

# Optimal Web-scale Tiering as a Flow Problem

Gilbert Leung<sup>1</sup> | Novi Quadrianto<sup>2</sup> | Alex Smola<sup>3</sup> | Kostas Tsioutsoulis<sup>3</sup>

1: eBay | 2: SML-NICTA & RISE-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;
- 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;
- 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.

## Motivating Example

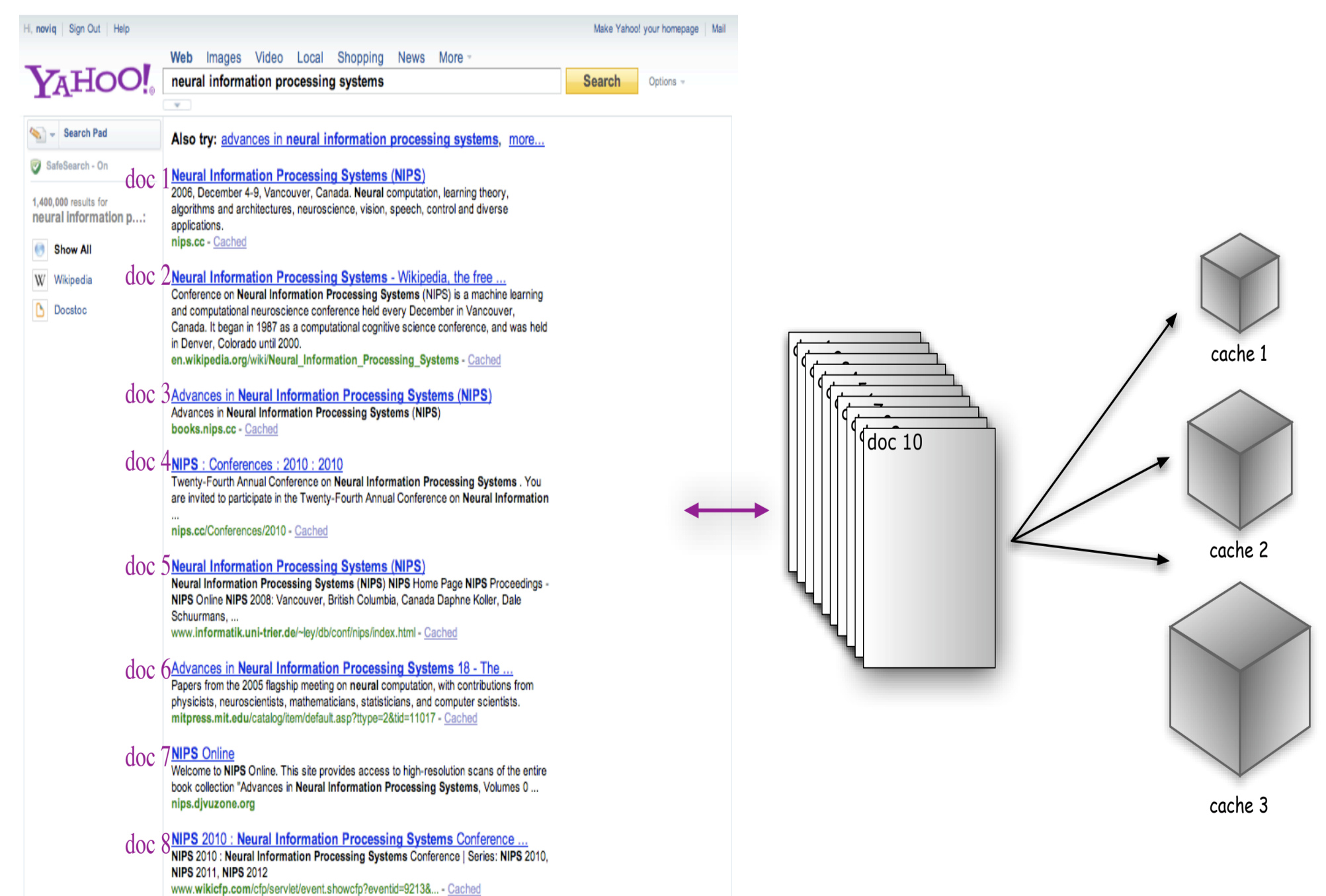
### The Tiering Problem

Goal:

- Cover incoming queries with **low latency and computational load**.
- Allocate documents to successive **tiers** or caches of decreasing **access frequency**, such that frequently accessed documents are found in the **highest tiers**.

Simple but Sub-Optimal Solution:

- Assign a value to each document and arrange them such that the highest valued documents reside in the highest levels of the cache;
- Sub-optimal**: to answer a given query well, a search engine returns not only a single but a **list** (of variable length, but typically 10) of documents.



## Other Applications

- Database record segmentation: optimize items frequently accessed together.
- Critical load factor determination in two-processor systems: optimize **pairs** of program modules that need to communicate with each other.
- Product portfolio selection: optimize products ordered together.

## Tiering Optimization Problem

- Documents**  $d \in D$  to be allocated to  $k$  **tiers**  $T = \{1, \dots, k\}$  (with  $k-1$  levels of cache): with  $t=1$  as prime or fastest cache, and  $t=k$  slowest or worst tier.
- Tier  $t$  has a cumulative capacity  $C_t$  (= sum capacities over  $t' \leq t$ ); and  $C_{k-1} < |D| \leq C_k$ .
- Queries**  $q \in Q$  from a **stream**, each with **value**  $v_q > 0$ , retrieves pages  $D_q \subseteq D$
- Bipartite graph**  $G$  with vertices  $D \cup Q$  and edges  $(d, q) \in E$  whenever document  $d$  should be retrieved for query  $q$ ; with  $|D|$  and  $|Q|$  very large (billions)
- Penalty**  $p_t > 0$  of falling through from tier  $t$  to  $t+1$ , i.e. tier-miss at level  $t < k$ .

## Integer Programming (Billion-Scale)

Assign each document  $d$  to a tier  $z_d \in T$ , minimizing total weighted access costs.

$$\begin{aligned} & \text{minimize} \sum_{z,u} v_q \sum_{t=1}^{u_q-1} p_t, \quad u_q := \max_{d \in D_q} z_d \\ & \text{subject to} \sum_{d \in D} \{z_d \leq t\} \leq C_t \text{ for each } t \in T \end{aligned} \quad (1)$$

Cost of query session  $q$  is determined by the **worst tier**  $u_q$  of the documents  $D_q$  retrieved.

## Two-Tier (single cache or $k=2$ ) Example

Define **fallthrough variables**  $x_d := z_d - 1$  and  $y_q := u_q - 1$  such that  $x_d, y_q \in \{0, 1\}$  and

$$x_d \leq y_q \text{ whenever } (q, d) \in G. \quad (2)$$

$$\text{minimize} \sum_{x,y} v_q y_q \quad \text{subject to (2) and } |D| - \sum_{d \in D} x_d \leq C_1 \quad (3)$$

**Relaxed Linear Program** with Lagrangian  $\lambda_1 \geq 0$ :

$$\text{minimize}_{x,y} \sum_{q \in Q} v_q y_q - \lambda_1 \sum_{d \in D} x_d \quad \text{subject to (2) and } x_d, y_q \in [0, 1] \quad (4)$$

**Total unimodular** constraints guarantee **integral** solution  $(x^*, y^*)$ , which induces observed capacity  $C_1^* = \sum_d x_d^*$ . Reformulate as an **Online Convex Program** in one variable:

$$\text{minimize}_x \sum_{q \in Q} \left[ v_q \max_{d \in D_q} x_d - \frac{\lambda_1}{|Q|} \sum_{d \in D} x_d \right], \quad x_d \in [0, 1]; \quad \lambda_1 \geq 0 \quad (5)$$

## General (multi-cache or $k > 2$ ) Formulation

**Thermometer Code**: For each  $d \in D$ , map  $z_d \rightarrow x_d \in \{0, 1\}^{k-1}$  such that  $x_{dt} \geq x_{d,t+1}$ . Thus,  $z_d = 1 + \sum_t x_{dt}$ . Similarly,  $u_q \rightarrow y_q \in \{0, 1\}^{k-1}$  with  $y_{qt} \geq y_{q,t+1}$ . Impose that

$$x_{dt} \leq y_{qt} \text{ for all } t \text{ whenever } (q, d) \in G. \quad (6)$$

**Relaxed Linear Program** with Lagrangians  $\lambda = (\lambda_1, \dots, \lambda_{k-1})^T$ :

$$\text{minimize}_{x,y} v^T y p - \mathbf{1}^T x \lambda, \quad \text{subject to (6) and } x_d, y_q \in [0, 1] \quad (7)$$

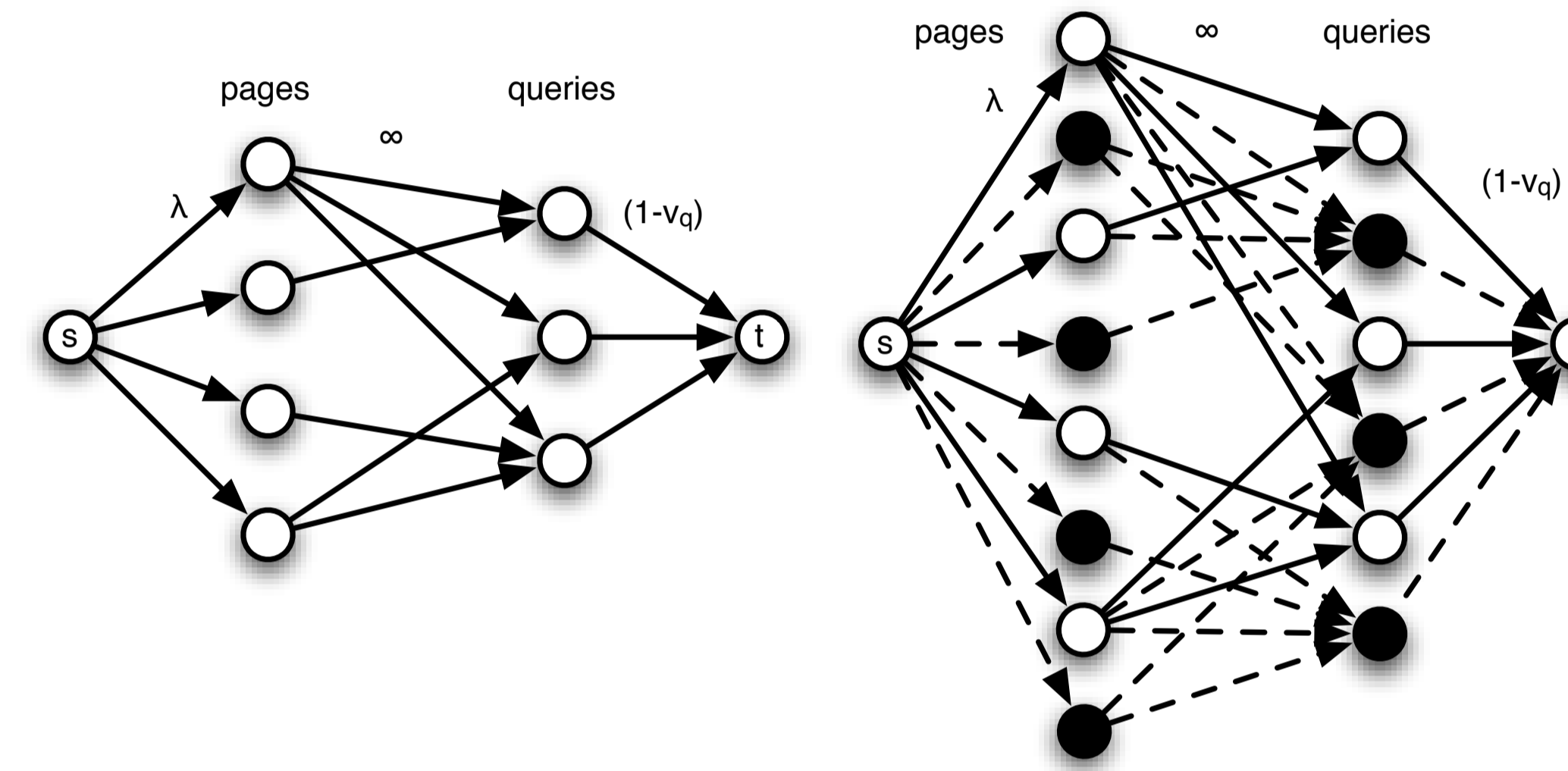
**Online Convex Programming**:

$$\text{minimize}_z \sum_q \ell_q \quad \text{with} \quad \ell_q(z) = v_q \max_{d \in D_q} (z_d - 1) + \frac{1}{|Q|} \sum_{d \in D} f_\lambda(z_d - 1) \quad (8)$$

where  $f_\lambda(x)$  is a convex extension of  $-\lambda_1 x$ , piecewise-linear with slope  $-\lambda_t$  for  $x \in (t-1, t)$ .

## Graph Cut Equivalence

Tiering optimization is equivalent to finding a maximum-flow, minimum-cut graph from source  $s$  to sink  $t$ .



**Left** Two-tier max-flow problem for 4 pages and 3 queries. The minimum cut of the directed graph needs to sever all pages leading to a query or alternatively it needs to sever the corresponding query incurring a penalty of  $(1 - v_q)$ .

**Right** Same configuration for 3 tiers. Black nodes and dashed edges represent a copy of the original graph — additionally each page in the original graph also has  $\infty$ -capacity link to the corresponding query in the additional graph.

## Implementation and Tricks

### Algorithm 1 Tiering Optimization

```

Initialize all  $z_d = 0$ 
Initialize  $n = 100$ 
for  $i = 1$  to MAXITER do
  for all  $q \in Q$  do
     $\eta = \frac{1}{\sqrt{n}}$  (learning rate)
     $n \leftarrow n + 1$ 
    Update  $z \leftarrow z - \eta \nabla \ell_q(z)$ 
    Project  $z$  to  $[1, k]^D$  via
     $z_d \leftarrow \max(1, \min(k, z_d))$ 
  end for
  aggregate gradient steps  $s(n) := \sum_{j=1}^n \eta_j$ . Integral approximation yields
  lazy updates  $\delta(n', n) := s(n') - s(n) \approx 2 \left[ \sqrt{n' + n_0} - \sqrt{n + n_0} \right]$ .
end for

```

- input data stream: query-value-result records  $(q, v_q, D_q)$
- update the tier preferences of the pages with the lowest scores for each level
- decrement the preferences for all other pages (lazy-update only pages retrieved by query)
- aggregate gradient steps  $s(n) := \sum_{j=1}^n \eta_j$ . Integral approximation yields **lazy updates**  $\delta(n', n) := s(n') - s(n) \approx 2 \left[ \sqrt{n' + n_0} - \sqrt{n + n_0} \right]$ .

- solutions for increasing  $\lambda_t$  form a **nested subset**: relaxing capacity constraints never demotes but only promotes pages.
  - average binary solutions  $(x_{dt}^*)$  over the entire solution path provides an ordering of pages into tiers.
- In practice,
- no need to run each  $\lambda$  value till convergence if averaging
  - even  $|\Lambda| = 5$  values of  $\lambda$  recover LP solver (optimal but slow) results for 2 tiers.

### Algorithm 2 Path Following

```

Initialize all  $(x_{dt}) = z_d \in [1, k]$ 
for each  $\lambda \in \Lambda$  do
  Refine variables  $x_{dt}(\lambda)$  by Algorithm 1 using a small number of iterations.
end for
Average the variables  $x_{dt} = \sum_{\lambda \in \Lambda} x_{dt}(\lambda) / |\Lambda|$ 
Sort the documents with the resulting total scores  $z_d$ 
Fill the ordered documents to tier 1, then tier 2, etc.

```

## Experiments

### Practical Considerations:

- may optimize with several  $\lambda$  values simultaneously: the main cost is sequential RAM read-write access rather than CPU speed
- Let  $Q_d$  be all queries retrieving page  $d$ , and

$$m_d := \max_{q \in Q_d} v_q \quad \text{and} \quad s_d := \sum_{q \in Q_d} v_q \quad (9)$$

The **max** and **sum** heuristics rank and tier pages by above scores.

- Two-tier case: any frequent query  $q$  with  $v_q > \lambda$  should have its associated pages  $D_q$  cached; Any document  $d$  for which  $s_d < \lambda$  will definitely **not** be in the cache.

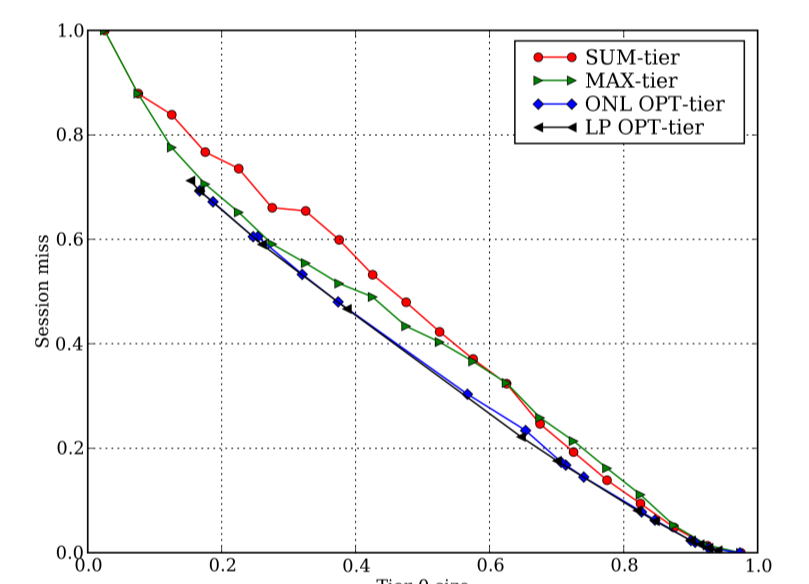
### Toy Data Experiments

- random query-page graph with 150 queries and 150 pages, each query retrieves 3 pages and  $v_q := 10(2 + q)^{-0.8}$

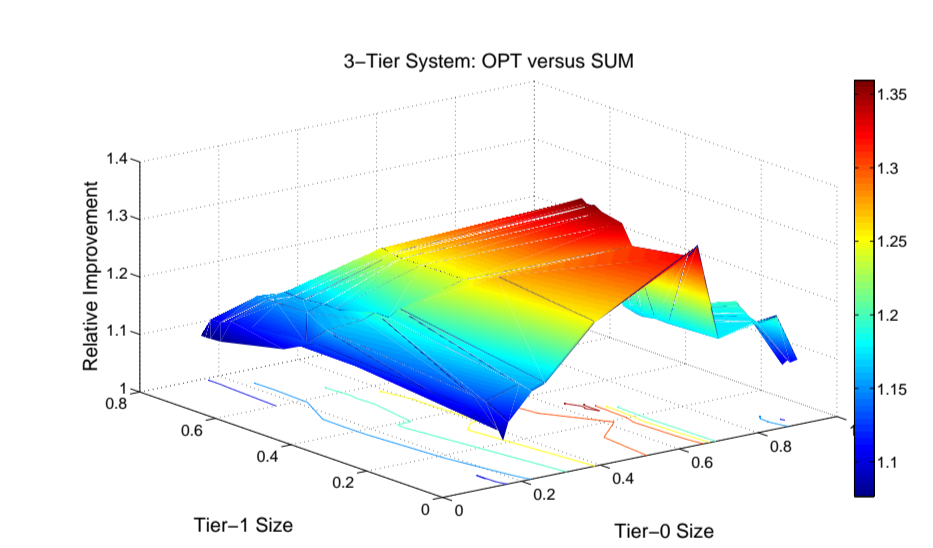
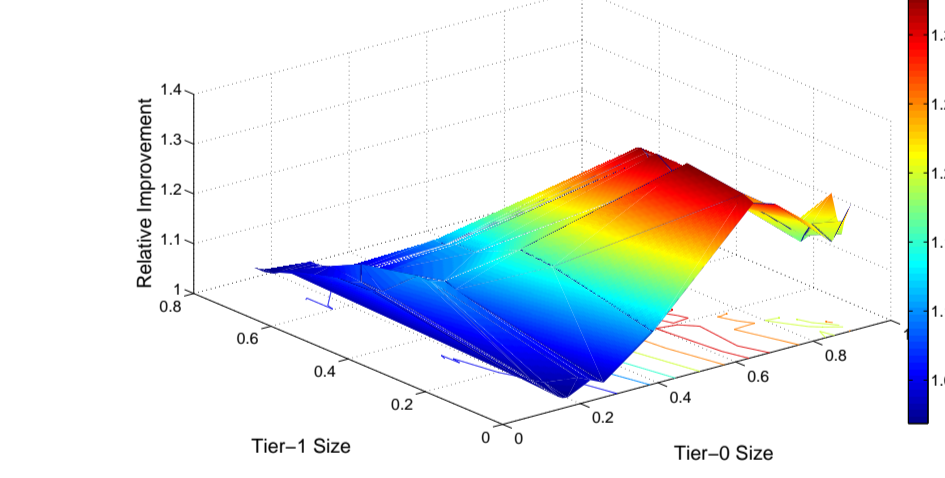
Two-tier optimization: (right)

- Session miss** evaluation: for each session, a miss occurs if any of the associated pages is not in cache, incurring  $v_q$  misses for query  $q$

Three-tier optimization: (below)



3-Tier System: OPT versus MAX



## Real Web-Scale 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;
- only record  $(q, d)$  pairs in **top 10** (first result page) for a given session ; aggregate the view counts to form  $v_q$
- dataset contains about  $10^8$  viewed documents and  $1.6 \cdot 10^7$  distinct queries. We excluded results viewed only once, yielding a final dataset of  **$8.4 \cdot 10^7$  documents**.
- Using **Path Following** interpolation with  $|\Lambda| = 5$ , this run requires only 2GB RAM and 20 minutes.

