Bede: Exploiting CXL-Memory for Cluster Job Scheduling

Pooneh Safayenikoo, Yuanchao Xu, Tanvir Ahmed Khan, Andrew Quinn
Center for Research in Systems and Storage
University of California, Santa Cruz

UC SANTA CRUZ
BaskinEngineering

Center for Research
in Systems and Storage

Motivation

CRSS

- ❖ **Scheduling is a key function in computer systems**
  - ➢ Managing clusters (e.g., Kubernetes, Mesos, Borg)
  - ➢ Handling data analytics (e.g., Spark, Hadoop)
  - ➢ Running machine learning and LLM jobs (e.g., PyTorch)
- ❖ **An efficient scheduler is crucial for large data centers**
  - ➢ Even small improvements can lead to millions in cost savings at scale

kubernetes

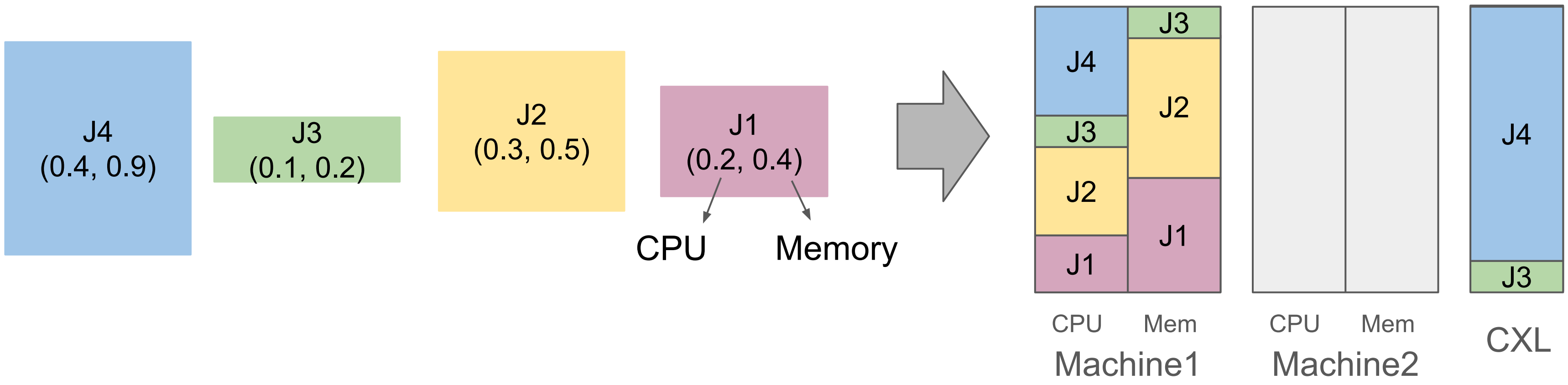
PyTorch MESOS APACHE Spark™

My Vision: Rethinking the Way We Schedule Cluster Jobs

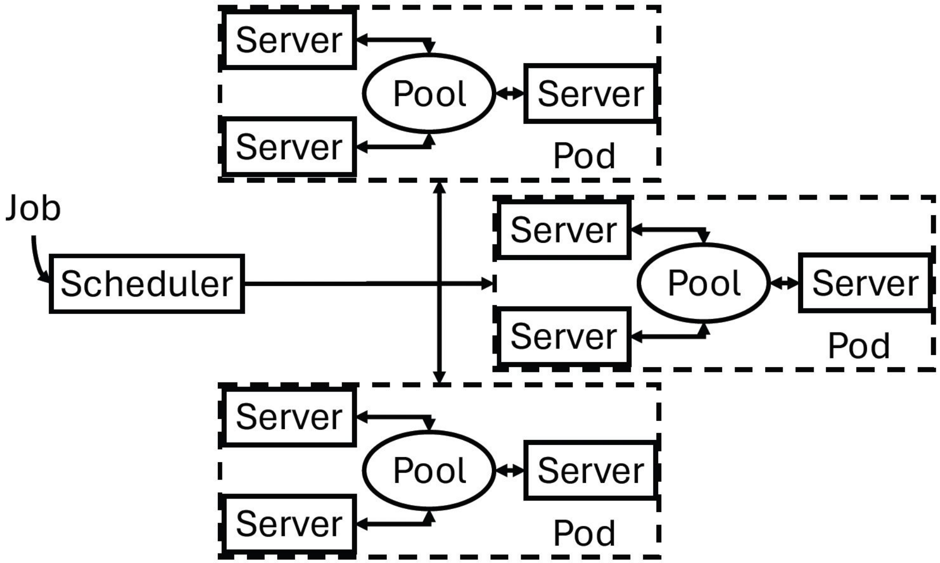
CRSS

❖ **Solution**

➢ CXL.mem shared memory pools can solve queuing delay!



Bede Design


Server
Pool
Server
Server
Pod
Job
Scheduler
Server
Pool
Server
Server
Pod
Server
Pool
Server
Server
Pod

Research Questions

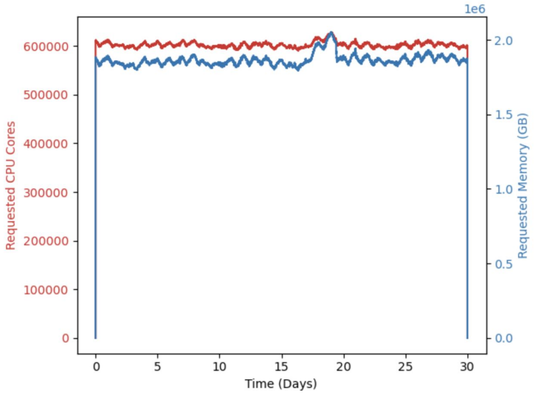
- **What's the potential of CXL.mem Memory Pooling to improve jobs scheduling performance?**
- **How do we configure CXL data center to achieve the best possible benefits?**
- **How do we schedule the jobs across the CXL and DRAM to achieve the best performance?**

Contributions

- **What's the potential of CXL.mem Memory Pooling to improve jobs scheduling performance?**
  - Conduct the study to show that CXL.mem has the potential to improve average completion time up to an order of magnitude.
- **How do we configure CXL data center to achieve the best possible benefits?**
  - Build the simulator to explore the Bede configuration space and show that small pod (8-16 machines) with most memory on DRAM achieved most of the performance benefit.
- **How do we schedule the jobs across the CXL and DRAM to achieve the best performance?**
  - Two new scheduling algorithms outperforms by on average 4.9X.

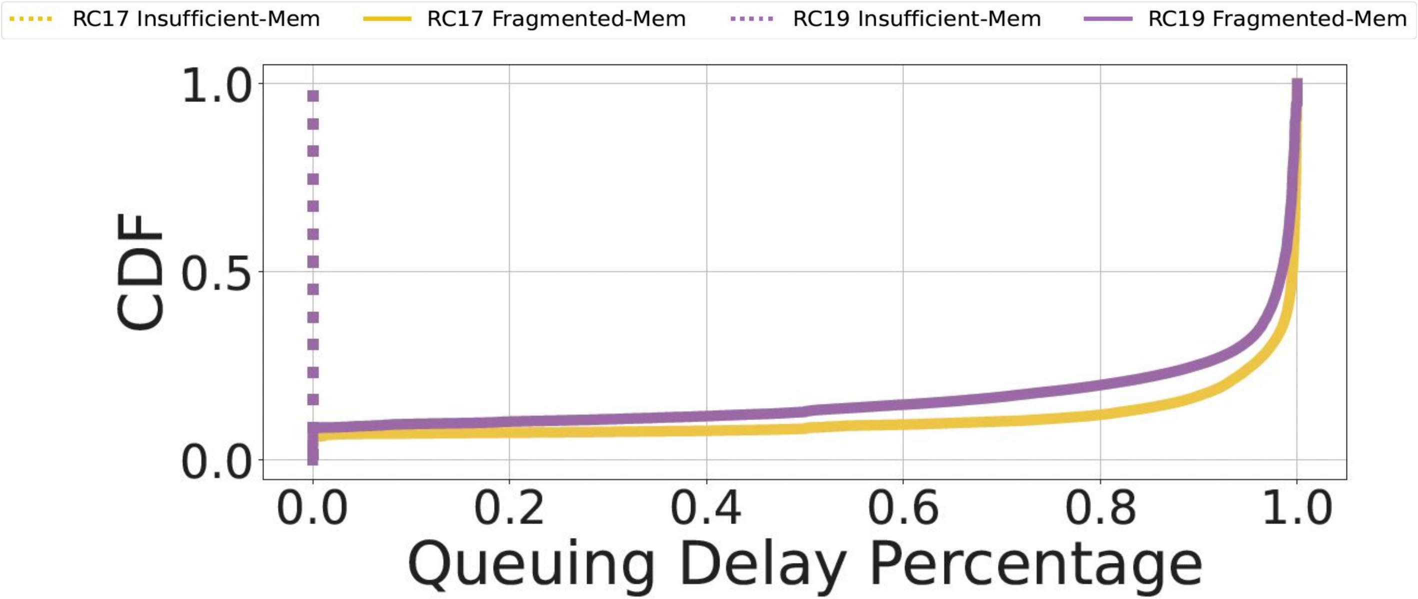
Overview of Study 1

- **Objective**
  - Investigate if CXL.mem can accelerate job performance in real-world cluster job scheduling scenarios and identify key properties of memory-based scheduling delays in real-world workloads.
- **Methodology**
  - Real world traces
    - Azure 17 and Azure 19
  - Configuration:
    - 100th percentile of requested CPU
    - Machines with 192 cores, matching large cloud instances
    - 50th, 75th, 85th, 95th percentile of requested memory
  - Scheduling algorithm
    - FIFO and SJF
    - Both use greedy resource allocation strategy

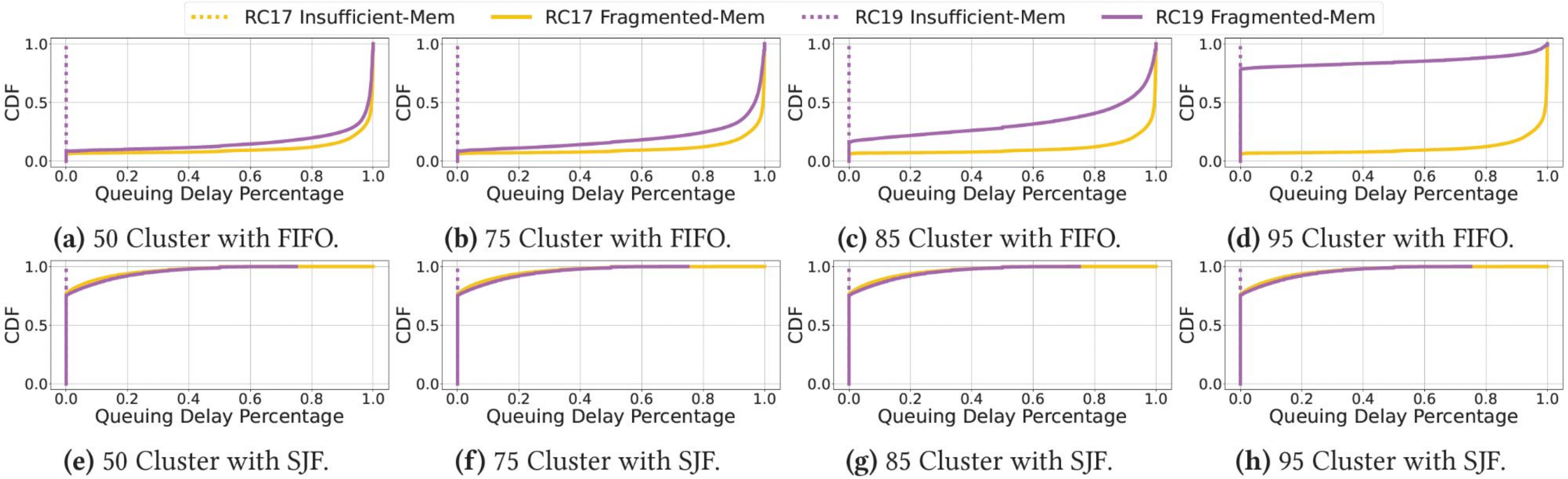


	Number of Cores	Memory 50	75	85	95
Azure17	346944	375GiB	378GiB	379 GiB	382 GiB
Azure19	628608	566 GiB	573 GiB	578 GiB	593 GiB

Scheduling Delay

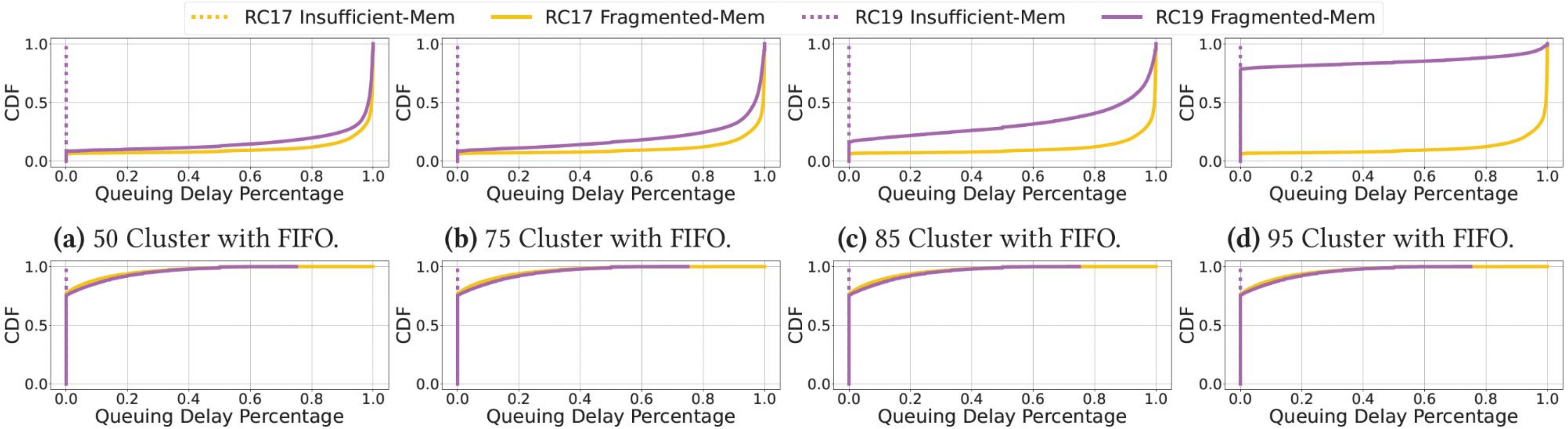


Scheduling Delay



Memory-Based Delays: Most jobs experience significant delays due to memory constraints, even with clusters near peak memory usage.

Scheduling Delay

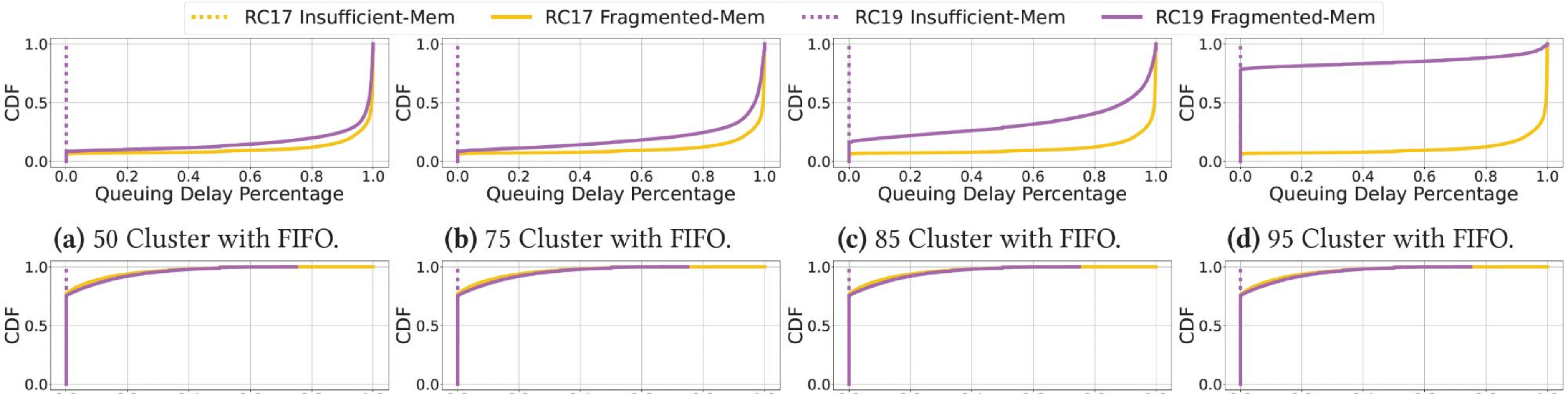


(a) 50 Cluster with FIFO. (b) 75 Cluster with FIFO. (c) 85 Cluster with FIFO. (d) 95 Cluster with FIFO.

(e) 50 Cluster with SJF. (f) 75 Cluster with SJF. (g) 85 Cluster with SJF. (h) 95 Cluster with SJF.

Fragmented Memory: Majority of delays occur due to fragmented memory, where total memory is sufficient but not available on any single machine.

Scheduling Delay



(a) 50 Cluster with FIFO. (b) 75 Cluster with FIFO. (c) 85 Cluster with FIFO. (d) 95 Cluster with FIFO.

(e) 50 Cluster with SJF. (f) 75 Cluster with SJF. (g) 85 Cluster with SJF. (h) 95 Cluster with SJF.

Scheduling Algorithms: SJF reduces scheduling delays by about 50% compared to FIFO but does not fully eliminate them.

Key Findings

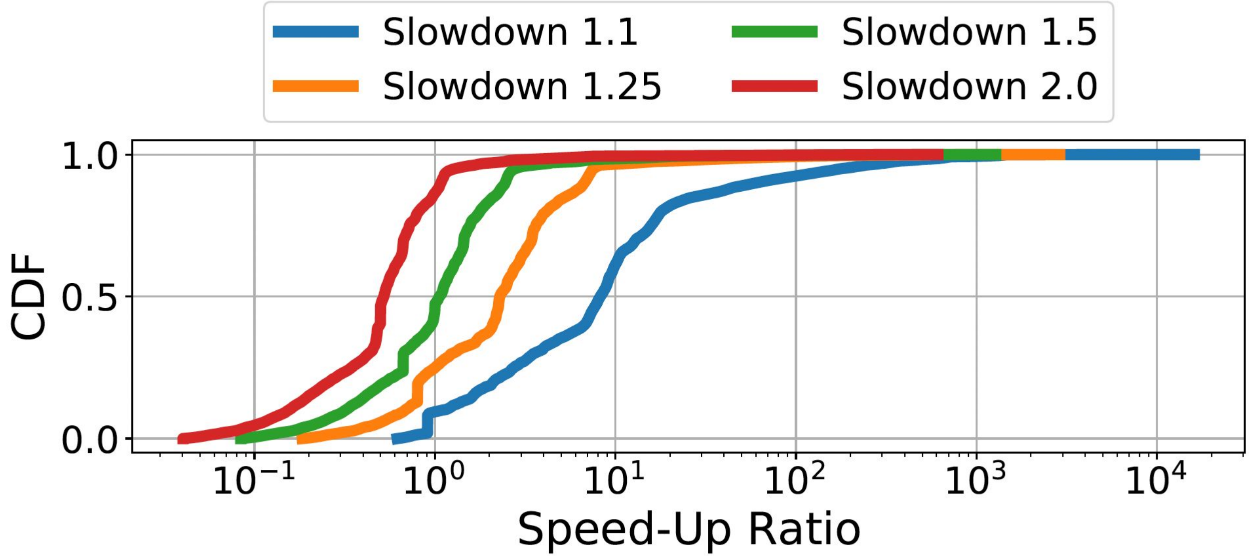
- **Insight 1**
  - **Memory-Based Delays:** Most jobs experience significant delays due to memory constraints, even with clusters near peak memory usage.
- **Insight 2**
  - **Fragmented Memory:** Majority of delays occur due to fragmented memory, where total memory is sufficient but not available on any single machine.
- **Insight 3**
  - **Scheduling Algorithms:** SJF reduces scheduling delays by about 50% compared to FIFO but does not fully eliminate them.

CXL.mem can help reduce scheduling delay by avoiding memory fragmentation within a cluster

Overview of Study 2

- **Objective**
  - Identifying whether the reduction in scheduling delay outweighs job slowdown when using a CXL.mem memory pool
- **Methodology**
  - Real world trace
    - Azure 17
  - Configuration
    - 100th percentile of requested CPU
    - 85 cluster (a cluster deployed with the 85th percentile of requested memory)
    - All of its memory in a single memory pool
    - Varying application slowdown from 1.1 to 2.0 (all applications have the same slowdown)
  - Scheduling algorithm
    - FIFO

Per-job Speedup with CXL under Slowdown Condition

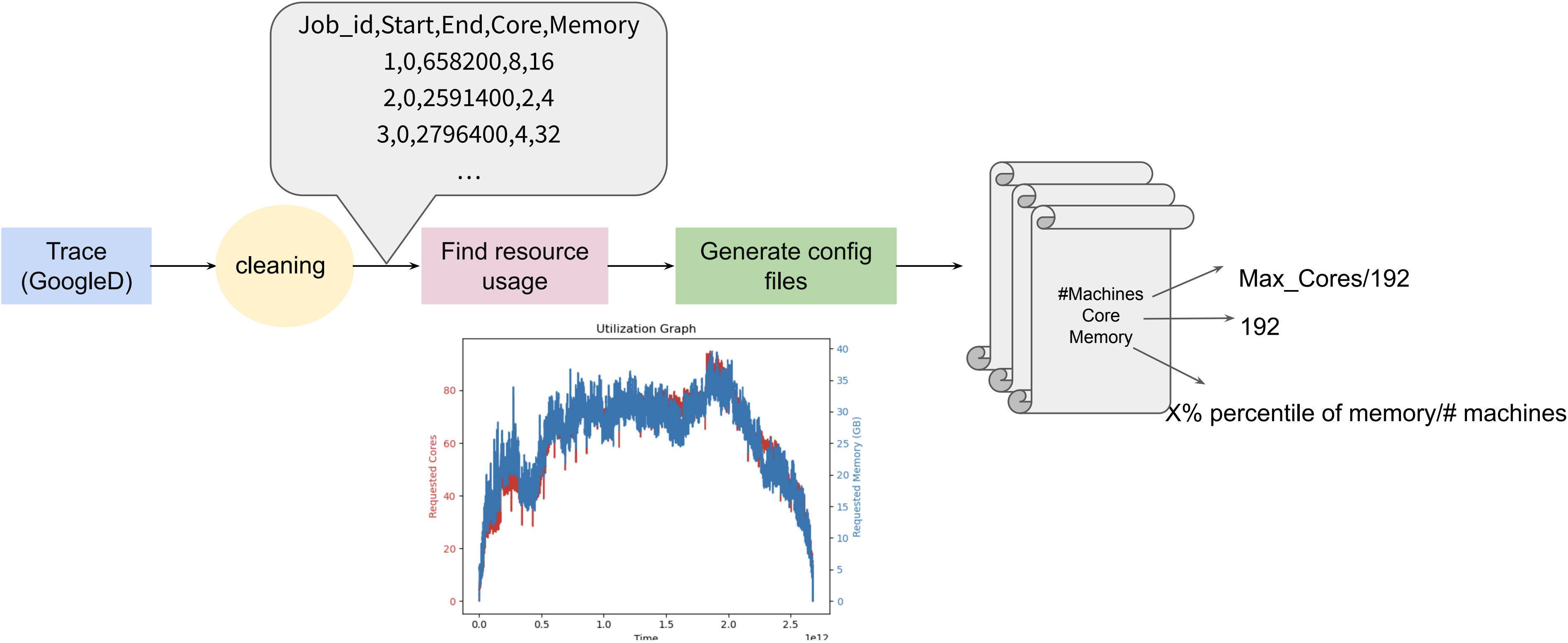


The median job is a nearly an order of magnitude faster when application slowdown is 10%

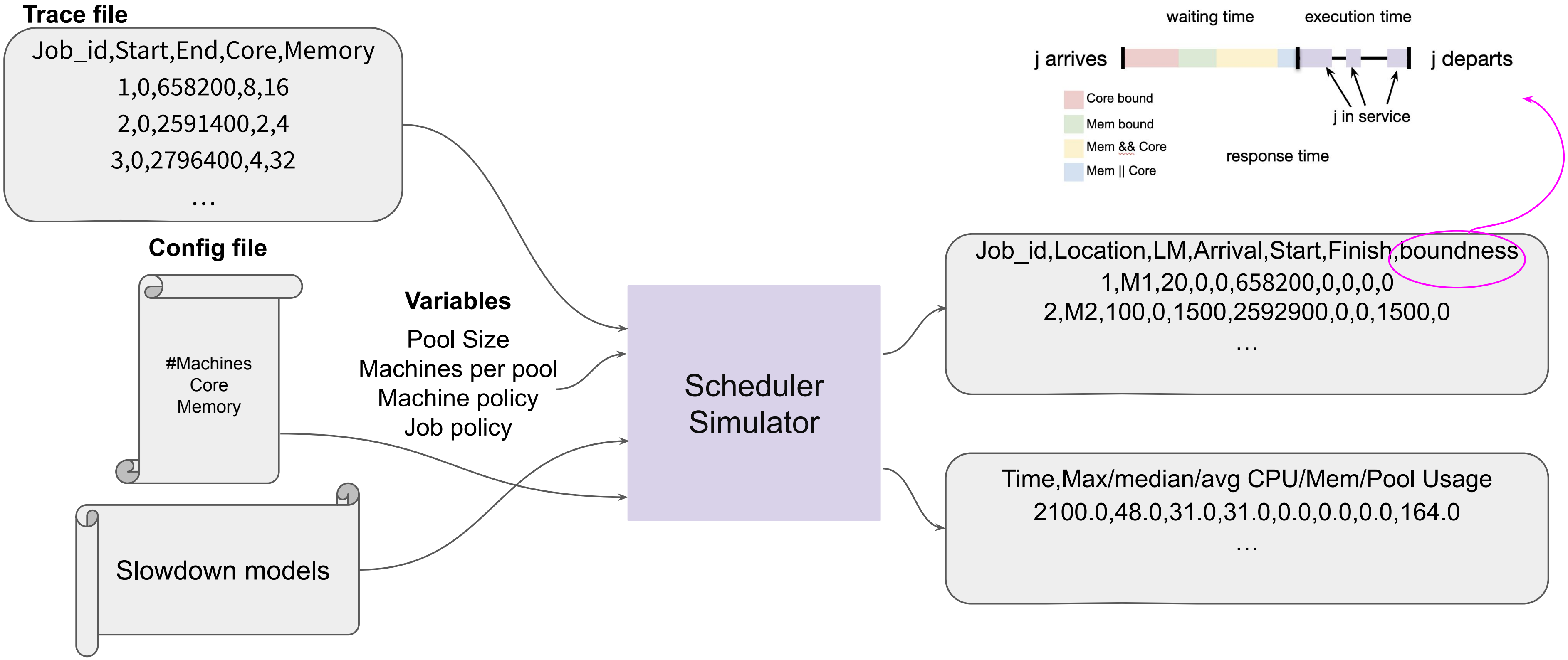
Contributions

- **What's the potential of CXL.mem Memory Pooling to improve jobs scheduling performance?**
  - Conduct the study to show that CXL.mem has the potential to improve average completion time by up to an order of magnitude.
- **How do we configure CXL data center to achieve the best possible benefits?**
  - Build the simulator to explore the Bede configuration space and show that small pod (8-16 machines) with most memory on DRAM achieved most of the performance benefit.
- **How do we schedule the jobs across the CXL and DRAM to achieve the best performance?**
  - Two new scheduling algorithms outperform prior work by 4.9X on average.

Simulator


Job_id,Start,End,Core,Memory
1,0,658200,8,16
2,0,2591400,2,4
3,0,2796400,4,32
...
Trace (GoogleD)
cleaning
Find resource usage
Generate config files
#Machines
Core
Memory
Max_Cores/192
192
X% percentile of memory/# machines
Utilization Graph
Requested Cores
Requested Memory (GB)
Time
1e12

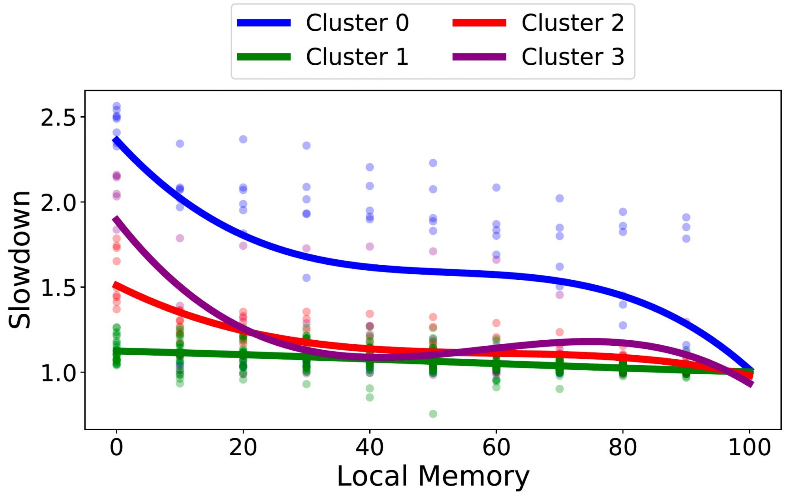
Simulator



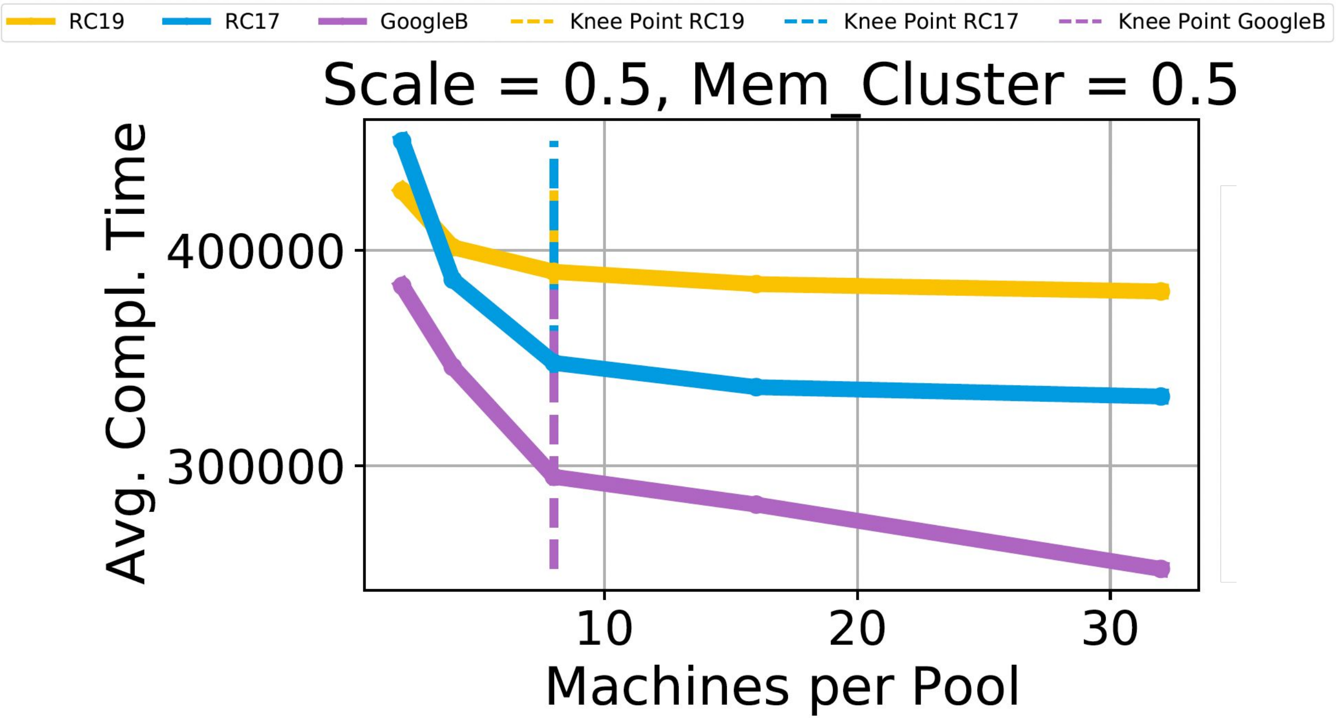
Trace file
Job_id,Start,End,Core,Memory
1,0,658200,8,16
2,0,2591400,2,4
3,0,2796400,4,32
…
Config file
#Machines
Core
Memory
Slowdown models
Variables
Pool Size
Machines per pool
Machine policy
Job policy
Scheduler Simulator
waiting time
execution time
j arrives
j departs
Core bound
Mem bound
Mem && Core
Mem || Core
j in service
response time
Job_id,Location,LM,Arrival,Start,Finish,boundness
1,M1,20,0,0,658200,0,0,0,0
2,M2,100,0,1500,2592900,0,0,1500,0
…
Time,Max/median/avg CPU/Mem/Pool Usage
2100.0,48.0,31.0,31.0,0.0,0.0,0.0,164.0
…

Slowdown Model

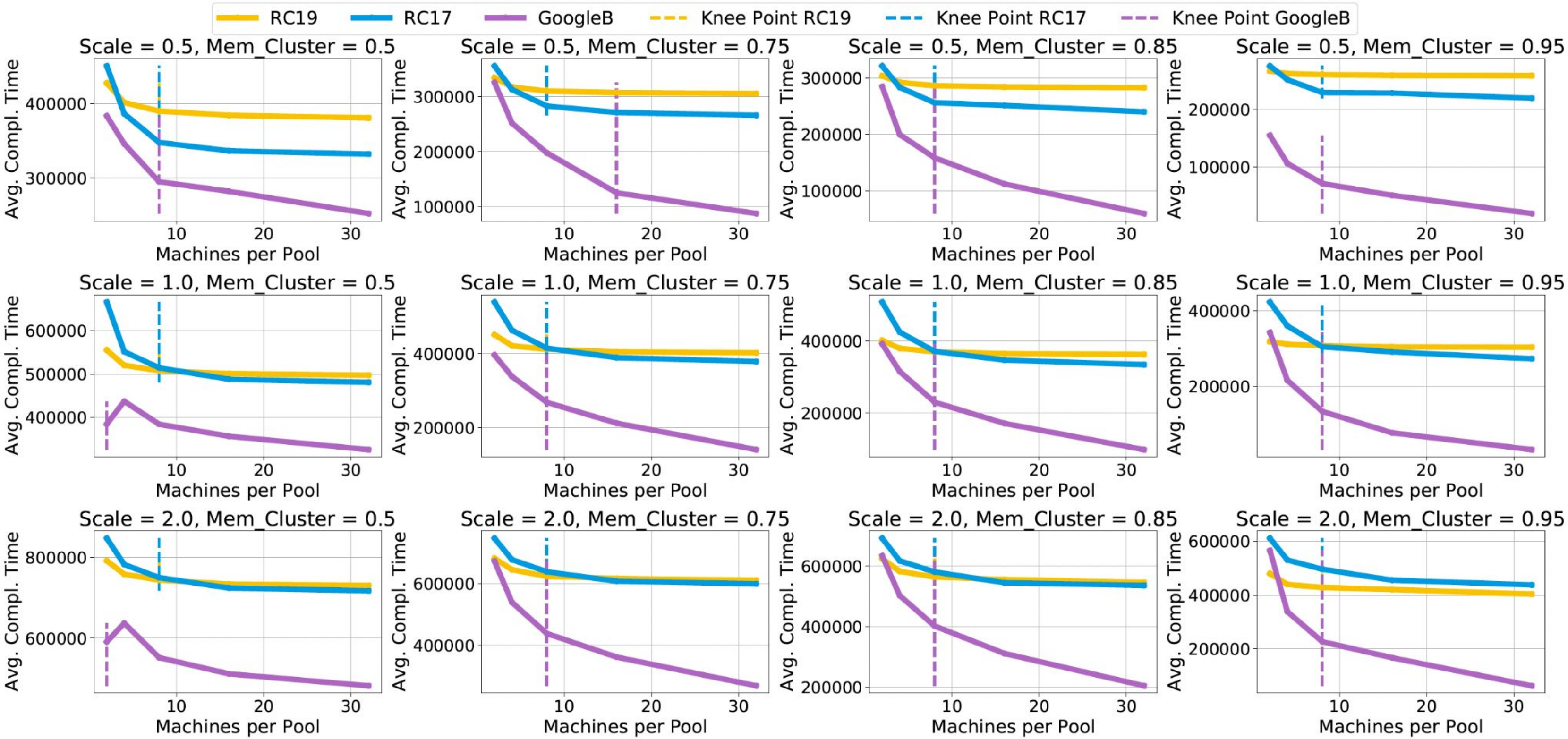
- **Slowdown Models**
  - Required to simulate CXL.mem memory pool impact on job latency.
  - Predicts slowdown when local memory resources are reduced.
- **Challenges**
  - CXL.mem memory pools are not yet commercially available.
- **Simulator Approach**
  - Uses a two-socket NUMA machine to simulate CXL.mem pools.
  - Varies local memory via mlock() and fits a degree 3 polynomial for slowdown.
- **Mitigation Strategy**
  - Includes a tunable **scale factor** to adjust CXL.mem pool performance relative to NUMA.
  - Example: Scale factor of 2 means CXL.mem is twice as slow as NUMA.



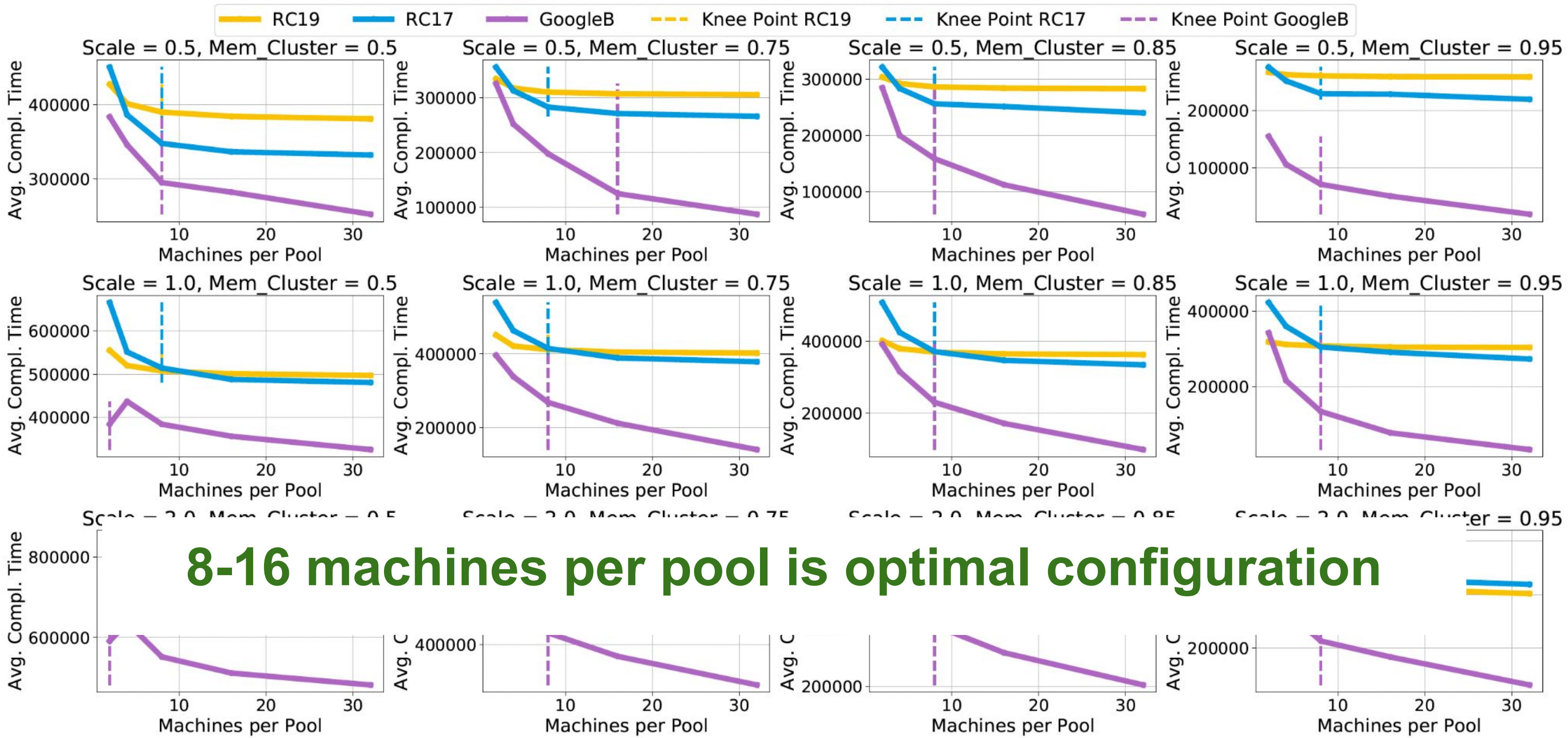
The number of Machines Per Pool



The number of Machines Per Pool

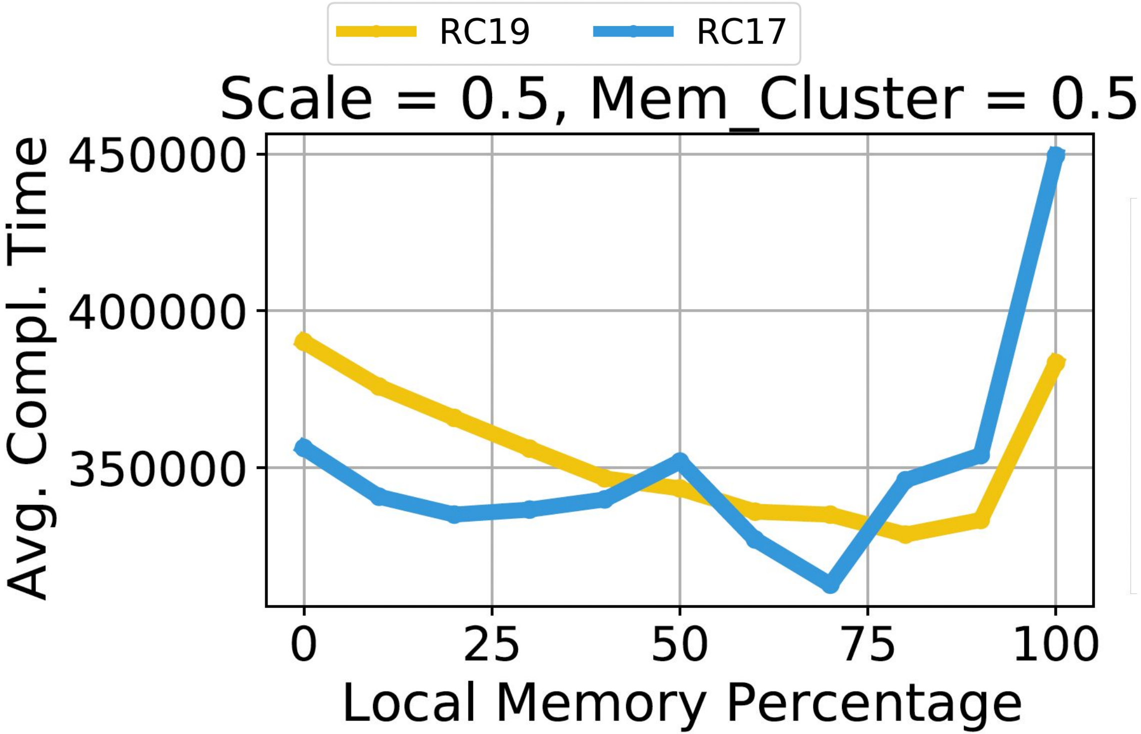


The number of Machines Per Pool

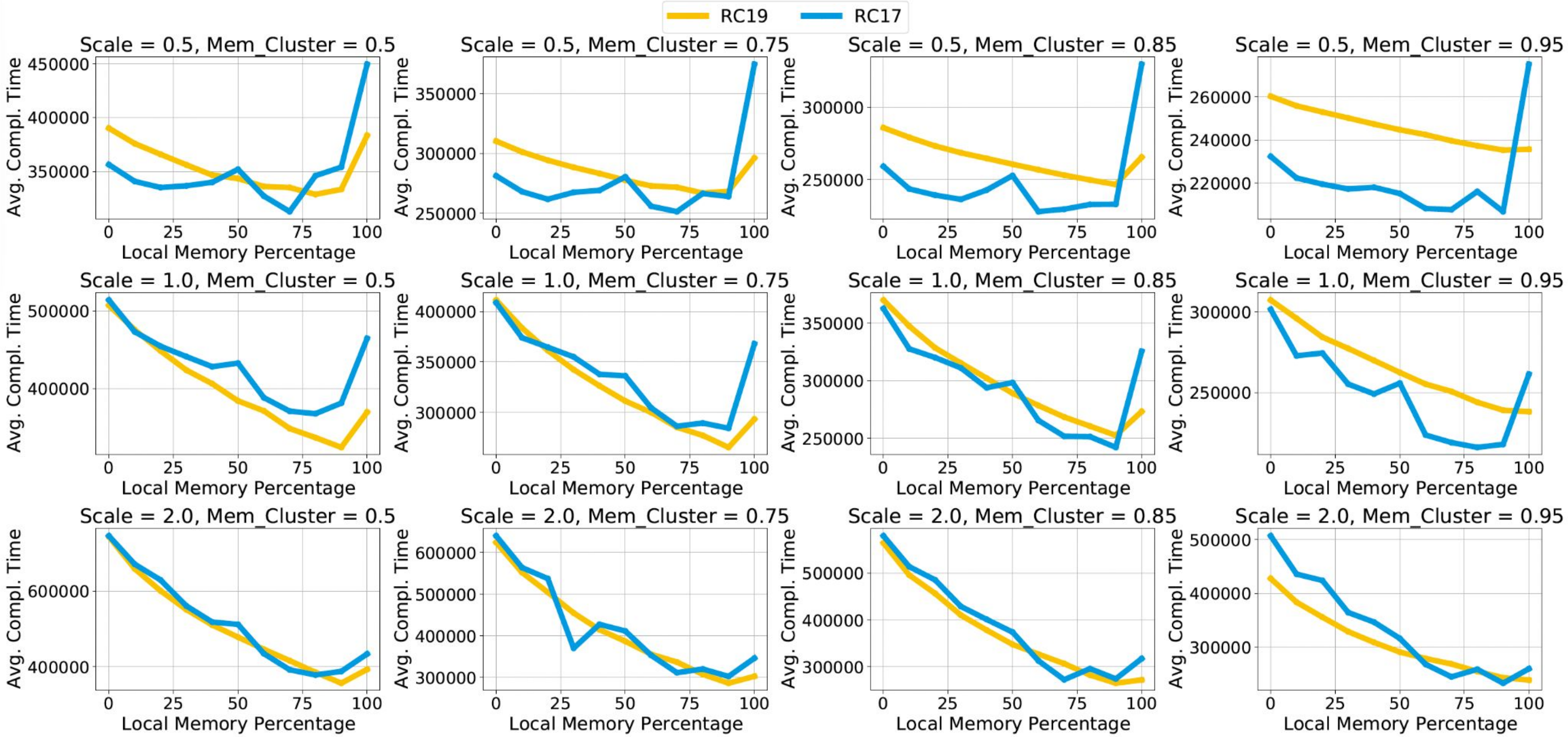


8-16 machines per pool is optimal configuration

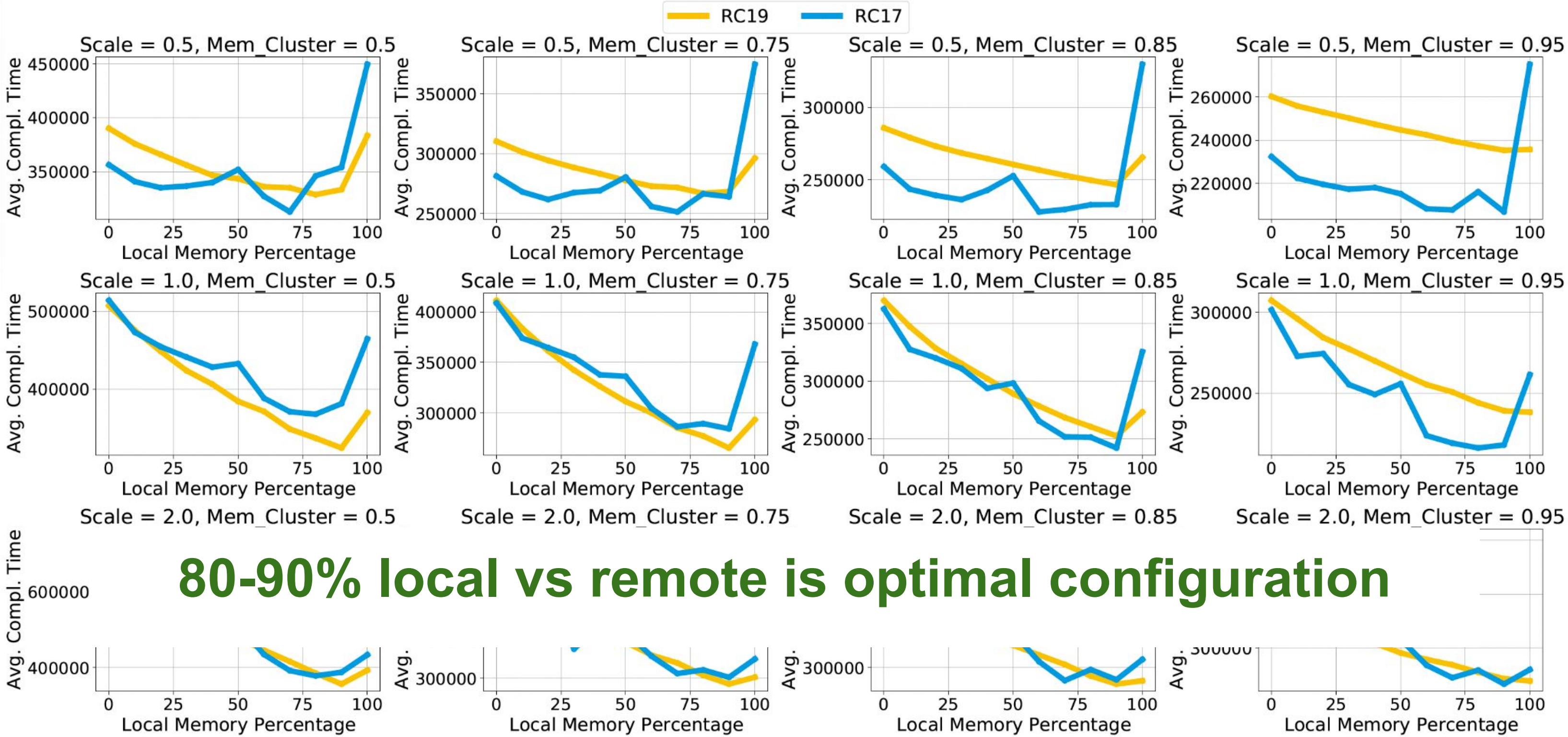
Server-Pool Memory Split



Server-Pool Memory Split



Server-Pool Memory Split



80-90% local vs remote is optimal configuration

Contributions

- **What's the potential of CXL.mem Memory Pooling to improve jobs scheduling performance?**
  - Conduct the study to show that CXL.mem has the potential to improve average completion time by up to an order of magnitude.
- **How do we configure CXL data center to achieve the best possible benefits?**
  - Build the simulator to explore the Bede configuration space and show that small pod (8-16 machines) with most memory on DRAM achieved most of the performance benefit.
- **How do we schedule the jobs across the CXL and DRAM to achieve the best performance?**
  - Two new scheduling algorithms outperform prior work by 4.9X on average.

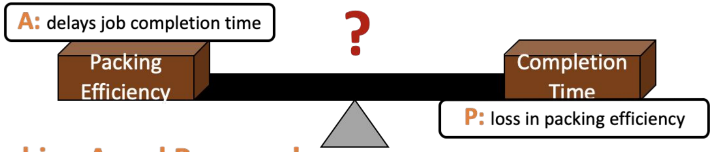
Intuitions

- **E-PVM (Enhanced version of the Parallel Virtual Machine)**
  - When a job arrives, it calculates the marginal cost of assigning the job to each machine and assigns the job to the machine with the smallest cost.

- **Tetris**
  - Packing heuristic + Job Completion Time Heuristic

"alignement" = Job's demand vector * Machine resource vector

"remaining work" = remaining # tasks & tasks' resource demands & tasks' durations



Combine A and P scores !

EVPM-Far

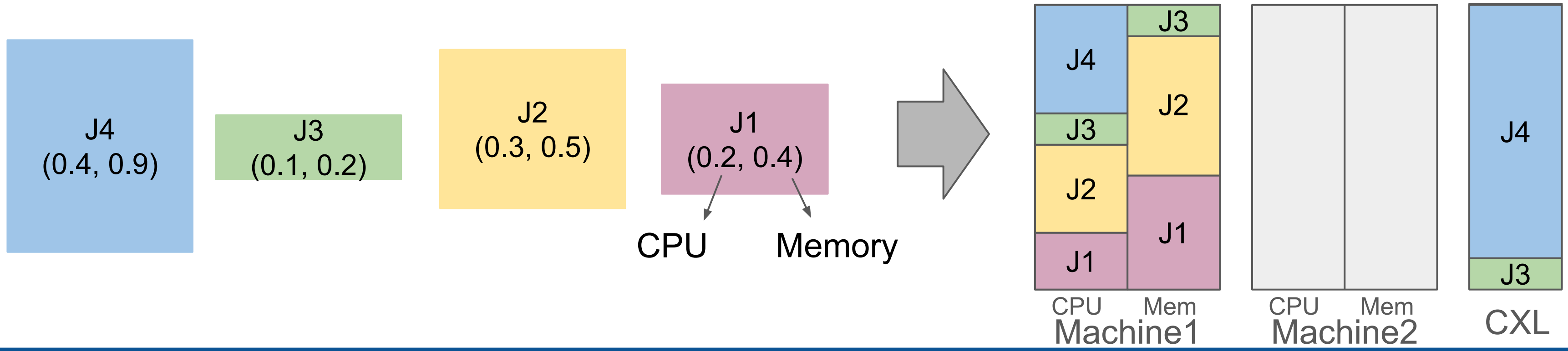
❖ **For job J, select the machine with the minimum score among N runnable machines:**

$$for\ M = 1, 2, ..., N$$

$$J_{Mem\_LM} = min(M_{unused\_Mem}, J_{Mem}), \quad J_{Mem\_Pool} = J_{Mem} - J_{Mem\_LM}$$

$$Score_M = (J_{CPU} * M_{unused\_CPU}) + \frac{1}{2}(J_{Mem\_LM} * M_{unused\_Mem}) + \frac{1}{2}(J_{Mem\_Pool} * Pool_{unused\_Mem})$$

$$S_{min} = min_{M=1}^{N} Score_M$$



T-Far

- **For machine M, select the job with the maximum score among P runnable jobs:**

If Job's demand vector < Machine resource vector

a = Job's demand vector * Machine resource vector

$$for\ J = 1, 2, ..., P$$

$$J_{Mem\_LM} = min(M_{unused\_Mem}, J_{Mem}), \quad J_{Mem\_Pool} = J_{Mem} - J_{Mem\_LM}$$

$$a_J = \left(\frac{J_{CPU}}{M_{CPU}} * M_{unused\_CPU}\right) + \left(\frac{J_{Mem\_LM}}{M_{Mem}} * M_{unused\_Mem}\right) + \left(\frac{J_{Mem\_Pool}}{Pool_{Mem}} * Pool_{unused\_Mem}\right)$$

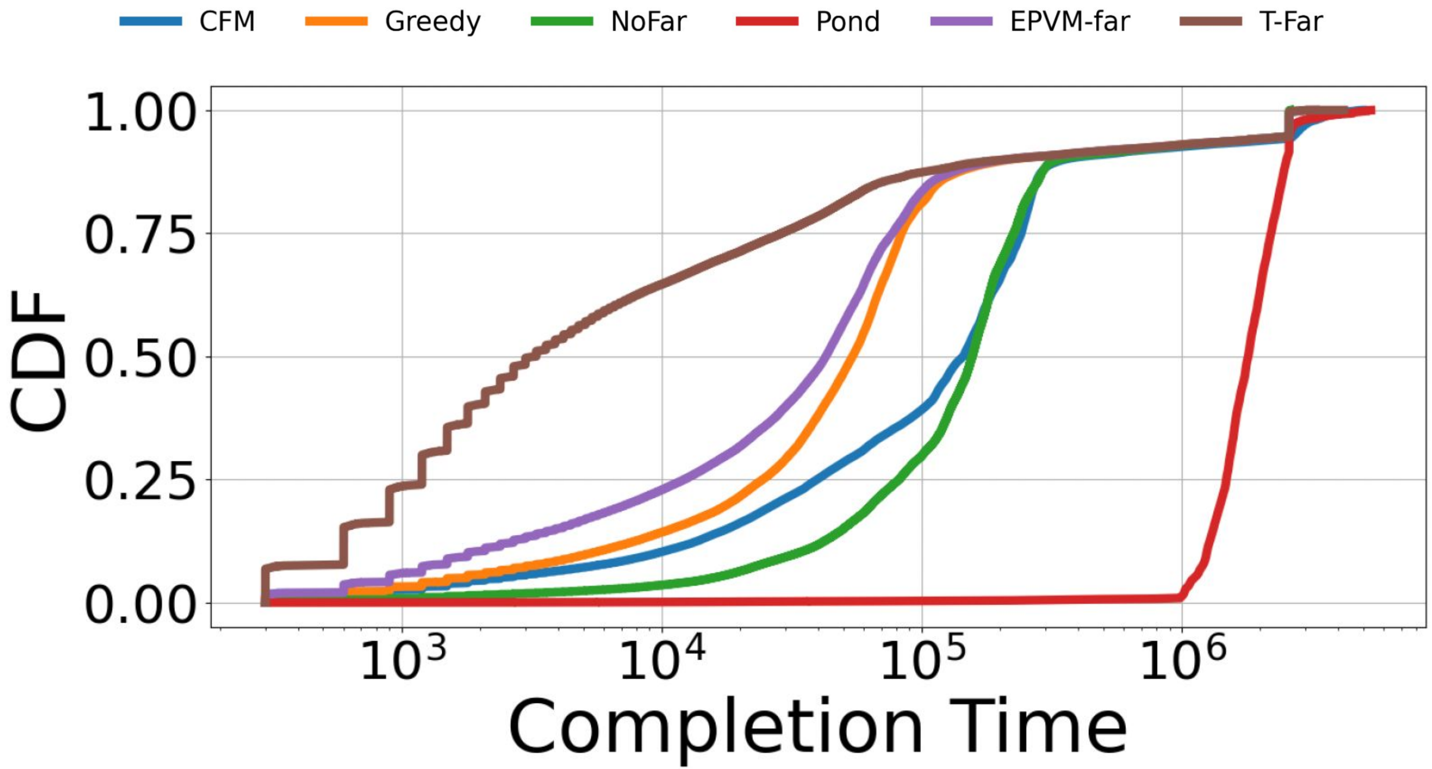
$$b_J = \left(\frac{J_{CPU}}{M_{CPU}} + \frac{J_{Mem\_LM}}{M_{Mem}} + \frac{J_{Mem\_Pool}}{Pool_{Mem}}\right) * J_{exe} * Slowdown(J_{Mem\_LM})$$

$$Score_J = a_J - \left(\frac{\overline{a_J}}{b_J}\right) * b_J$$

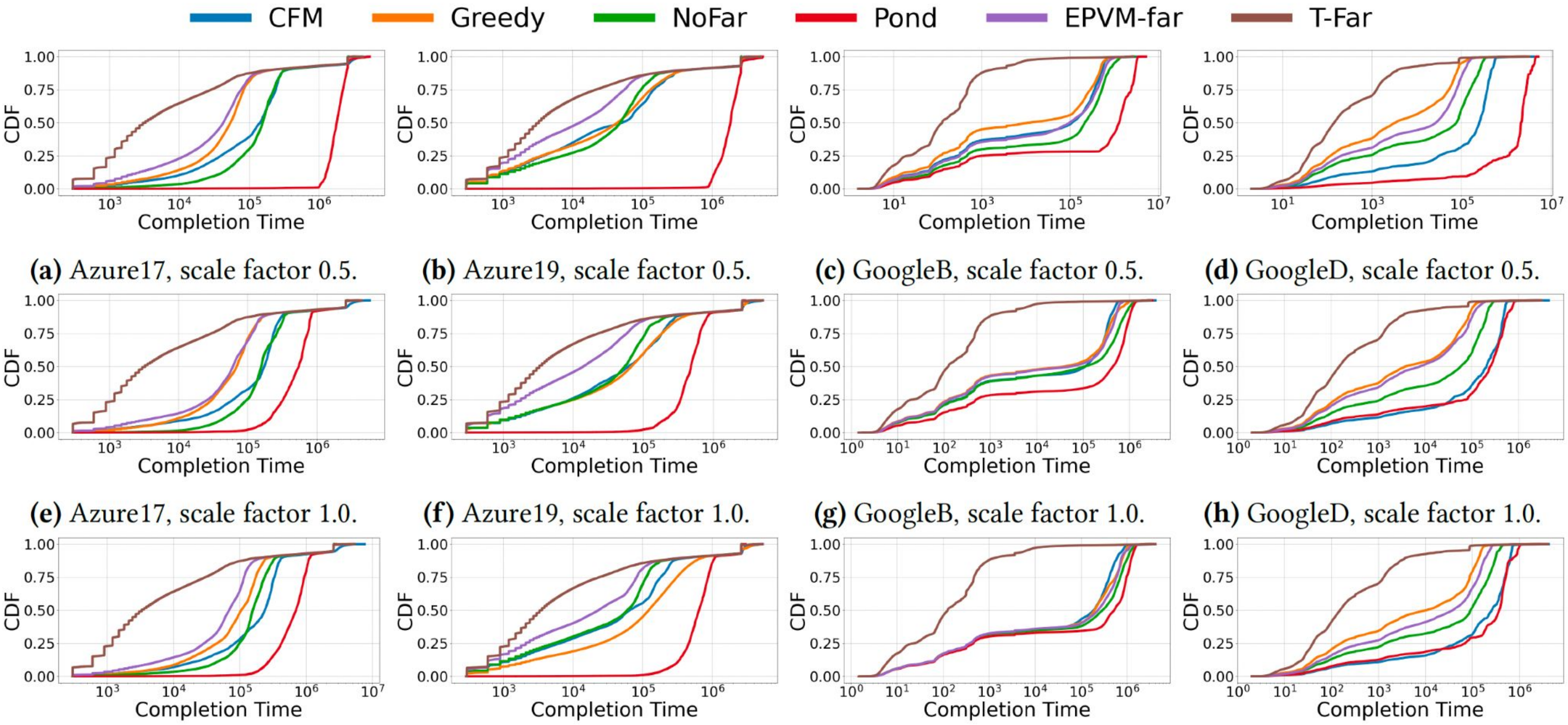
$$S_{max} = max_{J=1}^{P} Score_J$$

b = Job's demands * Job's durations * Job's slowdown

Scheduling Policies

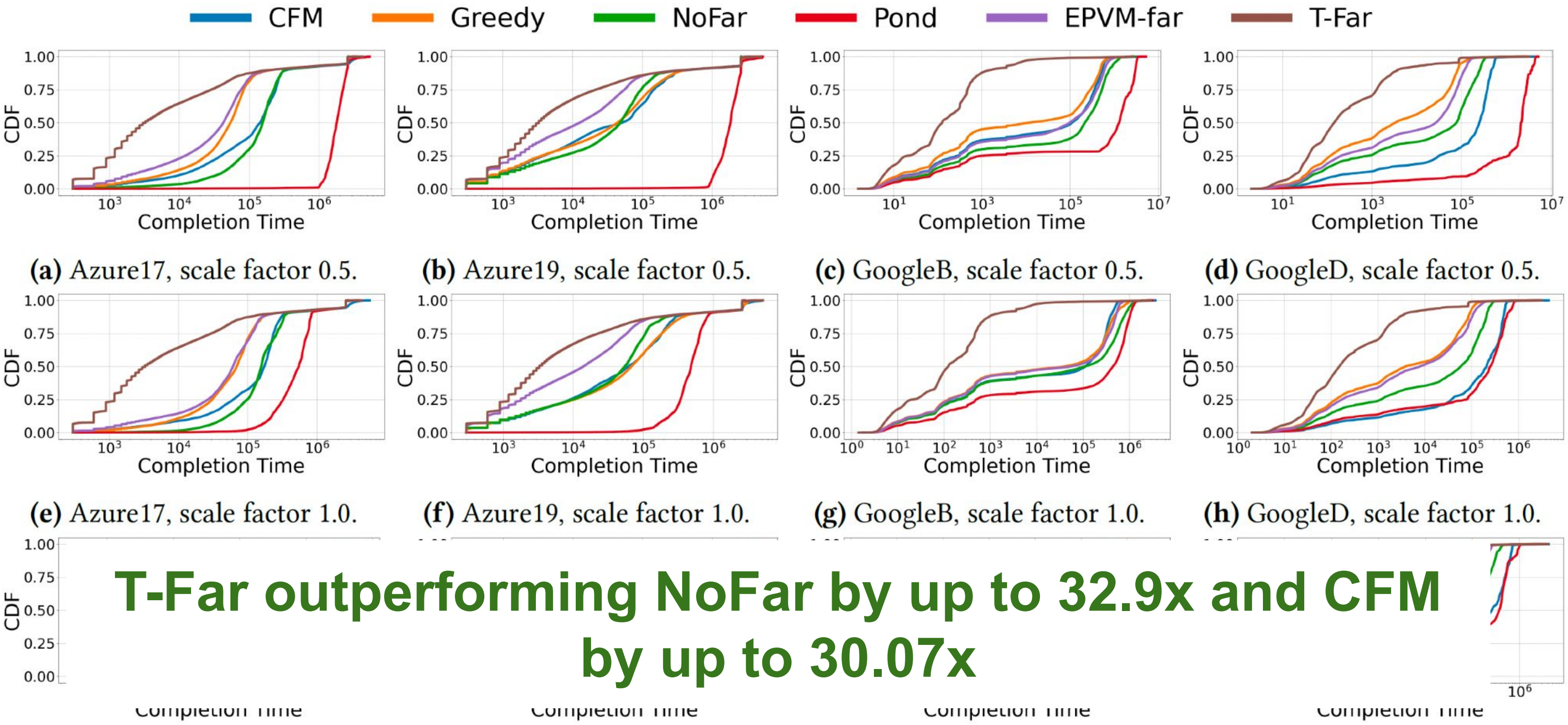


Scheduling Policies



(a) Azure17, scale factor 0.5.
(b) Azure19, scale factor 0.5.
(c) GoogleB, scale factor 0.5.
(d) GoogleD, scale factor 0.5.
(e) Azure17, scale factor 1.0.
(f) Azure19, scale factor 1.0.
(g) GoogleB, scale factor 1.0.
(h) GoogleD, scale factor 1.0.
(i) Azure17, scale factor 2.0.
(j) Azure19, scale factor 2.0.
(k) GoogleB, scale factor 2.0.
(l) GoogleD, scale factor 2.0.

Scheduling Policies



(a) Azure17, scale factor 0.5. (b) Azure19, scale factor 0.5. (c) GoogleB, scale factor 0.5. (d) GoogleD, scale factor 0.5.

(e) Azure17, scale factor 1.0. (f) Azure19, scale factor 1.0. (g) GoogleB, scale factor 1.0. (h) GoogleD, scale factor 1.0.

(i) Azure17, scale factor 2.0. (j) Azure19, scale factor 2.0. (k) GoogleB, scale factor 2.0. (l) GoogleD, scale factor 2.0.

T-Far outperforming NoFar by up to 32.9x and CFM by up to 30.07x

Conclusion

- **Resource Utilization & Performance Optimization**
    - Addressed the need for improved resource management in compute clusters.
    - Leveraged **Compute Express Link (CXL) memory pools** for enhanced resource efficiency.
- **Bede**
    - **Two new schedulers** optimize job placement across memory tiers.
    - **Configuration simulator** finds optimal cluster configurations for maximizing performance improvements
- **What did we learn?**
    - Batch jobs and their completion times in real-world traces
    - More realistic baselines other than FIFO and SJF (adding borg and tetris)
    - Low-utilization in clusters and oversubscription
    - Heterogeneous machines
- **Future work**
    - Alternative far memory technologies (RDMA and SSDs)
    - Real-world prototype
    - OSDI submission