Lecture 20: Online Algorithms & k-server

Rafael Oliveira

University of Waterloo
Cheriton School of Computer Science

rafael.oliveira.teaching@gmail.com

November 23, 2020

Overview

- Administrivia
- Online Algorithms: Randomized Lower Bounds
- k-server on a line
- Conclusion
- Acknowledgements

Rate this course!

Please log in to

https://evaluate.uwaterloo.ca/

from *November 24th until December 7th* and provide us with your evaluation and feedback on the course!

- This would really help me figuring out what worked and what didn't for the course
- And whether I should put memes or gifs into my slides...
- Teaching this course is also a learning experience for me :)

Competitive Analysis

- Input is given as a sequence $s = s_1, s_2, \ldots, s_n$ of events.

Competitive Analysis

- Input is given as a sequence $s = s_1, s_2, \ldots, s_n$ of events.
- Let $C_{opt}(s)$ be the *minimum cost* that *any algorithm* (even one that could look at the *entire input* beforehand) could achieve for input s

Competitive Analysis

- Input is given as a sequence $s = s_1, s_2, \ldots, s_n$ of events.
- Let $C_{opt}(s)$ be the *minimum cost* that *any algorithm* (even one that could look at the *entire input* beforehand) could achieve for input s
- Let $C_A(s)$ be the cost of your online algorithm on input s

Competitive Analysis

- Input is given as a sequence $s = s_1, s_2, \ldots, s_n$ of events.
- Let $C_{opt}(s)$ be the *minimum cost* that *any algorithm* (even one that could look at the *entire input* beforehand) could achieve for input s
- Let $C_A(s)$ be the cost of your online algorithm on input s

Definition (Deterministic Competitive Ratio)

A deterministic online algorithm A has *competitive ratio* k (aka k-competitive) if for all inputs s, we have:

$$C_A(s) \leq k \cdot C_{opt}(s) + O(1)$$

Competitive Analysis

- Input is given as a sequence $s = s_1, s_2, \ldots, s_n$ of events.
- Let $C_{opt}(s)$ be the *minimum cost* that *any algorithm* (even one that could look at the *entire input* beforehand) could achieve for input s
- Let $C_A(s)$ be the cost of your online algorithm on input s

Definition (Deterministic Competitive Ratio)

A deterministic online algorithm A has *competitive ratio* k (aka k-competitive) if for all inputs s, we have:

$$C_A(s) \leq k \cdot C_{opt}(s) + O(1)$$

Definition (Randomized Competitive Ratio)

A randomized online algorithm A has *competitive ratio* k (aka k-competitive) if for all inputs s, we have:

→ expectation over random bits used by A

$$\mathbb{E}[C_A(s)] \leq k \cdot C_{opt}(s).$$

Online Paging Problem

- Computer memory is hierarchical: cache $\rightarrow$ L1 $\rightarrow$ L2 $\rightarrow$ main memory

Online Paging Problem

- Computer memory is hierarchical: cache $\to$ L1 $\to$ L2 $\to$ main memory
- Memory can be modelled in the following way:
  - Each layer of memory is an array with certain number of pages (hence the name)

Online Paging Problem

- Computer memory is hierarchical: cache $\rightarrow$ L1 $\rightarrow$ L2 $\rightarrow$ main memory
- Memory can be modelled in the following way:
  - Each layer of memory is an array with certain number of pages (hence the name)
  - Page stores the content of the item and its location in main memory

Online Paging Problem

- Computer memory is hierarchical: cache $\rightarrow$ L1 $\rightarrow$ L2 $\rightarrow$ main memory
- Memory can be modelled in the following way:
  - Each layer of memory is an array with certain number of pages (hence the name)
  - Page stores the content of the item and its location in main memory
  - When we get a request, we first look up in cache, then L1, then L2, then main memory

Online Paging Problem

- Computer memory is hierarchical: cache $\rightarrow$ L1 $\rightarrow$ L2 $\rightarrow$ main memory
- Memory can be modelled in the following way:
  - Each layer of memory is an array with certain number of pages (hence the name)
  - Page stores the content of the item and its location in main memory
  - When we get a request, we first look up in cache, then L1, then L2, then main memory
  - If request is in cache, we have a *hit* $\leftrightarrow$ request takes negligible time

Online Paging Problem

- Computer memory is hierarchical: cache $\rightarrow$ L1 $\rightarrow$ L2 $\rightarrow$ main memory
- Memory can be modelled in the following way:
  - Each layer of memory is an array with certain number of pages (hence the name)
  - Page stores the content of the item and its location in main memory
  - When we get a request, we first look up in cache, then L1, then L2, then main memory
  - If request is in cache, we have a *hit* $\leftrightarrow$ request takes negligible time
  - Otherwise we have *miss* $\leftrightarrow$ need to fetch data from slower memory
  - *Have to* also copy new data & location to cache

Online Paging Problem

- Computer memory is hierarchical: cache $\rightarrow$ L1 $\rightarrow$ L2 $\rightarrow$ main memory
- Memory can be modelled in the following way:
  - Each layer of memory is an array with certain number of pages (hence the name)
  - Page stores the content of the item and its location in main memory
  - When we get a request, we first look up in cache, then L1, then L2, then main memory
  - If request is in cache, we have a *hit* $\leftrightarrow$ request takes negligible time
  - Otherwise we have *miss* $\leftrightarrow$ need to fetch data from slower memory
  - *Have to* also copy new data & location to cache
  - If cache full, *must delete* an old entry before copying new data

Online Paging Problem

- Computer memory is hierarchical: cache $\rightarrow$ L1 $\rightarrow$ L2 $\rightarrow$ main memory
- Memory can be modelled in the following way:
  - Each layer of memory is an array with certain number of pages (hence the name)
  - Page stores the content of the item and its location in main memory
  - When we get a request, we first look up in cache, then L1, then L2, then main memory
  - If request is in cache, we have a *hit* $\leftrightarrow$ request takes negligible time
  - Otherwise we have *miss* $\leftrightarrow$ need to fetch data from slower memory
  - *Have to* also copy new data & location to cache
  - If cache full, *must delete* an old entry before copying new data
- Main question: which entry of the cache to delete?

evict

Online Paging Problem

- Computer memory is hierarchical: cache $\rightarrow$ L1 $\rightarrow$ L2 $\rightarrow$ main memory
- Memory can be modelled in the following way:
  - Each layer of memory is an array with certain number of pages (hence the name)
  - Page stores the content of the item and its location in main memory
  - When we get a request, we first look up in cache, then L1, then L2, then main memory
  - If request is in cache, we have a *hit* $\leftrightarrow$ request takes negligible time
  - Otherwise we have *miss* $\leftrightarrow$ need to fetch data from slower memory
  - *Have to* also copy new data & location to cache
  - If cache full, *must delete* an old entry before copying new data
- Main question: which entry of the cache to delete?
- Cost function: *number of cache misses*

Online Paging Problem

- Computer memory is hierarchical: cache $\rightarrow$ L1 $\rightarrow$ L2 $\rightarrow$ main memory
- Memory can be modelled in the following way:
  - Each layer of memory is an array with certain number of pages (hence the name)
  - Page stores the content of the item and its location in main memory
  - When we get a request, we first look up in cache, then L1, then L2, then main memory
  - If request is in cache, we have a *hit* $\leftrightarrow$ request takes negligible time
  - Otherwise we have *miss* $\leftrightarrow$ need to fetch data from slower memory
  - *Have to* also copy new data & location to cache
  - If cache full, *must delete* an old entry before copying new data
- Main question: which entry of the cache to delete?
- Cost function: *number of cache misses*
- Simplification: assume we only have cache and main memory.

Lower Bound - Deterministic Paging Algorithms

Theorem

Any deterministic algorithm for paging with k pages is at least k-competitive!

- Proof by trolling.[1] Let's use $k + 1$ pages, and let A be our paging algorithm.

[1] Common lower bound technique for online algorithms, also commonly used online as well :)

Lower Bound - Deterministic Paging Algorithms

Theorem

Any deterministic algorithm for paging with k pages is at least k-competitive!

- Proof by trolling.[1] Let's use $k + 1$ pages, and let A be our paging algorithm. ↪ adaptive adversary
- **Input sequence:** at each step, request page that A *doesn't have*.

[1] Common lower bound technique for online algorithms, also commonly used online as well :)

Lower Bound - Deterministic Paging Algorithms

Theorem

Any deterministic algorithm for paging with k pages is at least k-competitive!

- Proof by trolling.[1] Let's use $k+1$ pages, and let A be our paging algorithm.
- **Input sequence:** at each step, request page that A *doesn't have*.
- A faults every single time.

[1] Common lower bound technique for online algorithms, also commonly used online as well :)

Lower Bound - Deterministic Paging Algorithms

Theorem

Any deterministic algorithm for paging with k pages is at least k-competitive!

- Proof by trolling.[1] Let's use $k + 1$ pages, and let A be our paging algorithm.
- **Input sequence:** at each step, request page that A *doesn't have*.
- A faults every single time.
- **Offline Algorithm:** on cache miss, delete page which is requested *furthest in the future*.

[1] Common lower bound technique for online algorithms, also commonly used online as well :)

Lower Bound - Deterministic Paging Algorithms

Theorem

Any deterministic algorithm for paging with k pages is at least k-competitive!

- Proof by trolling.[1] Let's use $k+1$ pages, and let A be our paging algorithm.
- **Input sequence:** at each step, request page that A *doesn't have*.
- A faults every single time.
- **Offline Algorithm:** on cache miss, delete page which is requested *furthest in the future*.
- When offline algorithm deletes a page, it's next delete happens after at least k steps.

[1] Common lower bound technique for online algorithms, also commonly used online as well :)

Randomized Online Algorithms & Game Theory

- Think of online algorithms as being a zero-sum, two-player game between you (the algorithm) and an adversary (the entity choosing the sequence of requests).

Randomized Online Algorithms & Game Theory

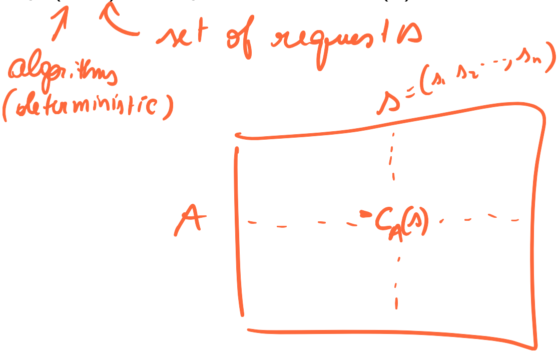
- Think of online algorithms as being a zero-sum, two-player game between you (the algorithm) and an adversary (the entity choosing the sequence of requests).
- Each of your strategies is a different *deterministic algorithm*

Randomized Online Algorithms & Game Theory

- Think of online algorithms as being a zero-sum, two-player game between you (the algorithm) and an adversary (the entity choosing the sequence of requests).
- Each of your strategies is a different *deterministic algorithm*
- Each of adversary's strategies is a sequence of requests

Randomized Online Algorithms & Game Theory

- Think of online algorithms as being a zero-sum, two-player game between you (the algorithm) and an adversary (the entity choosing the sequence of requests).
- Each of your strategies is a different *deterministic algorithm*
- Each of adversary's strategies is a sequence of requests
- Entry (A, s) of payoff matrix: $C_A(s)$

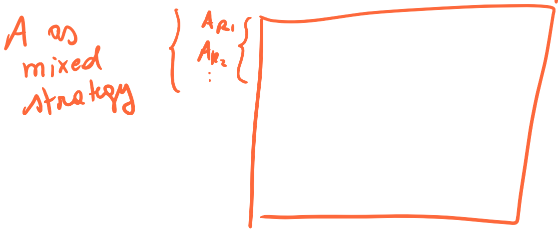


Randomized Online Algorithms & Game Theory

- Think of online algorithms as being a zero-sum, two-player game between you (the algorithm) and an adversary (the entity choosing the sequence of requests).
- Each of your strategies is a different *deterministic algorithm*
- Each of adversary's strategies is a sequence of requests
- Entry (A, s) of payoff matrix: $C_A(s)$
- Algorithm wants to minimize cost
- Adversary wants to maximize it

Randomized Online Algorithms & Game Theory

- Think of online algorithms as being a zero-sum, two-player game between you (the algorithm) and an adversary (the entity choosing the sequence of requests).
- Each of your strategies is a different *deterministic algorithm*
- Each of adversary's strategies is a sequence of requests
- Entry (A, s) of payoff matrix: $C_A(s)$
- Algorithm wants to minimize cost
- Adversary wants to maximize it
    - Randomized algorithm $\Leftrightarrow$ mixed strategies!



Randomized Online Algorithms & Game Theory

- Think of online algorithms as being a zero-sum, two-player game between you (the algorithm) and an adversary (the entity choosing the sequence of requests).
- Each of your strategies is a different *deterministic algorithm*
- Each of adversary's strategies is a sequence of requests
- Entry (A, s) of payoff matrix: $C_A(s)$
- Algorithm wants to minimize cost
- Adversary wants to maximize it
  - Randomized algorithm $\Leftrightarrow$ mixed strategies!
- As we showed in lecture 12, if one player is using *mixed strategy*, the other player has as best response a *pure strategy*

Randomized Online Algorithms & Game Theory

- Think of online algorithms as being a zero-sum, two-player game between you (the algorithm) and an adversary (the entity choosing the sequence of requests).
- Each of your strategies is a different *deterministic algorithm*
- Each of adversary's strategies is a sequence of requests
- Entry (A, s) of payoff matrix: $C_A(s)$
- Algorithm wants to minimize cost
- Adversary wants to maximize it
    - Randomized algorithm $\Leftrightarrow$ mixed strategies!
- As we showed in lecture 12, if one player is using *mixed strategy*, the other player has as best response a *pure strategy*

Theorem (Yao's minimax principle)

If for some input distribution, no deterministic algorithm is k-competitive, then no randomized algorithm is k-competitive!

Lower Bound - Randomized Paging Algorithms

1. Setting: $k+1$ distinct pages, cache of size k, n requests

[2] Here expectation is over the choice of input.

Lower Bound - Randomized Paging Algorithms

1. Setting: $k+1$ distinct pages, cache of size k, n requests
2. Distribution of inputs: *uniform distribution*

$s = (s_1, s_2, \ldots, s_n)$

$s_i \in \{1, 2, \ldots, k, k+1\}$

[2] Here expectation is over the choice of input.

Lower Bound - Randomized Paging Algorithms

1. Setting: $k+1$ distinct pages, cache of size k, n requests
2. Distribution of inputs: *uniform distribution*
3. Equivalently: each page has probability $\frac{1}{k+1}$ of being chosen

[2] Here expectation is over the choice of input.

Lower Bound - Randomized Paging Algorithms

1. Setting: $k+1$ distinct pages, cache of size k, n requests
2. Distribution of inputs: *uniform distribution*
3. Equivalently: each page has probability $\frac{1}{k+1}$ of being chosen
4. Online Algorithm
   - No matter what our (*fixed*) deterministic algorithm A does, only k pages in cache, with probability $\frac{1}{k+1}$ requested page *not in memory*

[2] Here expectation is over the choice of input.

Lower Bound - Randomized Paging Algorithms

1. Setting: $k+1$ distinct pages, cache of size k, n requests
2. Distribution of inputs: *uniform distribution*
3. Equivalently: each page has probability $\frac{1}{k+1}$ of being chosen
4. Online Algorithm
   - No matter what our (*fixed*) deterministic algorithm A does, only k pages in cache, with probability $\frac{1}{k+1}$ requested page *not in memory*
   - Expected number of requests per fault: $k+1$ (which is $O(k)$)

[2] Here expectation is over the choice of input.

Lower Bound - Randomized Paging Algorithms

1. Setting: $k+1$ distinct pages, cache of size k, n requests
2. Distribution of inputs: *uniform distribution*
3. Equivalently: each page has probability $\frac{1}{k+1}$ of being chosen
4. Online Algorithm
   - No matter what our (*fixed*) deterministic algorithm A does, only k pages in cache, with probability $\frac{1}{k+1}$ requested page *not in memory*
   - Expected number of requests per fault: $k+1$ (which is $O(k)$)
5. Offline Algorithm (OPT)
   - OPT can see the whole input beforehand (still use Farthest in Future)

[2] Here expectation is over the choice of input.

Lower Bound - Randomized Paging Algorithms

1. Setting: $k+1$ distinct pages, cache of size k, n requests
2. Distribution of inputs: *uniform distribution*
3. Equivalently: each page has probability $\frac{1}{k+1}$ of being chosen
4. Online Algorithm
   - No matter what our (*fixed*) deterministic algorithm A does, only k pages in cache, with probability $\frac{1}{k+1}$ requested page *not in memory*
   - Expected number of requests per fault: $k+1$ (which is $O(k)$)
5. Offline Algorithm (OPT)
   - OPT can see the whole input beforehand (still use Farthest in Future)
   - Farthest in Future faults only after $k+1$ distinct pages seen

[2] Here expectation is over the choice of input.

Lower Bound - Randomized Paging Algorithms

1. Setting: $k+1$ distinct pages, cache of size k, n requests
2. Distribution of inputs: *uniform distribution*
3. Equivalently: each page has probability $\frac{1}{k+1}$ of being chosen
4. Online Algorithm
   - No matter what our (*fixed*) deterministic algorithm A does, only k pages in cache, with probability $\frac{1}{k+1}$ requested page *not in memory*
   - Expected number of requests per fault: $k+1$ (which is $O(k)$)
5. Offline Algorithm (OPT)
   - OPT can see the whole input beforehand (still use Farthest in Future)
   - Farthest in Future faults only after $k+1$ distinct pages seen
   - Expected number of requests per fault:[2] $\Theta(k \log k)$ (see reference)

coupon collector

[2] Here expectation is over the choice of input.

Lower Bound - Randomized Paging Algorithms

$$\frac{n}{k} \le \log k \cdot \frac{n}{k \log n}$$

1. Setting: $k+1$ distinct pages, cache of size k, n requests
2. Distribution of inputs: *uniform distribution*
3. Equivalently: each page has probability $\frac{1}{k+1}$ of being chosen
4. Online Algorithm

$$\frac{n}{k+1}$$

   - No matter what our (*fixed*) deterministic algorithm A does, only k pages in cache, with probability $\frac{1}{k+1}$ requested page *not in memory*
   - Expected number of requests per fault: $k+1$ (which is $O(k)$)
5. Offline Algorithm (OPT)

$$\frac{n}{k \log n}$$

   - OPT can see the whole input beforehand (still use Farthest in Future)
   - Farthest in Future faults only after $k+1$ distinct pages seen
   - Expected number of requests per fault:[2] $\Theta(k \log k)$ (see reference)

Theorem

Any randomized algorithm for paging with k pages is $\Omega(\log k)$-competitive!

---

[2] Here expectation is over the choice of input.

k-server Problem

- **Setup:** we are given a metric space (X, d).

k-server Problem

- **Setup:** we are given a metric space (X, d).
- Online algorithm manages k mobile servers, each server is located at a point in X

k-server Problem

- **Setup:** we are given a metric space (X, d).
- Online algorithm manages k mobile servers, each server is located at a point in X
- A request specifies a point in X, to which a server *must be moved*, unless we already have a server there.

k-server Problem

- **Setup:** we are given a metric space (X, d).
- Online algorithm manages k mobile servers, each server is located at a point in X
- A request specifies a point in X, to which a server *must be moved*, unless we already have a server there.
- Main question: which server to move?

k-server Problem

- **Setup:** we are given a metric space (X, d).
- Online algorithm manages k mobile servers, each server is located at a point in X
- A request specifies a point in X, to which a server *must be moved*, unless we already have a server there.
- Main question: which server to move?
- Cost function: *total distance travelled*

k-server Problem

- **Setup:** we are given a metric space (X, d).
- Online algorithm manages k mobile servers, each server is located at a point in X
- A request specifies a point in X, to which a server *must be moved*, unless we already have a server there.
- Main question: which server to move?
- Cost function: *total distance travelled*
- Goal: minimize distance travelled

k-server Problem

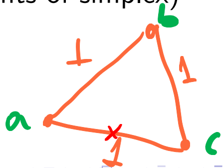
Paging problem   cache size 2

3 different pages

- **Setup:** we are given a metric space (X, d).
- Online algorithm manages k mobile servers, each server is located at a point in X
- A request specifies a point in X, to which a server *must be moved*, unless we already have a server there.
- Main question: which server to move?
- Cost function: *total distance travelled*
- Goal: minimize distance travelled
- Paging is special case of this problem (points of simplex)

2-server problem (size of cache)

3 equidistant requests



k-server Problem

- **Setup:** we are given a metric space (X, d).
- Online algorithm manages k mobile servers, each server is located at a point in X
- A request specifies a point in X, to which a server *must be moved*, unless we already have a server there.
- Main question: which server to move?
- Cost function: *total distance travelled*
- Goal: minimize distance travelled
- Paging is special case of this problem (points of simplex)
- Today's Simplification: assume X is a *line*. Think $X = \mathbb{R}$

Attempt 1: Greedy

1. Strategy: just move the server which is closest to the request to it

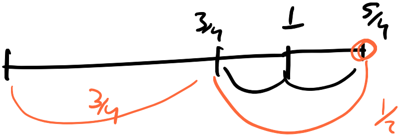
Attempt 1: Greedy

1. Strategy: just move the server which is closest to the request to it
2. Not competitive.

Attempt 1: Greedy

1. Strategy: just move the server which is closest to the request to it
2. Not competitive.
3. Scenario: two servers A and B, initially located at 0 and 1 respectively

Attempt 1: Greedy



1. Strategy: just move the server which is closest to the request to it
2. Not competitive.
3. Scenario: two servers A and B, initially located at 0 and 1 respectively
4. **Requests**: sequence given by $s_{2k-1} = 3/4$, $s_{2k} = 5/4$, for $k \geq 1$

$s_1 = \frac{3}{4} \quad s_2 = \frac{5}{4} \quad s_3 = \frac{3}{4} \quad s_4 = \frac{5}{4} \quad \cdots$

Attempt 1: Greedy

1. Strategy: just move the server which is closest to the request to it
2. Not competitive.
3. Scenario: two servers A and B, initially located at 0 and 1 respectively
4. **Requests**: sequence given by $s_{2k-1} = 3/4$, $s_{2k} = 5/4$, for $k \geq 1$
5. Only server B will move

Attempt 1: Greedy

1. Strategy: just move the server which is closest to the request to it
2. Not competitive.
3. Scenario: two servers A and B, initially located at 0 and 1 respectively
4. **Requests**: sequence given by $s_{2k-1} = 3/4$, $s_{2k} = 5/4$, for $k \geq 1$
5. Only server B will move
6. Best strategy: put A on $3/4$, B on $5/4$

$\frac{3}{4}$ $\frac{5}{4}$

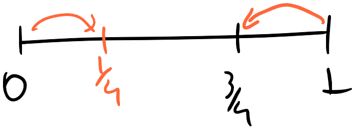
$\text{Cost}_{\text{OPT}}(s) = 1$

$\text{Cost}_{\text{Greedy}}(s) = \Omega(n)$

Attempt 2: Double Coverage (DC)

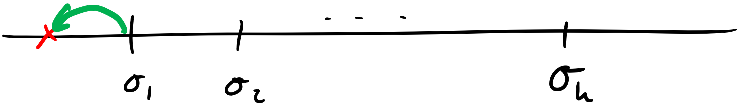
- If request falls between two servers, move both towards request at same rate until one reaches it

(simplification : never query exactly at half)



Attempt 2: Double Coverage (DC)

- If request falls between two servers, move both towards request at same rate until one reaches it
- Else, just move the closest server to the request.



Attempt 2: Double Coverage (DC)

- If request falls between two servers, move both towards request at same rate until one reaches it
- Else, just move the closest server to the request.

Theorem

For k servers, Double Coverage is k-competitive.

Attempt 2: Double Coverage (DC)

- If request falls between two servers, move both towards request at same rate until one reaches it
- Else, just move the closest server to the request.

Theorem

For k servers, Double Coverage is k-competitive.

1. How to model OPT (offline algorithm)?

Attempt 2: Double Coverage (DC)

- If request falls between two servers, move both towards request at same rate until one reaches it
- Else, just move the closest server to the request.

Theorem

For k servers, Double Coverage is k-competitive.

1. How to model OPT (offline algorithm)?
2. Will assume that OPT algorithm moves *exactly one server at a time*.

Attempt 2: Double Coverage (DC)

- If request falls between two servers, move both towards request at same rate until one reaches it
- Else, just move the closest server to the request.

Theorem

For k servers, Double Coverage is k-competitive.

1. How to model OPT (offline algorithm)?
2. Will assume that OPT algorithm moves *exactly one server at a time*.
3. This is w.l.o.g., because can convert *any offline strategy* into a strategy that moves one server per request, by deferring moves to the future

Practice problem: prove this!

Attempt 2: Double Coverage (DC)

- If request falls between two servers, move both towards request at same rate until one reaches it
- Else, just move the closest server to the request.

Theorem

For k servers, Double Coverage is k-competitive.

1. How to model OPT (offline algorithm)?
2. Will assume that OPT algorithm moves *exactly one server at a time*.
3. This is w.l.o.g., because can convert *any offline strategy* into a strategy that moves one server per request, by deferring moves to the future
4. How to analyze competitiveness?

Attempt 2: Double Coverage (DC)

- If request falls between two servers, move both towards request at same rate until one reaches it
- Else, just move the closest server to the request.

Theorem

For k servers, Double Coverage is k-competitive.

1. How to model OPT (offline algorithm)?
2. Will assume that OPT algorithm moves *exactly one server at a time*.
3. This is w.l.o.g., because can convert *any offline strategy* into a strategy that moves one server per request, by deferring moves to the future
4. How to analyze competitiveness?
5. Potential Function:
   - match each server from DC to a server of OPT
   - track changes as requests come

Potential Method - Recap

- In potential method, we have a potential function Φ_t for each time t

Potential Method - Recap

- In potential method, we have a potential function Φ_t for each time t
- Real cost of operation: c_t

Potential Method - Recap

- In potential method, we have a potential function Φ_t for each time t
- Real cost of operation: c_t
- Ammortized cost at time t:

$$\gamma_t = c_t + \Phi_t - \Phi_{t-1}$$

Potential Method - Recap

- In potential method, we have a potential function Φ_t for each time t
- Real cost of operation: c_t
- Ammortized cost at time t:

$$\gamma_t = c_t + \Phi_t - \Phi_{t-1}$$

- Total ammortized cost:

$$\sum_{t=1}^{n} \gamma_t = \sum_{t=1}^{n} c_t + \Phi_t - \Phi_{t-1}$$

$$= \Phi_n - \Phi_0 + \sum_{t=1}^{n} c_t$$

final initial actual total cost

Potential Method - Recap

- In potential method, we have a potential function Φ_t for each time t
- Real cost of operation: c_t
- Ammortized cost at time t:

$$\gamma_t = c_t + \Phi_t - \Phi_{t-1}$$

- Total ammortized cost:

$$\sum_{t=1}^{n} \gamma_t = \sum_{t=1}^{n} c_t + \Phi_t - \Phi_{t-1}$$

$$= \Phi_n - \Phi_0 + \sum_{t=1}^{n} c_t$$

- If potential function is always *non-negative*

$$\sum_{t=1}^{n} c_t \leq \Phi_0 + \sum_{t=1}^{n} \gamma_t$$

DC Analysis - Potential Function

Main idea: have the *ammortized cost* per request be (a multiple of) the cost of OPT, while the actual cost is the cost of DC.

DC Analysis - Potential Function

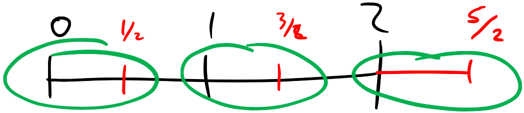
Main idea: have the *ammortized cost* per request be (a multiple of) the cost of OPT, while the actual cost is the cost of DC.

- Consider the state of DC and of OPT at time t

DC Analysis - Potential Function

Main idea: have the *ammortized cost* per request be (a multiple of) the cost of OPT, while the actual cost is the cost of DC.

- Consider the state of DC and of OPT at time t
- Let M_t be cost of minimum cost matching between DC's servers and OPT servers
- Let S_t be sum of pairwise distances of DC's servers



$S_0 = 1 + 2 + 1 = 4$

$M_0 = 3/2$

DC Analysis - Potential Function

Main idea: have the *ammortized cost* per request be (a multiple of) the cost of OPT, while the actual cost is the cost of DC.

- Consider the state of DC and of OPT at time t
- Let M_t be cost of minimum cost matching between DC's servers and OPT servers
- Let S_t be sum of pairwise distances of DC's servers
- Our potential function will be

$$\Phi_t = k \cdot M_t + S_t$$

DC Analysis - Potential Function

Main idea: have the *ammortized cost* per request be (a multiple of) the cost of OPT, while the actual cost is the cost of DC.

- Consider the state of DC and of OPT at time t
- Let M_t be cost of minimum cost matching between DC's servers and OPT servers
- Let S_t be sum of pairwise distances of DC's servers
- Our potential function will be

$$\Phi_t = k \cdot M_t + S_t$$

- Note that $\Phi_t \geq 0$ at all times

DC Analysis - Potential Function

Main idea: have the *ammortized cost* per request be (a multiple of) the cost of OPT, while the actual cost is the cost of DC.

- Consider the state of DC and of OPT at time t
- Let M_t be cost of minimum cost matching between DC's servers and OPT servers
- Let S_t be sum of pairwise distances of DC's servers
- Our potential function will be

$$\Phi_t = k \cdot M_t + S_t$$

- Note that $\Phi_t \geq 0$ at all times
- Use Amortized Analysis to compute amortized cost of DC

DC Analysis - Potential Function

Main idea: have the *ammortized cost* per request be (a multiple of) the cost of OPT, while the actual cost is the cost of DC.

- Consider the state of DC and of OPT at time t
- Let M_t be cost of minimum cost matching between DC's servers and OPT servers
- Let S_t be sum of pairwise distances of DC's servers
- Our potential function will be

$$\Phi_t = k \cdot M_t + S_t$$

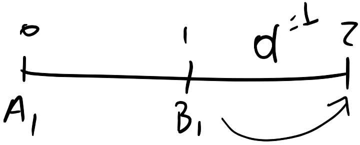
- Note that $\Phi_t \geq 0$ at all times
- Use Amortized Analysis to compute amortized cost of DC
- Break requests into two parts:
  - First account for OPT move
  - Then account for DC move

DC Analysis - Potential Function

1. OPT moves

DC Analysis - Potential Function

1. OPT moves
    - If OPT moves a distance d, the distance from the moved server to the matched DC's server increases by d



$$d(A_1^{(t)}, B_1^{(t)}) = 1$$

$$d(A_1^{(t+1)}, B_1^{(t+1)}) = 2$$

DC Analysis - Potential Function

1. OPT moves
   - If OPT moves a distance d, the distance from the moved server to the matched DC's server increases by d
   - So $M_{t+1} \leq M_t + d$

have matching DC OPT of cost $M_t + d$

$\Rightarrow M_{t+1} \leq M_t + d$

DC Analysis - Potential Function

1. OPT moves
   - If OPT moves a distance d, the distance from the moved server to the matched DC's server increases by d
   - So $M_{t+1} \leq M_t + d$
   - Thus potential increased (so far) by $\Phi_{t+1} - \Phi_t \leq k \cdot d$

k × distance that OPT travelled

$$\Phi_{t+1} - \Phi_t$$

$$k \cdot M_{t+1} + S_{t+1} - k \cdot M_t - S_t$$

$$k(M_{t+1} - M_t) \leq k \cdot d$$

$$M_{t+1} - M_t \leq d$$

DC Analysis - Potential Function

1. OPT moves
   - If OPT moves a distance d, the distance from the moved server to the matched DC's server increases by d
   - So $M_{t+1} \leq M_t + d$
   - Thus potential increased (so far) by $\Phi_{t+1} - \Phi_t \leq k \cdot d$
   - Real cost incurred by DC: $c_{t+1} = 0$

DC Analysis - Potential Function

1. OPT moves
   - If OPT moves a distance d, the distance from the moved server to the matched DC's server increases by d
   - So $M_{t+1} \leq M_t + d$
   - Thus potential increased (so far) by $\Phi_{t+1} - \Phi_t \leq k \cdot d$
   - Real cost incurred by DC: $c_{t+1} = 0$
   - Ammortized cost of DC: $\gamma_{t+1} \leq k \cdot d$

DC Analysis - Potential Function

1. OPT moves
   - If OPT moves a distance d, the distance from the moved server to the matched DC's server increases by d
   - So $M_{t+1} \leq M_t + d$
   - Thus potential increased (so far) by $\Phi_{t+1} - \Phi_t \leq k \cdot d$
   - Real cost incurred by DC: $c_{t+1} = 0$
   - Ammortized cost of DC: $\gamma_{t+1} \leq k \cdot d$
2. DC moves

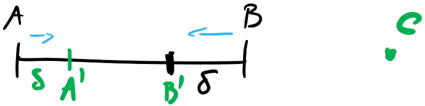
DC Analysis - Potential Function

2. DC moves
    1. The request falls between two servers A and B. Say that B is taken to the location requested.

DC Analysis - Potential Function

2. DC moves
   1. The request falls between two servers A and B. Say that B is taken to the location requested.
      - Both servers move a distance δ.

DC Analysis - Potential Function



$$d(A', C) + d(B', C) = d(A, C) - \delta + d(B, C) + \delta$$
$$= d(A, C) + d(B, C)$$

2. DC moves
   1. The request falls between two servers A and B. Say that B is taken to the location requested.
      - Both servers move a distance δ.
      - Thus pairwise distances decrease by 2δ (because they are in a line)

$$d(A', B') = d(A, B) - 2\delta$$

$$\boxed{S_{t+1} = S_t - 2\delta}$$

DC Analysis - Potential Function

2. DC moves
   1. The request falls between two servers A and B. Say that B is taken to the location requested.
      - Both servers move a distance δ.
      - Thus pairwise distances decrease by 2δ (because they are in a line)
      - Changes in other pairwise distances cancel out (because line)

DC Analysis - Potential Function

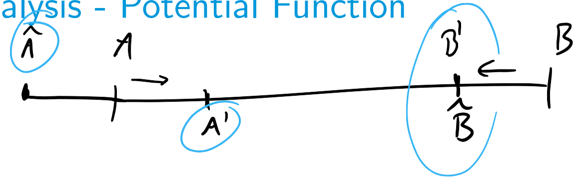
2. DC moves
   1. The request falls between two servers A and B. Say that B is taken to the location requested.
      - Both servers move a distance δ.
      - Thus pairwise distances decrease by 2δ (because they are in a line)
      - Changes in other pairwise distances cancel out (because line)
      - Thus S decreases by 2δ

DC Analysis - Potential Function

2. DC moves
   1. The request falls between two servers A and B. Say that B is taken to the location requested.
      - Both servers move a distance δ.
      - Thus pairwise distances decrease by 2δ (because they are in a line)
      - Changes in other pairwise distances cancel out (because line)
      - Thus S decreases by 2δ
      - B has match at destination (problem constraint)

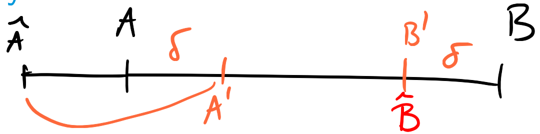
from OPT

DC Analysis - Potential Function



2. DC moves
    1. The request falls between two servers A and B. Say that B is taken to the location requested.
        - Both servers move a distance δ.
        - Thus pairwise distances decrease by 2δ (because they are in a line)
        - Changes in other pairwise distances cancel out (because line)
        - Thus S decreases by 2δ
        - B has match at destination (problem constraint)
        - A may be further from its match, but balanced by B's move

DC Analysis - Potential Function



if $(A, \hat{A})$ matched in M_t — $d(A', \hat{A}) = d(A, \hat{A}) + \delta$

$(B, \hat{B})$ matched in M_t — $d(B', \hat{B}) = d(B, \hat{B}) - \delta$

2. DC moves
    1. The request falls between two servers A and B. Say that B is taken to the location requested.
        - Both servers move a distance δ.
        - Thus pairwise distances decrease by 2δ (because they are in a line)
        - Changes in other pairwise distances cancel out (because line)
        - Thus S decreases by 2δ
        - B has match at destination (problem constraint)
        - A may be further from its match, but balanced by B's move
        - $M_{t+1} \leq M_t$

have another matching of cost M_t

$M_{t+1} \leq M_t$

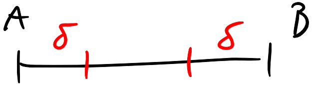
DC Analysis - Potential Function

$$M_{t+1} - M_t \leq 0$$

$$\Phi_{t+1} - \Phi_t = k\underbrace{(M_{t+1} - M_t)}_{\leq 0} + \underbrace{(S_{t+1} - S_t)}_{= -2\delta}$$

2. DC moves
   1. The request falls between two servers A and B. Say that B is taken to the location requested.
      - Both servers move a distance δ.
      - Thus pairwise distances decrease by 2δ (because they are in a line)
      - Changes in other pairwise distances cancel out (because line)
      - Thus S decreases by 2δ
      - B has match at destination (problem constraint)
      - A may be further from its match, but balanced by B's move
      - $M_{t+1} \leq M_t$
      - Potential Change: $\Phi_{t+1} - \Phi_t \leq k \cdot 0 - 2 \cdot \delta = -2 \cdot \delta$

DC Analysis - Potential Function



2. DC moves
   1. The request falls between two servers A and B. Say that B is taken to the location requested.
      - Both servers move a distance δ.
      - Thus pairwise distances decrease by 2δ (because they are in a line)
      - Changes in other pairwise distances cancel out (because line)
      - Thus S decreases by 2δ
      - B has match at destination (problem constraint)
      - A may be further from its match, but balanced by B's move
      - $M_{t+1} \leq M_t$
      - Potential Change: $\Phi_{t+1} - \Phi_t \leq k \cdot 0 - 2 \cdot \delta = -2 \cdot \delta$
      - Real cost incurred by DC: $c_{t+1} = 2\delta$

DC Analysis - Potential Function

2. DC moves
   1. The request falls between two servers A and B. Say that B is taken to the location requested.
      - Both servers move a distance δ.
      - Thus pairwise distances decrease by 2δ (because they are in a line)
      - Changes in other pairwise distances cancel out (because line)
      - Thus S decreases by 2δ
      - B has match at destination (problem constraint)
      - A may be further from its match, but balanced by B's move
      - $M_{t+1} \leq M_t$
      - Potential Change: $\Phi_{t+1} - \Phi_t \leq k \cdot 0 - 2 \cdot \delta = -2 \cdot \delta$
      - Real cost incurred by DC: $c_{t+1} = 2\delta$
      - Ammortized cost of DC: $\gamma_{t+1} \leq 2\delta - 2\delta = 0$

DC Analysis - Potential Function

2. DC moves
   1. The request falls between two servers A and B. Say that B is taken to the location requested.
      - Both servers move a distance δ.
      - Thus pairwise distances decrease by 2δ (because they are in a line)
      - Changes in other pairwise distances cancel out (because line)
      - Thus S decreases by 2δ
      - B has match at destination (problem constraint)
      - A may be further from its match, but balanced by B's move
      - $M_{t+1} \leq M_t$
      - Potential Change: $\Phi_{t+1} - \Phi_t \leq k \cdot 0 - 2 \cdot \delta = -2 \cdot \delta$
      - Real cost incurred by DC: $c_{t+1} = 2\delta$
      - Ammortized cost of DC: $\gamma_{t+1} \leq 2\delta - 2\delta = 0$
   2. Only one server moves (request outside the border)

DC Analysis - Potential Function

1. OPT moves distance d
   - Ammortized cost of DC: $\gamma_t \leq k \cdot d$
2. DC moves
   1. The request falls between two servers.
      - Ammortized cost of DC: $\gamma_t \leq 0$
   2. Only one server, say A, moves (request outside the border)

DC Analysis - Potential Function

1. OPT moves distance d
   - Ammortized cost of DC: $\gamma_t \leq k \cdot d$
2. DC moves
   1. The request falls between two servers.
      - Ammortized cost of DC: $\gamma_t \leq 0$
   2. Only one server, say A, moves (request outside the border)
      - Suppose A moved δ

DC Analysis - Potential Function

1. OPT moves distance d
   - Ammortized cost of DC: $\gamma_t \leq k \cdot d$
2. DC moves
   1. The request falls between two servers.
      - Ammortized cost of DC: $\gamma_t \leq 0$
   2. Only one server, say A, moves (request outside the border)
      - Suppose A moved δ
      - A has its match (from OPT's server) at destination

DC Analysis - Potential Function

1. OPT moves distance d
   - Ammortized cost of DC: $\gamma_t \leq k \cdot d$
2. DC moves
   1. The request falls between two servers.
      - Ammortized cost of DC: $\gamma_t \leq 0$
   2. Only one server, say A, moves (request outside the border)
      - Suppose A moved δ
      - A has its match (from OPT's server) at destination
      - $M_{t+1} \leq M_t - \delta$

DC Analysis - Potential Function

1. OPT moves distance d
   - Ammortized cost of DC: $\gamma_t \leq k \cdot d$
2. DC moves
   1. The request falls between two servers.
      - Ammortized cost of DC: $\gamma_t \leq 0$
   2. Only one server, say A, moves (request outside the border)
      - Suppose A moved δ
      - A has its match (from OPT's server) at destination
      - $M_{t+1} \leq M_t - \delta$
      - Each pairwise distance (A, B) (where B is another of DC's servers) increases by δ

DC Analysis - Potential Function

1. OPT moves distance d
   - Ammortized cost of DC: $\gamma_t \leq k \cdot d$
2. DC moves
   1. The request falls between two servers.
      - Ammortized cost of DC: $\gamma_t \leq 0$
   2. Only one server, say A, moves (request outside the border)
      - Suppose A moved δ
      - A has its match (from OPT's server) at destination
      - $M_{t+1} \leq M_t - \delta$
      - Each pairwise distance (A, B) (where B is another of DC's servers) increases by δ
      - Total distance increased: $S_{t+1} - S_t \leq (k-1) \cdot \delta$

DC Analysis - Potential Function

1. OPT moves distance d
   - Ammortized cost of DC: $\gamma_t \leq k \cdot d$
2. DC moves
   1. The request falls between two servers.
      - Ammortized cost of DC: $\gamma_t \leq 0$
   2. Only one server, say A, moves (request outside the border)
      - Suppose A moved δ
      - A has its match (from OPT's server) at destination
      - $M_{t+1} \leq M_t - \delta$
      - Each pairwise distance (A, B) (where B is another of DC's servers) increases by δ
      - Total distance increased: $S_{t+1} - S_t \leq (k-1) \cdot \delta$
      - Change in potential:

$$\Delta\Phi \leq -k \cdot \delta + (k-1) \cdot \delta = -\delta$$

$k(M_{t+1} - M_t)$ $(S_{t+1} - S_t)$

DC Analysis - Potential Function

1. OPT moves distance d
   - Ammortized cost of DC: $\gamma_t \leq k \cdot d$
2. DC moves
   1. The request falls between two servers.
      - Ammortized cost of DC: $\gamma_t \leq 0$
   2. Only one server, say A, moves (request outside the border)
      - Suppose A moved δ
      - A has its match (from OPT's server) at destination
      - $M_{t+1} \leq M_t - \delta$
      - Each pairwise distance (A, B) (where B is another of DC's servers) increases by δ
      - Total distance increased: $S_{t+1} - S_t \leq (k-1) \cdot \delta$
      - Change in potential:

$$\Delta\Phi \leq -k \cdot \delta + (k-1) \cdot \delta = -\delta$$

      - Real cost incurred by DC: $c_{t+1} = \delta$

DC Analysis - Potential Function

1. OPT moves distance d
   - Ammortized cost of DC: $\gamma_t \leq k \cdot d$
2. DC moves
   1. The request falls between two servers.
      - Ammortized cost of DC: $\gamma_t \leq 0$
   2. Only one server, say A, moves (request outside the border)
      - Suppose A moved δ
      - A has its match (from OPT's server) at destination
      - $M_{t+1} \leq M_t - \delta$
      - Each pairwise distance (A, B) (where B is another of DC's servers) increases by δ
      - Total distance increased: $S_{t+1} - S_t \leq (k-1) \cdot \delta$
      - Change in potential:

$$\Delta\Phi \leq -k \cdot \delta + (k-1) \cdot \delta = -\delta$$

      - Real cost incurred by DC: $c_{t+1} = \delta$
      - Ammortized cost at this step: $\gamma_{t+1} \leq \delta - \delta = 0$

DC Analysis - Wrapping Up

1. OPT moves distance d
    - Ammortized cost of DC: $\gamma_t \leq k \cdot d$
2. DC moves
    1. The request falls between two servers.
        - Ammortized cost of DC: $\gamma_t \leq 0$
    2. Only one server moves (request outside the border)
        - Ammortized cost at this step: $\gamma_t \leq \delta - \delta = 0$

DC Analysis - Wrapping Up

1. OPT moves distance d
   - Ammortized cost of DC: $\gamma_t \leq k \cdot d$
2. DC moves
   1. The request falls between two servers.
      - Ammortized cost of DC: $\gamma_t \leq 0$
   2. Only one server moves (request outside the border)
      - Ammortized cost at this step: $\gamma_t =\leq \delta - \delta = 0$

- By our potential function inequality, we have:

$$\sum_{t=1}^{n} c_t \leq \Phi_0 + \sum_{t=1}^{n} \gamma_t$$

C_A $\quad 0 \quad \leq k \cdot d_t(OPT)$

$\leq k \cdot C_{OPT}$

DC Analysis - Wrapping Up

1. OPT moves distance d
   - Ammortized cost of DC: $\gamma_t \leq k \cdot d$
2. DC moves
   1. The request falls between two servers.
      - Ammortized cost of DC: $\gamma_t \leq 0$
   2. Only one server moves (request outside the border)
      - Ammortized cost at this step: $\gamma_t =\leq \delta - \delta = 0$

- By our potential function inequality, we have:

$$\sum_{t=1}^{n} c_t \leq \Phi_0 + \sum_{t=1}^{n} \gamma_t$$

- Since $\gamma_t \leq k \cdot d$ whenever OPT moves d, and $\gamma_t \leq 0$ when OPT doesn't move, we have that $\sum_t \gamma_t \leq k \cdot C_{opt}$

DC Analysis - Wrapping Up

1. OPT moves distance d
   - Ammortized cost of DC: $\gamma_t \leq k \cdot d$
2. DC moves
   1. The request falls between two servers.
      - Ammortized cost of DC: $\gamma_t \leq 0$
   2. Only one server moves (request outside the border)
      - Ammortized cost at this step: $\gamma_t =\leq \delta - \delta = 0$

- By our potential function inequality, we have:

$$\sum_{t=1}^{n} c_t \leq \Phi_0 + \sum_{t=1}^{n} \gamma_t$$

- Since $\gamma_t \leq k \cdot d$ whenever OPT moves d, and $\gamma_t \leq 0$ when OPT doesn't move, we have that $\sum_t \gamma_t \leq k \cdot C_{opt}$
- Since Φ_0 is the initial state, we can regard it as constant (even 0, if require that servers start at a certain place)

Conclusion

- Online algorithms are important for many applications, when we need to make decisions right when we receive the information.

Conclusion

- Online algorithms are important for many applications, when we need to make decisions right when we receive the information.
- Applications in
  - Stock Market
  - Dating
  - Skiing
  - Caching
  - Machine Learning (regret minimization)
  - many more...

Conclusion

- Online algorithms are important for many applications, when we need to make decisions right when we receive the information.
- Applications in
  - Stock Market
  - Dating
  - Skiing
  - Caching
  - Machine Learning (regret minimization)
  - many more...
- *Competitive Analysis*: measures performance of our algorithm against best algorithm that could *see into the future*

Conclusion

- Online algorithms are important for many applications, when we need to make decisions right when we receive the information.
- Applications in
    - Stock Market
    - Dating
    - Skiing
    - Caching
    - Machine Learning (regret minimization)
    - many more...
- *Competitive Analysis*: measures performance of our algorithm against best algorithm that could *see into the future*
- Saw how to use *minimax theorem* in *Yao's principle* to prove lower bounds for randomized online algorithms.

Acknowledgement

- Lecture based largely on:
  - Lectures 18 & 20 of Karger's 6.854 Fall 2004 algorithms course
  - [Motwani & Raghavan 2007, Chapter 13]
- See Karger's Lecture 18 notes at

  `http://courses.csail.mit.edu/6.854/06/scribe/s23-onlineRandomLb.pdf`
- See Karger's Lecture 20 notes at

  `http://courses.csail.mit.edu/6.854/06/scribe/s24-paging.pdf`

References I

Motwani, Rajeev and Raghavan, Prabhakar (2007)
Randomized Algorithms