Aalo

Efficient Coflow Scheduling Without Prior Knowledge

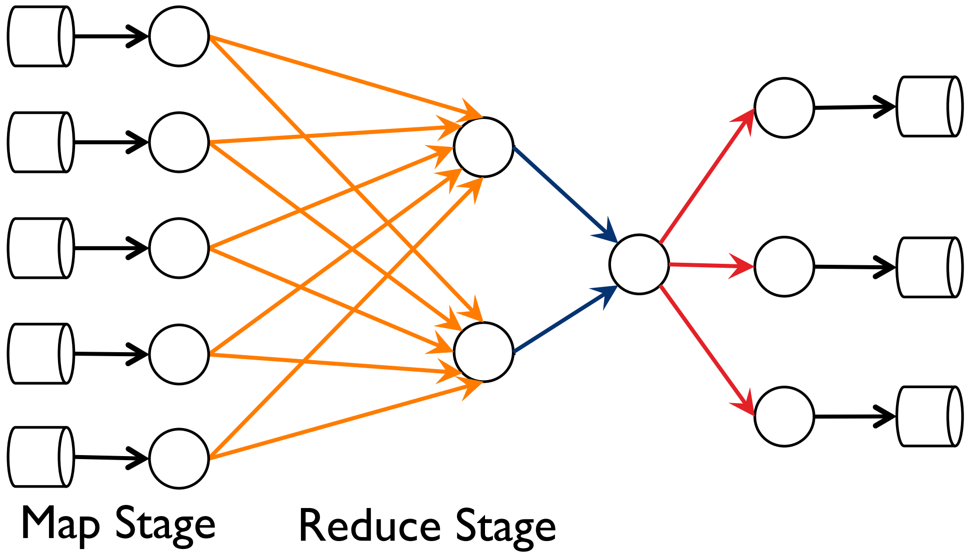
Mosharaf Chowdhury,
Ion Stoica

amplab UC Berkeley

Communication is Crucial

Performance

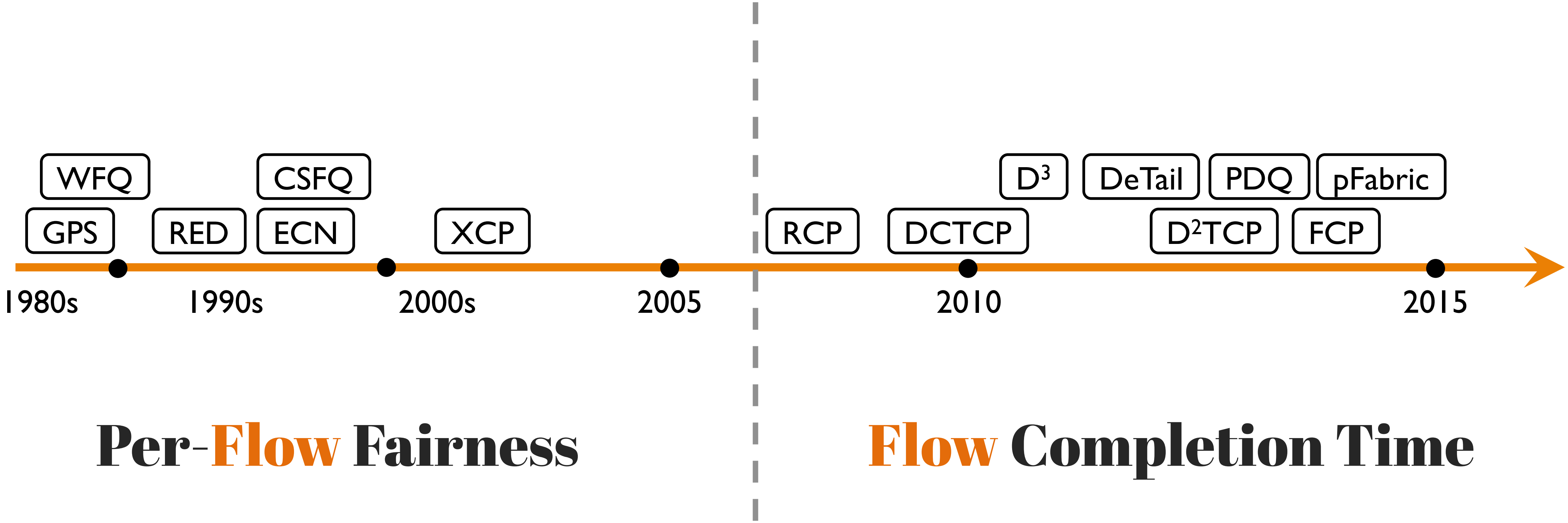
Facebook jobs spend ~**25%** of runtime on *average* in intermediate comm.[1]



As SSD-based and in-memory systems proliferate, the network is likely to become the primary bottleneck

1. Based on a month-long trace with 320,000 jobs and 150 Million tasks, collected from a 3000-machine Facebook production MapReduce cluster.

Flow-Based Solutions

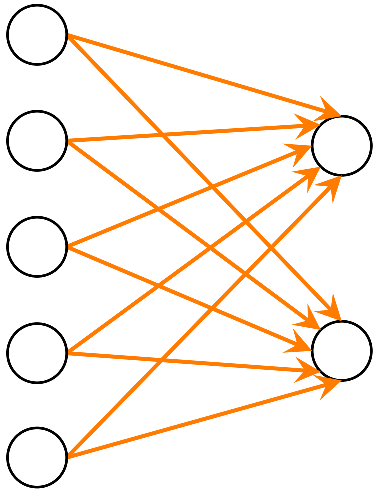


Independent flows cannot capture the collective communication patterns (e.g., shuffle) common in data-parallel applications

Coflow

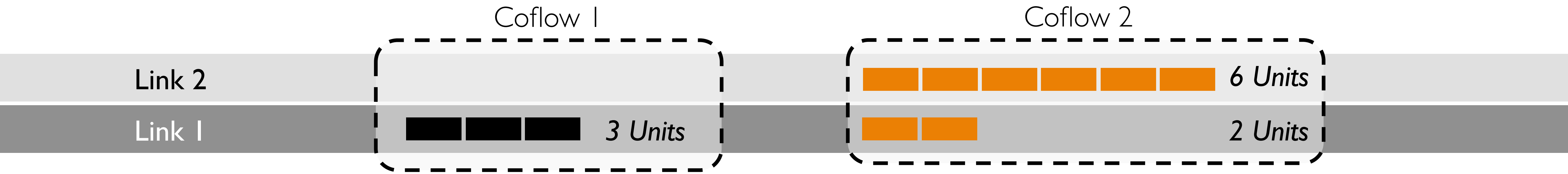
Communication abstraction for data-parallel applications to express their performance goals

1. Minimize completion times,
2. Meet deadlines, or
3. Perform fair allocation

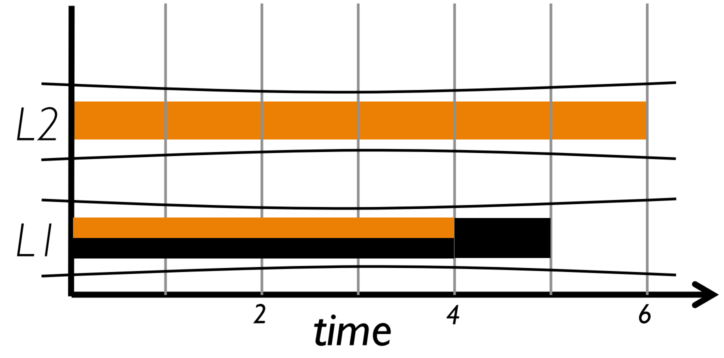


Benefits of Inter-Coflow Scheduling

Benefits of Inter-Coflow Scheduling

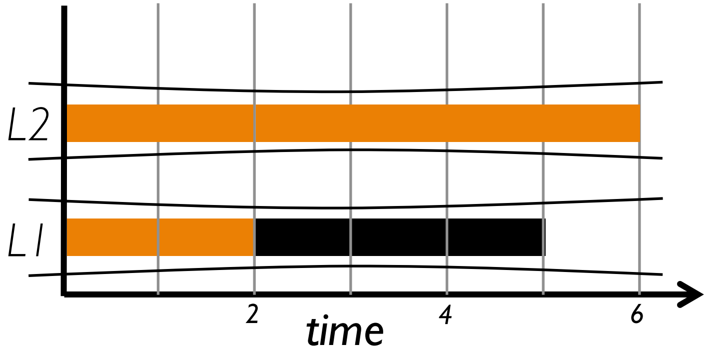


Fair Sharing



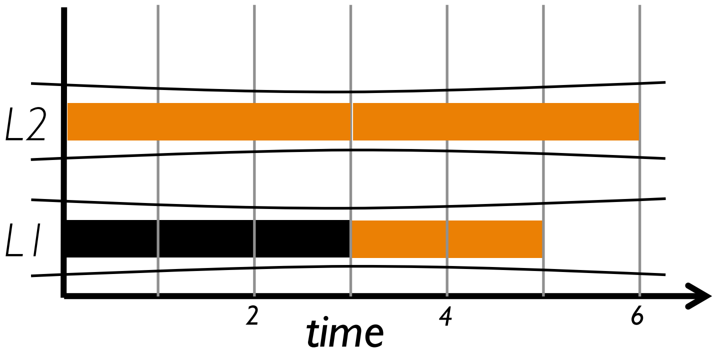
Coflow1 comp. time = 5
Coflow2 comp. time = 6

Smallest-Flow First



*Coflow1 comp. time = **5***
*Coflow2 comp. time = **6***

Smallest-Coflow First



*Coflow1 comp. time = **3***
*Coflow2 comp. time = **6***

Benefits *increases* with the number of coexisting coflows

Varys[1]

Efficiently schedules coflows leveraging complete and future information

1. The size of each flow,
2. The total number of flows, and
3. The endpoints of individual flows

1. Efficient Coflow Scheduling with Varys, SIGCOMM'2014.

Varys

Efficiently schedules coflows leveraging complete and future information

1. ~~The size of each flow,~~
2. ~~The total number of flows, and~~
3. ~~The endpoints of individual flows~~

← Pipelining between stages
← Speculative executions
← Task failures and restarts

Aalo

*Efficiently schedules coflows **without** complete and future information*

1. ~~The size of each flow,~~ ← Pipelining between stages
2. ~~The total number of flows, and~~ ← Speculative executions
3. ~~The endpoints of individual flows~~ ← Task failures and restarts

Coflow Scheduling

Minimize Avg. Comp. Time	Flows on a Single Link
With complete knowledge	Smallest-Flow-First

Coflow Scheduling

Minimize Avg. Comp. Time	Flows on a Single Link	Coflows in the Entire Network
With complete knowledge	Smallest-Flow-First	Varys[1], Smallest-Coflow-First[1]

1. Efficient Coflow Scheduling with Varys, SIGCOMM'2014.

Coflow Scheduling

Minimize Avg. Comp. Time	Flows on a Single Link	Coflows in the Entire Network
With complete knowledge	Smallest-Flow-First	Varys[1], Smallest-Coflow-First[1]
Without complete knowledge	Least-Attained Service (LAS)	

LAS: prioritize flow that has sent the least amount of data

1. *Efficient Coflow Scheduling with Varys, SIGCOMM'2014.*

Coflow-Aware LAS (CLAS)

Prioritize coflow that has sent the least total number of bytes

- The more a coflow has sent, the lower its priority
- Smaller coflows finish faster

Coflow-Aware LAS (CLAS)

Prioritize coflow that has sent the least total number of bytes

- The more a coflow has sent, the lower its priority
- Smaller coflows finish faster

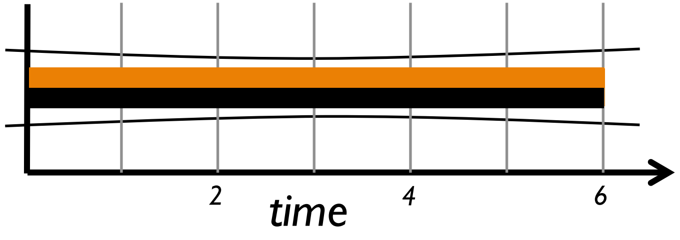
Challenges (also shared by LAS)

- Can lead to starvation
- Suboptimal for similar size coflows

Suboptimal for Similar Coflows

Reduces to fair sharing

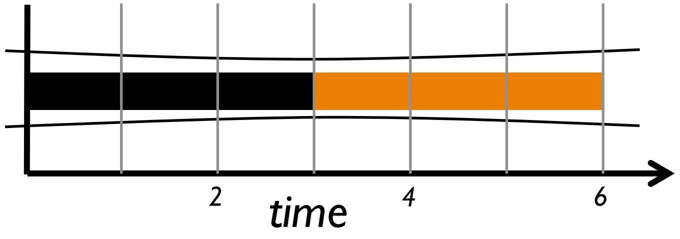
- Doesn't minimize average completion time



Coflow1 comp. time = 6
Coflow2 comp. time = 6

FIFO works well for similar coflows

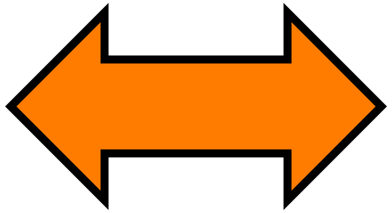
- Optimal when cflows are identical



Coflow1 comp. time = ***3***
Coflow2 comp. time = ***6***

Between a "Rock" and a "Hard Place"

Prioritize across dissimilar coflows



FIFO schedule similar coflows

Discretized Coflow-Aware LAS (D-CLAS)

Priority discretization

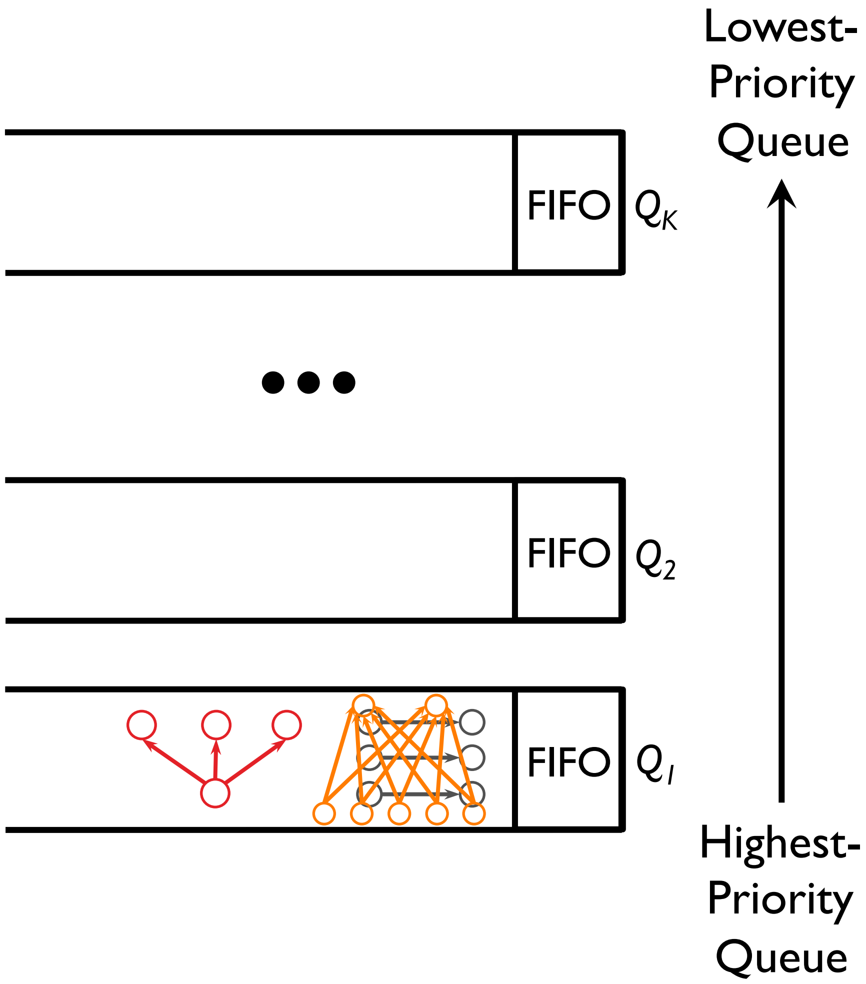
- Change priority when total # of bytes sent exceeds predefined thresholds

Scheduling policies

- FIFO within the same queue
- Prioritization across queue

Weighted sharing across queues

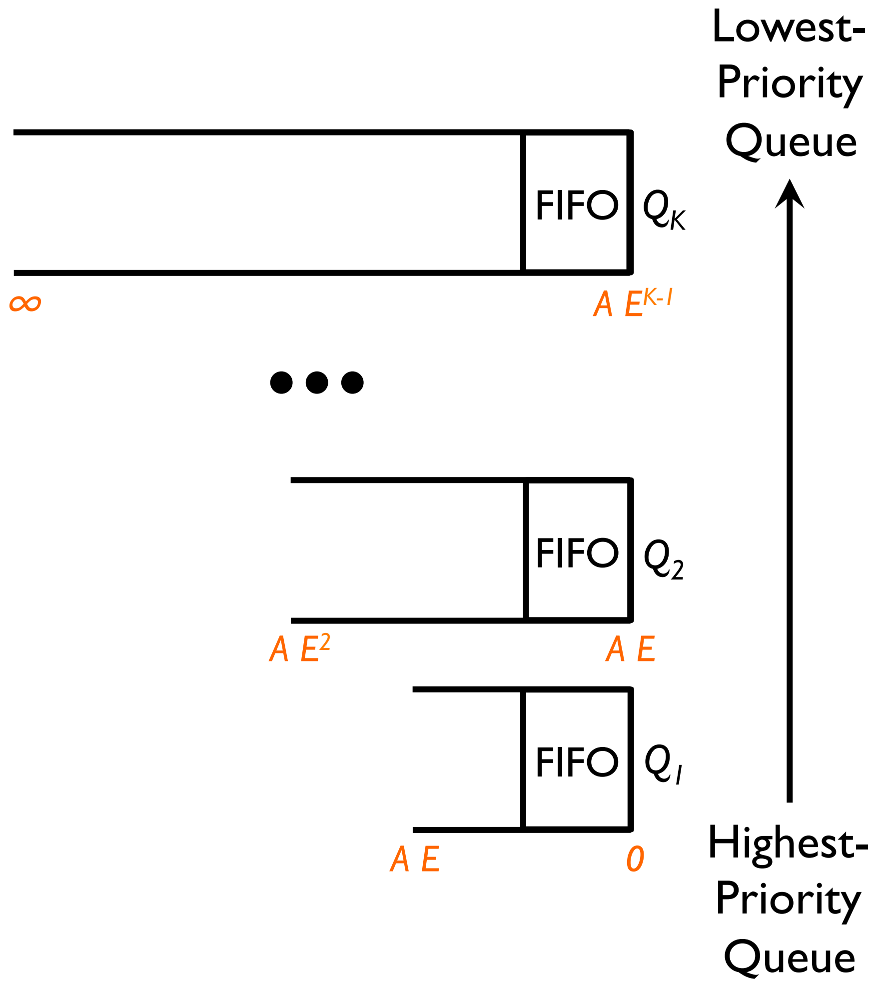
- Guarantees starvation avoidance



How to Discretize Priorities?

Exponentially spaced thresholds: $A \times E^i$

- A, E : constants
- $1 \leq i \leq K$: threshold constant
- K : number of the queues



Computing *Total* # of Bytes Sent

D-CLAS requires to know total # of bytes sent over all flows of a coflow

- Distributed computation over small time scales → challenging

Computing *Total* # of Bytes Sent

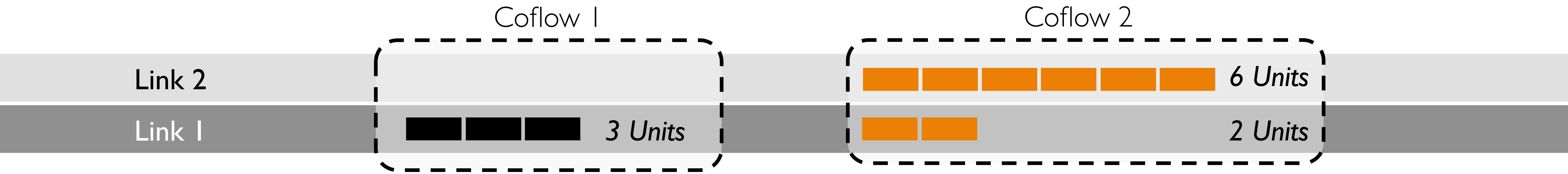
D-CLAS requires to know total # of bytes sent over all flows of a coflow

- Distributed computation over small time scales → challenging

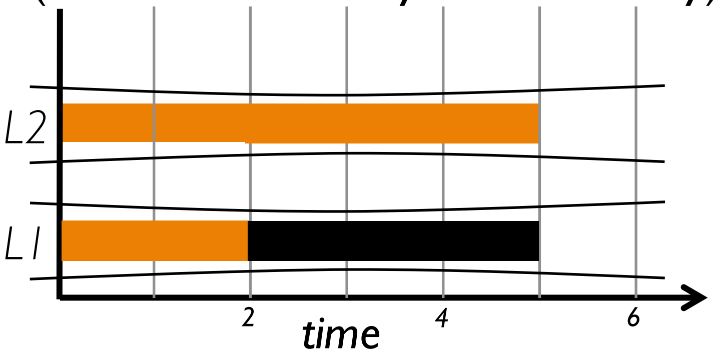
How much do we loose if we don't compute total # of bytes sent?

- D-LAS: make decisions based on total number of bytes sent *locally*

D-LAS Far From Optimal!

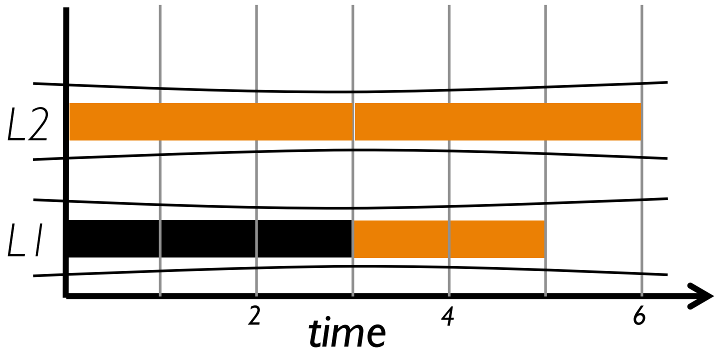


D-LAS
(decision on # of bytes sent locally)



Coflow1 comp. time = 6
Coflow2 comp. time = 6

D-CLAS



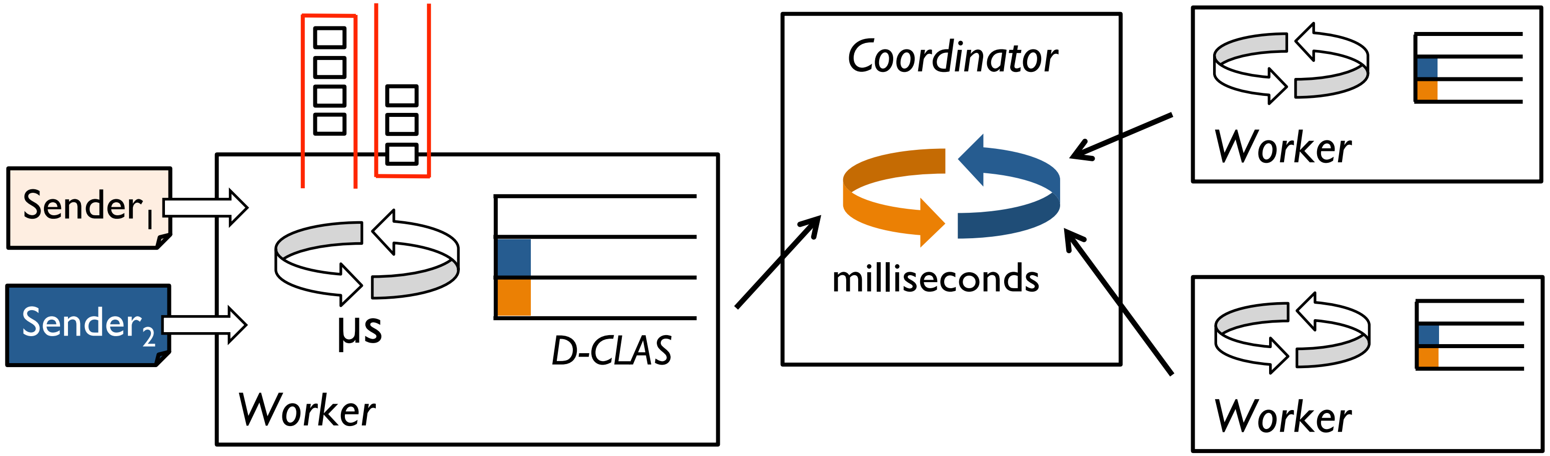
Coflow1 comp. time = ***3***
Coflow2 comp. time = ***6***

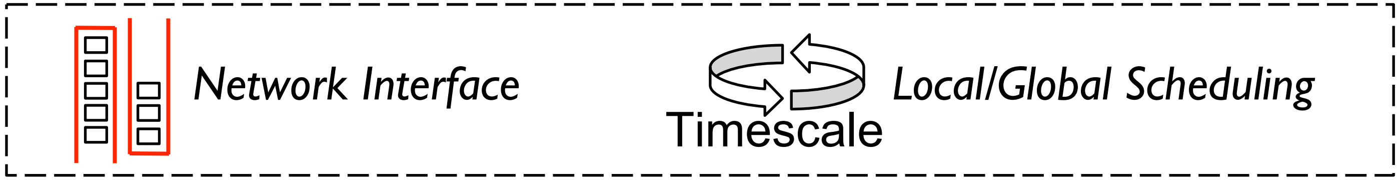
Aalo

*Efficiently schedules coflows **without** complete and future information*

1. Implement D-CLAS using a centralized architecture
2. Expose a **non-blocking** coflow API

Aalo Architecture





Details

Non-blocking: when a new coflow arrives at an output port

- Put its flow(s) in lowest priority queue and schedule them immediately
- No need to sync all flows of a coflow as in Varys

Details

Non-blocking: when a new coflow arrives at an output port

- Put its flow(s) in lowest priority queue and schedule them immediately
- No need to sync all flows of a coflow as in Varys

Compute total number of bytes sent

- Workers send info about active coflows periodically
- Coordinator computes total # of bytes sent, and relay this info back to workers
- Workers use this info to move coflows across queues

Minimal overhead for small flows

Evaluation

A 3000-machine trace-driven simulation matched against a 100-machine EC2 deployment

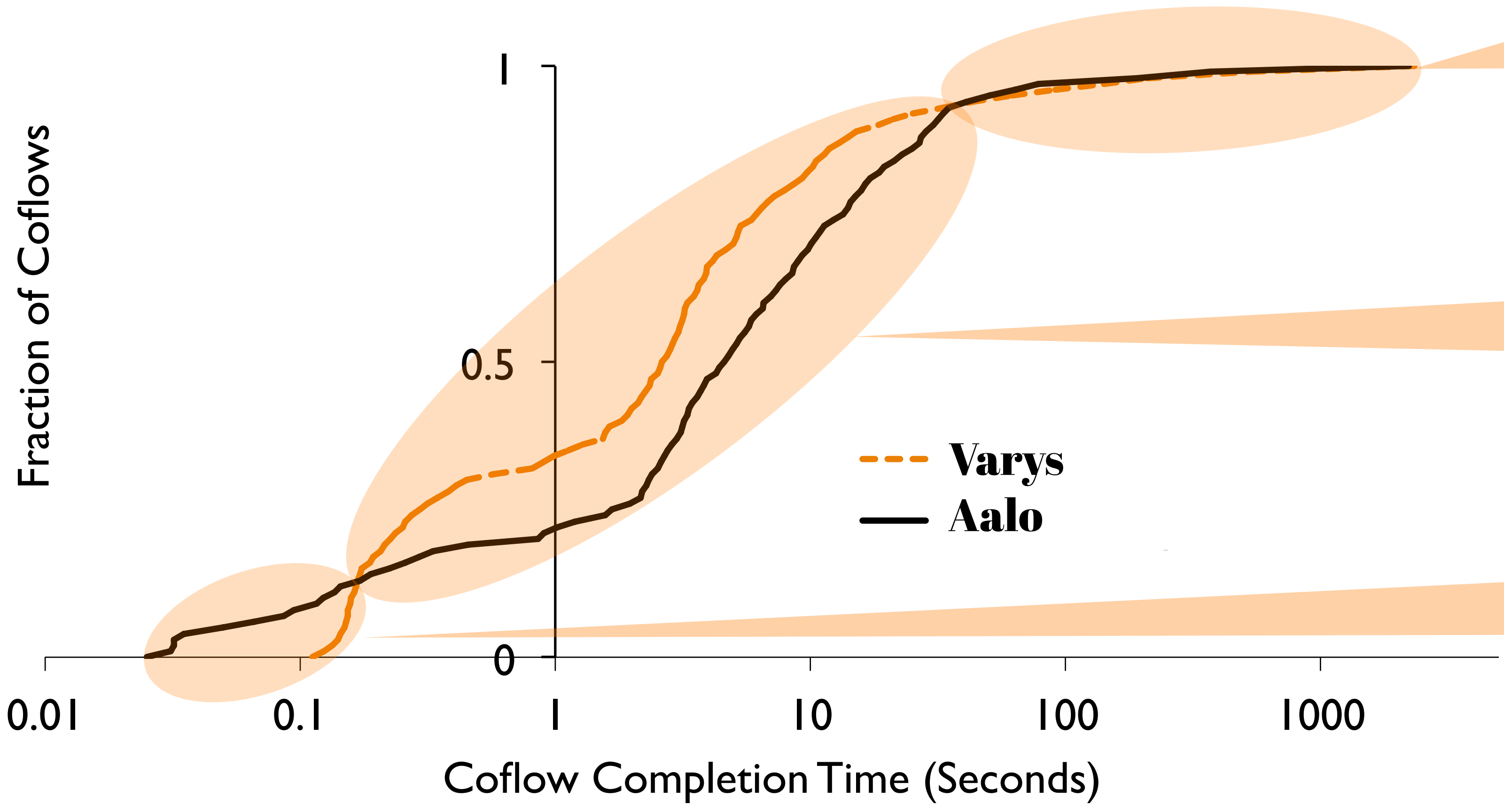
1. Can it approach clairvoyant solutions?
2. Can it scale gracefully?

YES

On Par with Clairvoyant Approaches [EC2]

	Comm. Improv.	Job Improv.
Per-Flow	**1.93X**	**1.18X**
Varys	**0.89X**	**0.91X**

Performance Breakdown [EC2]

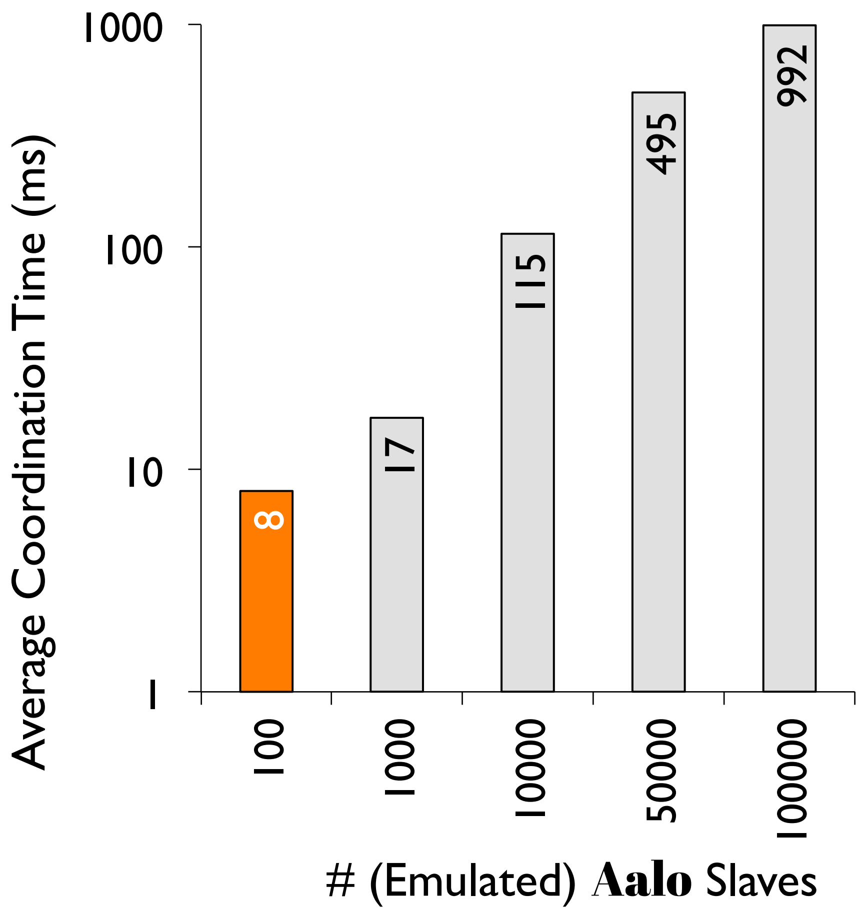


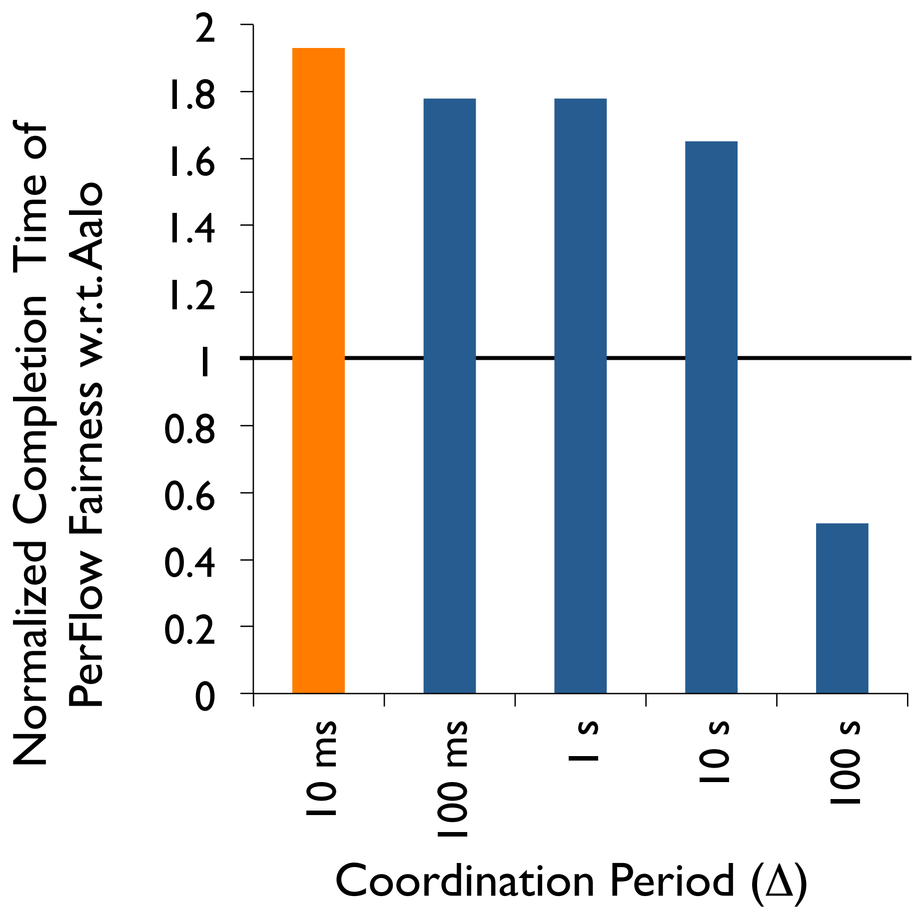
Similar for large coflows because they are in slow-moving queues

Performance loss for medium coflows by misscheduling them

Improvements for small coflows by avoiding coordination

What About Scalability? [EC2]





Aalo

*Efficiently schedules coflows **without** complete information*

- Makes coflows practical in presence of failures and DAGs
- Improved performance over flow-based approaches
- Provides a simple, non-blocking API

https://github.com/coflow

Mosharaf Chowdhury – mosharaf@umich.edu