Escalando caching em redes orientadas a conteúdo via mecanismos de histerese

Gabriel Mendonça[1] **Guilherme Domingues**[2]
Edmundo de Souza e Silva[1] **Rosa Leão**[1] **Daniel Menasché**[1]

[1]Universidade Federal do Rio de Janeiro (UFRJ), Rio de Janeiro, RJ

[2]Universidade do Estado do Rio de Janeiro (UERJ), Friburgo, RJ

{gabriel,guilhdom,edmundo,rosam,sadoc}@land.ufrj.br

Resumo. Caching *é um dos elementos fundamentais de sistemas em redes. Filtrando requisições para os custodiantes, caches reduzem a banda na rede e o atraso para os clientes. Entretanto, parametrizar políticas de cache do tipo* time-to-live *(TTL) pode não ser uma tarefa fácil. Nossas contribuições nesse trabalho são: 1) fórmulas fechadas para medidas de interesse em caches TTL baseados em RC, que permitem determinar de forma simples parâmetros ótimos para uma cache; 2) análise de desempenho a partir de* traces *reais; 3) uso de técnicas de aprendizado por máquina para simplificação das fórmulas obtidas, facilitando a parametrização e interpretação de resultados*

Abstract. *Caching is a fundamental element of networking systems since the early days of the Internet. By filtering requests towards custodians, caches reduce the bandwidth required by the latter and the delay experienced by clients. The requests which are not served by a cache, in turn, comprise its miss stream, which is a smoothed version of the stream of request arrivals. In this paper, we propose novel mechanisms to leverage hysteresis (i.e., smoothing) on cache evictions and insertions. The proposed solutions extend TTL-like mechanisms, and rely on two knobs to tune the time between insertions and evictions given a target hit rate. We show the particular improvement of the two thresholds strategy in reducing download times, making the system more predictable and accounting for different costs associated with object retrieval.*

1. Introdução

Caches aumentam o desempenho e a escalabilidade de sistemas de distribuição de conteúdo, e tal fato é bem conhecido há décadas. Recentemente, o uso de caches vem sendo advogado como estrutura fundamental na arquitetura da Internet, de forma a aproveitar a disponibilidade de memórias cada vez mais baratas em roteadores. Tais roteadores dotados de caches, que passam a chamar-se "cache-routers", trazem benefícios para o sistema em termos de diminuição de atrasos para os clientes e redução de banda exigida pelos servidores.

Quando cache-routers são interconectados, uma série de novas questões surgem. Por exemplo, é sabido que a interconexão de cache-routers pode gerar efeitos transientes na rede que se propagam a médio e longo prazo [Rosensweig et al., 2013]. Além disso, políticas clássicas de cache, como o LRU, não são ótimas neste contexto. Surge, daí,

a motivação para o estudo de novas formas de otimizar conjuntamente o roteamento e armazenamento de conteúdo, levando em conta a interconexão das caches. Tais mecanismos, no entanto, embora simples, muitas vezes fazem uso de parâmetros que precisam ser ajustados dinamicamente. Em larga escala, tais ajustes podem ser desafiadores.

Neste trabalho, propomos abordagens para aumentar a escalabilidade de políticas de parametrização de mecanismos de caching e roteamento. Tais abordagens baseiam-se numa combinação entre modelos de avaliação de desempenho e aprendizado por máquina. As ideias básicas consistem em 1) gerar amostras de métricas de desempenho usando um modelo de avaliação de desempenho clássico e 2) a partir de tais amostras, treinar um modelo de aprendizado por máquina, para obter expressões mais simples e/ou agregados de parâmetros que simplifiquem o ajuste dos mesmos.

Para políticas de caching, buscamos abordagens simples e flexíveis, e focamos em uma classe de algoritmos similares às caches do tipo *time-to-live* (TTL) denominadas *reinforced counters* (RC) [Domingues et al., 2017]. A política de caching usando contadores RC é facilmente extensível para levar em conta histerese e para levar em conta diferentes custos associados a diferentes conteúdos. Ela basicamente consiste em associar um contador a cada conteúdo requisitado, incrementá-lo a cada requisição e decrementá-lo em intervalos regulares de tempo. Quando o contador atinge um limiar inferior L o conteúdo é removido e quando atinge um limiar superior K o conteúdo é inserido. Um dos desafios consiste em determinar os valores ótimos de K e L, possivelmente por conteúdo. Para tal, adotamos a abordagem combinada de avaliação de desempenho e aprendizado por máquina descrita acima.

Em resumo, a principal contribuição deste trabalho consiste numa abordagem para aumentar a escalabilidade de parametrização de mecanismos de *caching*. Combinando a solução analítica de modelos de desempenho e aprendizado por máquina, propomos uma nova abordagem para determinação de parâmetros de mecanismos de *caching*. Em particular, a partir do modelo analítico proposto, obtemos fórmulas fechadas para medidas de interesse que permitem determinar de forma simples parâmetros ótimos para uma cache (Seção 3). Tais resultados tornaram possível o estudo de sistemas de forma escalável, usando *traces* reais (Seção 4). Resultados preliminares indicam que a flexibilidade ganha com o uso de caches RC com histerese traduz-se em desempenho superior com relação a outros modelos na literatura. Finalmente, o uso de técnicas de aprendizado por máquina permitiu a simplificação das fórmulas obtidas, e facilitou a parametrização e interpretação de resultados (Seção 5). Assim, de forma mais ampla, a combinação de métodos de avaliação de desempenho e aprendizado por máquina constitui também uma contribuição metodológica deste trabalho para a análise de sistemas de *caching*.

2. Por que histerese?

Nesta seção, consideramos diferentes aspectos relativos à importância de histerese para sistemas de cache. Iniciamos descrevendo brevemente os mecanismos de histerese em consideração.

Como mostra a Figura 1(a), quando um conteúdo está disponível, ele é servido pela cache e gera um cache *hit*. De tempos em tempos, um contador associado a cada conteúdo armazenado é decrementando – quando o RC é decrementado para L, o conteúdo é removido (Figura 1(b)). Por outro lado, quando o conteúdo é requisitado e

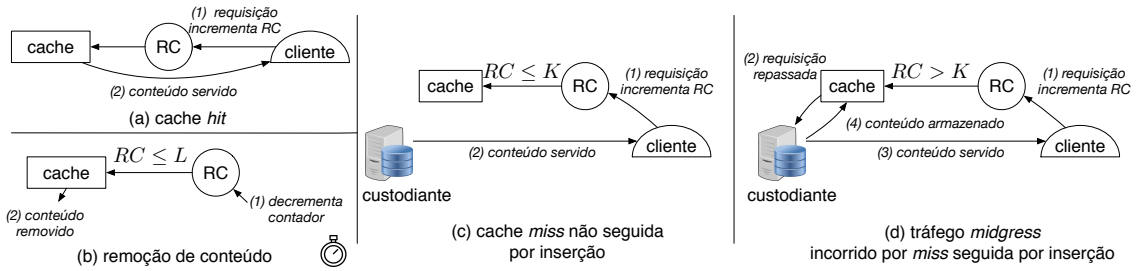


Figura 1. Reduzindo tráfego midgress pelo uso dos *reinforced counters*.

está indisponível, o RC é incrementado e, se permanecer menor que K, o conteúdo é servido pelo custodiante – *miss* não é seguido por uma inserção (Fig. 1(c)); caso contrário, conteúdo é servido e colocado em cache – *miss* é seguido por inserção (Fig. 1(d)).

Evitar distúrbios devido aos *one-hit wonders*: *One-hit wonders* são conteúdos que são acessados apenas uma vez e nunca mais. Tais conteúdos podem causar *churn* (distúrbios) significativos em sistemas de cache, sem benefícios. Filtrando o fluxo de entrada de requisições e prevenindo o *caching* de conteúdo que provavelmente será requisitado apenas uma vez, o conjunto de conteúdos armazenados na cache torna-se mais estável e menos sujeito a distúrbios transientes.

De forma mais geral, levar em conta a dinamicidade e a não-estacionariedade do catálogo de conteúdos e requisições é um dos desafios chaves associados a sistemas de cache. Achar um bom balanço entre filtragem de *one-hit wonders* e capacidade de reação à variações na demanda é um típico compromisso (*trade-off*) fundamental que precisa ser levado em conta por mecanismos de cache.

Aumentar o tempo de vida das memórias flash e diminuir o número de escritas em disco: Diminuindo o *churn* pode-se diminuir a taxa de escritas ao disco, o que pode traduzir-se em redução de atrasos. Além disso, se o sistema de cache for implementado via memórias flash, reduzir o número de escritas significa aumentar o tempo de vida das memórias. Memórias flash estão entre as memórias mais disponíveis atualmente. Elas são consideradas para implementação de caches em estudos de viabilidade de ICN, como por exemplo [Perino e Varvello, 2011].

Diminuir o tráfego *midgress* e a carga nos canais da rede: O tráfego em uma rede de distribuição de conteúdo (CDN) é dividido entre tráfego de ponta (*edge*), *midgress* e custodiante (*custodian*). O tráfego de ponta ocorre entre clientes e caches, enquanto que o tráfego custodiante é o tráfego que sai do custodiante. Todo o restante do tráfego é *midgress*. Esse tráfego é um *overhead*, que deve ser reduzido.

Tanto *misses* como inserções podem induzir tráfego *midgress*. Entretanto, *misses* seguidas por inserções podem gerar mais tráfego que *misses* não seguidas por inserções. Em particular, se o caminho do custodiante para o cliente for diferente daquele entre o custodiante e a cache, uma *miss* seguida por uma inserção pode gerar o dobro de fluxos, para alimentar os clientes e a cache, em comparação com uma *miss* que não seja seguida por uma inserção (vide Figura 1). Assim, reduzindo o *churn* um efeito colateral positivo consiste na redução do tráfego *midgress*.

Aumentar previsibilidade do sistema: Dado um valor alvo de cache *hit rate*, os

parâmetros do sistema possuem dimensões que podem ser ajustadas para atender ao *hit rate* alvo. Dentre estas dimensões, o tempo entre a remoção do conteúdo e a sua (possível) reinserção é um destes parâmetros. Mantendo conteúdos por mais tempo na cache, produzimos um sistema mais estável e predizível, em detrimento de tornar-se menos reativo a variações na demanda. A histerese exerce um papel importante nesse aspecto. Posicionamento estático é ótimo para cargas estacionárias [Tatarinov et al., 1997], mas subótimo caso contrário. Por outro lado, outros sistemas existentes reagem mais rápido a mudanças na carga, mas são mais difíceis de predizer [Carofiglio et al., 2016].

3. Análise dos *reinforced counters* (RC)

Nesta seção, estudamos políticas de inserção e remoção de conteúdo da cache, sob a hipótese de que as dinâmicas de cada conteúdo são desacopladas entre conteúdos. Dinâmicas desacopladas permitem análises tratáveis, e podem ser usadas para aproximar o desempenho de sistemas com capacidades fixas. Elas também são úteis no contexto de sistemas como caches de DNS e Amazon ElastiCache [Amazon, 2017].

Nas políticas introduzidas nesta seção, o *valor esperado* do número de itens na cache pode ser controlado. Seja π_c a probabilidade de que o conteúdo c esteja armazenado na cache. Dada uma coleção de N conteúdos, o valor esperado do número de conteúdos armazenados é $\sum_{i=1}^{N} \pi_i$, que pode ser controlado ou limitado superiormente, de acordo com as necessidades dos usuários.

Dado um conteúdo, assumimos que π pode ser ajustado usando *reinforced counters* (RC). Primeiro consideramos histerese nas inserções, e em seguida consideramos mecanismos que também permitem histerese nas remoções.

3.1. Caches TTL

A seguir, descrevemos brevemente o comportamento de caches *time to live* (caches TTL) [Dehghan et al., 2016, Domingues et al., 2017]. As caches baseados em *reinforced counters* (RC) são variantes das TTL.

Caches do tipo *time-to-live* (TTL) são mecanismos simples, flexíveis e robustos para armazenamento de conteúdo. Dentre suas inúmeras qualidades, caches TTL podem replicar o comportamento de caches padrão, do tipo LRU e FIFO. Uma cache TTL associa, a cada conteúdo i, um contador T_i. O contador é decrementado a cada unidade de tempo. Quando o contador de um determinado conteúdo atinge o valor zero, esse conteúdo é removido da cache. Quando o conteúdo é requisitado novamente, o conteúdo é re-inserido e o contador é então reiniciado ao valor T_i. Esse comportamento corresponde ao TTL sem *reset*.[1] O TTL com *reset* é similar. Entretanto, neste último caso toda vez que o conteúdo é requisitado, o contador é novamente setado para T_i, independente de o conteúdo estar ou não presente na cache.

3.2. RC com único limiar e histerese na inserção

A cada conteúdo associamos um contador por reforço RC [Domingues et al., 2017, Domingues et al., 2015], que é incrementado a cada vez que o conteúdo é requisitado,

[1]Note que numa cache TTL sem *reset*, o contador só é reiniciado quando o conteúdo é reinserido na cache.

Tabela 1. Tabela de notação. Variáveis se referem ao conteúdo c (subscrito omitido quando claro no contexto).

variável	descrição
N	número de conteúdos
λ_c	taxa de chegada ao conteúdo c
μ_c	taxa de tique do *timer* para decrementar o RC
γ_c	taxa de escrita, i.e., taxa com que conteúdo c é inserido na cache
π_c	fração de tempo em que conteúdo c passa na cache (sob chegadas Poisson, é igual ao *hit probability*)
h_c	*hit probability*
$E[B_c]$	tempo esperado que conteúdo c fica na cache, desde inserção
$E[R_c]$	tempo esperado que conteúdo c fica fora da cache, desde remoção
K_c	limiar de inserção (para histerese na inserção)
L_c	limiar de remoção (para histerese na remoção)
Δ_c	$K_c - L_c$

e é decrementado a cada tique de um relógio. Seja K o limiar de inserção e remoção de conteúdos (Figura 1). Quando o contador é incrementado de K para $K+1$, o conteúdo é armazenado. Quando o contador é decrementado de $K+1$ para K o conteúdo é removido. O contador faz tiques a cada $1/\mu$ segundos. A seguir, a não ser dito contrário, assumimos que o tempo entre tiques é exponencialmente distribuído, para gerar um modelo tratável (em nossos estudos com *traces*, assumimos tempo entre tiques determinístico). O mecanismo é similar a time-to-live (TTL) caches, aplicados em DNS e sistemas de web-caching [Berger et al., 2014, Fofack et al., 2012].

Consideramos restrições sobre o valor esperado do número de itens armazenados na cache. Num ambiente de nuvem, isto corresponde a usuários que incorrem custos proporcionais ao espaço médio utilizado [Amazon, 2017]. Para uma capacidade de armazenamento fixa, é conhecido que o desacoplamento dos conteúdos e restrições sobre ocupação média geram boas aproximações para LRU e FIFO [Fofack et al., 2012]. As probabilidades de *hit* sob LRU e FIFO são bem aproximadas por contadores TTL com e sem *reset*, respectivamente.

É bem conhecido o fato de que multiplexação estatística é um dos elementos chaves de políticas de armazenamento de conteúdo. Pode-se tomar proveito de multiplexação estatística removendo itens da cache quando (a) um novo item é inserido (e ocorre um overflow, no caso de caches de capacidade finita) ou (b) quando um item expira. Este último caso é particularmente relevante em caches TTL (e.g., DNS) onde entradas precisam ser renovadas de tempos em tempos para evitar que expirem.

Em seguida, assumimos que requisições para um determinado conteúdo chegam segundo um fluxo Poisson com taxa λ. A taxa λ também é conhecida como popularidade do conteúdo. Cabe lembrar que a dinâmica de inserção e remoção de cada conteúdo é desacoplada dos demais. Um conteúdo que não está na cache é trazido para cache quando o contador aumenta de K para $K+1$. A remoção ocorre quando o contador é decrementado de $K+1$ para K após um tique do relógio. Adotamos a mesmas hipóteses que [Fofack et al., 2012]: a inserção e remoção de um conteúdo não são influenciadas pelos outros conteúdos na cache. Conforme mencionado acima, a capacidade de armazenamento é modelada por meio do valor esperado do número de conteúdos na cache.

Objetivos Um de nossos objetivos é mostrar as vantagens em se adotar um mecanismo com dois controles para ajustes, μ e K. O tique do relógio μ e o limiar do RC K podem ser ajustados para se alcançar uma determinada fração de tempo alvo em que o conteúdo estará na cache (métrica em estado estacionário) e para se controlar o tempo médio entre entrada e saída do conteúdo na cache (métricas transientes, relacionadas com a taxa com que o conteúdo é inserido na cache). Um dos objetivos de se controlar métricas transientes é evitar que conteúdos sejam trocados com muita frequência, e ao mesmo tempo prevenir *starvation*, dando oportunidade para que todos os conteúdos sejam inseridos ou removidos da cache em um horizonte de tempo finito.

3.3. Métricas de interesse

3.3.1. Caso geral

Seja π_c a fração de tempo em que o conteúdo c está na cache. Para simplificar a apresentação, removemos o subscrito c quando este estiver claro a partir do contexto. Cada conteúdo se alterna entre estar presente e ausente da cache, e obtemos (teoria da renovação),

$$\pi = \frac{E[B]}{E[R] + E[B]} \tag{1}$$

onde $E[R]$ é o tempo médio para o conteúdo retornar à cache, uma vez que ele é removido, e $E[B]$ é o tempo médio que o conteúdo permanece na cache, uma vez inserido.

Seja $\gamma(K, \mu)$ a taxa com que o conteúdo é inserido na cache. Por conta das equações de balanço, em estado estacionário, $\gamma(K, \mu)$ é igual a taxa com que o conteúdo é removido da cache,

$$\gamma(K, \mu) = \frac{1}{E[B] + E[R]} \tag{2}$$

onde esta última igualdade pode ser facilmente derivada usando argumentos de teoria da renovação.

3.3.2. Caso específico de RC com um limiar único

A seguir, consideramos o caso em que o limiar de inserção é igual ao limiar de remoção, ou seja, só há histerese na inserção (Seção 3.2).

Nesse caso, o RC pode ser modelado como uma fila M/M/1, com o estado correspondendo ao valor do RC. Então, $\pi = \sum_{i=K+1}^{\infty}(1-\rho)\rho^i = \rho^{K+1}$, onde $\rho = \lambda/\mu$. Além disso, $\gamma(K, \mu) = \lambda\rho^K(1-\rho) = \mu\rho^{K+1}(1-\rho)$.

O tempo médio que o conteúdo ocupa a cache, $E[B]$, segue a partir de análise do tempo de alcance a estados. Ele é dado pelo tempo a alcançar o estado K começando do estado $K+1$. Então, $E[B]$ é dado pelo período ocupado de uma fila M/M/1, que vale $E[B] = 1/(\mu - \lambda)$. A partir de (1) sabemos que $\pi = (1/(\mu-\lambda))/(1/(\mu-\lambda) + E[R])$. Uma vez que o conteúdo é removido, leva-se $E[R]$ para reinserção, $E[R] = (1 - \pi)/(\pi(\mu - \lambda))$.

Tabela 2. Graus de liberdade na escolha dos parâmetros. As métricas de interesse alvo são a probabilidade de *hit*, taxa de inserção (que é igual a taxa de escrita) e o tempo antes de remoção (uma vez ocorrida a inserção).

mecanismo	métricas alvo	variáveis de controle disponíveis
TTL caches	probabilidade de *hit*	valor do TTL
RC (com $K = L = 0$)	probabilidade de *hit*	taxa de decréscimo do contador (μ)
RC com único limiar ($K = L \geq 0$)	probabilidade de *hit* e taxa de inserção	taxa de decréscimo do contador (μ) e limiar de inserção K
RC com dois limiares (K e L podem ser diferentes, $L \leq K$)	probabilidade de *hit*, taxa de inserção e tempo esperado de permanência	taxa de decréscimo do contador (μ), limiar de inserção K e limiar de remoção L

3.4. Graus de liberdade

A Tabela 2 resume as variáveis de controle disponíveis em diferentes mecanismos de cache, e seus impactos em diferentes métricas de desempenho. Enquanto caches TTL clássicas possuem um único controle, que permite ajustar a probabilidade de *hit*, RC com histerese permite mais ajustes por meio de dois limiares de inserção e remoção. No restante deste trabalho, vamos indicar tanto analiticamente quanto numericamente o impacto dos diferentes parâmetros nas métricas de interesse. Em particular, vamos indicar numericamente que as probabilidades de *hit* tendem a ser robustas (menos sensíveis) com relação a mudanças nas variáveis de controle, enquanto que a taxa de inserção (e correspondente número de escritas) tende a ser mais sensível aos limiares de inserção e remoção.

3.5. RC com dois limiares e histerese na inserção e remoção

Nesta seção, generalizamos o RC para contar com um terceiro parâmetro de controle L, $L \leq K$, da seguinte forma. Como antes, o contador é incrementado a cada requisição, e é decrementado a cada tique do relógio, a taxa μ. O conteúdo é inserido quando o RC passa de K para $K + 1$. Entretanto, para evitar remoções imediatas, o conteúdo não é removido quando o RC passa de $K + 1$ para K. Em vez disso, o conteúdo permanece no cache até que o (novo) limiar L seja atingido (veja Figura 2).

A seguir, indicamos como calcular os valores esperados $E[B]$ e $E[R]$ assumindo histerese na inserção e remoção. Tendo $E[B]$ e $E[R]$ pode-se calcular as métricas de interesse (Seção 3.3.1). Nas seções seguintes, iremos apontar como as métricas de interesse são afetadas pelos parâmetros disponíveis.

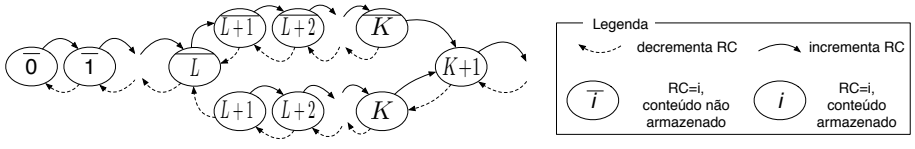


Figura 2. Contador por reforço com histerese: diagrama de transição de estados.

Derivação de métricas de interesse Seja $\xi(i)$ o tempo médio para incrementar o RC por i unidades, começando do estado $\overline{K + 1 - i}$, para $i = 1, \ldots, K - L + 1$. De forma análoga, seja $\nu(i)$ o tempo médio para decrementar o RC por i unidades, começando por

um estado j tal que $j \geq L + i$, for $i = 1, \ldots, K - L + 1$. Seja $\Delta = K - L$. Claramente, $E[R] = \xi(\Delta + 1)$ e $E[B] = \nu(\Delta + 1)$.

Proposição 1 *Temos que*

$$E[B] = \frac{K - L + 1}{\mu - \lambda}, \quad E[R] = \frac{1}{\mu - \lambda}\left(\frac{\rho^{-K-1} - \rho^{-L}}{1 - \rho} - (K - L + 1)\right) \tag{3}$$

A prova da proposição acima advém do fato de que $\nu(i)$ e $\xi(i)$ são dados pelas seguintes recursões:

$$\nu(i) = \begin{cases} \nu(i-1) + \rho\nu(1) + \mu^{-1}, & 2 \leq i \leq \Delta + 1 \\ 1/(\mu - \lambda), & i = 1 \end{cases} \tag{4}$$

e

$$\xi(i) = \begin{cases} \xi(i-1) + (\rho^{-K+i-2} - 1)/(\mu - \lambda), & 2 \leq i \leq \Delta + 1 \\ (\rho^{-K-1} - 1)/(\mu - \lambda), & i = 1 \end{cases} \tag{5}$$

Resolvendo as recursões, a proposição segue de imediato. Cabe destacar que o caso base $\nu(1)$ corresponde ao período ocupado de uma fila M/M/1, $\nu(1) = 1/(\mu - \lambda)$. De forma análoga, o caso base $\xi(1)$ corresponde ao tempo médio para ir do estado K ao estado $K+1$ numa fila $M/M/1/K+1$, que vale $\mu^{-1}(1/\tilde{\pi}_{K+1} - 1)$ onde $\tilde{\pi}_{K+1}$ é a probabilidade estado estacionário desta fila, $\tilde{\pi}_{K+1} = \rho^{K+1}(1 - \rho)/(1 - \rho^{K+2})$. Concluímos assim a derivação do resultado. Nas seções seguintes, iremos avaliar numericamente os resultados obtidos com o modelo, a fim de analisar sua sensibilidade com relação a diferentes parâmetros.

4. Benefícios da histerese em cargas reais

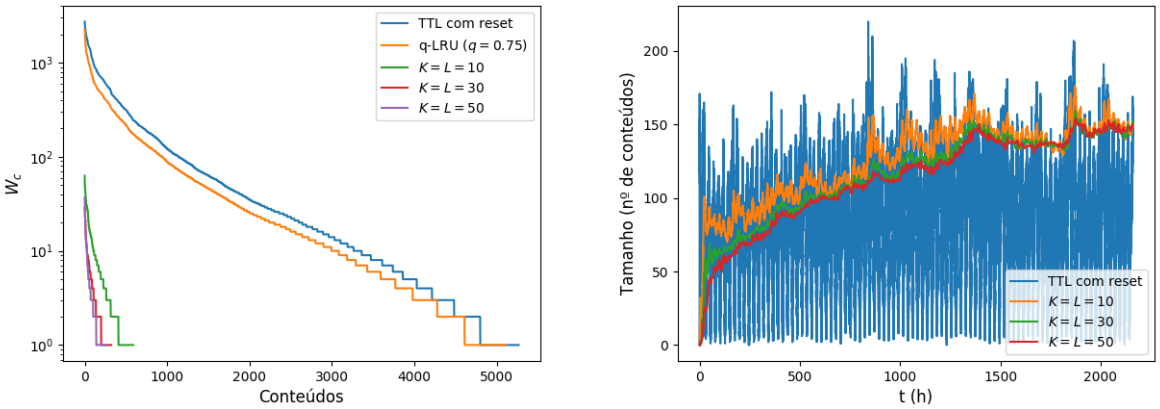
A seguir, nosso objetivo é mostrar que podemos nos beneficiar da flexibilidade do RC em cenários reais. Para tal, consideramos simulações *trace-driven* usando como entrada um *log* de acessos a um serviço de VoD brasileiro oferecido por um grande provedor. Através dessa abordagem, podemos a) ilustrar um cenário em que o RC é benéfico; e b) comparar RC com LRU e q-LRU [Garetto et al., 2016].

O catálogo considerado contém 5.267 conteúdos acessados por 25.933 clientes de 19 cidades no Brasil. Em 90 dias, os clientes fizeram 1.004.829 de requisições. Para cada requisição, sabemos 1) timestamp, 2) ID do usuário (anonimizado), e 3) nome do conteúdo.

Dentre os conteúdos, 435 são *one-hit wonders*, i.e., são requisitados apenas 1 vez e nunca mais. Esses conteúdos geram distorções desnecessárias sob LRU. Isto não ocorre sob RC com histerese ou variantes do LRU como q-LRU.

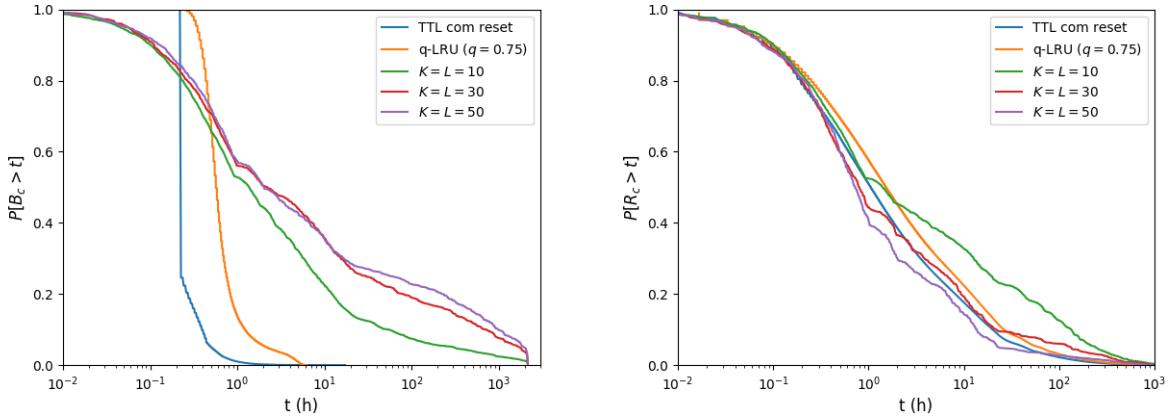
O tráfego entre 15h e 2h é bem aproximado por um fluxo Poisson com taxa $\lambda = 11,36$ visualizações por minuto. Já a popularidade pode ser modelada usando uma distribuição Zipf com decaimento exponencial, sendo que aproximadamente 50% dos acessos são para os 150 conteúdos mais populares.

A Figura 3(a) mostra o número de escritas à cache sob várias políticas. Quando adotamos a política RC, o número de escritas é menor que 100 em todos os casos. Para as demais políticas, o número de escritas é maior que 500 para mais de 1,000 conteúdos. Já



(a) número de escritas (b) ocupação da cache

Figura 3. (a) número de escritas à cache e (b) ocupação da cache ao longo do tempo, para TTL com *reset*, e RC com $K = L \in \{10, 20, 30\}$.



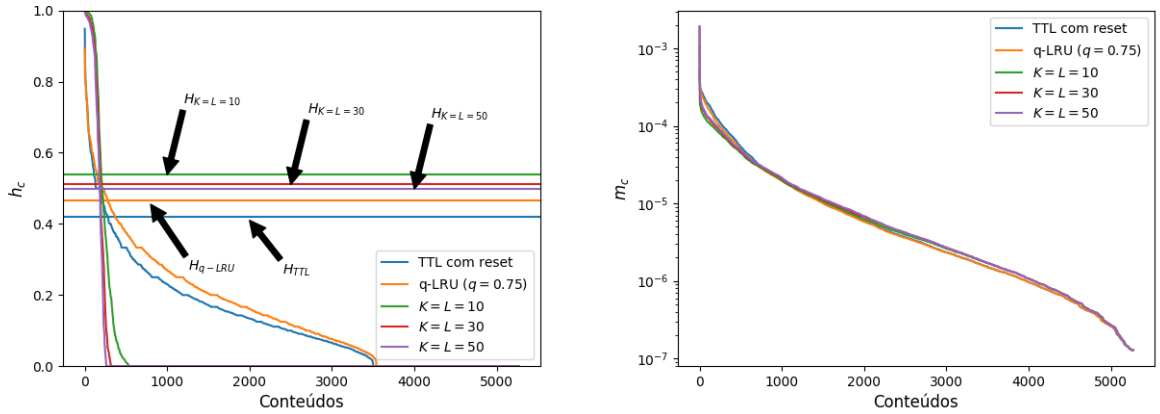
(a) CCDF do tempo de residência (b) CCDF do tempo de retorno

Figura 4. CCDF dos tempos de residência e retorno.

a Figura 3(b) mostra que a ocupação da cache varia substancialmente ao longo do tempo sob política TTL com *reset*, mas que o mesmo não ocorre com RC. Neste último caso, as alocações tendem a ficar mais estáticas ao longo do tempo. Verificamos que um TTL de 36 com decremento a cada 20 segundos ($\mu = 1/20s^{-1}$) gera uma ocupação média da cache de aproximadamente 150, condizente com a Figura 3(b). Assim, adotamos esses parâmetros em todos os resultados que se seguem para essa política.

A Figura 4 mostra a CCDF dos tempos de residência dos conteúdos na cache, e dos tempos de retorno. Ela indica que os tempos de residência segundo LRU são menores que aqueles obtidos via RC. Este fato está de acordo com os resultados obtidos na Figura 3, indicando que o RC garante estabilidade extra ao sistema por evitar que os conteúdos entrem e saiam com muita frequência da cache.

A Figura 5 mostra a probabilidade de *hit* e a taxa de *misses*, segundo a política TTL com reset e RC, para $K = L \in \{10, 20, 30\}$. Note que a taxa de misses é bem



(a) probabilidade de *hit* (b) taxa de *misses*

Figura 5. Probabilidade de *hit* e taxa de *misses* para TTL com *reset*, q-LRU e RC.

menos sensível à variação de parâmetros do que a probabilidade de *hit*. Além disso, a probabilidade de *hit* com RC apresentou resultados superiores, para as parametrizações consideradas, levando em conta a probabilidade de *hit* agregada para a cache como um todo (linhas horizontais na Figura 5(a)).

Nas Figuras 5(a) e 5(b) também reportamos resultados com q-LRU, uma variante do LRU que, após um *miss*, armazena um conteúdo com probabilidade q. Experimentamos com 3 diferentes valores de q ($q = 0.5, 0.75, 0.9$) e selecionamos $q = 0.75$ por produzir maiores probabilidades de *hit*. Entretanto, como mostrado nas Figuras 4(a) e 4(b), q-LRU não desvia significativamente do LRU padrão (aproximado pelo TTL com *reset*).

5. Identificando classes de parâmetros de histerese

Nesta seção, apresentamos resultados numéricos obtidos com o modelo analítico de RC proposto em trabalho anterior. Em seguida, usamos algoritmos de aprendizado por máquina, tomando como entrada os resultados numéricos obtidos com o modelo, para a) identificar fórmulas fechadas que relacionem diferentes parâmetros com as métricas de interesse e b) identificar o impacto dos parâmetros nas métricas de interesse. Não é de nosso conhecimento nenhum trabalho anterior que tenha adotado esta abordagem de usar algoritmos de aprendizado por máquina tomando como base amostras colhidas a partir de um modelo de avaliação de desempenho a fim de avaliar a sensibilidade deste último com relação aos seus parâmetros.

Para analisar o modelo, variamos os parâmetros $K = 10, 11, 12, \ldots, 40$ e $L = K - 10, K - 9, \ldots, K$. Para cada combinação (K, L), geramos 50 amostras de λ na faixa de 10^{-7} a 10^{-3} – inspirados nos *traces* descritos na última seção – e, para cada valor de λ, 50 amostras de μ no intervalo $(\lambda \cdot 1.001, \lambda \cdot 1.3)$. Para cada conjunto de parâmetros, calculamos π e γ usando as equações (1) e (2), sendo que $E[B]$ e $E[R]$ são dados por (3).

A Figura 6 ilustra a exploração do espaço de parâmetros. Ela mostra como variam a ocupação da cache, que no caso de fluxo Poisson é igual à probabilidade de hit (π), e a taxa de escritas (γ) em função dos demais parâmetros (K, L, λ, μ), sempre mantendo um fixo (eixo x) e variando os demais livremente. Notamos que fixar K ou L tem um

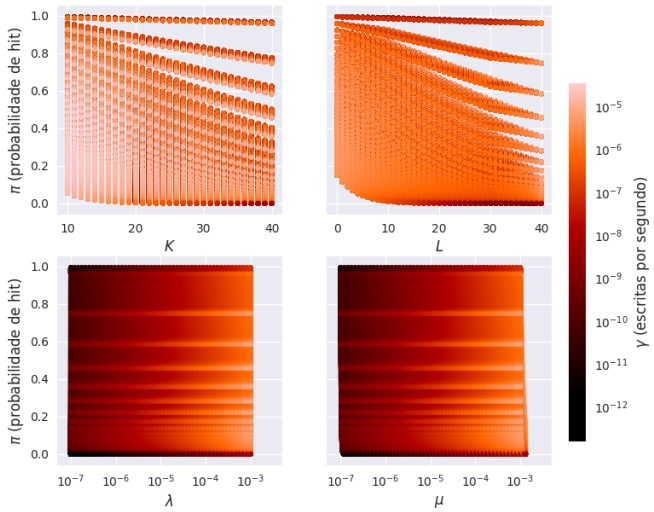


Figura 6. Probabilidade de *hit* (π) em função dos parâmetros (K, L, λ, μ), mantendo um fixo (eixo x) e variando os demais livremente. A escala de cores indica a taxa de escritas por segundo (γ).

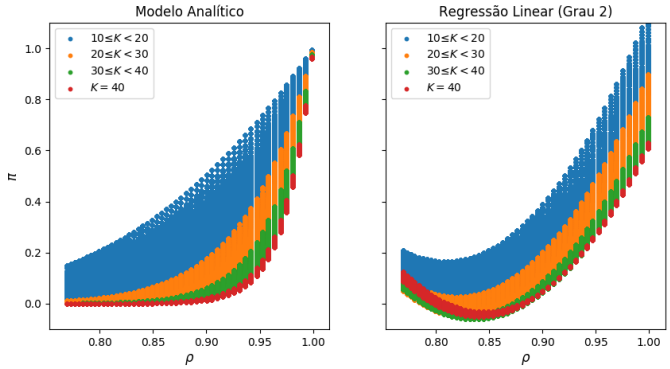


Figura 7. Probabilidade de hit (π), em função de ρ e K: (a) dados analíticos amostrados; (b) regressão linear com grau 2.

efeito restritivo sobre π. À medida que K e L aumentam, a probabilidade de *hit* passa a assumir faixas de valores cada vez mais estreitas, levando a um sistema mais previsível (o que em parte deve-se também ao fato de que as probabilidades assumem valores cada vez menores). Já os gráficos de λ e μ indicam que a probabilidade de *hit* π pode assumir uma ampla faixa de valores dependendo de uma combinação dos outros 3 parâmetros. Quanto a γ, notamos que valores menores de λ e μ estão associados a uma taxa de escritas menor. Isso é esperado, dado que as inserções estão associadas a uma chegada que incrementou o contador para $K + 1$ (e portanto dependem de λ) e as remoções ao decremento do RC (dado pela taxa μ) de $L + 1$ para L.

Regressão para simplificação de expressões A Figura 7(a) mostra uma visão diferente dos dados apresentados na Figura 6, indicando os valores possíveis de π para combinações de $\rho = \lambda/\mu$ e K. Esse gráfico corrobora com nossa intuição de que valores maiores de K

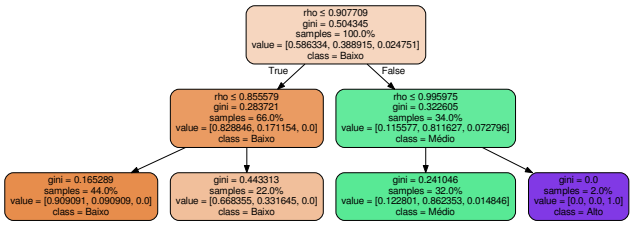


Figura 8. Árvores de decisão para π em função de ρ e Δ [acurácia alcançada de 82,88%]: a insensibilidade de π com relação a Δ fica evidente nas árvores, que não dependem de Δ

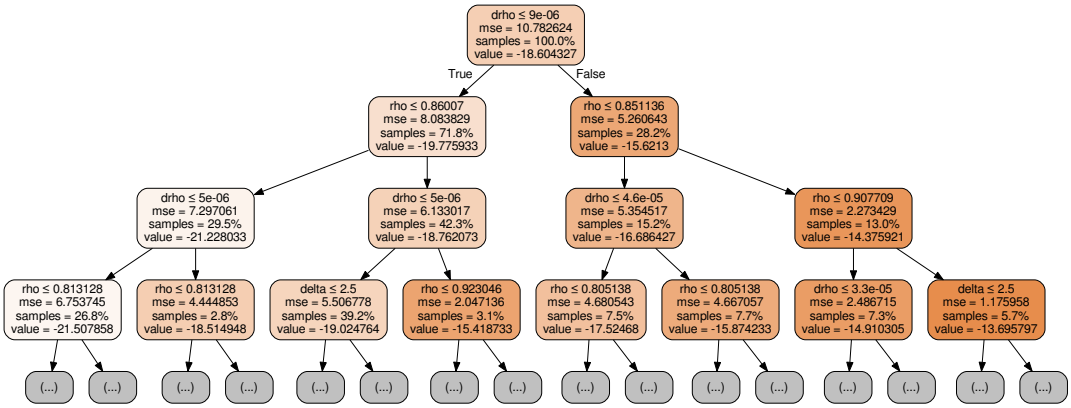


Figura 9. Árvores de decisão para γ em função de $\rho = \lambda/\mu$, $\mu - \lambda$ e $\Delta = K - L$: a sensibilidade de γ com relação a Δ fica clara na árvore, que possui alguns nós cuja decisão se dá em função do valor de Δ.

tornam π mais previsível. Além disso, notamos uma forte relação entre π e ρ.

Ao lado, na Figura 7(b), vemos a aproximação dada por uma regressão linear treinada a partir das amostras do modelo analítico. Usamos no treinamento as *features* K, K^2, L, L^2, KL, ρ, ρ^2, $K\rho$ e $L\rho$. Percebemos pelo gráfico que a regressão captura razoavelmente bem o comportamento do modelo analítico, levando a um RMSE (*Root Mean Square Error*) de 0.0645 na previsão da probabilidade de *hit*, mesmo prevendo valores fora do intervalo $[0, 1]$ (que poderiam ser truncados). Esse modelo possui como vantagem o fato de fornecer uma fórmula fechada para a regressão que pode ser facilmente interpretada, diferentemente de outros modelos mais complexos. De acordo com os coeficientes do modelo treinado, π cresce quadraticamente com ρ, sendo fortemente dependente dessa variável, e linearmente com K e L, sendo também dependente dos fatores $K\rho$ e $L\rho$.

Cabe destacar que pode-se inferir diretamente a partir da Seção 3.3.2 que, se $K = L$, então $\pi = \rho^{K+1}$, o que explica a relação linear quando $K = 0$ e quadrática quando $K = 1$. Ou seja, analiticamente é possível verificar facilmente o motivo de, sob certas condições, haver uma relação simples entre π e ρ. Entretanto, acreditamos que a abordagem automatizada aqui apresentada pode ser útil quando a avaliação analítica não for viável ou trivial. A abordagem automatizada é particularmente útil quando se deseja fazer "cortes" não triviais nos dados, e.g., $K = \kappa L$.

Classificação para análise de sensibilidade As Figuras 8 e 9 ilustram como árvores de decisão podem ser usadas para análise de sensibilidade. Elas corroboram os resultados via *traces* obtidos na seção anterior, que indicavam que π é menos sensível a K e L em comparação com a taxa de escritas γ. Para tal, dividimos os valores de π em baixo $[0, 0.1]$, médio $(0.1, 0.9]$ e alto $(0.9, 1]$ e usamos uma árvore de decisão para classificação.

Nas árvores, cada nó não folha corresponde a uma decisão. Na árvore de classificação (Figura 8) todos os nós possuem a indicação da fração de instâncias do tipo baixo, médio e alto que eles capturam. Na árvore de regressão (Figura 9) os nós possuem a indicação de qual o valor médio (e respectivo RMSE) da métrica alvo (no caso, γ) para as amostras classificadas naquele ramo.

Verificamos então que a árvore naturalmente ignorou os valores de K e L ao determinar a influência de ρ sobre π (Figura 8). Entretanto, o mesmo não ocorre com a árvore de γ (Figura 9). Neste último caso, usamos a árvore para regressão de γ, que foi normalizado usando a escala logaritmica. A árvore leva em conta os valores de Δ e de $\mu - \lambda$ (denotado por *drho* na árvore), bem como o valor de ρ, no resultado da regressão. Nós mais no topo indicam parâmetros que afetam de forma mais significativa a quantidade de interesse, enquanto que nós de decisão mais próximos às folhas representam ajustes mais finos. Assim, temos um nó raiz que depende de $\mu - \lambda$ na árvore da Figura 9, indicando que tal quantidade impacta de forma significativa a taxa de escritas na cache, γ. No quarto nível, vemos a influência de Δ.

6. Trabalhos relacionados

A literatura sobre sistemas de cache é vasta, e inclui trabalhos sobre caches isoladas [Liu et al., 1998], caches conectadas em topologias hierárquicas [Garetto et al., 2016] e em topologias gerais [Rosensweig et al., 2013]. Neste trabalho, consideramos caches governadas por mecanismos RC, uma variante dos mecanismos TTL [Fofack et al., 2012], levando em conta histerese.

Embora histerese esteja no âmago dos sistemas de cache, o tema recebeu pouca atenção na literatura até então. Dentre os trabalhos relacionados, destaca-se [Domingues et al., 2015]. O presente trabalho estende [Domingues et al., 2015] por 1) considerar cargas de dados reais, 2) usar algoritmos de aprendizado por máquina para avaliar a sensibilidade dos parâmetros e 3) obter fórmulas fechadas para as métricas de interesse.

Dentre os benefícios da histerese, destacamos o fato de ela ser útil para lidar com o desafio de *one-hit wonders*. Tal desafio foi discutido, por exemplo, em [Shafiq et al., 2016], onde sugere-se não armazenar em cache a primeira requisição a cada conteúdo. Neste trabalho, em contrapartida, generalizamos tal ideia usando o conceito de histerese.

Sistemas de caching sensíveis a escritas em disco também têm recebido atenção recente [Neglia et al., 2017]. A redução da utilização do disco diminui o tempo de resposta, e aumenta a validade de memórias flash [Garetto et al., 2016, Perino e Varvello, 2011]. No presente trabalho, contribuímos no sentido de mostrar como a histerese pode também ajudar neste sentido.

7. Conclusão

Caches são peças fundamentais na Internet desde sua concepção. Entretanto, a interconexão de caches traz novos desafios, relacionados a parametrização de variáveis de controle que afetam a taxa de acertos e o número de escritas na cache. Neste artigo, mostramos como *reinforced counters* e histerese podem ser úteis para lidar com tais desafios. Nossa abordagem envolve modelos analíticos, e resultados numéricos com *traces* reais. Também indicamos como mecanismos de aprendizado por máquina podem auxiliar na parametrização de modelos analíticos. Para tal, abordamos como a regressão pode ser usada para encontrar simplificações ao modelo em determinados regimes e árvores de decisão podem ser usadas para análise de sensibilidade. Vislumbramos que esta abordagem se aplique também a outros problemas e que possa abrir caminhos para trabalhos futuros na interseção entre aprendizado por máquina e avaliação de desempenho.

Referências

Amazon (2017). Amazon Elastic Cache: Amazon Web Services. `http://aws.amazon.com/elasticache/`.

Berger, D. S., Gland, P., Singla, S., e Ciucu, F. (2014). Exact analysis of TTL cache networks. *Elsevier Performance Evaluation*, 79:2–23.

Carofiglio, G., Mekinda, L., e Muscariello, L. (2016). Analysis of latency-aware caching strategies in information-centric networking. In *Content Caching and Delivery in Wireless Networks*, page 5. ACM.

Dehghan, M., Massoulie, L., Towsley, D., Menasche, D., e Tay, Y. (2016). A utility optimization approach to network cache design. In *INFOCOM*.

Domingues, G., e Silva, E. d. S., Leão, R. M., Menasché, D. S., e Towsley, D. (2017). Enabling opportunistic search and placement in cache networks. *Computer Networks*, 119:17–34.

Domingues, G., Leao, R. M., Menasche, D. S., et al. (2015). Flexible content placement in cache networks using reinforced counters. *SBRC (arXiv:1501.03446)*.

Fofack, N. C., Nain, P., Neglia, G., e Towsley, D. (2012). Analysis of TTL-based cache networks. In *IEEE VALUETOOLS*.

Garetto, M., Leonardi, E., e Martina, V. (2016). A unified approach to the performance analysis of caching systems. *TOMPECS*, 1(3):12.

Liu, Z., Nain, P., Niclausse, N., e Towsley, D. (1998). Static caching of web servers. In *Multimedia Computing and Networking Conference*.

Neglia, G., Carra, D., Feng, M., Janardhan, V., Michiardi, P., e Tsigkari, D. (2017). Access-time-aware cache algorithms. *TOMPECS*, 2(4):21.

Perino, D. e Varvello, M. (2011). A reality check for content centric networking. In *ACM SIGCOMM ICN*, pages 44–49. ACM.

Rosensweig, E. J., Menasche, D. S., e Kurose, J. (2013). On the steady-state of cache networks. In *INFOCOM, 2013 Proceedings IEEE*, pages 863–871. IEEE.

Shafiq, M. Z., Khakpour, A. R., e Liu, A. X. (2016). Characterizing caching workload of a large commercial content delivery network. In *IEEE INFOCOM*.

Tatarinov, I., Rousskov, A., e Soloviev, V. (1997). Static caching in web servers. In *Computer Communications and Networks*, pages 410–417. IEEE.