Adaptive TTL Caches for Content Delivery

SANJAY SHAKKOTTAI
The University of Texas At Austin

WITH SOUMYA BASU, ADITYA SUNDARARAJAN, JAVAD GHADERI, AND RAMESH SITARAMAN
Cache Design in Content Delivery

• Millions of objects of multiple types, each type has own requirement

• 25 m objects of total size 25 TB and 504 m requests in a 9 day trace from one Akamai server (several thousand servers worldwide)

• Correlated arrivals with complex inter-arrival distribution

• Significant fraction of rare arrivals, e.g. objects with a few requests

• Guarantee hit rate for QoS and decrease cache size for reducing cost
Time-to-Live (TTL) Cache

• Popular caching scheme with good theoretical guarantees

• **Algorithm:** Fixed TTL value $\theta$ for all objects
  
  • Cache miss: Object not in cache
    • Fetch object from server
  
  • Cache object with TTL $\theta$
  
  • Cache hit: Object present in cache
    • Reset TTL of the object to $\theta$
  
  • On timer expiry: Evict object from cache
Deficiencies in Existing Approach

- A popular approach in designing cache is model based
  - Assume an underlying content request model
  - Tractable approximations, e.g. Che’s formula/approximation
  - Design cache using analytical expression of hit rate and size

\[
\text{Hit rate} = 1 - \exp(-\lambda_m T_c) \\
\text{Cache size} = \sum_m (1 - \exp(-\lambda_m T_c))
\]

- Approximate request model lead to error in analytical expression
- Tractable approximations difficult for complex caching models
Deficiencies in Existing Approach: Fixed TTL

- Compute TTL value for hit rate and estimate size for the TTL value
- Performance of TTL cache with approx. on a 9 day long Akamai trace
Dynamic TTL Adaptation

- **Goal:** Achieve target hit rate $h^*$

- **Adaptation of TTL:** $\text{TTL} \uparrow \Rightarrow \text{Hit rate} \uparrow$, $\text{TTL} \downarrow \Rightarrow \text{Hit rate} \downarrow$
  - Cache hit on $l^{th}$ arrival
    - Decrease TTL: $\theta(l) = \left( \theta(l - 1) - \frac{1}{l} (1 - h^*) \right)^+$
  - Cache miss on $l^{th}$ arrival
    - Increase TTL: $\theta(l) = \theta(l - 1) + \frac{1}{l} (h^*)$

**Converges to TTL $\theta^*$ providing hit rate $h^*$ for stationary traffic**
Drawback: Transient Arrivals

- Rare objects have negligible contribution to hit rate
- 70% of all objects, 10% of the traffic, are requested only once
- Size occupied by rare objects = (TTL value) \times (Arrival rate)
- Significant number of rare objects leads to large cache wastage

Which objects are rare?
How to filter rare objects?
Filtering Rare objects: Shadow Cache

- Separate shadow cache along with main TTL cache – Deep cache
- On a new arrival cache object **label** in shadow cache with TTL $\theta$
- Upon a hit in shadow cache object enters deep cache with TTL $\theta$
- Similar to LRU-2Q or Bloom filter + LRU

**Result:** Dynamic TTL with $\theta(l)$ projected on $[0,L]$ converges to TTL $\theta^*$ in the presence of rare object traffic if $h^*$ is feasible (w.r.t. L).
Infrequent Bursty Arrivals

- Filtering differentiates rare and popular objects to reduce cache size
- **Problem**: Infrequent bursty arrivals suffer from filtering

Example: Bursty requests with typically 4 arrivals, and bursts separated in time beyond TTL.

Can we capture multi-time-scale dynamics?
Filtering TTL Cache

- A filtering TTL cache has three parts:
  - Deep Cache (DC), Shadow Cache (SdC), and **Shallow Cache (SC)**
  - Deep Cache captures the popular objects with TTL $\theta$
  - Shadow Cache filters out rare objects with TTL $\theta$
  - **Shallow Cache** captures infrequent burstiness with TTL $\psi < \theta$
- **Joint adaptation of $\psi$ and $\theta$ to meet hit rate and size targets**
Filtering TTL Cache (f-TTL): Deep Cache Hit

Object enters Deep Cache

- Object present in DC
- TTL is reset

Cache Hit in Deep Cache
Filtering TTL Cache (f-TTL): Cache Miss

- Object not present
- Label not present
- Fetch from server

Cache Miss
Filtering TTL Cache (f-TTL): Shallow Cache Hit

- **Shallow Cache Hit**
  - Object present in SC
  - Label present in SdC
  - Move object from SC to DC
Filtering TTL Cache (f-TTL): Shadow Cache Hit

- **Deep Cache**
- **Shadow Cache**
  - $\psi^*$
  - $\theta^*$
  - Object enters Deep Cache
  - Label evicted
- **Shallow Cache**
  - $\theta^*$
  - $\psi^*$
  - Object enters Deep Cache
  - Label evicted

- **Shadow Cache Hit**
  - **Object not present**
  - **Label present in SdC**
  - **Fetch from server**
  - **Not counted as cache hit**
Filtering TTL Cache (f-TTL): Objectives

- Achieve target hit rate $h^*$
- Achieve size $\lambda s^*$ ($\lambda$ is arrival rate)
- $s^*$ is average (over time and objects) occupancy duration of a request
- Occupancy duration is duration from arrival to eviction or TTL reset
Filtering TTL Cache (f-TTL): Adaptation

• The adaptation of $\theta$ is the same as the dynamic TTL cache
  • Increase $\theta$ on cache miss and shadow cache hit
  • Decrease $\theta$ on shallow or deep cache hit

• The adaptation of $\psi$ is to meet $s^*$ (size target == occupancy duration)
  • Estimate the occupancy duration $s_{est}(l)$
  • Increase $\psi$ if $s^* > s_{est}(l)$ and decrease otherwise
F-TTL Cache: Time Scale Separation

• Faster adaptation of \( \theta \) compared to \( \psi \) — Time scale separation
  • Deep Cache Adaptation: In faster time scale, \( \psi \) (shallow cache) is quasi-static while \( \theta \) adjusts to attain hit rate
  • Shallow Cache Adaptation: In slower time scale, \( \psi \) adapts to attain size

\[
\theta(l) = \left( \theta(l-1) - \frac{1}{l^\alpha} (Y(l) - h^*) \right)^+
\]
\[
\alpha \in (0.5,1) \text{ and } Y(l) = 1 \text{ if Deep cache/ Shallow cache hit, 0 o/w }
\]

\[
\psi(l) = \min\{\theta(l), \left( \psi(l-1) + \frac{1}{l} (s^* - s_{est}(l)) \right)^+ \}
\]
Filtering TTL Cache (f-TTL): Estimating $s_{est}$

- At adaptation instance, checking the size is misleading
- Remaining TTL timer value for request object $\phi(l)$
- Estimating the occupancy duration
  - Deep/Shallow cache hit: $s_{est}(l) = \theta(l - 1) - \phi(l)$
  - Shadow cache hit: $s_{est}(l) = \theta(l - 1)$
  - Cache miss: $s_{est}(l) = \psi(l - 1)$

- Size 3 at the first arrival instance
- Size changes between two arrival instances
Truncating Parameters – Towards Actor-Critic

• TTL value $\theta$ may become unbounded in presence of rare objects
  • $\alpha\%$ of the traffic from rare objects $\Rightarrow h^* > (100 - \alpha)\%$ infeasible
  • We need to project $\theta(l)$ on [0,L] with large but finite L

**Projection for robustness against rare objects**

• **Problem:** Suppose the operator sets attainable $h^*$ but sets $s^*$ too small
  • We can argue that $(\theta(l), \psi(l)) \rightarrow (L,0)$, resulting in an achieved hit rate of $h < h^*$

**Approach:** Fictitious dynamics through an actor-critic algorithm
An Actor-Critic Algorithm

- **Approach:** Separate observation and action – **Actor-critic algorithm**
- **Critic parameters** $\vartheta$ and $\delta$ record hit rate and size rate, resp.
- **Actor parameters** $\theta$ and $\psi$ are used in the f-TTL algorithm
- Actors are functions of critics: $\theta = L \vartheta$ and $\psi = L \Gamma_{\epsilon}(\vartheta, \delta)$
  - Saturation function $\Gamma_{\epsilon}(\vartheta, \delta) = \begin{cases} 
  \delta, & \text{if } \vartheta \leq 1 - 1.5\epsilon \\
  \vartheta, & \text{if } \vartheta \geq 1 - 0.5\epsilon \\
  \text{interpolation, } o/w 
  \end{cases}$
- Time scale separation in $\vartheta$ and $\delta$ adaptation to ensure convergence
Details: Actor Critic Adaptation

• **Cache hit:** Let the object be of type $t$, with size $w$ and at the time of request its TTL timer be $\phi > 0$
  
  - $\vartheta(l) = \max \left( 0, \vartheta(l - 1) - \frac{1}{l^\alpha} (1 - h^*) \right)$, $\alpha \in (0.5, 1)$
  
  - $\delta(l) = \min \left( 1, \max \left( 0, \vartheta_t(l - 1) + \frac{1}{l} (s^* - \theta(l - 1) + \phi) \right) \right)$

• **Shadow hit or miss:** Let the object be of type $t$, with size $w$
  
  - $\vartheta(l) = \min \left( 1, \vartheta(l - 1) + \frac{1}{l^\alpha} h^* \right)$
  
  - $\delta(l) = \min \left( 1, \max \left( 0, \vartheta(l - 1) + \frac{1}{l} (s^* - \psi(l - 1)) \right) \right)$
Filtering TTL Cache (f-TTL): Guarantees

• Bursty arrivals, and rare objects following a ‘Rarity condition’ (objects with asymptotic zero hit rate) present

• Two-timescale stochastic approximation based proof technique (using methods in Borkar 97; conditions from Kushner-Clark, Kushner-Yin)

**Filtering TTL with actor-critic adaptation converges to \((\theta^*, \delta^*)\) a.a.s.**

• If \(h^*\) feasible with threshold \(L\) then \(h^*\) is achieved
• Either achieves size \(\lambda s\) where \(s \leq s^*\)
• Or collapses to a pure shadow cache mode, i.e. \(\psi = 0\)

• Generalize to multiple types with different \(h^*\) and \(s^*\)
• Generalize to different object sizes with modified adaptation
Performance on Akamai Traces

• Modified Algorithm with constant step size adaptation
• 9 day trace with 504m requests from 25m distinct objects
• Average error for hit rate targets 0.4, 0.5, 0.6, 0.7, and 0.8 : < 1.3%

Fig 1: d-TTL Convergence Plot

Fig 2: f-TTL Convergence Plot
Performance on Akamai Traces

• Variable sized version of the d-TTL and f-TTL Algorithms
• Size rate target = 50% of d-TTL : Size rate achieved = 49% of d-TTL

Fig 3: Hit rate vs Average Cache Size curve
Thank You