Coflow

Recent Advances and What's Next?

Mosharaf Chowdhury

University of Michigan

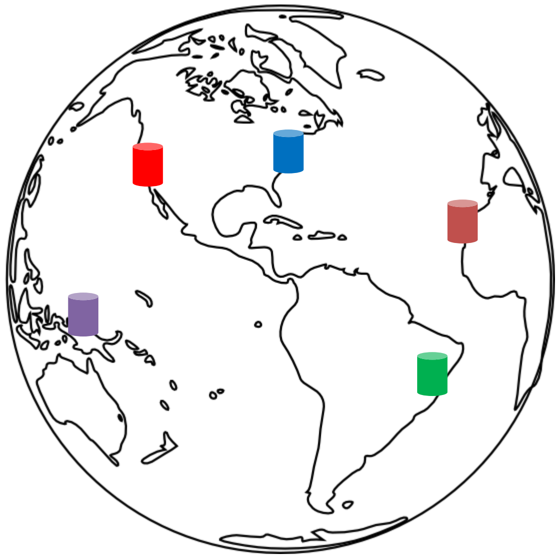
Rack-Scale Computing

Proactive Analytics
Before You Think!

Datacenter-Scale Computing

Coflow Networking	*Open Source*
Apache Spark	*Open Source*
Cluster File System	*Facebook*
Resource Allocation	*Microsoft*
DAG Scheduling	*Apache YARN*
Cluster Caching	*Alluxio*

Geo-Distributed Computing



Fast Analytics
Over the WAN

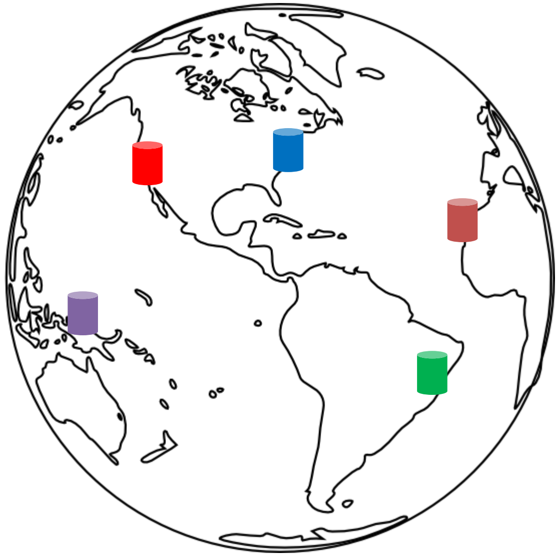
Rack-Scale Computing

< 0.01 ms

Datacenter-Scale Computing

~ 1 ms

Geo-Distributed Computing



> 100 ms

Big Data

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

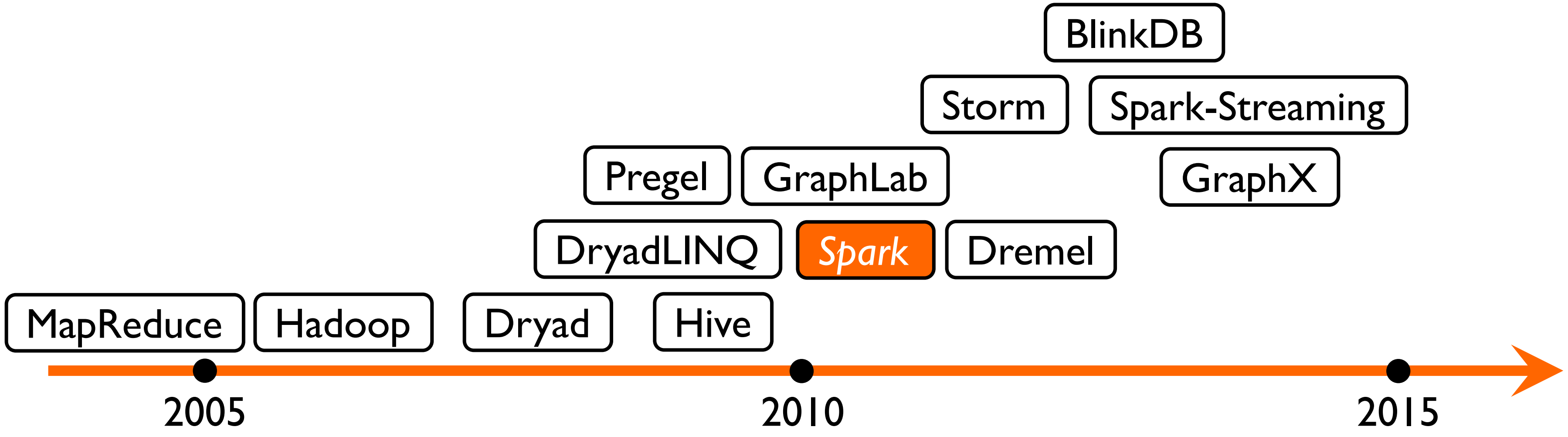
Increasing variety of sources

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

Cheaper disks, SSDs, and memory

Stalling processor speeds

Big Datacenters for Massive Parallelism



1. Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing, NSDI'2012.

Distributed Data-Parallel Applications

Multi-stage dataflow

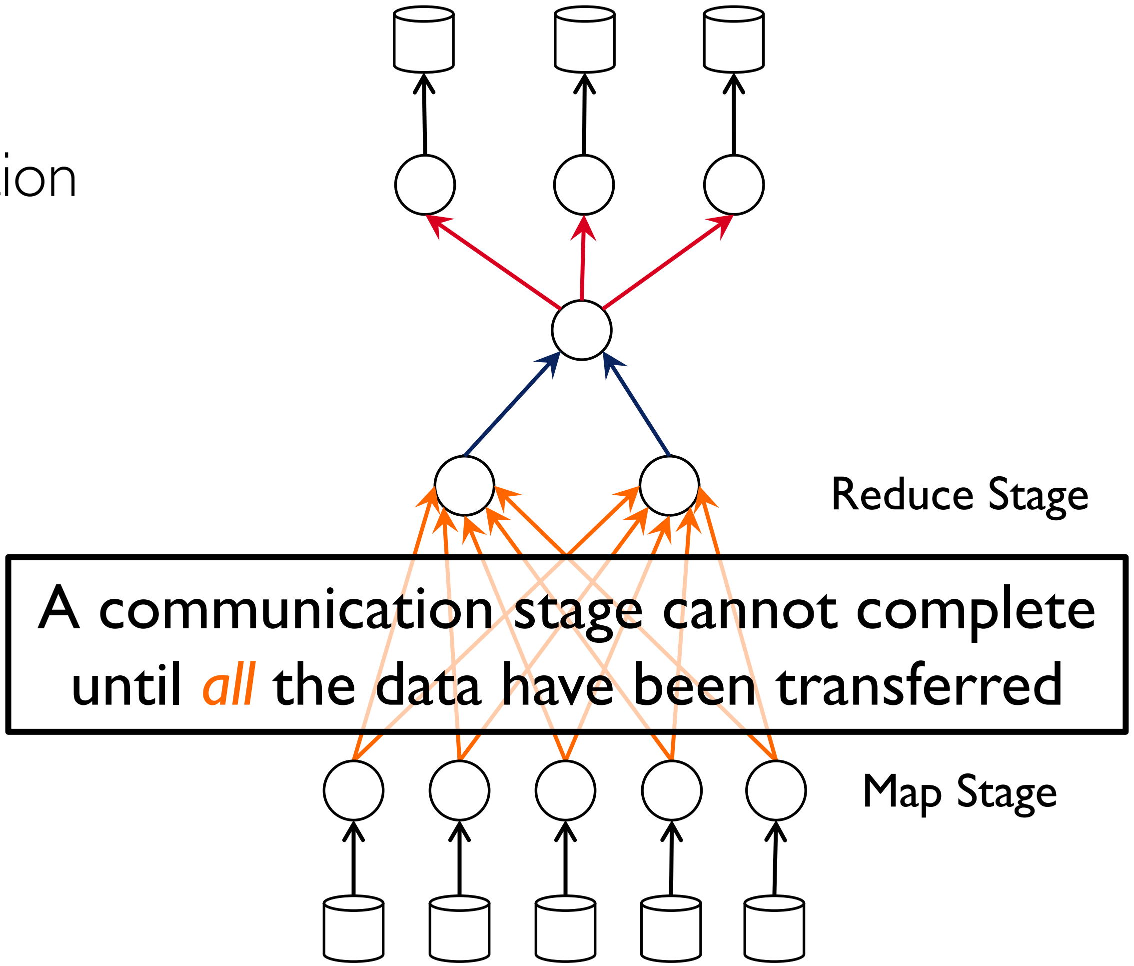
- Computation interleaved with communication

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.

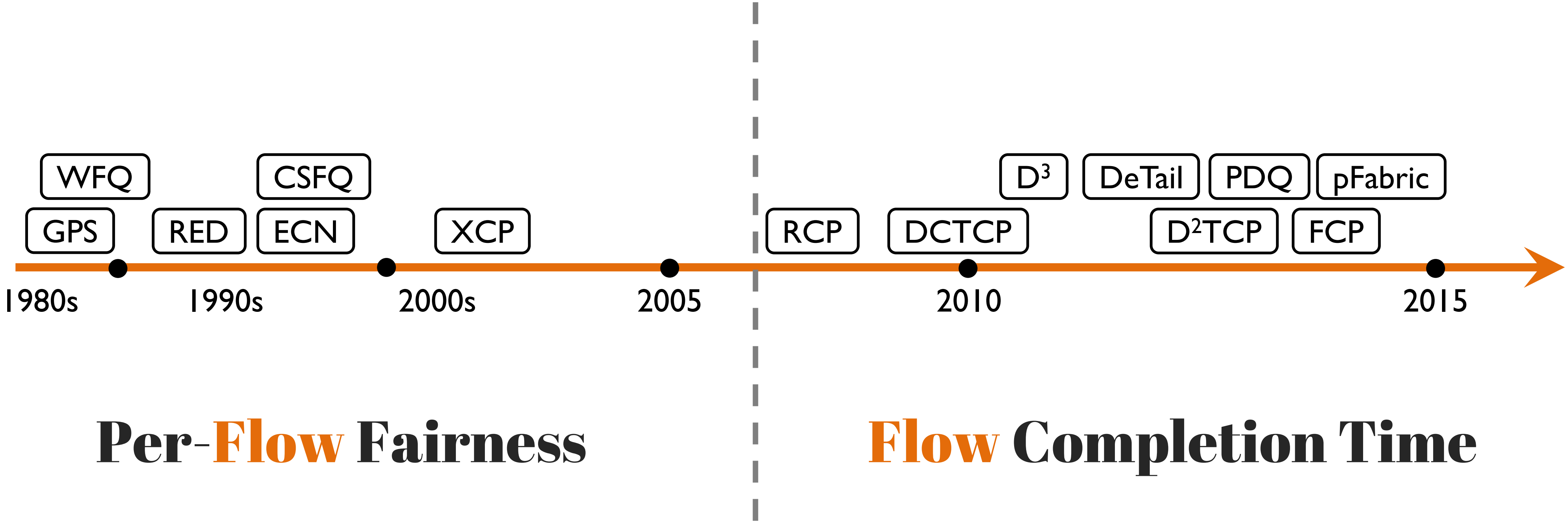
Flow

Transfers data from a source to a destination

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

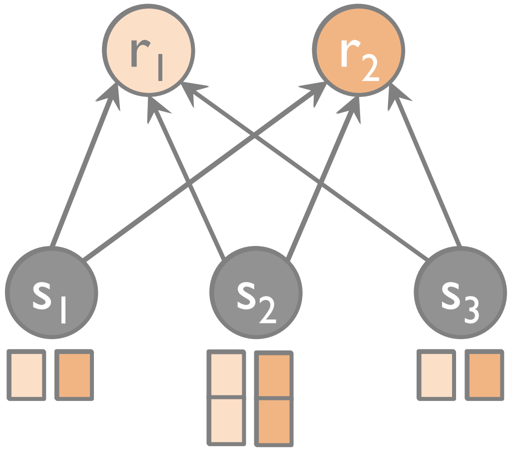
Faster Communication Stages: Traditional Networking Approach

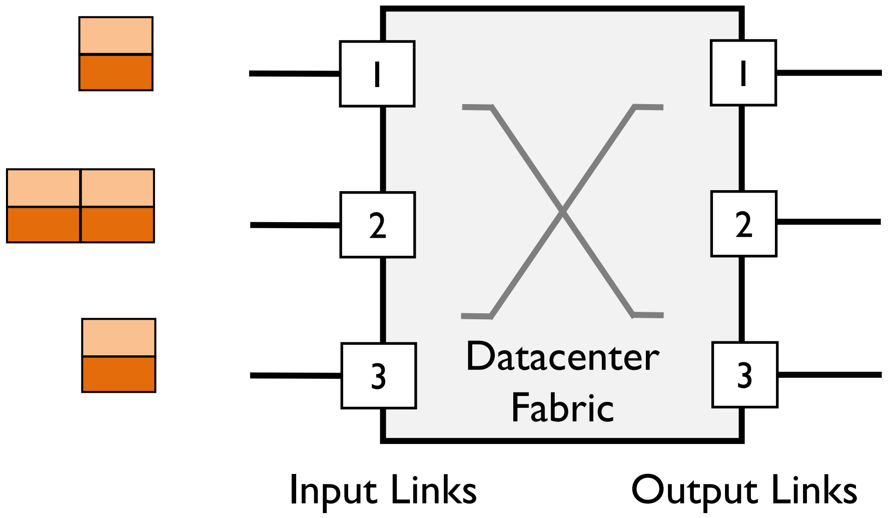
Existing Solutions



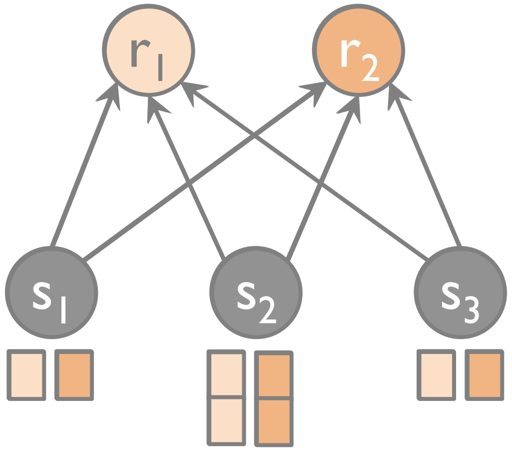
Independent flows cannot capture the collective communication behavior common in data-parallel applications

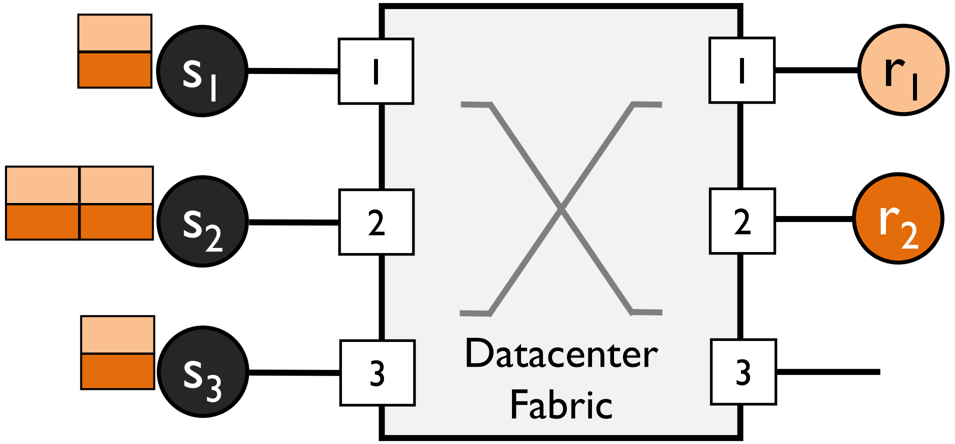
Why Do They Fall Short?



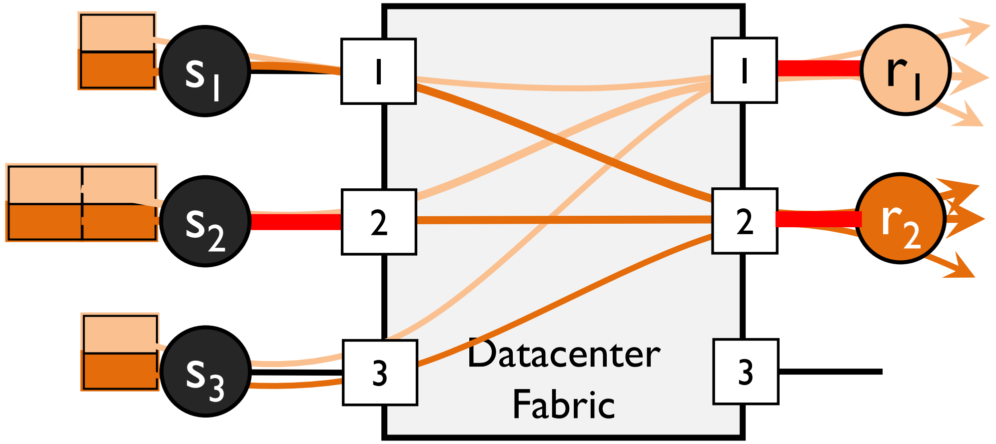


Why Do They Fall Short?

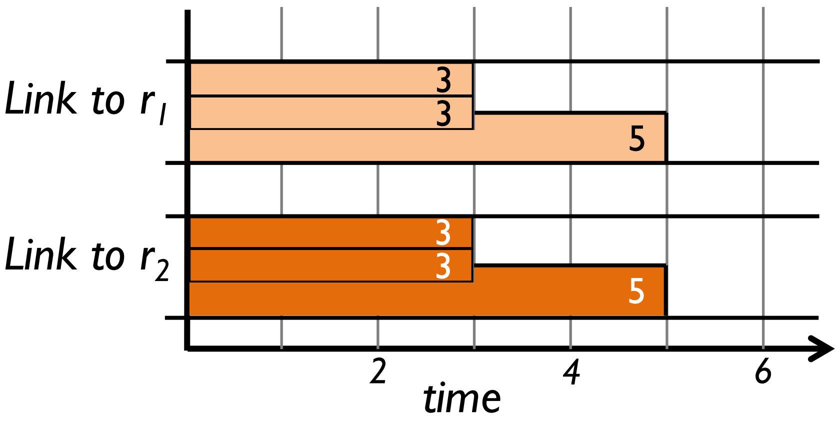




Why Do They Fall Short?



Per-Flow Fair Sharing

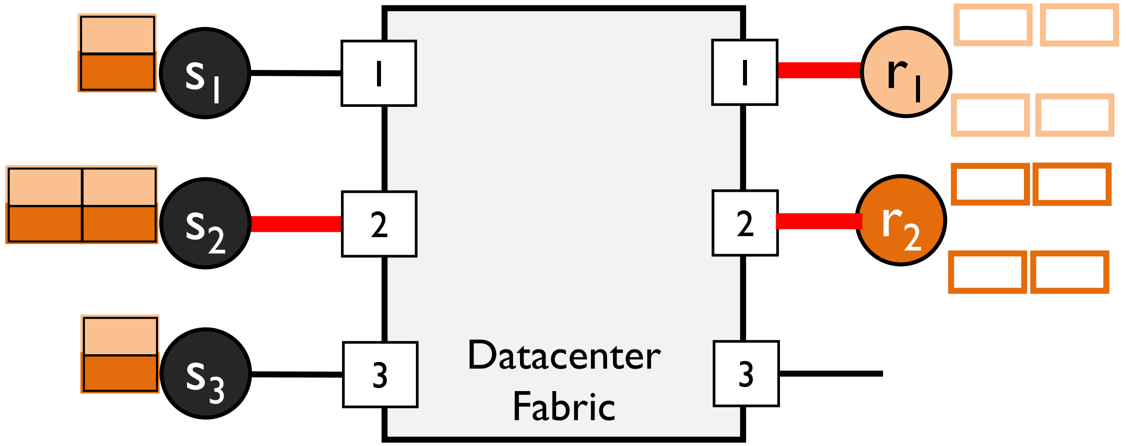


Shuffle Completion Time = **5**

Avg. Flow Completion Time = **3.66**

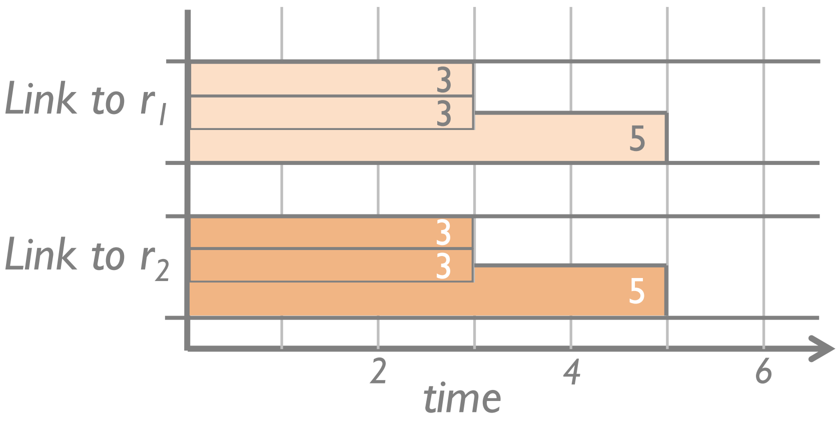
Solutions focusing on flow completion time cannot further decrease the shuffle completion time

Improve Application-Level Performance[1]



Slow down faster flows to *accelerate* slower flows

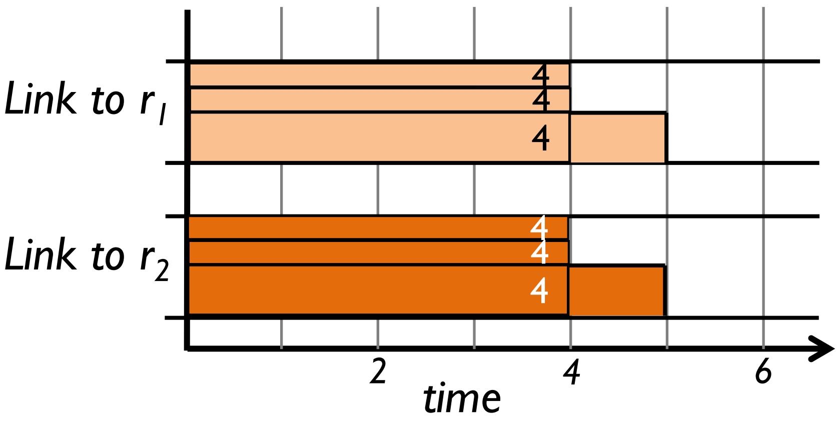
Per-Flow Fair Sharing



Shuffle Completion Time = **5**

Avg. Flow Completion Time = **3.66**

Data-Proportional Allocation



Shuffle Completion Time = **4**

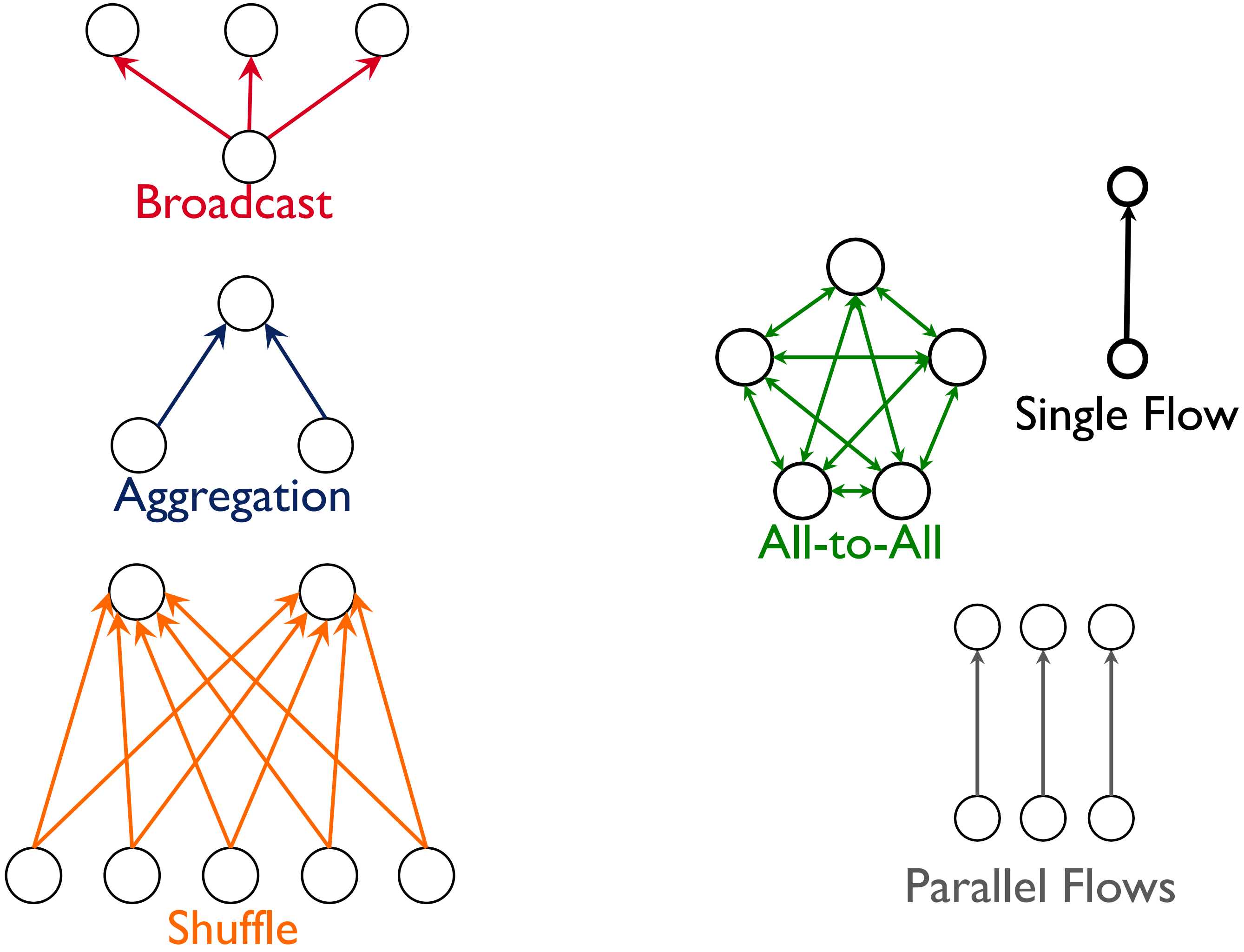
Avg. Flow Completion Time = **4**

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

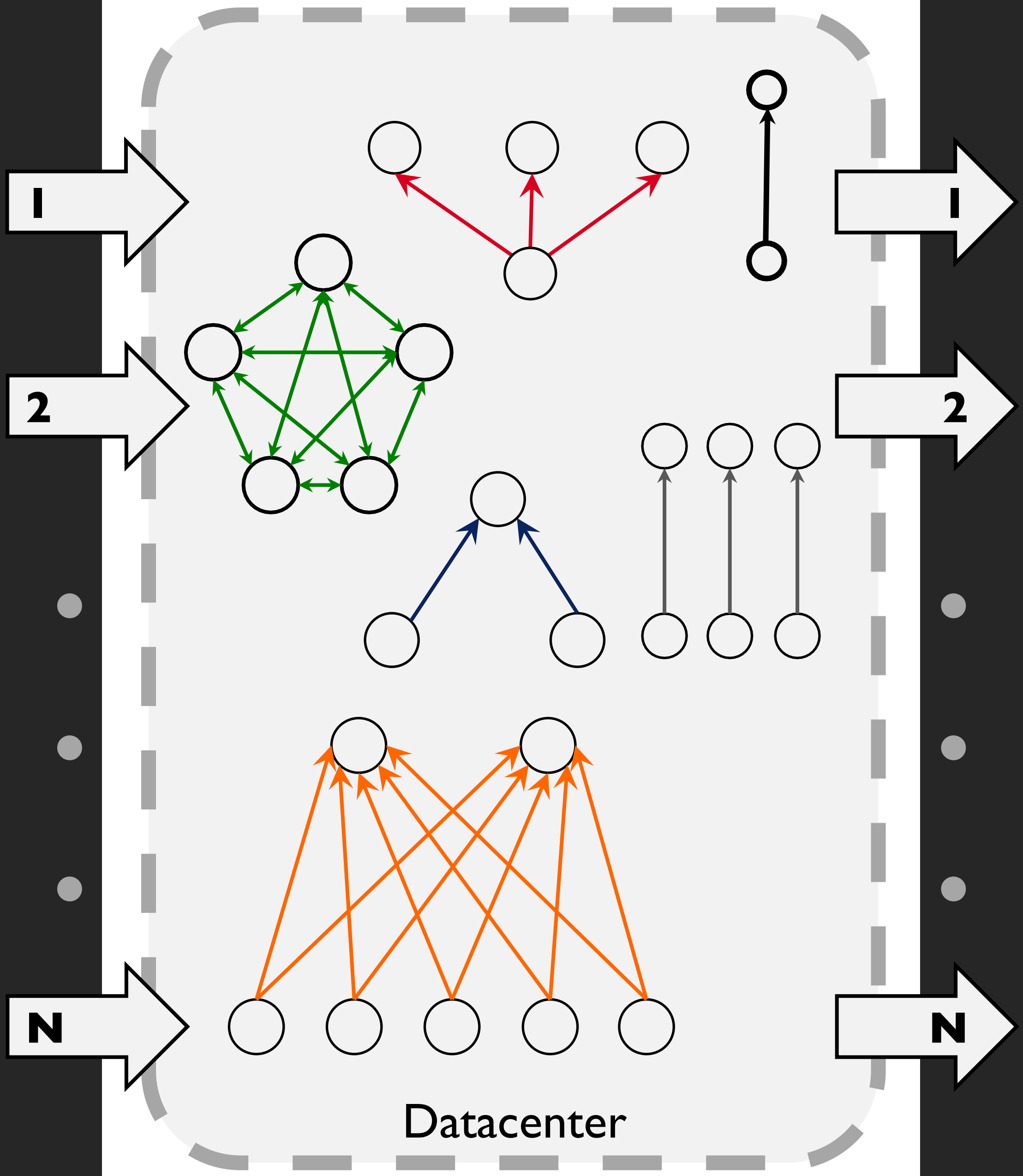
Coflow

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

1. Size of each flow;
2. Total number of flows;
3. Endpoints of individual flows;
4. Dependencies between coflows;


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], Aalo[2] & HUG[3]

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

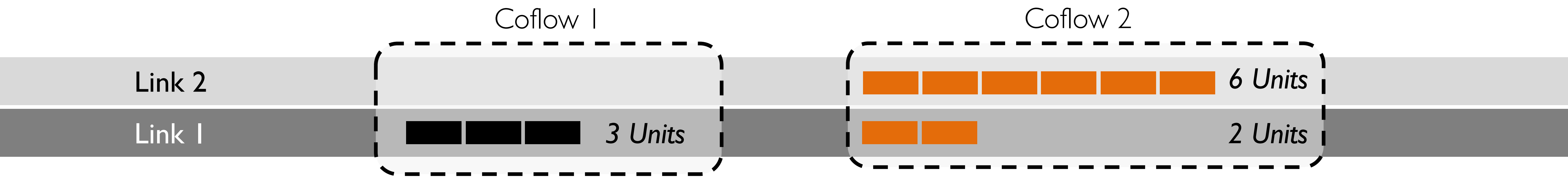
Faster, application-aware data transfers throughout the network

Consistent calculation and enforcement of scheduler decisions

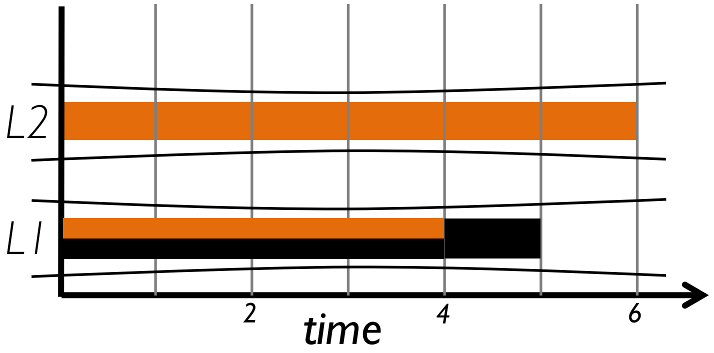
Decouples network optimizations from applications, relieving developers and end users

1. Efficient Coflow Scheduling with Varys, SIGCOMM'2014.
2. Efficient Coflow Scheduling Without Prior Knowledge, SIGCOMM'2015.
3. HUG: Multi-Resource Fairness for Correlated and Elastic Demands, NSDI'2016.

Benefits of Inter-Coflow Scheduling

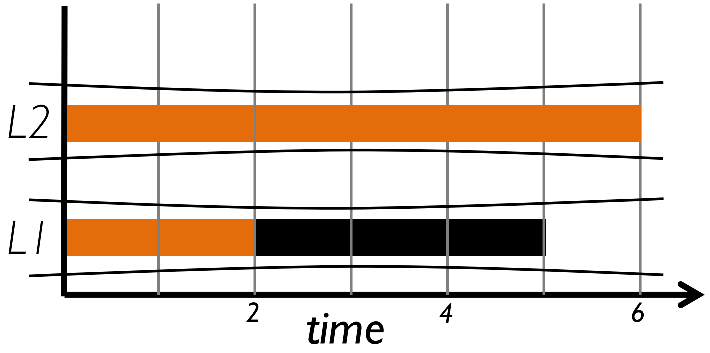


Fair Sharing



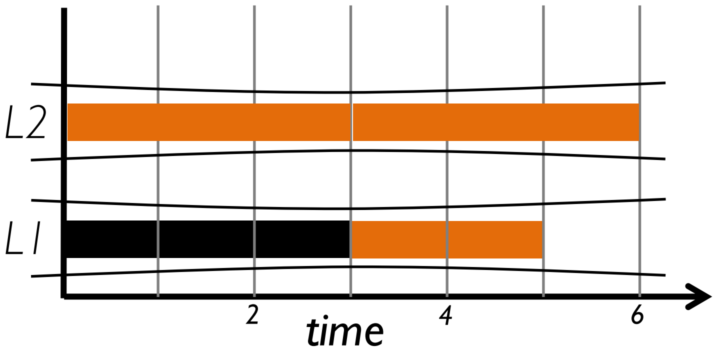
Coflow1 comp. time = 5
Coflow2 comp. time = 6

Smallest-Flow First[1,2]



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

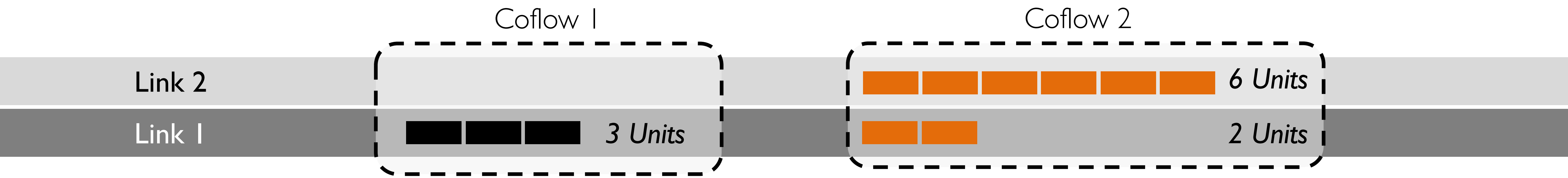
The Optimal



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

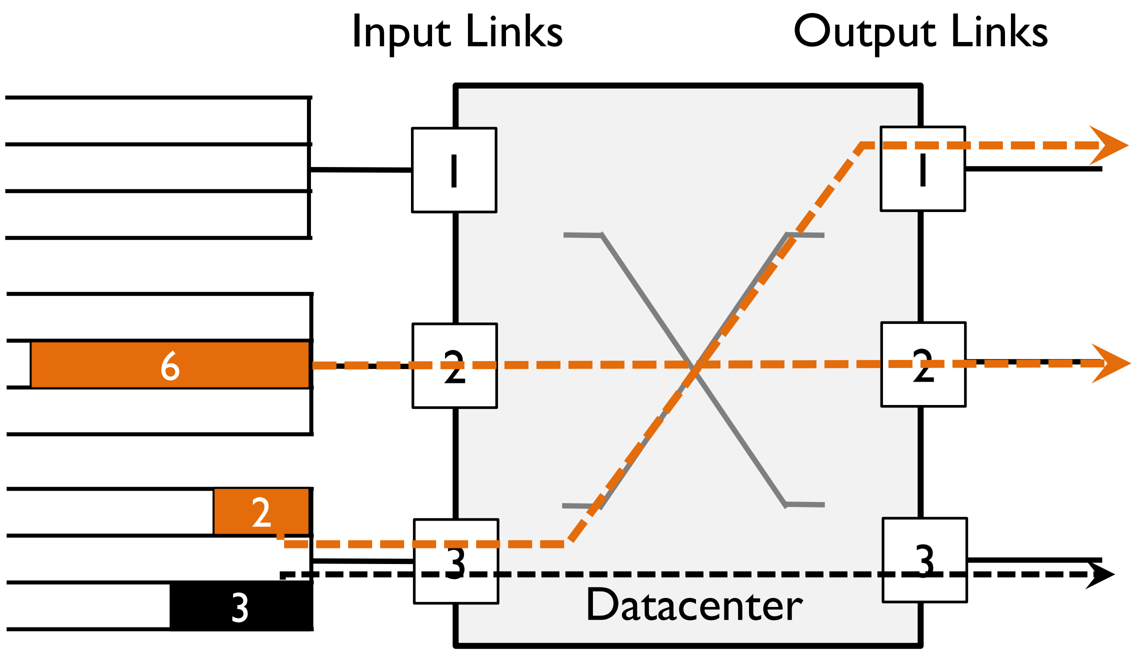
1. Finishing Flows Quickly with Preemptive Scheduling, SIGCOMM'2012.
2. pFabric: Minimal Near-Optimal Datacenter Transport, SIGCOMM'2013.

Inter-Coflow Scheduling is NP-Hard



Concurrent Open Shop Scheduling
with Coupled Resources

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



Many Problems to Solve

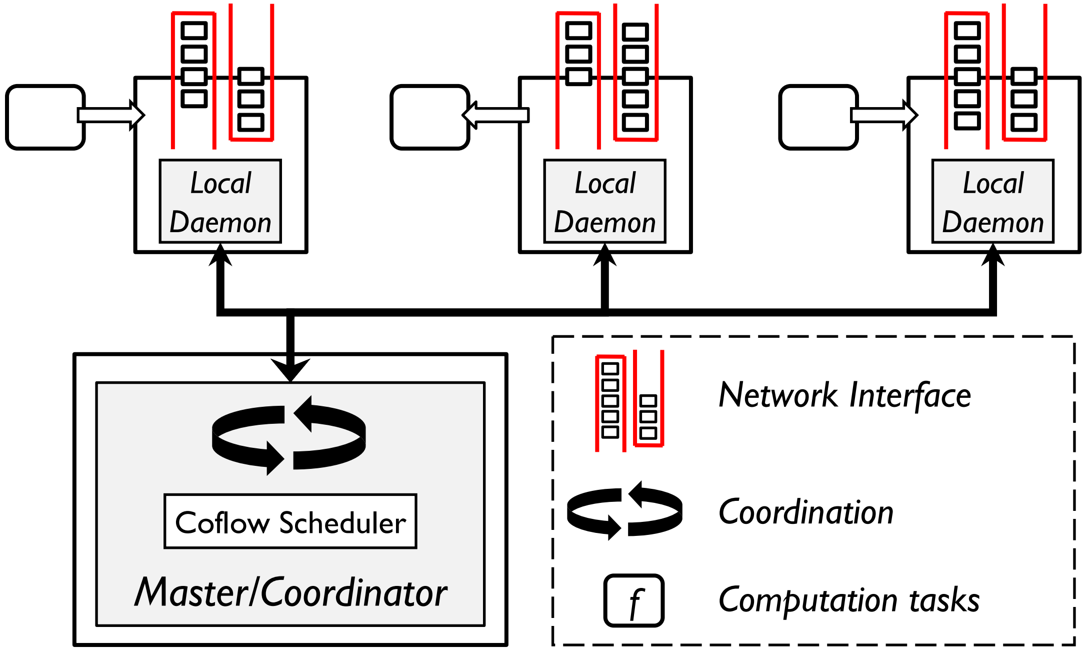
	Clairvoyant	Objective	Optimal
Varys	Yes	Min CCT	No
Aalo	No	Min CCT	No
HUG	No	Fair CCT	Yes

Coflow-Based 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



Coflow API

Change the applications

- At the very least, we need to know what a coflow is
- For clairvoyant versions, we need more information

Changing the framework can enabled ALL jobs to take advantage of coflows

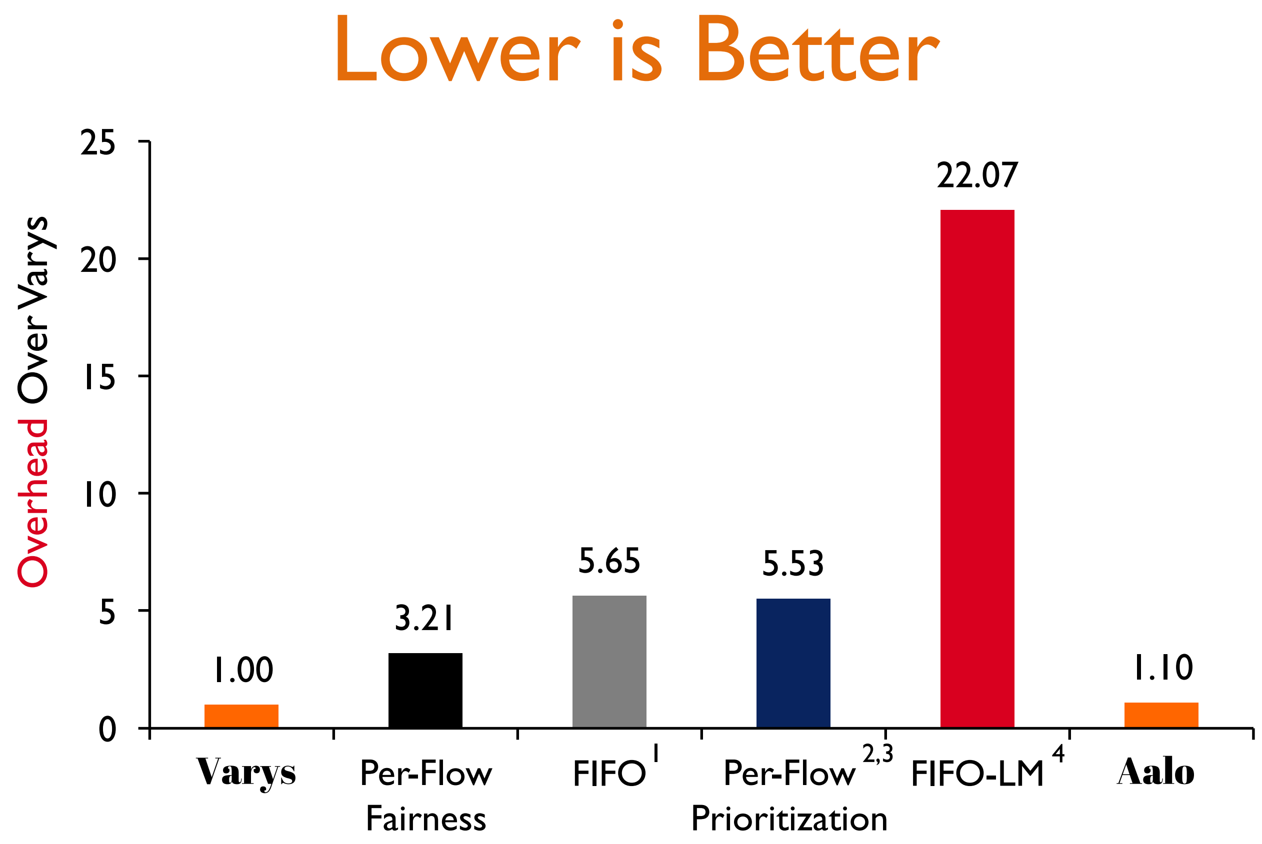
DO NOT change the applications[1]

- Infer coflows from traffic network traffic patterns
- Design robust coflow scheduler that can tolerate misestimations

Our current solution only works for coflows without dependencies; we need DAG support!

1. CODA: Toward Automatically Identifying and Scheduling Coflows in the Dark, SIGCOMM'2016.

Performance Benefits of Using **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*

The Need for Coordination

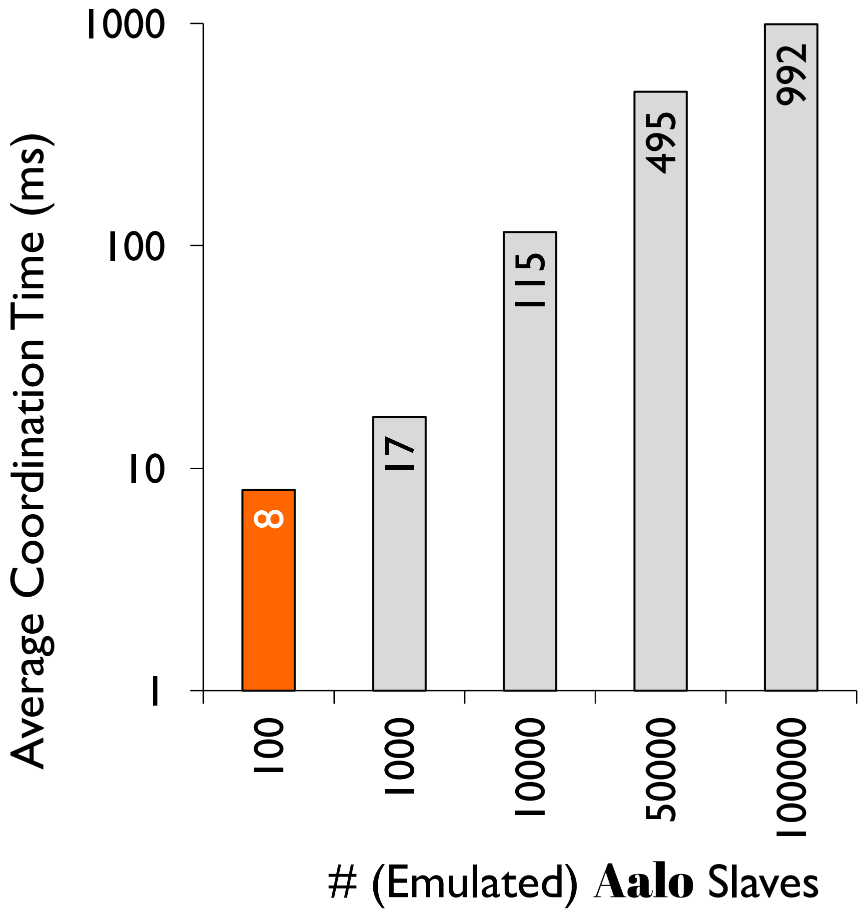
Coordination is necessary to determine realtime

- Coflow size (sum);
- Coflow rates (max);
- Partial order of coflows (ordering);

Can be a large source of overhead

- Does not impact too much for *large coflows in slow networks,* but …

How to perform decentralized coflow scheduling?



Coflow-Aware Load Balancing

Especially useful in asymmetric topologies

- For example, in the presence of switch or link failures

Provides an additional degree of freedom

- During path selection
- For dynamically determining load balancing granularity

Increased need for coordination, but at an even higher cost

Coflow-Aware Routing

Relevant in topologies w/o full bisection bandwidth

- When topologies have temporary in-network oversubscriptions
- In geo-distributed analytics

Scheduling-only solutions do not work well

- Calls for routing-scheduling joint solutions
- Must take network utilization into account
- Must avoid frequent path changes

Increased need for coordination

Coflows in Circuit-Switched Networks

Circuit switching is relevant again due to the rise of optical networks

- Provides very high bandwidth
- Expensive to setup new circuits

Co-scheduling applications and coflows

- Schedule tasks so that we can reuse already-setup circuits
- Perform in-network aggregation using existing circuits instead of waiting for new circuits to be created

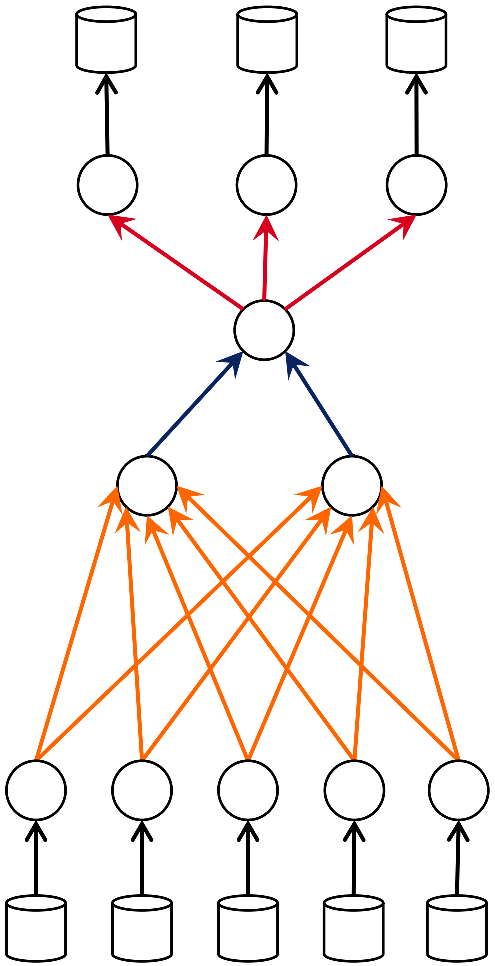
Extension to Multiple Resources[1]

A DAG of coflows is very similar to a job DAG of stages

- Same principle applies, but with new challenges

Consider both fungible (b/w) and non-fungible resources (cores)

- Across the entire DAG



Coflow

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

Key open challenges

1. Better theoretical understanding
2. Efficient solutions to deal with decentralization, topologies, multi-resource settings, estimations over DAG, circuit-switching, etc.

More information

1. Papers: http://www.mosharaf.com/publications/
2. Software/simulator/workloads: https://github.com/coflow