Staying up to Date with Online Content Changes Using Reinforcement Learning for Scheduling

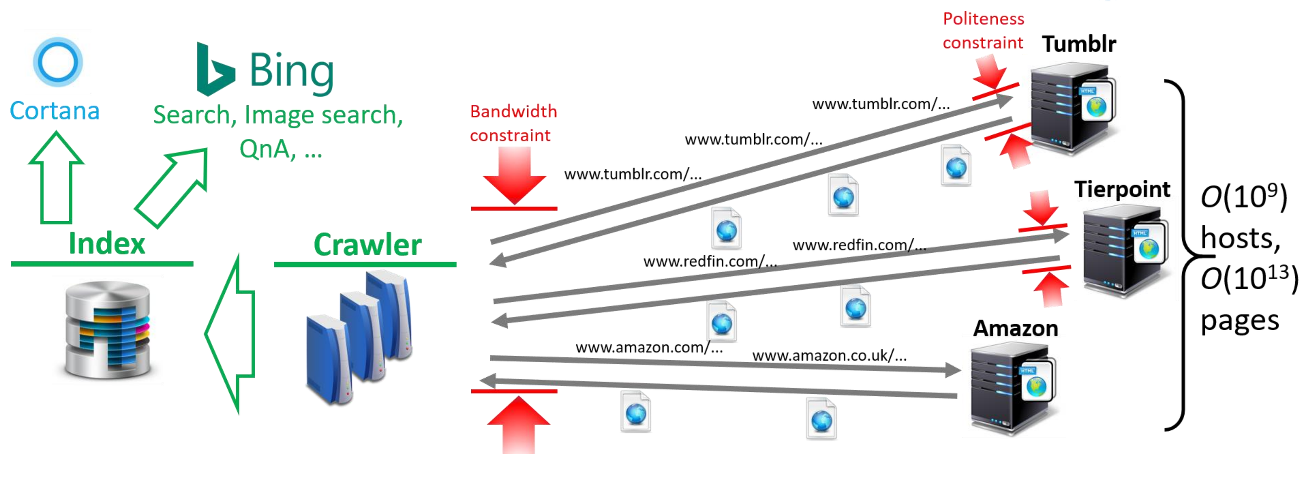
Andrey Kolobov
Microsoft Research
akolobov@microsoft.com

Yuval Peres
yperes@gmail.com

Cheng Lu
Microsoft Bing
cheng.lu@microsoft.com

Eric Horvitz
Microsoft Research
horvitz@microsoft.com

Web crawl scheduling



Fresh knowledge of online content **is critical for search engines** & other *trackers.*
They collect Web data by downloading (***crawling***) pages.
Freshness crawl revisits indexed URLs to pick up changes.

How do we efficiently schedule crawl to maximize freshness across billions of pages under crawl bandwidth constraints?

Model

Set of sources W is fixed. Each source (e.g., URL) w *in* W has **importance score μ_w** and Poisson **change rate Δ_w**.

Want: policy π saying when to crawl each w in W to **minimize** average importance-weighed **staleness penalty**

Average over a long time period

Expectation over possible change and crawl sequences

$$J^\pi = \lim_{T\to\infty} \mathop{\mathbb{E}}_{\substack{CrSeq\sim\pi,\\ ChSeq\sim P(\vec{\Delta})}} \left[\frac{1}{T}\int_0^T \sum_{w\in W} \mu_w C(N_w(t))\,dt\right]$$

C(N) is a *staleness penalty function*

$N_w(t)$ is the number of times page w changed by time t since its latest crawl

subject to crawl bandwidth constraint R.

A popular *C(N) is* **binary penalty [3]**: *C(0) = 0, C(N > 0) = 1*

Our Contributions

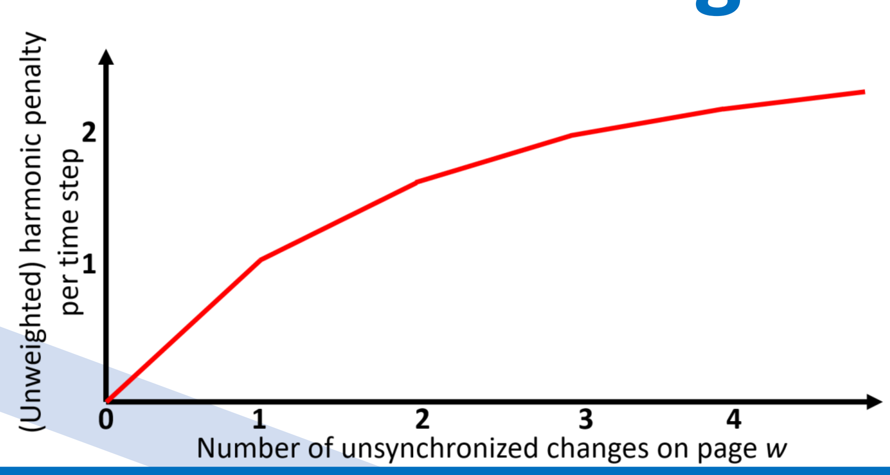
- New intuitive & efficiently optimizable objective for freshness crawling, *harmonic staleness penalty.*
- *LambdaCrawl* – an optimal efficient crawl scheduling algorithm for partial, full & mixed page change observability.
- *LambdaLearnAndCrawl* – RL for minimizing staleness under unknown page change rates Δ_w
- Approximation algorithm with data-dependent perf. guarantees

New Objective: Minimizing Harmonic Penalty

Let *C(N)* = $H_N = 1 + \frac{1}{2} + \dots + \frac{1}{N}$ *if N > 0,* else *C(0) = 0.*

H_N is the *N*-th harmonic number.

Under harmonic C(N), J^π is equivalent to



Why specifically harmonic penalty?

- Has diminishing penalty property – makes intuitive sense.
- Guarantees that every source with change rate $\Delta_w > 0$ gets crawled occasionally (no such guarantee for binary *C(N)*!).
- Induces near-optimal policies w.r.t. binary *C(N)* as well
- Efficient to optimize (unlike natural alternatives, e.g. *log(1+N)*)

LambdaCrawl: Crawl Optimization Under Known Parameters

Incomplete Change Observations

For sources in set W^-, tracker finds out that they changed only when crawling them. Consider policies that crawl each w in with some Poisson rate ρ_w.

Want: *Crawl rates* $\vec{p} = (\rho_w)_{w\in W^-}$ *maximizing*

$$\overline{J}^\pi = -J^\pi = \sum_{w\in W^-} \mu_w \ln\left(\frac{\rho_w}{\Delta_w + \rho_w}\right)$$

subject to

$$\sum_{w\in W^-} \rho_w = R, \rho_w \geq 0 \text{ for all } w \in W^-.$$

Has a unique optimum for all $\rho_w \geq 0$.
Solve optimally using Lagrange multiplier method + binary search on multiplier λ

Complete Change Observations

For sources in set W^o, tracker gets notified about every change (e.g., using telemetry). Consider policies that crawl source w with probability p_w on each notification.

Want: *Crawl probabilities* $\vec{p} = (p_w)_{w\in W^o}$ *maximizing*

$$\overline{J}^\pi = -J^\pi = \sum_{w\in W^o} \mu_w \ln(p_w)$$

subject to

$$\sum_{w\in W^o} p_w\Delta_w = R, \quad 0 \leq p_w \leq 1 \text{ for all } w \in W^o$$

Has a unique optimum for all p_w in [0,1].
Solve optimally by repeatedly applying Lagrange multiplier method to O(|W^o|) inequality-constraint-free relaxations (a bit tricky!)

Mixed Observations

In reality, tracker must deal with a mix W = $W^- \cup W^o$

Want: *Crawl rates* $\vec{p} = (\rho_w)_{w\in W^-}$ *and crawl probabilities* $\vec{p} = (p_w)_{w\in W^o}$ *maximizing*

$$\overline{J}^\pi = -J^\pi = \sum_{w\in W^-} \mu_w \ln\left(\frac{\rho_w}{\Delta_w + \rho_w}\right) + \sum_{w\in W^o} \mu_w \ln(p_w)$$

subject to

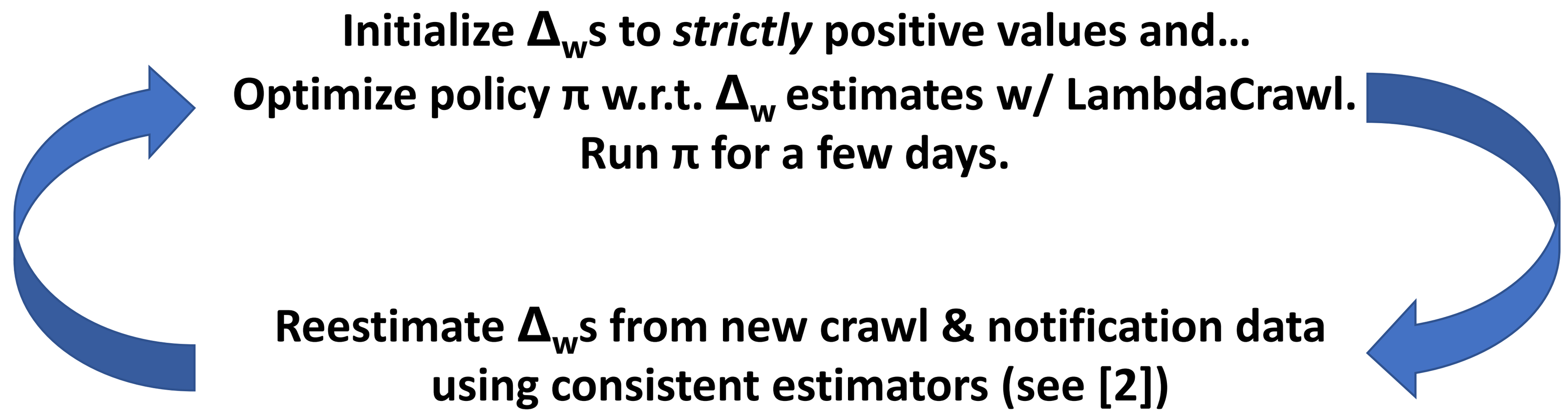
$$\sum_{w\in W^-} \rho_w + \sum_{w\in W^o} p_w\Delta_w = R,$$

$$\rho_w > 0 \text{ for all } w \in W^-, \quad 0 < p_w \leq 1 \text{ for all } w \in W^o$$

Boils down to finding the (unique!) optimal bandwidth split R = $R^o + R^-$
Easy concave maximization on [0, R]

LambdaLearnAndCrawl: RL for Crawling

In reality, initially we don't know source change rates Δ_w!
Initialize Δ_ws to *strictly* positive values and…
Optimize policy π w.r.t. Δ_w estimates w/ LambdaCrawl.
Run π for a few days.

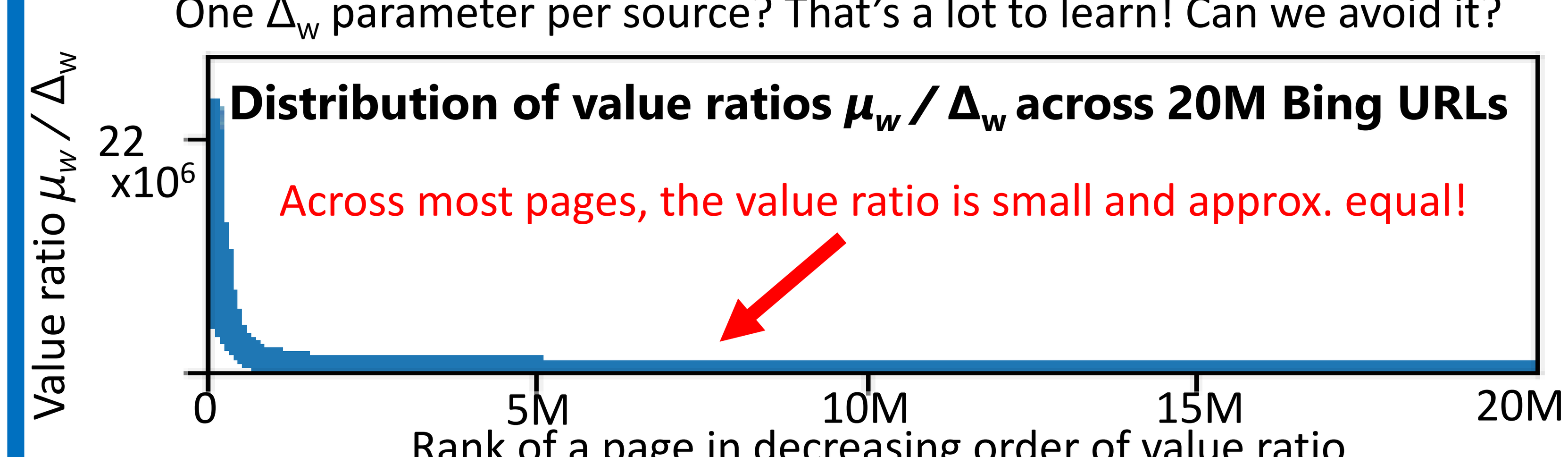


Reestimate Δ_ws from new crawl & notification data using consistent estimators (see [2])

Guaranteed to converge to optimal π* (assuming stationary Δ_ws).

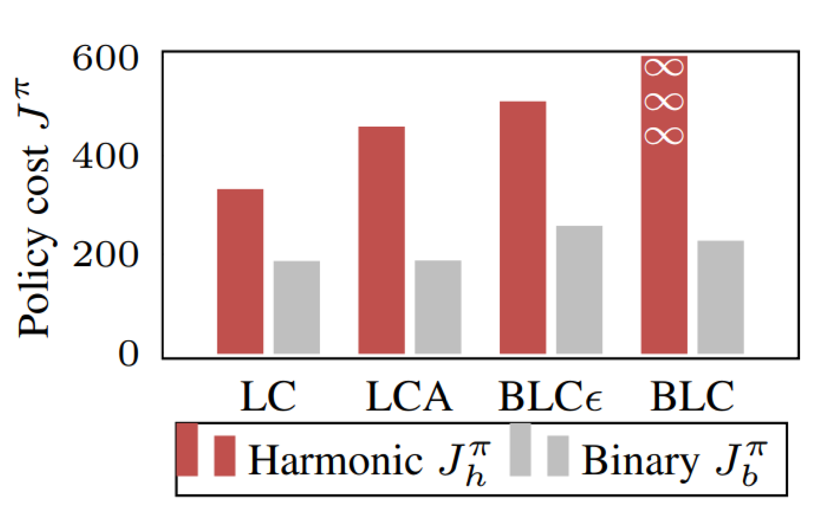
Data-Dependent Approximation

One Δ_w parameter per source? That's a lot to learn! Can we avoid it?

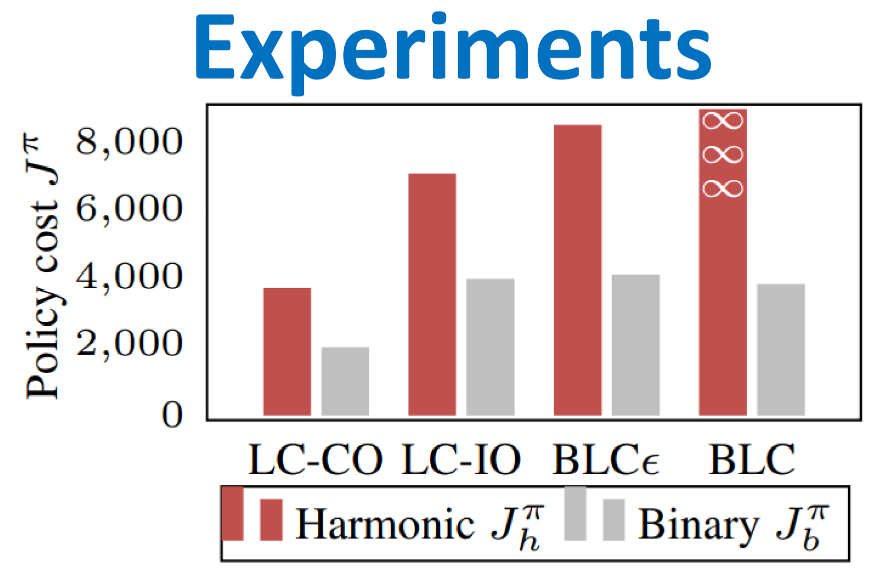


Theorem: across the constant value ratio region, the optimal crawl rate $\rho_w = \alpha\,\mu_w R$ – independent of change rate

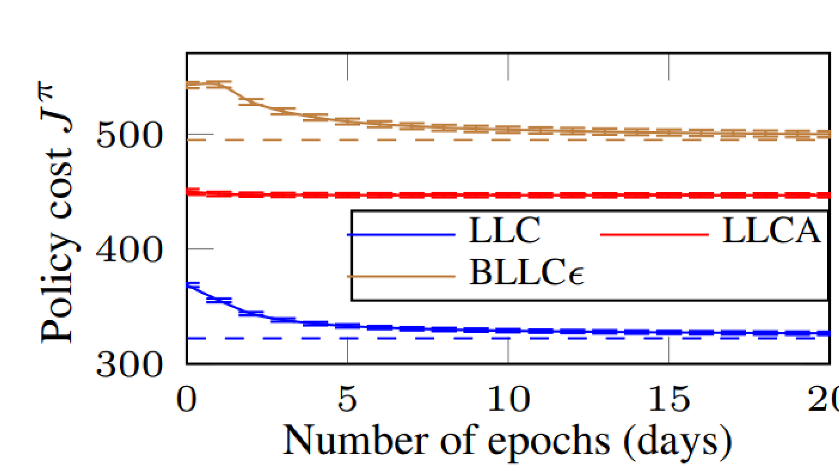
Experiments



LambdaCrawl (LC) outperforms BinaryLambdaCrawl (BLC) [1] w.r.t. harmonic penalty *and* evenw.r.t. binary penalty, for which BLC is optimal under incomplete



Using complete observations as LambdaCrawl-ComplObs (LC-CO) does yields huge performance gains



LambdaLearnAndCrawl converges quickly, and LambdaLearnAndCrawlApprox even faster (though to a worse solution)

References

[1] Y. Azar, E. Horvitz, E. Lubetzky, Y. Peres, and D. Shahaf. "Tractable Near-optimal Policies for Crawling." PNAS-2018
[2] J. Cho and H. Garcia-Molina. "Estimating frequency of change." ACM Transactions on Internet Technology, vol. 3(3), 2003
[3] J. Cho and H. Garcia-Molina. "Effective page refresh policies for web crawlers." ACM Transactions on Database Systems, vol. 28 (4), 2003