Dynamic Load Balancing for Ordered Data-Parallel Regions in Distributed Streaming Systems

Scott Schneider, Joel Wolf, Kirsten Hildrum, Rohit Khandekar and Kun-Lung Wu

presented by

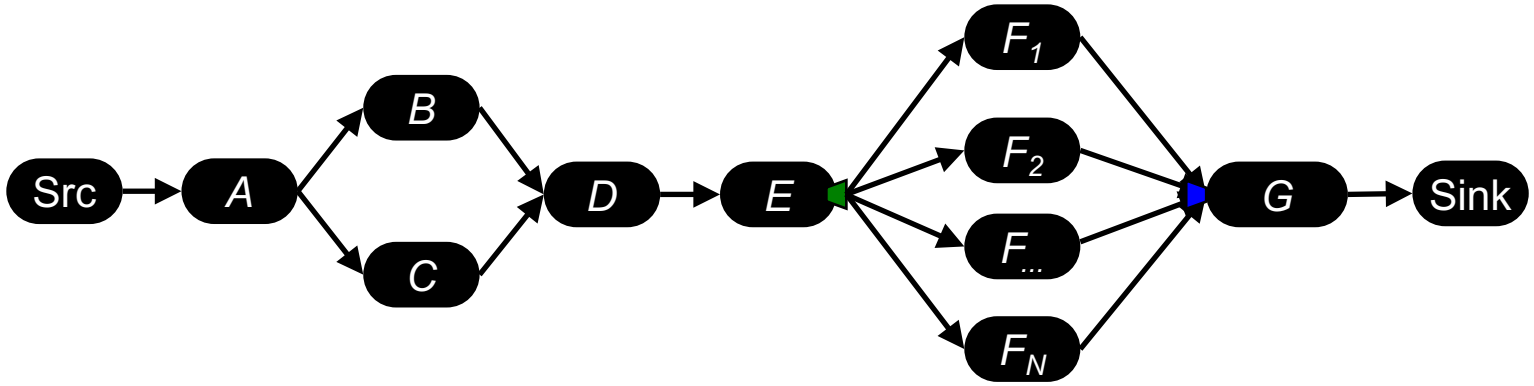
Scott Schneider

IBM Research

scott.a.s@us.ibm.com

Assumptions & Problem Statement

- Ordered parallel regions in streaming applications



- Assuming F_i are stateless operators, we want to:
  - Balance the load across all F_i in the presence of external load
  - Without using global knowledge—*only* local knowledge at the splitter
  - Without imposing significant performance cost

Blocking Time

- Calculate *blocking time* by performing non-blocking sends at transport level:

```
while ret != SENT:
  ret = nonblocking_send(c, data)
  if ret == WOULD_BLOCK:
    time = block_until_available(c)
    waited += time
```

  - `nonblocking_send(c, data)` → `sendto` system call on a TCP socket with flags set to "don't wait"
  - `block_until_available(c)` → `select` system call on TCP socket, recording how long it waited

Bad Idea 1: Data Transport Rerouting

- **The idea**: we're *electing* to block—why not send it elsewhere?

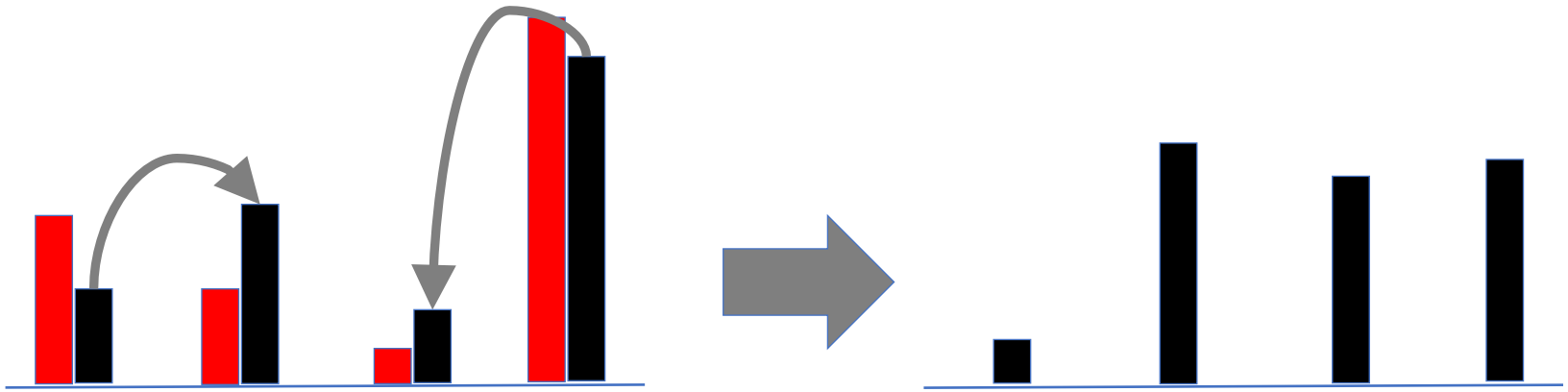
```
while ret != SENT:
  ret = nonblocking_send(c, data)
  if ret == WOULD_BLOCK:
    for i in range(max_channels):
      ret = nonblocking_send(i, data)
      if ret == SENT:
        break

    if ret != SENT:
      block_until_available(c)
```

Rerouting Results

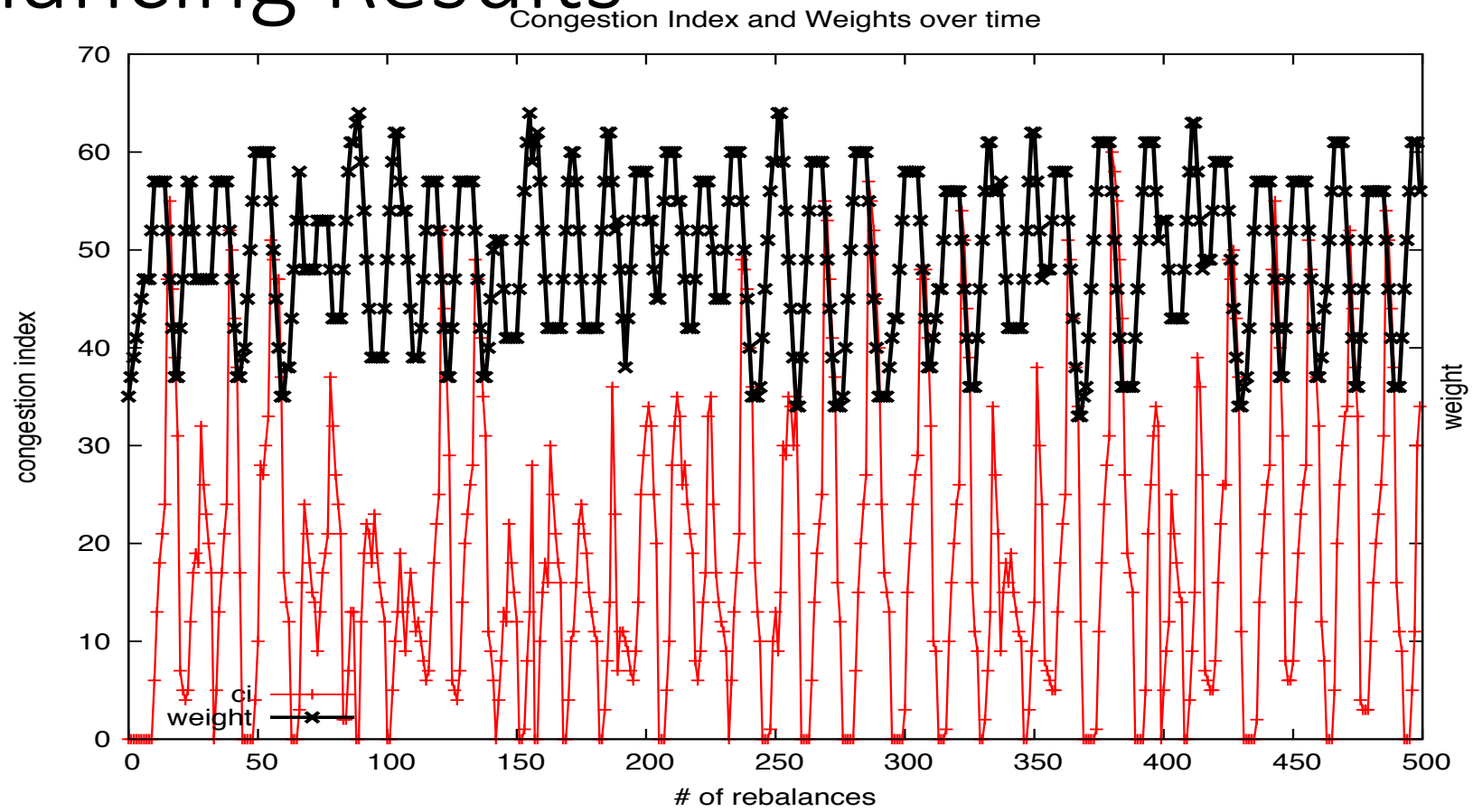
- *Medium* tuple cost
  - 200,000 tuples/s blind-blocking
  - 200,000 tuples/s rerouting; 0.4% of tuples get rerouted
- *Heavy* tuple cost
  - 26,000 tuples/s blind-blocking
  - 38,000 tuples/s rerouting; 7.5% of tuples get rerouted
- Clearly doesn't work, but *why*?
  - Too little: not enough tuples get rerouted
  - Too late: only acting when system is overloaded

Bad Idea 2: Congestion Index Balancing

- **The idea**: calculate a *congestion index* and redistribute allocation weight across channels so as to balance congestion indices



CI Balancing Results

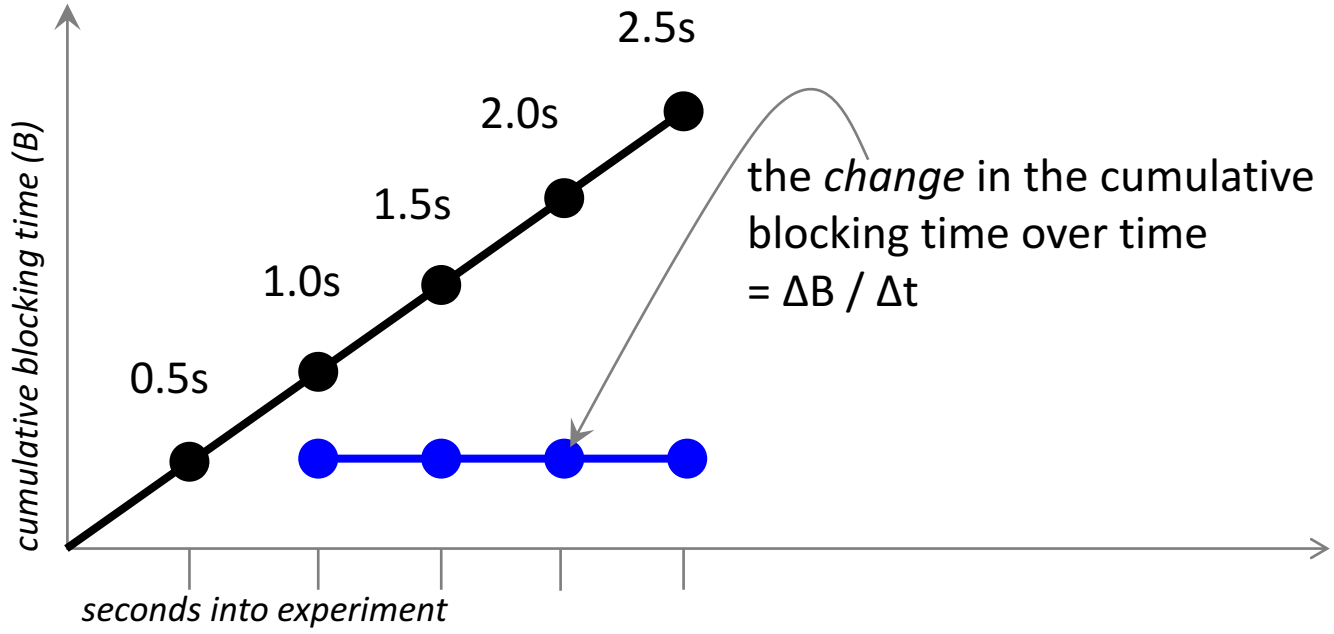


Clearly doesn't work, but *why*?

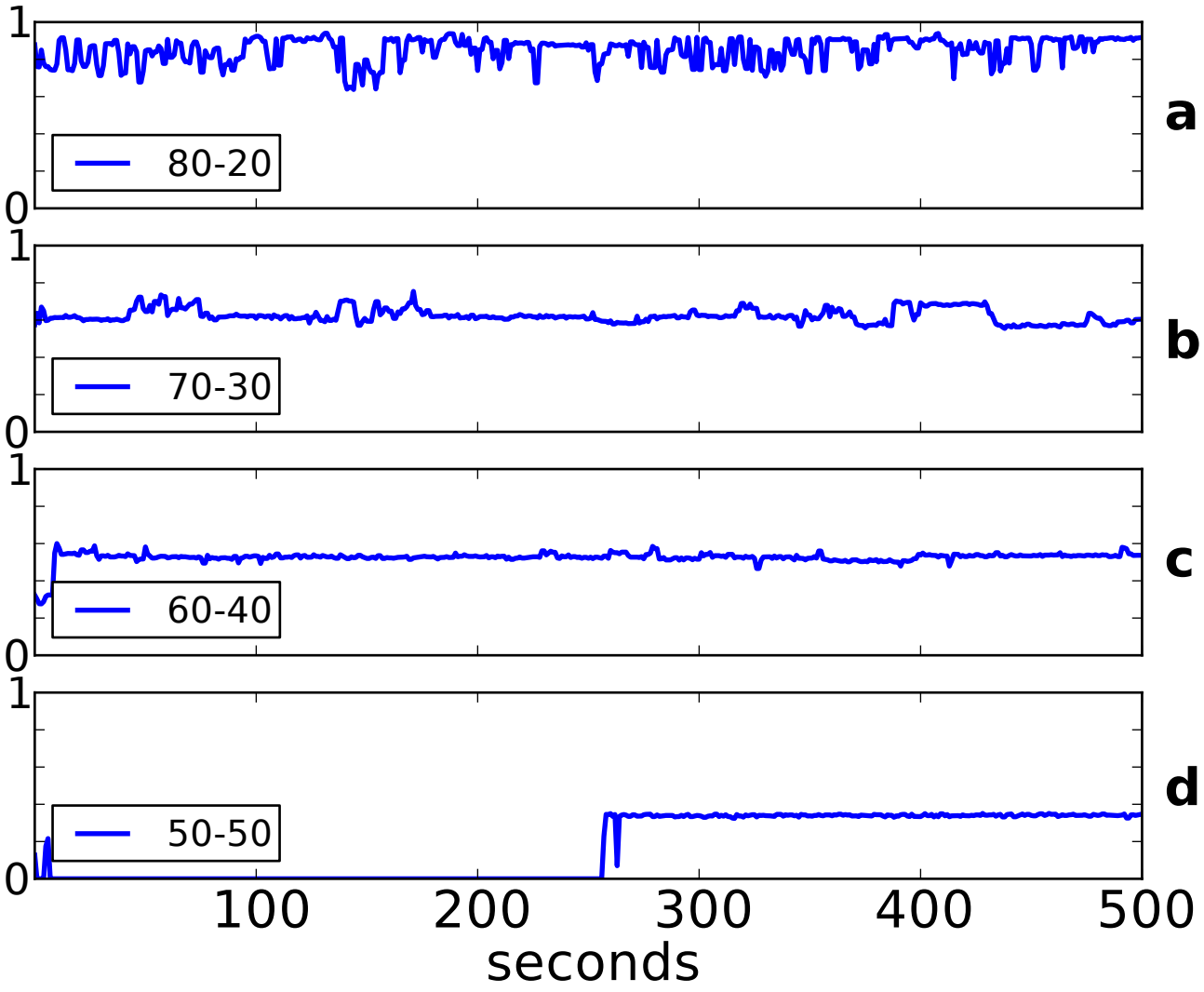
- Reacts only to instantaneous information—no history
- CI "clips" important information

Blocking Rate

- Calculate *blocking rate* by comparing blocking times over fixed periods:

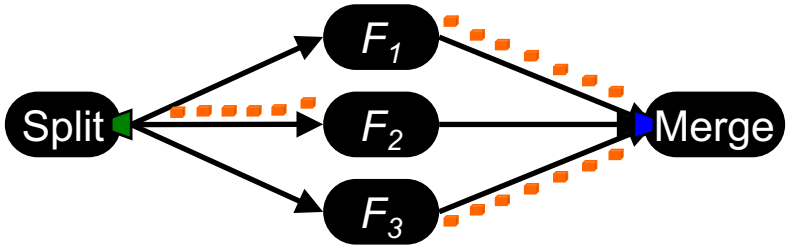


Blocking Rate is a Good Indicator of Load



Challenges

- In-order merges



- Per-connection throughput
- Drafting

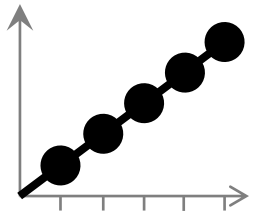
- Blocking is rare

Main Insight

- The first derivative of the blocking time when a connection is at a particular allocation weight is a constant
- More formally, $dB_{w_i} / dt = P_{w_i}$ where w_i is the weight allocation for channel i
- How can we use this?
  - P_{w_i} is roughly the probability of channel i blocking at w
  - Explore values of w_i to observe various P_{w_i}'s
  - Build a function for each connection, where $F_i(w_i)$ = probability of blocking for connection i
  - Minimize max $\{F_1(w_1), F_2(w_2), \ldots, F_N(w_N)\}$

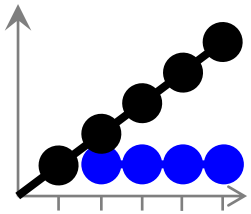
Local Load Balancing

1



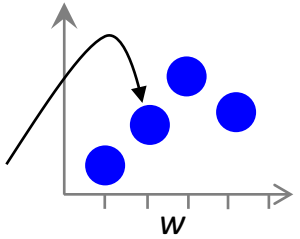
for each connection *j*, measure *cumulative blocking time* at w_j

2



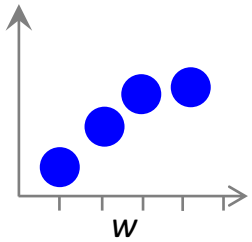
for each connection *j*, compute *blocking rate* at w_j

3



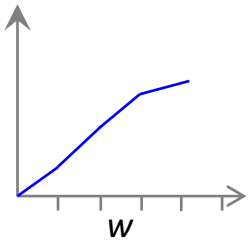
for each connection *j*, incorporate new blocking rate at w_j into history

4



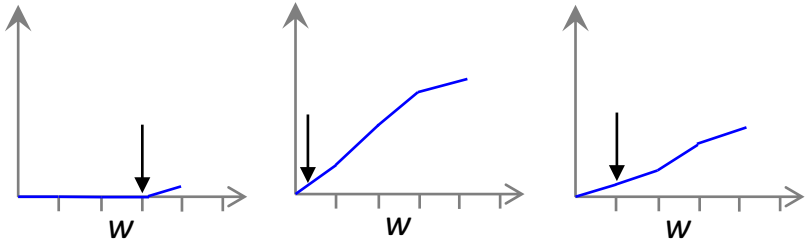
for each connection *j*, monotonize the raw data

5



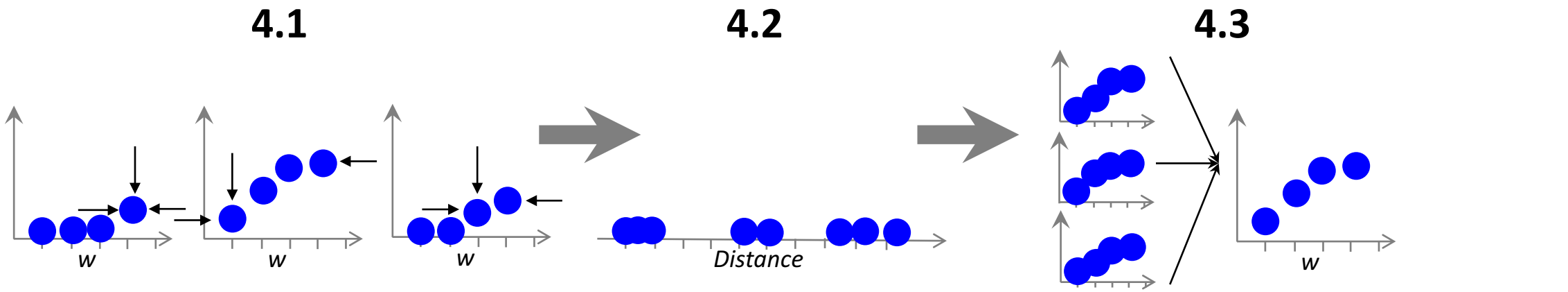
for each connection *j*, create $F_j(w_j)$ from monotonized data

6



optimize across all $F_j(w_j)$

Plus Clustering



for all connections *j* and *k*, compare $w_{j,s}$ to $w_{k,s}$; $F_j(w_{j,s})$ to $F_k(w_{k,s})$; and $F_j(w_{j,R})$ to $F_k(w_{k,R})$

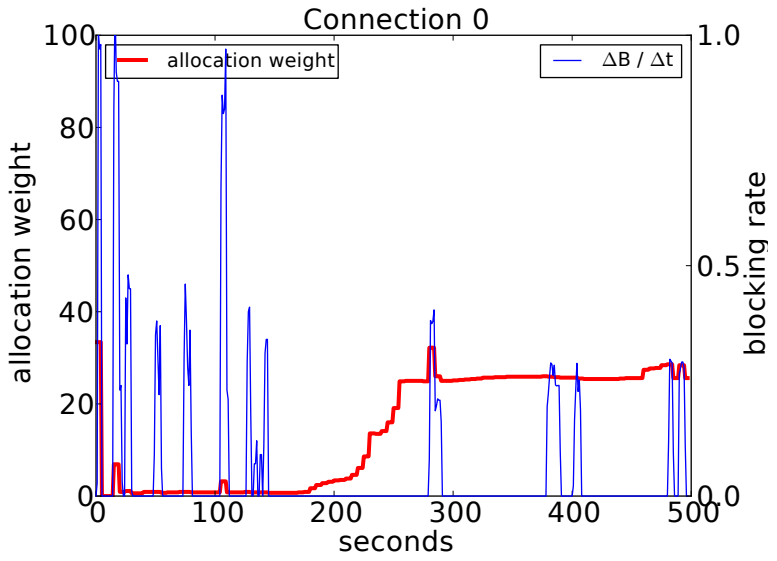
cluster based on max of those values

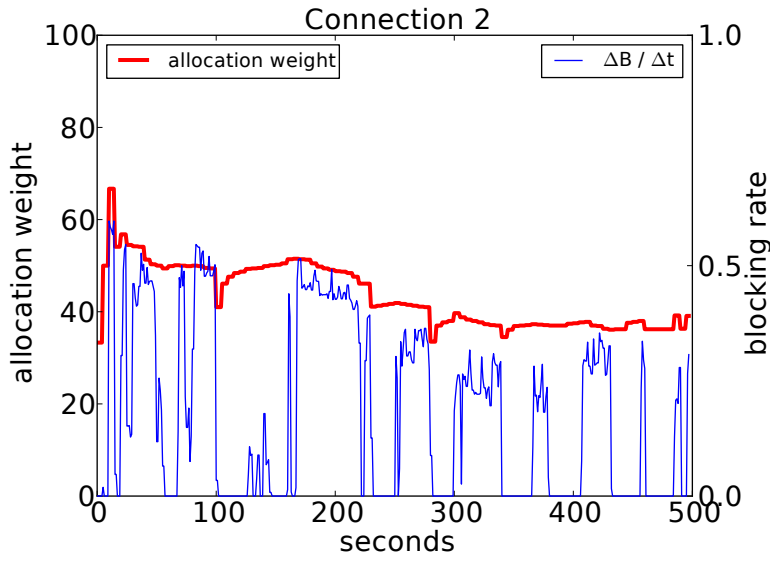
create average functions for each cluster

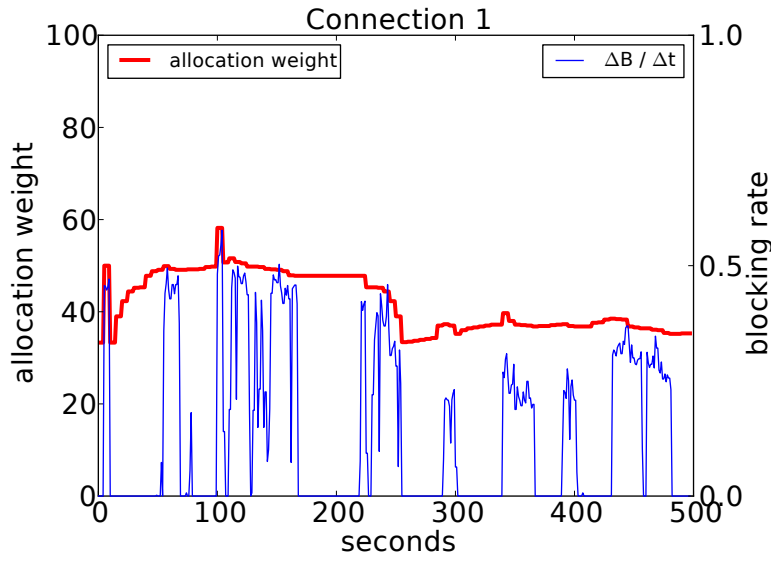
Experimental Setup

- Synthetic workloads, pumping tuples as fast as data-parallel regions can handle them
- Simulated load on some of the workers
- Tuples have a simulated “cost” to simulate actual work done
- All workers are separate processes on a different host from splitter, communicating over TCP

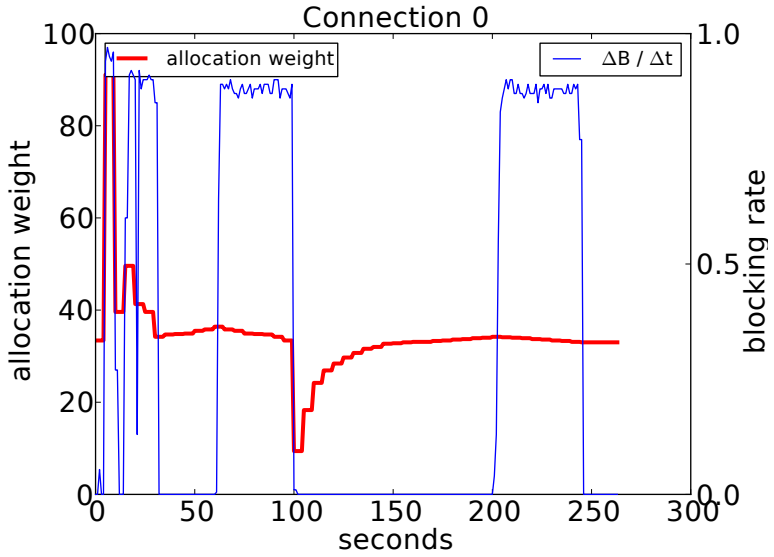
3 PEs With Load Imbalance

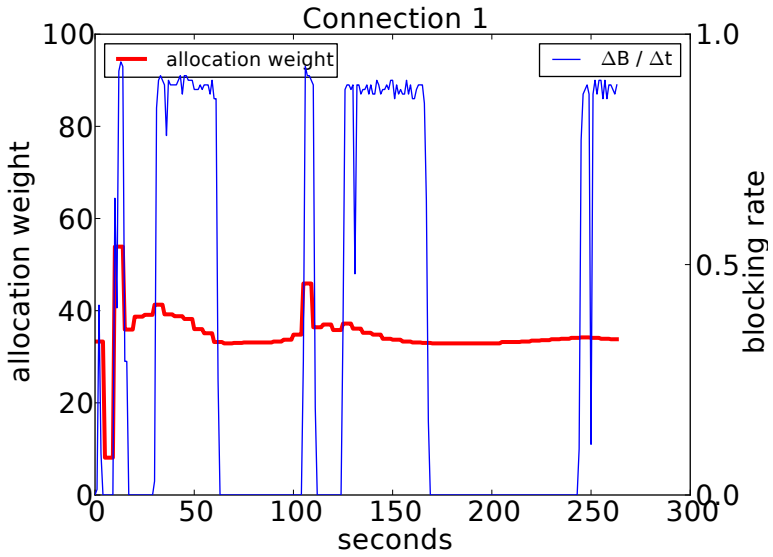


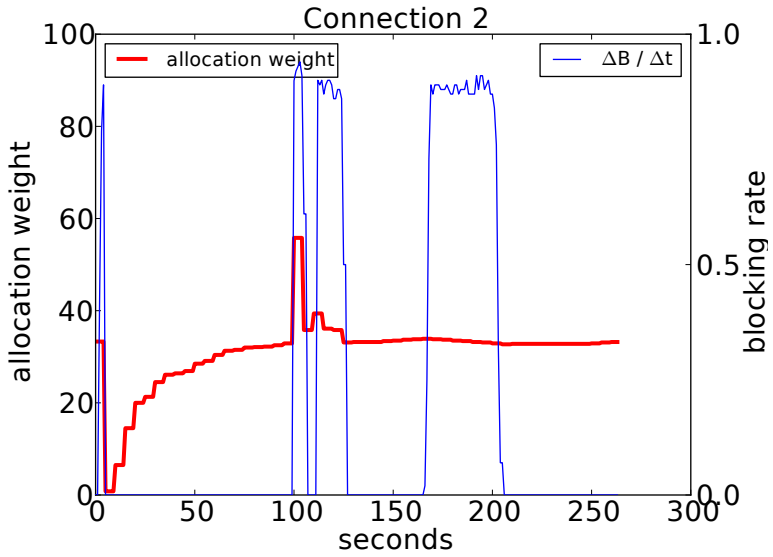




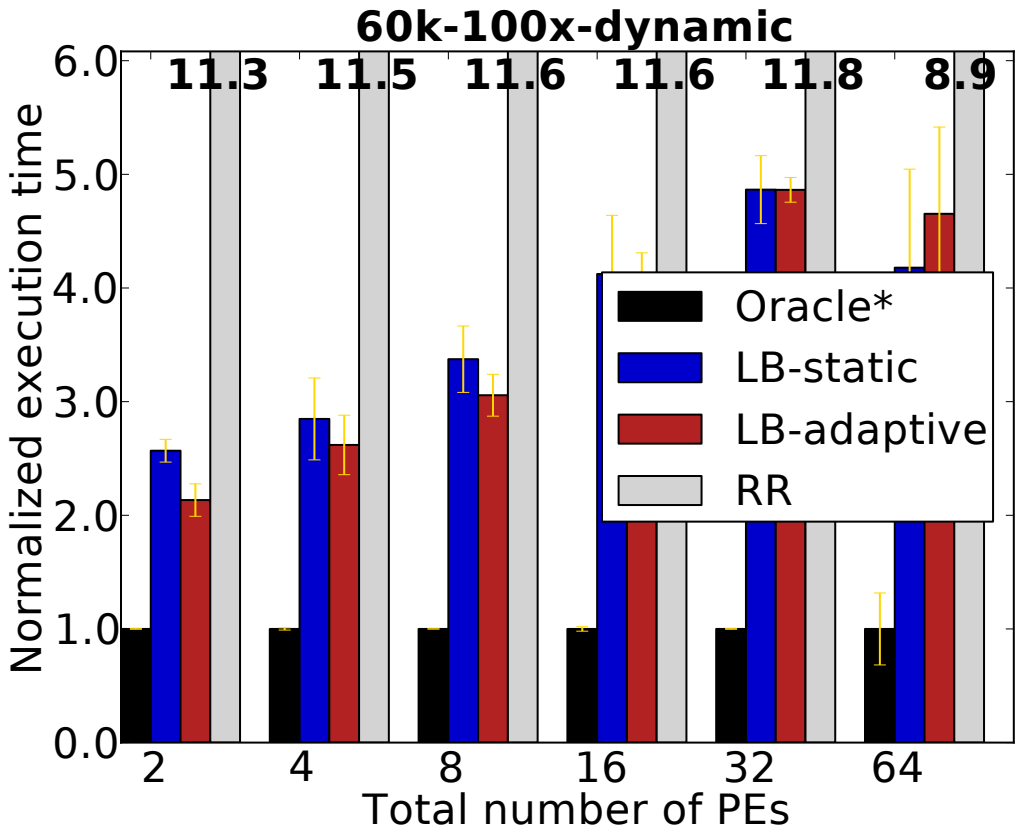
3 PEs With No Load Imbalance

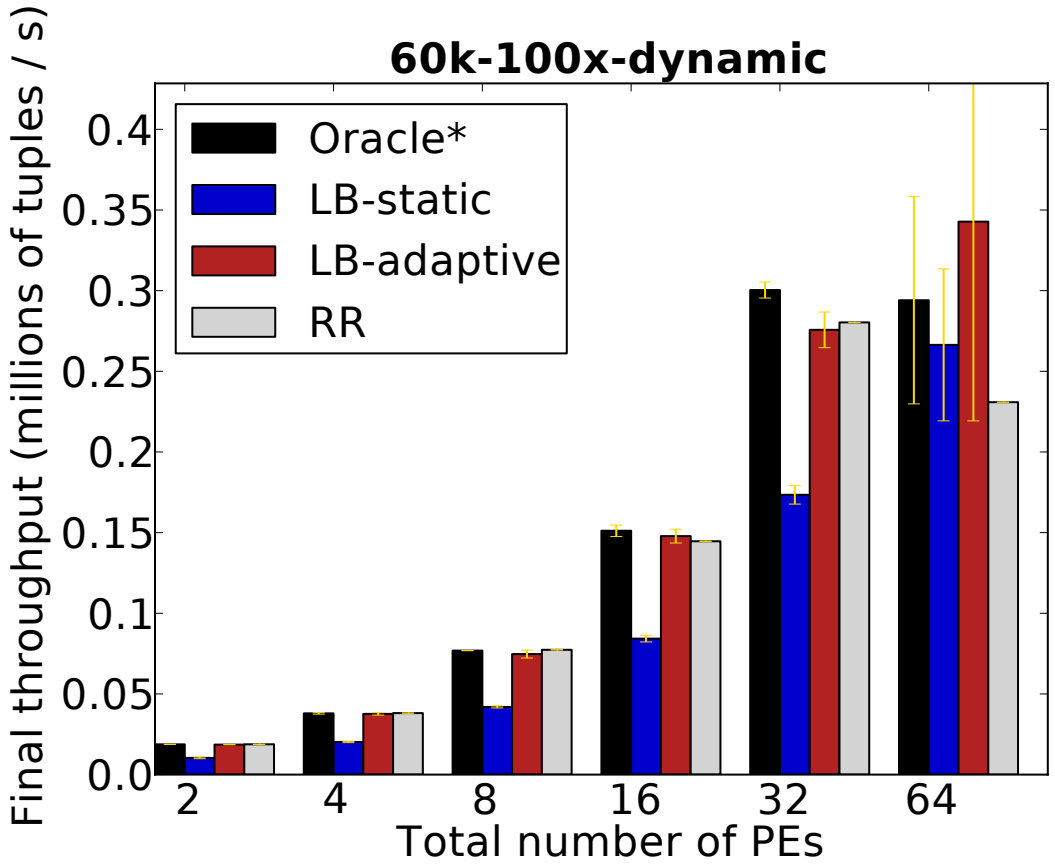




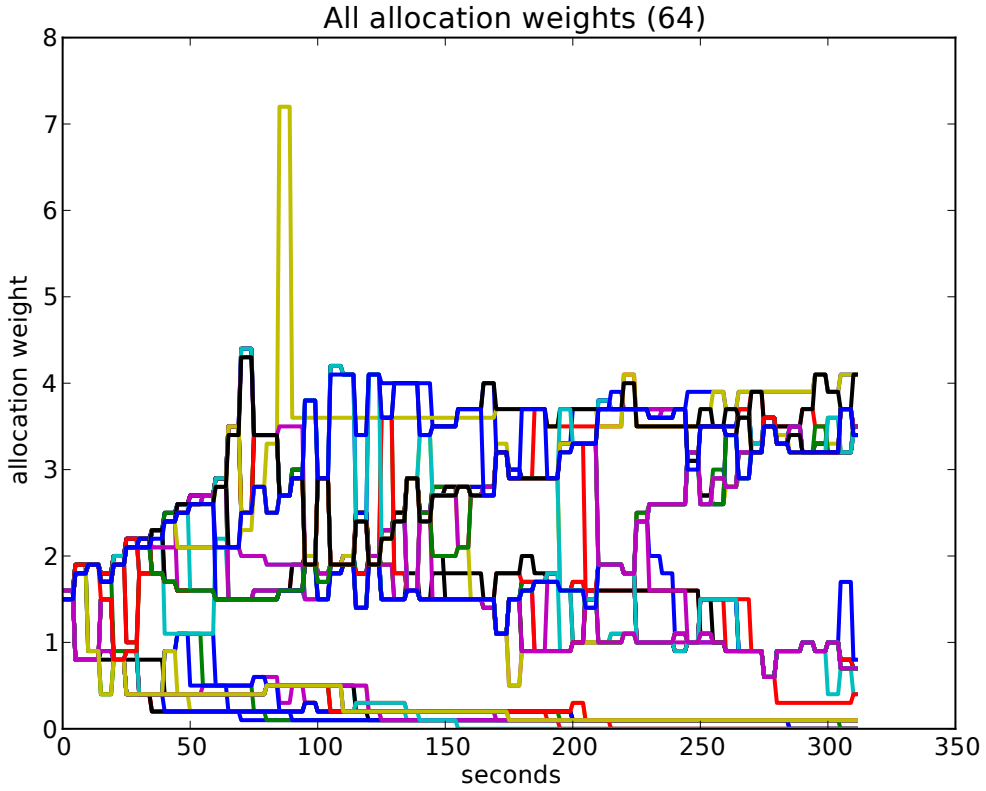


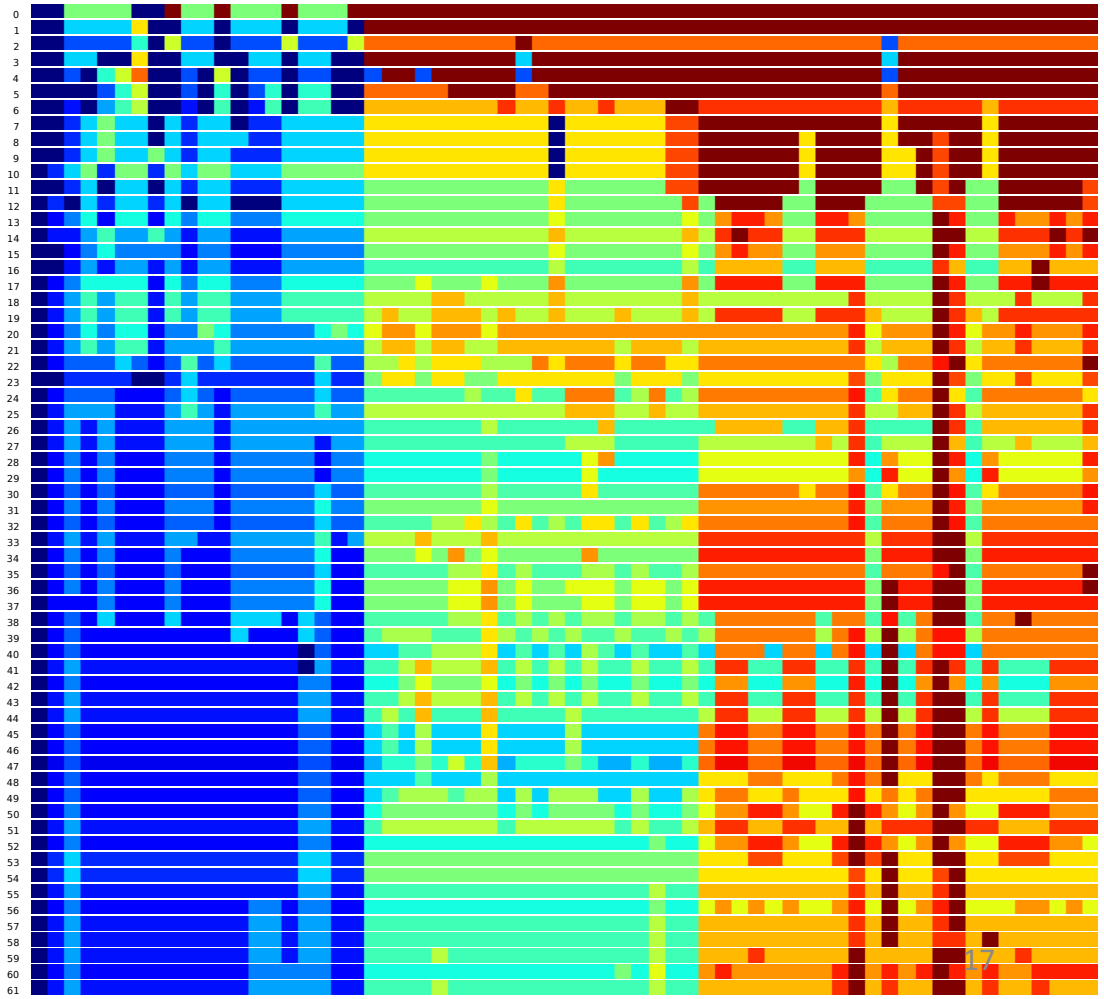
Aggregate Results With Clustering



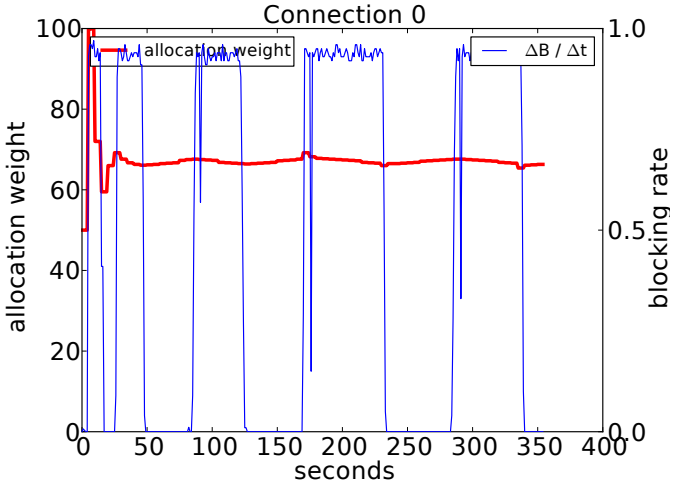


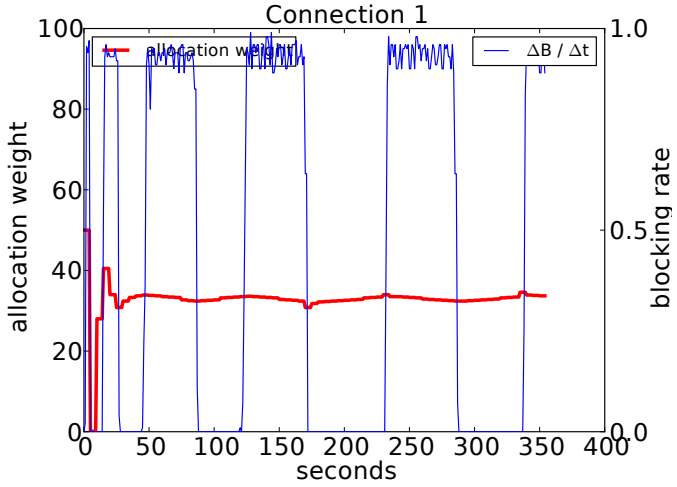
Clustering Detail With Three Groups

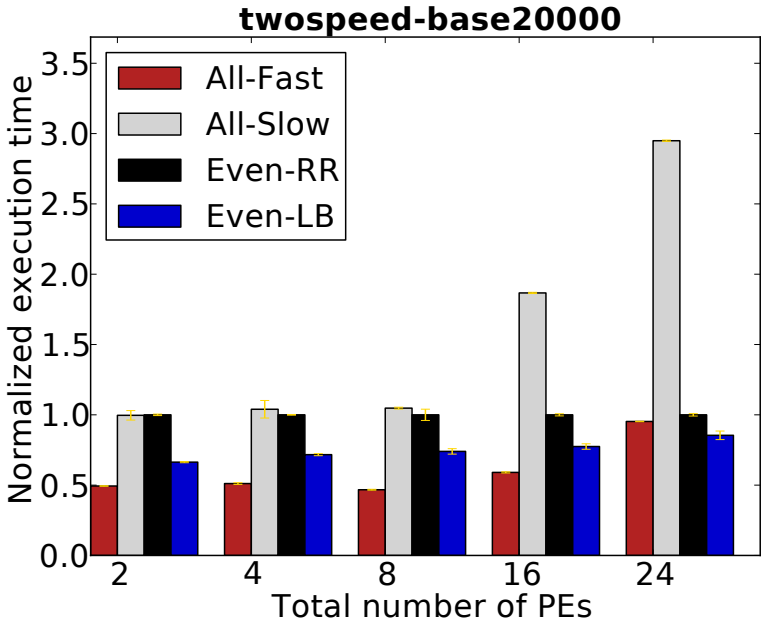


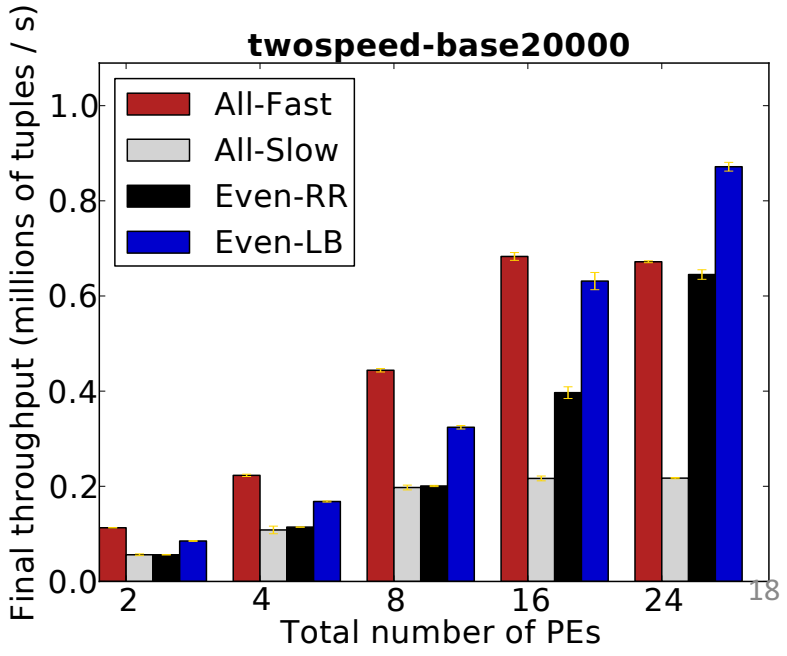


PEs on Heterogeneous Hosts









Questions?

Backup

Separable Minimax RAPs

- Separable minimax *resource allocation problems* (RAPs) with non-decreasing functions can be naturally transformed into separable convex RAPs
- And then solved via one of three algorithms of increasing speed but also increasing complexity
  - **Fox**: greedy, very simple
  - **Galil Megiddo**: logarithmic, faster
  - **Frederickson Johnson**: geometric/selection problem, fastest, complicated
- Our problem is small → use Fox
- Min and max constraints okay

Clustering Distance Function

$$\text{distance}(F_j, F_k) = \max(|\log(w_{j,s} / w_{k,s})|,$$
$$\alpha \mid \log(F_j(w_{j,s}) / F_k(w_{k,s})|,$$
$$\alpha \mid \log(F_j(w_{j,R}) / F_k(w_{k,R})|)$$

- $w_{j,s}$ is the *service rate* for connection j; it's the point where blocking becomes non-zero
- $F_j(w_{j,s})$ is the blocking connection j observes at that service rate
- $F_j(w_{j,R})$ is the blocking rate connection j observes with a high amount of load