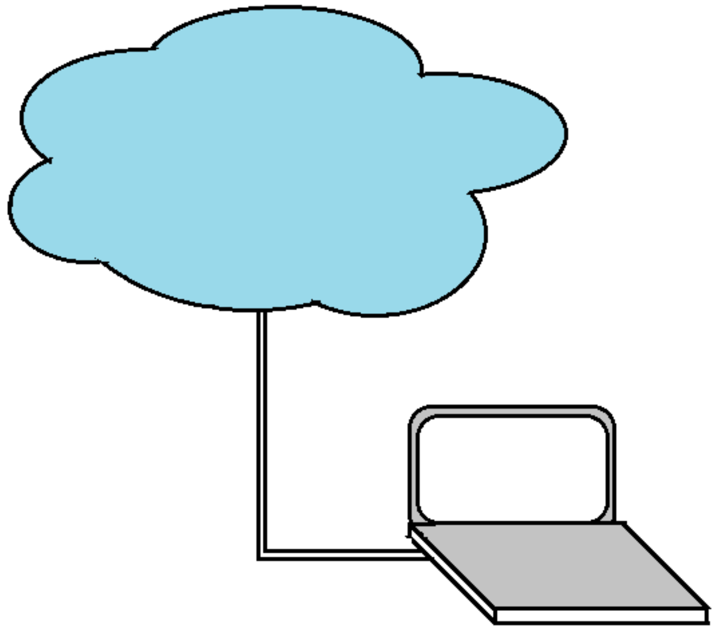
KEEP-ALIVE CACHING FOR THE HAWKES PROCESS



- **SUSHIRDEEP NARAYANA**
- **IAN A. KASH**

DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF ILLINOIS AT CHICAGO

uai2023

UIC UNIVERSITY OF ILLINOIS CHICAGO

What is Serverless Computing?

Cloud Abstraction

- Users pay for usage, do not pay for resources
- Users do not worry about servers

  Provisioning, Reserving, and Configuring – all managed by cloud provider

What is Serverless Computing?

Cloud Abstraction

- Users pay for usage, do not pay for resources
- Users do not worry about servers

  Provisioning, Reserving, and Configuring – all managed by cloud provider

- Serverless - Function as a Service (FaaS)

- Users upload code of their functions to the cloud
- Functions get executed when “triggered” or “invoked” by events

Serverless Computing

- Examples

Serverless Computing

- Examples

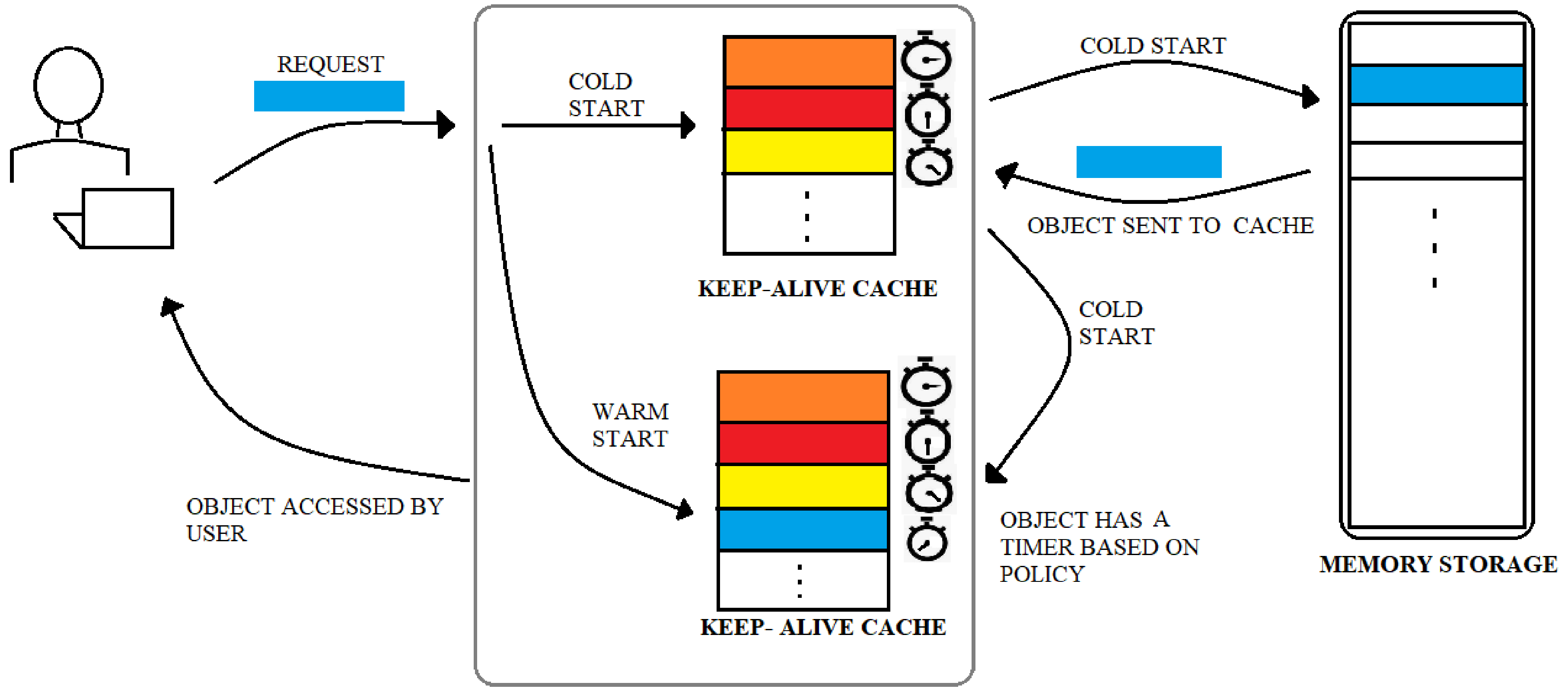
AWS Lambda

Azure Functions

CLOUDFLARE Workers

Google Cloud Functions

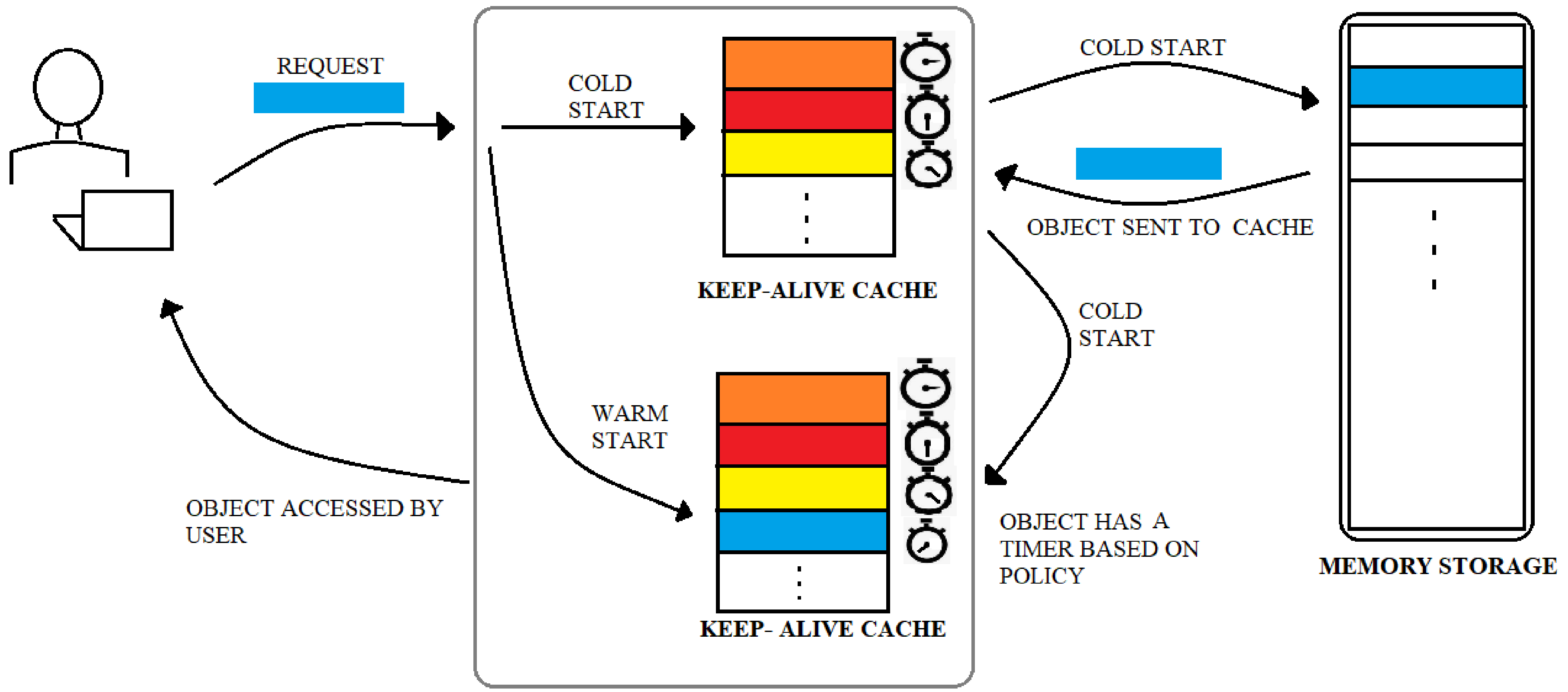
Performance in Serverless Computing



Performance in Serverless Computing

- Quicker function execution - code, environment, and libraries to be in memory

$\Rightarrow$ Warm Start (Cache Hit)



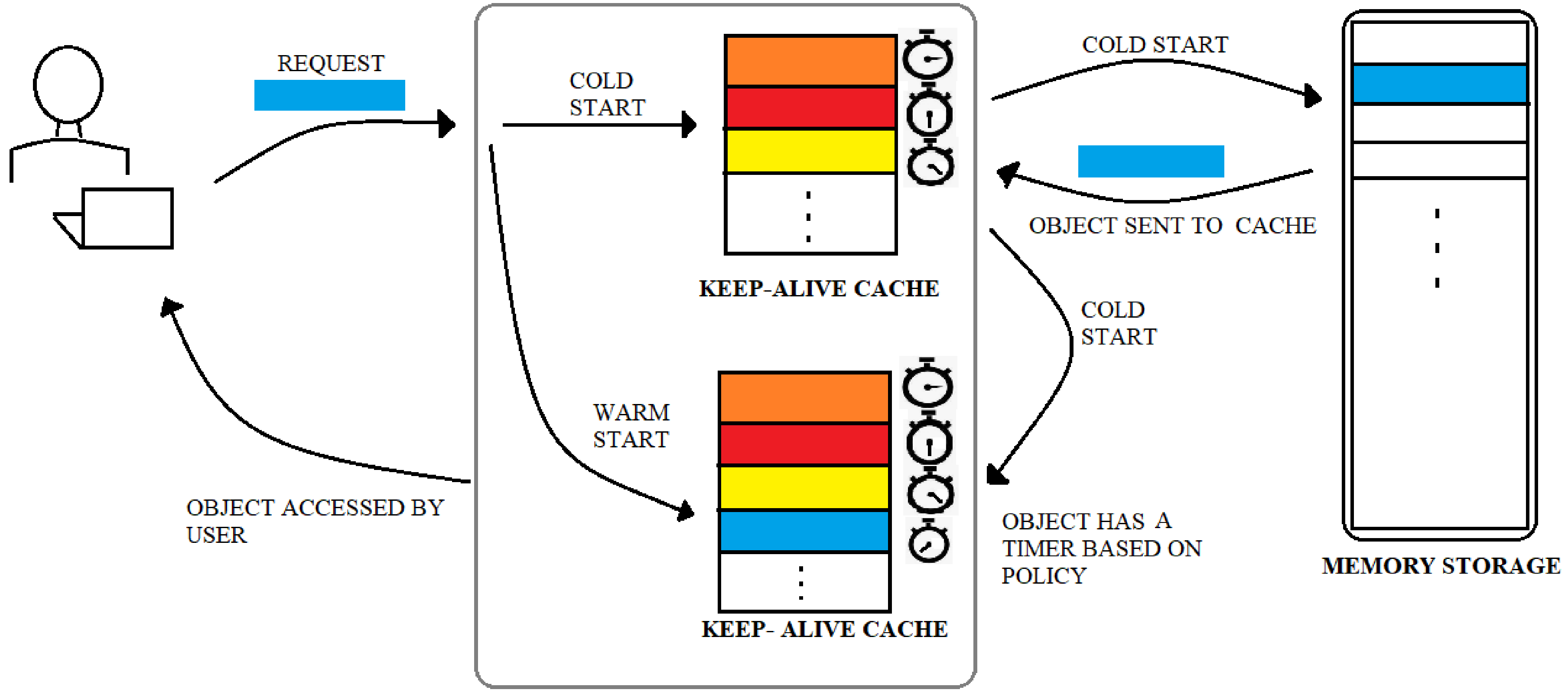
Performance in Serverless Computing

- Quicker function execution - code, environment, and libraries to be in memory

  ➡ Warm Start (Cache Hit)

- Slower Function execution - code, environment, and libraries not in memory

  ➡ Cold Start (Cache Miss)



Challenges in Serverless Computing

- ➢Challenge faced by cloud providers – *cost effective policies*

- ➢ Users are billed on application execution

  -Providers seek high performance with low resource costs

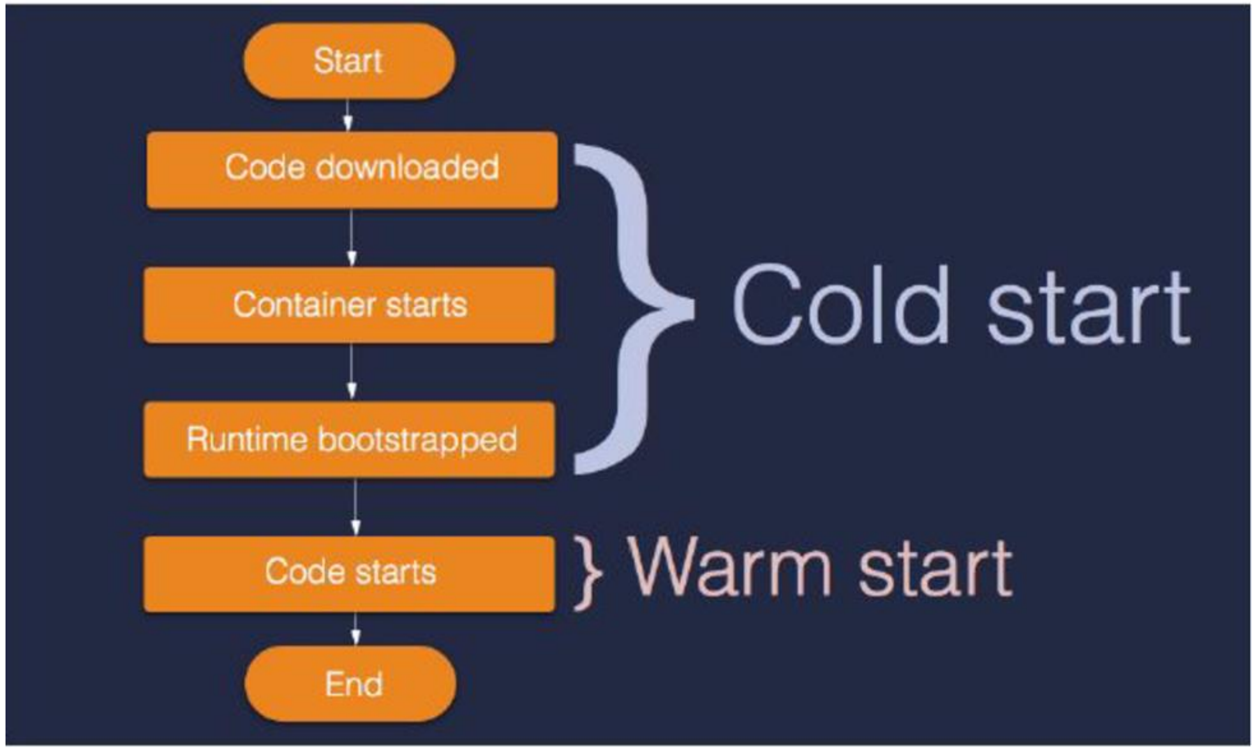
Challenges in Serverless Computing

- Challenge faced by cloud providers – *cost effective policies*

- Users are billed on application execution

  -Providers seek high performance with low resource costs

- Cost of a cold start



AWS Lambda

Introduction to Keep-Alive Caches

- Keep-alive caches
  - serverless computing
  - cost associated with the time an item stays in the cache

Introduction to Keep-Alive Caches

- Keep-alive caches
  - serverless computing
  - cost associated with the time an item stays in the cache

- Different compared to traditional caches – fixed cache size

- Decision in traditional cache policies

  What object to evict from the cache when the cache is full ?

Keep-Alive Caches

- An object in keep-alive cache has a time policy associated to it

- Cache size is not a concern as applications are in cloud computing

Keep-Alive Caches

- ➢An object in keep-alive cache has a time policy associated to it
- ➢Cache size is not a concern as applications are in cloud computing
- ➢Decision - *When is it worth to keep an object in the cache?*

Answer : Model as trade-off problem between

Expected time an object is kept in the cache

vs

Probability of a cache miss

Keep-Alive Cache Policy

- Keep-alive cache policy governed by the following parameters

Keep-Alive Cache Policy

➢Keep-alive cache policy governed by the following parameters

1) Pre-warming window, τ_{pw} = Time policy waits before it loads the application image

Keep-Alive Cache Policy

- Keep-alive cache policy governed by the following parameters

1) Pre-warming window, τ_{pw} = Time policy waits before it loads the application image

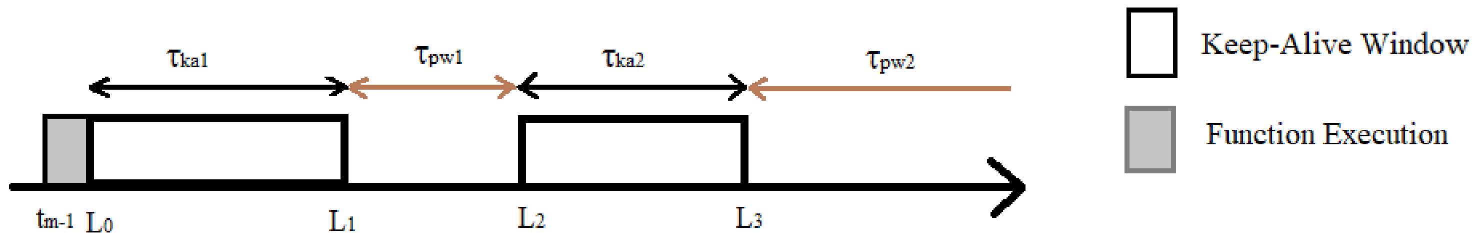
2) Keep-alive window, τ_{ka} = Time during which the application image is kept-alive either after pre-warming or after function execution

Keep-Alive Cache Policy

- **Keep-alive cache policy** = Sequence of keep-alive windows (and pre-warming windows) during which application image moved in & out of cache

Keep-Alive Cache Policy

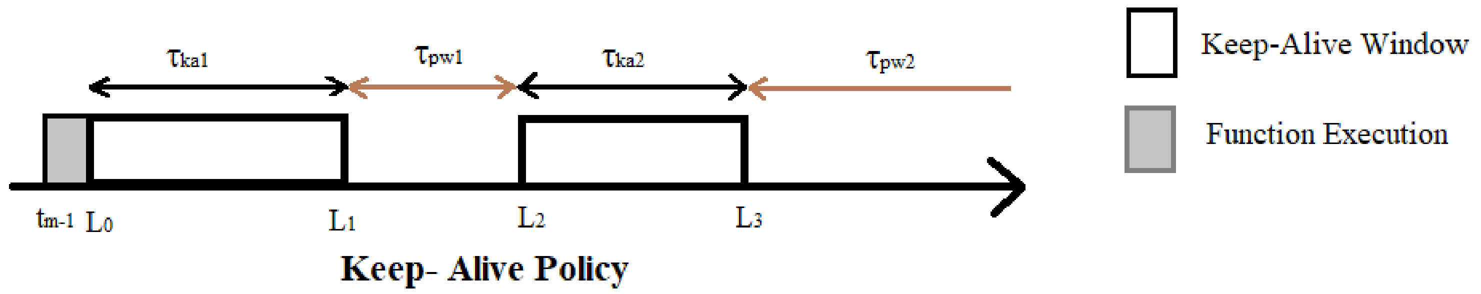
- **Keep-alive cache policy** = Sequence of keep-alive windows (and pre-warming windows) during which application image moved in & out of cache



Keep- Alive Policy

Keep-Alive Cache Policy

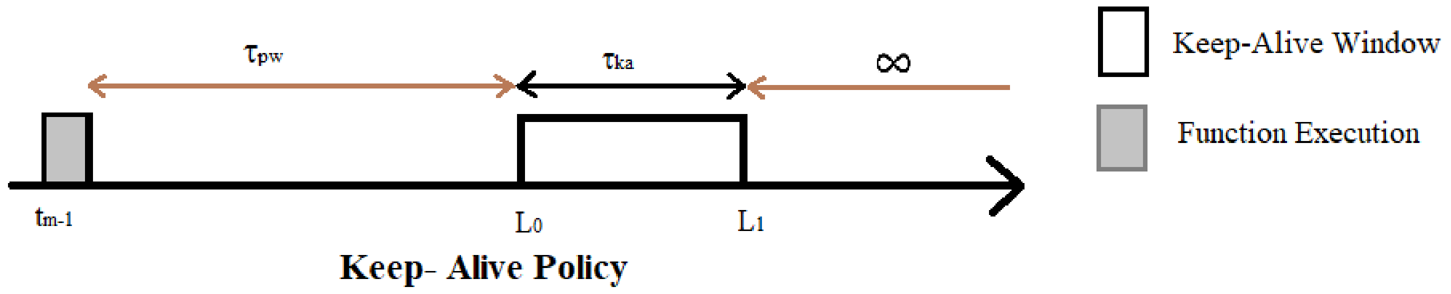
- **Keep-alive cache policy** = Sequence of keep-alive windows (and pre-warming windows) during which application image moved in & out of cache



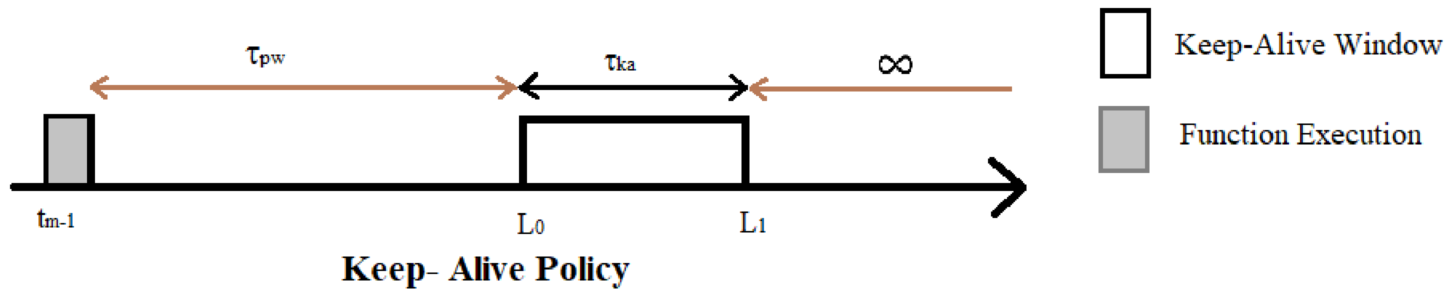
- **Goal**

  Optimal Keep-Alive Cache Policy with theoretical guarantees for serverless computing?

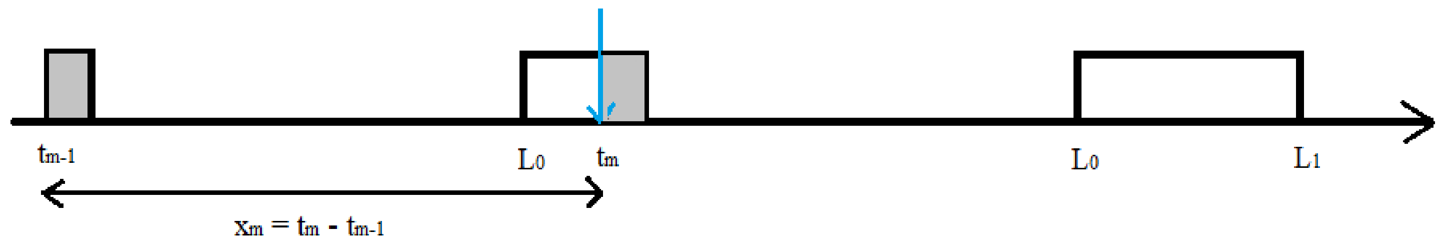
Single Window Policy



Single Window Policy



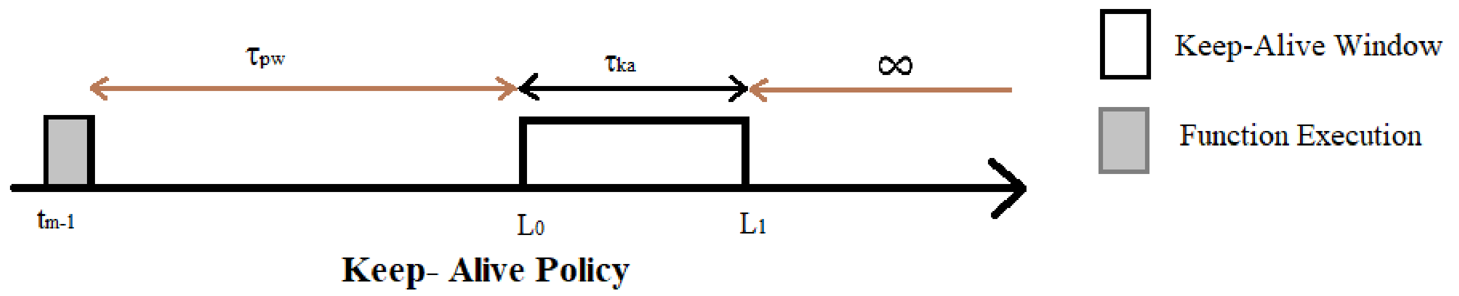
Keep- Alive Policy



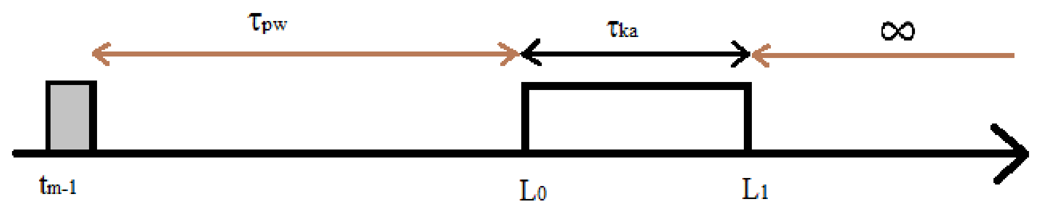
Warm Start

$$cost\left(x_m, \left(\tau_{pw}, \tau_{ka}\right), H\right) = c_p \cdot \left(x_m - \tau_{pw}\right)$$

Single Window Policy



Single Window Policy

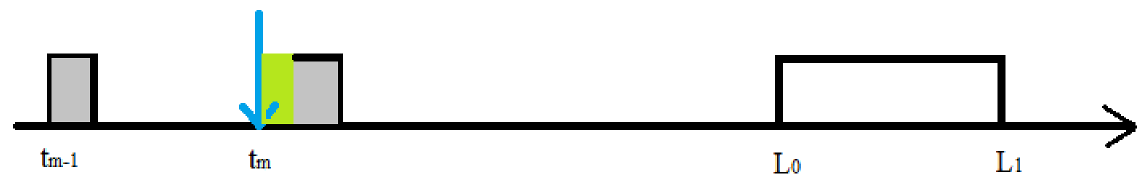


Keep- Alive Policy

☐ Keep-Alive Window

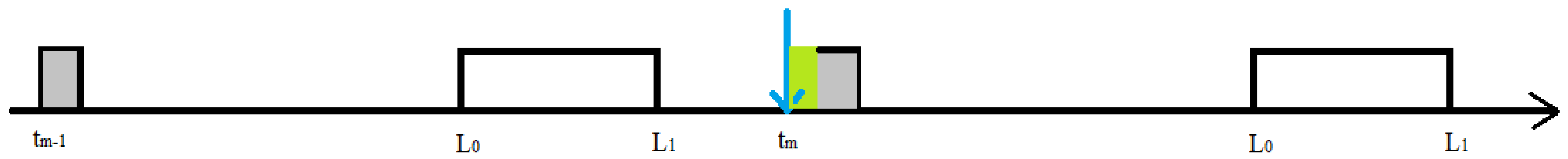
▨ Function Execution

🟩 Cold-Start Cost



$$cost(x_m, (\tau_{pw}, \tau_{ka}), H) = c_{cs}$$

Cold Start



Cold Start

$$cost(x_m, (\tau_{pw}, \tau_{ka}), H) = c_p \cdot \tau_{ka} + c_{cs}$$

Expected Cost of a Cache Policy

- Since x_m is not known to provider
  – use distribution of *x* over past arrivals *H*

Expected Cost of a Cache Policy

- Since x_m is not known to provider

  – use distribution of *x* over past arrivals *H*

- $$\mathbb{E}[cost(\pi(\cdot|H))] = \int_0^\infty \pi(x\,|H) \cdot g(x|H)dx + c_{cs}$$

$$\pi(\cdot|H) = \begin{cases} 1, & x \in [L_0, L_1] \cup [L_2, L_3] \cup \cdots \cup [L_{k-2}, L_{k-1}] \\ 0, & otherwise \end{cases}$$

$$g(x|H) = c_p\big(1 - F(x|H)\big) - c_{cs} \cdot f(x|H)$$

$g(x|H)$ = Instantaneous cost when the keep-alive window is active after x

Optimal Cache Policy

- Hazard rate $\lambda(x|H) = \frac{f(x|H)}{1-F(x|H)}$

  - Conditional probability that the arrival process dies in the next instant, given that it has survived up to *x*

Optimal Cache Policy

- Hazard rate $\lambda(x|H) = \frac{f(x|H)}{1-F(x|H)}$

  - Conditional probability that the arrival process dies in the next instant, given that it has survived up to x

- **Theorem**

  Points of sequence of keep-alive windows over an inter-arrival for optimal policy are at $0, \infty$ or solutions to equation $c_p - (c_{cs} \cdot \lambda(x|H)) = 0$ where the sign changes

Optimal Cache Policy

- Hazard rate $\lambda(x|H) = \frac{f(x|H)}{1-F(x|H)}$

  - Conditional probability that the arrival process dies in the next instant, given that it has survived up to *x*

- **Theorem**

  Points of sequence of keep-alive windows over an inter-arrival for optimal policy are at $0, \infty$ or solutions to equation $c_p - (c_{cs} \cdot \lambda(x|H)) = 0$ where the sign changes

Takeaway :

⇒ Hazard rate $\lambda(x|H)$ determines the characterization of optimal policy

Optimal Policy for Poisson Process

- Application invocations arrivals – Poisson process

  Probability density function $f(t) = \lambda \cdot e^{-\lambda t}, \;\; t \geq 0$

Optimal Policy for Poisson Process

- Application invocations arrivals – Poisson process

  Probability density function $f(t) = \lambda \cdot e^{-\lambda t}, \quad t \geq 0$

- **Optimal Cache Policy – Two possibilities**

Optimal Policy for Poisson Process

- ❑Application invocations arrivals – Poisson process

  Probability density function $f(t) = \lambda \cdot e^{-\lambda t}, \; t \geq 0$

- ❑**Optimal Cache Policy – Two possibilities**

1) Keep-alive window – always active $\tau_{ka} = \infty$, if $\frac{c_p}{c_{cs}} \leq \lambda$

   Expected cost $= \frac{c_p}{\lambda}$

2) Always encounter a cold start $\tau_{ka} = 0$, if $\frac{c_p}{c_{cs}} > \lambda$

   Expected cost $= c_{cs}$

Hawkes Process

❑ Hawkes Process

- “self-exciting” $\Rightarrow$ each arrival increases rate of future arrivals

Hawkes Process

- ❑ Hawkes Process
  - "self-exciting" $\Rightarrow$ each arrival increases rate of future arrivals

- ❑ Hawkes process with exponential excitation

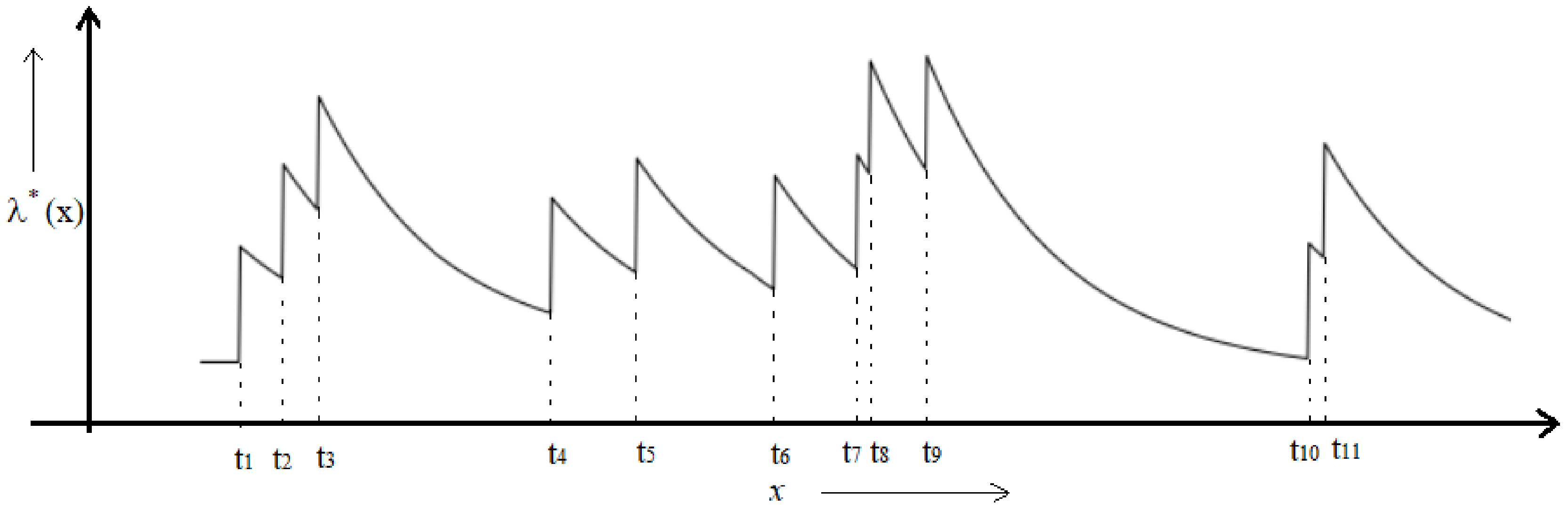
$$\lambda(x|H) = \frac{f(x|H)}{1 - F(x|H)} = \lambda_0 + \sum_{x>t_i} \alpha \cdot e^{-\beta(x - t_i)}$$

Hawkes Process

- Hawkes Process
  - "self-exciting" $\Rightarrow$ each arrival increases rate of future arrivals

- Hawkes process with exponential excitation

$$\lambda(x|H) = \frac{f(x|H)}{1 - F(x|H)} = \lambda_0 + \sum_{x>t_i} \alpha \cdot e^{-\beta(x - t_i)}$$



Optimal Policy for Hawkes Process

- Optimal cache policy has one of three forms with $\tau_{pw} = 0$, and τ_{ka} as follows:

Optimal Policy for Hawkes Process

- Optimal cache policy has one of three forms with $\tau_{pw} = 0$, and τ_{ka} as follows:

  1) $\tau_{ka} = \infty,$ when $\forall x, \quad \frac{c_p}{c_{cs}} < \lambda(x\,|H)$

Optimal Policy for Hawkes Process

- Optimal cache policy has one of three forms with $\tau_{pw} = 0$, and τ_{ka} as follows:

  1) $\tau_{ka} = \infty,$ when $\forall x,\quad \frac{c_p}{c_{cs}} < \lambda(x\,|H)$

  2) $\tau_{ka} = 0,$ when $\frac{c_p}{c_{cs}} > \lambda(x = 0\,|H)$

Optimal Policy for Hawkes Process

- Optimal cache policy has one of three forms with $\tau_{pw} = 0$, and τ_{ka} as follows:

1) $\tau_{ka} = \infty$, when $\forall x, \quad \frac{c_p}{c_{cs}} < \lambda(x \mid H)$

2) $\tau_{ka} = 0$, when $\frac{c_p}{c_{cs}} > \lambda(x = 0 \mid H)$

3) $\tau_{ka} = \frac{1}{\beta} \cdot \left(\log \alpha + \log\left(\sum_{j=1}^{m-1} e^{\beta(t_j - t_{m-1})}\right) - \log\left(\frac{c_p}{c_{cs}} - \lambda_0\right)\right)$, otherwise

Fixed Keep-Alive Policy

- Disadvantage of Optimal keep-alive policy

  - Need to recompute the policy after every arrival

- Simple, history independent policy

Fixed Keep-Alive Policy

- Disadvantage of Optimal keep-alive policy

  - Need to recompute the policy after every arrival

- Simple, history independent policy

- Fixed policy is a 2-factor approximation with respect to the optimal keep-alive policy

  where $\tau_{fixed} = \frac{c_{cs}}{c_p}$

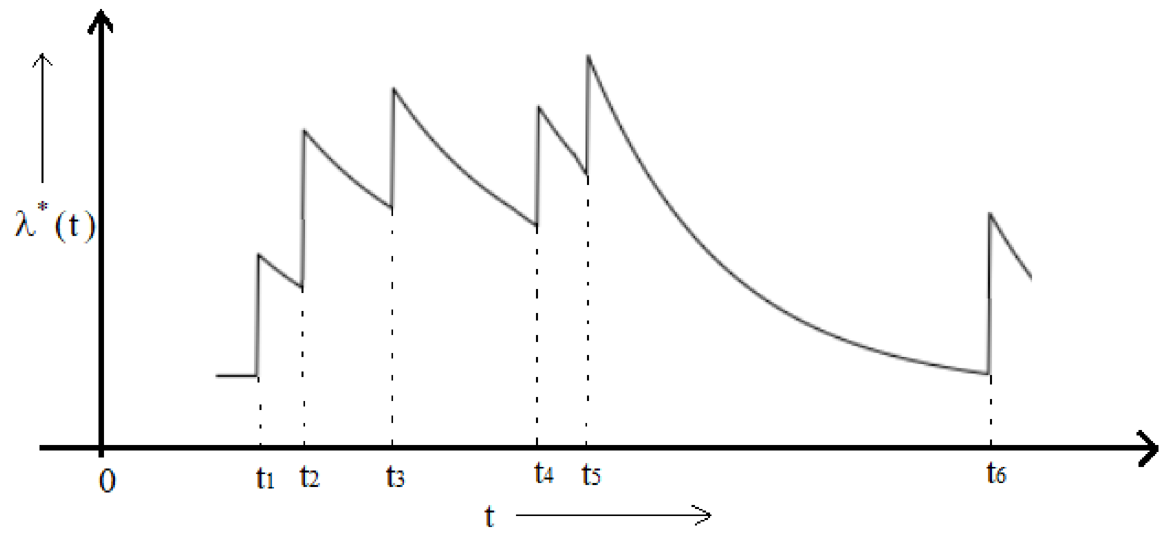
This result follows from classic Ski-rental problem

Optimized Time-to-Live (TTL) Policy

Optimized Time-to-Live (TTL) Policy

- Computation

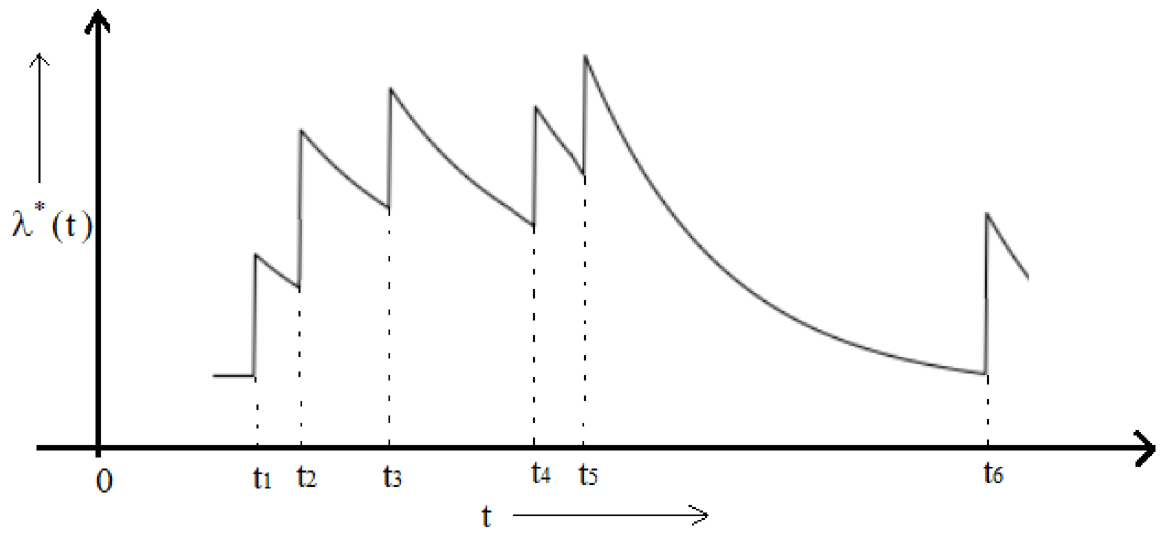
1)



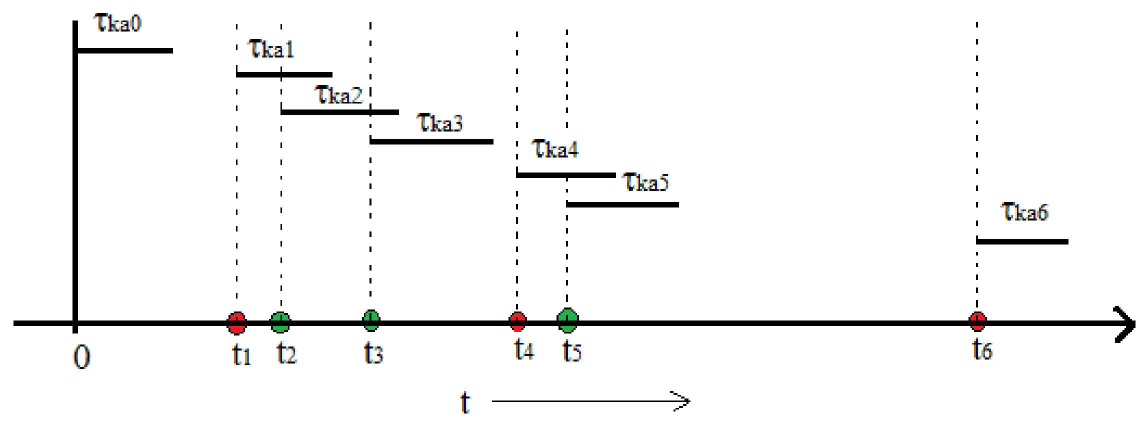
Optimized Time-to-Live (TTL) Policy

- Computation

1)



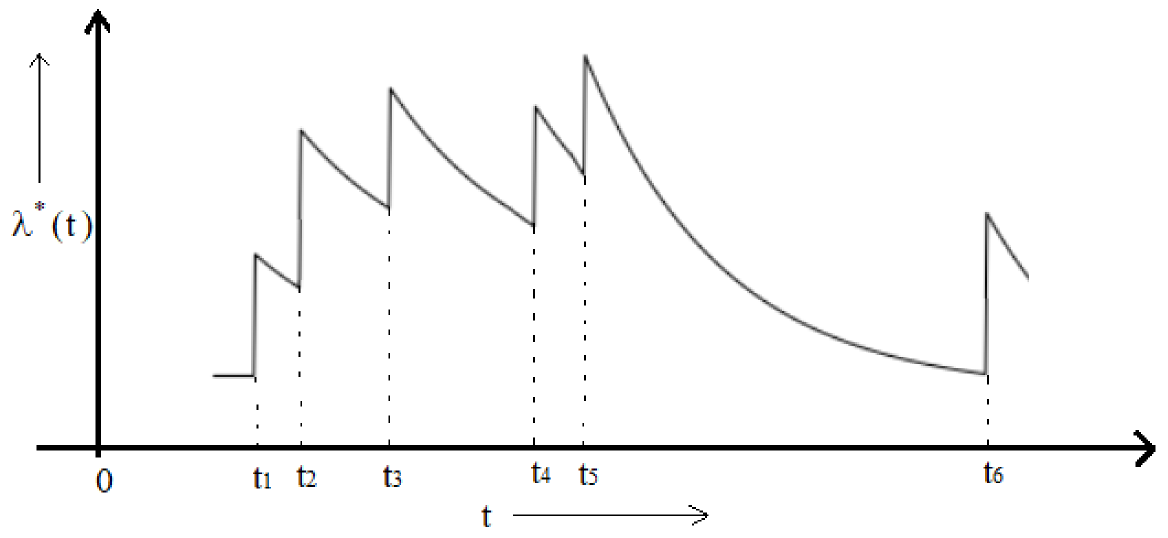
2)



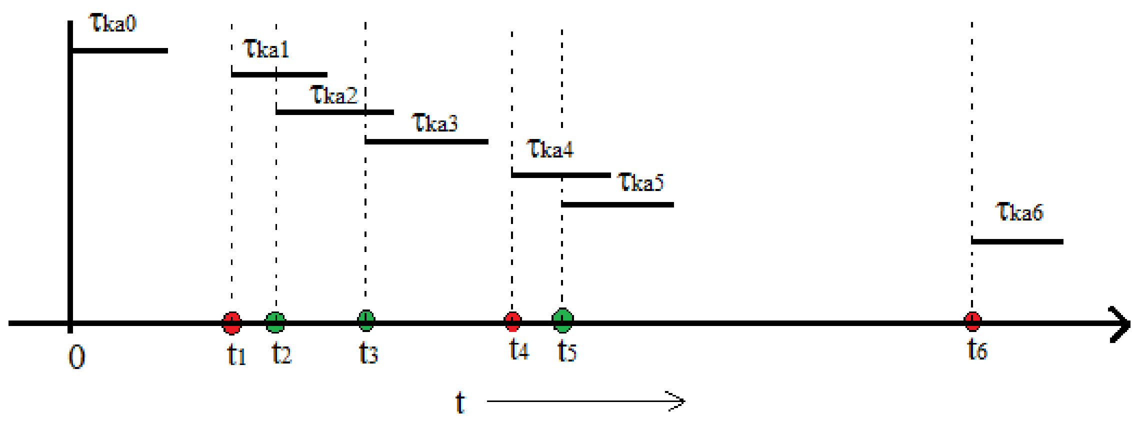
Optimized Time-to-Live (TTL) Policy

- Computation

1)



2)



3) Optimal TTL window
=Empirical average of
optimal windows

$$\underline{\tau_{ttl}} = (1/6)\, \Sigma_i\, \tau_{kai}$$

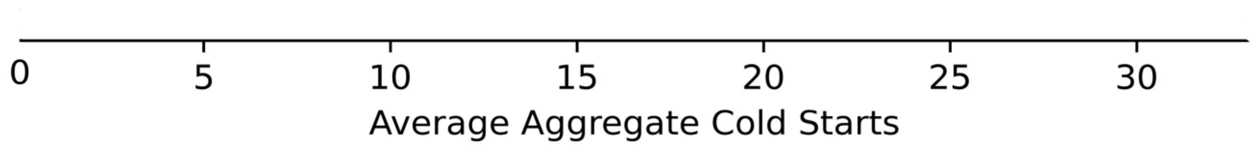
Azure Data Experiments

- Test Performance of Optimal and Optimized TTL policy
  for only Hawkes process fitted applications

Azure Data Experiments

- Test Performance of Optimal and Optimized TTL policy

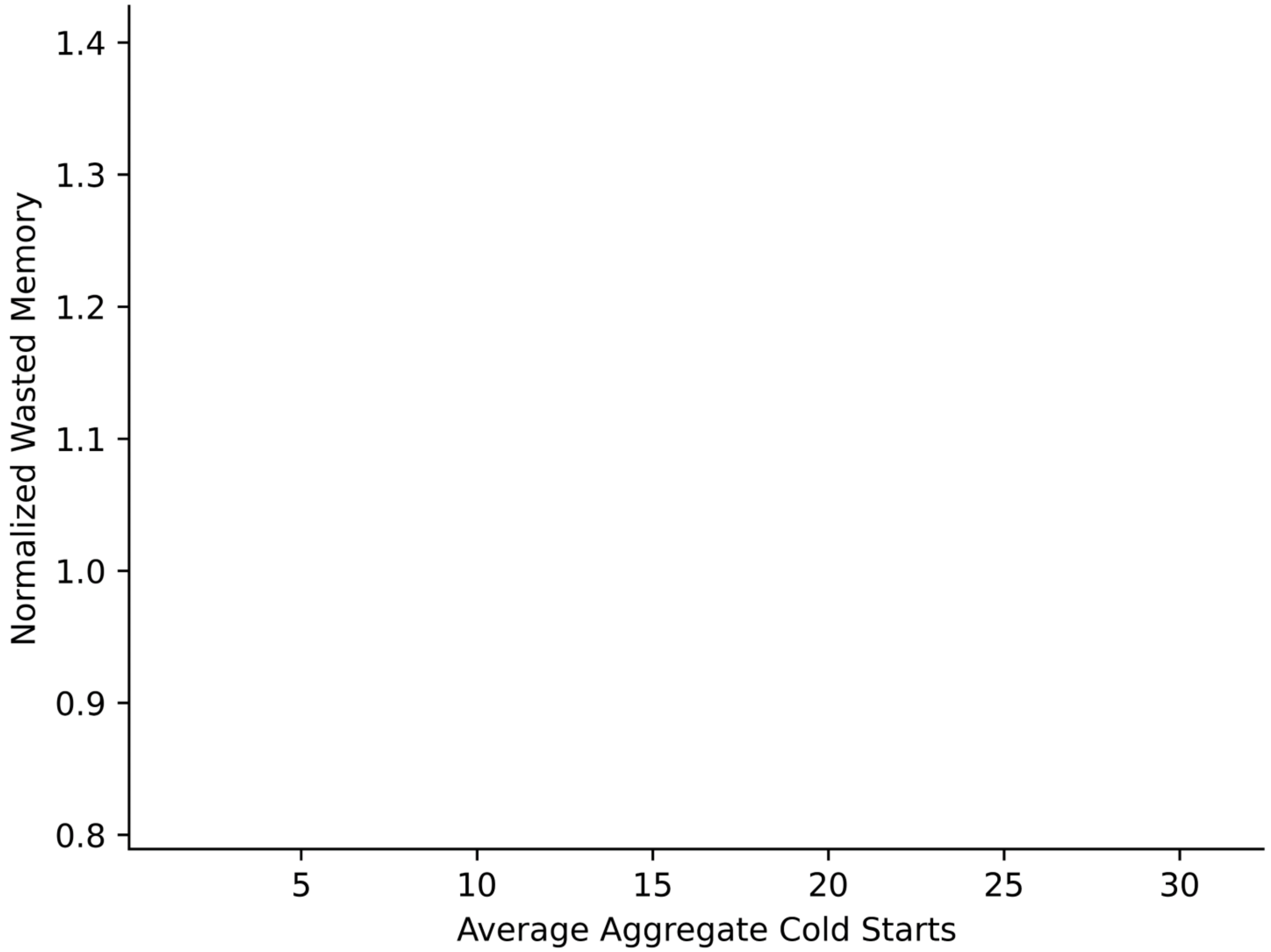
  for only Hawkes process fitted applications



Azure Data Experiments

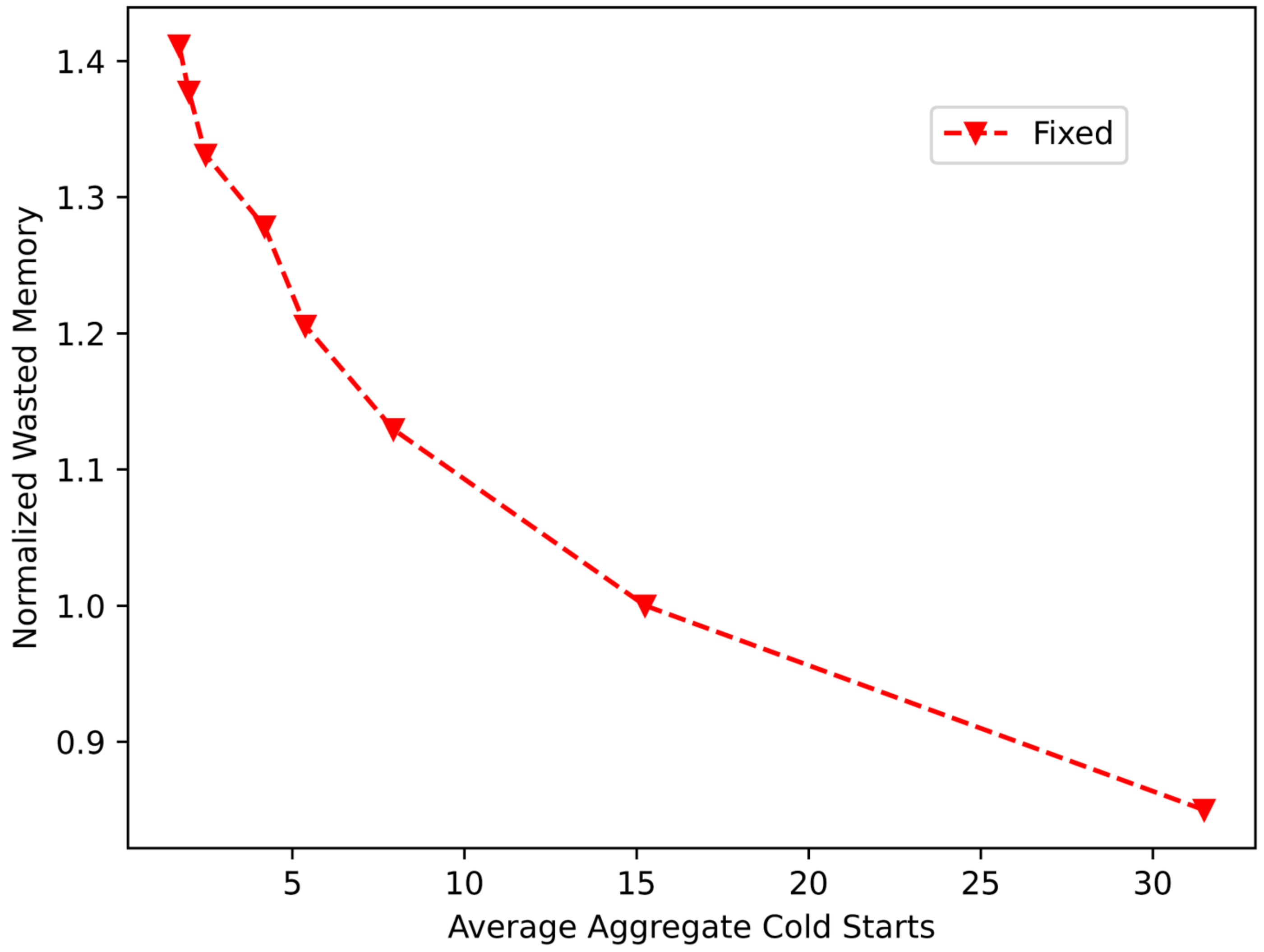
- Test Performance of Optimal and Optimized TTL policy

  for only Hawkes process fitted applications



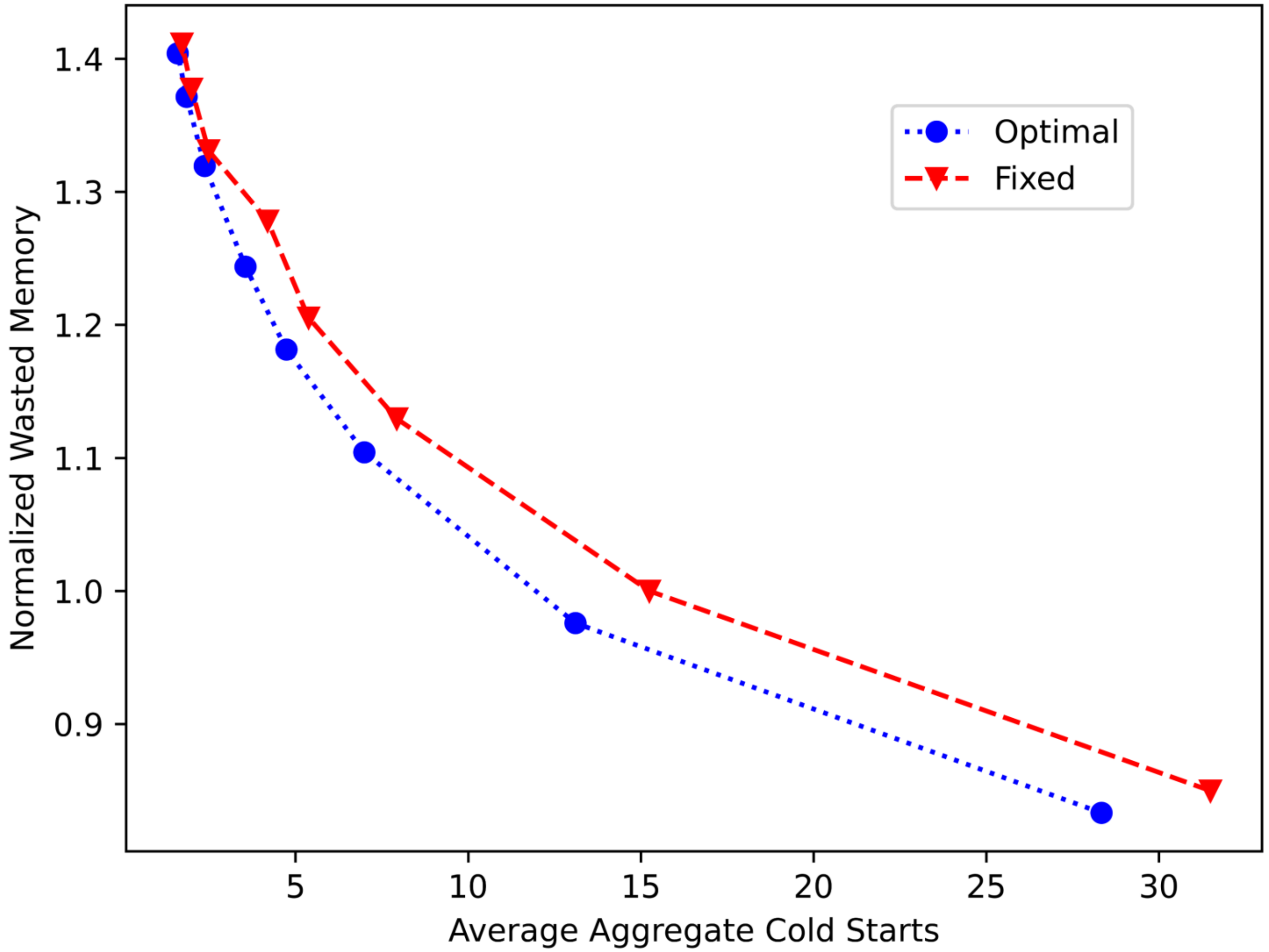
Azure Data Experiments

- Test Performance of Optimal and Optimized TTL policy
  for only Hawkes process fitted applications



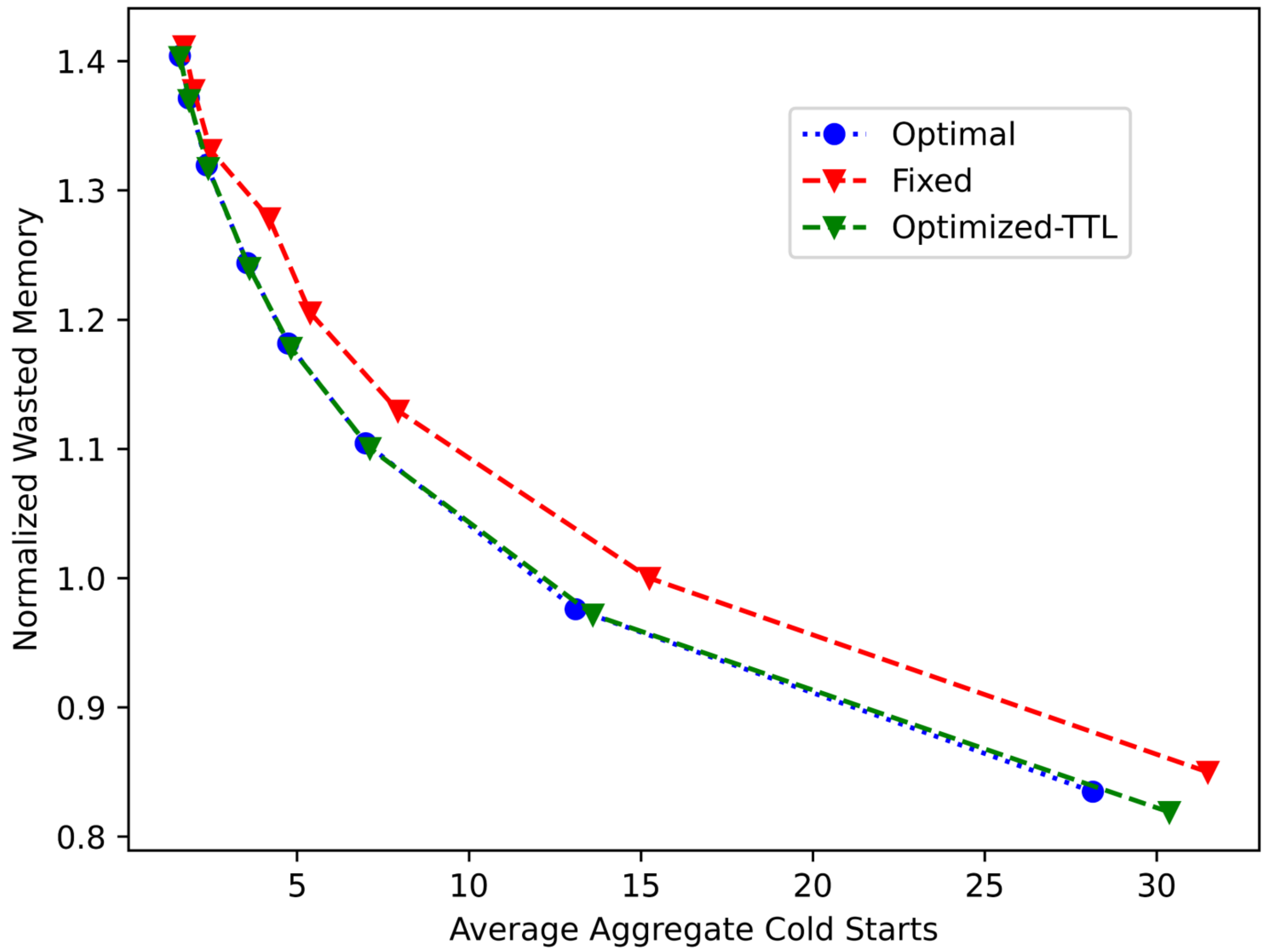
Azure Data Experiments

- Test Performance of Optimal and Optimized TTL policy for only Hawkes process fitted applications



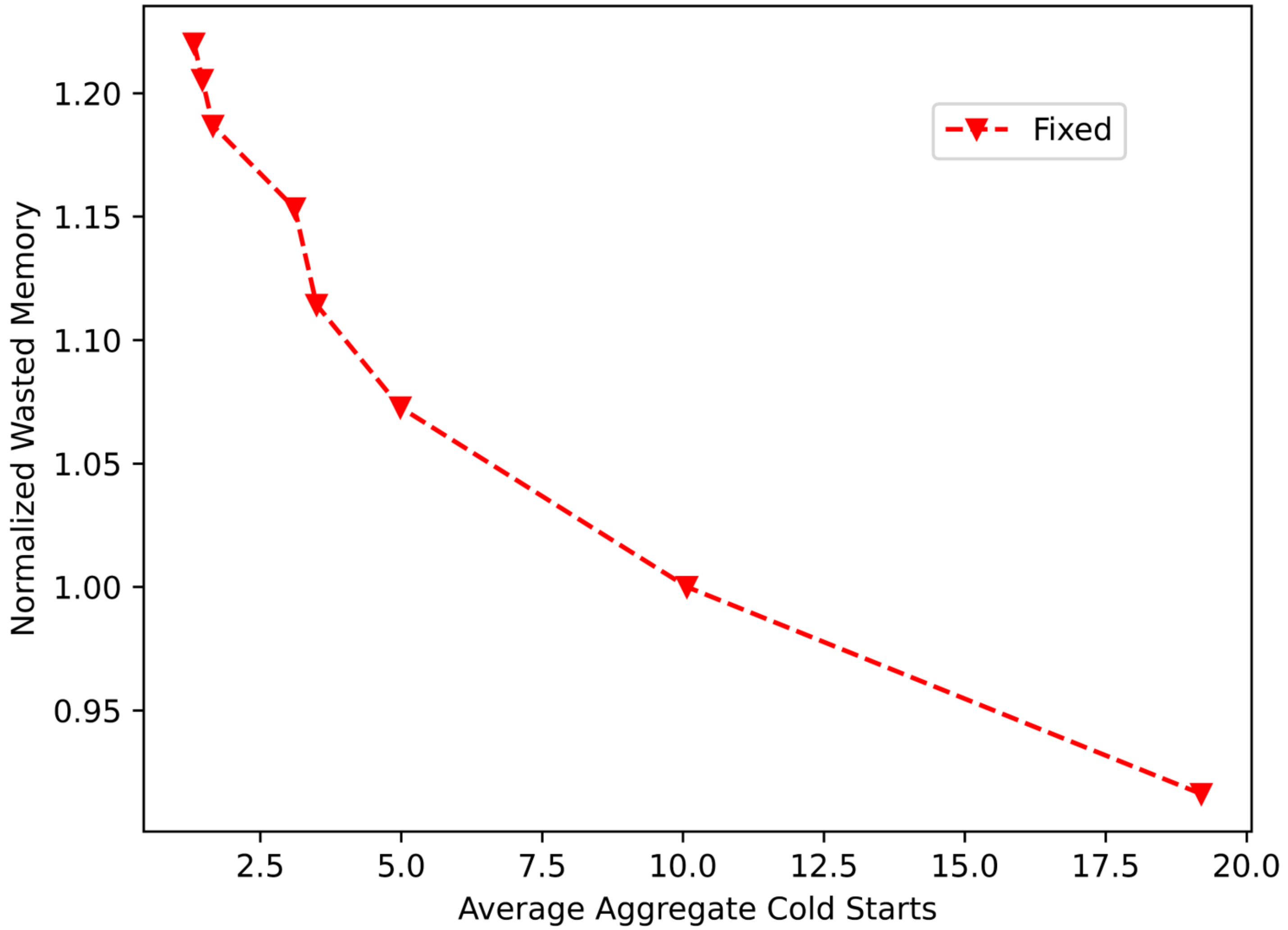
Azure Data Experiments

- Test Performance of Optimal and Optimized TTL policy
  for only Hawkes process fitted applications



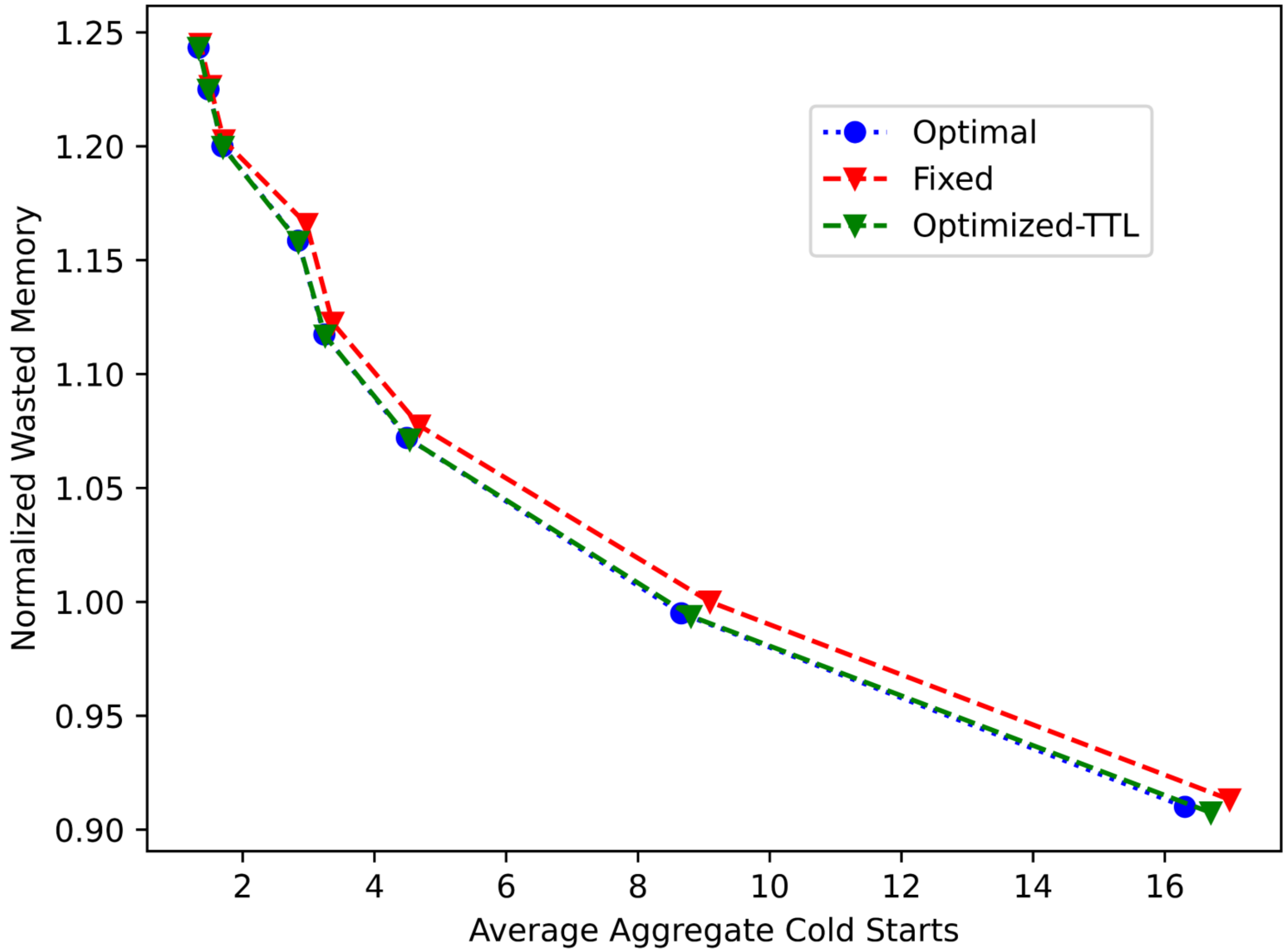
Azure Data Experiments

- Test Performance of Optimal and Optimized TTL policy on all applications (applied fixed policy on remaining non-fitted apps)



Azure Data Experiments

- Test Performance of Optimal and Optimized TTL policy on all applications (applied fixed policy on remaining non-fitted apps)



CONCLUSION

- Presented a model for keep-alive cache policies
  - captures trade-off between :- **cost of keeping objects in cache** *vs* **cost of cache misses**
  - applications in serverless computing

CONCLUSION

- ➢Presented a model for keep-alive cache policies
  - captures trade-off between :- **cost of keeping objects in cache** *vs* **cost of cache misses**
  - applications in serverless computing

- ➢ Characterized optimal cache policies, and optimized TTL policies for the Hawkes process

CONCLUSION

- ➢Presented a model for keep-alive cache policies
  - captures trade-off between :- **cost of keeping objects in cache** *vs* **cost of cache misses**
  - applications in serverless computing

- ➢ Characterized optimal cache policies, and optimized TTL policies for the Hawkes process

- ➢Evaluation on Azure data trace
  - our approach yields small, yet economically meaningful improvements at scale of datacenter

THANK YOU

EXTRA BACKUP SLIDES

Azure Data Experiments

- Experimented on Azure Data set

Mohammad Shahrad, Rodrigo Fonseca, Íñigo Goiri, Gohar Chaudhry, Paul Batum, Jason Cooke, Eduardo Laureano, Colby Tresness, Mark Russinovich, and Ricardo Bianchini. 2020. Serverless in the wild: characterizing and optimizing the serverless workload at a large cloud provider. In Proceedings of the 2020 USENIX Conference on Usenix Annual Technical Conference (USENIX ATC'20). USENIX Association, USA, Article 14, 205–218.

- Available at : https://github.com/Azure/AzurePublicDataset

- Traces collect invocation counts binned in 1-min intervals for   applications

- Evaluation metrics

  Wasted memory – Total wasted memory time normalized w.r.t. 10 min fixed policy

  Cold start- Average number of cold starts per application

Azure Data Experiments

- Fixed Keep-alive policy applied for 5, 10 , 20, 30, 45, 60, 90, and 120 minutes
- Optimal Policy applied with $c_p = 1$, $c_{cs} = 5, 10, 20, 30, 45, 60, 90, 120$ units

- Hawkes process parameters

  - Estimated via Minimum Negative Log-Likelihood

- Hawkes process parameters fitness checked

  -via Random time change theorem and KS test similarity measure

  - around 25% of applications pass the similarity test measure