Optimal Web-scale Tiering as a Flow Problem

Gilbert Leung1 | Novi Quadrianto2 | Alex Smola3 | Kostas Tsioutsiouliklis3

Abstract

Motivating Example

The Tiering Problem

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;
- \(v_q \in (0, V)_t\), the value for a query \(q\);
- \(t = \{1, \ldots, k\}\), the \(k\) different tiers with its associated aggregate capacity \(C_t\) for \(t \leq k\);
- 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 \(p_q\) 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. \(u_q = \max_{d \in D \cap Q} z_d\).

Integer Programming:

\[
\min \sum_{q \in Q} \max_{z \in \{0, \ldots, C_t\}} \sum_{j=1}^{r} p_j z_j \quad s.t. \quad z_j \in \{1, \ldots, k\}, \sum_{j=1}^{r} z_j |D| \leq C_t, \quad 1 \leq t \leq k
\]

Two-tier (single cache system):

\[
\min \sum_{q \in Q} \max_{z \in \{0, 1\}} \sum_{j=1}^{2} p_j z_j \quad s.t. \quad z_j \in \{0, 1\}, \sum_{j=1}^{2} z_j \leq |D| = C
\]

(Reformulation as) Linear Integer Programming:

\[
\min \sum_{q \in Q} \max_{z \in \{0, 1\}} \sum_{j=1}^{2} p_j z_j \quad s.t. \quad z_j \geq 0 \quad \forall (q, d) \in G, \quad z_q, u_q \in \{0, 1\} \quad \sum_{j=1}^{2} z_j \leq |D| = C
\]

(Reformulation as) Linear Programming:

\[
\min \sum_{q \in Q} \max_{z \in \{0, 1\}} p_q z_q - \lambda \sum_{j=1}^{2} z_j \quad s.t. \quad z_j \geq 0 \quad \forall (q, d) \in G, \quad z_q, u_q \in \{0, 1\} \quad \lambda \geq 0
\]

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

\[
\min \sum_{q \in Q} p_q z_q - \lambda \sum_{j=1}^{2} z_j \quad s.t. \quad z_j \geq 0 \quad \forall (q, d) \in G, \quad z_q, u_q \in \{0, 1\} \quad \lambda \geq 0
\]

Algorithm

Initializie all \(z = 0\)
Set \(s = 100\)
for \(i = 1\) to MAXITERS do
for all \(q \in Q\) do
\(\eta = \epsilon / \lambda\) (learning rate)
\(n = n + 1\) (increment counter)
Update \(z_q = z_q - \eta p_q / \sum_{j=1}^{r} p_j z_j\)
Project \(z\) to \([0, 1]^{|D|}\) via \(z_j \leftarrow \min(0, \min(1, z_j))\)
end for
end for

Experiments

Practical Considerations:

- Lazy updates: only updating docs that are retrieved by a query. Define \(s(n) = \sum_{q \in Q} q_j\) as an aggregate gradient step and let \(d(n, t) = s(n) - s(n-1)\). Its approximations:

\[
d(n, t) = \sum_{j=1}^{n} \frac{d_j(n, t)}{n} = \frac{1}{n} \sum_{j=1}^{n} \left[ q_j \sum_{q \in Q} z_q - \sum_{j=1}^{n} v_q \right].
\]

- Data reduction: any query occurring more frequently than \(T\) will automatically ensure that the associated pages are cached. As well, any document \(d\) for which \(\sum q_j z_q\) is displayed less than \(T\) 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 \(V = 100 = 10^2\).
- Session miss evaluation: for each session \(q\), a miss occurs if any one of the associated pages is not found in cache, incurring \(q_j\) 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, documents) 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 10^6 viewed documents and 1.6 - 10^7 distinct queries. We excluded results viewed only once, yielding a final data set of 8.4 - 10^6 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;
- 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;