Optimization for Search via Consistent Hashing & Balanced Partitioning

Vahab Mirrokni
NYC Algorithms Research, Google Research
NYC Algorithms overview

Ad Optimization (search & display)

common expertise: online allocation problems

Large-Scale Graph Mining

tools: PPR, local clustering, ...

Infrastructure & Large-Scale Optimization

tools: balanced partitioning
Outline: Three Stories

● Consistent Hashing for Bounded Loads
● Application of Balanced Partitioning to Web search
  ○ Main idea: cluster query stream to improve caching
  ○ Balanced Graph Partitioning: Algorithms and Empirical Evaluation
● Online Robust Allocation
  ○ Simultaneous Adversarial and Stochastic Optimization
  ○ Mixed Stochastic and Adversarial Models
Consistent Hashing with Bounded Loads for Dynamic Bins

- Vahab Mirrokni (Google NYC)
- Mikkel Thorup (Visitor / U. Copenhagen)
- Morteza Zadimoghaddam (Google NYC)
Problem: Consistent Hashing for Dynamic Bins

- Hash balls into bins
- Both balls and **bins are dynamic**
- Main Objectives:
  - **Uniformity:** Hard capacities
  - **Consistency:** Minimize movements
- Remarks:
  - Update time is not the main concern
  - We need a memoryless system based on state (balls/bins)
Previous Approaches

- Consistency Hashing/Chord (Dynamic): Hash balls and bins into a circle, and put each ball in the next bin on the circle.
- Power of two choices (Static): Try two random bins & send to the smaller one.

Active balls and bins are marked with blue.
# Related Work

<table>
<thead>
<tr>
<th></th>
<th>Max Load</th>
<th>Avg Relocation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Chord</strong> [Stoica, Morris, Karger, Kaashoek, Balakrishnan 2001] <strong>Consistent Hashing</strong> [Karger, Lehman, Leighton, Panigrahy, Levine, Lewin 1997]</td>
<td>(\text{density} \times \log(n) / \log\log(n))</td>
<td>(O(\text{density}))</td>
</tr>
<tr>
<td>Totally Random Hash Function</td>
<td>(\text{density} \times \log(n) / \log\log(n))</td>
<td>(O(\text{density}))</td>
</tr>
<tr>
<td><strong>Balanced Allocations</strong> [Azar, Broder, Karlin, Upfal 1999] <strong>Cuckoo Hashing</strong> [Pagh, Rodler 2001]</td>
<td>(\text{density} \times \log\log(n))</td>
<td>(O(\text{density}))</td>
</tr>
<tr>
<td>Linear Probing with tight capacity</td>
<td>(\text{density})</td>
<td>Large in simulations - Cycle length in a random permutation (\Omega(n))?</td>
</tr>
<tr>
<td><strong>Our approach: Linear Probing with (1+(\varepsilon)) extra multiplicative capacity</strong></td>
<td>(\text{density} \times (1+\varepsilon))</td>
<td>(O(\text{density}/\varepsilon^2))</td>
</tr>
</tbody>
</table>

* density is the average load, i.e. number of balls divided by number of bins*
Results: Provable performance guarantees

Method: Linear Probing with \((1+\epsilon)\) extra multiplicative capacity

- Uniformity: max load is \((1+\epsilon) \times \text{average load}\)
- Relocations is at most:
  - \(O(1/\epsilon^2)\) per ball operation for \(\epsilon < 1\)
  - \(1 + O(\log(1+\epsilon)/\epsilon^2)\) per ball operation for \(\epsilon > 1\) (theoretical)
  - The bounds for bin operation is multiplied by density = \#balls / \#bins
- For \(\epsilon > 1\), the extra relocation term disappears in the limit
Take-home point 1

- You want to achieve desirable load balancing with consistency in dynamic environments? Then use:

  **Linear probing with \((1+\varepsilon)\) extra multiplicative capacity**

- Good theoretical and empirical properties for:
  - Load Balancing: Deals with hard capacities
  - # of Movements: Bounded by a constant \(O(\text{density}/\varepsilon^2)\)
Application of Balanced Partitioning to Web search

○ Eng Team: Bartek Wydrowski, Ray Yang, Richard Zhuang, Aaron Schild (PhD intern, Berkeley)
○ Research Team: Aaron Archer, Kevin Aydin, Hossein Bateni, Vahab Mirrokni
Balanced graph partitioning

- Given graph $G = (V,E)$ with:
  - node weights $w_v$
  - edge costs $c_e$
  - # clusters $k$
  - imbalance tolerance $\epsilon > 0$

- Goal: partition $V$ into sets $P = \{C_1, ..., C_k\}$ s.t.
  - node weight balanced across clusters, up to $(1+\epsilon)$ factor
  - minimize total cost of edges cut
Some observations in Web search backend

- Caching is very important for efficient Web search.
- Query stream more uniform → caching more efficient.
- A lot of machines are involved.

Idea:

Try to make query stream more uniform at each cache.
Routing Web search queries

- Machine layout: $R$ roots, sharing $L$ leaves
- The corpus is doc-sharded.
- Each leaf serves 1 shard.
  - Root forwards query to 1 replica in each shard, combines leaf results.

Q: For each shard, which replica to pick?

[Old answer] Uniformly at random.

[New answer] This talk.
Design

- [Old] Root selects leaf uniformly at random.
  - Leaf caches look ~same.
- [New] Terms in query vote based on clustering.
  - Specializes cache in replica $r$ to terms in cluster $r$.

Example diagram with $k=3$ replicas.
Algorithm

Offline:
Leaf logs → term-query graph.
Cluster terms into k buckets, using balanced graph partitioning.
Store term-bucket affinity mapping.

Online:
Root loads term-bucket affinities into memory at startup.
Terms in query hold weighted vote to select replica $r$.
Send query to replica $r$ for each doc shard.
Clustering objectives

**Balanced**: Aim for roughly equal working set size in each cluster.

**Small cut size**: cut \{term, query\} edge $\leftrightarrow$ query assigned to different cluster than term, so probable cache miss.
Clustering solution

Example clustering with $k=3$ replicas.

Cut edges: query routed to non-preferred replica for that term, so less likely to be in cache
Input to balanced partitioner

- $p_t = \Pr[\text{term t in cache in } \textit{preferred} \text{ replica}]$
- $q_t = \Pr[\text{term t in cache in any } \textit{non-preferred} \text{ replica}]$
- $size_t = \text{size of t's data in memory pages} = \text{cost of cache miss}$

$$c_{\{\text{cat, cat video}\}} = (p_{\text{cat}} - q_{\text{cat}}) \cdot \text{size}_{\text{cat}}$$

$$w_{\text{cat}} = p_{\text{cat}} \cdot \text{size}_{\text{cat}}$$
Balanced Partitioning via Linear Embedding

Kevin Aydin, Hossein Bateni, Vahab Mirrokni, WSDM 2015

Paper Here
Balanced graph partitioning

● Given graph $G=(V,E)$ with:
  ○ node weights $w_v$
  ○ edge costs $c_e$
  ○ # clusters $k$
  ○ imbalance tolerance $\epsilon > 0$

● Goal: partition $V$ into sets $P=\{C_1, \ldots, C_k\}$ s.t.
  ○ node weight balanced across clusters, up to $(1+\epsilon)$ factor
  ○ minimize total cost of edges cut
We need scalable, *distributed* algorithms

- O(1)-apx. NP-hard, so rely on principled heuristics.
- Example run of our tool:
  - 100M nodes, 2B edges
  - <1 hour on 1000 machines
- Uses affinity clustering as a subroutine.
- Affinity scalability:
  - 10B nodes, 9.5T edges
  - 20 min on 10K machines
Linear embedding: outline of algorithm

Three-stage algorithm:

1. Reasonable initial ordering
   - hierarchical clustering

2. Semi-local moves
   - improve by swapping pairs

3. Introduce imbalance
   - dynamic programming
   - min-cut
Step 1: initial embedding

- Space-filling curves (geo graphs)
- Hierarchical clustering (general graphs)
Affinity hierarchical clustering

- Keep heaviest edge incident to each node.
- Contract connected components.
- Scalable parallel version of Boruvka's algorithm for MST.

iterate
Datasets

- **Social graphs**
  - Twitter: 41M nodes, 1.2B edges (source: [KLPM'10])
  - LiveJournal: 4.8M nodes, 42.9M edges (source: SNAP)
  - Friendster: 65.6M nodes, 1.8B edges (source: SNAP)

- **Geo graphs**
  - World graph: 500M+ nodes, 1B+ edges (source: internal)
  - Country graphs (filtered versions of World graph)
Related work

- FENNEL [Tsourakakis et al., WSDM’14]
  - Microsoft Research
  - Streaming algorithm
- UB13 [Ugander & Backstrom, WSDM’13]
  - Facebook
  - Balanced label propagation
- Spinner [Martella et al., arXiv'14]
- METIS (in-memory) [Karypis et al. ’95-'15]
Comparison to previous work: LiveJournal graph

Cut size as a percentage of total edge weight in graph. (x%) denotes imbalance.

<table>
<thead>
<tr>
<th>k</th>
<th>Spinner (5%)</th>
<th>UB13 (5%)</th>
<th>Affinity (0%)</th>
<th>Combination (0%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>38%</td>
<td>37%</td>
<td>35.71%</td>
<td>27.5%</td>
</tr>
<tr>
<td>40</td>
<td>40%</td>
<td>43%</td>
<td>40.83%</td>
<td>33.71%</td>
</tr>
<tr>
<td>60</td>
<td>43%</td>
<td>46%</td>
<td>43.03%</td>
<td>36.65%</td>
</tr>
<tr>
<td>80</td>
<td>44%</td>
<td>47.5%</td>
<td>43.27%</td>
<td>38.65%</td>
</tr>
<tr>
<td>100</td>
<td>46%</td>
<td>49%</td>
<td>45.05%</td>
<td>41.53%</td>
</tr>
</tbody>
</table>
Comparison to previous work: Twitter graph

Cut size as a percentage of total edge weight in graph. (x%) denotes imbalance.

<table>
<thead>
<tr>
<th>k</th>
<th>Spinner (5%)</th>
<th>Fennel (10%)</th>
<th>Metis (2-3%)</th>
<th>Combination (0%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>15%</td>
<td>6.8%</td>
<td>11.98%</td>
<td>7.43%</td>
</tr>
<tr>
<td>4</td>
<td>31%</td>
<td>29%</td>
<td>24.39%</td>
<td>18.16%</td>
</tr>
<tr>
<td>8</td>
<td>49%</td>
<td>48%</td>
<td>35.96%</td>
<td>33.55%</td>
</tr>
</tbody>
</table>
Main result of 2nd part

25% fewer cache misses!

Translates to greater QPS throughput for the same hardware.
Take-home point 2

- Fundamental optimization models + good logs data + scalable algorithms → big improvements in data center operations.
- When splitting query stream to distribute load, might as well cluster it to improve caching.
  - Idea is generally applicable; nothing special about Web search!
Online (Robust) Ad Allocation

Why (not) to rely on data
Online Ad Allocation: Budgeted Allocation

- Budgeted Fixed Nodes (Advertisers), Online Nodes (Users), and Weighted Edges between them

Goal: Assign online nodes to Advertisers maximizing revenue respecting budgets

Revenue(Greedy) = 4 + 2 + 2 = 8
Revenue(Optimum) = 2 + 1 + 6 + 1 + 1 = 11

Performance Ratio = 8/11

Depends on Instance and Arrival Order of Online Nodes
Online Weighted Matching (Display Ads)

Fixed Nodes (Advertisers) with Capacities, Online Nodes (Users), and weighted edges between them

**Goal**: Assign online nodes to Advertisers maximizing weight of the allocation respecting capacities

Performance of Green Allocation: Cardinality 3, Weight 8
Algorithm Alg is $\alpha$-competitive if:

- **Worst Case/Adversarial**
  \[
  \frac{\text{Revenue(Alg)}}{\text{Revenue(OPT)}} > \alpha
  \]
  should hold for all instances and arrival orders

- **Stochastic**
  \[
  \frac{\mathbb{E}[\text{Revenue(Alg)}]}{\text{Revenue(OPT)}} > \alpha
  \]
  should hold for all instances

We take expected values over all instances
Online Ad Allocation: Adversarial order

Theorem [MSVV’05, FKMMP’09]: In worst-case, Primal-dual Algorithm is \((1-1/e)\)-competitive.

Greedy is \((1/2)\)-competitive

\[
\begin{align*}
\max & \sum_{i,a} b_{ia} x_{ia} \\
\sum_a x_{ia} & \leq 1 \quad (\forall i) \\
\sum_i b_{ia} x_{ia} & \leq B(a) \quad (\forall a) \\
x_{ia} & \geq 0 \quad (\forall, i, a)
\end{align*}
\]
Primal-dual Algorithm [FKMMP’09, FKHMS’11]

Primal

\[
\begin{align*}
\max & \quad \sum_{i,a} w_{ia} x_{ia} \\
\sum_{a} x_{ia} & \leq 1 \quad (\forall \, i) \\
\sum_{i} x_{ia} & \leq n(a) \quad (\forall \, a) \\
x_{ia} & \geq 0 \quad (\forall \, i, a)
\end{align*}
\]

Dual

\[
\begin{align*}
\min & \quad \sum_{a} n(a) \beta_{a} + \sum_{i} z_{i} \\
\beta_{a} + z_{i} & \geq 0 \quad (\forall \, i, a) \\
\beta_{a}, z_{i} & \geq 0 \quad (\forall \, i, a)
\end{align*}
\]

- Primal solution shows the allocation
- Maintain Dual Variable $\beta_{a}$ for each advertiser $a$
- Assign $i$ to advertiser $a$, maximizing: $w_{ia} - \beta_{a}$
- Update $\beta_{a}$ online after each allocation (function of capacity constraint). Initialize at 0.
Theorem [DH’09, FHKMS’11]: In stochastic model, dual-based algorithm is a \((1-\varepsilon)\)-competitive.

Assumptions are invalid! Data is an imperfect guide.
Reality is Far from Predictions: Traffic Spikes

Breaking news

One-off events

Exciting sporting events
Use Forecasts, but Don’t Trust Them Completely

Hybrid algorithm:
Learn Duals, blend them with adversarial duals

Theorem: ???
Goal: A Theory of Partially Accurate Forecasts
Simultaneous Adversarial & Stochastic Approximation

- Adversarial Setting: too pessimistic, real world data has non-adversarial structure
- Stochastic Input: too optimistic, violated by traffic spikes

**Practice:** mixture of these two settings is used

**Goal:** theoretically model these mixture settings

**One way to model:** simultaneous competitive algorithms

**Goal:** Design robust algorithms that achieve the best competitive ratios in each case, i.e. robust against traffic spikes and performs better in stochastic case.
Simultaneous Adversarial & Stochastic Approximation

[Mirrokni Oveis Gharan Zadimoghaddam, SODA 12]

<table>
<thead>
<tr>
<th></th>
<th>Algorithm</th>
<th>Hardness</th>
</tr>
</thead>
<tbody>
<tr>
<td>un-weighted</td>
<td>(1-1/e, 1-(\varepsilon)) [KVV], Ours</td>
<td>(1-1/e, 1) [KVV]</td>
</tr>
<tr>
<td>weighted</td>
<td>Balance gets</td>
<td>(4(\varepsilon)^{1/2}, 1-(\varepsilon))</td>
</tr>
<tr>
<td></td>
<td>(1-1/e, 0.76) [MSVV], Ours</td>
<td>(1-1/e, 0.976) Ours</td>
</tr>
</tbody>
</table>
**Simultaneous Adversarial & Stochastic Approximation**

For **unweighted instances**: Assigning each online node to the least congested advertiser achieves $(1-1/e, 1-\varepsilon)$-approximation in adversarial and stochastic settings with large budgets.

**Our Result for weighted instances:**
We show that primal-dual algorithm achieves $(1-1/e, 0.76)$-approximation of with large budgets.
Neither Adversarial nor Stochastic

Reality is not bimodal!

Every day, forecasts are a ‘little’ inaccurate.

Small, but non-random (adversarial?) deviations from forecast.

Design algorithms with performance that degrades gracefully with forecast accuracy?

![Comparing Algorithms Graph]

- Greedy: 69.8
- Primal-Dual: 82.6
- Learning Duals: 87.2
Modeling Traffic Spikes

Algorithm knows forecast: f items from distribution D (with finite support)

1. At each time step, adversary can either:
   a. Create an arbitrary item
   b. Draw an item from D

2. After f items have been drawn from D, adversary can terminate input.

Measure forecast accuracy by parameter $\lambda$: How much noise did adversary add?

$\lambda = \frac{\text{OPT(Forecast)}}{\text{OPT(Forecast } \cup \text{ Adversarial Items)}}$

[Esfandiari, Korula, Mirrokni, EC’15]
Allocating with Traffic Spikes

Allocate items according to forecast, ‘reserving’ budget for forecast items. When algorithm detects adversarial items, use worst-case algorithm to assign, using remaining budgets.
Take-home Points 3: Robust Ad Allocation

- Good algorithms for online allocation in adversarial settings
- Good algorithms in stochastic settings
- Hybrid settings? Need better models for partially accurate forecasts!
  - Simultaneous Adversarial and Stochastic Approximation → SODA 2012 paper
  - Mixed Adversarial and Stochastic Models → EC 2015 paper
Conclusion: Examples of “Algorithms in the Field”

“Examples of “Algorithms in the Field of Infrastructure Optimization”

1. Dynamic load balancing with bounded hashing: Enhanced Linear Probing
2. Balanced partitioning to improve caching in web search and beyond
3. Online Optimization: Hybrid adversarial and stochastic models.
   - Simultaneous Adversarial and Stochastic Approximation → SODA 2012 paper
   - Mixed Adversarial and Stochastic Models → EC 2015 paper
NYC Algorithms overview

Ad Optimization (search & display)

common expertise: online allocation problems

Large-Scale Graph Mining
tools: PPR, local clustering, ...

Infrastructure & Large-Scale Optimization
tools: balanced partitioning