Prefetching and Caching for Minimizing Service Costs: Optimal and Approximation Strategies

Guocong Quan, Atilla Eryilmaz, Jian Tan, Ness Shroff

The Ohio State University

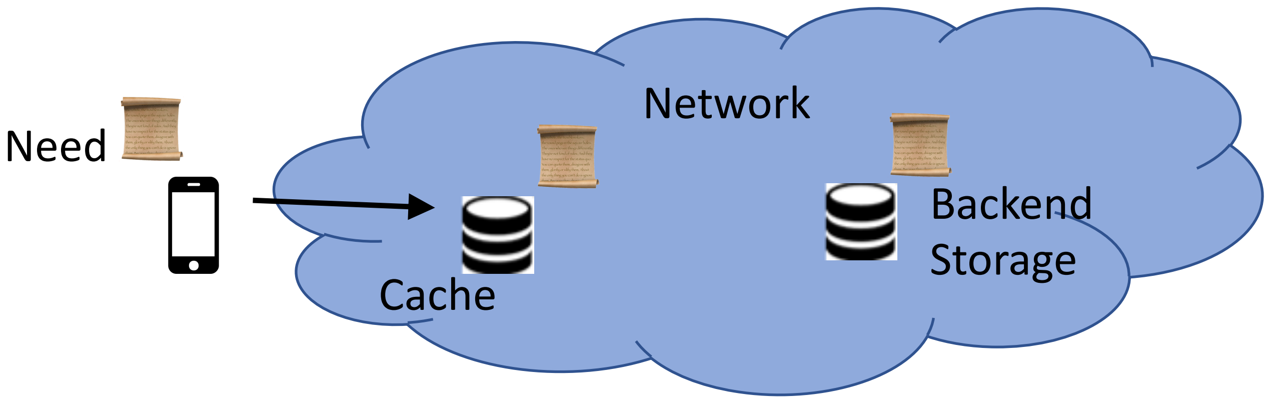
IFIP Performance 2020

Prefetching & Caching

- Use caching to store the data closer to users

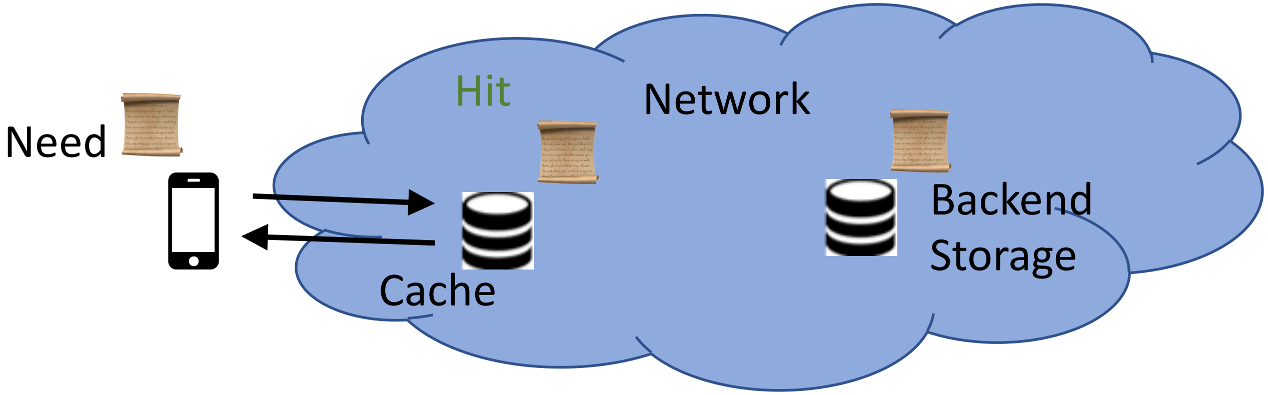
Prefetching & Caching

- Use caching to store the data closer to users



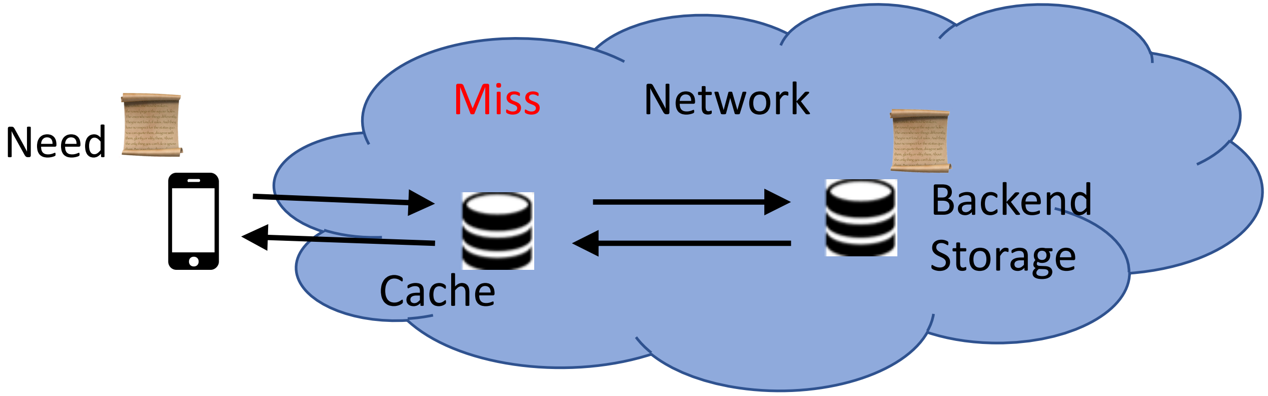
Prefetching & Caching

- Use caching to store the data closer to users



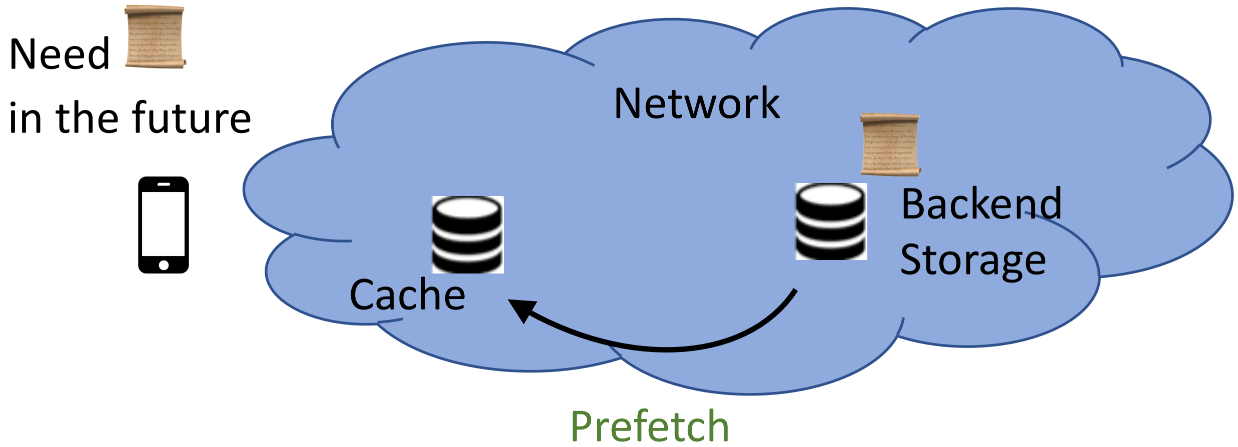
Prefetching & Caching

- Use caching to store the data closer to users



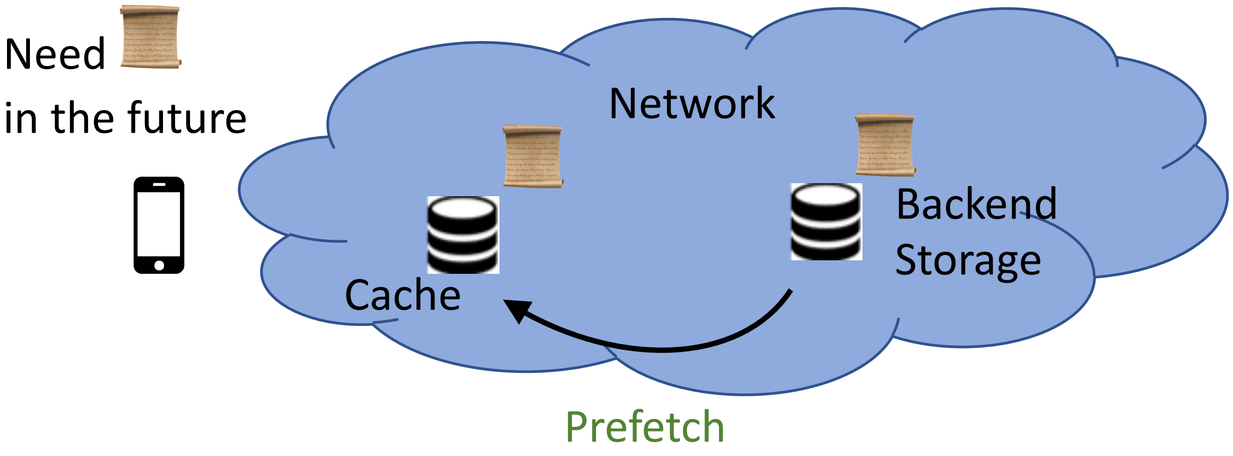
Prefetching & Caching

- Use caching to store the data closer to users
- Use prefetching to load the data into caches before users' requests



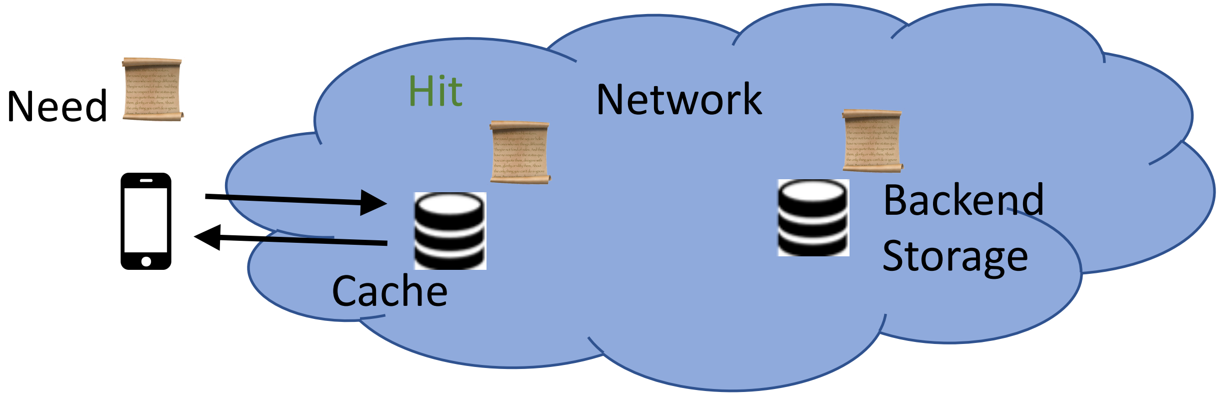
Prefetching & Caching

- Use caching to store the data closer to users
- Use prefetching to load the data into caches before users' requests



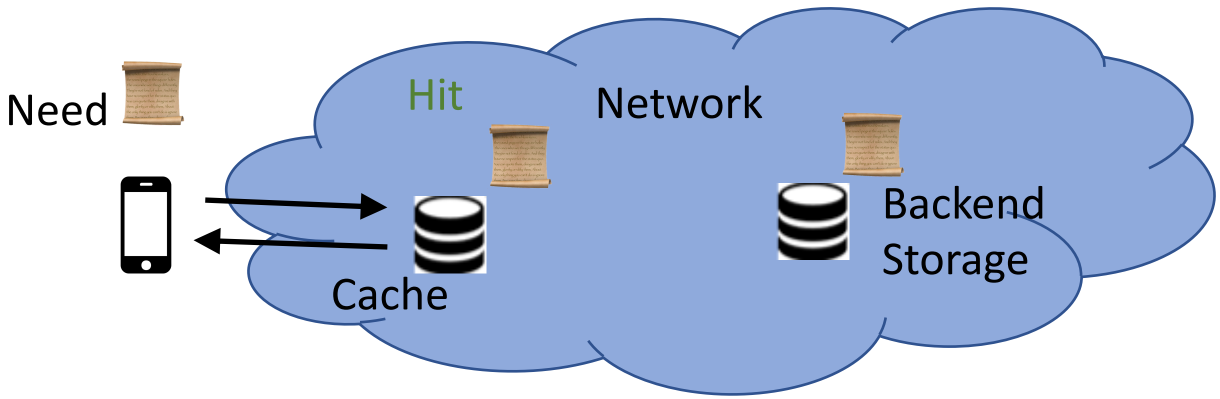
Prefetching & Caching

- Use caching to store the data closer to users
- Use prefetching to load the data into caches before users' requests



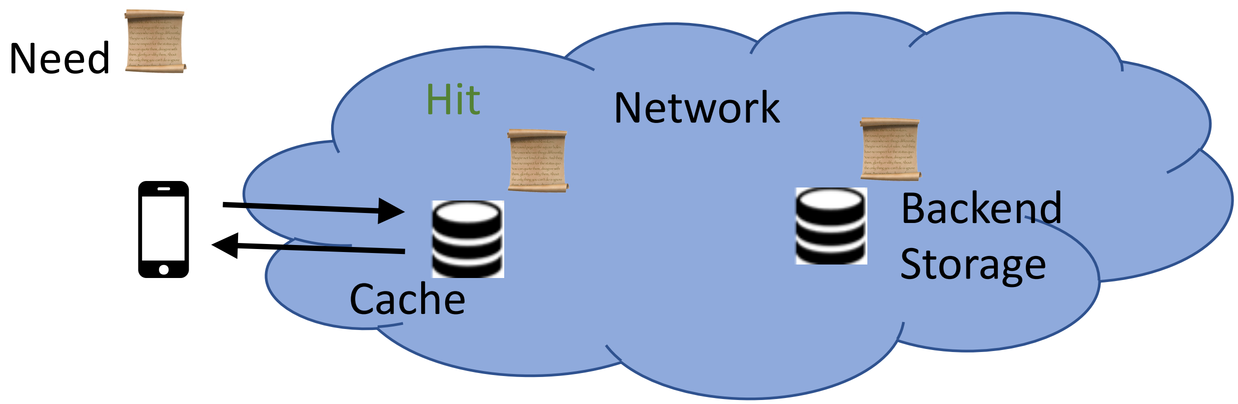
Prefetching & Caching

- Use caching to store the data closer to users
- Use prefetching to load the data into caches before users' requests
- Benefits of prefetching and caching:
  - Service delay is lower
  - Prefetching is less time-sensitive, compared to fetching



Prefetching & Caching

- Use caching to store the data closer to users
- Use prefetching to load the data into caches before users' requests
- Benefits of prefetching and caching:
  - Service delay is lower
  - Prefetching is less time-sensitive



Is prefetching always a good choice?

If not, for which requests should we choose to prefetch?

Problem Formulation

- $d_i, 1 \leq i \leq M$: a set of data items

Problem Formulation

- $d_i, 1 \leq i \leq M$: a set of data items
- $R_n, 1 \leq n \leq N$: a sequence of requests for the data items (assume to be known)

Problem Formulation

- $d_i, 1 \leq i \leq M$: a set of data items
- $R_n, 1 \leq n \leq N$: a sequence of requests for the data items (assume to be known)
- Data retrieval operations:

Operation	Time	Caching	Cost
Fetch			
Prefetch			

Problem Formulation

- $d_i, 1 \leq i \leq M$: a set of data items
- $R_n, 1 \leq n \leq N$: a sequence of requests for the data items (assume to be known)
- Data retrieval operations:

Operation	Time	Caching	Cost
Fetch	After the request		
Prefetch	Before the request		

Problem Formulation

- $d_i, 1 \le i \le M$: a set of data items
- $R_n, 1 \le n \le N$: a sequence of requests for the data items (assume to be known)
- Data retrieval operations:

Operation	Time	Caching	Cost
Fetch	After the request	May not load into cache	
Prefetch	Before the request	Have to load into cache	

Problem Formulation

- $d_i, 1 \leq i \leq M$: a set of data items
- $R_n, 1 \leq n \leq N$: a sequence of requests for the data items (assume to be known)
- Data retrieval operations:

Operation	Time	Caching	Cost
Fetch	After the request	May not load into cache	1
Prefetch	Before the request	Have to load into cache	$c \in [0,1]$

Problem Formulation

- $d_i, 1 \leq i \leq M$: a set of data items
- $R_n, 1 \leq n \leq N$: a sequence of requests for the data items (assume to be known)
- Data retrieval operations:

Operation	Time	Caching	Cost
Fetch	After the request	May not load into cache	1
Prefetch	Before the request	Have to load into cache	$c \in [0,1]$

- Optimal offline prefetching & caching:
  - Goal: Minimize the overall cost by choosing proper data retrieval operations to serve the request sequence
  - Constraint: The cache size is limited

Problem Formulation

- $d_i, 1 \leq i \leq M$: a set of data items
- $R_n, 1 \leq n \leq N$: a sequence of requests for the data items (assume to be known)
- Data retrieval operations:

Operation	Time	Caching	Cost
Fetch	After the request	May not load into cache	1
Prefetch	Before the request	Have to load into cache	$c \in [0,1]$

- Optimal offline prefetching & caching:
  - Goal: Minimize the overall cost by choosing proper data retrieval operations to serve the request sequence
  - Constraint: The cache size is limited

Question 1: Should we prefetch or fetch the requested item, if it is not stored in the cache?

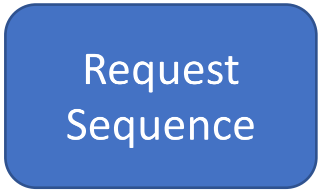
Question 2: Which data item should be evicted, if we load a new data item into the cache?

Optimal Policy via Min-Cost Flow

Flow-based optimal offline policy π_{OPT}:

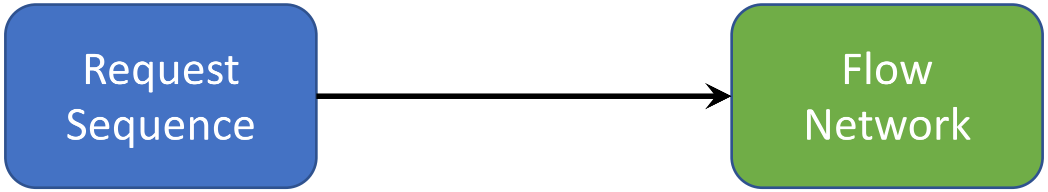
Optimal Policy via Min-Cost Flow

Flow-based optimal offline policy π_{OPT}:



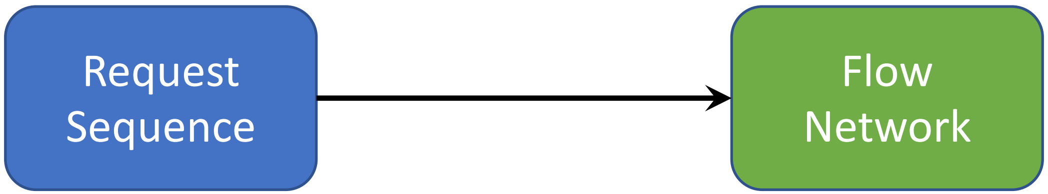
Optimal Policy via Min-Cost Flow

Flow-based optimal offline policy π_{OPT}:



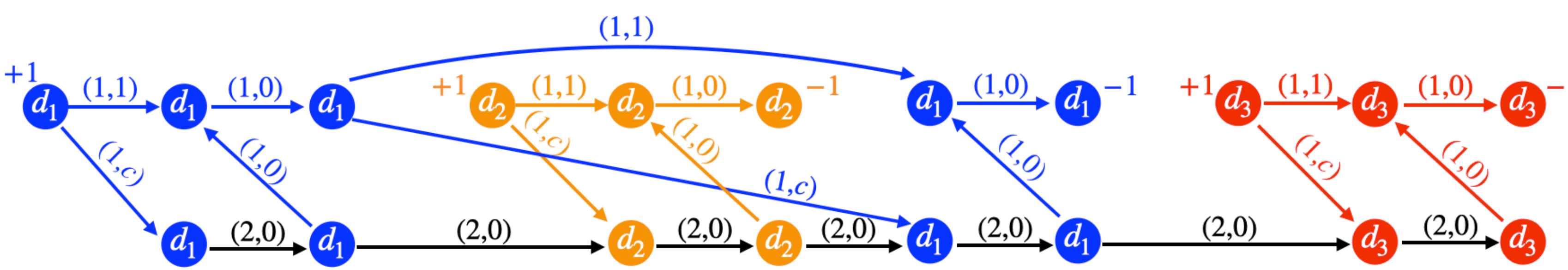
Optimal Policy via Min-Cost Flow

Flow-based optimal offline policy π_{OPT}:



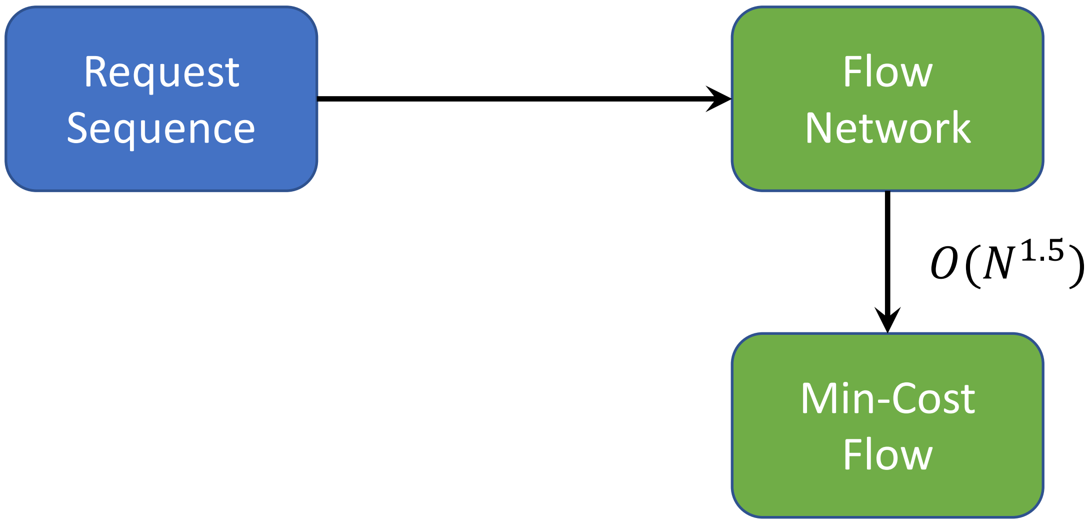
Cache of size 2

Request sequence: d_1, d_2, d_1, d_3



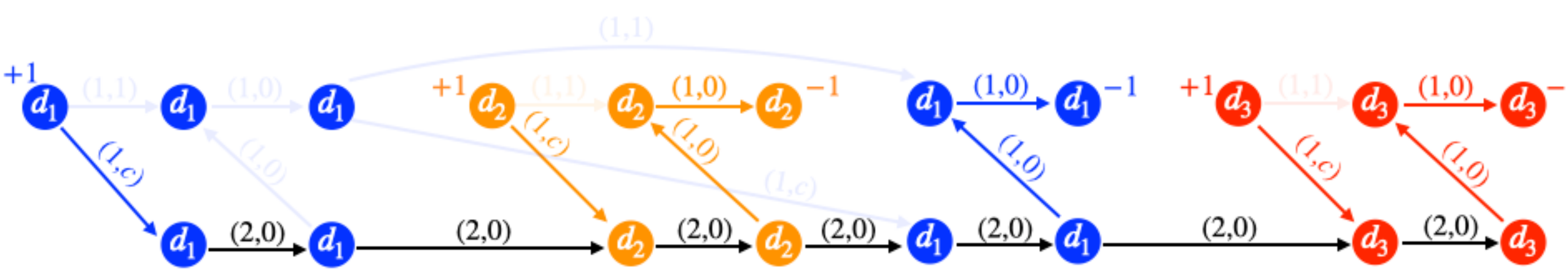
Optimal Policy via Min-Cost Flow

Flow-based optimal offline policy π_{OPT}:



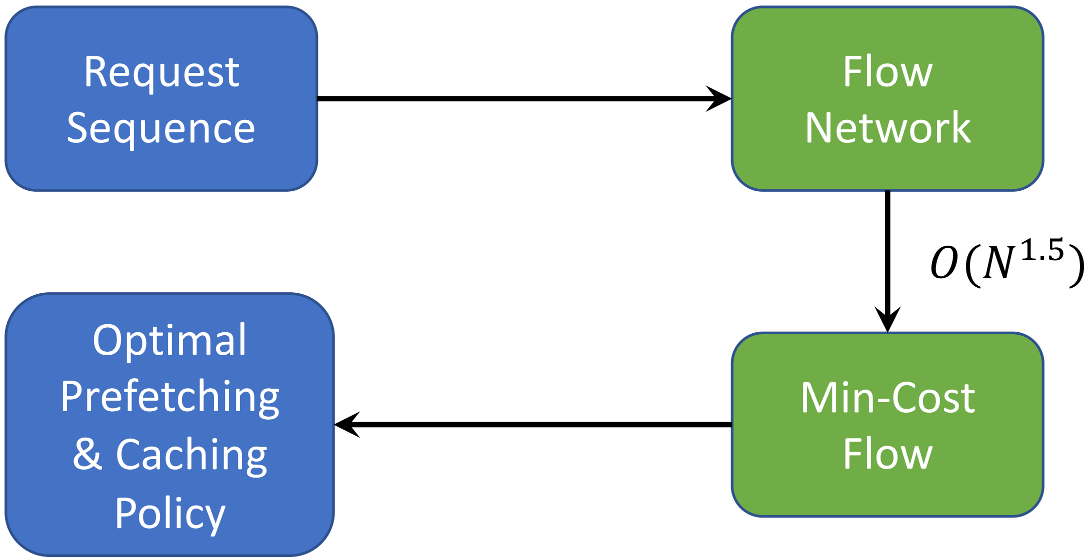
Cache of size 2

Request sequence: d_1, d_2, d_1, d_3



Optimal Policy via Min-Cost Flow

Flow-based optimal offline policy π_{OPT}:

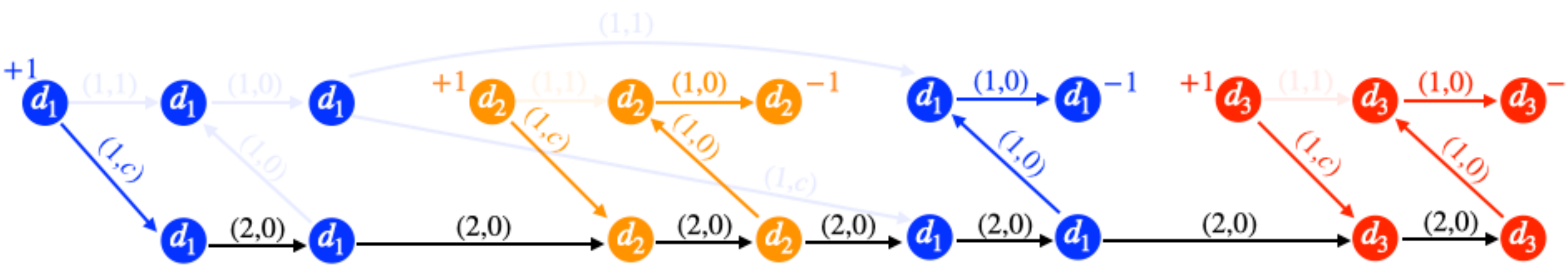


Cache of size 2

Request sequence: d_1, d_2, d_1, d_3

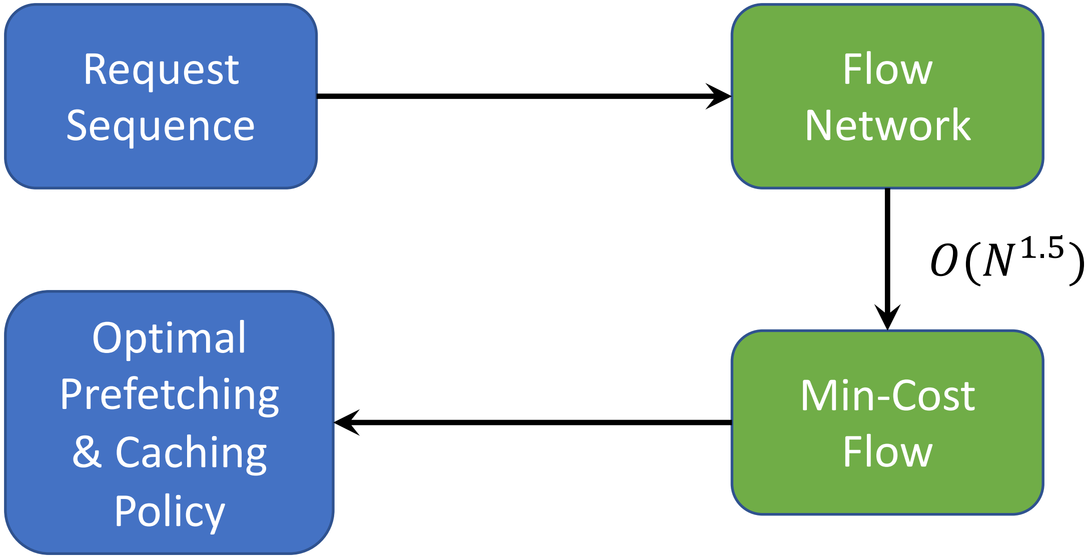
Prefetch $R_1 = d_1$, Prefetch $R_2 = d_2$,

Hit $R_3 = d_1$, Prefetch $R_4 = d_3$ & evict d_1



Optimal Policy via Min-Cost Flow

Flow-based optimal offline policy π_{OPT}:



Drawbacks:

- Cannot analytically answer the two proposed questions
- Need to know all future requests to make optimal decisions
- The decision for a request is unavailable unless the entire sequence is processed

Characteristics of OPT

- Question 1: Should we prefetch or fetch a data item, if it is not stored in the cache?
- Question 2: Which data item should be evicted, if we cache a new data item?

Characteristics of OPT

- Question 1: Should we prefetch or fetch a data item, if it is not stored in the cache?
- Question 2: Which data item should be evicted, if we choose to prefetch and cache is full?

Theorem 1: There exists an optimal policy that always evicts the farthest-in-future (FF) item.

- FF item: the item that is stored in the cache, and requested farthest in future.

Characteristics of OPT

- Question 1: Should we prefetch or fetch a data item, if it is not stored in the cache?
- Question 2: Which data item should be evicted, if we choose to prefetch and cache is full?

Theorem 1: There exists an optimal policy that always evicts the farthest-in-future (FF) item.

- FF item: the item that is stored in the cache, and requested farthest in future.

Theorem 2: Assuming the FF eviction policy is adopted, prefetching a missed data item is the optimal decision if any of the following two conditions is satisfied:
C1: There exists a request for a *popular* item in *near future*, but that data item is not cached.
C2: The prefetching cost c is *sufficiently low*.

Characteristics of OPT

- Question 1: Should we prefetch or fetch a data item, if it is not stored in the cache?
- Question 2: Which data item should be evicted, if we choose to prefetch and cache is full?

Theorem 1: There exists an optimal policy that always evicts the farthest-in-future (FF) item.

- FF item: the item that is stored in the cache, and requested farthest in future.

Theorem 2: Assuming the FF eviction policy is adopted, prefetching a missed data item is the optimal decision if any of the following two conditions is satisfied:
C1: There exists a request for a *popular* item in *near future*, but that data item is not cached.
C2: The prefetching cost c is *sufficiently low*.

Approximation Strategy

Approximation policy π_A:

When the requested data is not cached, prefetch the data item and evict the FF item, if

- $c \leq \sqrt{2}/2$
- Or, any of C1 and C2 is satisfied

Otherwise, fetch the data and do not load it into the cache.

Approximation Strategy

Approximation policy π_A:

When the requested data is not cached, prefetch the data item and evict the FF item, if

- $c \leq \sqrt{2}/2$
- Or, any of C1 and C2 is satisfied

Otherwise, fetch the data and do not load it into the cache.

- π_A only requires near future information.
- The time complexity is $O(N)$.
- π_A achieves near optimal performance.

Competitive Ratio

Always prefetching policy π_P:

Always prefetch the requested data item if it is not cached, and evict the FF item.

Always fetching policy π_F (Belady's algorithm):

Always fetch the missed data item and adopt the FF eviction policy.

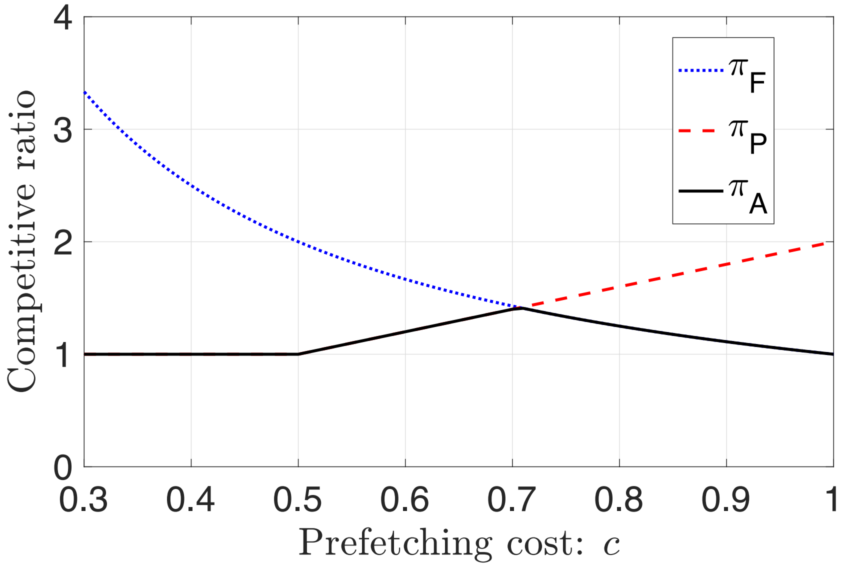
Competitive Ratio

Always prefetching policy π_P:

Always prefetch the requested data item if it is not cached, and evict the FF item.

Always fetching policy π_F (Belady's algorithm):

Always fetch the missed data item and adopt the FF eviction policy.



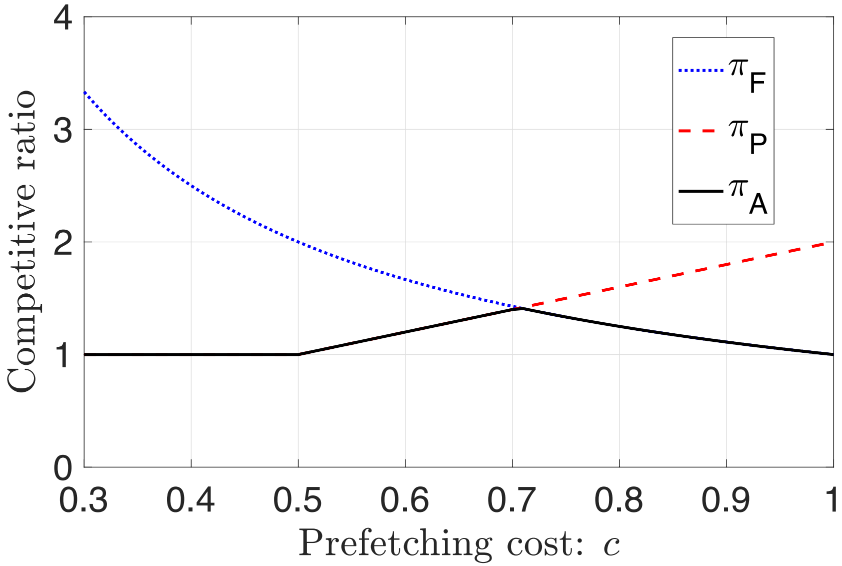
Competitive Ratio

Always prefetching policy π_P:

Always prefetch the requested data item if it is not cached, and evict the FF item.

Always fetching policy π_F (Belady's algorithm):

Always fetch the missed data item and adopt the FF eviction policy.



The competitive ratio of π_A is

- equal to 1 for $c \leq 0.5$ or $c = 1$
- at most $\sqrt{2}$
- always less than the ones of π_P and π_F

Evaluation

Optimal static policy π_S:

Store the most popular items in the cache. Never update the cache content.

Evaluation

Optimal static policy π_S:

Store the most popular items in the cache. Never update the cache content.

- Cache size: 20
- Popularity distribution: $p_i = \mathbb{P}[R_n = d_i], 1 \leq i \leq 10^6, 1 \leq n \leq 10^5$

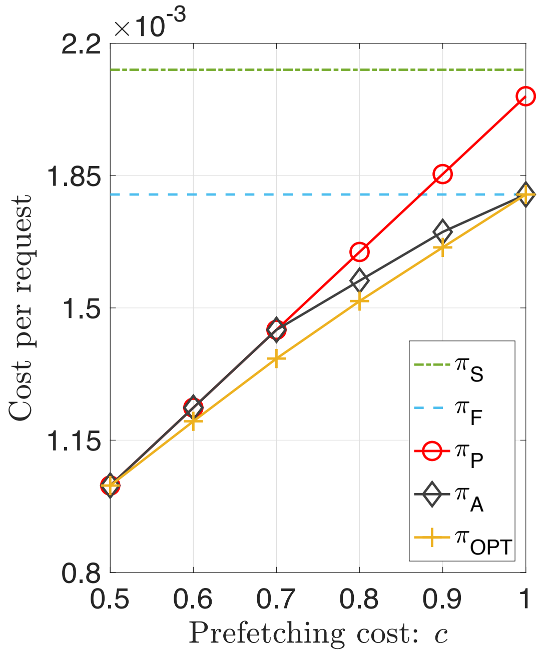
Evaluation

Optimal static policy π_S:

Store the most popular items in the cache. Never update the cache content.

- Cache size: 20
- Popularity distribution: $p_i = \mathbb{P}[R_n = d_i], 1 \leq i \leq 10^6, 1 \leq n \leq 10^5$

$$p_i = c_1 \cdot \exp(-0.3i)$$



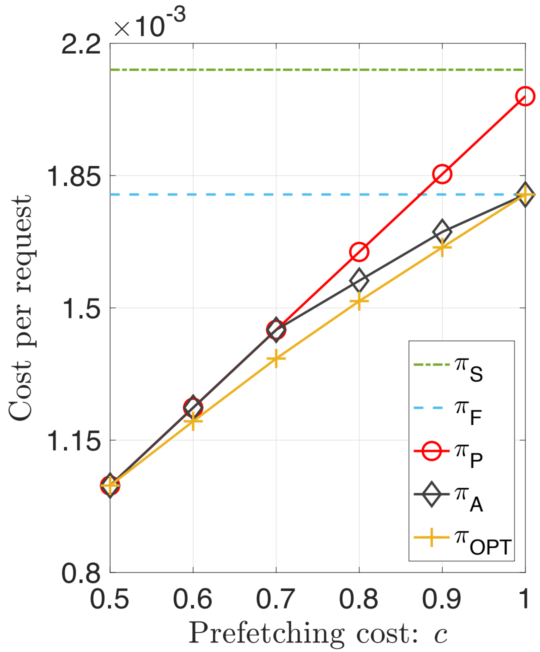
Evaluation

Optimal static policy π_S:

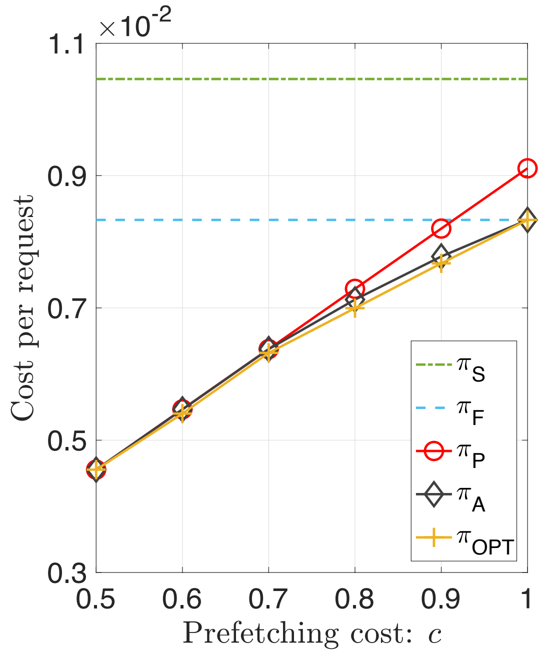
Store the most popular items in the cache. Never update the cache content.

- Cache size: 20
- Popularity distribution: $p_i = \mathbb{P}[R_n = d_i], 1 \le i \le 10^6, 1 \le n \le 10^5$

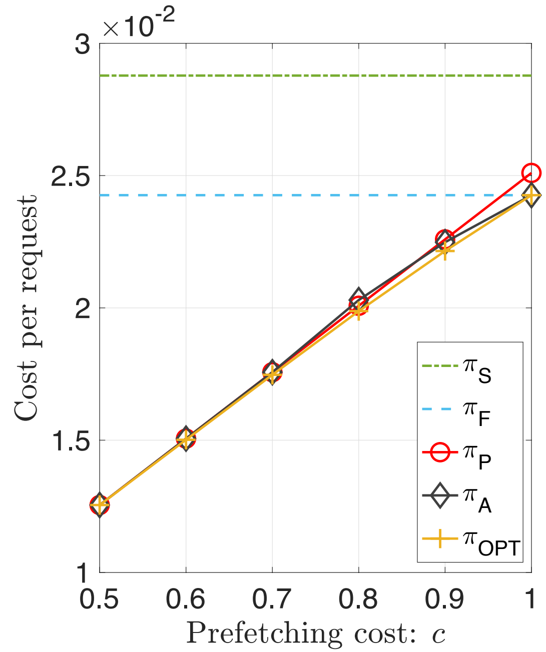
$p_i = c_1 \cdot \exp(-0.3i)$



$p_i = c_2 \cdot \exp(-i^{0.6})$



$p_i = c_3 \cdot i^{-2}$



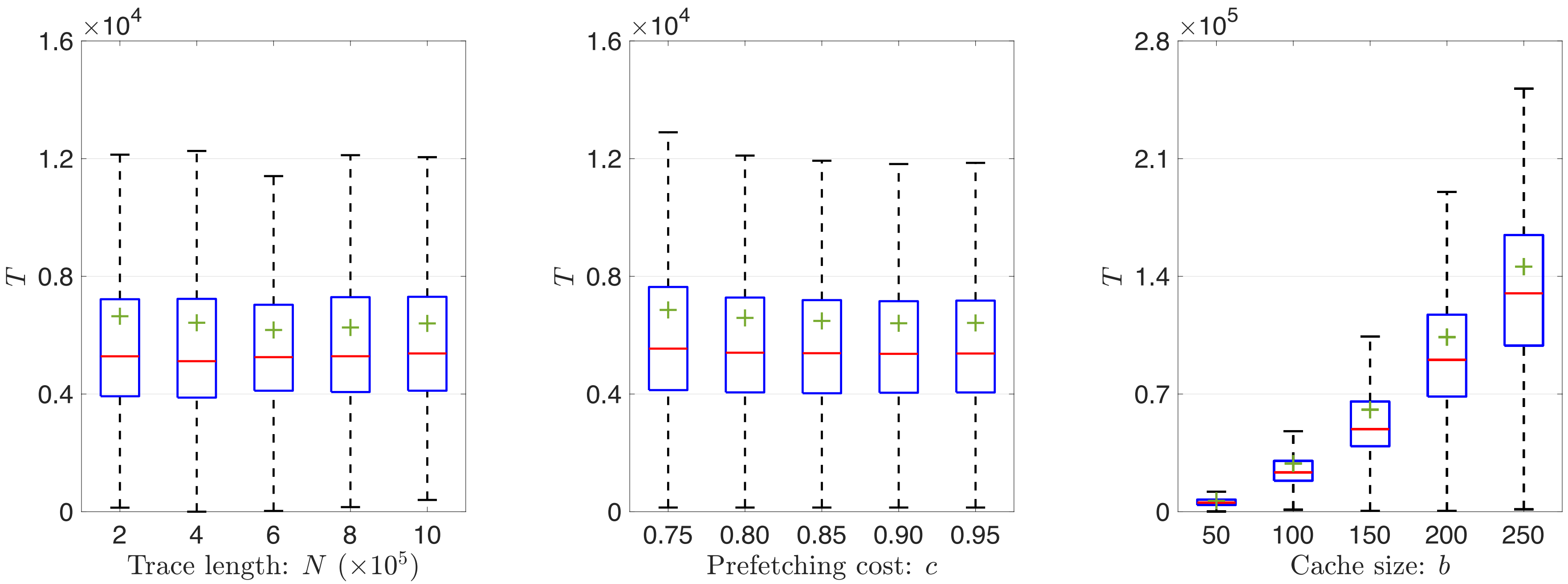
Evaluation

- T: the number of future requests required by π_A to make decisions
- $p_i = c \cdot i^{-2}, 1 \leq i \leq 10^6, 1 \leq n \leq N$

Evaluation

- T: the number of future requests required by π_A to make decisions
- $p_i = c \cdot i^{-2}, 1 \le i \le 10^6, 1 \le n \le N$

$c = 0.85, b = 50$ | $N = 5\times10^5, b = 50$ | $N = 5\times10^5, c = 0.85$



Summary

- Formulated a cost-based optimal offline prefetching and caching problem

Summary

- Formulated a cost-based optimal offline prefetching and caching problem
- Proposed a flow-based optimal offline policy

Summary

- Formulated a cost-based optimal offline prefetching and caching problem
- Proposed a flow-based optimal offline policy
- Analytically characterized the optimal offline policy
  - Optimal eviction policy
  - Sufficient conditions for optimal prefetching

Summary

- Formulated a cost-based optimal offline prefetching and caching problem
- Proposed a flow-based optimal offline policy
- Analytically characterized the optimal offline policy
  - Optimal eviction policy
  - Sufficient conditions for optimal prefetching
- Designed a lightweight approximation policy
  - Require near-future information
  - Achieve near-optimal performance

Q & A

Thank you!