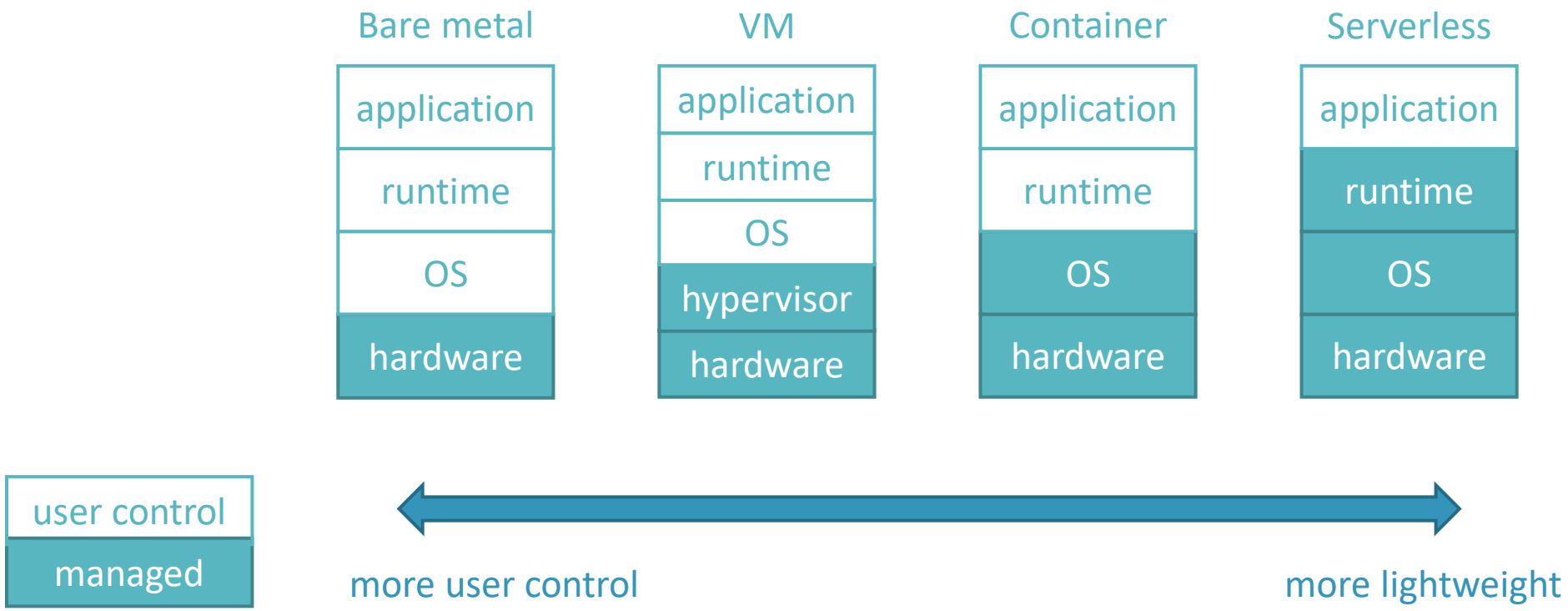
Sophon: Efficient Container-grained Serverless Scheduling

Lixiang Ao, Geoff Voelker, George Porter

UC San Diego
JACOBS SCHOOL OF ENGINEERING
Computer Science and Engineering

Serverless: a lightweight cloud computing paradigm

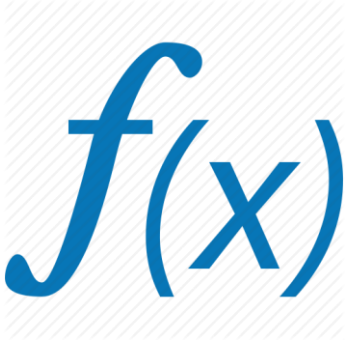


Serverless characteristics

No explicit server
users do not manage server, network, storage, etc.

Elasticity
scale from zero to thousands of instances within seconds

Pay-as-you-go
only bill actual user code time, every 100ms



Stateless
ephemeral functions handle each request individually

What happens when invoking a function

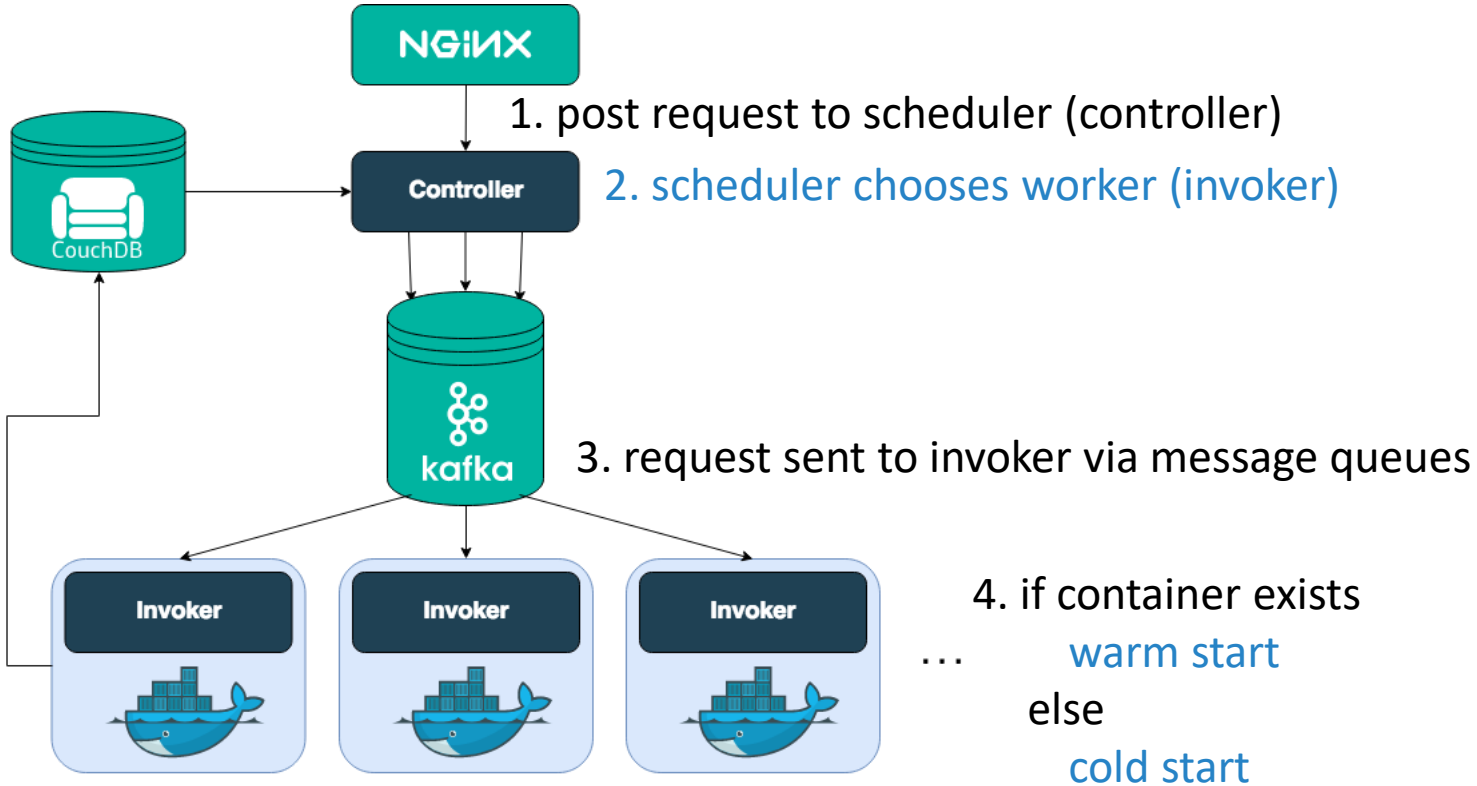
cold start:
>= 2s

1. Loading user image/code
2. Launch security sandboxes (VMs, containers)
3. Initialize runtime, libraries, code
4. Start function

warm start: < 10ms

A warm environment is cached till it expires or till evicted when resource is low.
Whether cold start or warm start is mostly decided by function scheduling

OpenWhisk scheduling overview



1. post request to scheduler (controller)

2. scheduler chooses worker (invoker)

3. request sent to invoker via message queues

4. if container exists

... warm start

else

cold start

OpenWhisk scheduling overview

1. assign each function a sequence of invokers
2. try each invoker in sequence until find enough memory slots

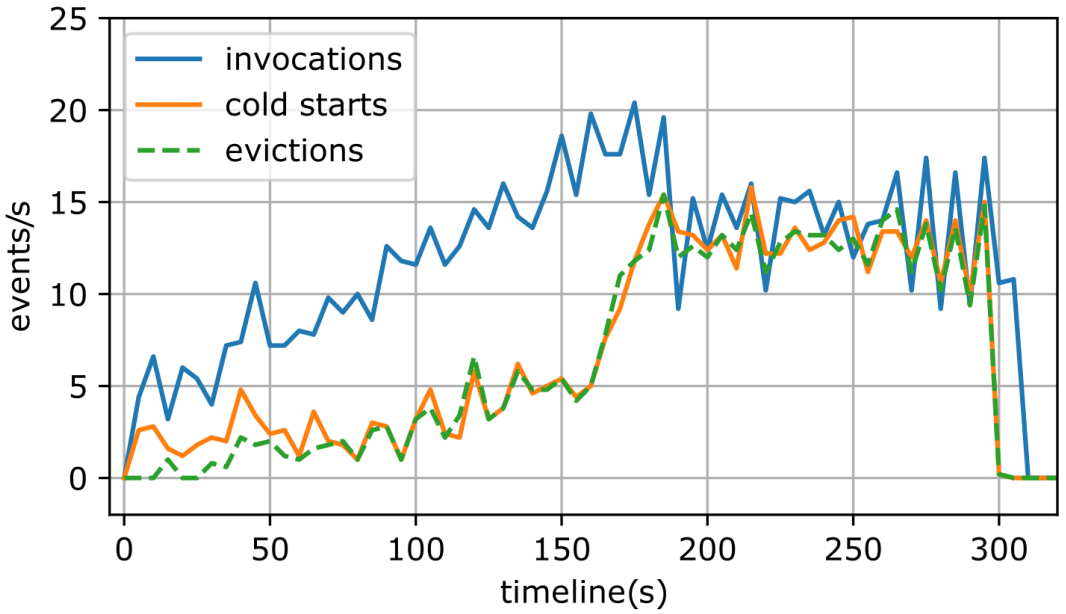
funcA: requires 1 slot
funcA Seq: [0 1 2 3 4]
try 0 ✗
try 1 ✓

funcB: requires 2 slot
funcB Seq: [4 1 3 0 2]
try 4 ✗
try 1 ✗
try 3 ✓

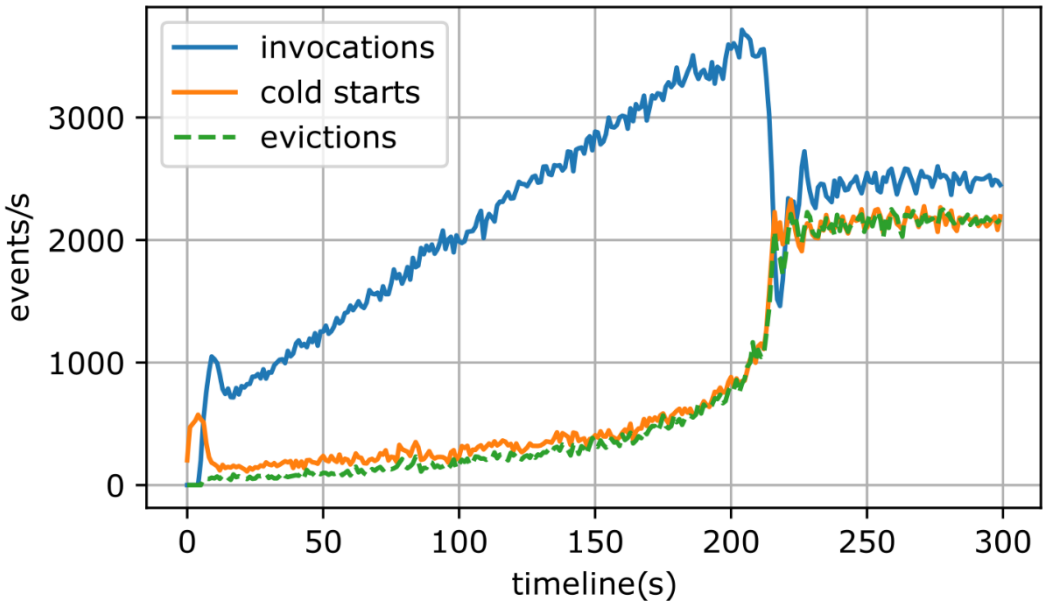
A function tends to concentrate invocations on few Invokers, increasing warm start rate

Problems

Testing using real-world serverless workloads from Azure Functions Traces[1]



10 node OpenWhisk test bed



1000 node simulation

[1] Shahrad, M., Fonseca, R., Goiri, Í., Chaudhry, G., Batum, P., Cooke, J., Laureano, E., Tresness, C., Russinovich, M. and Bianchini, R., 2020. Serverless in the Wild: Characterizing and Optimizing the Serverless Workload at a Large Cloud Provider. *ATC 2020*.

Key insights

- Container contention: Under high workload, different functions compete for container resources, creating unnecessary evictions and cold starts and degrade performance, a phenomenon we dubbed “container thrashing”.
- Root cause: The scheduler makes placement decisions on a node granularity. It only considers the amount of resources on a node, not container states, a key factor in serverless performance.

- Idea: scheduling at container granularity instead of node granularity.

Sophon: Container-grained serverless scheduling

	Node-grained scheduling	Container-grained scheduling
Resource considerations	Amount of resource (RAM, CPU) on a node	Amount of resource on a node + container states
Resource updates	Resource amount changes	Resource amount changes + container state transitions
Scheduling decisions	Choose node	Choose node + container

Sophon design & implementation

- Integrated in OpenWhisk
- Add container states maintaining/monitoring/transitioning functionality to the scheduler, Invoker, and the messaging components.
- Both scheduler and Invokers can update container states
- State transition conflicts are resolved by Invokers

Serverless-tailored scheduling policies

- Goals
  - Increase warm start rate => More concentrated placement
  - Avoid unnecessary evictions => Less concentrated placement
- Sophon's container-grained scheduling enables striking the balance between the two conflicting goals

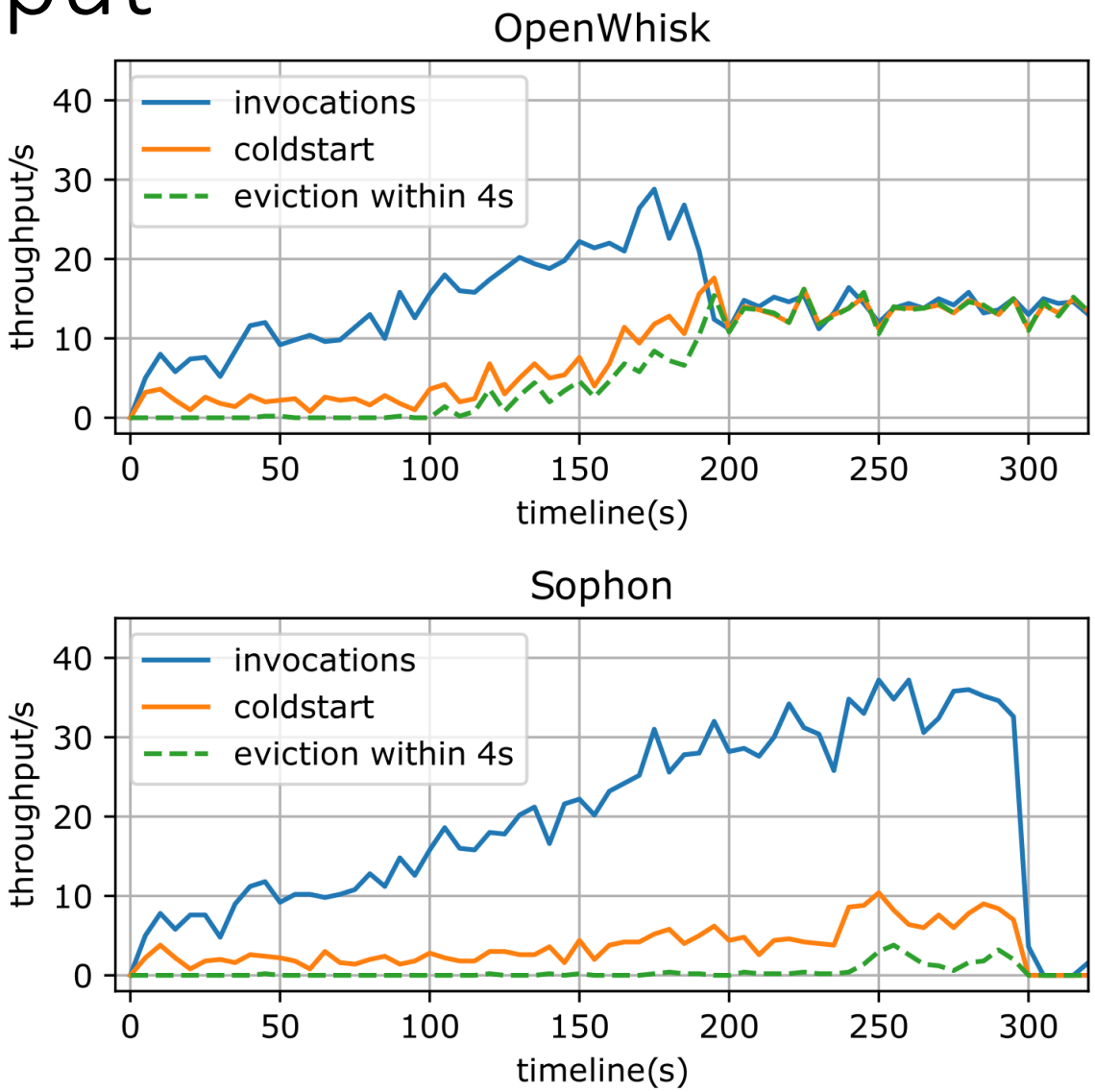
Serverless-tailored scheduling policies

- Cost model: Choose candidate w/ smallest cost
  - Increase warm start rate => distance cost *D*
  - Avoid unnecessary evictions => eviction cost *E*
- D = candidate's index in the Invoker sequence divided by # of Invokers
- E = $\sum_{i \in C} e^{-\lambda t_i}$ where C is all the containers that will be evicted by the decision, t_i is the idle time of container i, λ is decay rate parameter
- Total cost = $W_d \times D + W_e \times E$, where $W_d + W_e = 1$
- λ, W_d, W_e are chosen empirically. We use 0.3, 0.05, and 0.95.

Evaluation

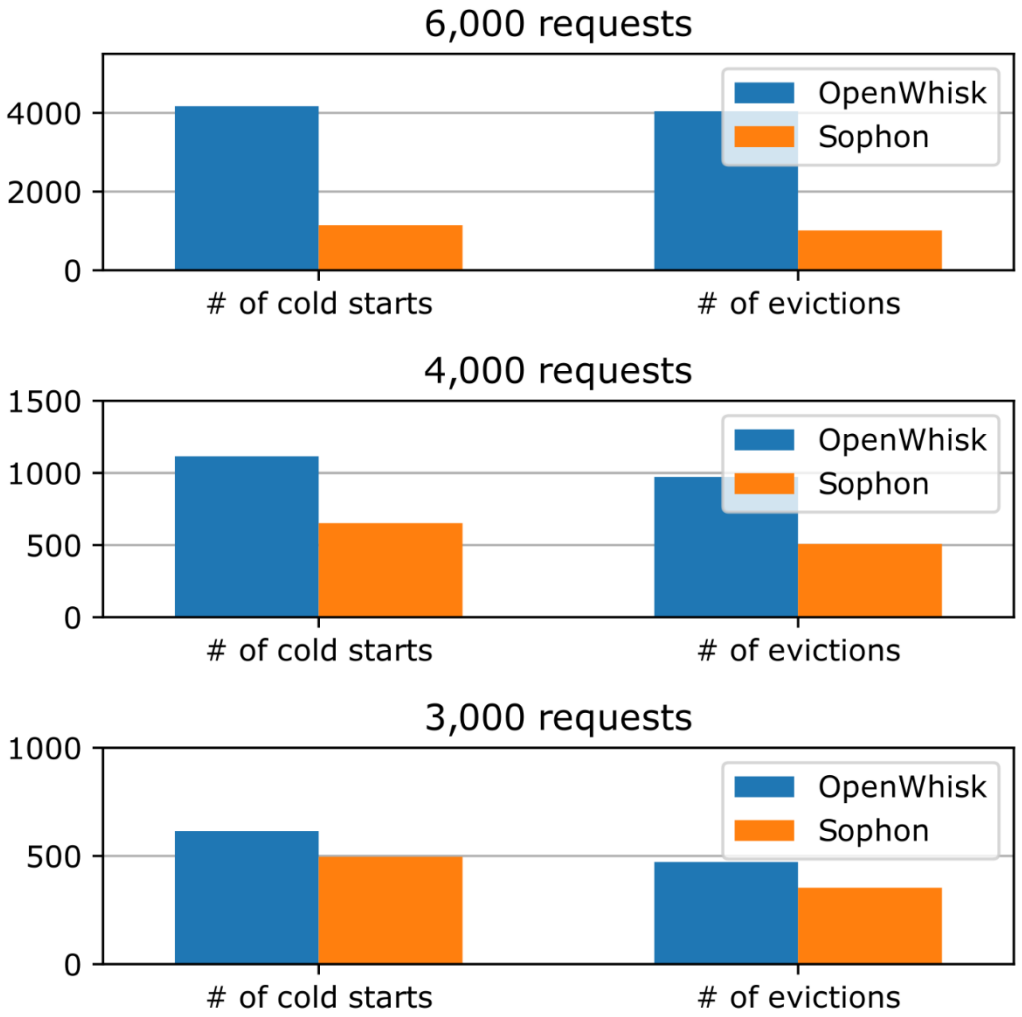
- Scheduling quality of Sophon: throughput, cold starts/evictions, latency
- Justify chosen parameters

Throughput



Sophon prevents container thrashing, increases stable throughput by 80%

Cold starts/evictions

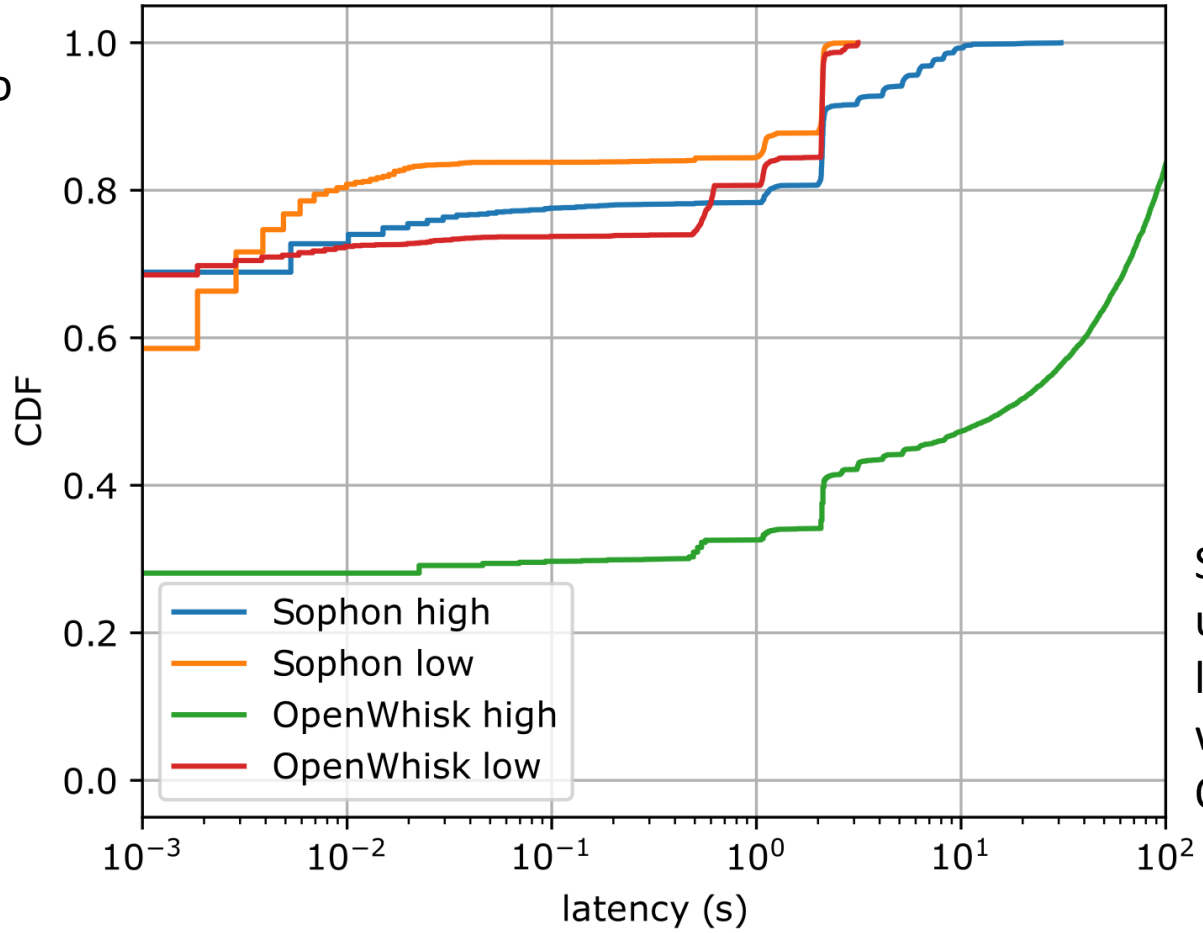


Sophon reduces number of cold starts and evictions by up to 73%. Sophon performs better under high, medium, and low workloads.

Invocation Latency

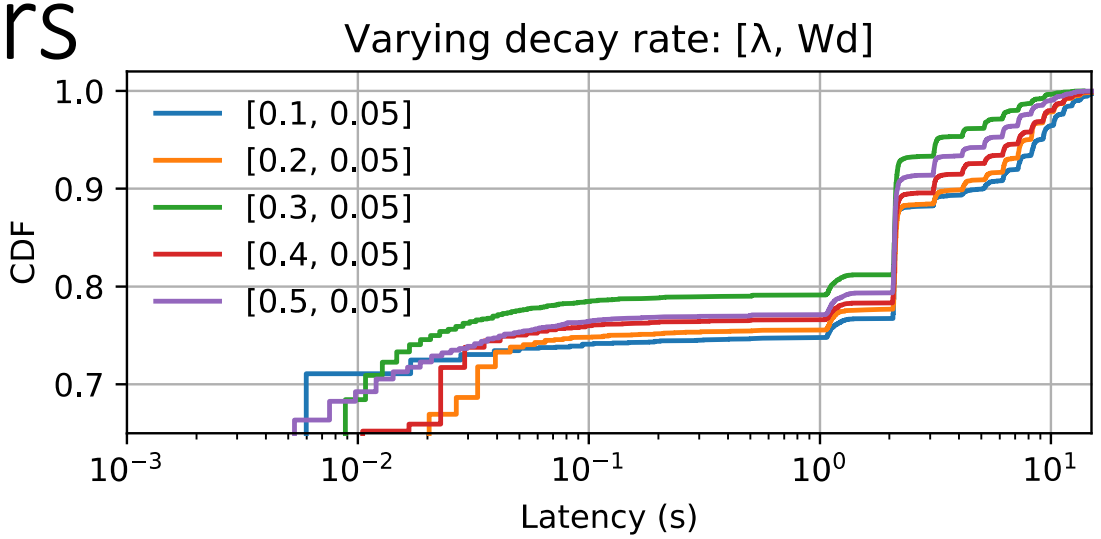
From scheduler receives request to function is started on an Invoker.

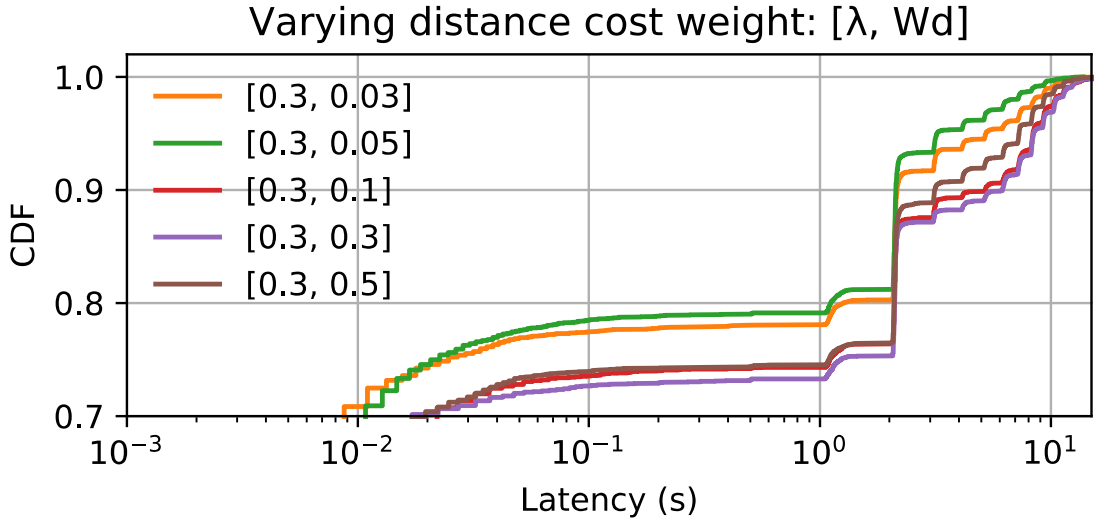
Test with low, high workloads.



Sophon has 28% lower average latency under low workloads and significantly lower latency under high workload when container thrashing exists in OpenWhisk.

Parameters





$\lambda = 0.3, W_d = 0.05$ are optimal parameters.

Conclusion

- Existing node-grained serverless scheduling ignores container contention between functions, creating container thrashing that degrades system performance.
- Sophon uses container-grained scheduling mechanism to prevent container thrashing, and cost model-based policies to balance conflicting scheduling goals.
- We integrate Sophon with OpenWhisk, providing up to 80% higher stable throughput and significantly lower latency.


CNS
THANK YOU!
Q&A