Coflow

Mending the Application-Network Gap in Big Data Analytics

Mosharaf Chowdhury

amplab UC Berkeley

Big Data

The volume of data businesses want to *make sense of* is increasing

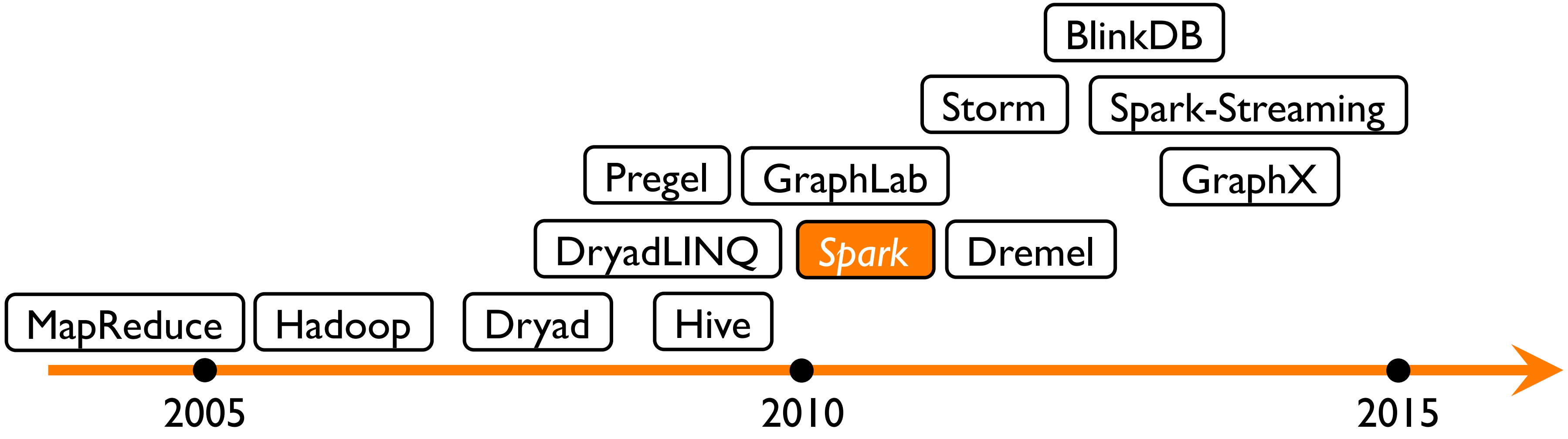
Increasing variety of sources

- Web, mobile, wearables, vehicles, scientific, …

Cheaper disks, SSDs, and memory

Stalling processor speeds

Big Datacenters for Massive Parallelism



Data-Parallel Applications

Multi-stage dataflow

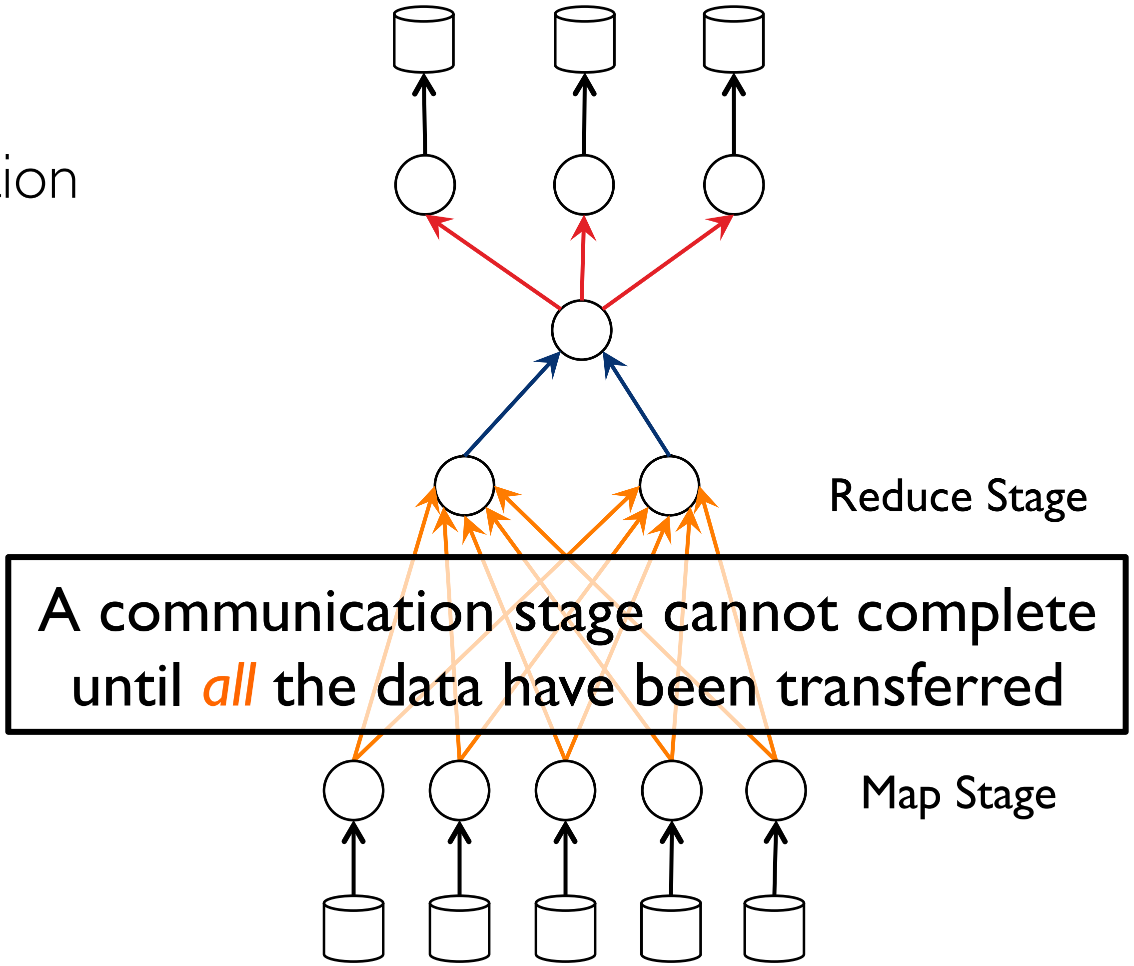
- Computation interleaved with communication

Computation Stage (e.g., Map, Reduce)

- Distributed across many machines
- Tasks run in parallel

Communication Stage (e.g., Shuffle)

- Between successive computation stages



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

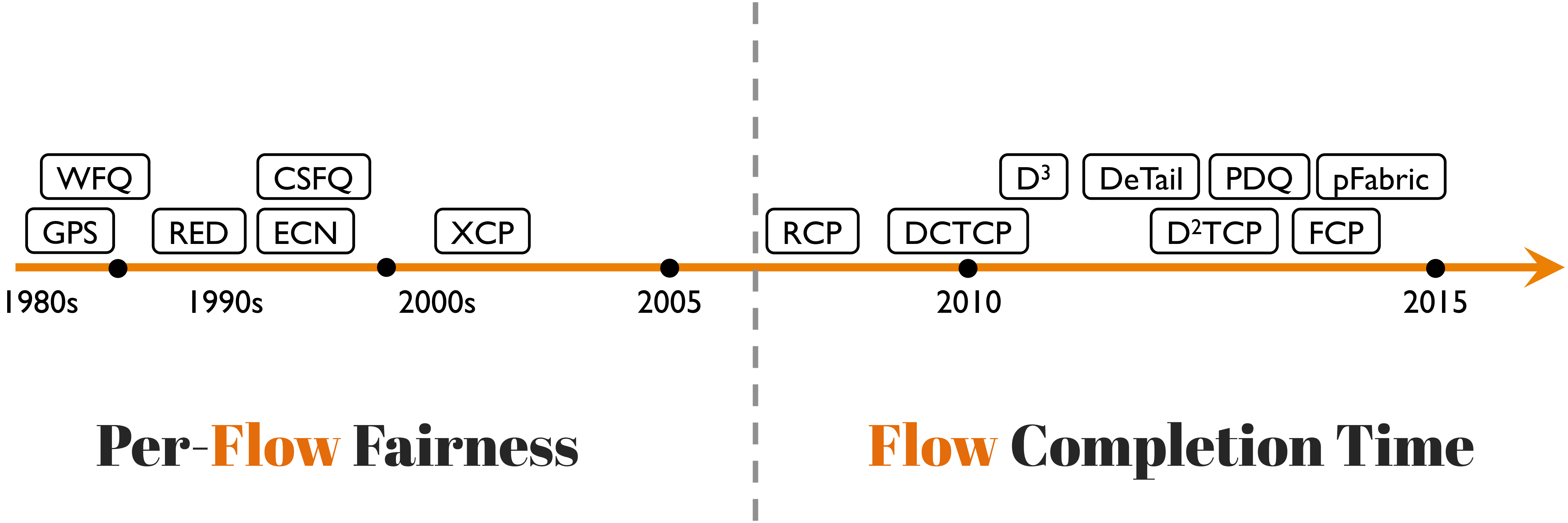
Transfers data from a source to a destination

Independent unit of allocation, sharing, load balancing, and/or prioritization

Faster Communication Stages: Networking Approach

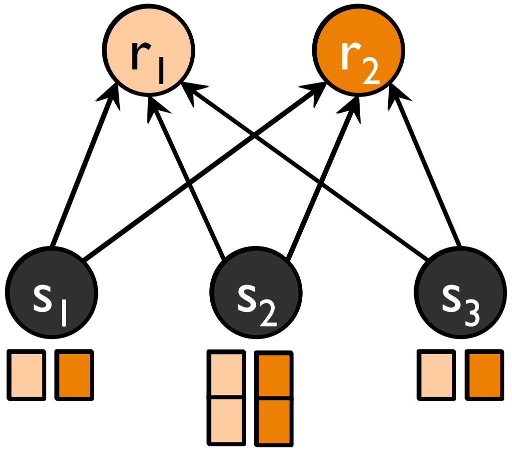
"Configuration should be handled at the system level"

Existing Solutions

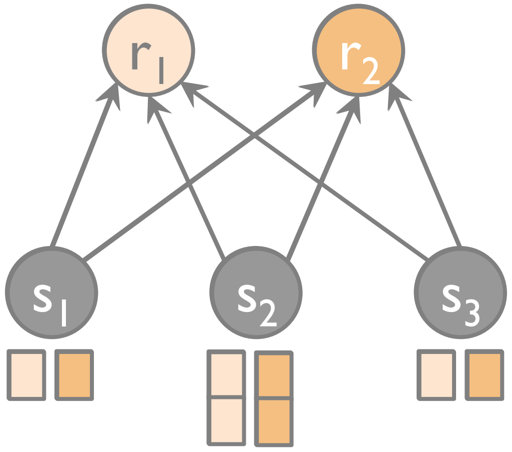


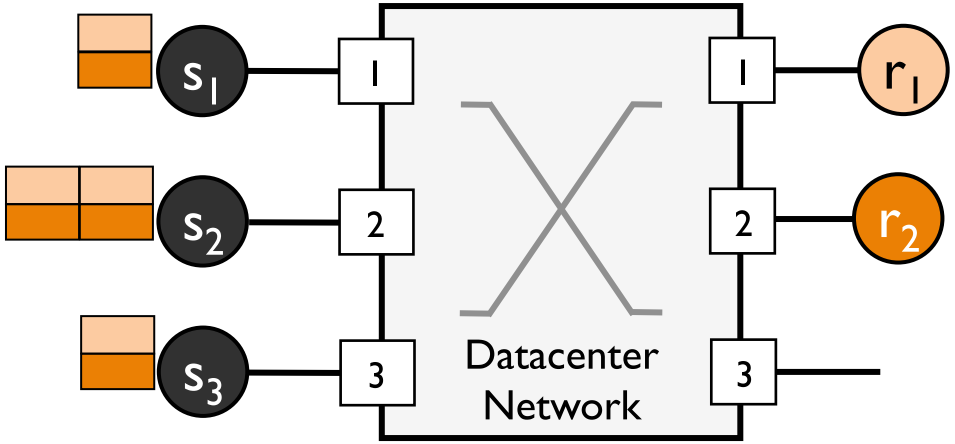
Independent flows cannot capture the collective communication behavior common in data-parallel applications

Why Do They Fall Short?

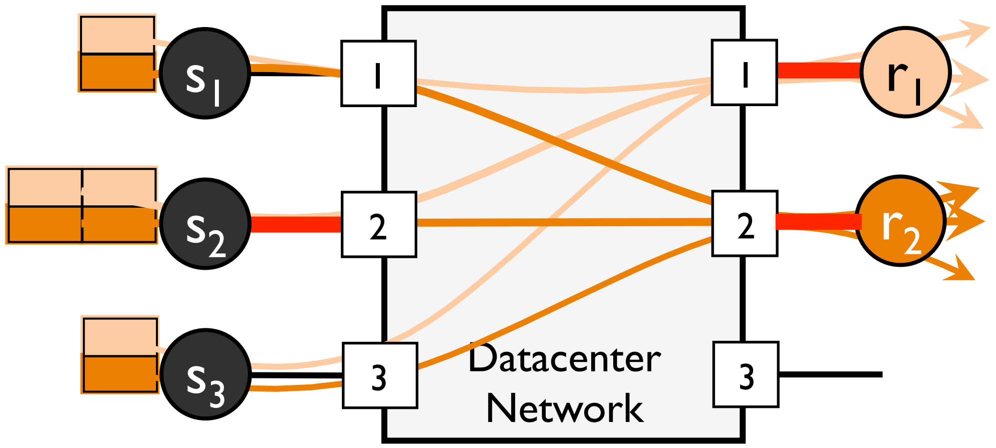


Why Do They Fall Short?

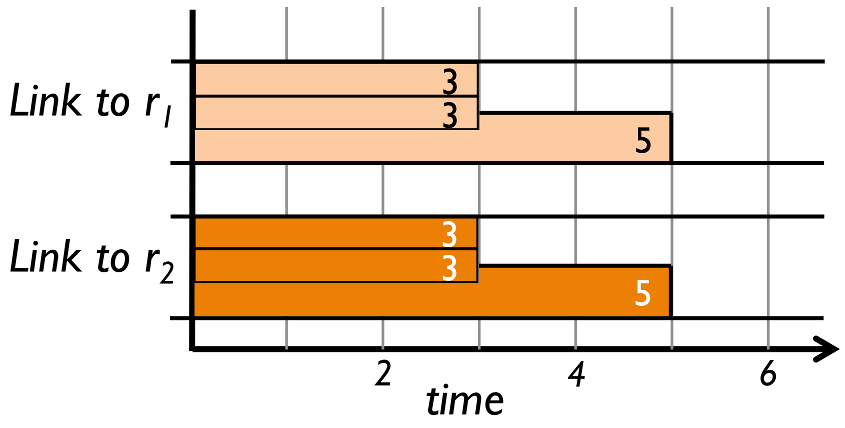




Why Do They Fall Short?



Per-Flow Fair Sharing

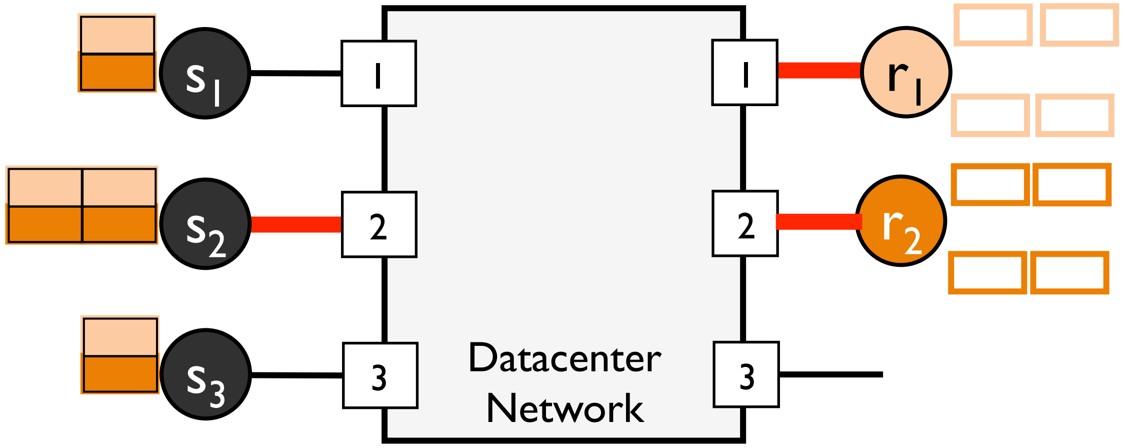


Shuffle Completion Time = **5**

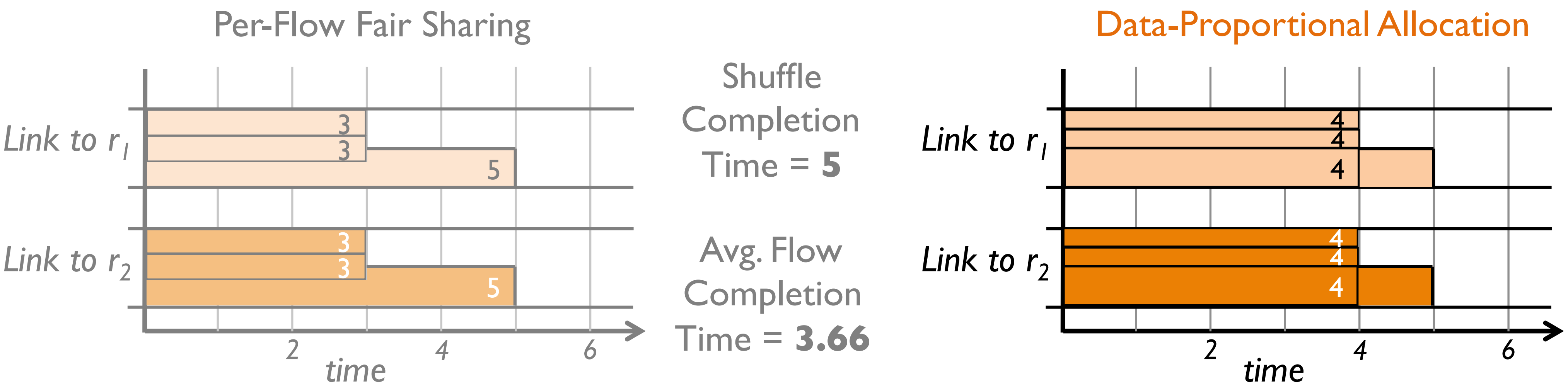
Avg. Flow Completion Time = **3.66**

Solutions focusing on flow completion time cannot further decrease the shuffle completion time

Improve Application-Level Performance[1]



Slow down faster flows to *accelerate* slower flows



Per-Flow Fair Sharing: Shuffle Completion Time = **5**; Avg. Flow Completion Time = **3.66**

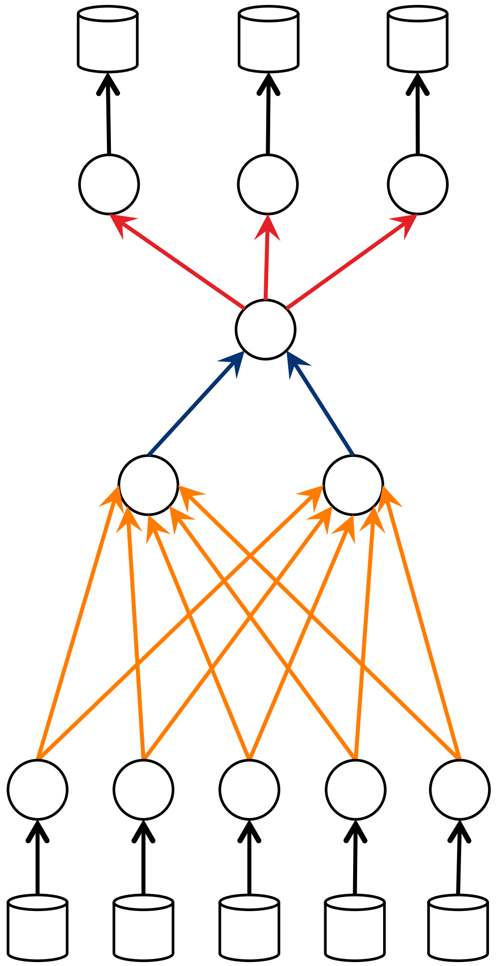
Data-Proportional Allocation: Shuffle Completion Time = **4**; Avg. Flow Completion Time = **4**

1. Managing Data Transfers in Computer Clusters with Orchestra, SIGCOMM'2011.

Faster Communication Stages: Systems Approach

"Configuration should be handled by the end users"

Applications know their performance goals, but they have no means to let the network know



Faster Communication Stages: **Systems Approach**

"Configuration should be handled by the end users"

MIND THE GAP

Faster Communication Stages: **Networking Approach**

"Configuration should be handled at the system level"

MEND THE GAP

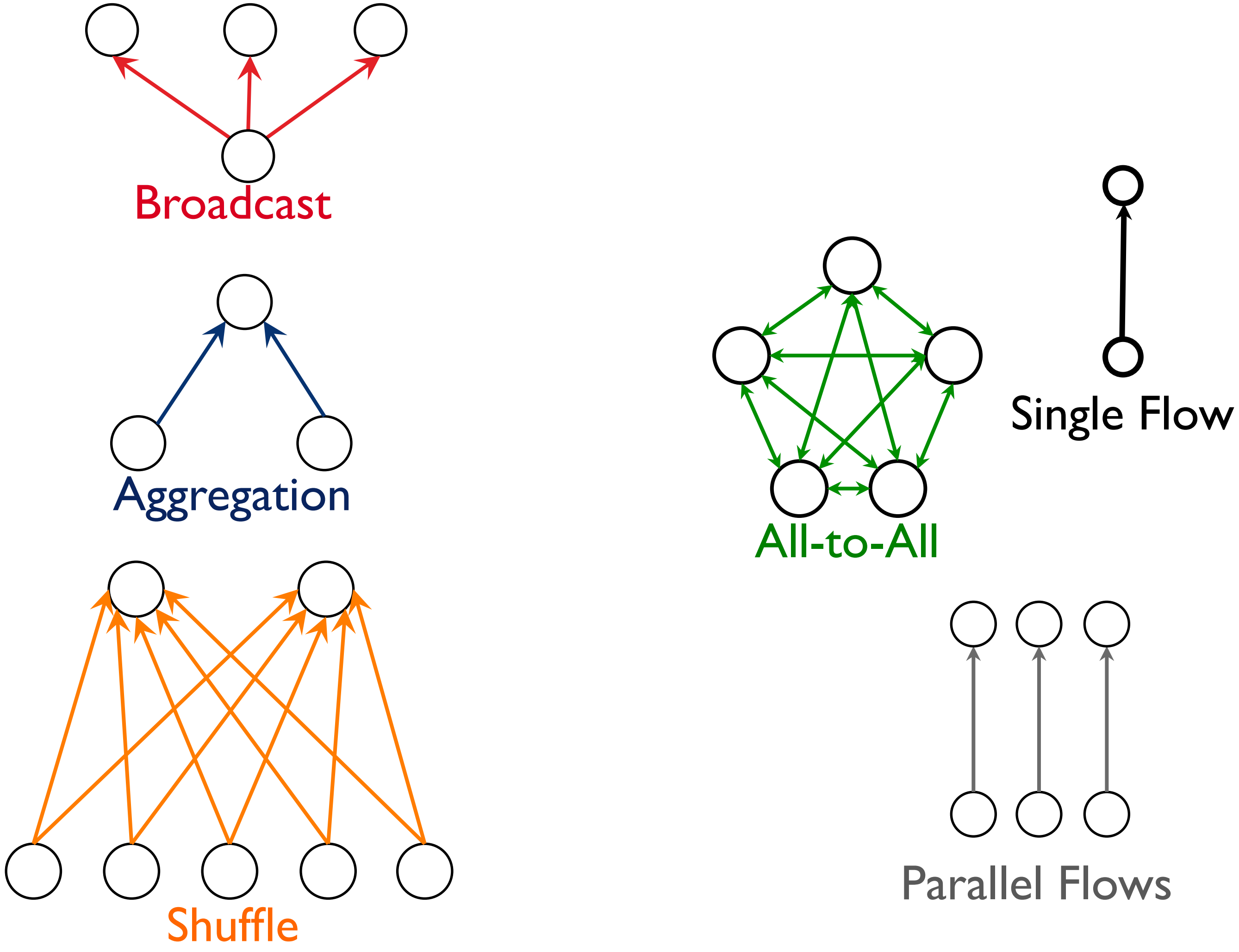
Holistic Approach

Applications and the Network Working Together

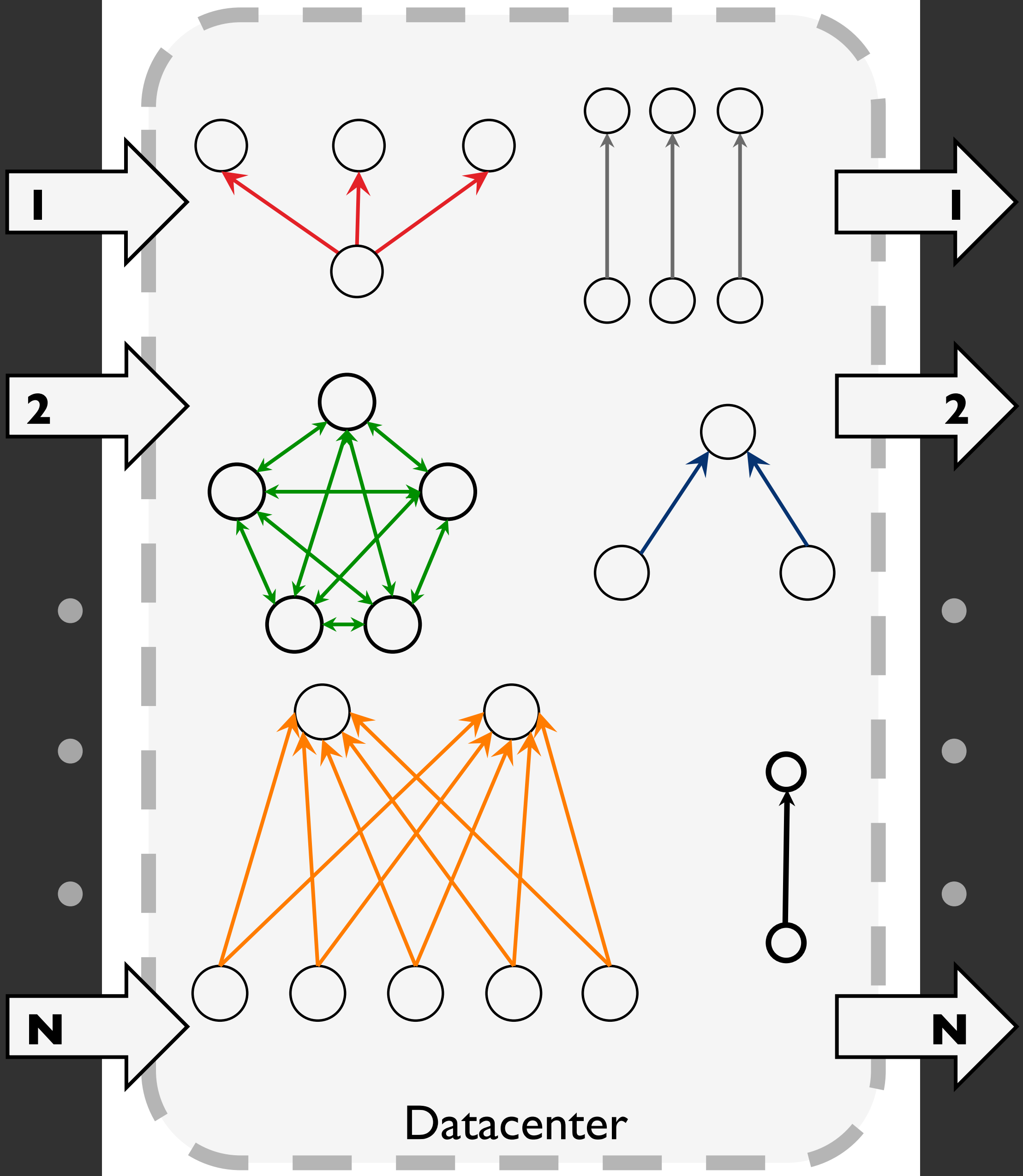
Coflow

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

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


Broadcast
Aggregation
Shuffle
All-to-All
Single Flow
Parallel Flows

How to schedule coflows online ...



#1 ... for faster completion of coflows?

#2 ... to meet more deadlines?

#3 ... for fair allocation of the network?

Varys[1]

Enables coflows in data-intensive clusters

1. Coflow Scheduler
2. Global Coordination
3. The Coflow API

Faster, application-aware data transfers throughout the network

Consistent calculation and enforcement of scheduler decisions

Decouples network optimizations from applications, relieving developers and end users

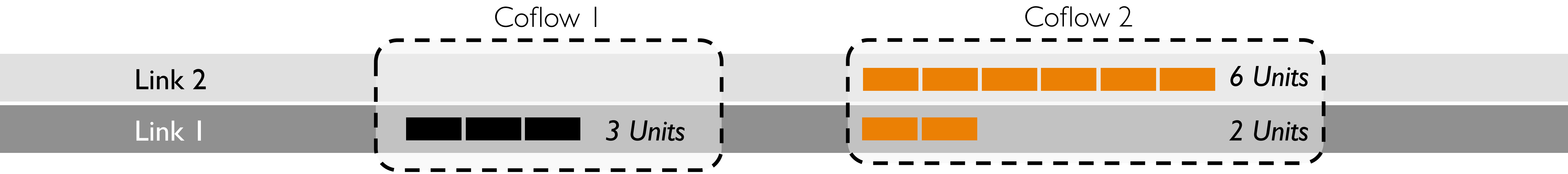
1. Efficient Coflow Scheduling with Varys, SIGCOMM'2014.

Coflow

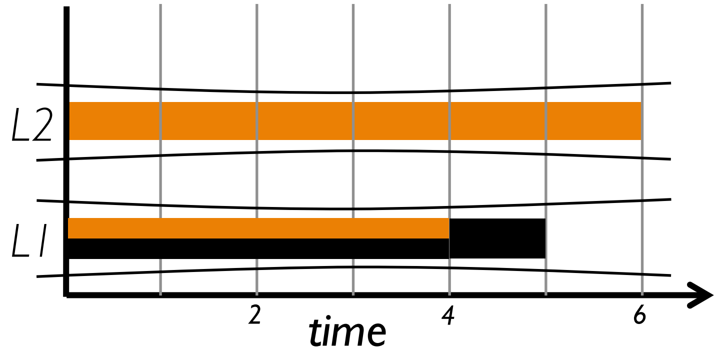
Communication abstraction for data-parallel applications to express their performance goals

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

Benefits of Inter-Coflow Scheduling

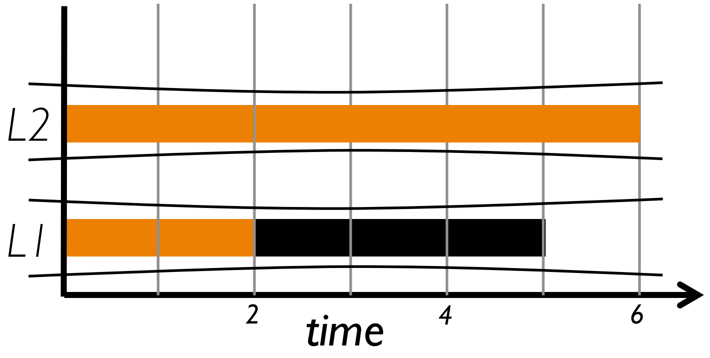


Fair Sharing



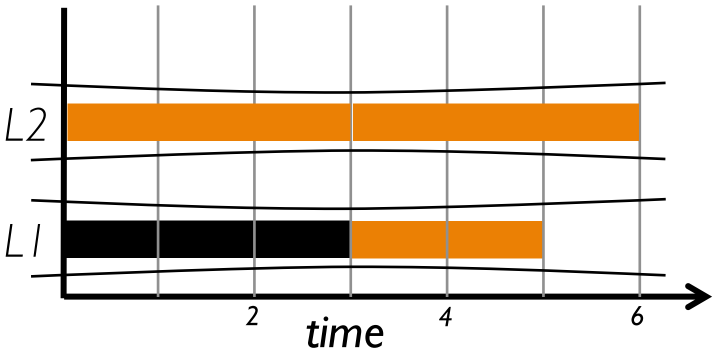
Coflow1 comp. time = 5
Coflow2 comp. time = 6

Smallest-Flow First[1,2]



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

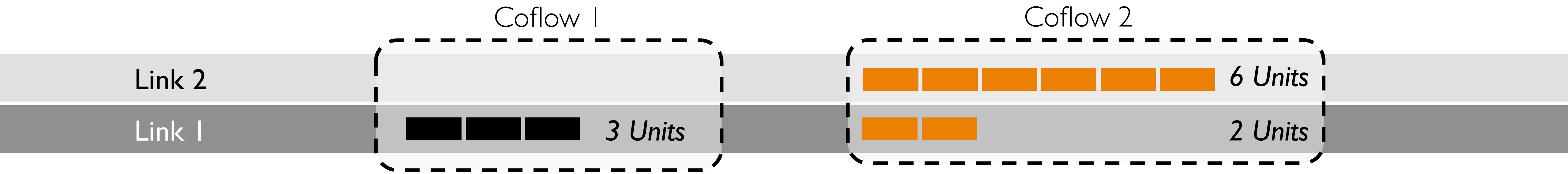
The Optimal



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

1. Finishing Flows Quickly with Preemptive Scheduling, SIGCOMM'2012.
2. pFabric: Minimal Near-Optimal Datacenter Transport, SIGCOMM'2013.

Inter-Coflow Scheduling is **NP-Hard**

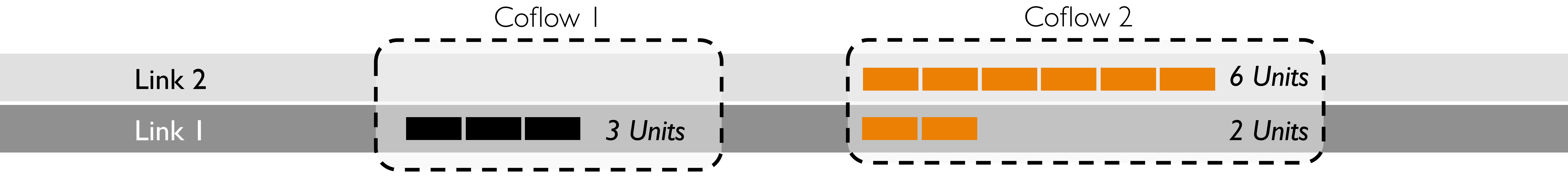


Concurrent Open Shop Scheduling[1]

- Examples include job scheduling and caching blocks
- Solutions use a **ordering** heuristic

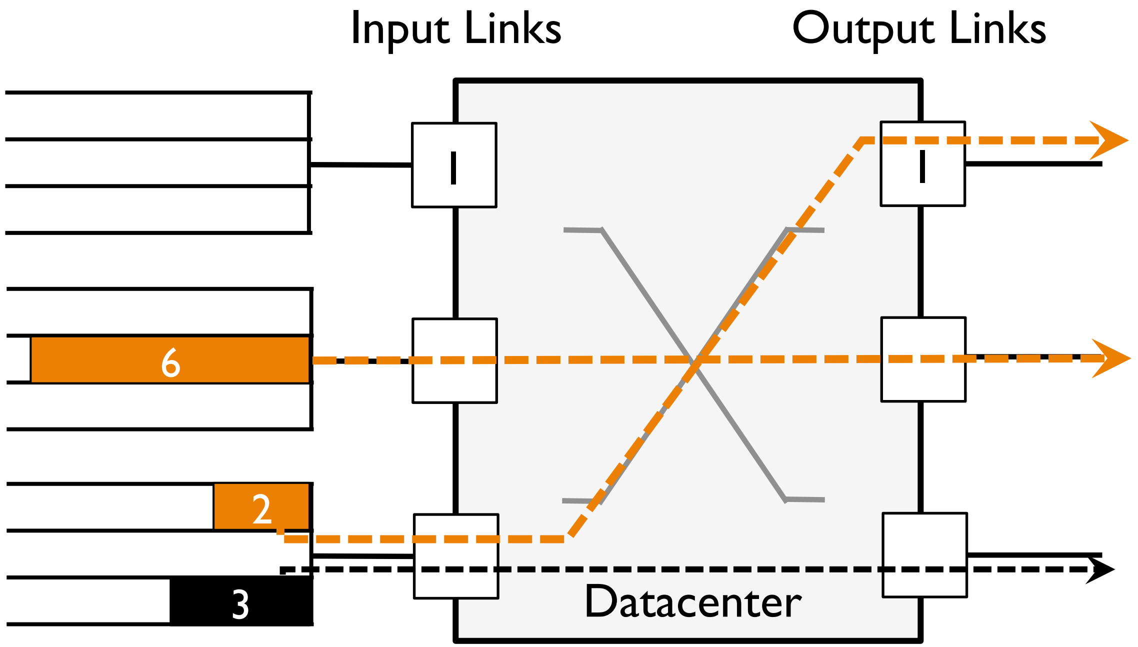
1. Finishing Flows Quickly with Preemptive Scheduling, SIGCOMM'2012.
2. pFabric: Minimal Near-Optimal Datacenter Transport, SIGCOMM'2013.

Inter-Coflow Scheduling is NP-Hard



Concurrent Open Shop Scheduling
with Coupled Resources

- Examples include job scheduling and caching blocks
- Solutions use a **ordering** heuristic
- Consider **matching** constraints



Varys

Employs a two-step algorithm to minimize coflow completion times

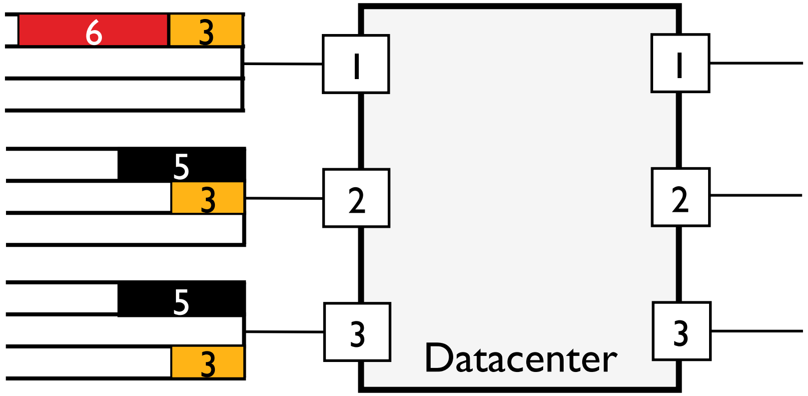
1. Ordering heuristic

   Keep an ordered list of coflows to be scheduled, preempting if needed

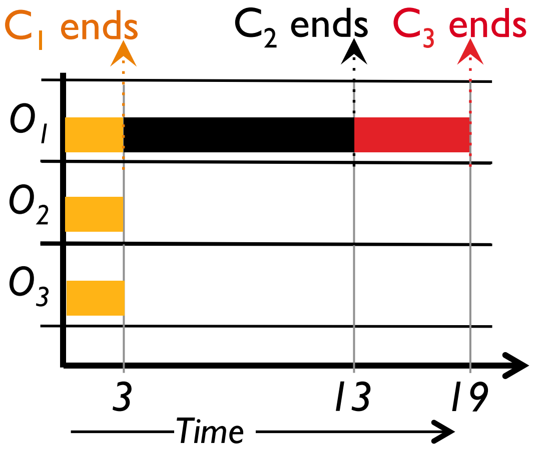
2. Allocation algorithm

   Allocates minimum required resources to each coflow to finish in minimum time

Ordering Heuristic



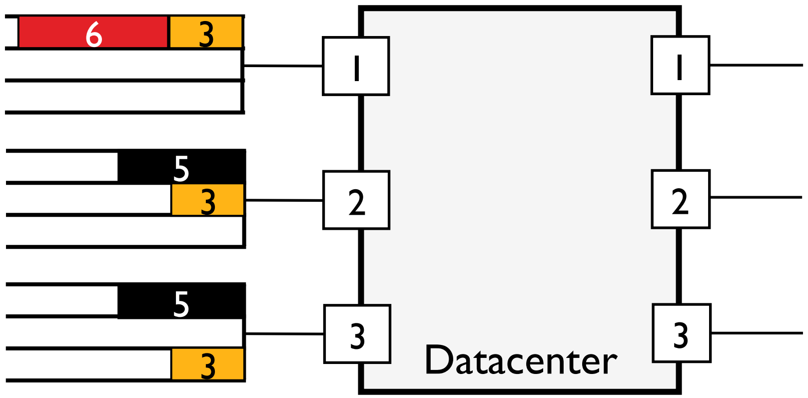
	C_1	C_2	C_3
Length	3	5	6



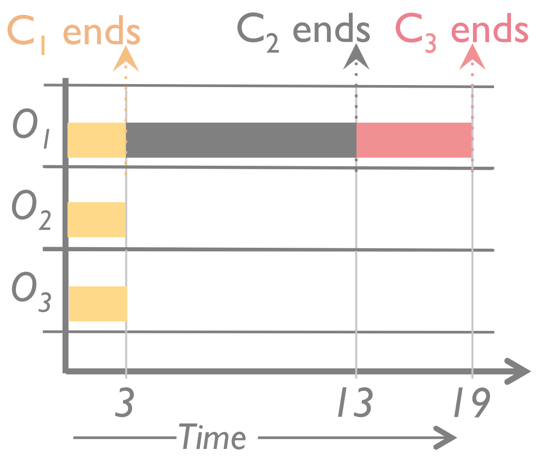
Shortest-First

(Total CCT = 35)

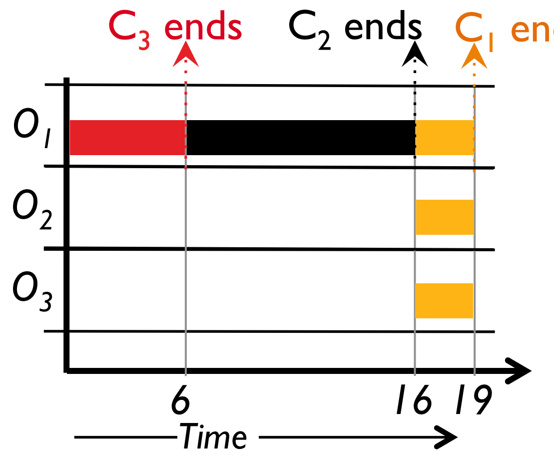
Ordering Heuristic



	C_1	C_2	C_3
Width	3	2	1



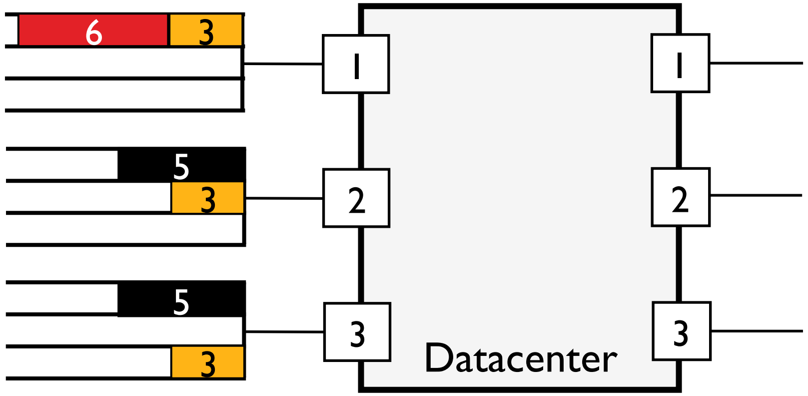
Shortest-First **(35)**



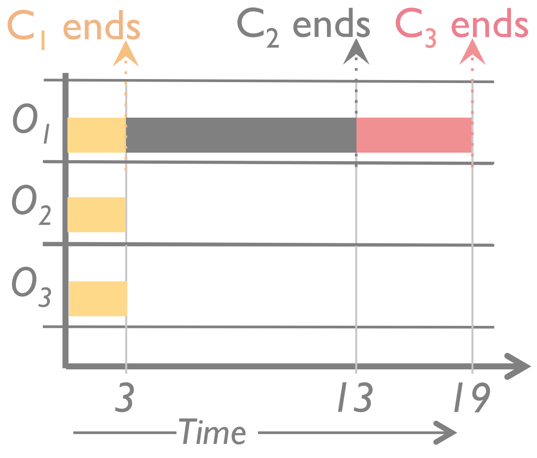
Narrowest-First

(Total CCT = 41)

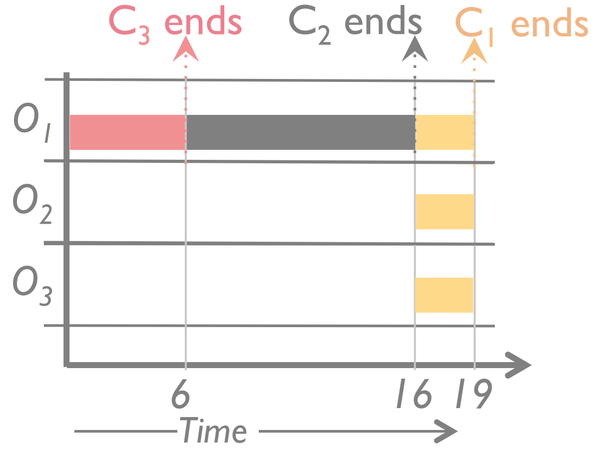
Ordering Heuristic



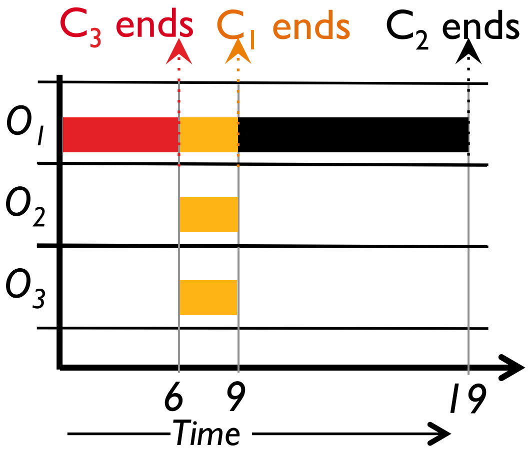
	C_1	C_2	C_3
Size	9	10	6



Shortest-First (35)

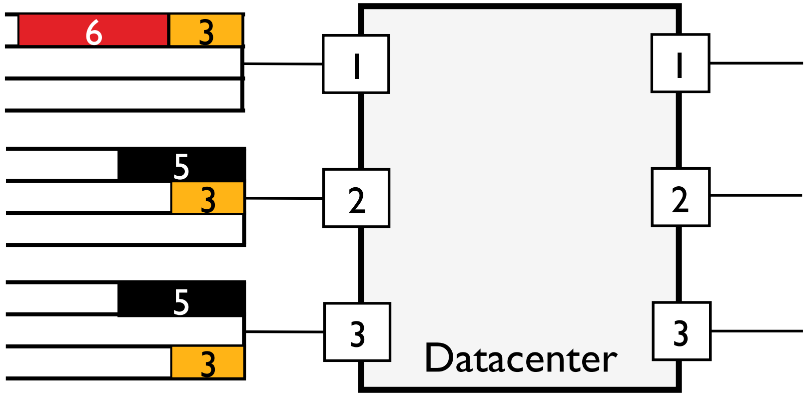


Narrowest-First (41)

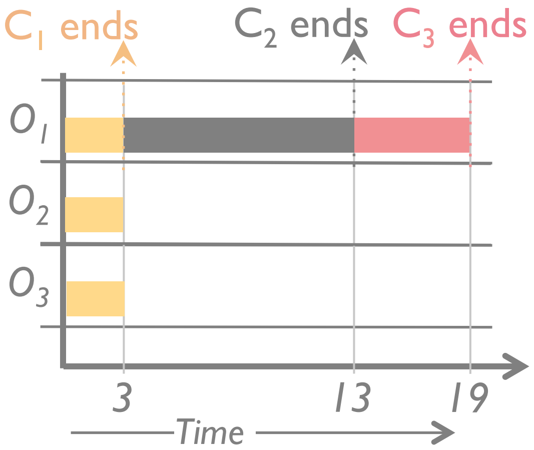


Smallest-First (34)

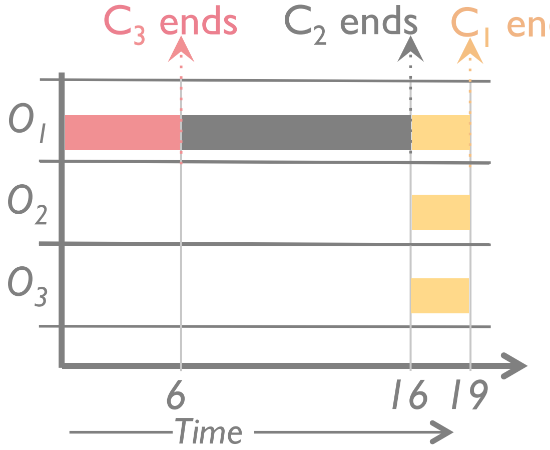
Ordering Heuristic



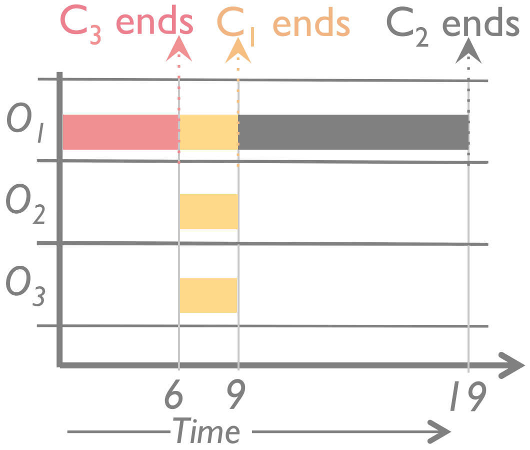
	C_1	C_2	C_3
Bottleneck	3	10	6



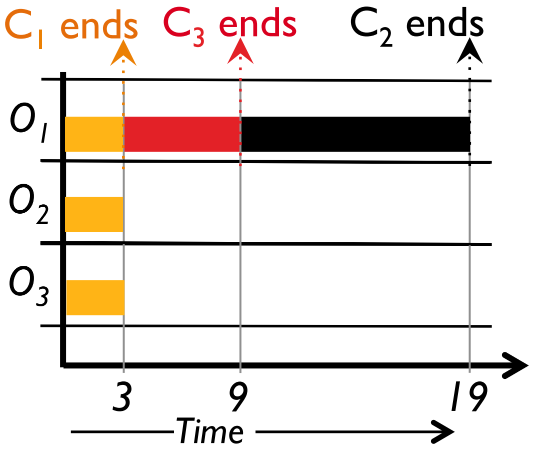
Shortest-First **(35)**



Narrowest-First **(41)**



Smallest-First **(34)**



Smallest-Bottleneck **(31)**

Allocation Algorithm

A coflow cannot finish before its very last flow

Finishing flows faster than the bottleneck cannot decrease a coflow's completion time

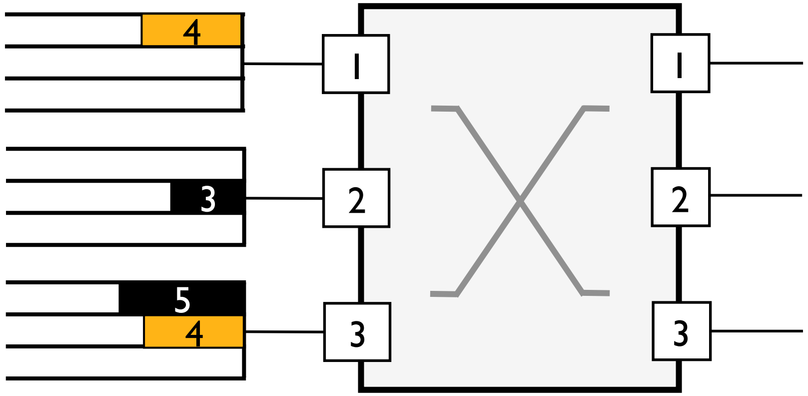
Allocate minimum flow rates such that all flows of a coflow finish together on time

Varys

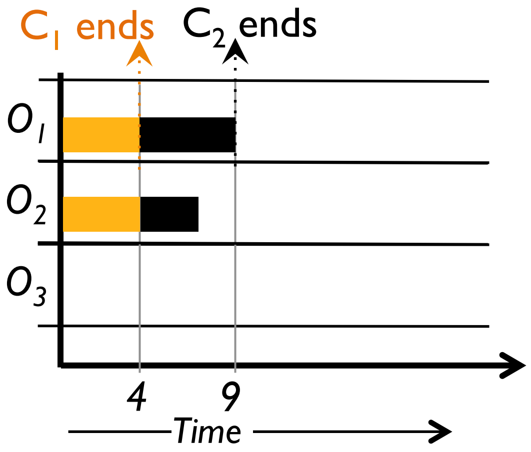
Enables coflows in data-intensive clusters

1. Coflow Scheduler — *Faster, application-aware data transfers throughout the network*
2. Global Coordination — *Consistent calculation and enforcement of scheduler decisions*
3. The Coflow API — *Decouples network optimizations from applications, relieving developers and end users*

The Need for Coordination

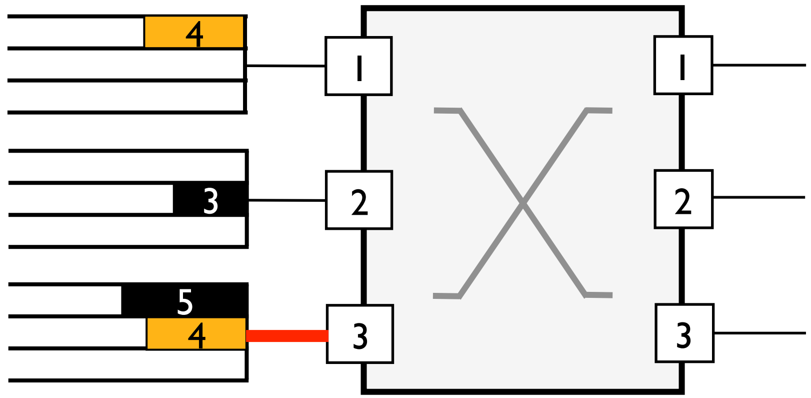


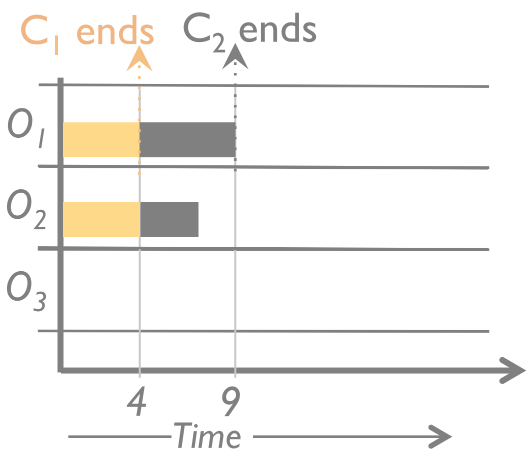
	C_1	C_2
Bottleneck	4	5



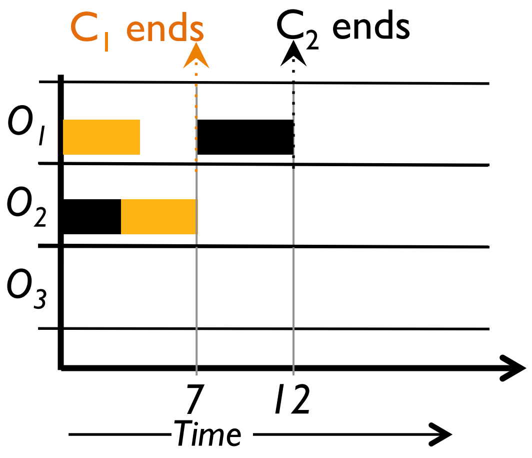
Scheduling *with* Coordination (Total CCT = 13)

The Need for Coordination





Scheduling
with
Coordination
(Total CCT = 13)



Scheduling
without
Coordination
(Total CCT = 19)

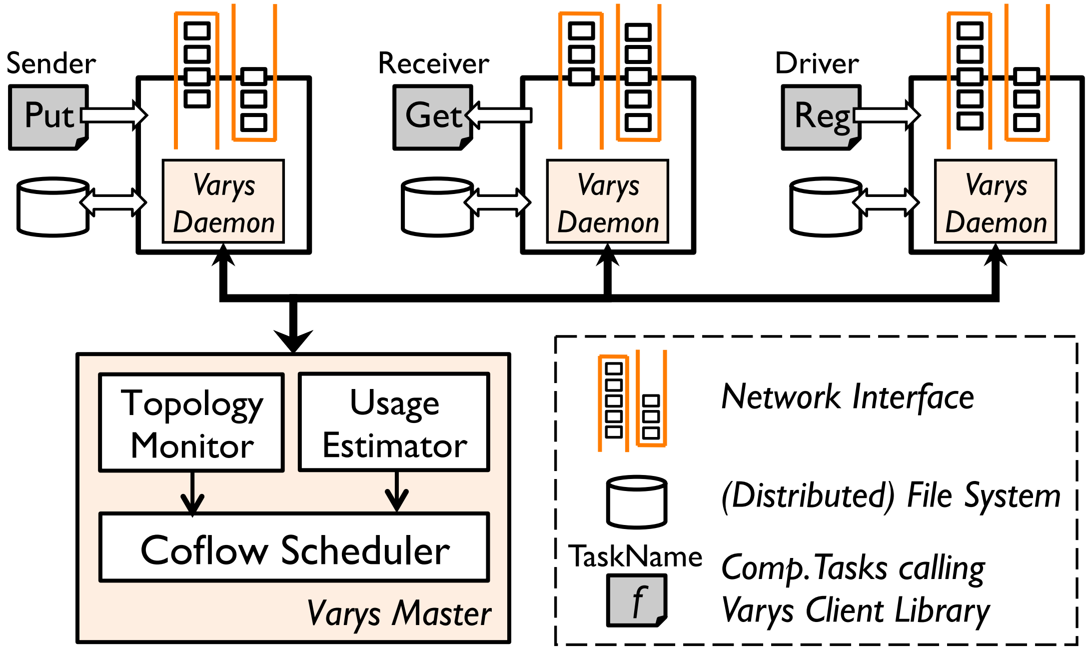
Uncoordinated local decisions *interleave* coflows, hurting performance

Varys Architecture

Centralized master-slave architecture

- Applications use a client library to communicate with the master

Actual *timing* and *rates are* determined by the coflow scheduler



Varys

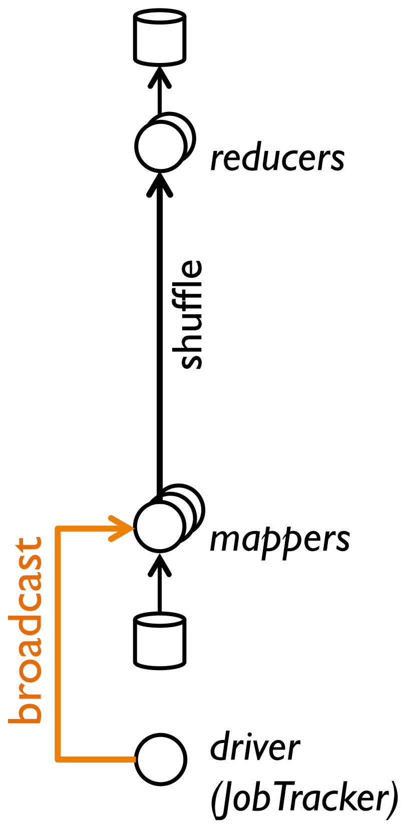
Enables coflows in data-intensive clusters

1. Coflow Scheduler — *Faster, application-aware data transfers throughout the network*
2. Global Coordination — *Consistent calculation and enforcement of scheduler decisions*
3. The Coflow API — *Decouples network optimizations from applications, relieving developers and end users*

The Coflow API

1. **NO** *changes to user jobs*
2. **NO** *storage management*

- register
- put
- get
- unregister



@driver

b ← **register**(*BROADCAST*)
s ← register(*SHUFFLE*)

id ← *b*.**put**(*content*)
…

b.**unregister**()
s.unregister()

@mapper

b.**get**(*id*)
…
id_{sl} ← s.**put**(*content*)
…

@reducer

s.**get**(id_{sl})
…

Evaluation

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

1. Does it improve performance?
2. Can it beat non-preemptive solutions?
3. Do we *really* need coordination?

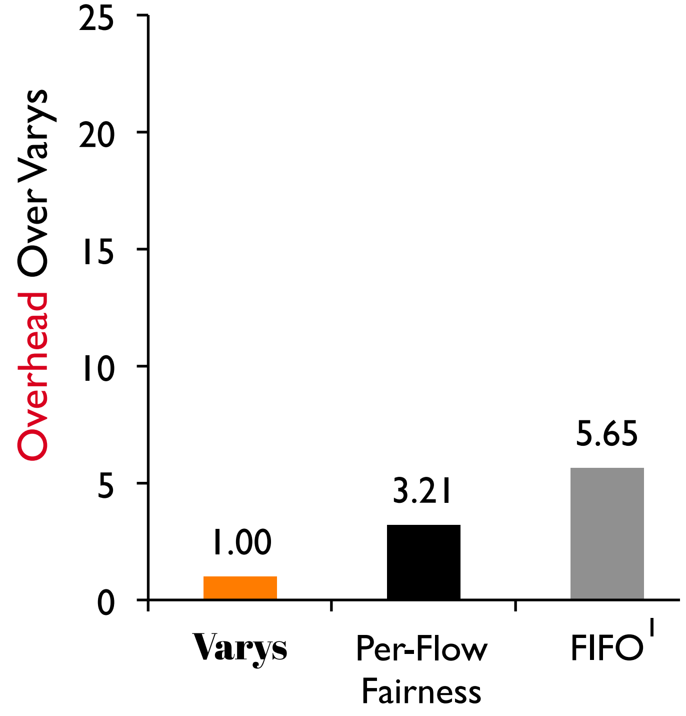
YES

Better than Per-Flow Fairness

Comm. Heavy

	Comm. Improv.	Job Improv.
EC2	3.16X	2.50X
Sim.	4.86X	3.39X

Preemption is Necessary [Sim.]

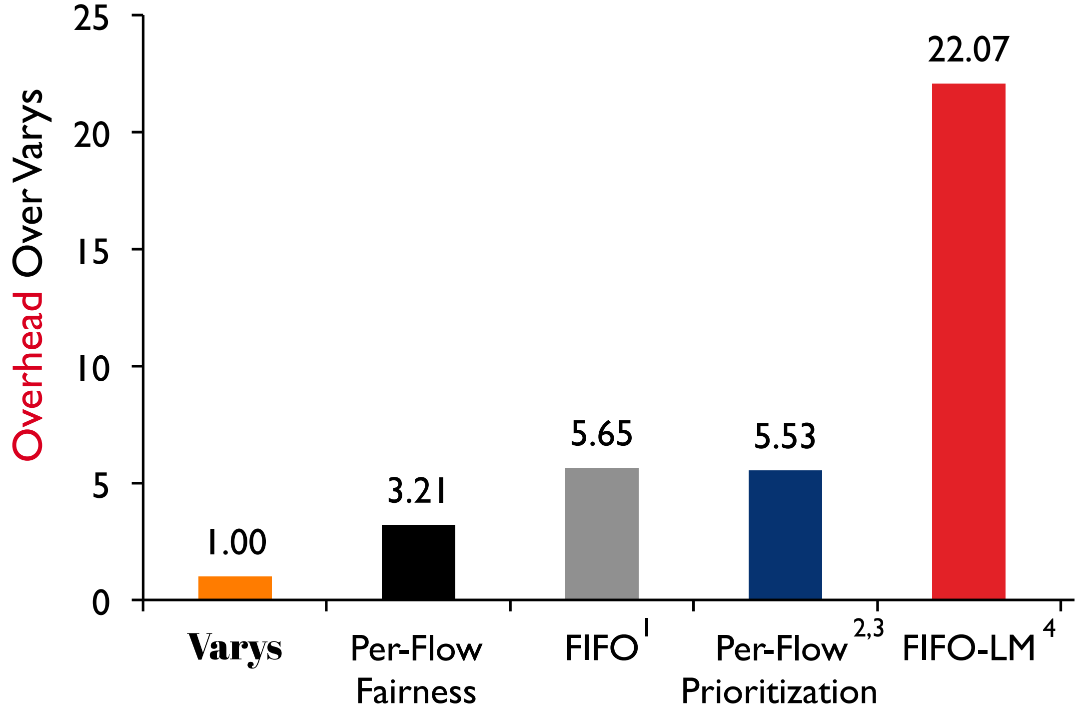


NO

Starvation

1. Managing Data Transfers in Computer Clusters with Orchestra, SIGCOMM'2011

Lack of Coordination Hurts [Sim.]



Smallest-flow-first (per-flow priorities)

- Minimizes flow completion time

FIFO-LM[4] performs decentralized coflow scheduling

- Suffers due to local decisions
- Works well for small, similar coflows

1. Managing Data Transfers in Computer Clusters with Orchestra, SIGCOMM'2011
2. Finishing Flows Quickly with Preemptive Scheduling, SIGCOMM'2012
3. pFabric: Minimal Near-Optimal Datacenter Transport, SIGCOMM'2013
4. Decentralized Task-Aware Scheduling for Data Center Networks, SIGCOMM'2014

Coflow

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

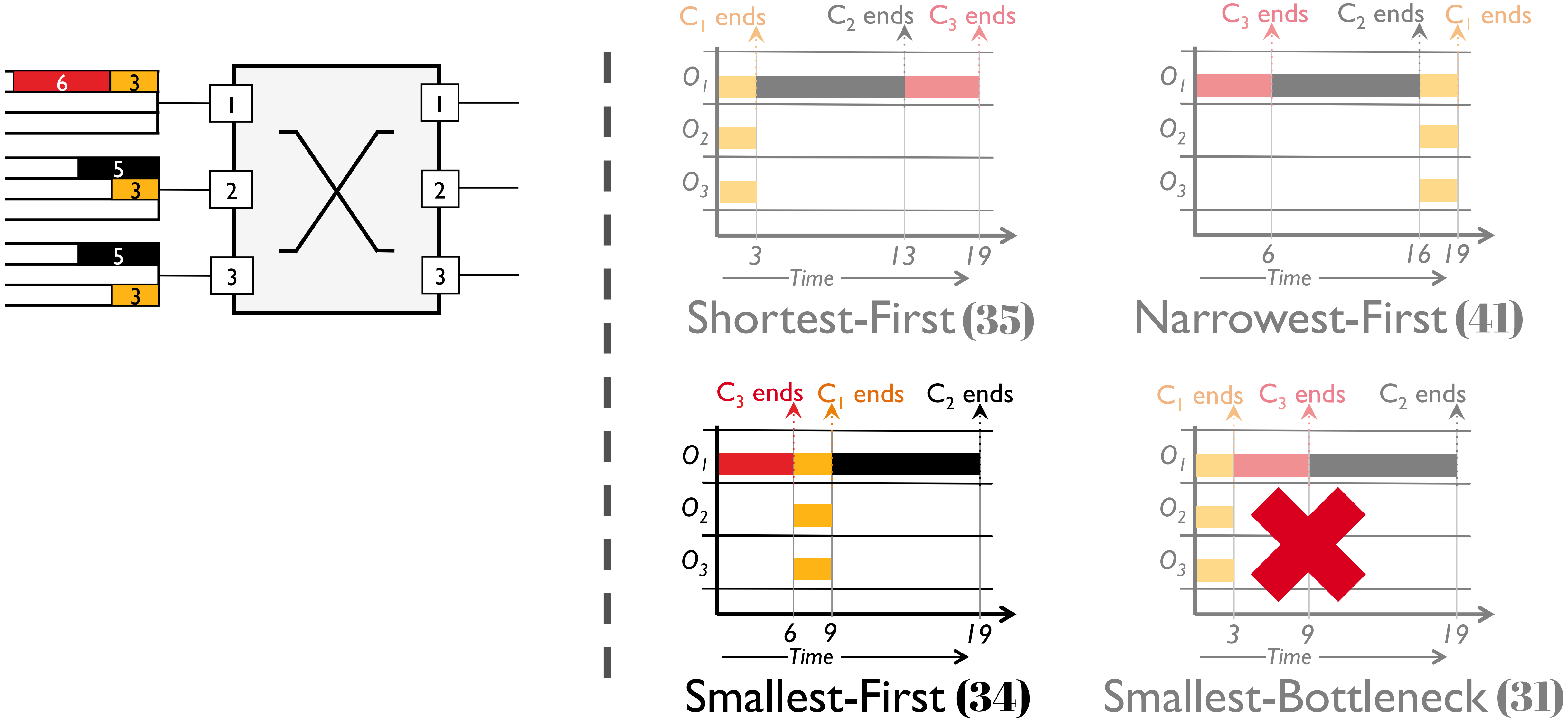
1. ~~The size of each flow,~~ ← Pipelining between stages
2. ~~The total number of flows, and~~ ← Speculative executions
3. ~~The endpoints of individual flows.~~ ← Task failures and restarts

How to Perform Coflow Scheduling *Without* Complete Knowledge?

Implications

Minimize Avg. Comp. Time	Flows in a Single Link	Coflows in an Entire Datacenter
With complete knowledge	Smallest-Flow-First	Ordering ~~by Bottleneck Size~~ + ~~Data-Proportional~~ Rate Allocation
Without complete knowledge	Least-Attained Service (LAS)	?

Revisiting Ordering Heuristics



Coflow-Aware LAS (CLAS)

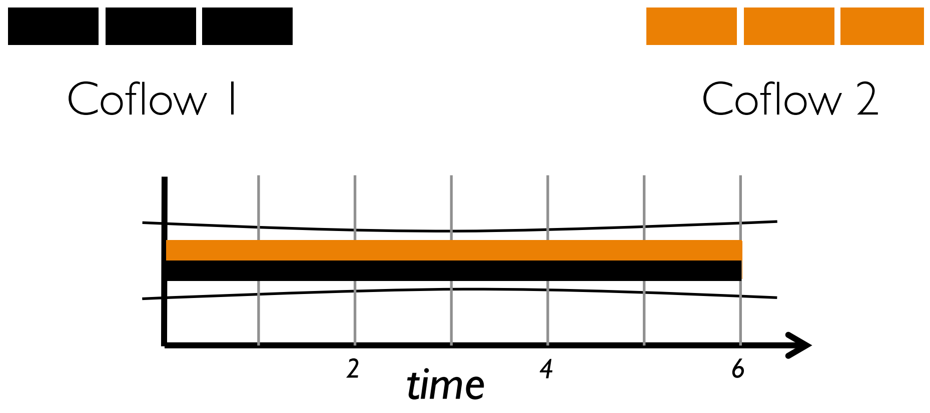
Set priority that decreases with how much a coflow has *already* sent

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

Use *total size* of coflows to set priorities

- Avoids the drawbacks of full decentralization

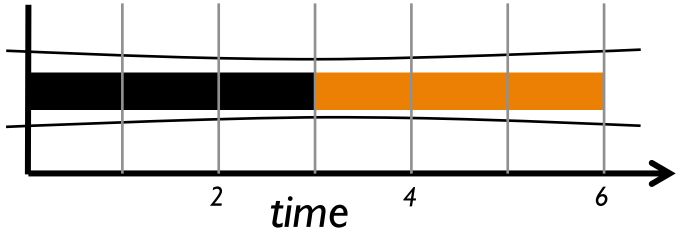
Coflow-Aware LAS (CLAS)



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

Continuous priority reduces to fair sharing when similar coflows coexist

- Priority oscillation



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

FIFO works well for similar coflows

- Avoids the drawbacks of full decentralization

Discretized Coflow-Aware LAS (D-CLAS)

Priority discretization

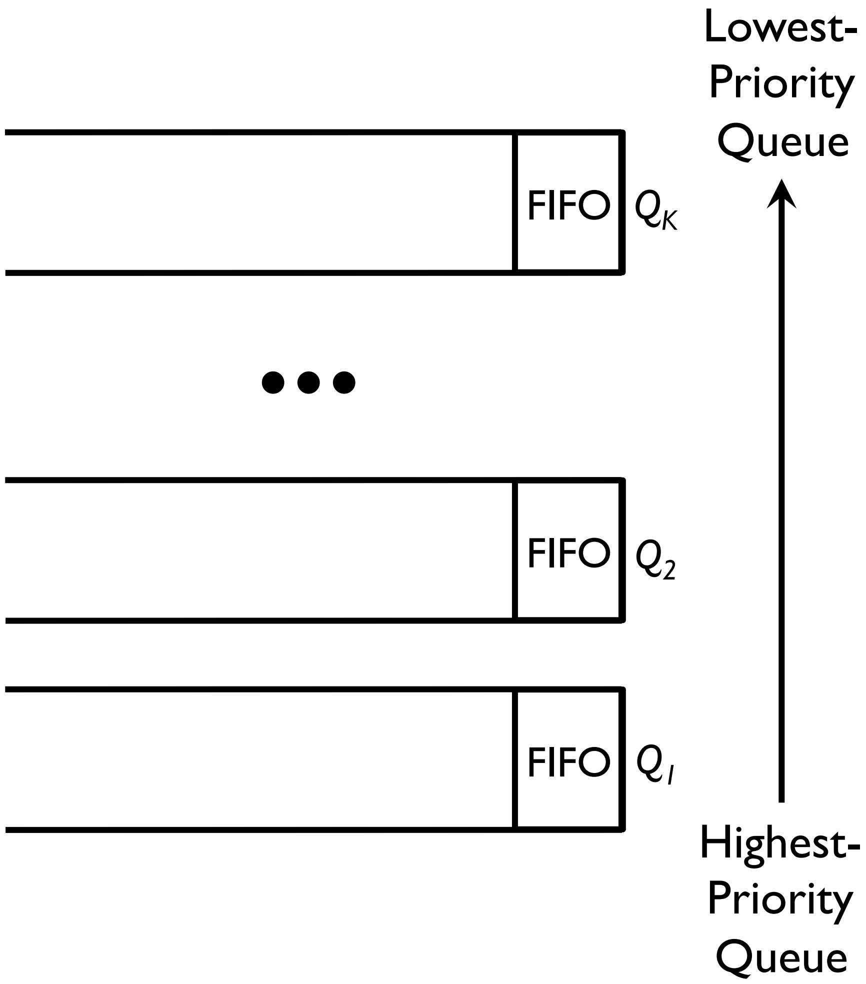
- Change priority when total size exceeds predefined thresholds

Scheduling policies

- FIFO within the same queue
- Prioritization across queue

Weighted sharing across queues

- Guarantees starvation avoidance



How to Discretize Priorities?

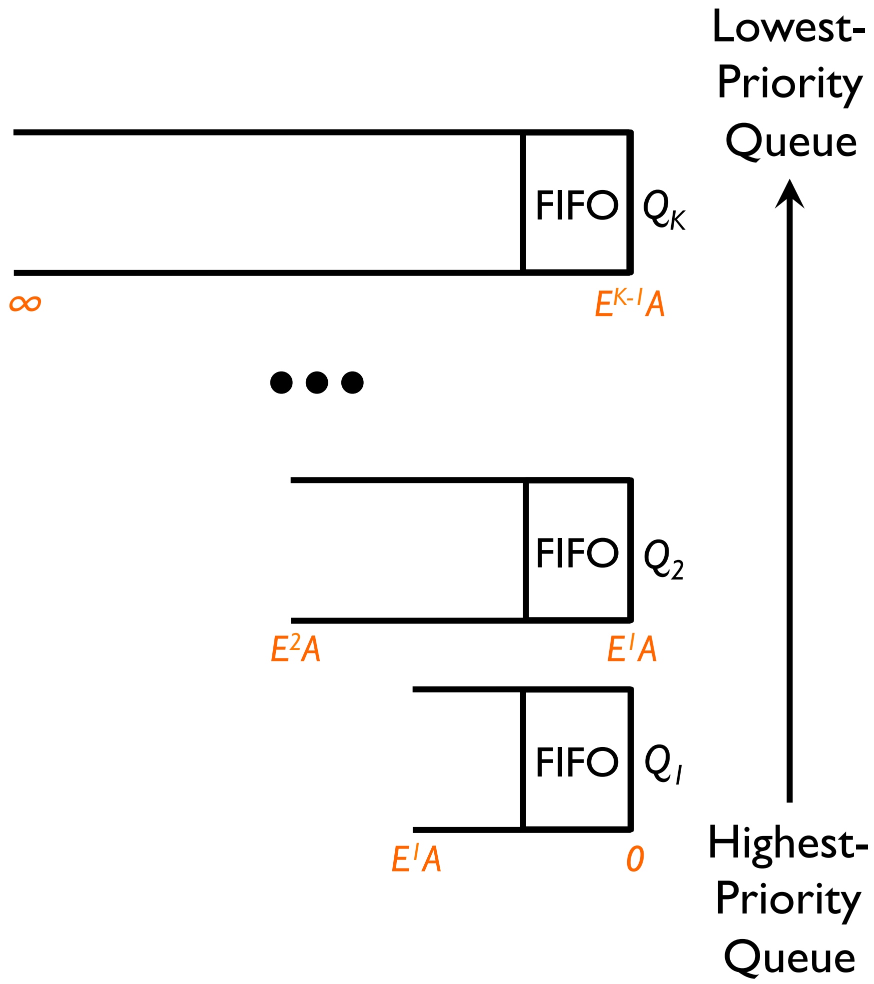
Exponentially spaced thresholds

- K : number of queues
- A : threshold constant
- E : threshold exponent

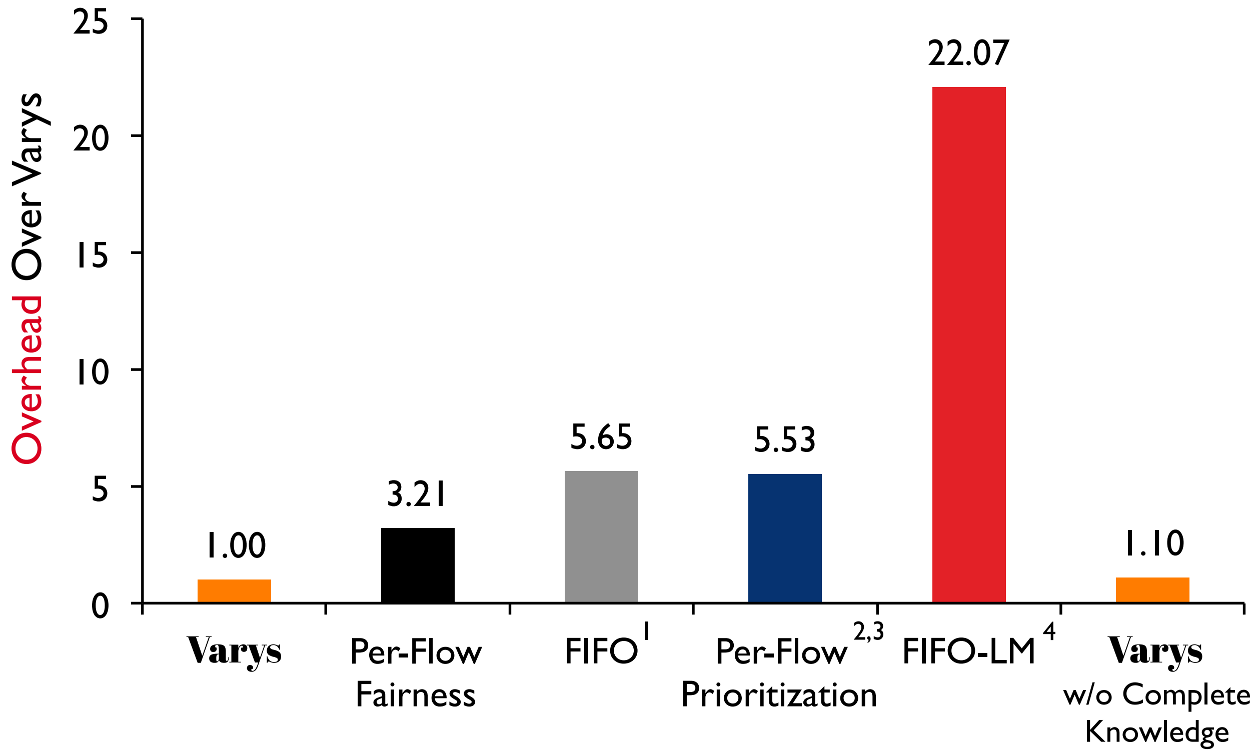
Loose coordination suffices to calculate global coflow sizes

- Slaves make independent decisions in between

Small coflows (smaller than E^1A) do not experience coordination overheads!



Closely Approximates **Varys** [Sim. & EC2]



1. *Managing Data Transfers in Computer Clusters with Orchestra, SIGCOMM'2011*
2. *Finishing Flows Quickly with Preemptive Scheduling, SIGCOMM'2012*
3. *pFabric: Minimal Near-Optimal Datacenter Transport, SIGCOMM'2013*
4. *Decentralized Task-Aware Scheduling for Data Center Networks, SIGCOMM'2014*

My Contributions

Coflow

Application-Aware Network Scheduling

My Contributions

Orchestra
SIGCOMM'11

Varys
SIGCOMM'14

Aalo
SIGCOMM'15

Application-Aware Network Scheduling

My Contributions

Spark NSDI'12	**Sinbad** SIGCOMM'13		*Network-Aware Applications*
Orchestra SIGCOMM'11	**Varys** SIGCOMM'14	**Aalo** SIGCOMM'15	*Application-Aware Network Scheduling*
FairCloud SIGCOMM'12	**HARP** SIGCOMM'12	**ViNEYard** ToN'12	*Datacenter Resource Allocation*

My Contributions

Spark NSDI'12 *Top-Level Apache Project*	**Sinbad** SIGCOMM'13 *Merged at Facebook*		*Network-Aware Applications*
Orchestra SIGCOMM'11 *Merged with Spark*	**Varys** SIGCOMM'14 *Open-Source*	**Aalo** SIGCOMM'15 *Open-Source*	*Application-Aware Network Scheduling*
FairCloud SIGCOMM'12 *@HP*	**HARP** SIGCOMM'12 *@Microsoft Bing*	**ViNEYard** ToN'12 *Open-Source*	*Datacenter Resource Allocation*

Communication-First Big Data Systems

In-Datacenter Analytics

- Cheaper SSDs and DRAM, proliferation of optical networks, and resource disaggregation will make network the primary bottleneck

Inter-Datacenter Analytics

- Bandwidth-constrained wide-area networks

End User Delivery

- Faster and responsive delivery of analytics results over the Internet for better end user experience

Systems MEND THE GAP **Networking**

Better capture application-level performance goals using *coflows*

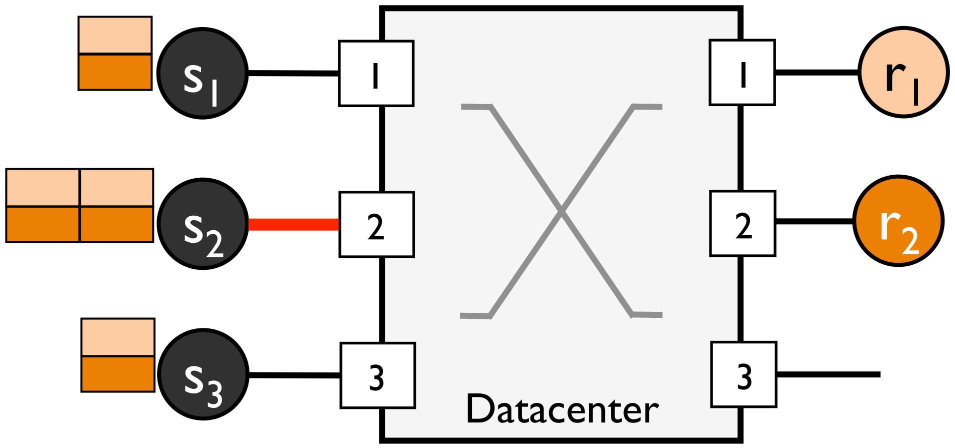
Coflows improve application-level performance and usability

- Extends networking and scheduling literature

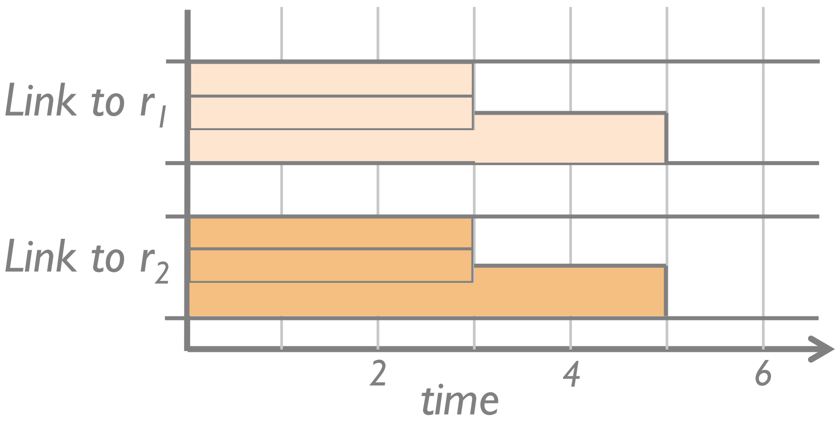
Coordination – even if not free – is worth paying for in many cases

mosharaf@cs.berkeley.edu
http://mosharaf.com

Improve Flow Completion Times



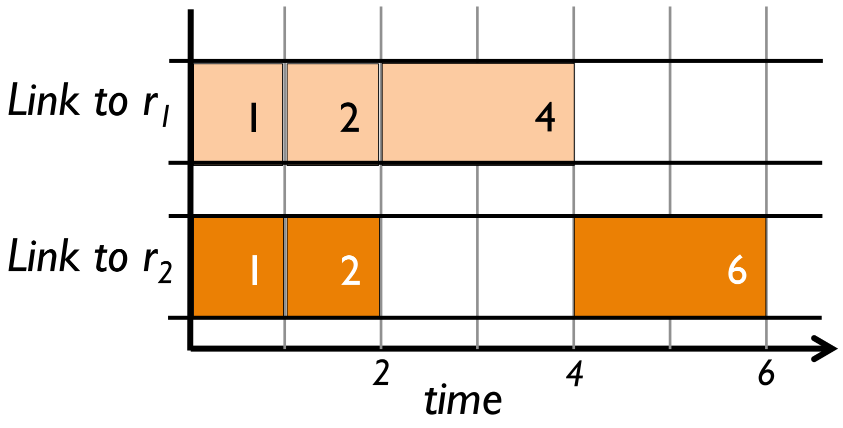
Per-Flow Fair Sharing



Shuffle Completion Time = **5**

Avg. Flow Completion Time = **3.66**

Smallest-Flow First[1,2]

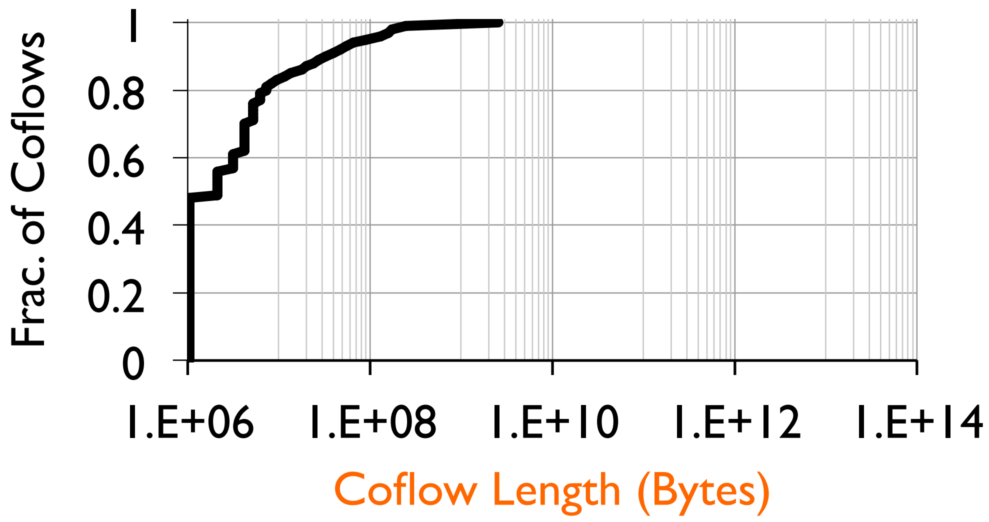


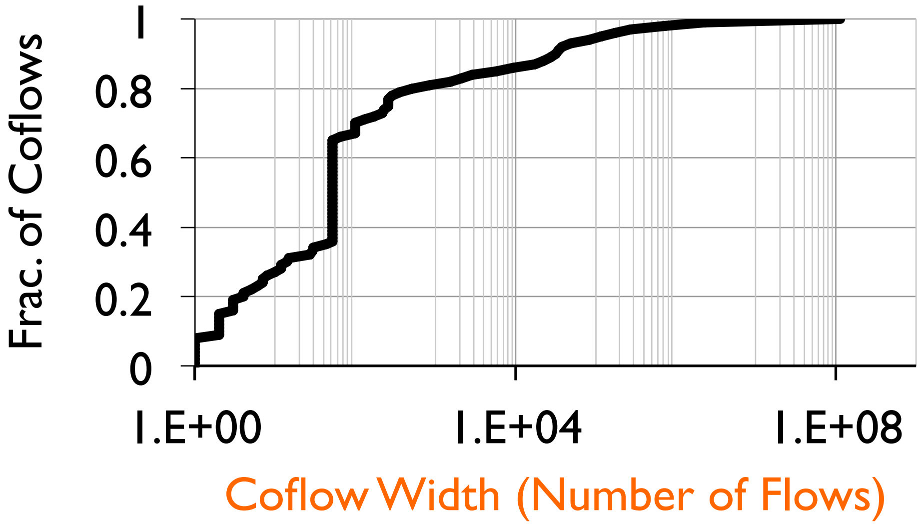
Shuffle Completion Time = **6**

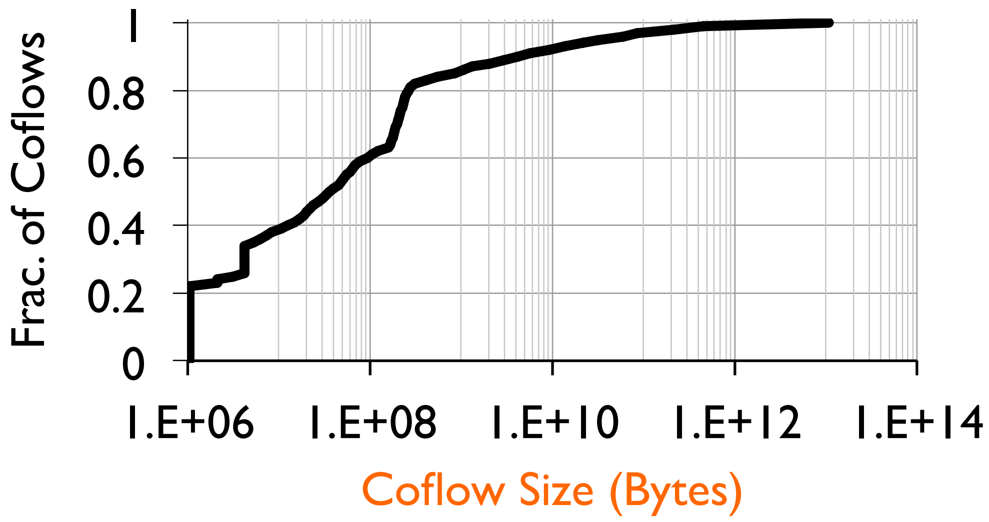
Avg. Flow Completion Time = **2.66**

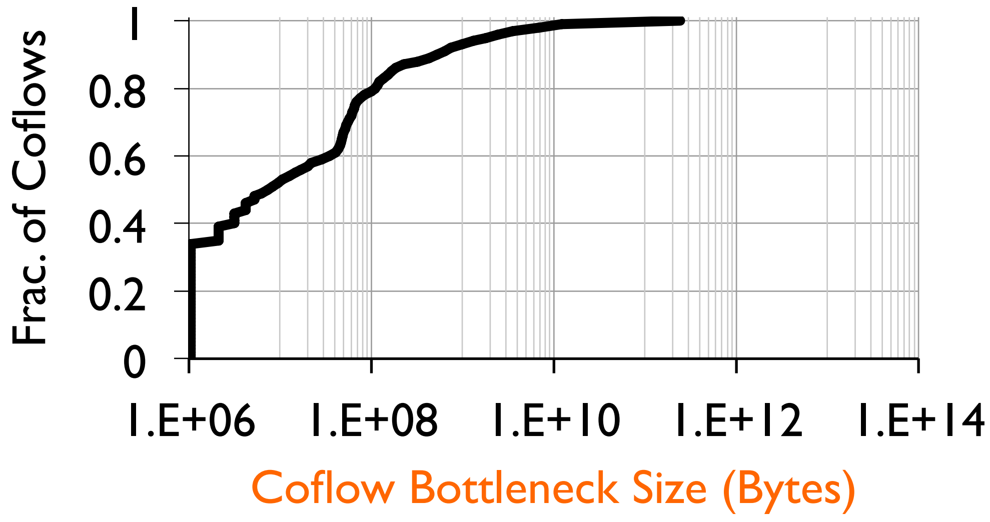
1. Finishing Flows Quickly with Preemptive Scheduling, SIGCOMM'2012.
2. pFabric: Minimal Near-Optimal Datacenter Transport, SIGCOMM'2013.

Distributions of Coflow Characteristics





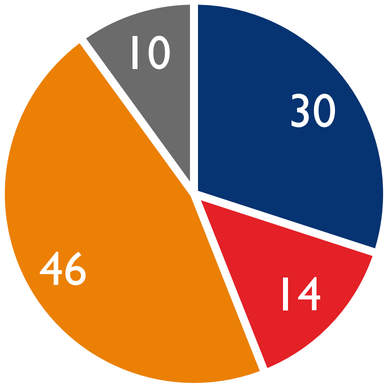


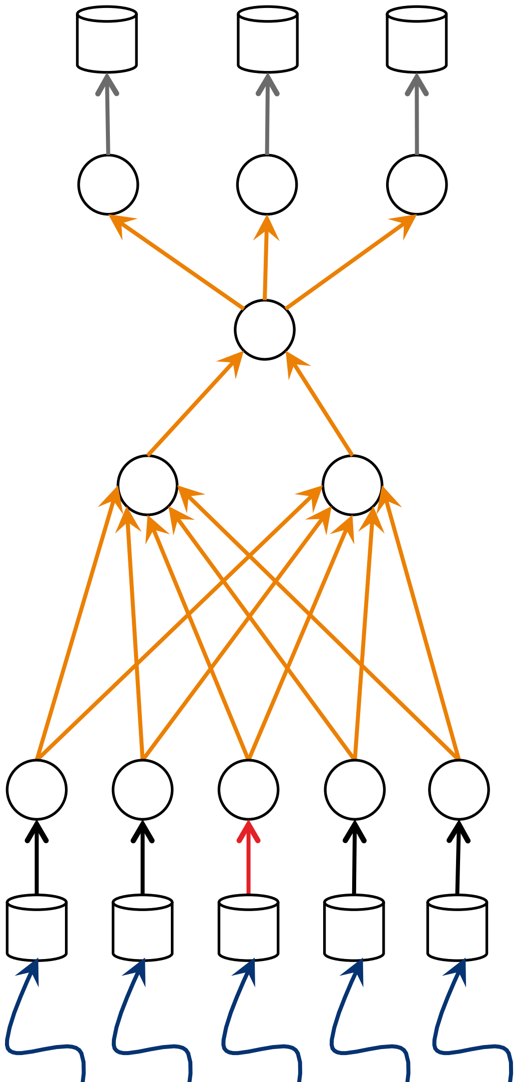


Traffic Sources

1. **Ingest and replicate** new data
2. **Read** input from remote machines, when needed
3. **Transfer** intermediate data
4. **Write and replicate** output

Percentage of Traffic by Category at ***Facebook***

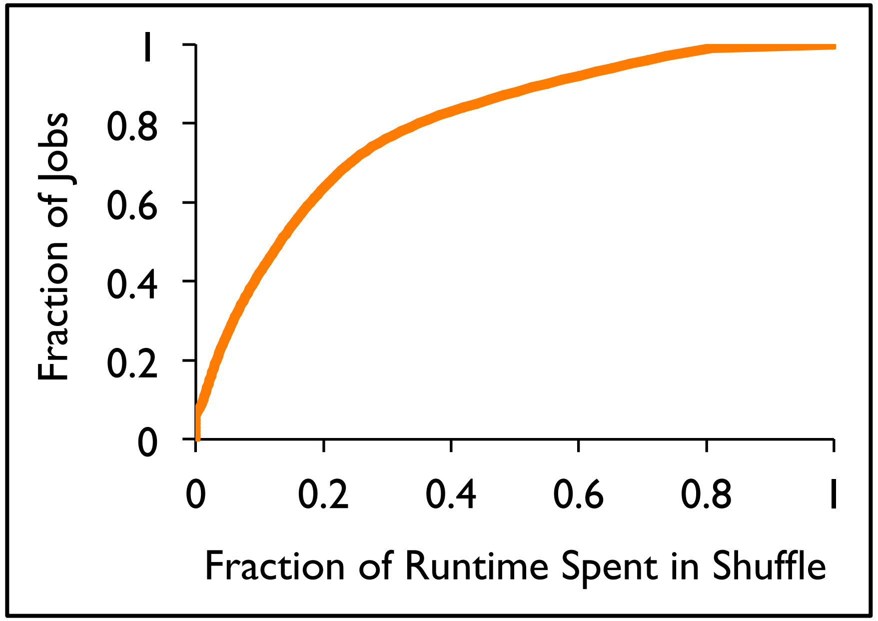




Distribution of Shuffle Durations

Performance

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



Month-long trace from a 3000-machine MapReduce production cluster at Facebook

320,000 jobs
150 Million tasks

Theoretical Results

Structure of optimal schedules

- *Permutation schedules might not always lead to the optimal solution*

Approximation ratio of COSS-CR

- *Polynomial-time algorithm with constant approximation ratio* $(\frac{64}{3})$[1]

The need for coordination

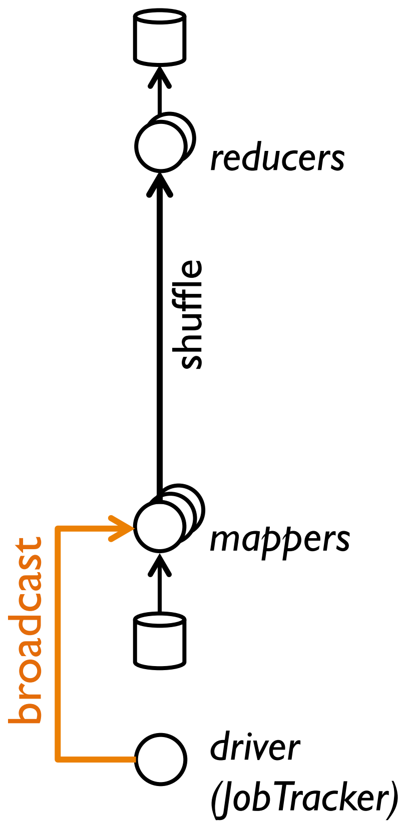
- Fully decentralized schedulers can perform arbitrarily worse than the optimal

1. Due to Zhen Qiu, Cliff Stein, and Yuan Zhong from the Department of Industrial Engineering and Operations Research, Columbia University, 2014

The Coflow API

1. NO changes to user jobs
2. NO storage management

- register
- put
- get
- unregister



@driver

$b \leftarrow$ register(*BROADCAST, numFlows*)

$s \leftarrow$ register(*SHUFFLE, numFlows,* {*b*})

id $\leftarrow$ *b*.**put**(*content, size*)

…

b.**unregister**()

s.unregister()

@mapper

b.**get**(*id*)

…

$id_{sl} \leftarrow s$.**put**(*content, size*)

…

@reducer

s.**get**(id_{sl})

…

Varys

Employs a two-step algorithm to support coflow deadlines

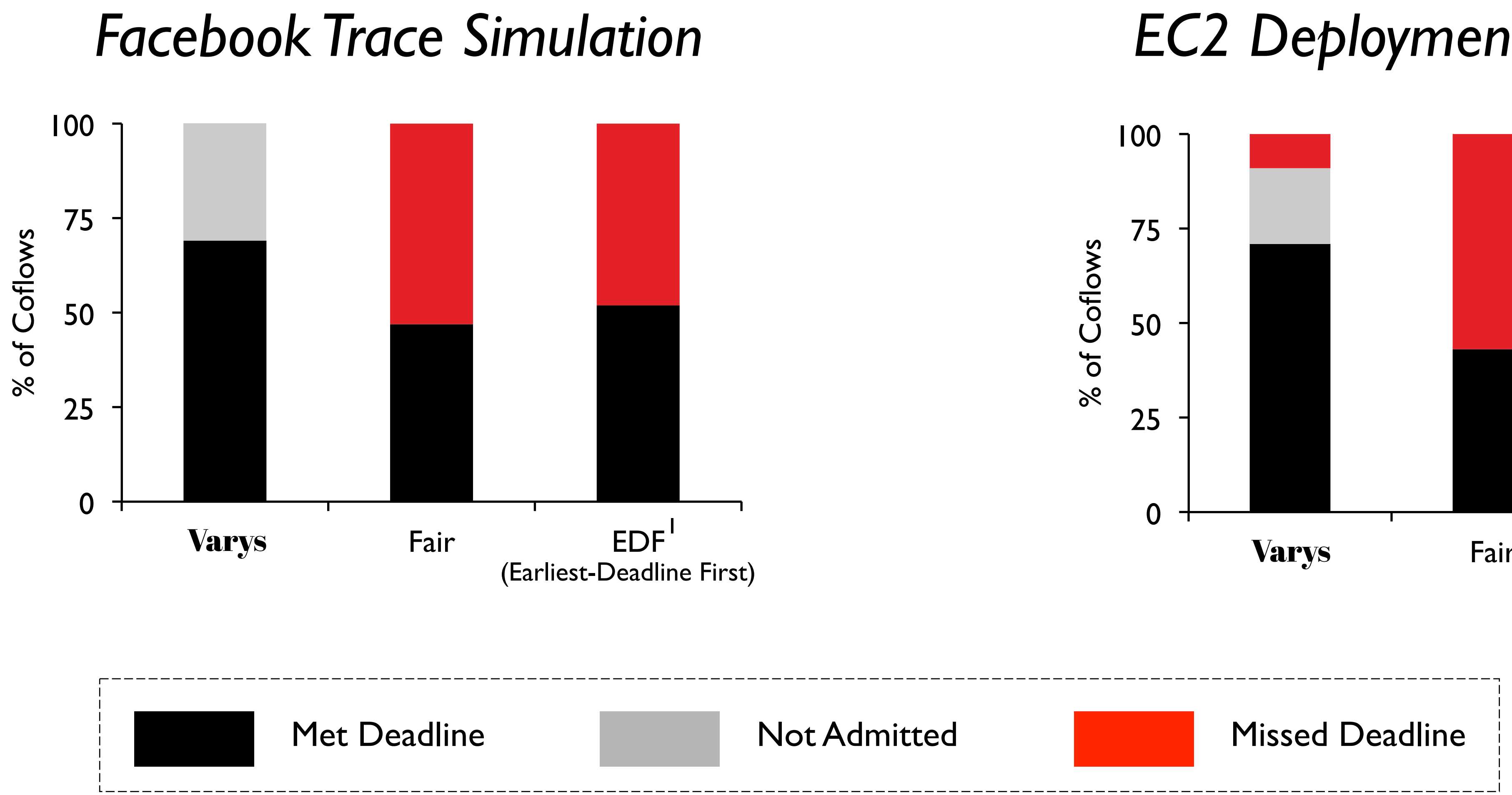
1. Admission control

   Do not admit any coflows that cannot be completed within deadline without violating existing deadlines

2. Allocation algorithm

   Allocate minimum required resources to each coflow to finish them at their deadlines

More Predictable



1. *Finishing Flows Quickly with Preemptive Scheduling, SIGCOMM'2012*

Optimizing Communication Performance: Systems Approach

"Let users figure it out"

	# Comm. Params*
Spark-v1.1.1	**6**
Hadoop-v1.2.1	**10**
YARN-v2.6.0	**20**

*Lower bound. Does not include *many* parameters that can indirectly impact communication; e.g., number of reducers etc. Also excludes control-plane communication/RPC parameters.

Experimental Methodology

Varys deployment in EC2

- **100** m2.4xlarge machines
- Each machine has **8** CPU cores, **68.4** GB memory, and **1** Gbps NIC
- **~900** Mbps/machine during all-to-all communication

Trace-driven simulation

- Detailed replay of a day-long Facebook trace (circa October 2010)
- **3000**-machine,**150**-rack cluster with **10:1** oversubscription