Online Demand Scheduling with Failovers

Konstantina Mellou Marco Molinaro **Rudy Zhou***

Microsoft Research, Redmond Carnegie Mellon

* work performed as intern in Cloud Operations Research (CORE) group at Microsoft Research, Redmond

Robust Assignments

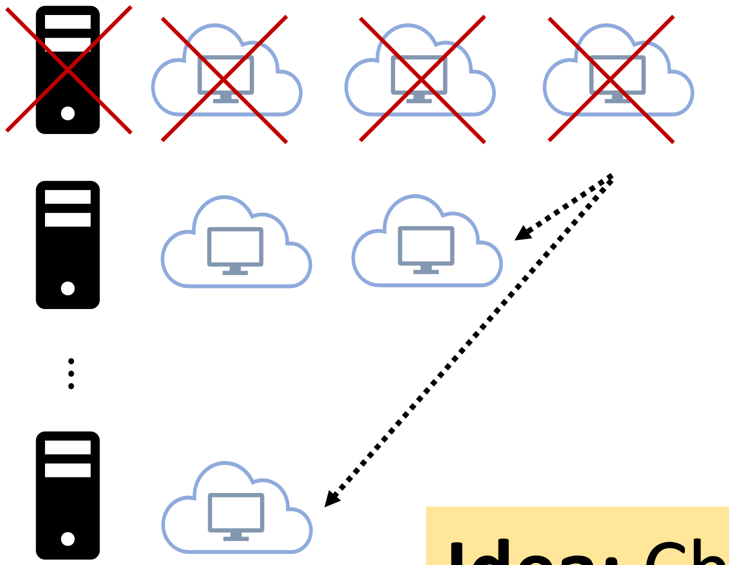
- What happens if a machine fails?

Robust Assignments

- What happens if a machine fails?

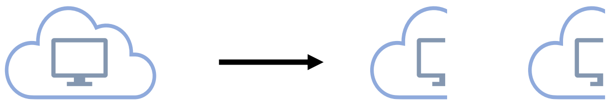
Robust Assignments

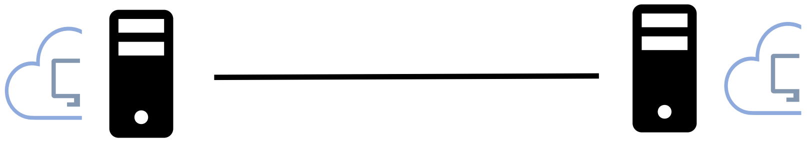
- What happens if a machine fails?
- How to reassign?



Idea: Change assignment process to make reassignment in case of failure easier

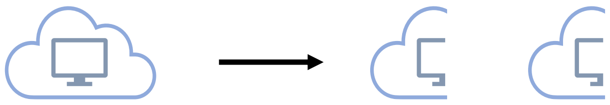
New Model: Redundancy

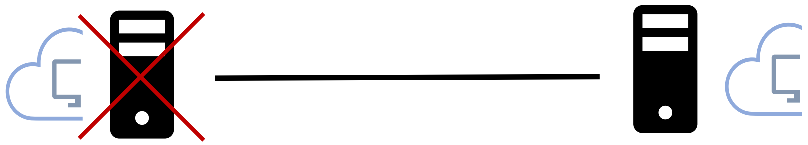
- Inspired by real systems architectures
- Split each demand in half: 
- Assign each half to distinct machines



Chaojie Zhang, Alok Gautam Kumbhare, Ioannis Manousakis, Deli Zhang, Pulkit A. Misra, Rod Assis, Kyle Woolcock, Nithish Mahalingam, Brijesh Warrier, David Gauthier, Lalu Kunnath, Steve Solomon, Osvaldo Morales, Marcus Fontoura, Ricardo Bianchini:
Flex: High-Availability Datacenters With Zero Reserved Power. ISCA 2021

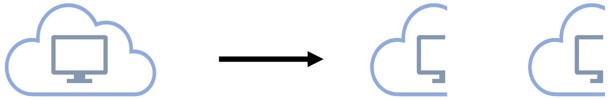
New Model: Redundancy

- Inspired by real systems architectures
- Split each demand in half: 
- Assign each half to distinct machines
- If a machine fails, reassign all half-demands there to the machine of their other half, possibly with increased capacity **(failover scenario)**



Chaojie Zhang, Alok Gautam Kumbhare, Ioannis Manousakis, Deli Zhang, Pulkit A. Misra, Rod Assis, Kyle Woolcock, Nithish Mahalingam, Brijesh Warrier, David Gauthier, Lalu Kunnath, Steve Solomon, Osvaldo Morales, Marcus Fontoura, Ricardo Bianchini:
Flex: High-Availability Datacenters With Zero Reserved Power. ISCA 2021

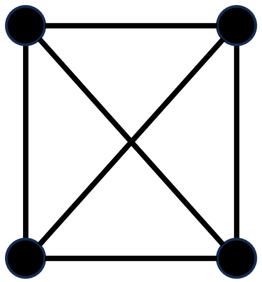
New Model: Redundancy

- Inspired by real systems architectures
- Split each demand in half: 
- Assign each half to distinct machines
- If a machine fails, reassign all half-demands there to the machine of their other half, possibly with increased capacity **(failover scenario)**

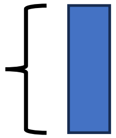
Chaojie Zhang, Alok Gautam Kumbhare, Ioannis Manousakis, Deli Zhang, Pulkit A. Misra, Rod Assis, Kyle Woolcock, Nithish Mahalingam, Brijesh Warrier, David Gauthier, Lalu Kunnath, Steve Solomon, Osvaldo Morales, Marcus Fontoura, Ricardo Bianchini:
Flex: High-Availability Datacenters With Zero Reserved Power. ISCA 2021

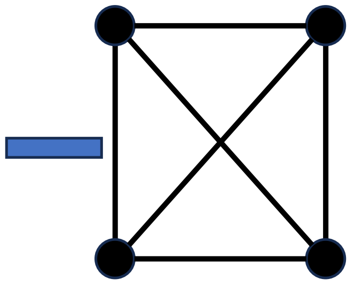
Online Demand Scheduling with Failover

- m machines

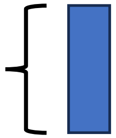


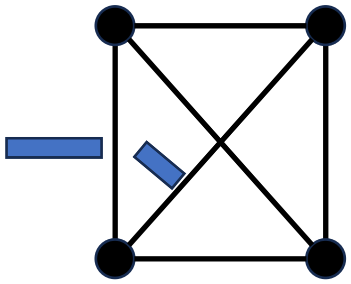
Online Demand Scheduling with Failover

- m machines
- n demands arrive online with sizes: 
- Must assign demand to edge (pair of machines) upon arrival such that:
  - **Nominal constraint**: Load incident to each machine is ≤ 1
  - **Failover constraint:** In every failover scenario (single machine failure), the load incident to each machine is $\leq B$

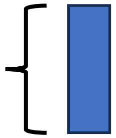


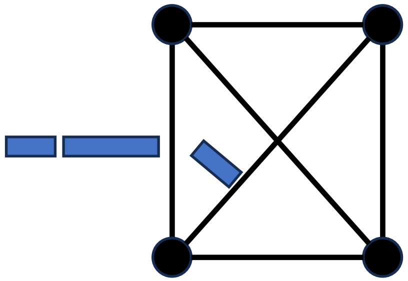
Online Demand Scheduling with Failover

- m machines
- n demands arrive online with sizes: 
- Must assign demand to edge (pair of machines) upon arrival such that:
  - **Nominal constraint**: Load incident to each machine is ≤ 1
  - **Failover constraint:** In every failover scenario (single machine failure), the load incident to each machine is $\leq B$

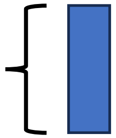


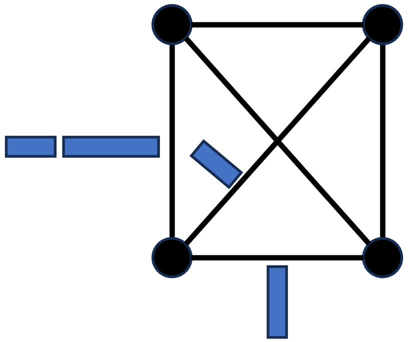
Online Demand Scheduling with Failover

- m machines
- n demands arrive online with sizes: 
- Must assign demand to edge (pair of machines) upon arrival such that:
  - **Nominal constraint**: Load incident to each machine is ≤ 1
  - **Failover constraint:** In every failover scenario (single machine failure), the load incident to each machine is $\leq B$

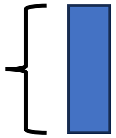


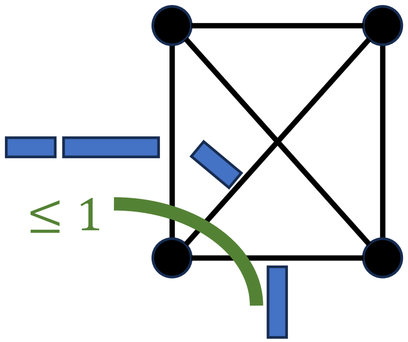
Online Demand Scheduling with Failover

- m machines
- n demands arrive online with sizes: 
- Must assign demand to edge (pair of machines) upon arrival such that:
  - **Nominal constraint**: Load incident to each machine is ≤ 1
  - **Failover constraint:** In every failover scenario (single machine failure), the load incident to each machine is $\leq B$

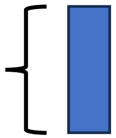


Online Demand Scheduling with Failover

- m machines
- n demands arrive online with sizes: 
- Must assign demand to edge (pair of machines) upon arrival such that:
  - **Nominal constraint**: Load incident to each machine is ≤ 1
  - **Failover constraint:** In every failover scenario (single machine failure), the load incident to each machine is $\leq B$

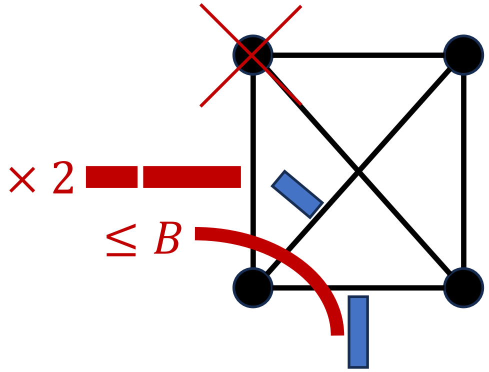


Online Demand Scheduling with Failover

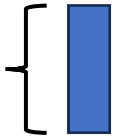
- m machines
- n demands arrive online with sizes: 
- Must assign demand to edge (pair of machines) upon arrival such that:
  - **Nominal constraint:** Load incident to each machine is ≤ 1
  - **Failover constraint:** In every failover scenario (single machine failure), the load incident to each machine is $\leq B$

$$\sum_{v \neq u} Load_{uv} + \max_{v \neq u} Load_{uv} \leq B$$

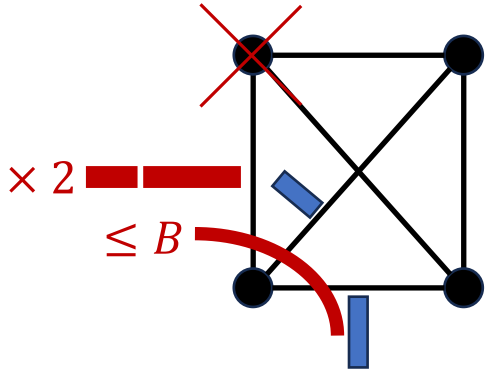
for every machine u



Online Demand Scheduling with Failover

- m machines
- n demands arrive online with sizes: 
- Must assign demand to edge (pair of machines) upon arrival such that:
  - **Nominal constraint**: Load incident to each machine is ≤ 1
  - **Failover constraint:** In every failover scenario (single machine failure), the load incident to each machine is $\leq B$

$\sum_{v \neq u} Load_{uv} + \max_{v \neq u} Load_{uv} \leq B$
for every machine u



Goal: maximize utilization (total size of assigned demands) until first demand that needs to be rejected

Related Work

- Multiple Knapsack [Chandra Chekuri, Sanjeev Khanna. SIAM J. Comput. 2005]
- Coupled placement [Madhukar R. Korupolu, Adam Meyerson, Rajmohan Rajaraman, Brian Tagiku. Math. Prog. 2015]
- Do not capture failover constraints (depends on how load is arranged on a machine's edges)

Our Results

Compared to optimal offline policy that knows all demands but also assigns demands in same order until rejecting

Theorem (Worst Case): $(\frac{\mathbf{1}}{\mathbf{2}} - o(1))$ - competitive deterministic algorithm

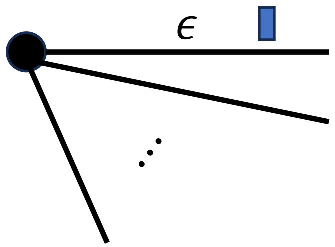
Theorem (Stochastic): If all demand sizes are drawn i.i.d. from an unknown distribution, then $(\mathbf{1} - o(1))$ – competitive algorithm w.h.p.

- No deterministic algorithm is better than $\frac{1}{2}$ - competitive

Worst Case: Idea

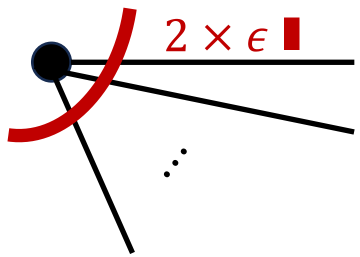
$\sum_{v \neq u} Load_{uv} + \max_{v \neq u} Load_{uv} \leq B$

for every machine u



$\sum_{v \neq u} Load_{uv} + \max_{v \neq u} Load_{uv} \leq B$
for every machine u

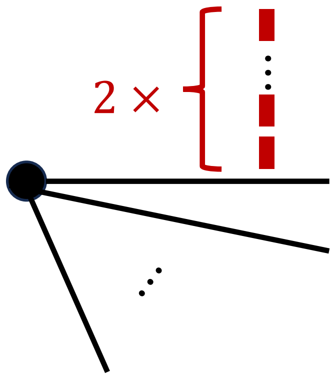
Worst Case: Idea



$\sum_{v \neq u} Load_{uv} + \max_{v \neq u} Load_{uv} \leq B$

for every machine u

Worst Case: Idea

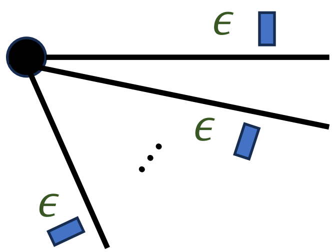


- Minimize impact of failover by spreading out demands

$\sum_{v \neq u} Load_{uv} + \max_{v \neq u} Load_{uv} \leq B$

for every machine u

Worst Case: Idea

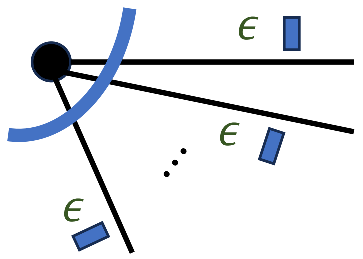


- Minimize impact of failover by spreading out demands

$\sum_{v \neq u} Load_{uv} + \max_{v \neq u} Load_{uv} \leq B$
for every machine u

Worst Case: Idea

$degree = \frac{1}{\epsilon} - 1$ (assume $B = 1$)

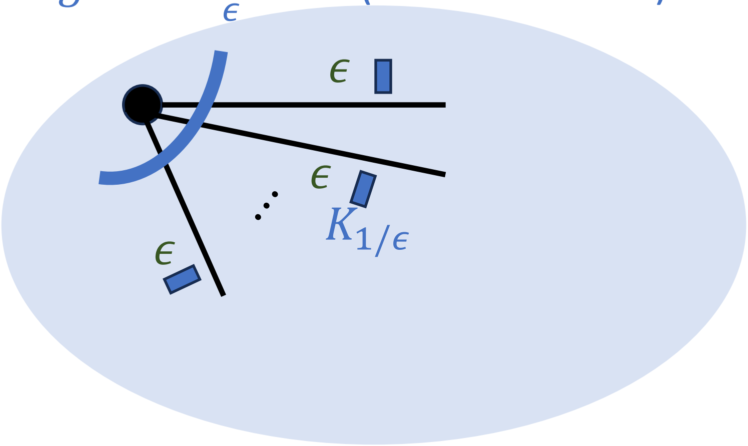


- Minimize impact of failover by spreading out demands

$\sum_{v \neq u} Load_{uv} + \max_{v \neq u} Load_{uv} \leq B$
for every machine u

Worst Case: Idea

$degree = \frac{1}{\epsilon} - 1$ (assume $B = 1$)



- Minimize impact of failover by spreading out demands
- Ideally, want to make clique of machines for same size demands

$\sum_{v \neq u} Load_{uv} + \max_{v \neq u} Load_{uv} \leq B$
for every machine u

Worst Case: Idea

- Minimize impact of failover by spreading out demands
- Ideally, want to make clique of machines for same size demands
- … but need to make sure we don't run out of machines

Worst Case: Algorithm

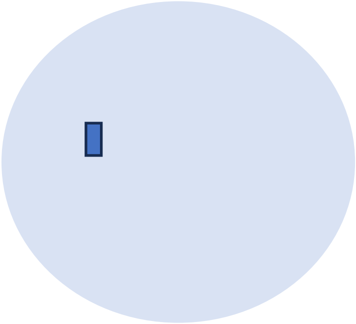
- Assume $B = 1 \Rightarrow$ want to arrange demands of size $\frac{1}{k}$ in a K_k
- Assume all demand sizes are $\frac{1}{k}$ for integer k

Algorithm: If a demand of size $\frac{1}{k}$ arrives, assign it to an open edge in a reserved K_k; otherwise reserve a new K_k and assign it there

Worst Case: Algorithm

- Assume $B = 1 \Rightarrow$ want to arrange demands of size $\frac{1}{k}$ in a K_k
- Assume all demand sizes are $\frac{1}{k}$ for integer k

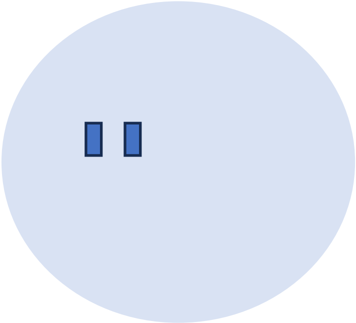
Algorithm: If a demand of size $\frac{1}{k}$ arrives, assign it to an open edge in a reserved K_k; otherwise reserve a new K_k and assign it there



Worst Case: Algorithm

- Assume $B = 1 \Rightarrow$ want to arrange demands of size $\frac{1}{k}$ in a K_k
- Assume all demand sizes are $\frac{1}{k}$ for integer k

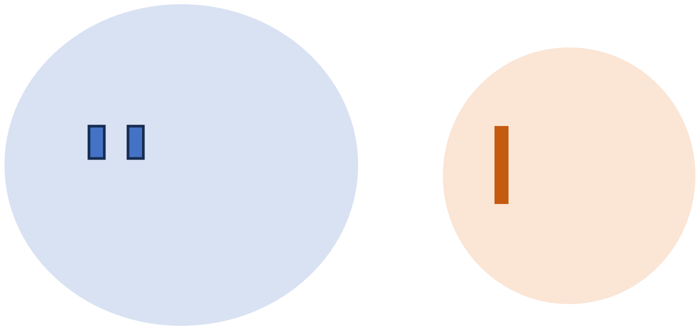
Algorithm: If a demand of size $\frac{1}{k}$ arrives, assign it to an open edge in a reserved K_k; otherwise reserve a new K_k and assign it there



Worst Case: Algorithm

- Assume $B = 1 \Rightarrow$ want to arrange demands of size $\frac{1}{k}$ in a K_k
- Assume all demand sizes are $\frac{1}{k}$ for integer k

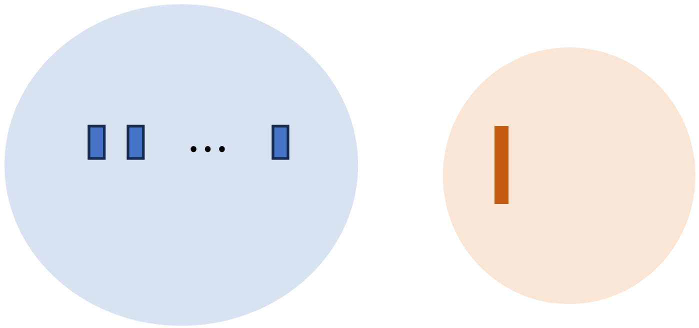
Algorithm: If a demand of size $\frac{1}{k}$ arrives, assign it to an open edge in a reserved K_k; otherwise reserve a new K_k and assign it there



Worst Case: Algorithm

- Assume $B = 1 \Rightarrow$ want to arrange demands of size $\frac{1}{k}$ in a K_k
- Assume all demand sizes are $\frac{1}{k}$ for integer k

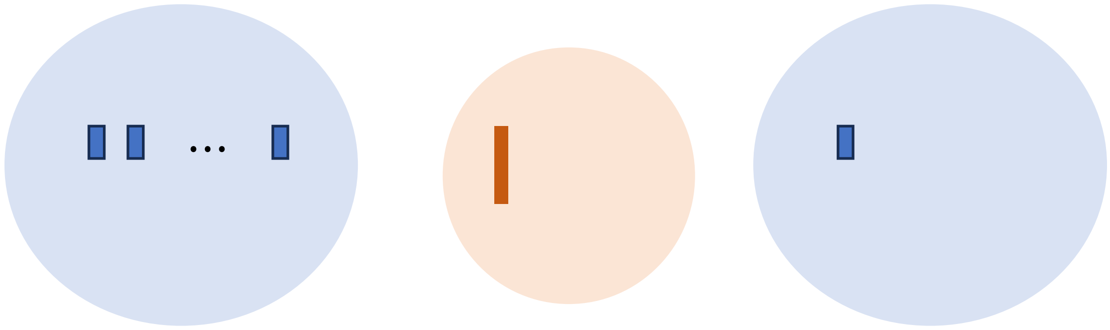
Algorithm: If a demand of size $\frac{1}{k}$ arrives, assign it to an open edge in a reserved K_k; otherwise reserve a new K_k and assign it there



Worst Case: Algorithm

- Assume $B = 1 \Rightarrow$ want to arrange demands of size $\frac{1}{k}$ in a K_k
- Assume all demand sizes are $\frac{1}{k}$ for integer k

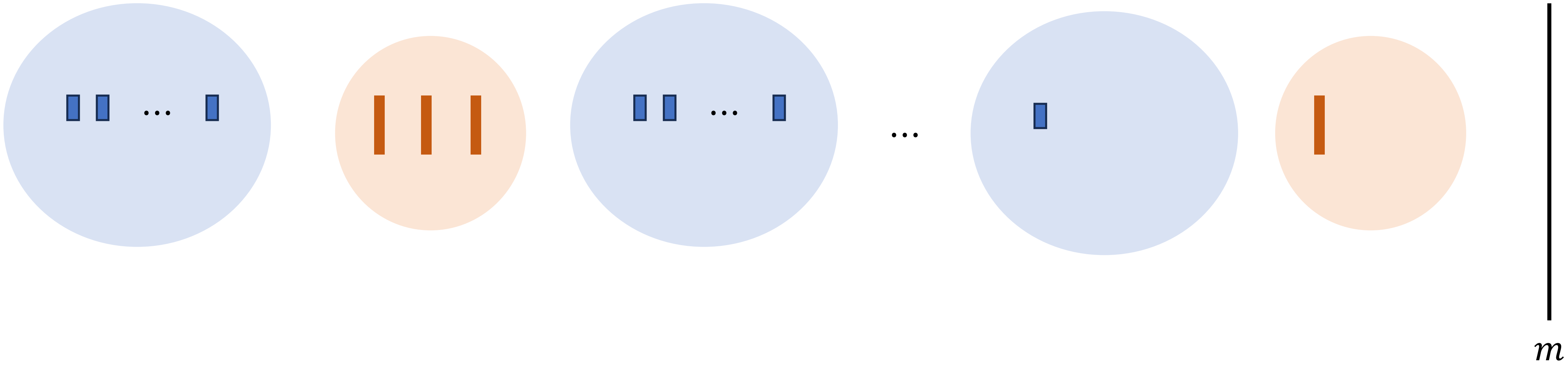
Algorithm: If a demand of size $\frac{1}{k}$ arrives, assign it to an open edge in a reserved K_k; otherwise reserve a new K_k and assign it there



Worst Case: Algorithm

- Assume $B = 1 \Rightarrow$ want to arrange demands of size $\frac{1}{k}$ in a K_k
- Assume all demand sizes are $\frac{1}{k}$ for integer k

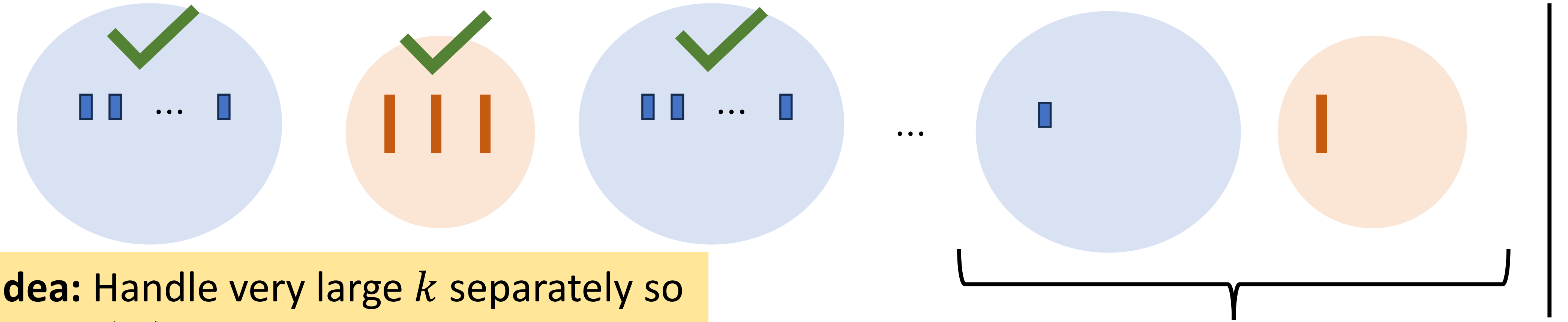
Algorithm: If a demand of size $\frac{1}{k}$ arrives, assign it to an open edge in a reserved K_k; otherwise reserve a new K_k and assign it there



Worst Case: Algorithm

- Assume $B = 1 \Rightarrow$ want to arrange demands of size $\frac{1}{k}$ in a K_k
- Assume all demand sizes are $\frac{1}{k}$ for integer k

Algorithm: If a demand of size $\frac{1}{k}$ arrives, assign it to an open edge in a reserved K_k; otherwise reserve a new K_k and assign it there



Idea: Handle very large k separately so $k \leq o(m) \Rightarrow$ waste $o(m)$ machines

Our Results

Compared to optimal offline policy that knows all demands but also assigns demands in same order until rejecting

Theorem (Worst Case): $(\frac{1}{2} - o(1))$ - competitive deterministic algorithm

Theorem (Stochastic): If all demand sizes are drawn i.i.d. from an unknown distribution, then $(1 - o(1))$ – competitive algorithm w.h.p.

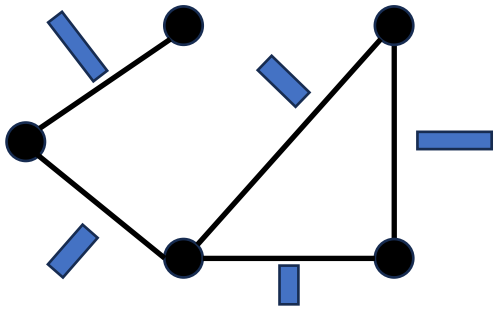
- No deterministic algorithm is better than $\frac{1}{2}$ - competitive

Stochastic Model: Idea

- Each demand size drawn i.i.d. from distribution μ

Algorithm: Suppose we already assigned first n' demands:
- Compute (near-) optimal assignment of realized demand sizes into minimum number of machines
- Use this assignment to assign the subsequent n' online arrivals

Realized first n'

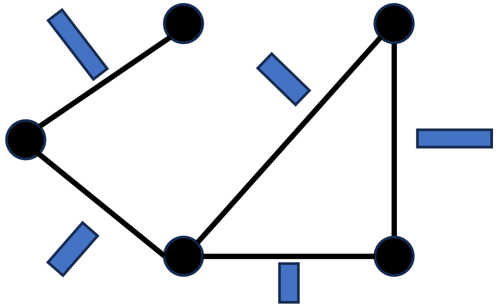


Stochastic Model: Idea

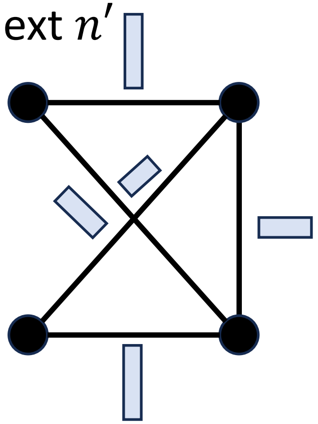
- Each demand size drawn i.i.d. from distribution μ

Algorithm: Suppose we already assigned first n' demands:
- Compute (near-) optimal assignment of realized demand sizes into minimum number of machines
- Use this assignment to assign the subsequent n' online arrivals

Realized first n'



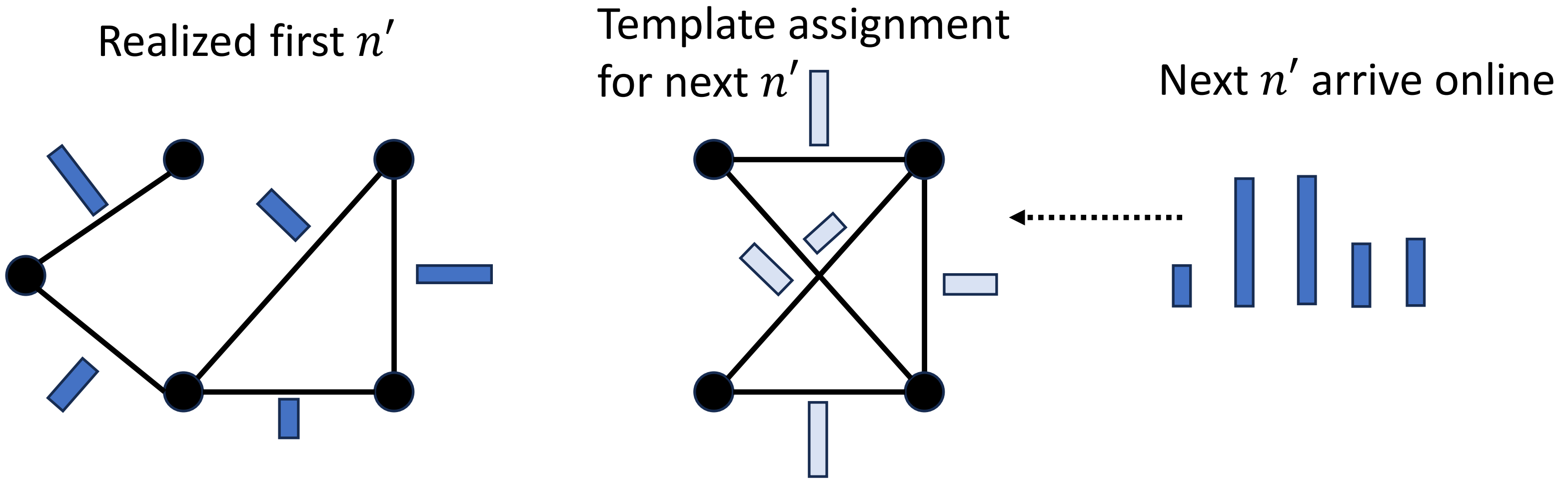
Template assignment for next n'



Stochastic Model: Idea

- Each demand size drawn i.i.d. from distribution μ

Algorithm: Suppose we already assigned first n' demands:
- Compute (near-) optimal assignment of realized demand sizes into minimum number of machines
- Use this assignment to assign the subsequent n' online arrivals



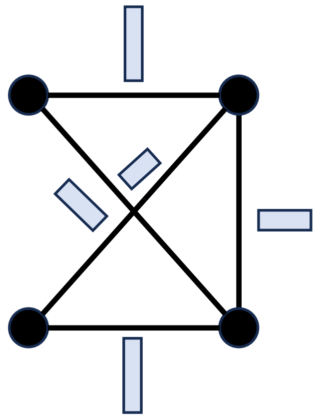
Template Assignment

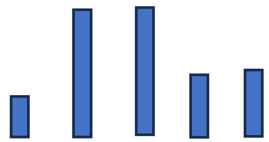
Theorem (Monotone Matching): Given sequences $X_1, \ldots, X_n$ and $Y_1, \ldots, Y_n$ (arriving online) drawn i.i.d. from the same distribution, we can compute a matching from Y's to X's such that w.h.p.:

- If Y_i is matched to X_j, then $Y_i \leq X_j$
- At most $o(n)$ of the Y's are unmatched

How to use template:

- Compute monotone matching from n' next online arrivals to the realized first n'
- If matched, then assign arrival to corresponding slot in template
- Else, assign arrival to its own separate edge





Wansoo T. Rhee, Michel Talagrand. SIAM J. Comput. 1993

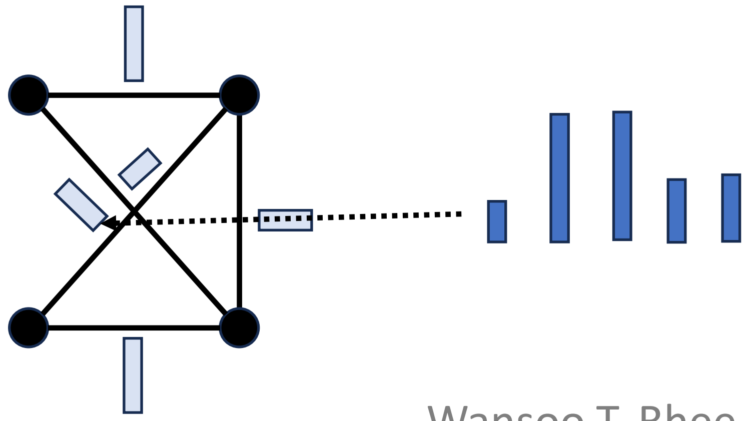
Template Assignment

Theorem (Monotone Matching): Given sequences $X_1, \ldots, X_n$ and $Y_1, \ldots, Y_n$ (arriving online) drawn i.i.d. from the same distribution, we can compute a matching from Y's to X's such that w.h.p.:

- If Y_i is matched to X_j, then $Y_i \leq X_j$
- At most $o(n)$ of the Y's are unmatched

How to use template:

- Compute monotone matching from n' next online arrivals to the realized first n'
- If matched, then assign arrival to corresponding slot in template
- Else, assign arrival to its own separate edge



Wansoo T. Rhee, Michel Talagrand. SIAM J. Comput. 1993

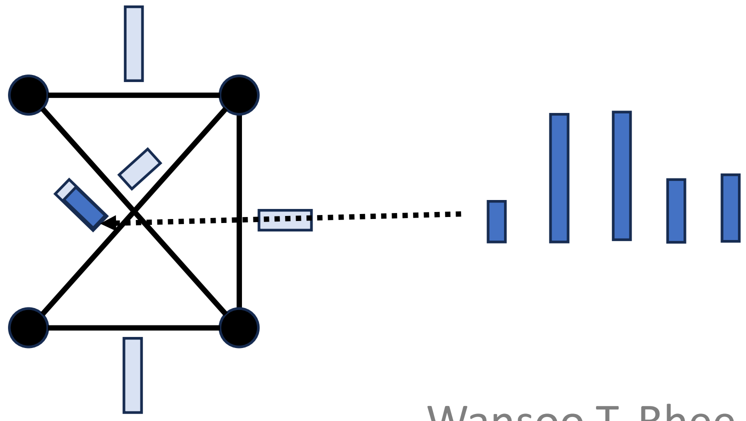
Template Assignment

Theorem (Monotone Matching): Given sequences $X_1, \ldots, X_n$ and $Y_1, \ldots, Y_n$ (arriving online) drawn i.i.d. from the same distribution, we can compute a matching from Y's to X's such that w.h.p.:

- If Y_i is matched to X_j, then $Y_i \leq X_j$
- At most $o(n)$ of the Y's are unmatched

How to use template:

- Compute monotone matching from n' next online arrivals to the realized first n'
- If matched, then assign arrival to corresponding slot in template
- Else, assign arrival to its own separate edge



Wansoo T. Rhee, Michel Talagrand. SIAM J. Comput. 1993

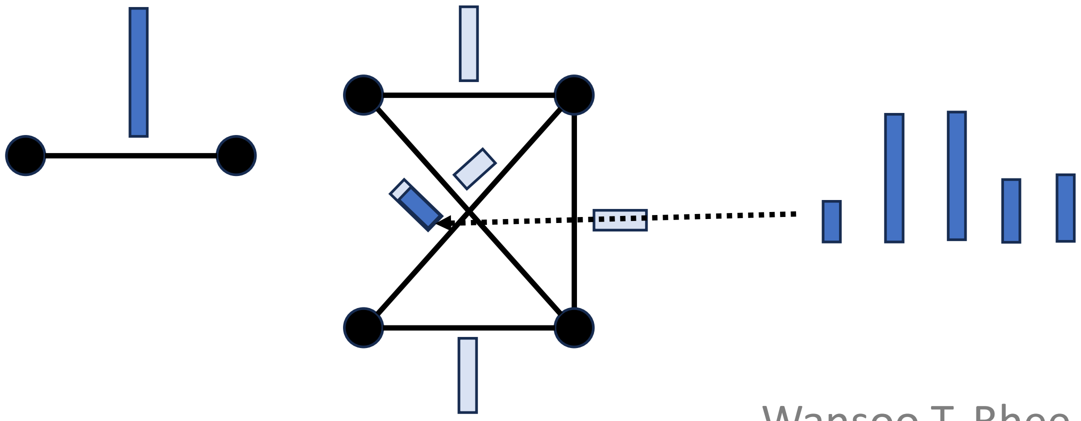
Template Assignment

Theorem (Monotone Matching): Given sequences $X_1, \dots, X_n$ and $Y_1, \dots, Y_n$ (arriving online) drawn i.i.d. from the same distribution, we can compute a matching from Y's to X's such that w.h.p.:

- If Y_i is matched to X_j, then $Y_i \leq X_j$
- At most $o(n)$ of the Y's are unmatched

How to use template:

- Compute monotone matching from n' next online arrivals to the realized first n'
- If matched, then assign arrival to corresponding slot in template
- Else, assign arrival to its own separate edge



Wansoo T. Rhee, Michel Talagrand. SIAM J. Comput. 1993

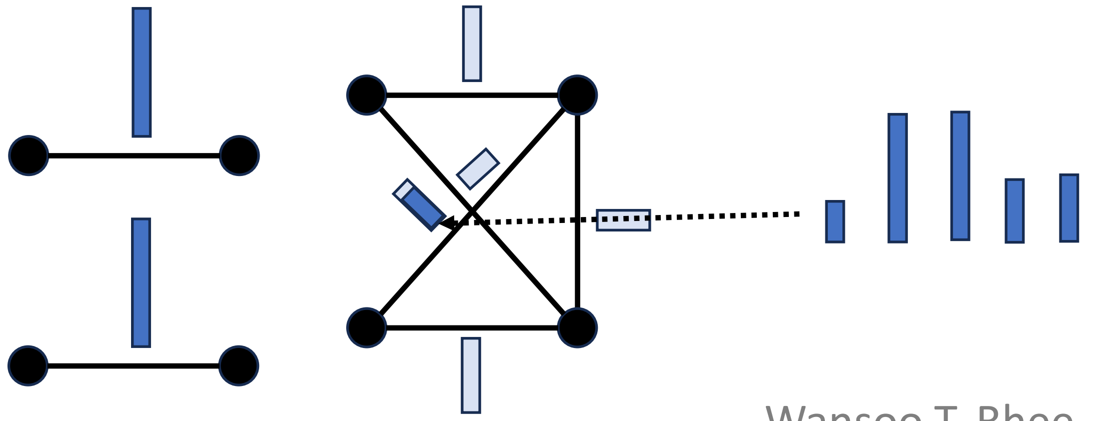
Template Assignment

Theorem (Monotone Matching): Given sequences $X_1, \dots, X_n$ and $Y_1, \dots, Y_n$ (arriving online) drawn i.i.d. from the same distribution, we can compute a matching from Y's to X's such that w.h.p.:

- If Y_i is matched to X_j, then $Y_i \leq X_j$
- At most $o(n)$ of the Y's are unmatched

How to use template:

- Compute monotone matching from n' next online arrivals to the realized first n'
- If matched, then assign arrival to corresponding slot in template
- Else, assign arrival to its own separate edge



Wansoo T. Rhee, Michel Talagrand. SIAM J. Comput. 1993

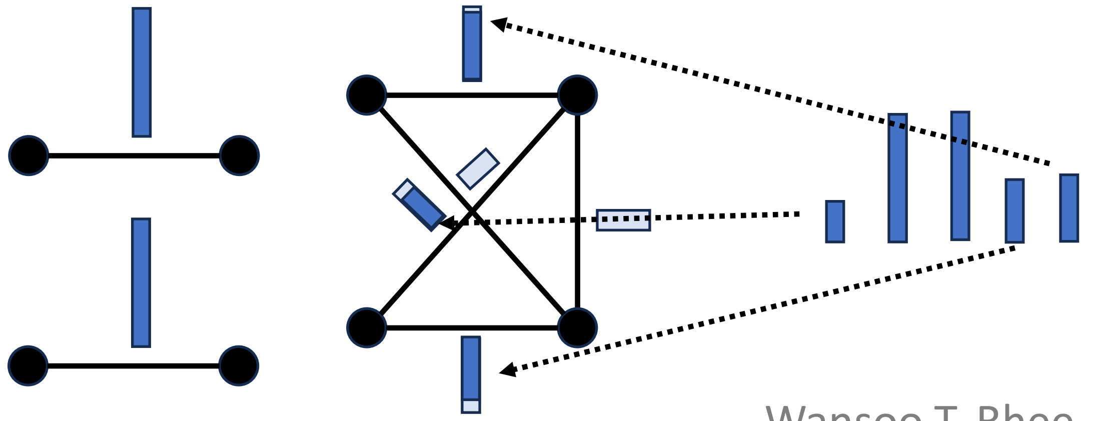
Template Assignment

Theorem (Monotone Matching): Given sequences $X_1, \dots, X_n$ and $Y_1, \dots, Y_n$ (arriving online) drawn i.i.d. from the same distribution, we can compute a matching from Y's to X's such that w.h.p.:

- If Y_i is matched to X_j, then $Y_i \leq X_j$
- At most $o(n)$ of the Y's are unmatched

How to use template:

- Compute monotone matching from n' next online arrivals to the realized first n'
- If matched, then assign arrival to corresponding slot in template
- Else, assign arrival to its own separate edge



Wansoo T. Rhee, Michel Talagrand. SIAM J. Comput. 1993

Conclusion

- Introduce Online Demand Scheduling with Failover
- **Worst Case**
  - Competitive ratio $\rightarrow \frac{1}{2}$ as $m \rightarrow \infty$
  - Tight lower bound
  - Reserve cliques for different sizes
- **Stochastic i.i.d.:**
  - Competitive ratio $\rightarrow 1$ as $m \rightarrow \infty$
  - Learn from past arrivals using template assignments
  - Monotone matching theorem