Lecture 9: Online Data

Xiaotie Deng

AIMS Lab
Department of Computer Science
Shanghai Jiaotong University

November 15, 2017

1 Page Replacement Algorithm

2 Non-Clairvayant Scheduling

3 Resource Augmentation

Page Replacement Algorithm

Operating System Memory Scheduling

- Two levels of memory
  - Small size fast memory
  - Large size slow memory
- Page fault: data accessed is not in fast memory
  - Upload the page containing the data needed.
- Objective: Design page replacement policy to minimize page fault.

Optimal Page Replacement

- Clairvoyant: Knowing the page sequence to be accessed in advance.
- Online/Non-Clairvoyant: Not knowing the page sequence to be accessed in advance.
- Objective: Design page replacement policy so that the online page fault against offline page fault is minimized (in the worst/average case).
  - Given a page arrival sequence z, $Cost(A, z)$ represents the # of page fault by algorithm A.
  - $OPT(z)$ represents the minimum # of page faults by the best clairvoyant algorithm knowing the sequence z of page arrivals.
- Competitive ratio $\max_{all\ z}\{\frac{Cost(A,z)}{OPT(z)}\}$

OPT(z)

- The furthest in the future algorithm.
- Simplification
  - Prove for the case when the # of pages is $k+1$, with extension by induction to more pages.
- Proof Idea: if an algorithm removes a different page, there is always another algorithm removes the furthest one with less or equal page fault.

Reference: L. A. Belady. A study of replacement algorithms for a virtual storage computer. IBM Systems Journal, 5(2):78101, 1967.

Lower Bound k for Any Deterministic Algorithm

- Consider k pages and $N = k + 1$ pages as potential inputs.
- Fix the online algorithm A. We repeat the following phase to general the input sequence one at a time.
  - As the memory contains k pages. We choose the next input with the page not in the memory.
  - One page will be replaced out of the memory.
- Competitive ratio at least k.

FIF is Always Better

- Let the Cache have $a_1, a_2, \cdots, a_k$ and a new page a_0 comes in.
- Let a_k is removed by FIF, but algorithm A removes a_{k-1} to take in a_0.
- We establish an Algorithm B which
  - Operate the same way as A does if neither a_{k-1} nor a_k is involved.
  - If A removes a_k to take in some page a_i, FIF removes a_{k-1} to take in a_i. Both are in the same state and we apply inductive hypotheiss.
  - If A takes in a_{k-1} to remove a_j, FIF takes in a_k to remove a_j, and apply inductive hypothesis.
-

LRU with competitive ratio k

- On page fault when a new page is to be added, the pages to keep is the most recently used ones.
  - The least recently used one is removed.
- Break input into subsequences $\sigma_1, \sigma_2, \cdots, \sigma_b$ such that
  - σ_i is the longest sequence where k pages appeared in the input.
- LRU faults by $\leq k$ times each block, total bk times in the input sequence.
- Any algorithm faults by at least once at each block transition, total at least $b - 1$ faults.
- Competitive ratio $\leq k$.

LRU with Augmented Resource

- LRU for k cache against FIF for h cache has a competitive ratio of $k/(k-h+1)$.
  - During each phase σ, there are k pages requested but only h slots for FIF.
  - At least $k-h+1$ faults (including the transition point) are incurred during this phase for FIF.
  - As in the above discussion, k faults are incurred in one phase for LRU.
- Corollary: Competitive Ratio is two if we choose $h = k/2$.

Non-Clairvayant Scheduling

System scheduling of unknown processing time jobs

- Inputs of n jobs with unknown processing jobs
- Objective: execute all jobs with the minimum total completion time.
- Model: preemptive scheduling

Difficulty with non-preemptive scheduling

- Worst case:
  - n jobs of length $\epsilon_1 < \epsilon_2 \cdots \epsilon_{n-1} < 1$.
  - Worst total completion time: $n * 1 + (n-1) * \epsilon_{n-1} + \cdots + \epsilon_1$.
  - Best total completion time: $1 * 1 + 2 * \epsilon_1 + \cdots + n * \epsilon_{n-1}$.
- What is the online execution time?
  - Halting Lemma: Finding whether a program halt or not is uncomputable.
  - Corollary: Finding execution time of a program is uncomputable.

Competitive ratio

- Given input $I \in \mathcal{I}$,
  - Define $OPT(I) = min_A Cost(A, I)$ as the cost of the best algorithm for this input I.
- Competitive ratio of an algorithm A is defined as
  - $CR(A, \mathcal{I}) = \max_{I \in \mathcal{I}} \frac{Cost(A,I)}{OPT(I)}$
- Competitive ratio of a problem is defined as
  - $\min_A CR(A, \mathcal{I})$.
- Example: system scheduling with non-preemptive scheduling has competitive ratio n for total completion time as $n \to 0$.

Can we do better?

- Preemptive scheduling: Jobs can be taken off the processor and resume later without penalty
  - Example: Operating system scheduling, round-robin.
- Round robin completes same length jobs simultaneously.
  - For jobs $x_1 < x_2 < \cdots < x_n$, the total completion time is $RR(x_1, \cdot, x_n) = nx_1 + \cdot + [(n-i)x_i + x_i + \cdot + x_1] + \cdot + [x_n + \cdot + x_1]$.
- Competitive ratio:
  - The best total completion time (shortest job first) is $OPT(x_1, x_2, \cdots, x_n) = nx_1 + (n-1)x_2) + \cdots + x_n$.
- $RR(\vec{x}) \leq 2OPT(\vec{x})$.
  - Extra time x_i waited in RR is the same as the actual time x_j has to wait for x_i in OPT.

Resource Augmentation

Why do We Buy More Resources?

- In the Paging problem, we saw that more resources can improve the performance.
- Therefore, more computational resource would be helpful.
- In scheduling, $\sum_j (C_j - r_j)$ measures total time jobs stayed in the system since arrival: r_j arrival time, C_j completion time.
- With full information, shortest remaining processing time *SRPT* first is optimum.
- With unknown information about arrival and unknown execution time, no competitive algorithm is possible.
- If all arrive at time zero, Round Robin is 2-competitive (Motwani, et al., 1994).
- Balance Algorithm (Kalyanasundaram and Pruhs), with an extra of $(1+\epsilon)$ factor of higher speed, is $(1+\epsilon)$-approximation of offline optimum algorithm.

Augmented Speed

- Let $\mathcal{J} = \{J_1, J_2, \cdots, J_n\}$ be the collection of jobs, $J_i = (r_i, x_i)$ with r_i unknown arrival time and x_i unknown execution time, which are revealed when jobs arrive and executions complete.
- (s, α)-competitive refers to an online algorithm which is an s-factor faster than and achieves an approximation algorithm with α-factor of those for the optimal full-information algorithm.
- At an s-speed processor, Job of length x_i takes time x_i/s to finish.
- On an algorithm S, the flow time of job J_j is denoted by $c_j(S)$, the flow time $F_j(S) = c_j(S) - r_j$.
- We are interested in minimizing the total (sum of) flow time(s).

Augmented Resource Approximation

- We denote the Balance algorithm as B and the shortest remaining processing time algorithm as A, which is offline optimum.
  - Let $w(i,t)$ the amount of time spent on J_i by B before time t.
  - Process arrived jobs to equalize $w(j,t)$.
- Theorem: $(1+\epsilon)$-speed B is an $(1+\epsilon)$-approximation of the unit-speed optimum offline algorithm A, c_j for $c_j(B)$.
  - Proof: Follows from Key Lemma.
- Active jobs: U(t,s) (jobs not completed at time t at speed s).
- Key Lemma: $|U_B(t,1+\epsilon)| \leq (1+\epsilon)|U_A(t,1)|$, written as U_B and U_A for short.
  - Proof: Total flow time for algorithm S is $\int_0^\infty |U_S(t,s)|dt$
- Goal: $\epsilon|U_B - U_A| \leq |U_A|$.

Notations and Properties

- Focus on jobs $\mathcal{J}$ arrived before time t.
- Later arrivals are irrelevant to our algorithms A and B by now.
- Property: $w(i,u) \leq w(j,u)$ and $u \geq r_j \implies$ $\forall t \geq u : w(i,t) \leq w(j,t)$
  - Jobs are executed with smallest $w(i,t)$ first.
  - Jobs of the same smallest $w(i,t)$ executed by RR.
  - Execution order of any job subset fixed on the last job arrival.
- Therefore, we write $w(i)$ and $w(i) \leq w(j)$ for Algorithm B as there is no ambiguity.
- Let X and Y be active jobs at time t by Algorithms B and A, respectively.

Key Ideas

- Exchange argument
  - Place time should be spent on X to be charged to those for Y.
- Exchange Trees: rename $X \backslash Y$ as $\{X - Y = \{1, 2, \cdots, p\}$ such that $t \geq r_1 \geq r_2 \geq \cdots \geq r_p \geq 0$. (And Let $Y = \{p + 1, p + 2, \cdots, q\}$.
  - Build an exchange tree T_i for each $i \in X \backslash Y$.
  - Place i at level 0 of T_i.
  - There is an edge from a job j_{l-1} at level $l - 1$ to another job j_l if (a) j_l does not appear at any other level above and (b) j_l is executed in BALANCE instead j_{l-1} at some time.
- NOTE: j_l could arrive before i does.
- Edge $a \rightarrow b$ in T_i iff a and b overlap in BALANCE.

Proof Structure I

1. Set $W(\mathcal{K}, I)$ the total work done by B on the set $\mathcal{K}$ of jobs during time interval I.
2. Lemma $\forall i \in X \backslash Y \ W(T_i \cap U_A, [0, t]) \geq \epsilon \sum_{j=1}^{i} w(j)$
   - As i is not in Y, A must finish it before time t, i.e., spends at least $\sum_{j=1}^{i} w(j)$ work: $t - r_i \geq \sum_{j=1}^{i} w(j)$.
   - Since B was non-idle during $[s_i, t]$, any job run by B during this period must be in T_i.

$$W(T_i, [s_i, t]) = (1+\epsilon)(t-s_i) \geq (t-s_i)+\epsilon(t-r_i) \geq (t-s_i)+\epsilon \sum_{j=1}^{i} w(j)$$

- A completes all jobs in $Y^c \cap T_i$ before time t. Therefore, A spent at most $(t - s_i)$ work on them.
- The claim follows: $W(T_i \cap U_A, [0, t]) \geq \epsilon \sum_{j=1}^{i} w(j)$

Proof Structure II

1. $W(T_i \cap U_A, [0, t])$ can be partitioned in to p parts, P_i, $i = 1, 2, \cdots p$, which contains $\epsilon w(i)$ units of work done on jobs in $T_i \cap U_A$.
2. Proof: Apply the previous lemma.

Proof Structure III

1. $\epsilon|U_B - U_A| \leq |U_Q|$.
2. Proof: For $1 \leq i \leq p$, $j \in T_i \cap U_A$, set $g(i,j)$ the fraction of work in P_i that B did on j. If $i > p$, or $j \notin T_i \cap U_A$, then $g(i,j) = 0$.
   - $\sum_{j=p+1}^{q} g(i,j) = \epsilon w(i)$
   - $\epsilon|U_B - U_A| = \epsilon p = \sum_{i=1}^{p} \frac{1}{w(i)} \epsilon w(i) = \sum_{i=1}^{p} \sum_{j=p+1}^{q} \frac{1}{w(i)} g(i,j)$
3. Claim follows since $w(i) \geq w(j)$ and $\sum_{i=1}^{p} g(i,j) \leq w(j)$.

Assignment III

1. Implement Morris Counting. Show a run of 20 input elements, and calculate the average at each position for the first 20 positions by 1000 runs. Choosing an appropriate distribution for the input data and justify your choice.
2. Consider the paging problem. Find the best competitive ratio when the algorithm has the extra power of one letter look-ahead.
3. Write a program to empirically test the Balanced Algorithm with the SRPT algorithm on input data following uniform distribution and following the negative exponential distribution.