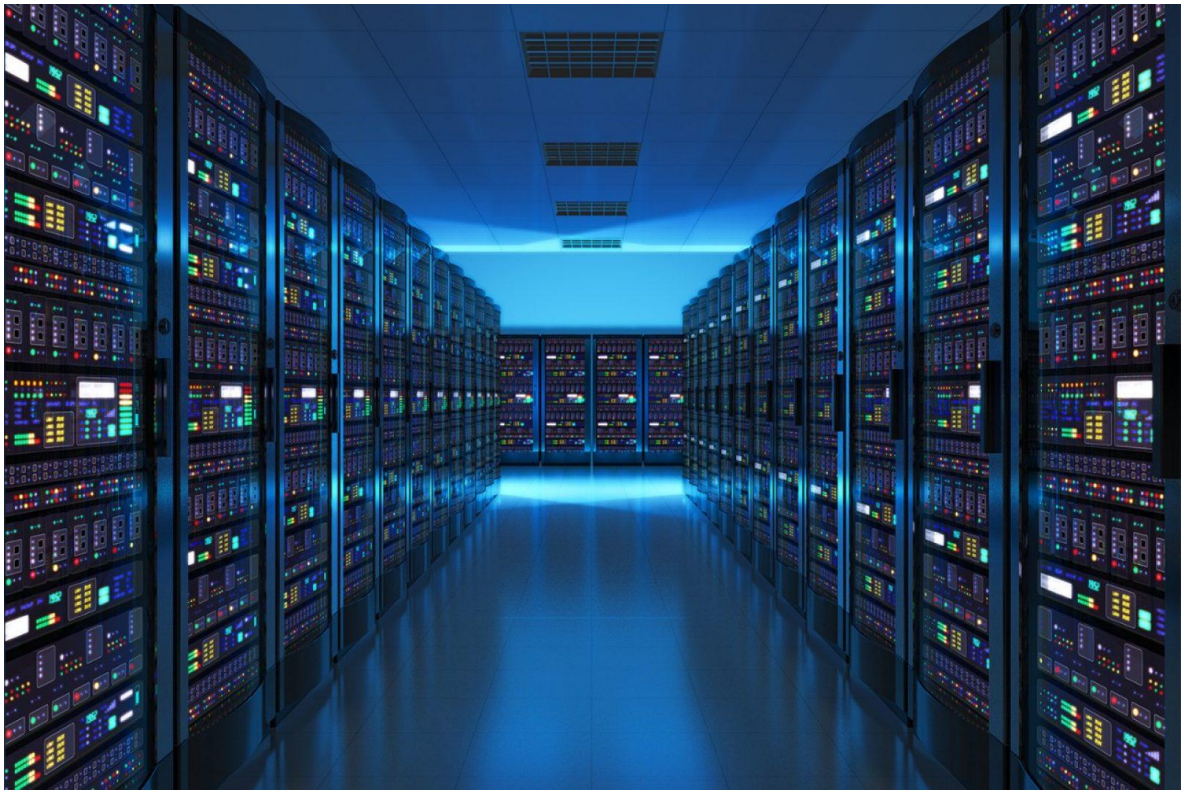
Fair-EDF: A Latency Fairness Framework for Shared Storage Systems

Yuhan Peng[1], Peter Varman[2]
Department of Computer Science[1]
Department of Electrical and Computer Engineering[2]
Rice University

11th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage '19), Renton, WA

Clustered Storage Systems



ceph

GlusterFS

Powered By vmware vSAN™

amazon web services™

Latency Support

- Guaranteeing the response time
  - Meet explicitly-specified QoS latency bounds
  - Implicitly optimize average/tail latencies

- Request scheduling using metadata (tags)
  - Needs enough server capacity to ensure latencies
    - Admission control: overload avoidance or detection

Overload Avoidance Approaches

- Regulate input request traffic using token bucket
  - Token bucket parameters based on QoS requirements
  - pClock (SIGMETRICS PER '07), Nested QoS (Intel Tech Journal '12), PriorityMeister (SoCC '14)

- Requests delayed to conform to client's QoS input rate
  - Clients which exceed their input rate incur high latency
  - Cascading delays can cause excessive latency violations

Overload Detection Approaches

- RT-OPT (CASES Workshop ‘99)
  - Offline algorithm for streaming videos
  - Drops minimum number of overloaded requests
  - All taken requests’ deadlines are guaranteed

- MittOS (SOSP ’17)
  - Predicts whether a request can meet its deadline
  - System support for dropping or redirecting requests

Latency Support Difficulties

- Admission control in a distributed context is difficult
  - Aggregate traffic control may be feasible
  - Run-time variability makes per server control hard
- Requires accurate server workload estimation
  - Inaccurate estimation may cause cascading deadline misses
  - Needs significant capacity overprovisioning

This Paper

- Detecting and resolving server overloading by dropping requests
- Requests chosen to
  - Minimize number of dropped requests
  - Shape client QoS (% deadline violations)
- Assumes OS support like MittOS to support the dropped requests

EDF Scheduler

- Low implementation overhead
- Deadlines guaranteed if capacity is enough
- Limitations
  - EDF cannot guarantee deadlines when overloaded
  - The delays aggregate when overloaded
  - Does not provide QoS

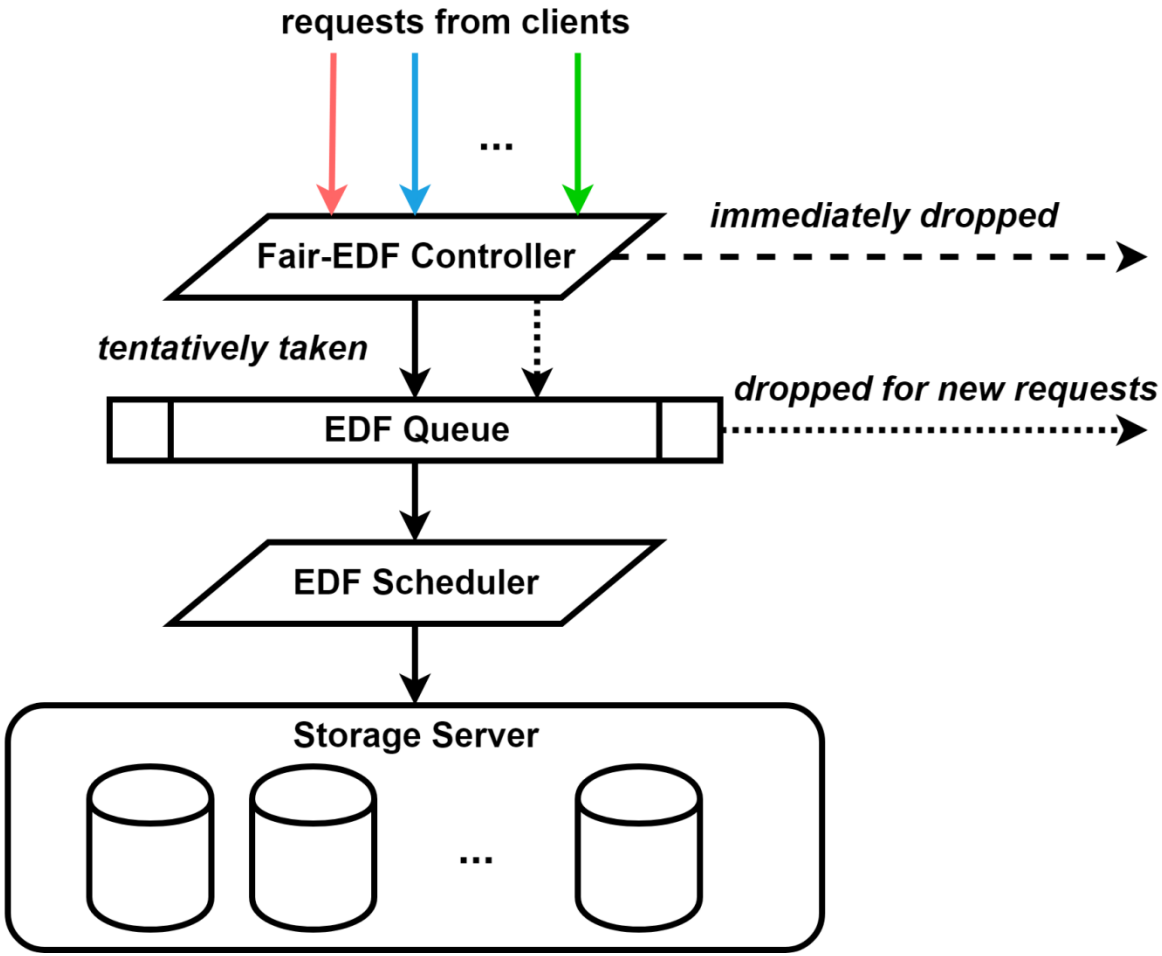
Problem Statement

- Each request r
  - Arrival time t_r
  - Deadline d_r
  - Service time σ
- Successful: r completes no later than d_r
- Wasted:
  - r is dropped (not scheduled)
  - r completes after d_r

Problem Statement

- Client i's success ratio f_i
  - The fraction of Client i's requests that succeed
- System success ratio F
  - The fraction of the total requests that succeed
- Goal: min-max criterion
  - Maximizing F
  - Equalizing f_i, i.e. maximize the lowest client success ratio

Fair-EDF Framework



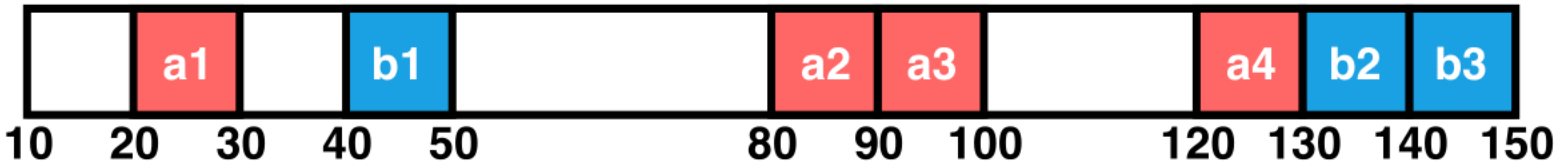
Occupancy Chart

- Maintains the current set of taken requests
  - Detects overload condition
  - Provides information for which request to drop

- Each request r occupies one time slot of length σ
  - Indicates the latest time to dispatch r to ensure all requests with deadlines later than d_r are successful
- Partitioned into busy and idle intervals
  - If no interval cross the current time T_{now}, the server is not overloaded, i.e. all requests' deadlines will be met

Occupancy Chart: Illustration 1

- $\sigma = 10$
- $T_{now} = 10$

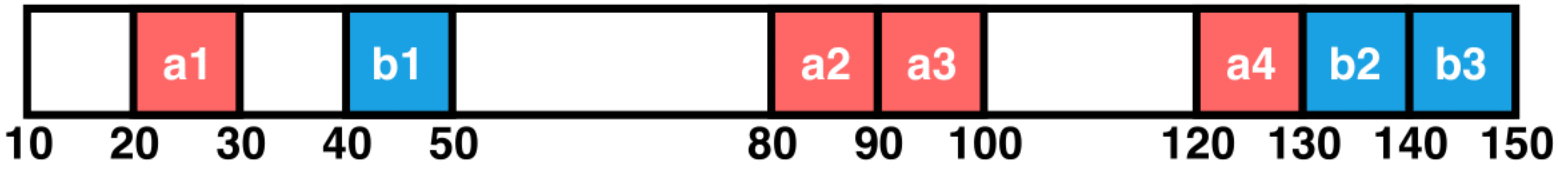


Occupancy Chart: Handling New Request

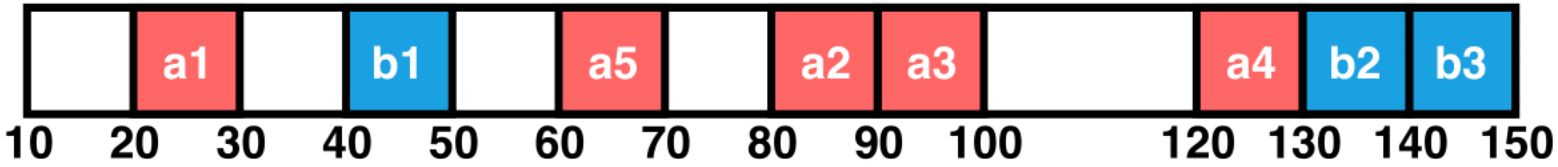
- Step 1: update the occupancy chart
- Locate d_r in the occupancy chart
  - If d_r is in a idle interval: insert a new interval for r
  - If d_r is in a busy interval: add r to the interval and reduce the left bound of that interval by σ
- Merge the overlapping intervals

Occupancy Chart: Illustration 2

- Before

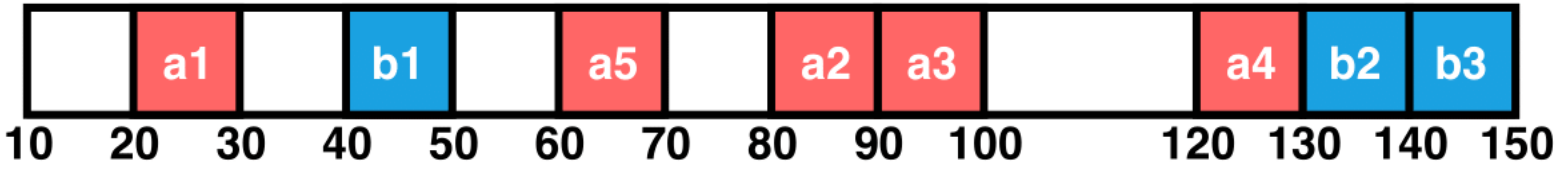


- A new request a_5 arrives with deadline 70
  - Create and insert the interval [60, 70]

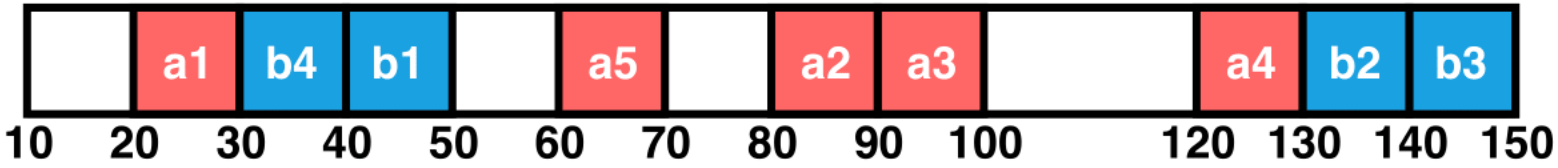


Occupancy Chart: Illustration 3

- Before



- A new request b_4 arrives with deadline 45
  - Left shift the interval [40, 50] by 10
  - Resulting interval [30, 50]
    - Needs a merge since overlaps with [20, 30]

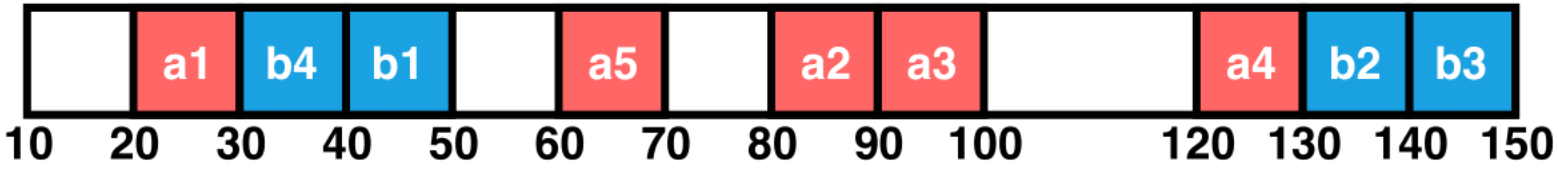


Occupancy Chart: Handling New Request

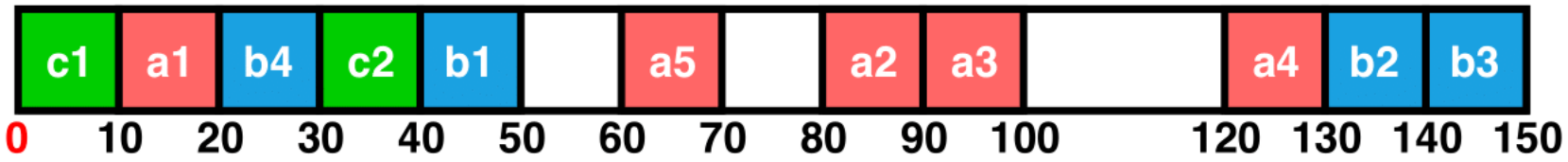
- Step 2: drop an overloaded request, if necessary
- The occupancy chart may become overloaded after adding the new request
  - The first interval $[L_0, R_0]$ has $L_0 < T_{now}$
  - Drop one from the first interval, based on QoS
    - Drop a request from the candidate set that belongs to a client with the highest success ratio

Occupancy Chart: Illustration 4

- Before



- Two requests c_1 and c_2 arrive with deadlines 20 and 45



- Drop one from $\{a_1, b_1, b_4, c_1, c_2\}$

Performance Evaluation

- Implemented Fair-EDF in OpenMP
- Run experiments on a Linux server
  - Hard disk: Intel® SSD DC S3700
  - CPU: Intel® Xeon® E5-2697
- Random uncached reads from a large file
  - The server has a profiled capacity of 7300 IOPS
  - We use $\sigma = 137\mu s$

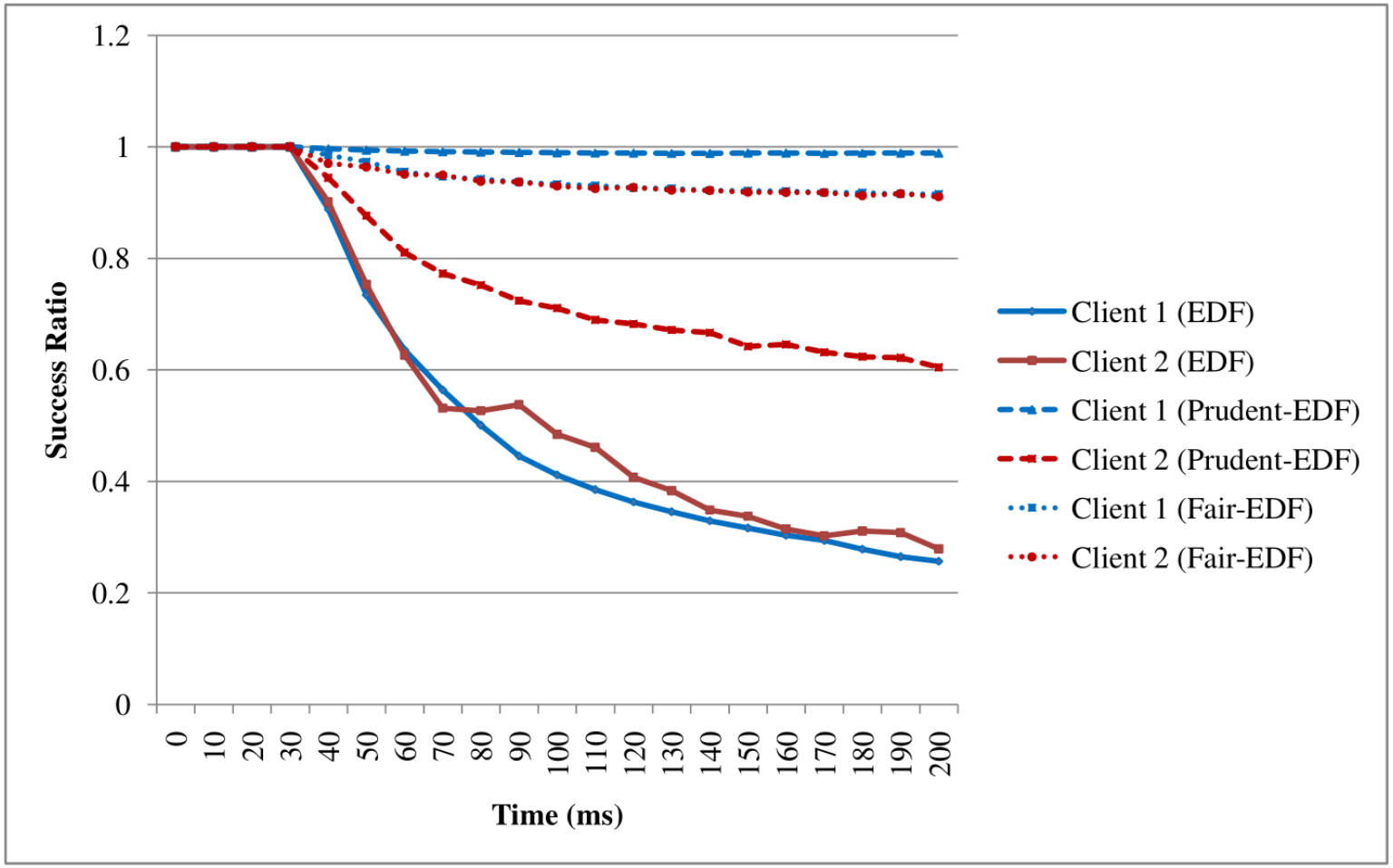
Evaluation Methodology

- Compare Fair-EDF against (normal) EDF and Prudent-EDF
  - Prudent-EDF seeks requests in EDF order
  - Prudent-EDF drops requests with deadline already missed
  - Prudent-EDF drops minimum number of request possible, but cannot shape the QoS profiles

Experiment 1: 2 Clients

- Configuration:
  - Client 1 sends one request every 0.15ms
    - 6667 IOPS
  - Client 2 sends a burst of 15 requests every 10ms
    - 1500 IOPS
  - Total load 8167 IOPS > server capacity 7300 IOPS
    - System success ratio can be at most 0.89
  - Client 1 requests' latency bound = 0.5ms
  - Client 2 requests' latency bound = 25ms

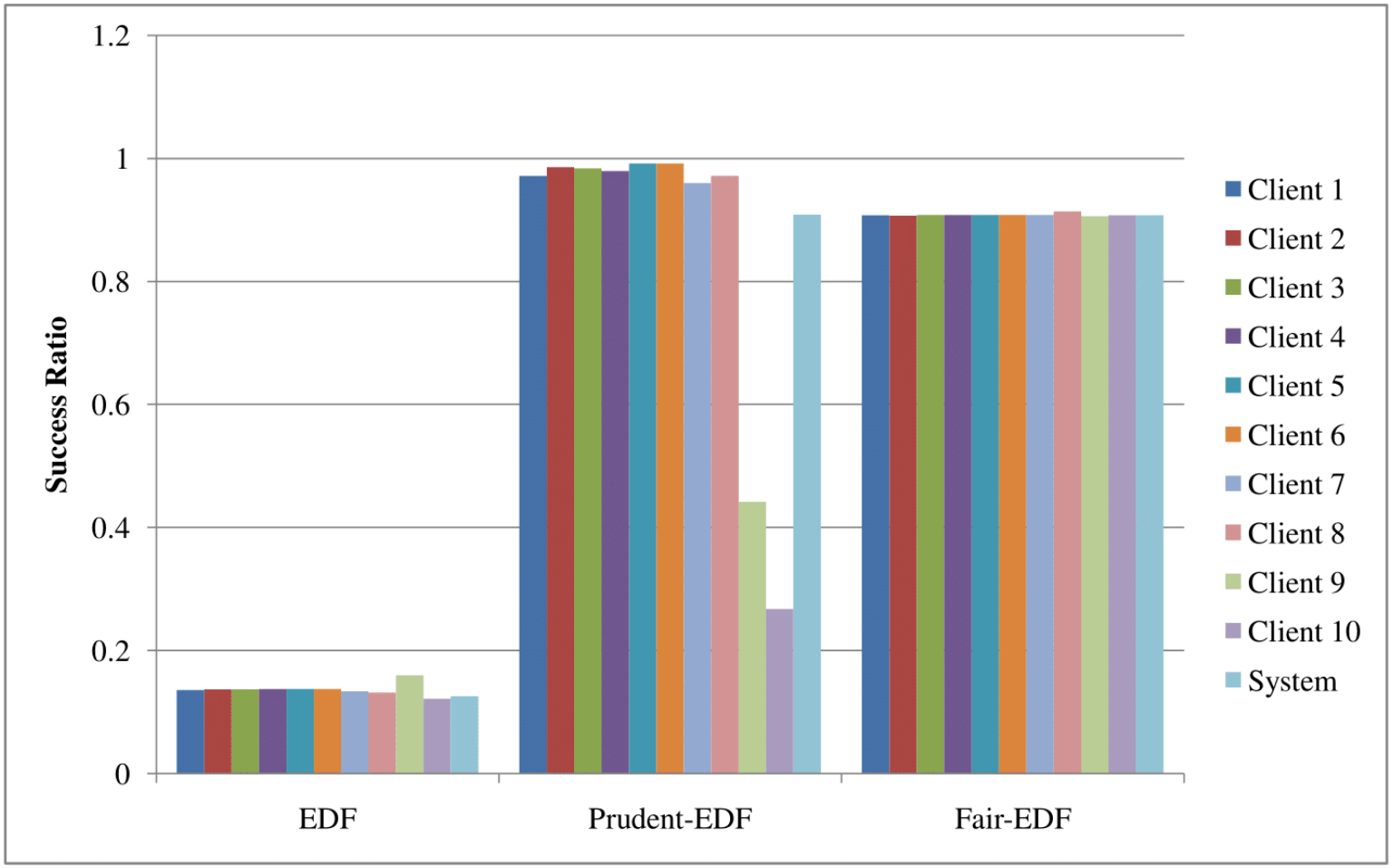
Experiment 1: Success Ratio For Both Clients



Experiment 2: 10 Clients

- Configuration:
  - Clients 1 ~ 8 send requests with different fixed rates
    - Smaller latency bound
  - Client 9, 10 send requests in bursty manners
    - Higher latency bound
  - Total load 8000 IOPS > server capacity 7300 IOPS
    - System success ratio can be at most 0.91

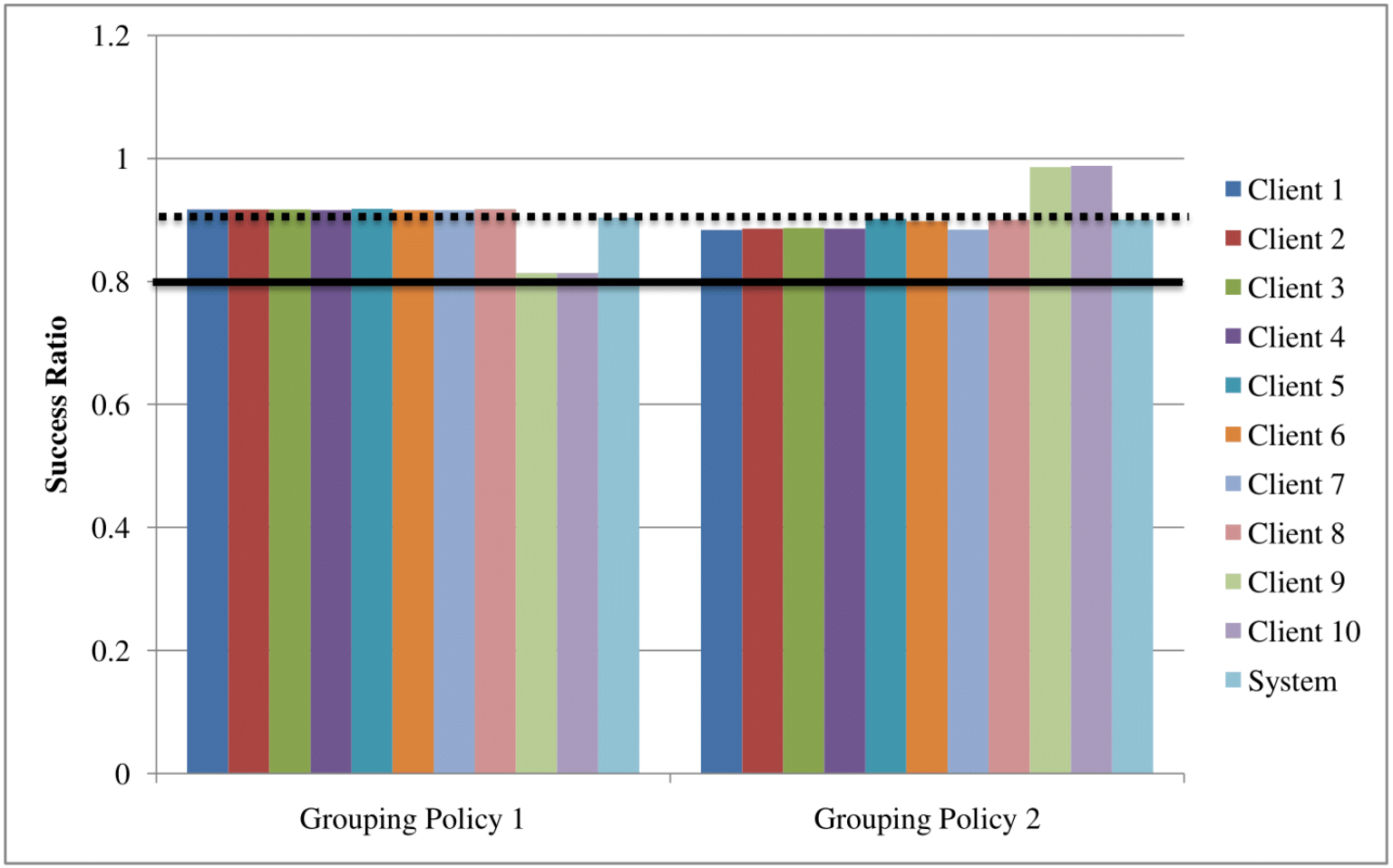
Experiment 2: Success Ratio For All Clients



Experiment 3: Adding SLOs

- Configuration:
  - Same as Experiment 2, and adding SLOs
    - Gold group: desired success ratio = 0.9
    - Silver group: desired success ratio = 0.8
    - Goal: equalizing relative success ratio / desired success ratio
  - Try two group policies
    - Policy 1: Gold = {1 ~ 8}, Silver = {9, 10}
    - Policy 2: Gold = {9, 10}, Silver = {1 ~ 8}

Experiment 3: Success Ratio For All Clients



Summary

- Fair-EDF: fairness control for latency QoS
  - Extends the standard EDF scheduler
  - Use occupancy chart for overload detection
  - Drops minimum number of requests
  - Good for streaming applications

- Future work
  - Supporting more QoS policies
  - Optimize the algorithm for better scalability

Questions to Ask

- How does the industry think the importance of this problem?
- Would there be QoS-per-client or will just categories (platinum, gold, silver) be more useful?
- How to use machine learning in the framework for better prediction?

Backup Slide: Handling Dropped Requests

- Most old storage latency solutions tried to give isolation
  - Delaying requests from clients behaving abnormally

- We have extended the idea to allow for unknown service times by using a secondary queue
  - "Latency Fairness Scheduling for Shared Storage Systems", IEEE NAS 2019