Mace, J. *et al.*, "2DFQ: Two-Dimensional Fair Queuing for Multi-Tenant Cloud Services," *Proc. of ACM SIGCOMM '16*, 46(4):144-159, Aug. 2016.

Allison McDonald
Nitish Paradkar

Introduction

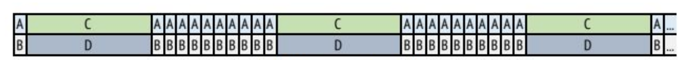
- Server has limited capacity
- Requests by clients are queued
- Crucial to provide resource isolation to ensure that a single tenant cannot get more than its fair share of resources

Differences From Traditional Queueing

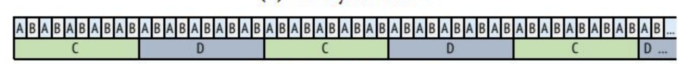
- Resource concurrency: threads in shared process handle requests concurrently
- Large cost variance: request costs vary by at least 10000.
- Unknown and unpredictable resource costs: Unknown at schedule time and difficult to estimate
    - Wrong estimates are penalized heavily
    - Network schedulers rely on this information

Bursty Schedules from WFQ and WF^{2}Q

- Both methods below observe long term fairness
- Goal is to achieve smoother schedules and achieve fairness in a smaller time scale



(a) Bursty schedule

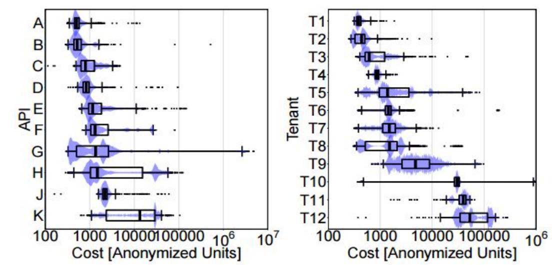


(b) Smooth schedule

Key Insights

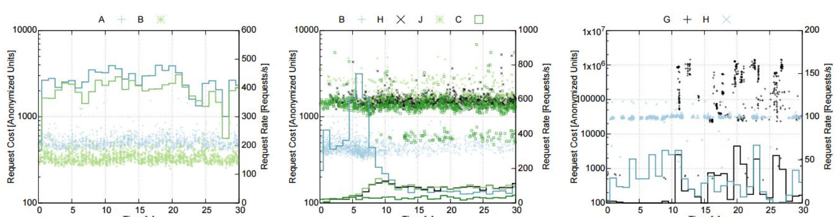
- Separate requests with different costs across different worker threads
- Use cost estimation to locate unpredictable requests and separate them from predictable requests
- 2D = across both time and different threads
- Desired traits
  - Work conserving
  - Achieve fairness over shorter time periods

High Request Cost Variability



- Costs can vary by 4 orders of magnitude
- Bursty scheduling adversely affects tenants with small requests
  - Get serviced in high throughput bursts than evenly paced over time

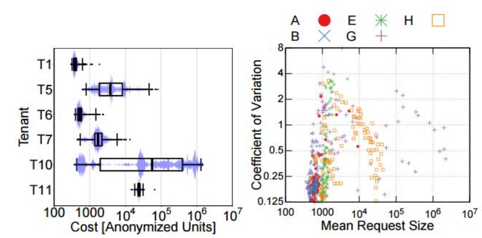
Unknown Request Costs



(a) Stable tenant (T_2) (b) Stable with gradual changes (T_3) (c) Unstable tenant with frequent changes (T_{10})

Request rates can be unpredictable and vary by around 1.5 orders of magnitude

Unknown Request Costs



- Each API has tenants using it in both stable and unpredictable ways
- Can use request cost estimations, but these fall apart for more unpredictable tenants

Unknown Request Costs

- Incorrect cost estimates can lead to bursty schedules
- Expensive request gets predicted as an inexpensive request
  - Blocks worker thread for longer than expected
  - Scheduler can incorrectly schedule up to N (number of threads) requests
- Insight: give good service to predictable tenants

WFQ

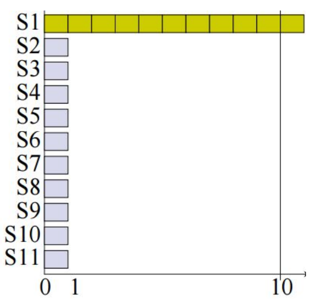
$$S(r_f^j) = \max\{v(A(r_f^j)), F(r_f^{j-1})\}$$

$$F(r_f^j) = S(r_f^j) + l_f^j / \Phi_f$$

- $A(r_f^j)$ = walltime arrival time of packet
- $F(r_f^j)$ = virtual finish time
- l_f^j = size of request
- Φ_f = weight of the flow

Worst Case Weighted Fair Queueing (WF^{2}Q)

- WFQ might get ahead of GPS

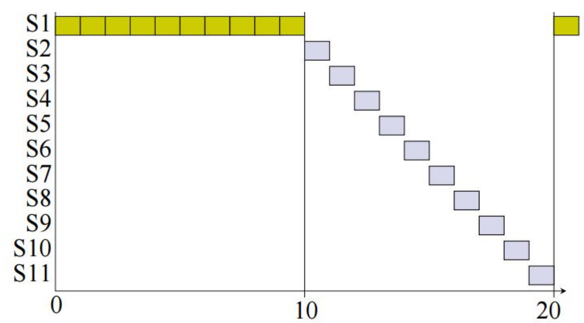


$W_1 = 0.5$
$W_2 = W_3 = \ldots = W_{11} = 0.05$

Each packet is of length 1

http://www.eng.tau.ac.il/~boaz/comnet/lec08.pdf

Worst Case Weighted Fair Queueing (WF^{2}Q)

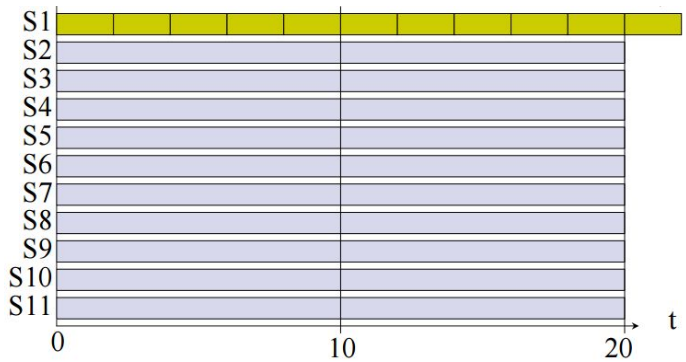


$F_1 = 2, 4, 6, 8, 10, \ldots$

$F_2 = F_3 = \ldots = F_{11} = 20$

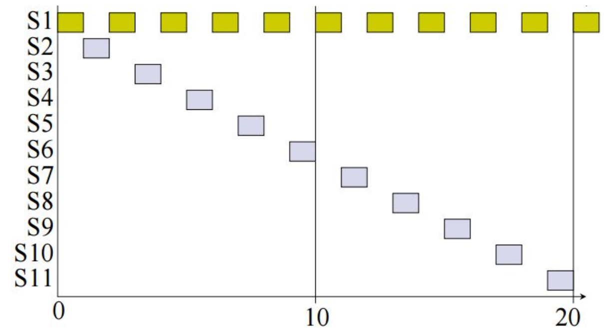
http://www.eng.tau.ac.il/~boaz/comnet/lec08.pdf

Worst Case Weighted Fair Queueing (WF^{2}Q)



http://www.eng.tau.ac.il/~boaz/comnet/lec08.pdf

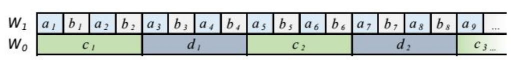
Worst Case Weighted Fair Queueing (WF^{2}Q)



Consider a request to be eligible only when it would have begun service in GPS

http://www.eng.tau.ac.il/~boaz/comnet/lec08.pdf

Two-Dimensional Fair Queueing (known costs)



(a) Ideal request schedule over time on two threads

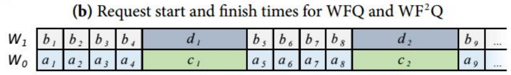
Request	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	
Start Time	0	1	2	3	4	5	6	7	8	...
Finish Time	1	2	3	4	5	6	7	8	9	

Request	c_1	c_2	c_3	
Start Time	0	4	8	...
Finish Time	4	8	12	

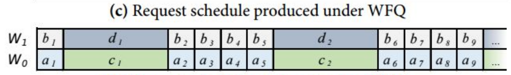
Request	b_1	b_2	b_3	b_4	b_5	b_6	b_7	b_8	b_9	
Start Time	0	1	2	3	4	5	6	7	8	...
Finish Time	1	2	3	4	5	6	7	8	9	

Request	d_1	d_2	d_3	
Start Time	0	4	8	...
Finish Time	4	8	12	

(b) Request start and finish times for WFQ and WF^{2}Q



(c) Request schedule produced under WFQ



(d) Request schedule produced under WF^{2}Q

Two-Dimensional Fair Queueing (known costs)

- WF^{2}Q allows burstiness when multiple worker threads are free and only large requests are available.
- Small requests eligible for either all threads or no threads.
- New eligibility condition on thread i: $S(r_j) - (i / n)*l_j$ where $0 \le i < n$.
- Small requests become eligible on high-index threads first and tend to be serviced before low-indexed threads can service them.

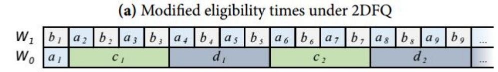
Request	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	
Eligible W_0	0	1	2	3	4	5	6	7	8	...
Eligible W_1	-0.5	0.5	1.5	2.5	3.5	4.5	5.5	6.5	7.5	

Request	c_1	c_2	c_3	
Eligible W_0	0	4	8	...
Eligible W_1	-2	2	6	

Request	b_1	b_2	b_3	b_4	b_5	b_6	b_7	b_8	b_9	
Eligible W_0	0	1	2	3	4	5	6	7	8	...
Eligible W_1	-0.5	0.5	1.5	2.5	3.5	4.5	5.5	6.5	7.5	

Request	d_1	d_2	d_3	
Eligible W_0	0	4	8	...
Eligible W_1	-2	2	6	

(a) Modified eligibility times under 2DFQ



(b) Request schedule produced under 2DFQ.

Extended Two-Dimensional Fair Queuing ($2DFQ^E$)

Pessimistic Cost Estimation

- Safer to estimate a cheap request as inexpensive than an expensive request as cheap
- Treat unpredictable tenants as expensive
- L^i_{max} = cost of largest request
- c_r = true cost of just-completed request
- If $c_r > L^i_{max}$, set $L^i_{max} = c_r$
- Otherwise, set $L^i_{max} = \alpha L^i_{max}$, where $\alpha < 1$ (but close to 1)

Extended Two-Dimensional Fair Queuing ($2DFQ^E$)

Bookkeeping: Retroactive Charging

- l_r is the estimated cost of a request
- c_r is the actual cost of a request
- Upon completion of a request, $c_r - l_r$ is incorporated into the virtual start and finish times of a tenant
- This ensures long-run fairness

Extended Two-Dimensional Fair Queuing ($2DFQ^E$)

Bookkeeping: Refresh Charging

- If a tenant transitions from many small requests to many expensive requests, L^i_{max} will be underestimated at first
- If a request is especially expensive, up to N worker threads could begin processing expensive requests before the value of L^i_{max} is updated
- Refresh Charging periodically checks long-running requests and updates L^i_{max} while expensive request is still being processed
- Non-negligible overhead — optimal at 10ms

Evaluation of 2DFQ

- Discrete event simulator with synthetic workloads and traces from Azure Storage
- Comparing 2DFQ, WFQ and WF^2Q
  - SFQ, MSF^2, and DRR were also used in evaluation but omitted because their improvements minimally influence fairness bounds

(With slides adapted from author's SIGCOMM talk: http://conferences.sigcomm.org/sigcomm/2016/files/program/sigcomm/Session04-Paper01-2DFQ-Jonathan-Slides.pdf)

Evaluation Metrics

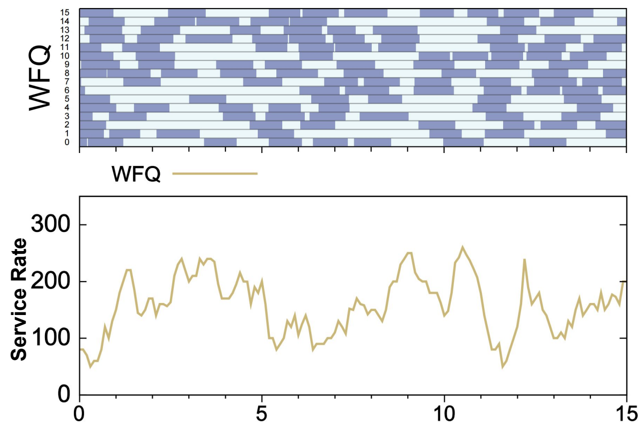
Service lag: difference between service a tenant should have received with GPU (Nr, where N = number of threads and r = processing rate) and actual work done

Service rate: work done per 100ms

Latency: time between request being enqueued and finishing processing

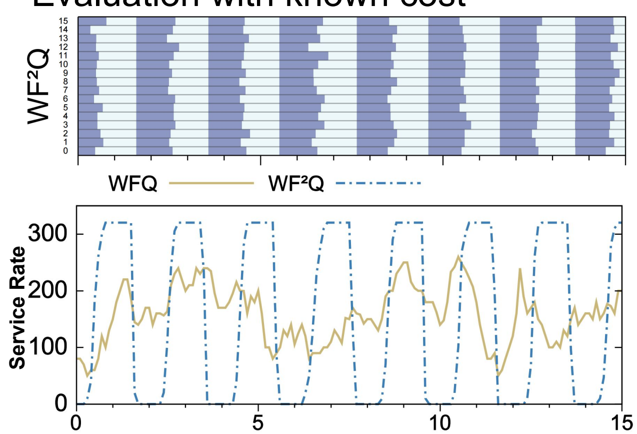
Gini index: instantaneous measure of scheduling fairness

Evaluation with known cost



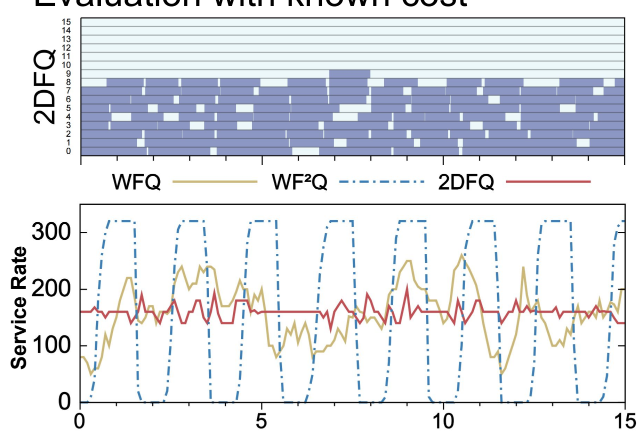
Synthetic data
Costs known
16 threads
1000 units/second

Evaluation with known cost



Synthetic data
Costs known
16 threads
1000 units/second

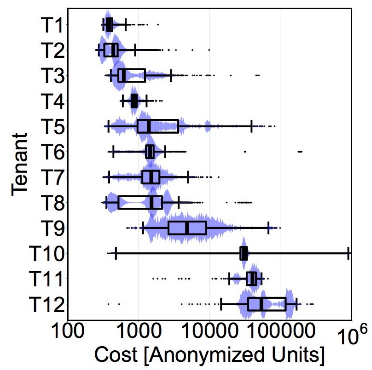
Evaluation with known cost



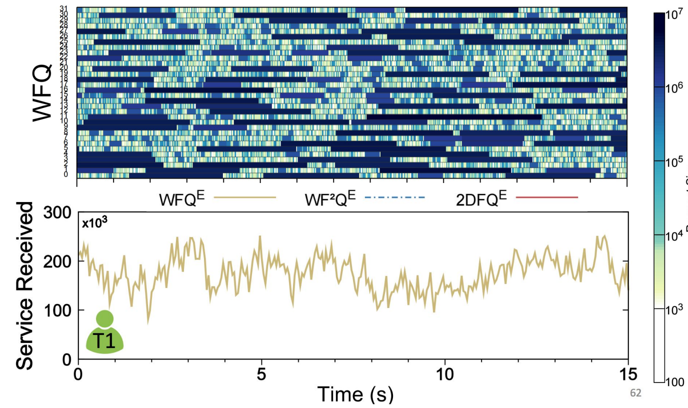
Synthetic data
Costs known
16 threads
1000 units/second

Evaluation with known cost

- Using Azure production traces
  - 250 randomly chosen tenants from 50 servers, plus $T_1 \ldots T_{12}$
- Evaluate the effects on T_1, a tenant with low request costs and low unpredictability

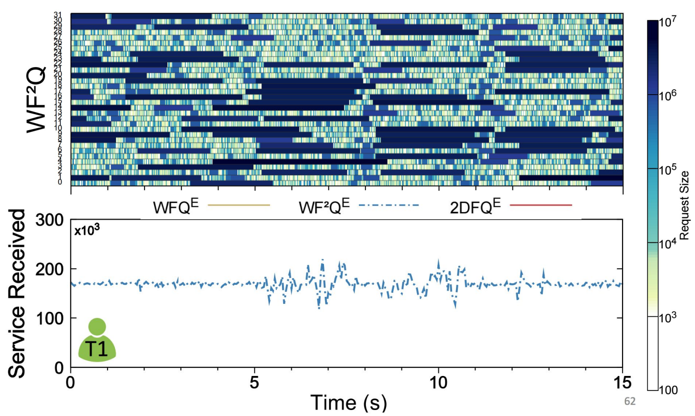


Evaluation with known cost



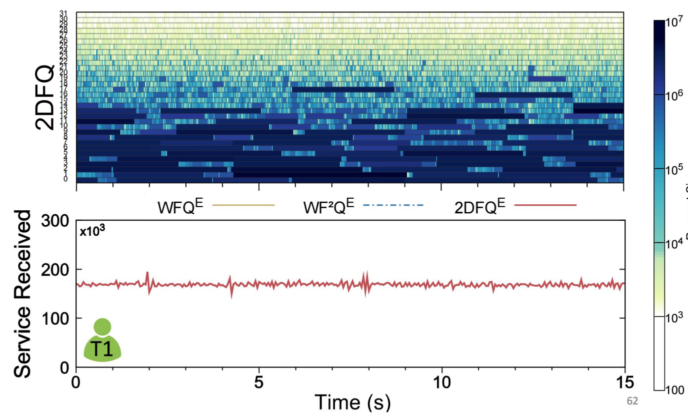
250 Azure workloads
Costs known
32 threads
1 million units/second

Evaluation with known cost



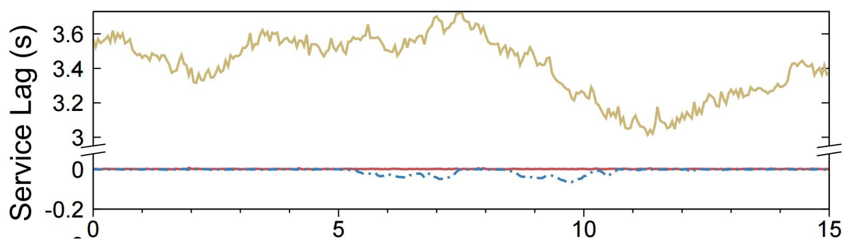
250 Azure workloads
Costs known
32 threads
1 million units/second

Evaluation with known cost

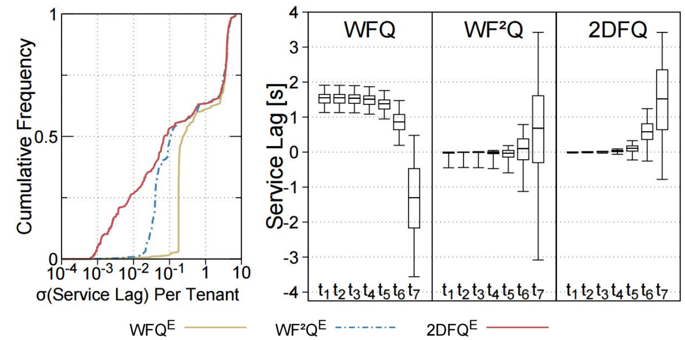


250 Azure workloads
Costs known
32 threads
1 million units/second

Evaluation with known cost



Evaluation with known cost



Where $t_1 \ldots t_7$ are fixed-cost tenants submitting requests of size 2^8, 2^{10}, 2^{12}, …, 2^{20}

Evaluation with unknown costs

- Using $2DFQ^E$ with α = 0.99
- Added refresh and retroactive bookkeeping to WFQ and WF^2Q to create WFQ^E and WF^2Q^E

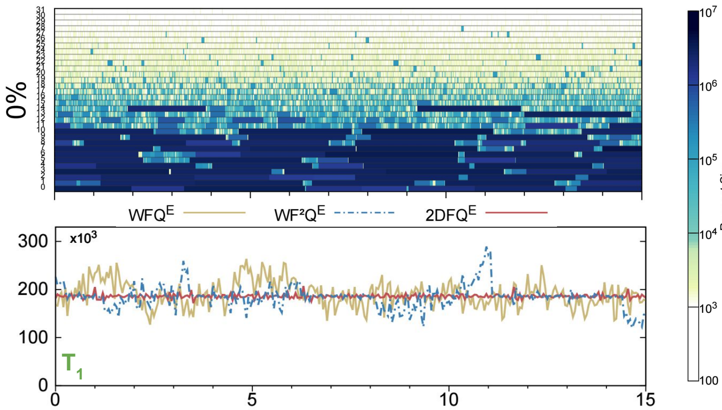
Evaluation with unknown costs

- 300 randomly selected tenants from Azure data
- Added unpredictability by sampling from all Azure data without regard for server or account

3 Experiments:

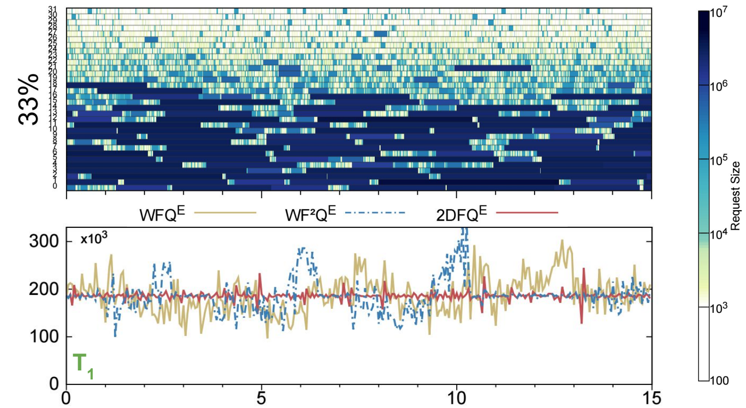
- 0% unpredictable: Only real tenant data
- 33% unpredictable: 33% arbitrary sampling
- 66% unpredictable: 66% arbitrary sampling

Evaluation with unknown costs



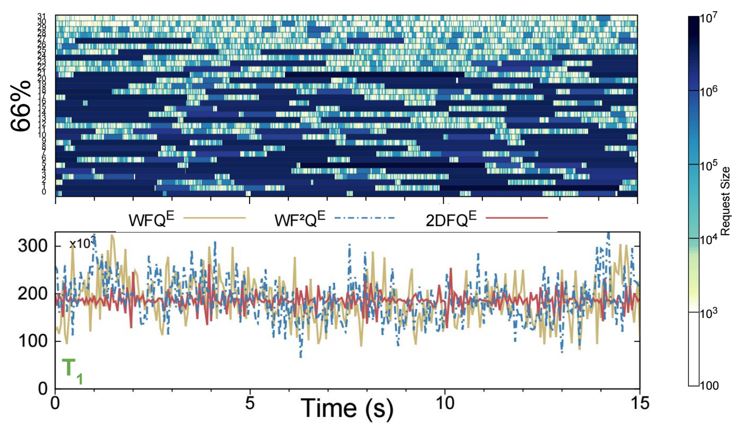
33

Evaluation with unknown costs



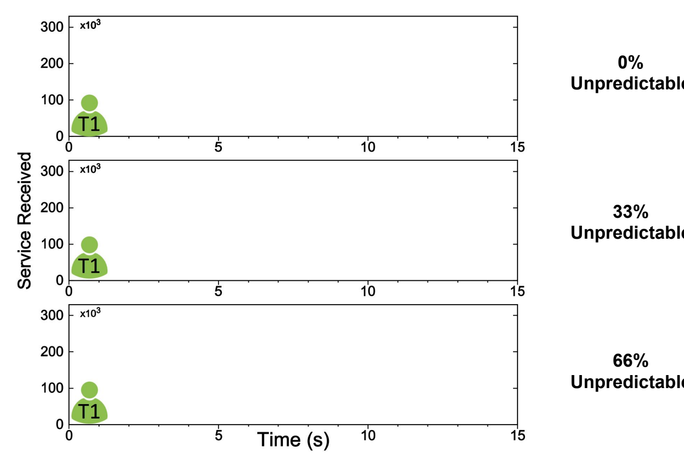
34

Evaluation with unknown costs



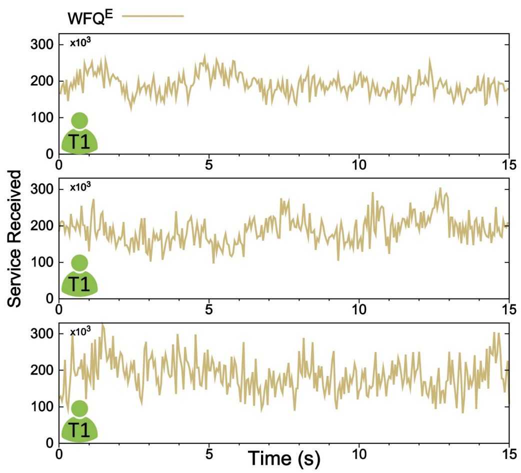
35

Unknown



36

Unknown

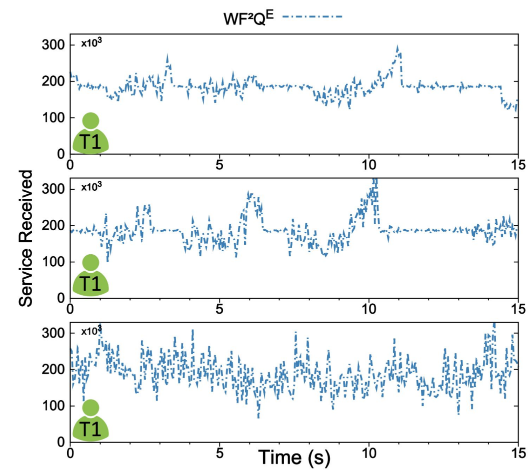


0% Unpredictable

33% Unpredictable

66% Unpredictable

Unknown

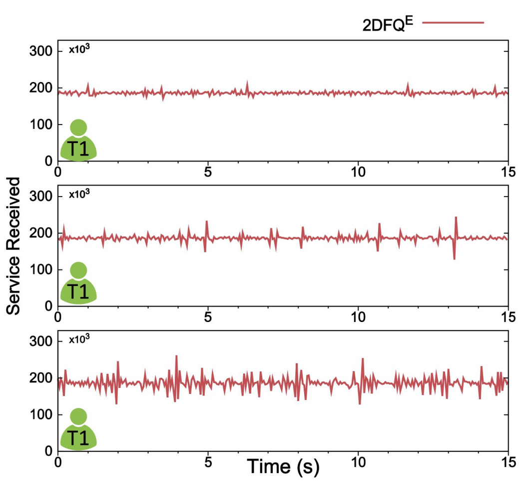


0% Unpredictable

33% Unpredictable

66% Unpredictable

Unknown

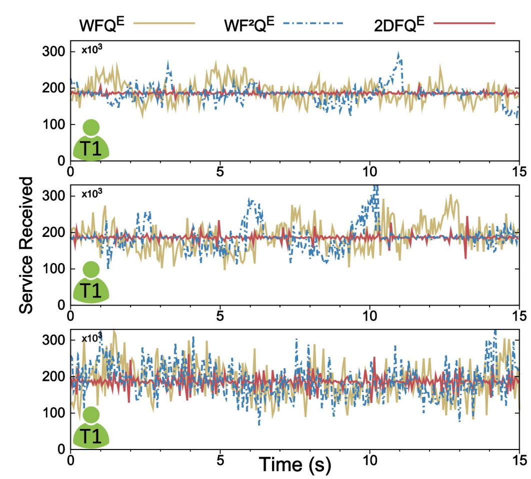


0% Unpredictable

33% Unpredictable

66% Unpredictable

Unknown



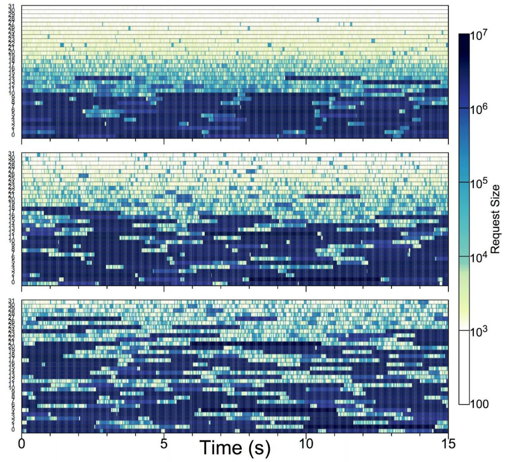
0% Unpredictable

33% Unpredictable

66% Unpredictable

Unknown

2DFQ^E

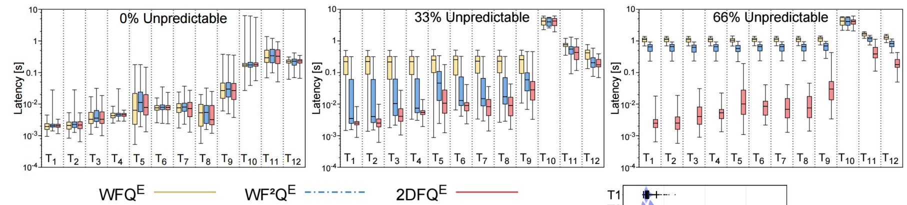


0% Unpredictable

33% Unpredictable

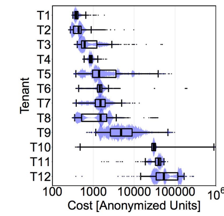
66% Unpredictable

Evaluation with unknown costs

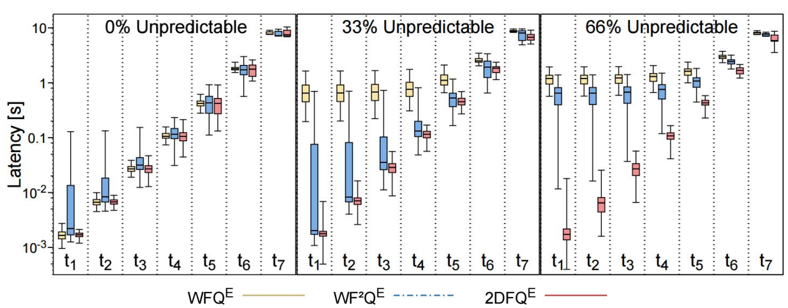


Latencies of smaller tenants ($T_1 \ldots T_4$) are less impacted by unpredictable tenants

Latencies of largely unpredictable tenants (T_{10}) are unaffected

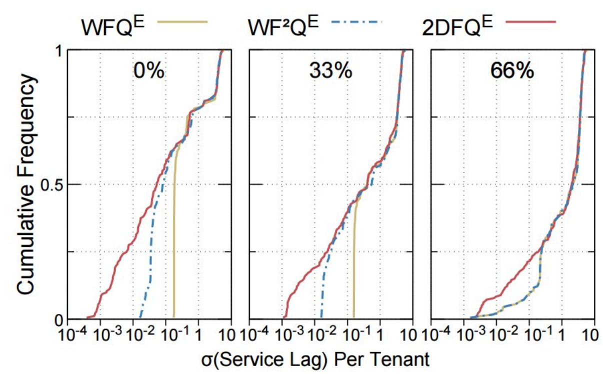


Evaluation with unknown costs



Where $t_1 \ldots t_7$ are fixed-cost tenants submitting requests of size 2^8, 2^{10}, 2^{12}, …, 2^{20}

Evaluation with unknown costs



Evaluation with unknown cost: production workloads

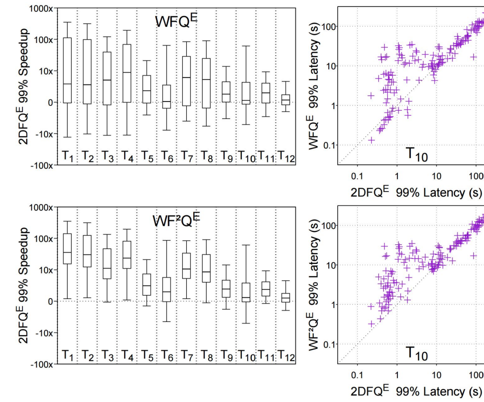
- 150 experiments from Azure data
- Randomly vary several parameters:
  - Number of worker threads (2-64)
  - Number of tenants (0-400)
  - Replay speed (0.5-4x)
  - Number of backlogged tenants (0-100)
  - Number of artificially expensive tenants (0-100)
  - Number of unpredictable tenants (0-100)

Evaluation with unknown cost: production workloads

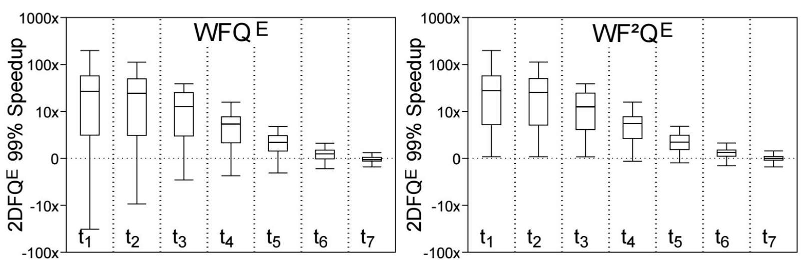
99th percentile latency speedups

For T_1, median improvement over WFQ^E of 3.8x and 142x over WF^2Q^E

T_{10} latencies were usually unimproved, but when they were it was by a large factor

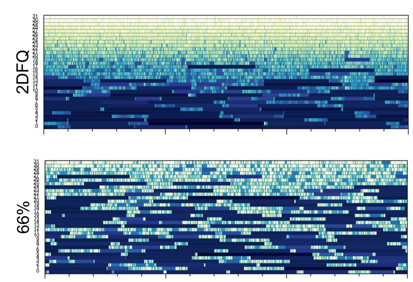


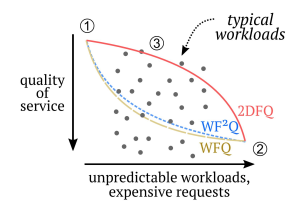
Evaluation with unknown cost: production workloads



Where $t_1 \ldots t_7$ are fixed-cost tenants submitting requests of size 2^8, 2^{10}, 2^{12}, …, 2^{20}

Summary





Discussion

- Analyze the tradeoffs between how aggressively tenants with unpredictable costs and separated from predictable costs
- Can take a very long time to classify a previously expensive thread as a cheap one
  - Try changing alpha parameter to be dynamic
- Benefits of keeping system work-conserving
  - Always keep a set of threads dedicated to only serving inexpensive, predictable requests