COT 6936: Topics in Algorithms

Giri Narasimhan
ECS 254A / EC 2443; Phone: x3748
giri@cs.fiu.edu

http://www.cs.fiu.edu/~giri/teach/COT6936_S12.html

https://moodle.cis.fiu.edu/v2.1/course/view.php?id=**174**

COT 6936: Topics in Algorithms

Online Algorithms

Randomized Algorithm: RANDOM

- On a miss:
  - Evict an item chosen uniformly at random from all k items

Randomized Marker Algorithm

- Each of k pages has a marker bit
- Algorithm proceeds in rounds with invariant:
  - At start of round all pages are unmarked
- In each round
  - If request is a hit: mark page
  - If request is a miss:
    - If all cache pages are marked: start next round by unmarking all locations
    - Else evict (randomly) unmarked page and mark it
- This algorithm is $2H_k$-competitive

Only change

Marker Algorithm

- Marker algorithm is k-competitive
  - In each round, algorithm has k misses
  - OPT has at least one miss because $k+1$ distinct pages are accessed including the last access from previous round
- Randomized Marker Algorithm is (ln k)-competitive
- Random is k-competitive

Load Balancing

- Jobs arrive in a stream to be processed immediately on one of many processors
  - n processors and m jobs
- Centralized algorithm
  - Round robin ensures that each processor gets at most ceil(m/n) jobs
- Assume centralization is not possible
  - Assign jobs uniformly at random to processors
  - How do we analyze the situation? How balanced is the assignment?

Randomized Load Allocation: Analysis

- Y_{ij} = indicator (binary) r.v. for event [job j is assigned to processor i]
  - $E[Y_{ij}] = 1/n$
- X_i = r.v. for number of jobs to processor I
- $E[X_i] = \Sigma_j E[Y_{ij}] = m/n$
- $\Pr[X_i > c] < e^{c-1}/e^c$ using Chernoff bounds

Randomized Load Allocation: Analysis

- Case $m = n$
  - With high probability (at least $1-(1/n)$), no processor gets more than $\Theta(\log n/\log\log n)$ jobs
- As m increases, imbalance goes away!
  - Choose $m > 16n \ln n$
- Case $m = \Omega(n \log n)$
  - With high probability (at least $1-(1/n^2)$), every processor gets between $m/2n$ and $2m/n$ jobs

Packet Routing in Distributed Systems

- Model network as a directed graph
- Every message is discretized into packets each sent separately on a s-t path
- Constraint: 1 packet/step through edge e
  - Need to queue requests along edge e
  - Decide when to release packet from s (schedule)
  - Need queue management policy
    - Prioritize packets with closest/farthest destinations?
- Goal: Find a schedule of minimum duration

Packet Routing in Distributed Systems 2

- Packets $1, \ldots, N$ and paths $P_1, \ldots, P_N$
- Packet Schedule: which packet through which edge at what time
- Minimize Duration: for every packet to be delivered to destination
- Obstacles:
  - Long paths (duration ≥ d longest path length)
  - Congestion on bottleneck edges (duration ≥ c number of paths sharing same edge)
  - Clearly, Duration ≥ $\max(c,d) = \Omega(c+d)$

Packet Routing in Distributed Systems 3

- Schedule: choose arbitrarily or FCFS
  - Duration = $O(cd)$
  - Could happen in real situations because packets are very badly timed with respect to each other and groups of size c reach edge e simultaneously
  - Need better solution

Randomized Algorithm

- Scheduling Algorithm for each packet i
  - Packet i chooses random delay s from range $[1..r]$ and waits at source for s steps
  - Then packet attempts to move one edge per step until destination is reached
- If delays chosen so that no 2 packets reach same edge at same time, then duration = $r+d$
  - Problem: r needs to be very large

Randomized Alg.: Group time into blocks

- Scheduling Algorithm for each packet i
  - Group b consecutive steps into blocks
  - Packet i chooses random delay s from range $[1..r]$ and waits at source for s blocks
  - Then packet attempts to move one edge per block until destination is reached
- Result [Leighton, Maggs, Rao, 88]
  - If event E (more than b packets arrive at edge e at start of same block) does not occur, then duration is at most $b(r + d)$

Randomized Packet Routing: Analysis

- Bound Pr(E)
  - Union of events by considering each edge
  - Union of events by considering each time block
- F_{et} = event $[N_{et} > b]$
- N_{et} = r.v. for # of packets scheduled to appear at start of block t at edge e
- X_{eti} = indicator (binary) r.v. for event [packet i appears at start of block t at edge e]
- $Pr(E) = Pr[U_{e,t}F_{et}]$ and $E[N_{et}] = \Sigma_i E[X_{eti}]$

Randomized Packet Routing: Analysis

- Theorem:
  - Choose $r = c / (q \log(mN))$ and $b = 3c/r$
    - Here q is a carefully chosen constant
  - With high probability, the duration of the schedule for the packets is $O(c + d \log(mN))$
    - N = # of packets
    - m = # of edges

Permutation Routing Problem

- Model
  - Given directed graph on N vertices (processors)
  - Communication in synchronous steps
  - At most one packet per link per step
  - Each processor needs to send one packet to a unique destination (Destinations ≅ Permutation)
  - Need route and queueing discipline for conflicts
  - How many steps does the whole thing take?

Permutation Routing Problem

- Oblivious algorithm
  - route only depends on destination
- Theorem:
  - For any deterministic oblivious permutation routing algorithm on a network of N nodes each of out-degree d, there is an instance of permutation routing that requires $\Omega((N/d)^{1/2})$ steps
- Can randomized algorithms do any better?
  - Valiant's Algorithm, 1982

Rand. Permutation Routing in Hypercube

- Phase 1
  - Pick a random intermediate destination for each packet and route it there
- Phase 2
  - Route every packet from intermediate destination to final destination
- Routing in each phase: Bit fixing strategy
  - Given s and t addresses are n-bit vectors, flip leftmost bit of ID in which current node differs from destination ID and send it to that neighbor

O(n) steps

k-Server Problem

- Problem: to efficiently "move" around k servers in a metric space (weighted graph) to service requests that appear online at the points of metric space

General Paradigm: k-Server Problem

- Given:
  - n-vertex metric space (i.e., weighted graph),
  - k servers with initial locations, and
  - (online) request sequence with location
    - Request to be served by server at given location
- Goal: minimize distance travelled by servers
- Variants: symmetric or asymmetric

Mobile

k-Server Problem: Applications

- Paging
  - node ≈ page of address space
  - All distances = 1
- Weighted Caching
  - Fonts in a printer or a bitmap display
- Two-headed Disk Drives

What we know: k-Server Problem

- Lower Bound on competitiveness (k) applies from before
- Conjecture: Upper bound for competitiveness is k [MMS, 1990]

Greedy Algorithm

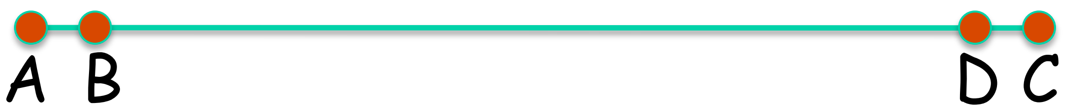
- Let the nearest server serve the request
  - It minimizes the cost of each individual request
  - How competitive is this algorithm?

Balance Algorithm

- Choose a server that would have moved the minimum total distance of any server
  - Takes care of previous bad example since eventually the second server would be employed
  - Tends to use all servers equally
  - Can be shown to be k-competitive if k = n-1
  - Can do poorly in other situations
  - Not 2-competitive for k = 2

Follow-OPT

- On i^{th} request compute final configuration X achieved by OPT
- Use the server that would result in the same configuration X



RES Algorithm for k = 2

- Define Residues
  - $R_c(\sigma,S) = c \cdot C_{OPT}(\sigma,S) - C_A(\sigma)$
- v_1 = location of last request
- v_2 = location of other server
- Figures out which server would result in smaller residues.
- RES is 2-competitive

HARMONIC Algorithm

- Natural, memoryless, randomized algorithm
  - Choose a server with probability inversely proportional to its distance to request location
- Expected to be α-competitive
  - $\alpha = 3^{17000}$ for $k = 3$
  - $\alpha = O(k2^k)$ for general k

Related Problems and Results

- Points on a Line
- Points on a circle
- Points on a tree

- (2n-1)-competitive algorithms exist

Work Function (WF) Algorithm

- Compute the configuration X_i achieved by OPT and closest to previous configuration X_{i-1}
  - Very expensive computationally
- WF is (2k-1)-competitive
- WF is 2-competitive for k = 2

Notation

- Metric Space M with n vertices and distance function $d(\bullet,\bullet)$
- Configuration S = subset of k vertices from M (location of the k servers)
- Requests: $\sigma = \{r_1, r_2, \ldots\}$ from vertices of M
- Solution: Sequence of configurations $S_0, S_1, \ldots$
- Algorithm A: $D_A(S_0,\sigma) = \Sigma_t\, d(S_{t-1}, S_t)$
  - $d(S_a, S_b)$ = min weight matching between S_a & S_b
- Analysis: $D_A(S_0,\sigma) \leq \boxed{\rho} D_{OPT}(S_0,\sigma) + f(S_0)$

Performance Ratio

OPT: Offline Algorithm

- Argue that you only need to consider lazy moves (no unnecessary moves)
- Use dynamic programming
  - Recurrence?
  - Subproblems?

Function of states & request seq

$$C_{OPT}(\epsilon, S) = \begin{cases} 0, & \text{if } S = S_0 \\ \text{undefined}, & \text{otherwise.} \end{cases}$$

$$C_{OPT}(\sigma v, S) = \begin{cases} \min_T C_{OPT}(\sigma, T) + d(T, S), & \text{if } v \text{ is covered in } S \\ \text{undefined}, & \text{otherwise.} \end{cases}$$

Important Open Problems

- Minimize ρ, where
  - $D_A(S_0,\sigma) \leq \rho D_{OPT}(S_0,\sigma) + f(S_0)$
- Competitive ratio of Algorithm/Problem
- k-Server Conjecture: For every metric space, the competitive ratio of the k-server problem is exactly k
- Randomized k-Server Conjecture: For every metric space, there exists a randomized algorithm with competitive ratio O(log k)