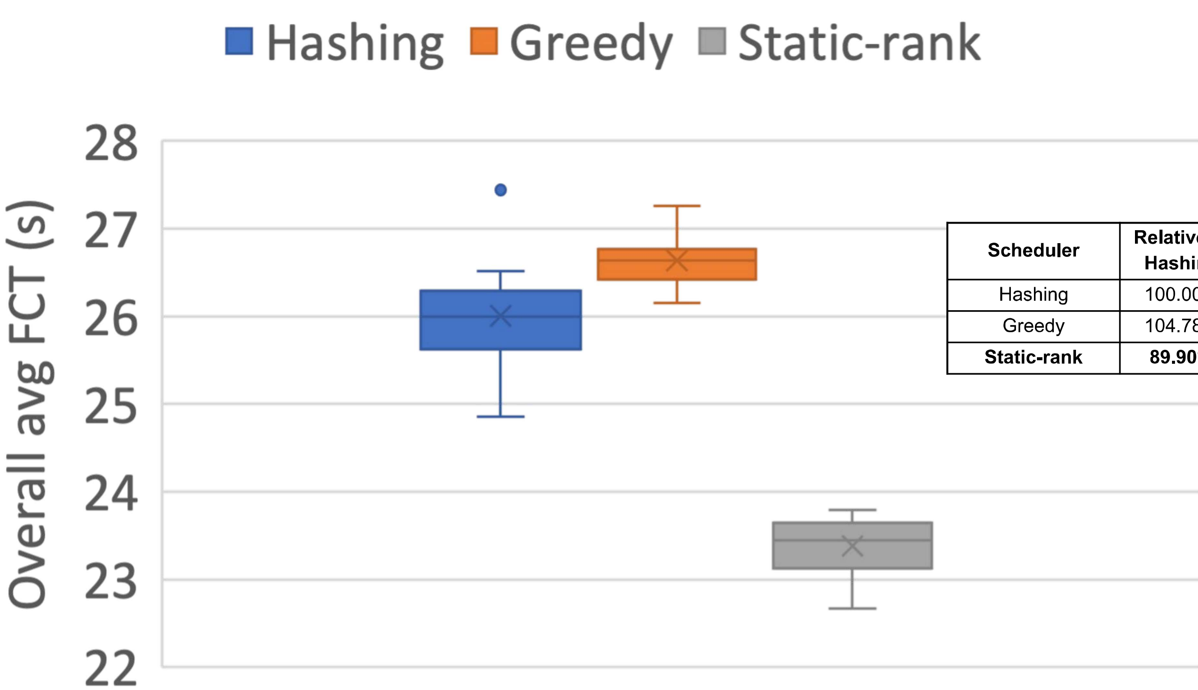
Proof of Concept Experiment

> **An Apache OpenWhisk cluster**
  – 10 workers
  – Each worker with 8 CPU cores, 16 GBs RAM

> **Workload**
  – 10 serverless applications
  – Realworld invocation traces from Microsoft Azure Functions

> **Schedulers**
  – Hashing (OpenWhisk default)
  – Greedy (AWS Lambda)
  – Static-rank (a fitness function) **** our heuristic approach***

Motivation Result



Scheduler	Relative to Hashing
Hashing	100.00%
Greedy	104.78%
Static-rank	**89.90%**

Server Assessment

> **A static fitness function**
> – Scores are used to characterize fitness of attributes

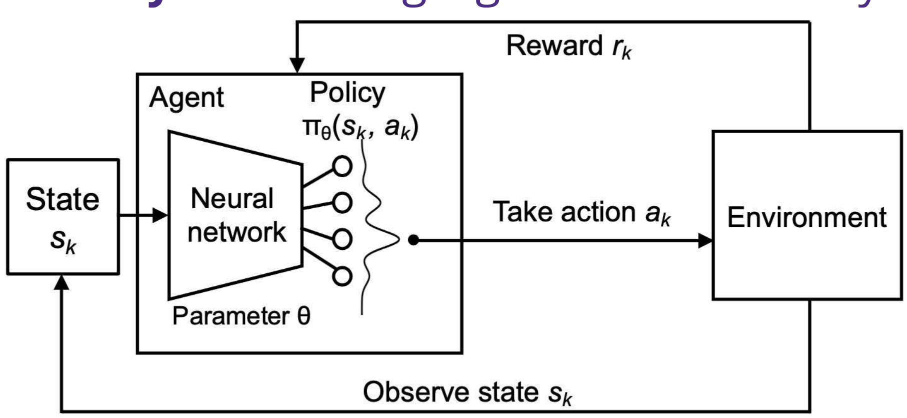
Server	CPU Score	Memory Score	Disk Score	Network Score	Infrastructure Score	Load Score	Overall Score
#1	0.2	0.1	0.05	0.05	0.1	0.1	0.6
#2	0.15	0.05	0.05	0.05	0.05	0.15	0.5
#3	0.3	0.1	0.05	0.05	0.1	0.2	**0.8**

Can we automate this? Yes!

> **A self-learning function using Reinforcement Learning (RL)**

(Deep) Reinforcement Learning (DRL)

> **Environment:** FaaS platform
> **Agent:** scheduler
> **State:** server/function information
> **Action:** schedule a function to a server
> **Reward:** performance of function execution
> **Policy:** scheduling algorithm learned by agent



Policy Gradient in DRL Training

- > **Learn policies by performing *gradient ascent* directly on the parameters of neural networks**
- > **Gradient Ascent**
  - Push up the probabilities of actions that lead to higher rewards, and push down the probabilities of actions that lead to lower rewards, until arriving at the optimal policy
- > **Reward**
  - provides feedback
- > **Actor-Critic**
  - **Actor network** outputs decisions and receives rewards
  - **Critic network** outputs values to judge actor network
  - The policy distribution is updated with the ***Advantage***
  - Advantage = rewards - values

Reinforcement Learning for FaaS Scheduling

- > **Challenges**
- > **Objective**
- > **FaaSRank**

Challenges

- **Server assessment**
  - How to compose together available metrics to assess individual servers to make reasonable trade-offs between cold starts and resource contention in real-time?
- **Cluster scalability**
  - Can the neural networks adapt to scalable clusters?
- **Huge action space**
  - Can the RL agent efficiently explore the action space?

Objective

- **Function Completion Time (FCT): the time from function arrival until its completion**
  - Initialization overhead
  - Waiting time in any platform queues
  - Function execution time
- **Average FCT: averaged over an individual function or workload**
- **Our goal is to minimize the average FCT of an entire workload**

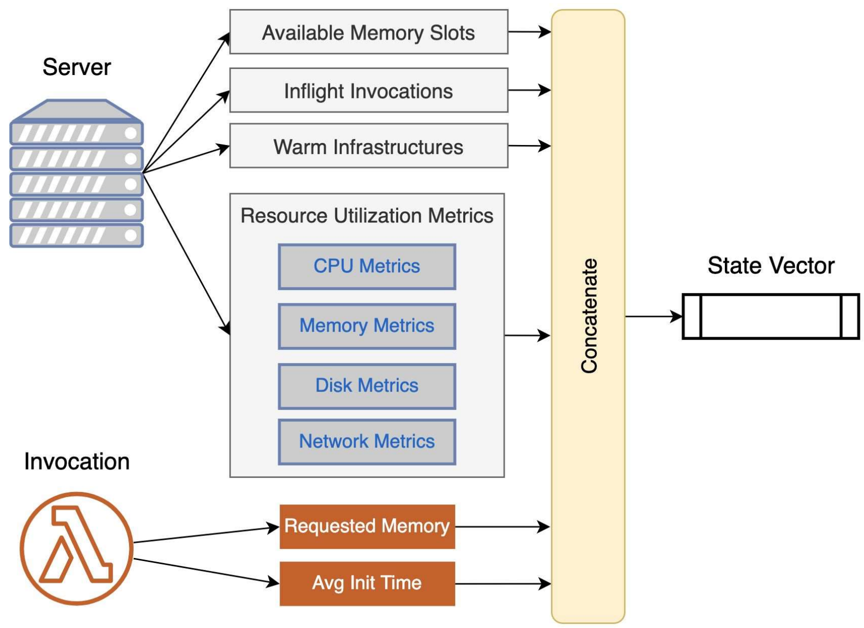
FaaSRank

- A RL-based scheduler for serverless FaaS platforms
  - 22 features of server state
  - 5 features of controller state
- Given any workloads, FaaSRank tries to
  - Minimize overall average FCT
  - Scale to any size of cluster
  - Efficient exploration of the action space

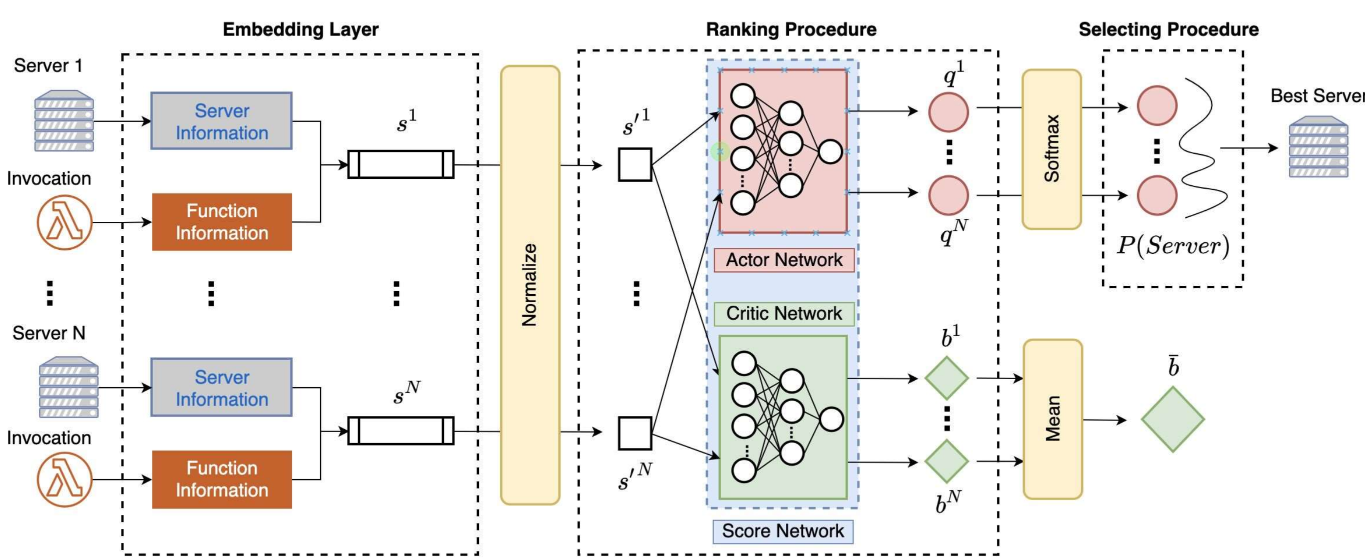
Design

- Server Assessment (Policy Network Embedding Layer)
- Score-Rank-Select (Policy Network)
- Training FaaSRank

Policy Network Embedding Layer



Policy Network



Training FaaSRank

- **Proximal Policy Optimization (PPO)**
  - State-of-the-art, efficient and performant, devised by Open-AI, 3930+ citations
  - Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O., Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347, 2017.
- **Training proceeds in *episodes*. In each episode:**
  - A series of client function invocations arrive at the FaaS platform
  - When all of the function invocations finish, the episode is considered complete

FaaSRank Training Algorithm

- **Initialize parameters of actor and critic network**
- **For episode 1, 2, 3, … do:**
  - Run policy in environment until termination
  - Collect trajectory (state-action pairs)
  - Discount rewards
  - Compute baseline values from critic network
  - Compute advantage = rewards - baseline values
  - Use advantages to update both actor and critic network
- **End for**

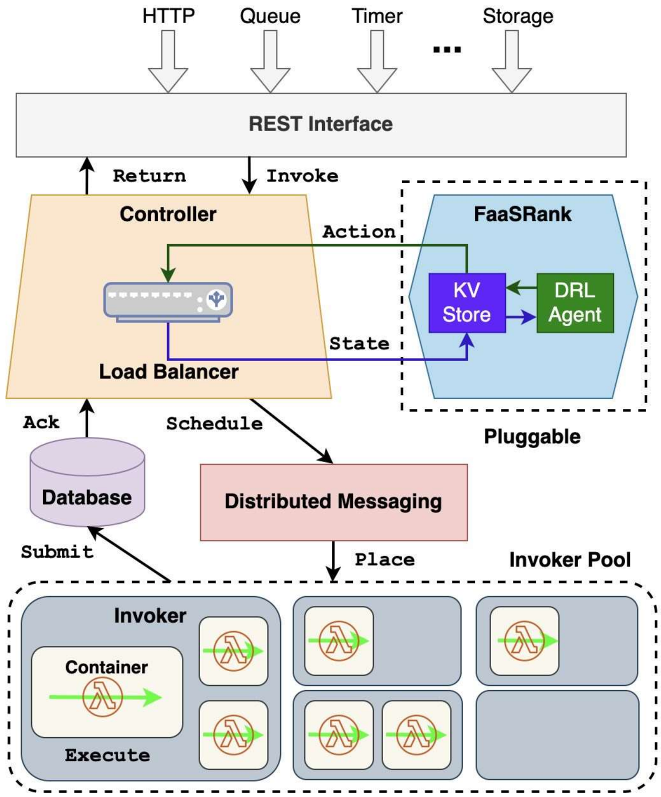
Implementation

> FaaSRank Integrated with OpenWhisk

Apache OpenWhisk

> An open source, distributed serverless platform
> Execute functions (fx) in response to events at any scale
> Manage the infrastructure, servers, and scaling using Docker containers
> Support functions in Node.js, Go, Java, Scala, PHP, Python, Ruby, Swift, Ballerina, .NET, and Rust

OpenWhisk with FaaSRank



Evaluation

- > Experimental Setups
- > Results

Baseline Schedulers

- **Hashing**
  - OpenWhisk
- **Round-robin**
- **Least-connections**
- **Greedy**
  - AWS Lambda
- **Static-rank**
  - We created Static-rank to investigate resource utilization aware scheduling prior to developing FaaSRank
  - Overall Score = 2*CPU + 1.5*Mem + Disk + Net Load_Avg + Available_Mem_Slots

Testbed Clusters

- **Compute Canada Cloud**
  - 13 VMs, 1 inference engine, 1 frontend, 1 backend, 10 workers
  - Each with 8 CPU cores (Intel Xeon Skylake IBRS 2.50GHz), 16 GBs RAM
- **AWS EC2**
  - Spot Instances
  - 13 c5d.2xlarge VMs, 1 inference engine, 1 frontend, 1 backend, 10 workers
  - Each with 8 CPU cores (Intel Xeon Platinum 8124M 3.00GHz), 32 GBs RAM

Applications

Application	Type	Memory (MBs)	Avg Cold FCT (s)	Avg Warm FCT (s)
Dynamic Html (DH)	Web App	512	4.45	2.34
Email Generation (EG)	Web App	256	2.20	0.21
Image Processing (IP)	Multimedia	256	5.88	3.52
Video Processing (VP)	Multimedia	512	6.86	1.19
Image Recognition (IR)	ML	512	4.28	0.09
K Nearest Neighbors (KNN)	ML	512	4.99	1.11
Gradient Descent (GD)	ML	512	4.15	0.60
Arithmetic Logic Unit (ALU)	Scientific	256	5.72	3.45
Merge Sorting (MS)	Scientific	256	3.87	1.94
DNA Visualization (DV)	Scientific	512	8.57	3.11

Applications

> **Characterized on a mini OpenWhisk cluster**
  - **AWS EC2 Dedicated Host**
  - **1 user, 1 frontend, 1 backend, 1 worker**

> **Cold and warm runtimes are average FCT of 10 times of experiments**

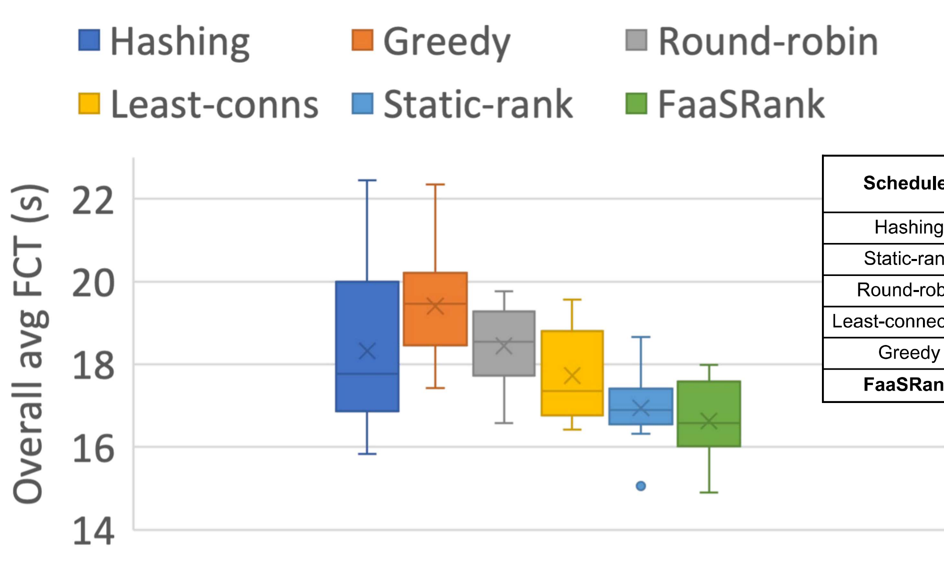
> **Collected from**
  - **SeBS**: A Serverless Benchmark Suite for Function-as-a-Service Computing
  - Characterizing Serverless Platforms with **ServerlessBench**
  - **ENSURE**: Efficient Scheduling and Autonomous Resource Management in Serverless Environments

Workload Traces

- Adapted serverless traces from Microsoft Azure Functions
- Trace IDs:
  - Common trace: SC (Canada Cloud), SA (AWS)
  - Unique traces: M1-10 (AWS)

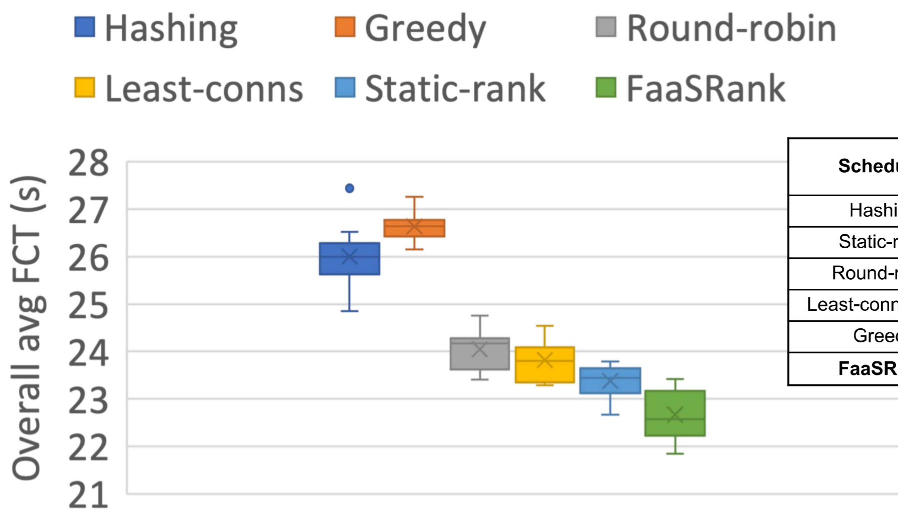
WL	Load	Agg CPU Time	Num calls	Avg IAT	Len
SC	93.75 %	4368.71 s	292	0.262 s	60 s
SA	132.9 %	6196.89 s	408	0.184 s	60 s
M1	56.67 %	2640.94 s	209	0.219 s	37 s
M2	57.00 %	2656.32 s	178	0.242 s	36 s
M3	57.01 %	2656.69 s	201	0.255 s	44 s
M4	59.05 %	2751.87 s	201	0.217 s	35 s
M5	71.83 %	3347.33 s	226	0.236 s	44 s
M6	74.95 %	3492.49 s	253	0.251 s	53 s
M7	80.22 %	3738.29 s	256	0.244 s	52 s
M8	82.54 %	3846.15 s	276	0.215 s	48 s
M9	86.40 %	4026.24 s	318	0.210 s	54 s
M10	100.00 %	4659.86 s	295	0.242 s	59 s

Overall Average FCT (Canada Cloud)



Scheduler	Relative to Hashing
Hashing	100.00%
Static-rank	109.01%
Round-robin	100.64%
Least-connections	96.72%
Greedy	105.89%
FaaSRank	**90.75%**

Overall Average FCT (AWS)



Scheduler	Relative to Hashing
Hashing	100.00%
Static-rank	102.44%
Round-robin	92.48%
Least-connections	91.60%
Greedy	102.43%
FaaSRank	**89.90%**

Avg FCTs (Common trace – Canada Cloud)

Color						
Rank	1st	2nd	3rd	4th	5th	6th

Application	Hashing	Static-rank	Round-robin	Least-connections	Greedy	FaaSRank
DH	100.00	102.57	102.79	108.19	116.18	98.55
EG	100.00	114.37	125.33	114.93	116.56	106.89
IP	100.00	102.37	123.07	112.79	127.88	94.12
VP	100.00	92.26	97.56	98.23	97.73	81.88
IR	100.00	100.75	113.68	97.21	117.53	103.85
KNN	100.00	92.84	101.15	91.66	112.31	82.71
ALU	100.00	76.86	88.36	79.70	92.33	75.46
MS	100.00	83.07	90.50	85.29	94.71	82.76
GD	100.00	101.69	102.24	104.42	103.78	97.69
DV	100.00	107.59	106.81	109.56	115.60	102.86

***All values are average FCTs (sec) normalized as a percentage (%) relative to Hashing scheduler**

Avg FCTs (Common trace - AWS)

Color						
Rank	1st	2nd	3rd	4th	5th	6th

Application	Hashing	Static-rank	Round-robin	Least-connections	Greedy	FaaSRank
DH	100.00	93.39	97.83	96.39	105.67	89.86
EG	100.00	87.98	96.32	92.20	107.10	82.65
IP	100.00	84.62	87.80	86.62	98.10	83.17
VP	100.00	91.93	98.68	92.48	104.51	93.91
IR	100.00	87.76	97.24	91.23	110.82	83.65
KNN	100.00	88.60	90.63	91.84	100.19	86.86
ALU	100.00	88.20	88.91	88.21	98.08	84.84
MS	100.00	90.45	95.61	93.76	105.44	88.76
GD	100.00	88.56	89.44	90.90	102.12	86.17
DV	100.00	90.05	93.85	93.67	103.91	93.71

***All values are average FCTs (sec) normalized as a percentage (%) relative to Hashing scheduler**

Unique Traces (M1-10 AWS)

Color						
Rank	1st	2nd	3rd	4th	5th	6th

Workload	Hashing	Static-rank	Round-robin	Least-connections	Greedy	FaaSRank
1	100.00	77.31	81.17	72.13	121.55	80.82
2	100.00	77.12	79.94	80.52	116.19	85.23
3	100.00	64.68	70.20	66.36	101.86	70.40
4	100.00	84.99	92.13	85.39	108.00	81.14
5	100.00	76.91	77.88	74.32	109.43	75.98
6	100.00	60.10	63.57	60.00	106.41	57.75
7	100.00	73.83	82.81	70.31	123.53	70.04
8	100.00	88.19	88.76	88.13	107.53	90.84
9	100.00	78.86	80.48	92.53	104.77	80.11
10	100.00	79.59	78.41	83.79	99.99	77.20

***All values are average FCTs (sec) normalized as a percentage (%) relative to Hashing scheduler**

Conclusions

> FaaSRank can automatically learn good policies for function scheduling in serverless platforms

> FaaSRank outperforms five baseline schedulers by achieving a better overall performance for serverless workloads

Questions

Thank You!