Optimal Web-scale Tiering as a Flow Problem

Gilbert Leung[1] | Novi Quadrianto[2] | Alex Smola[3] | Kostas Tsioutsiouliklis[3]

1: eBay | 2: SML-NICTA & RSISE-ANU | 3: Yahoo! Research

Abstract

- We present a fast online solver for large scale parametric max-flow problems as they occur in portfolio optimization, inventory management, computer vision, and logistics;
- Our algorithm solves an integer linear program in an online fashion;
- It exploits total unimodularity of the constraint matrix and a Lagrangian relaxation to solve the problem as a convex online game;
- The algorithm generates approximate solutions of max-flow problems by performing stochastic gradient descent on a set of flows;
- We apply the algorithm to optimize tier arrangement of over 80 Million web pages on a layered set of caches to serve an incoming query stream optimally.

Motivating Example

The Tiering Problem

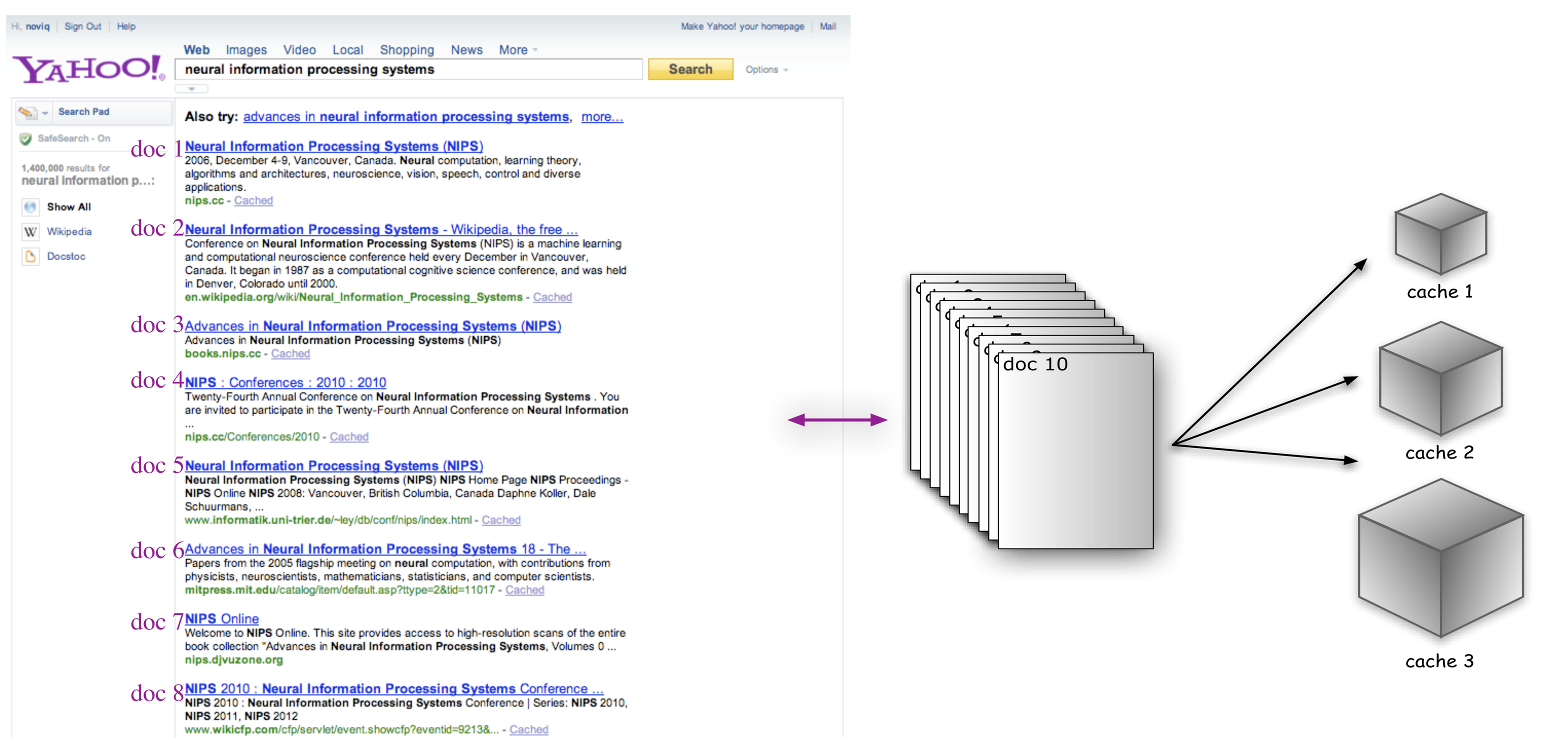
Goal:

- Select documents to be stored in successive tiers or caches of decreasing access frequency
- such that frequently accessed documents are found in the highest tiers
- thus the search engine will be able to cover incoming queries with low latency and computational load.

One proposed solution:

- Assign a value to each document and arrange them such that the highest valued documents reside in the highest levels of the cache;
- But this is sub-optimal.

Reason: to answer a given query well, a search engine returns not only a single document but a list of r (typically $r = 10$) documents.



Other Similar Problems

- Database record segmentation: queries $\rightarrow$ subsets of data items being retrieved by users and documents $\rightarrow$ all data items;
- Critical load factor determination in two-processor systems: queries $\rightarrow$ pairs of program modules that need to communicate with each other and documents $\rightarrow$ all program modules;
- Product portfolio selection: queries $\rightarrow$ historical orders and documents $\rightarrow$ products;
- ???.

Tiering Optimization Problem

Problem Setting

What we have:

- $d \in D$, the documents we would like to cache; $q \in Q$, the queries arriving at a search engine;
- $v_q \in (0, V)$, the value for a query q;
- $T = \{1, \ldots, k\}$, the k different tiers with its associated aggregate capacity C_t for $t' \leq t$;
- a bipartite graph G with vertices $D \cup Q$ and edges $(d, q) \in E$ whenever document d should be retrieved for query q;
- a penalty p_t of incurring a tier-miss of level $t > 1$.

What we want:

- an assignment of each document to a tier, $z_d \in T$.

Online Programming

The cost of access (per query) is determined by the worst case tier of the documents associated with the query, i.e. $u_q := \max_{d:(q,d)\in G} z_d$. Integer Programming:

$$\underset{z}{\text{minimize}} \sum_{q\in Q} v_q \sum_{j=1}^{\max_{d:(q,d)\in G} z_d} p_j \text{ s.t. } z_d \in \{1, \ldots, k\};\ \sum_{d\in D} I_{\{z_d \leq t\}} \leq C_t,\ \forall\, 1 \leq t \leq k \quad (1)$$

Two-tier (single cache system):

$$\underset{z}{\text{minimize}} \sum_{q\in Q} v_q \max_{d:(q,d)\in G} z_d \text{ s.t. } z_d \in \{0, 1\};\ \sum_{d\in D} z_d \geq |D| - C \quad (2)$$

(Reformulation as) Linear Integer Programming:

$$\underset{z,u}{\text{minimize}} \sum_{q\in Q} v_q u_q \text{ s.t. } u_q \geq z_d \text{ for all } (q, d) \in G;\ z_d, u_q \in \{0, 1\};\ \sum_{d\in D} z_d \geq |D| - C \quad (3)$$

(Relaxation as) Linear Programming:

$$\underset{z,u}{\text{minimize}} \sum_{q\in Q} v_q u_q - \lambda \sum_{d\in D} z_d \text{ s.t. } u_q \geq z_d \text{ for all } (q, d) \in G;\ z_d, u_q \in [0, 1];\ \lambda \geq 0 \quad (4)$$

(Reformulation as) Linear Programming (in term of one variable):

$$\underset{z}{\text{minimize}} \sum_{q\in Q} \underbrace{\left[v_q \max_{d:(q,d)\in G} z_d - \frac{\lambda}{|Q|} \sum_{d\in D} z_d \right]}_{\ell_q} \text{ s.t. } z_d, u_q \in [0, 1];\ \lambda \geq 0 \quad (5)$$

Algorithm

```
Initialize all z = 0
Set n = 100
for i = 1 to MAXITER do
  for all q ∈ Q do
    η = 1/√n (learning rate)
    n ← n + 1 (increment counter)
    Update z ← z − η∂_z ℓ_q(z)
    Project z to [0, 1]^D via z_d ← max(0, min(1, z_d))
  end for
end for
```

Experiments

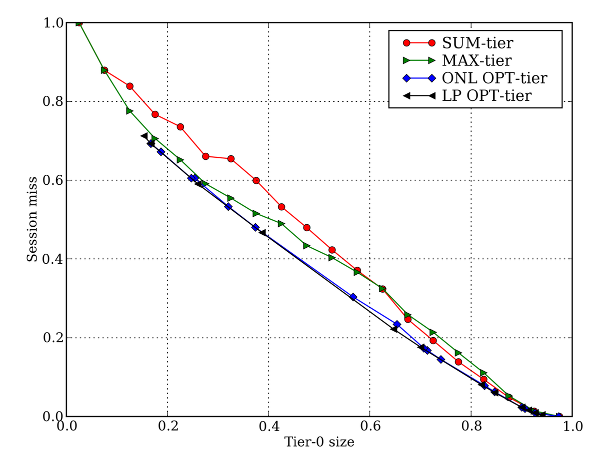
Practical Considerations:

- Lazy updates: only updating docs that are retrieved by a query. Define $s(n) := \sum_{j=1}^{n} \eta_j$ as an aggregate gradient step and let $\delta(n', n) := (s(n') - s(n))$. Its approximations:

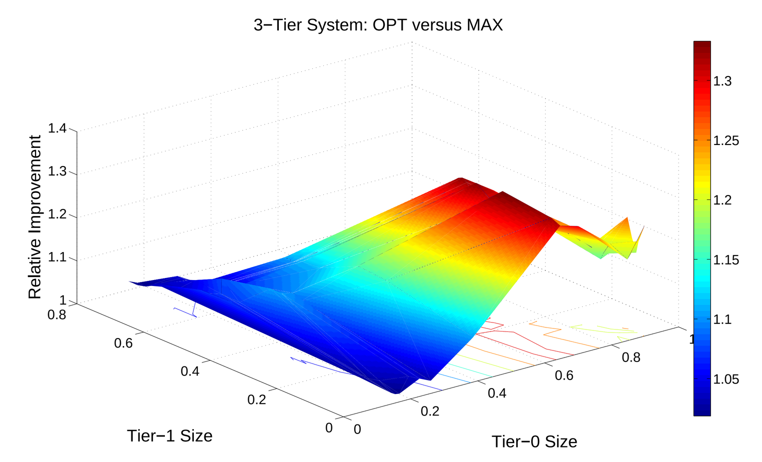
$$\delta(n', n) = \sum_{j=n+1}^{n'} \frac{1}{\sqrt{j + n_0}} \approx \int_n^{n'} \frac{1}{\sqrt{j + n_0}} 2\left[\sqrt{n' + n_0} - \sqrt{n + n_0}\right];$$

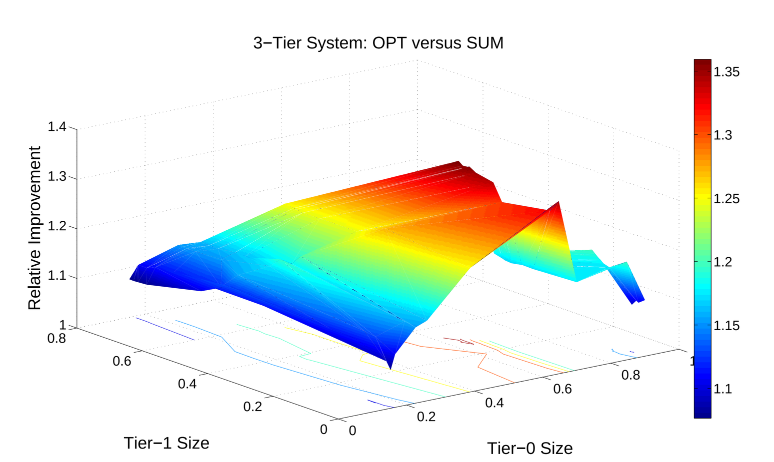
- Data reduction: any query occurring more frequently v_q than λ will automatically ensure that the associated pages are cached. As well, any document d for which $\sum_{q\in Q} v_q$ is displayed less than λ will definitely *not* be in the cache.

Toy Data Experiments



- A random bipartite query-page graph using 150 queries and 150 pages. Each query vertex has a degree of 3, and value $v_q := 10(2 + q)^{-0.8}$;
- Session miss evaluation: for each session q, a miss occurs if any one of the associated pages is not found in cache, incurring v_q misses for that session;
- Result comparisons with the max and sum heuristics;
- Left figures: 2-tier system; Bottom: 3-tier system.





Web Data Experiments

- Data come from the logs for one week of September 2009 containing results from the top geographic regions which include a majority of the search engine's user base;
- We only record a (query, document) pair, appears in top 10 (first result page) for a given session and we aggregate the view counts of such results, which will be used for the session value;
- In its entirety this subset contains about 10^8 viewed documents and $1.6 \cdot 10^7$ distinct queries. We excluded results viewed only once, yielding a final data set of $8.4 \cdot 10^7$ documents.