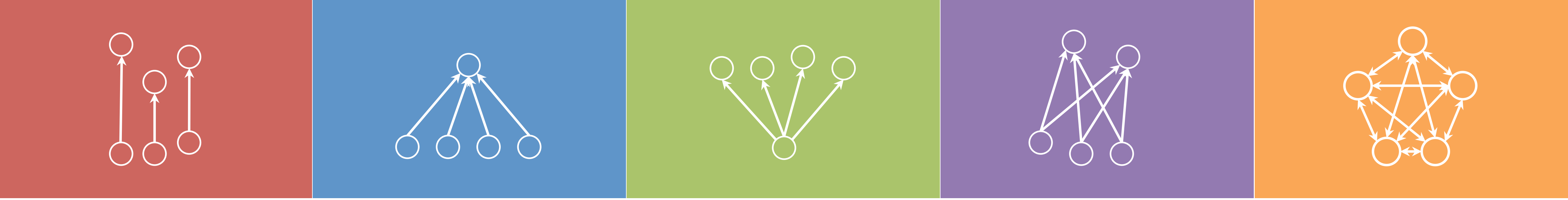


Coflow

Efficiently Sharing Cluster Networks

Mosharaf Chowdhury

Qualifying Exam, UC Berkeley

Apr 11, 2013

Network Matters

Typical Facebook jobs spend 33% of running time in communication

- Weeklong trace of MapReduce jobs from a 3000-node production cluster

Iterative algorithms depends on per-iteration communication time

- Monarch[1] spends up to 40% of the iteration time communicating

Communication often limits scalability

- Recommendation system for the Netflix challenge[2]

1. Design and Evaluation of a Real-Time URL Spam Filtering Service, IEEE S&P'11.

Network Sharing is Well Studied

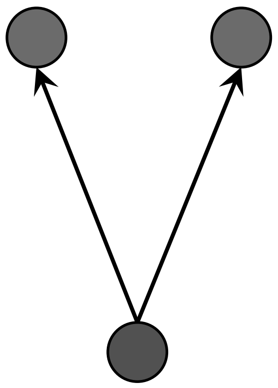
Many articles on different aspects and contexts

- Fairness, efficiency, predictability, and resilience
- Policies, mechanisms, algorithms, architectures, and APIs
- Internet, local area, mobile/wireless, sensor, and datacenters

What is Common?

They use the same abstraction of a *flow*

- A sequence of packets
- Point-to-point
- Endpoints are **fixed**



Each flow is independent

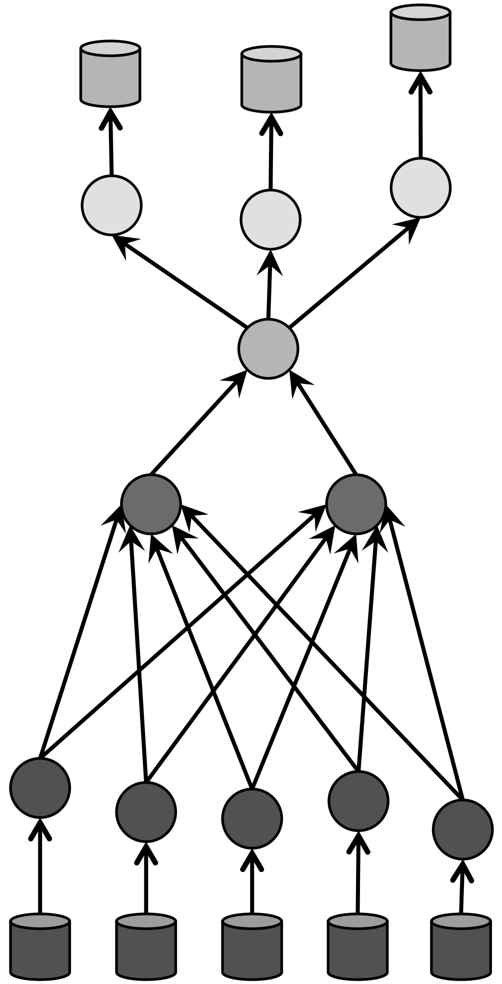
- Unit of allocation, sharing, load balancing etc.

Cluster Networks

Too many flows

Not enough application semantics

- How, if at all, are flows related?
- What does an application care about?
- Must the endpoints of a flow *be fixed?*



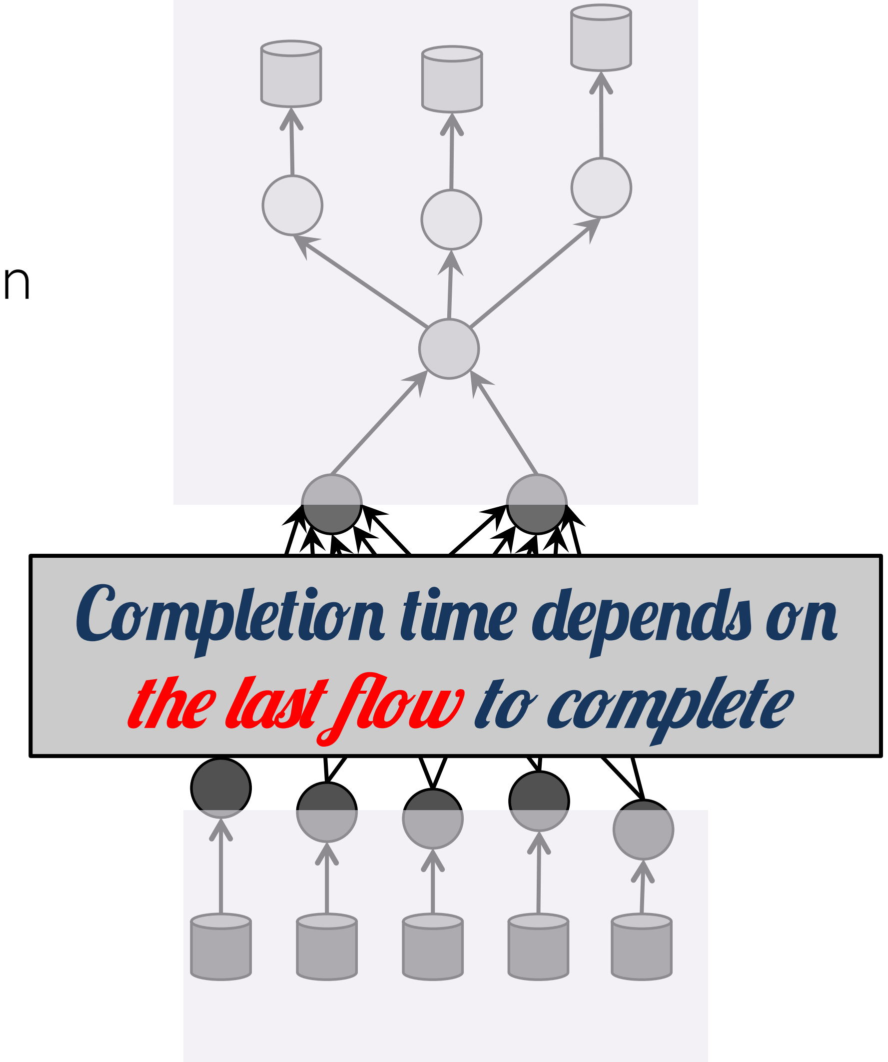
Cluster Applications

Multi-Stage *Data Flows*

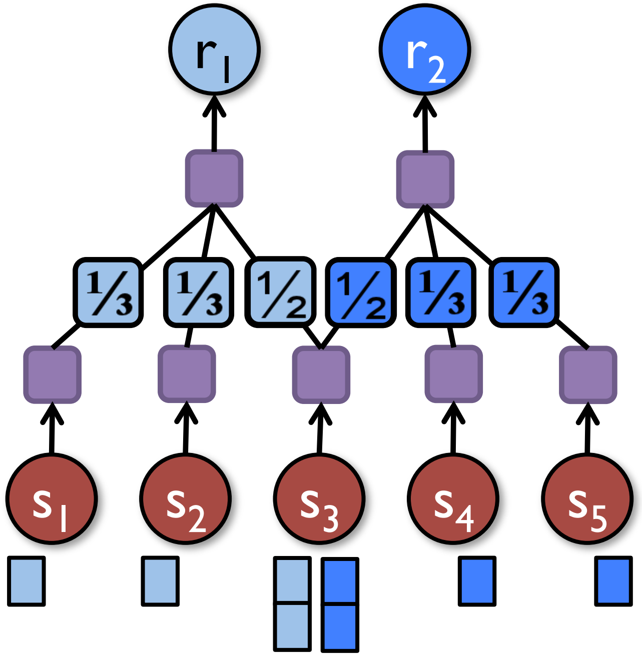
- Computation interleaved with communication
- ***Barriers*** between stages are common

Communication

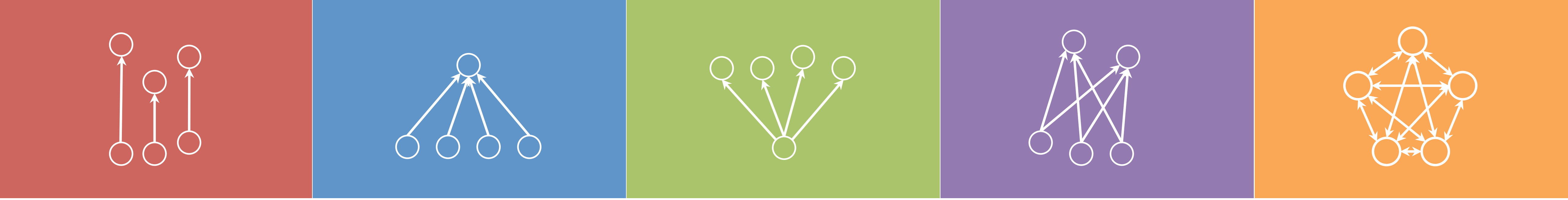
- *Structured*
- Between machine groups



How Does It Change Things?



Links to r_1 & r_2 are full:	3 time units
Link from s_3 is full:	2 time units
Completion time:	5 time units



Coflow

Represents a collection of *one or more flows*

- Captures and conveys an application's intent to the network

+ Performance-centric allocation
+ Flexibility for cluster applications

- Coordination causes complexity

Minimal Coordination [Orchestra[1]]

Micro-management is infeasible in large clusters

- Scaling to O(10K) nodes

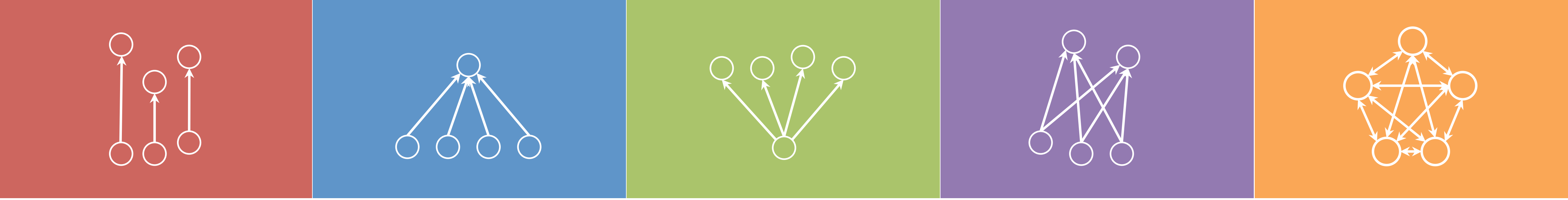
Full decentralization lacks control

- Optimizing individual flows would be an example

Orchestra optimizes individual coflows for applications

- Decentralized broadcast and shuffle algorithms
- Centralized ordering of coflows

1. Managing Data Transfers in Computer Clusters with Orchestra, Appeared at SIGCOMM'11.



Coflow

Represents a collection of *one or more flows*

+ Performance-centric allocation
+ Flexibility for cluster applications

- Coordination causes complexity
- Fixed endpoints are restrictive

Endpoint Flexible Transfers [Usher[1]]

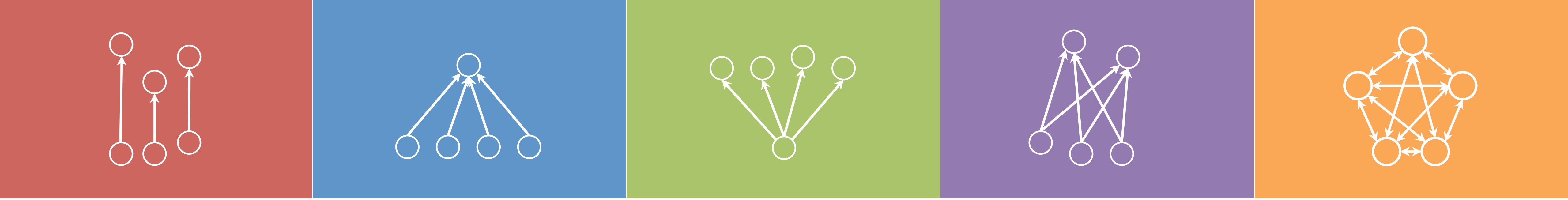
Communication always takes place between fixed endpoints

- The network does not determine the placement

Usher enables constrained anycast

- Takes constraints from applications like distributed file systems
- Dictates applications where to put the destination
- Decreases network imbalance and makes other coflows faster

1. Leveraging Flexibility in Endpoint Placement for a Snappier Network, Submitted to SIGCOMM'13.



Coflow

Represents a collection of *one or more flows*

+ Performance-centric allocation
+ Flexibility for cluster applications

- Coordination causes complexity
- Fixed endpoints are restrictive
- Managing concurrent coflows

Outline

1. The case for flow coordination
2. Optimizing individual coflows
3. Flexible endpoint placement
4. Managing coexisting coflows

Outline

1. The case for flow coordination
2. Optimizing individual coflows
3. Flexible endpoint placement
4. Managing coexisting coflows

Orchestra

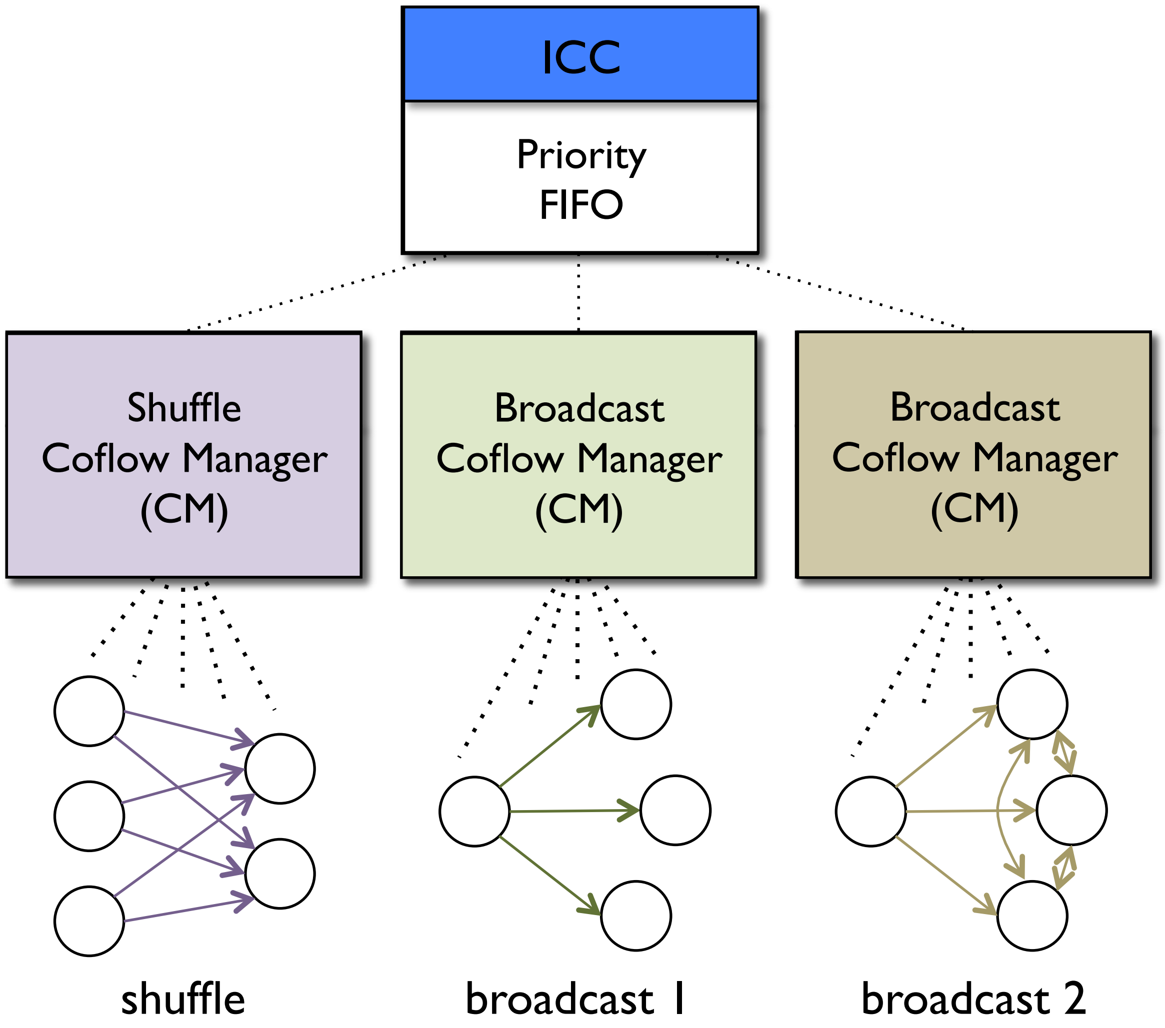
Optimize at the level of coflows instead of individual flows

A coflow manager (CM) selects appropriate algorithm based on

- Number of participants,
- Size of data,
- Level of oversubscription

Inter-coflow coordinator (ICC)

- Enforces simple ordering between coflows

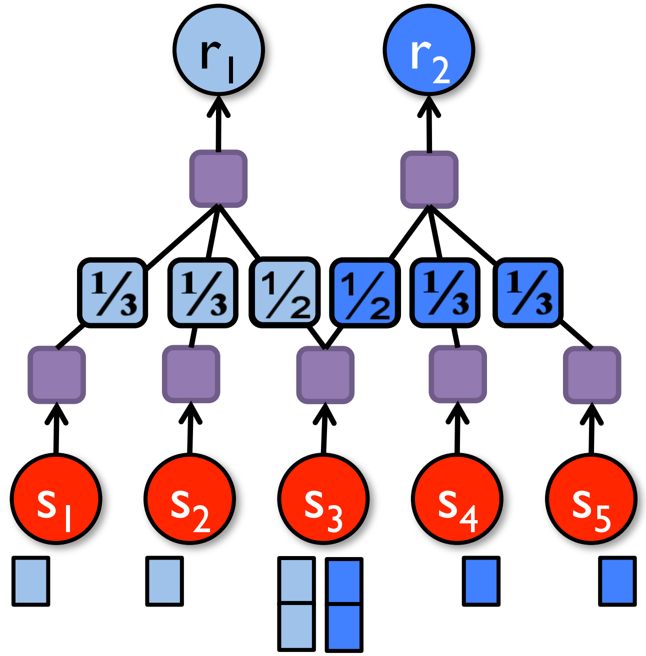


Many-to-Many/Shuffle *Status Quo*

Transfers output of one stage to be used as input of the next

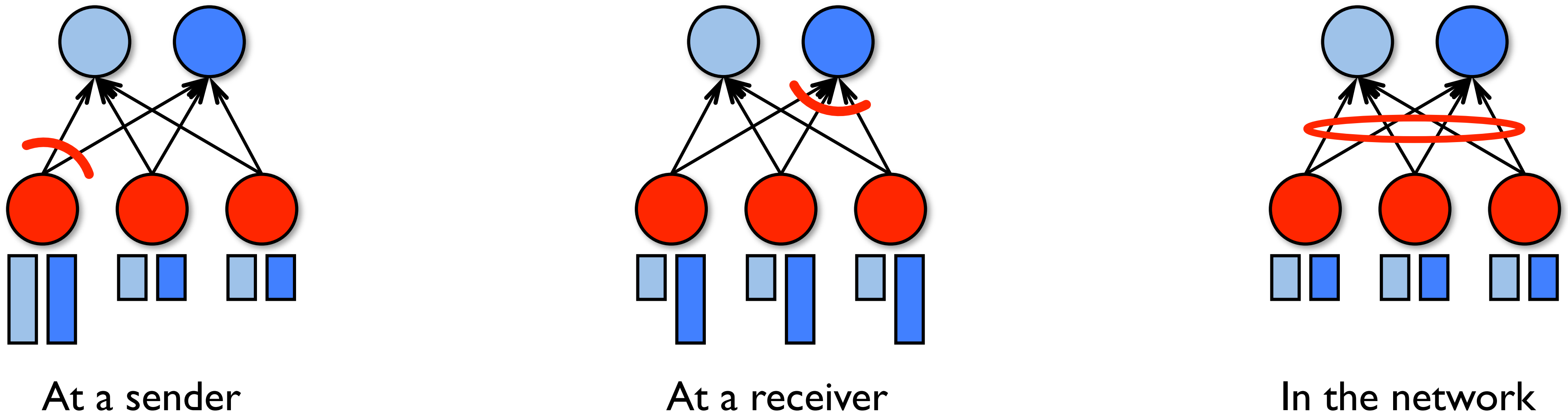
Widespread use

- **All** MapReduce jobs at Facebook
- Any SQL query that joins or aggregates data



Links to r_1 and r_2 are full:	3 time units
Link from s_3 is full:	2 time units
Completion time:	5 time units

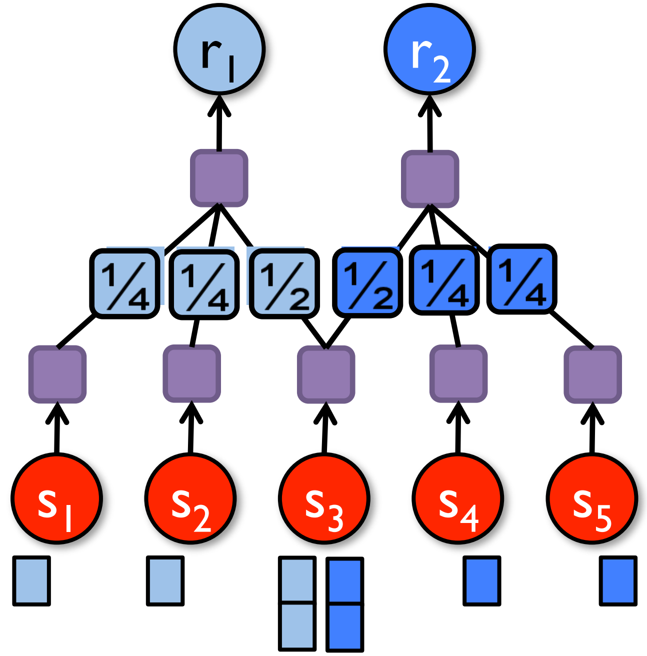
Shuffle Bottlenecks



An optimal shuffle schedule keeps at least one link fully utilized throughout the transfer

Weighted Shuffle Scheduling (WSS)

Allocate rates to each flow, proportional to the total amount of data it transfers



Completion time: 4 time units

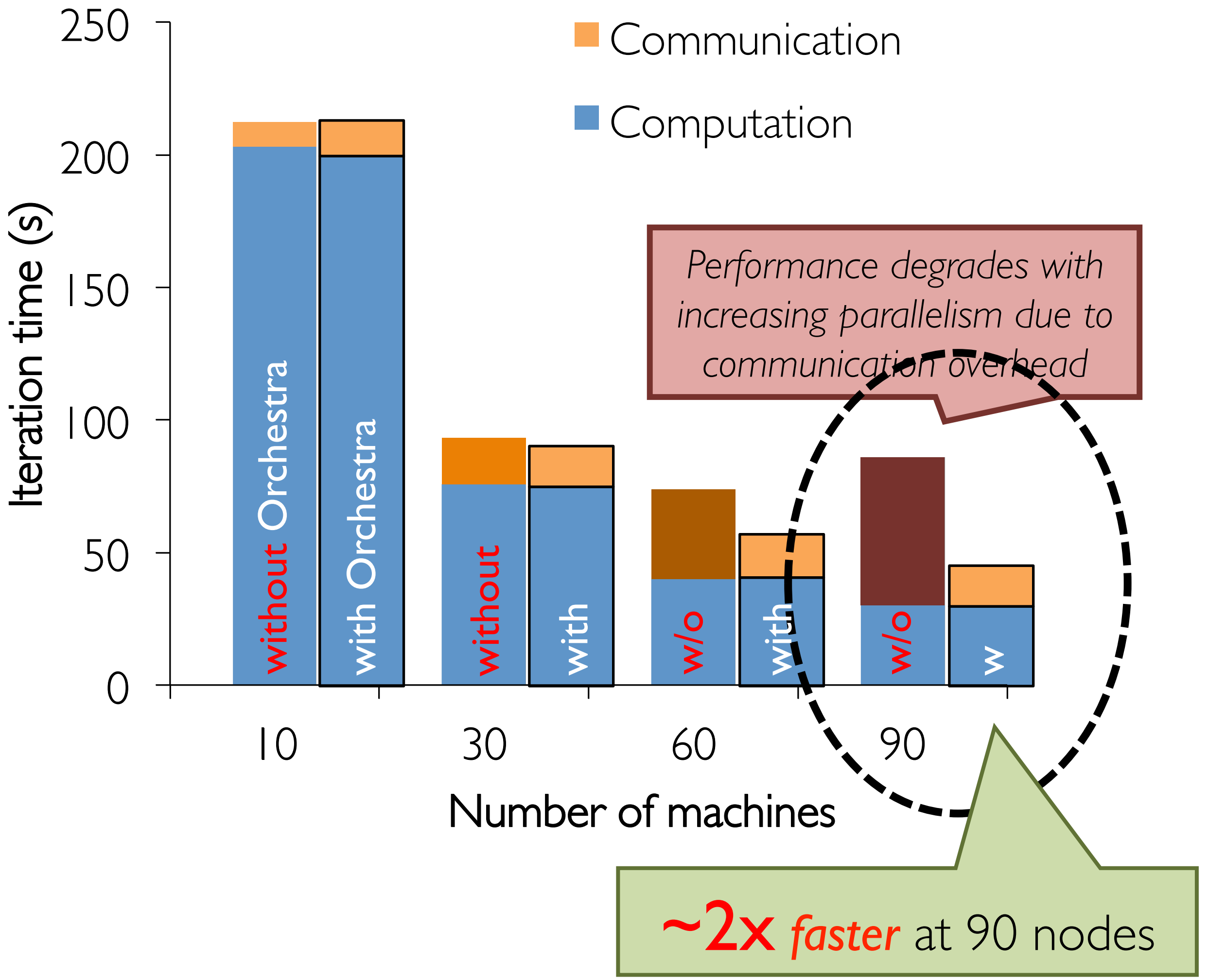
Up to 1.5X improvement

Orchestra in Action : Netflix Challenge

Movie recommendation system using collaborative filtering

Implemented in Spark

Better *scaling* characteristics



What About Other Coflows?

Broadcast/One-to-Many

- Cooperative BitTorrent
- **4.5X** faster than the status quo

Aggregation/Many-to-One

- Direct application of WSS

AllReduce

- Heavily used in matrix-based computations (e.g., machine learning)
- Aggregates data to a single node, then broadcasts to everyone

Outline

1. The case for flow coordination
2. Optimizing individual coflows
3. Flexible endpoint placement
4. Managing coexisting coflows

Distributed File Systems

Pervasive in BigData clusters

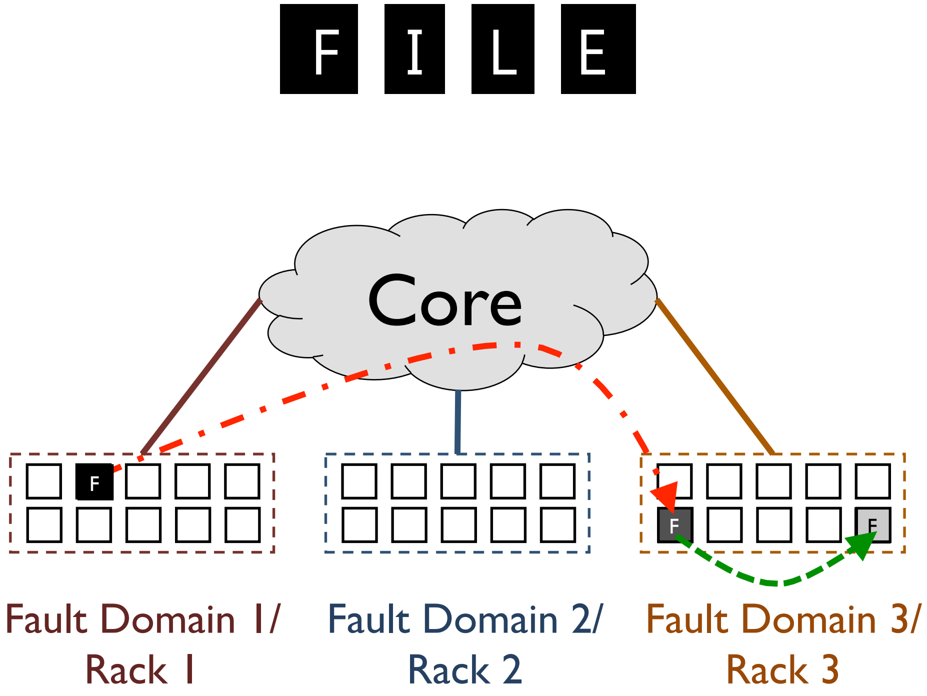
- Different frameworks read from and write to the same DFS

Files are divided into blocks

- Typically 256MB blocks

Each block is replicated to

- 3 machines for *fault-tolerance*
- 2 fault domains for *partition-tolerance*
- Uniformly randomly



Locations do not matter
as long as constraints are met

Network-Aware Replica Placement

Constrained anycast

- Destination of the transfer is determined by the network
- Move replication traffic out of the way of coflows

Will network-awareness matter? YES

- More than **40%** of all network traffic comes from DFS replication
- Almost **50%** of the time downlinks have high imbalance[1] ($C_v > 1$).[2]

Does it matter to DFS clients/users? YES

- More than **37%** of all tasks write to the DFS.

1. Imbalance considering all cross-rack bytes. Calculated in 10s bins.
2. Coefficient of variation, C_v = (stdev/mean).

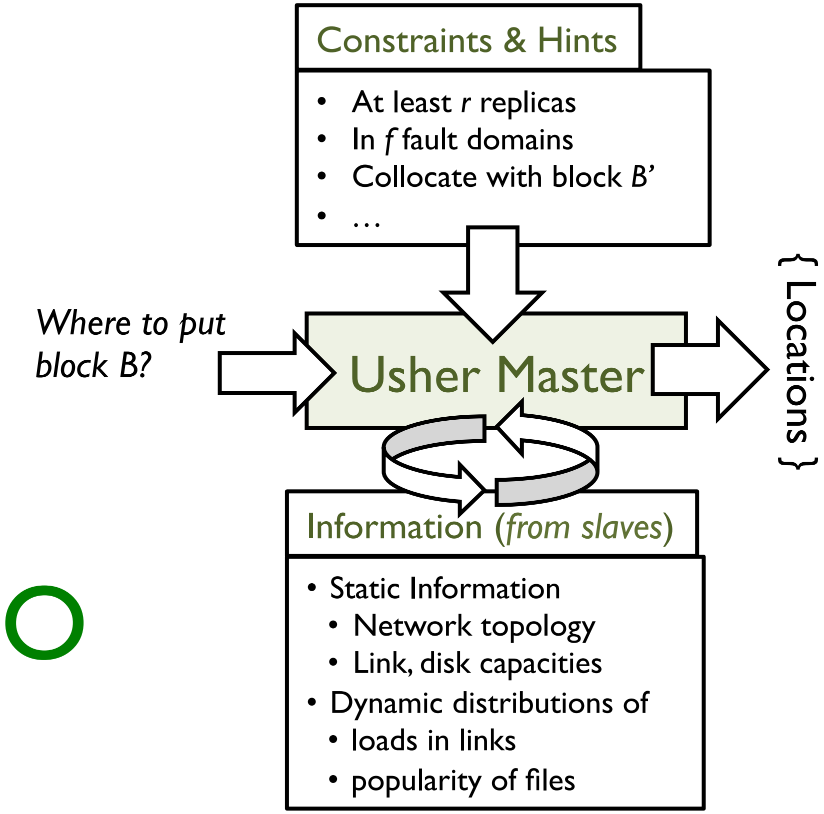
Usher Overview

Performs network-aware replica placement

Takes online decisions

Decreases network imbalance

Does it impact the storage balance? NO



Why Not?

Greedy placement is optimal under these conditions

	Observations	Implications
1	Network hotspots are stable in the short term (5-10 sec)	Individual blocks can be used for packing[1]

1. It takes 5 seconds to write a 256MB block, which is shorter than most hotspot durations.

Faster. More Balanced.

Implemented and integrated with HDFS

- Pluggable replica placement policy

EC2 Deployment

Jobs run 1.26X faster

Blocks written 1.3X faster

Facebook Trace Simulation

Jobs run 1.39X faster

Blocks written 1.58X faster

Upper bound of the optimal is 1.89X

The network became more balanced

Storage remained balanced

Future Research

Applications of Constrained Anycast

- Rebuilding of lost blocks for erasure-coded storage systems
- Input collocation to decrease network traffic instead of just load balancing
- Read from non-local storage depending on contention

In-Memory Storage Systems

- Network is the bottleneck for memory-to-memory communication

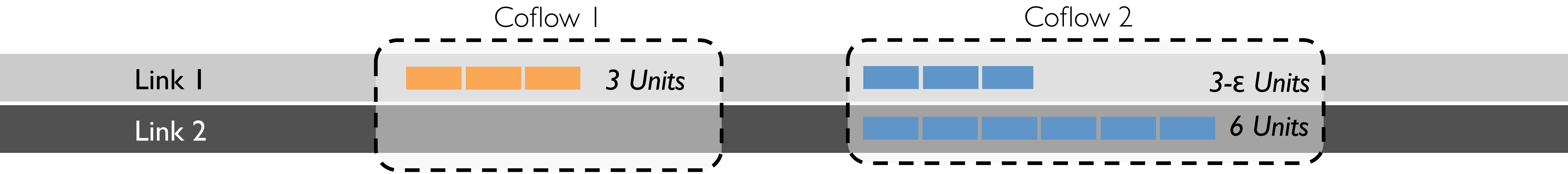
DFS Read/Write Coflows

- Collection of parallel flows

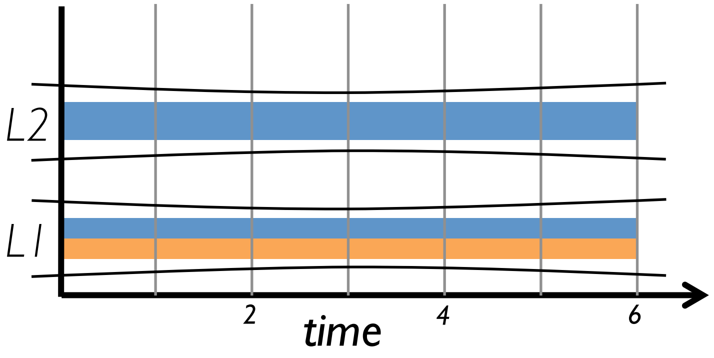
Outline

1. The case for flow coordination
2. Optimizing individual coflows
3. Flexible endpoint placement
4. Managing coexisting coflows

Why Inter-Coflow Coordination?

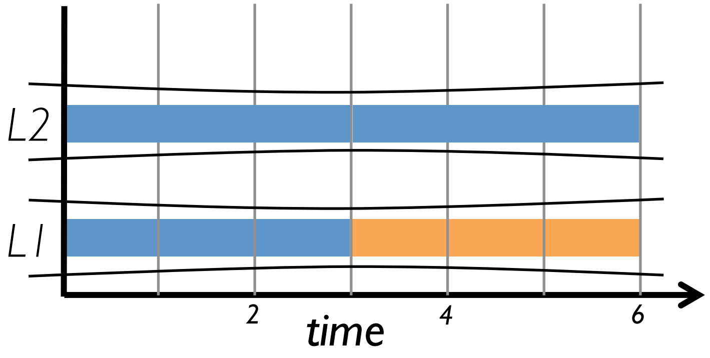


Fair Sharing



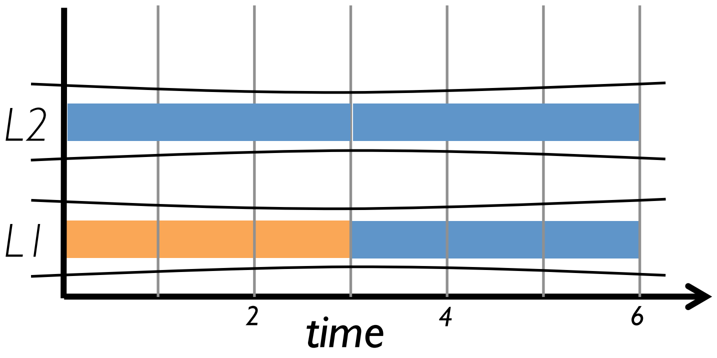
Coflow1 comp. time = 6
Coflow2 comp. time = 6

Flow-level Prioritization[1]



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

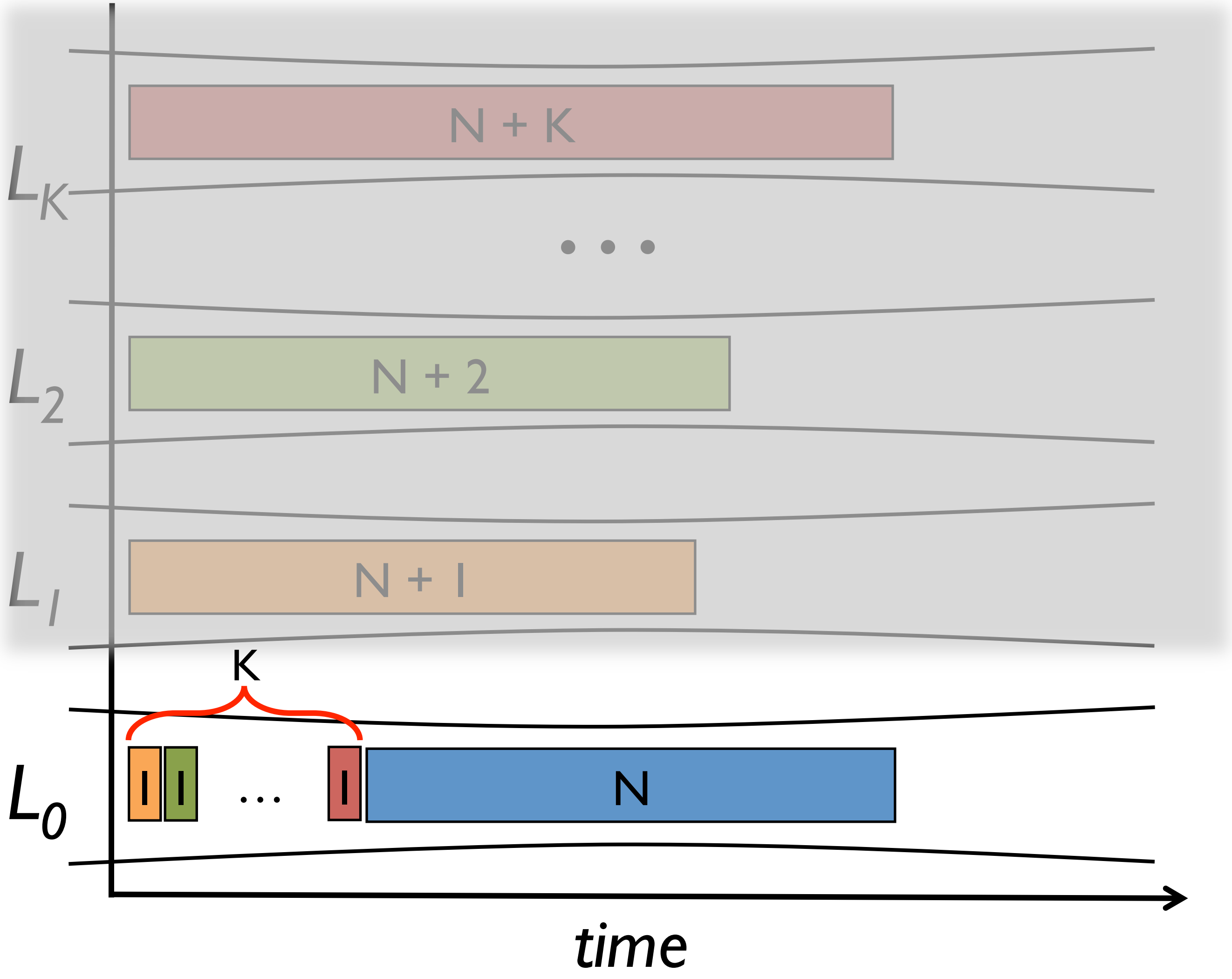
The Optimal



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

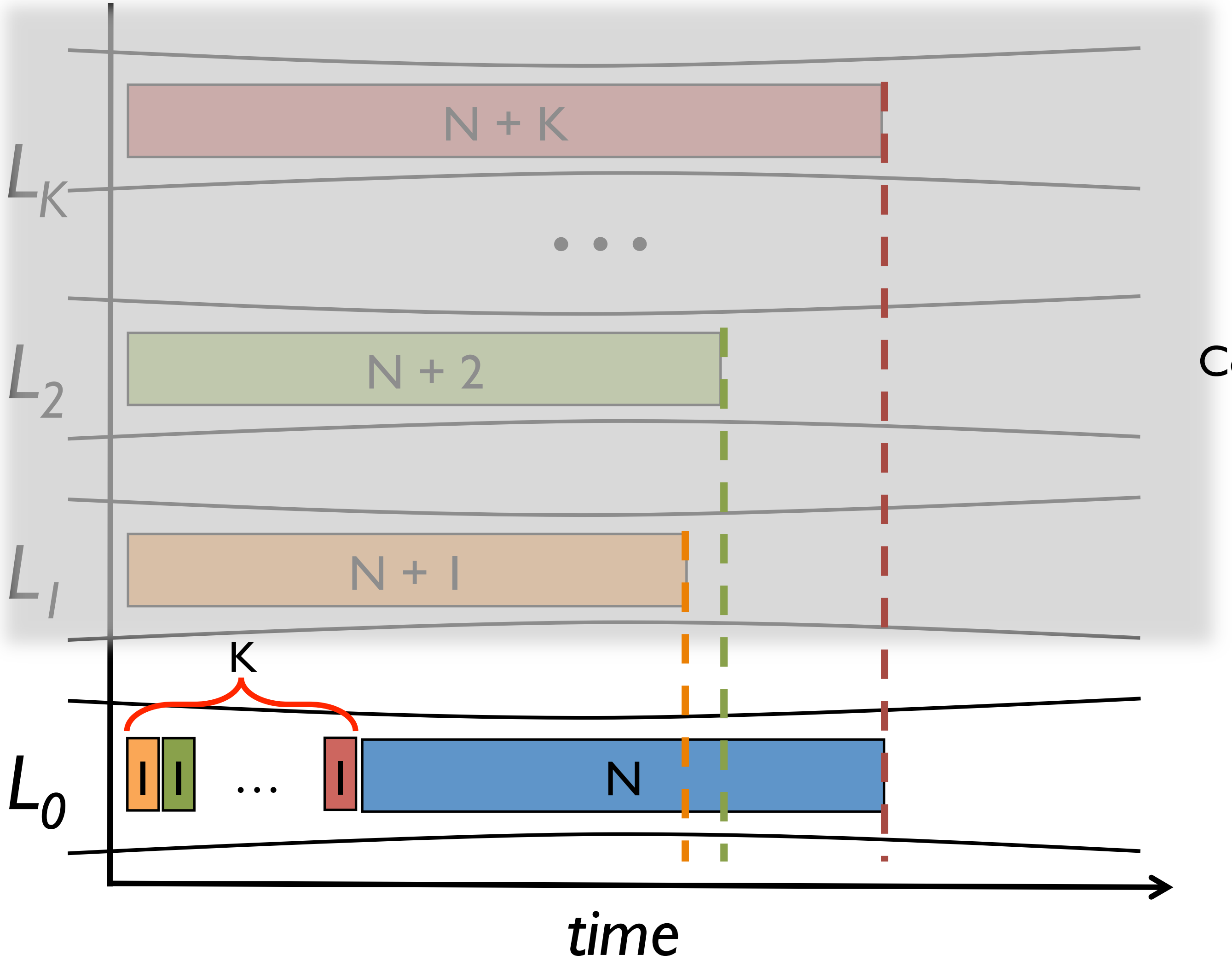
1. Finishing Flows Quickly with Preemptive Scheduling, SIGCOMM'12.

How Much Better Can We Do?



Completion time of the blue coflow considering only L_0 $= K + N$

How Much Better Can We Do?



Completion time of the
blue coflow considering only L_0 $= K + N$

Completion time considering all links $= N$

Improvement $= \frac{K}{N} + 1$

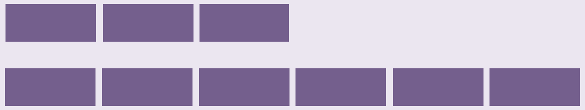
No change for other coflows

What is the optimal order of coflows?

NP-Hard

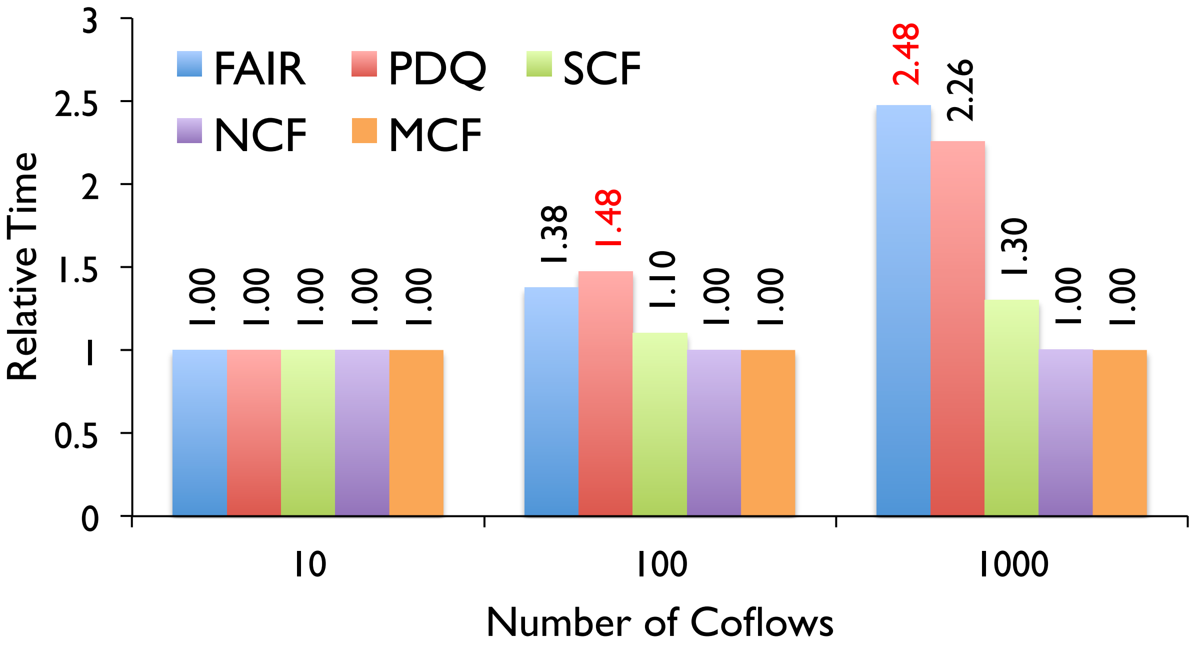
Preliminary Simulation

Length	Size of the largest flow
Width	Total number of flows
Size	Sum of all flows



Length	6
Width	2
Size	9

FAIR	Fair sharing on each link
PDQ	Shortest flow first
SCF	Shortest coflow first
NCF	Narrowest coflow first
MCF	Smallest coflow first



Simulated on 100 links
Width of coflows varied from 1 to 100
Length of each flow varied from 1 to 10
Offline, i.e., all coflows arrive at the beginning
Averaged over 25 runs

Summary

The network is a key resource in cluster computing

- Unlike other resources, it remains agnostic to application requirements

We proposed the *coflow* abstraction and three components to

- Optimize common coflows in isolation (Orchestra)
- Balance the network using constrained anycast (Usher)
- Express and schedule concurrent coflows (Maestro)

Related Work

MPI Communication Primitives

- No coordination among coflows

Cloud and HPC Schedulers

- Limited to *independent* resources like computing and memory; ignore the network

Full Bisection Bandwidth Networks

- Mechanism for faster network, not for better management within/across apps

Distributed File Systems

- Ignore the network even though generate a large chunk of cluster traffic

Software-Defined Networking

- Provides control plane abstractions and can act as an enabler of coflows

Timeline

April 2013 to September 2013

- Develop a fast approximation algorithm for inter-coflow scheduling
- Implement the ICC in the application layer
- Port communication patterns in *Spark* and *Hadoop* to the coflow API

October 2013 to April 2014

- Explore the notion of *fairness* among coflows
- Implement the AllReduce coflow

May 2014 to December 2014

- Apply constrained anycast to other contexts
- Complete an SDN integration of the coflow API

Why Are We So Excited?

Task scheduling in data centers

- Tasks without data locality constraints (e.g., reducer stage)

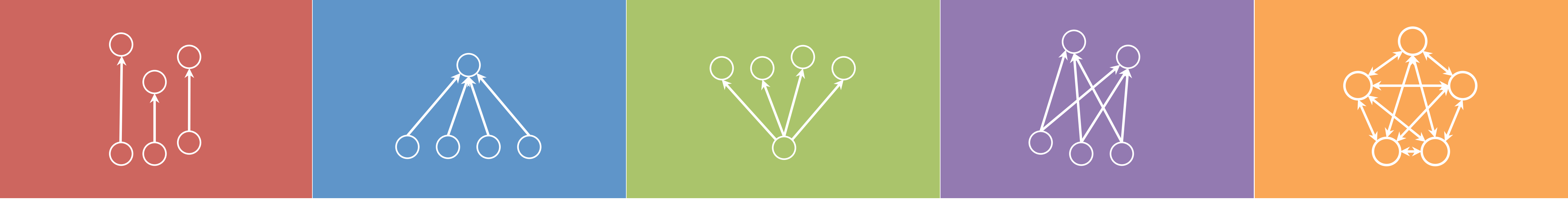
Sub-resource prioritization in SPDY[1]

- We can design SPDR ;)

Many-core systems

- Scheduling memory requests in shared DRAM systems[2]
- Coordinated communication across multiple cores

1. SPDY Protocol Specification, http://www.chromium.org/spdy/spdy-protocol.
2. Distributed Order Scheduling and its Application to Multi-Core DRAM Controllers, PODC'08.



Coflow

Use it!

Mosharaf Chowdhury
http://www.mosharaf.com/

BACKUP

Communication Matters

Typical job in Facebook spends 33% of running time in the shuffle phase

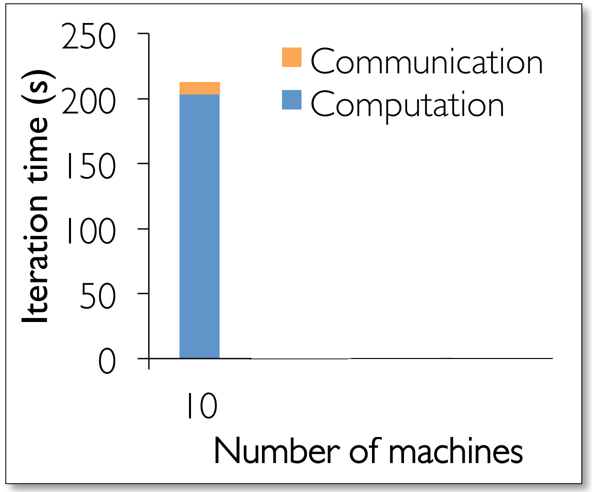
- Weeklong trace of MapReduce jobs from a 3000-node production cluster

Iterative algorithms depends on per-iteration communication time

- Monarch[1] spends up to 40% of the iteration time in shuffle

Communication often limits scalability

- Recommendation system for the Netflix challenge[2]



1. *Design and Evaluation of a Real-Time URL Spam Filtering Service, IEEE S&P'11.*

Network Sharing is Well Studied

Many articles on different aspects of network sharing and allocation

- Policies, mechanisms, algorithms, architectures, APIs, fairness, performance etc.

Many articles on sharing different types of networks

Google Scholar Query	Number of Results
network sharing +**"internet"**	1,420,000
network sharing +**"mobile"**	808,000
network sharing +**"wireless"**	407,000
network sharing +**"sensor"**	140,000
network sharing +**"local area"**	134,000
network sharing +**"wide area"**	93,400
network sharing +**"vehicular"**	36,000
network sharing +**"data center"**	26,000

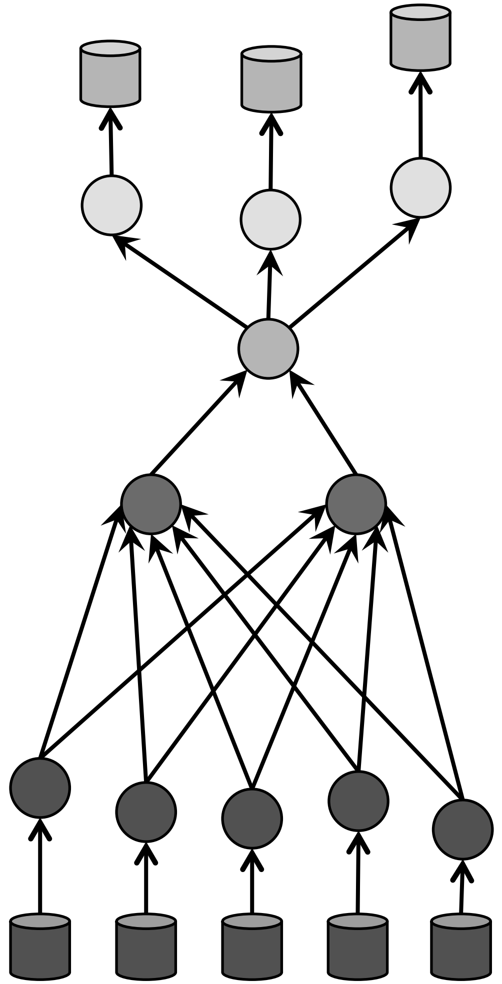
Cluster Applications

Multi-Stage *Data Flows*

- Computation interleaved with communication
- ***Barriers*** between stages are common

Communication

- *Structured*
- Between machine groups



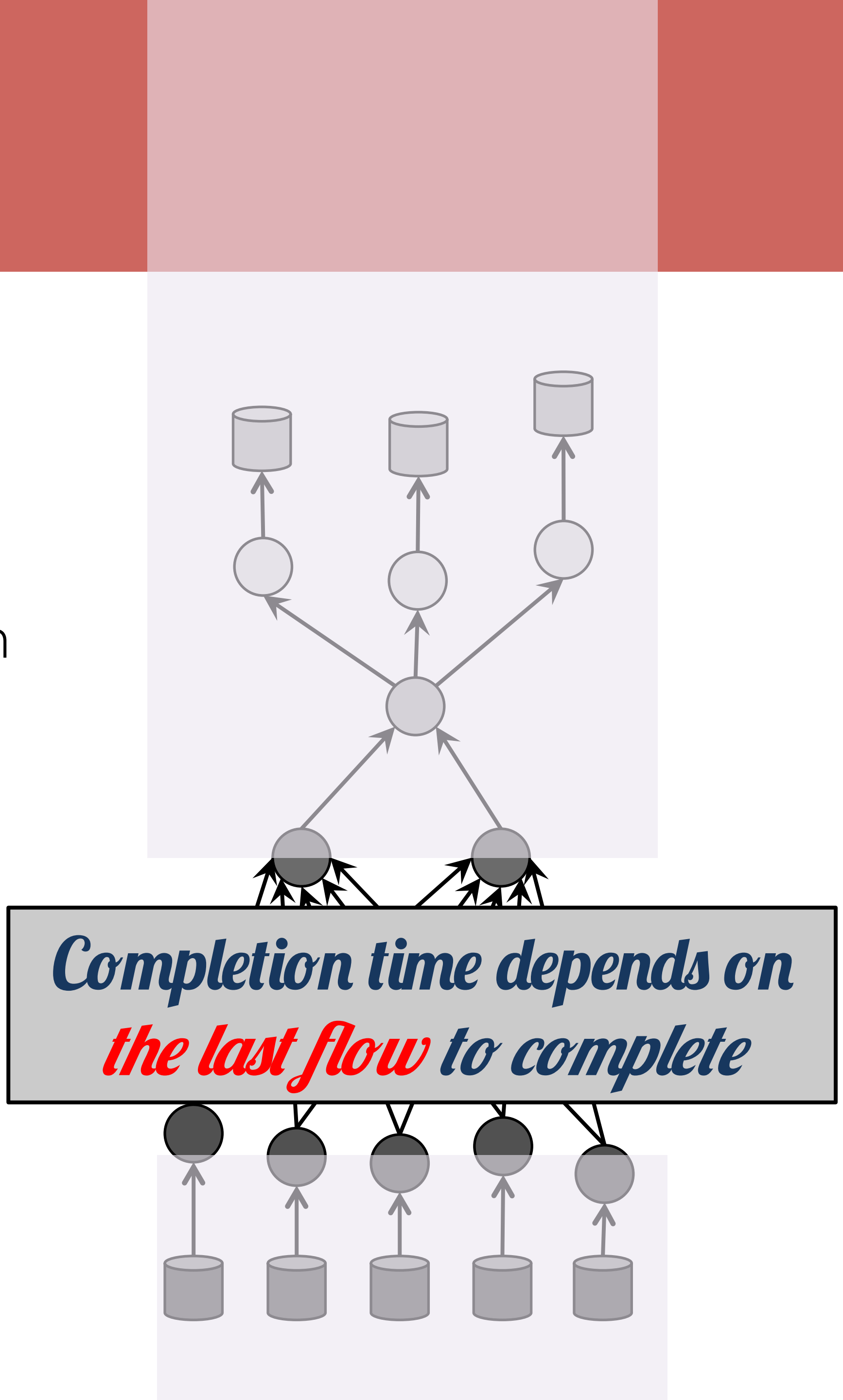
Cluster Applications

Multi-Stage *Data Flows*

- Computation interleaved with communication
- ***Barriers*** between stages are common

Communication

- *Structured*
- Between machine groups



Cooperative Broadcast

Send the same data to all receivers

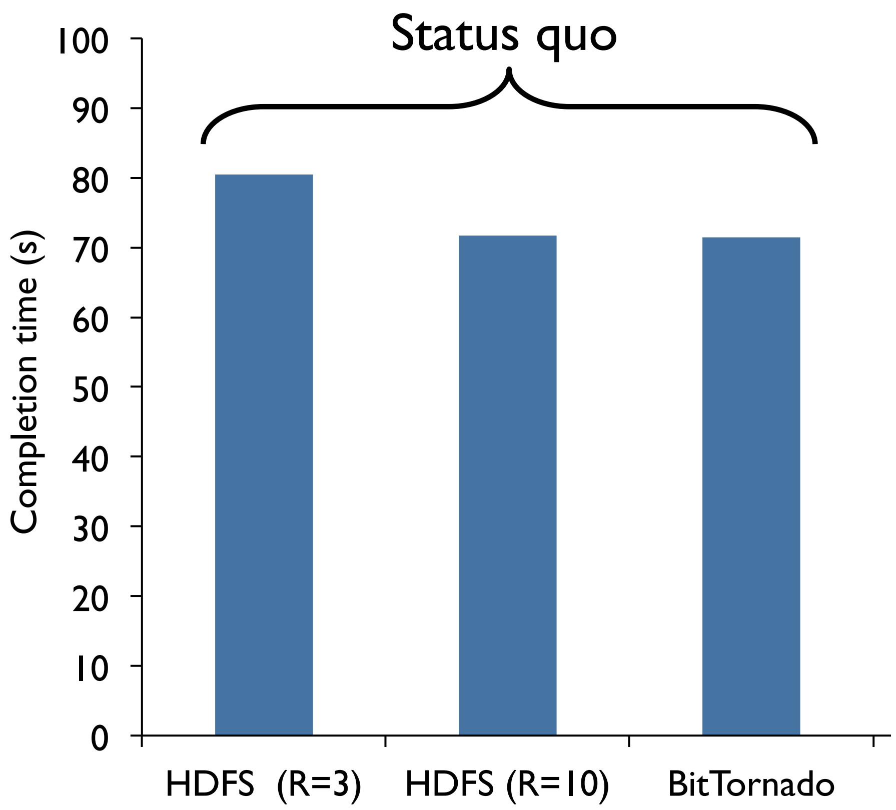
- Fast, scalable, and resilient

Peer-to-peer mechanism optimized for cooperative environments

	Observations	Design Decisions
1	High-bandwidth, low-latency network	✓ Large block size (4-16MB)

Performance

1GB data to 100 receivers on EC2



Up to **4.5X** faster than status quo
Ships with *Spark*

Not so much faster for

- Small data (<10MB)
- Fewer receivers (<10)

Additional **2X** speedup with topology info

Topology-Aware Broadcast

Up to 2X faster than vanilla implementation

Many data center networks employ tree topologies

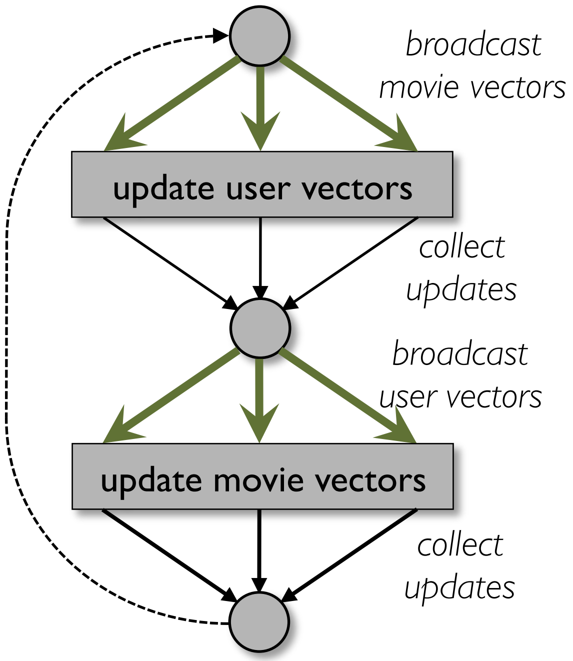
Each rack should receive exactly one copy of broadcast

- Minimize cross-rack communication

Topology information reduces cross-rack data transfer

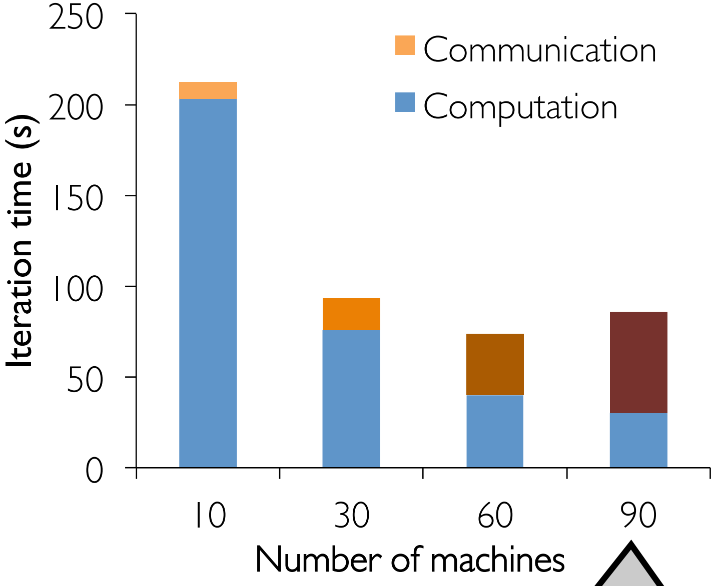
- Mixture of spherical Gaussians to infer network topology

Orchestra in Action



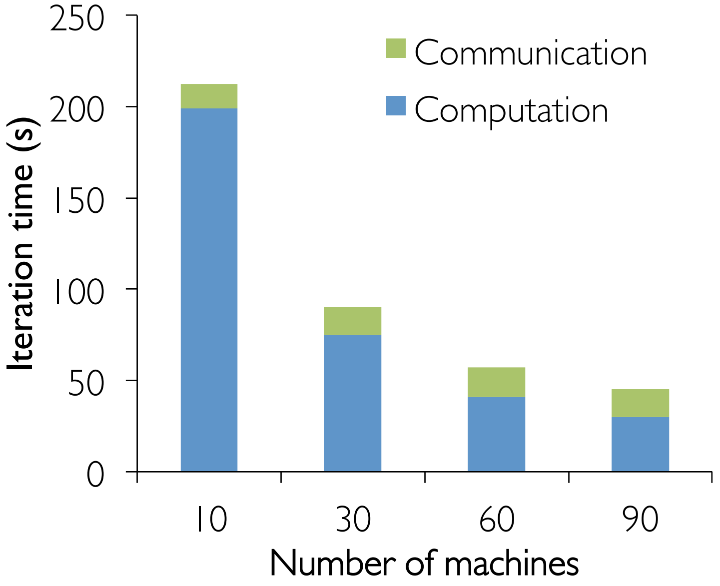
Collaborative Filtering
using ***Alternating Least Squares***

Without Orchestra



Performance degrades with increasing parallelism due to communication overhead

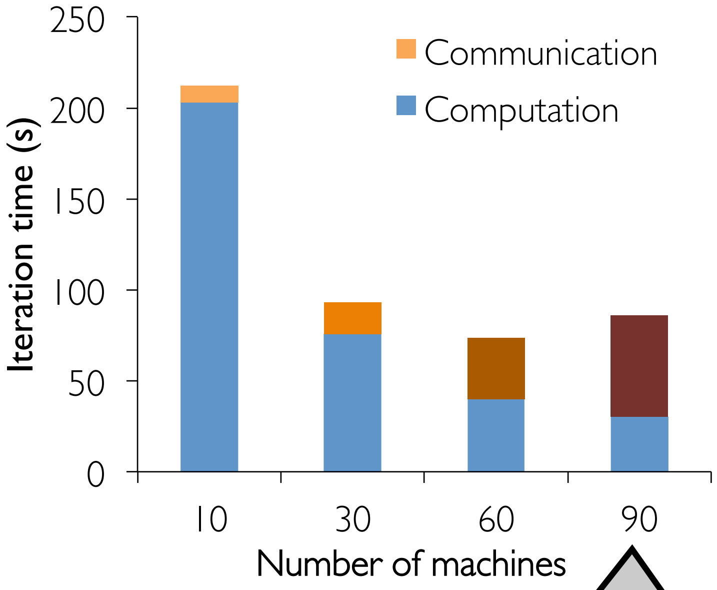
With Orchestra



~2x *faster* at 90 nodes[1]

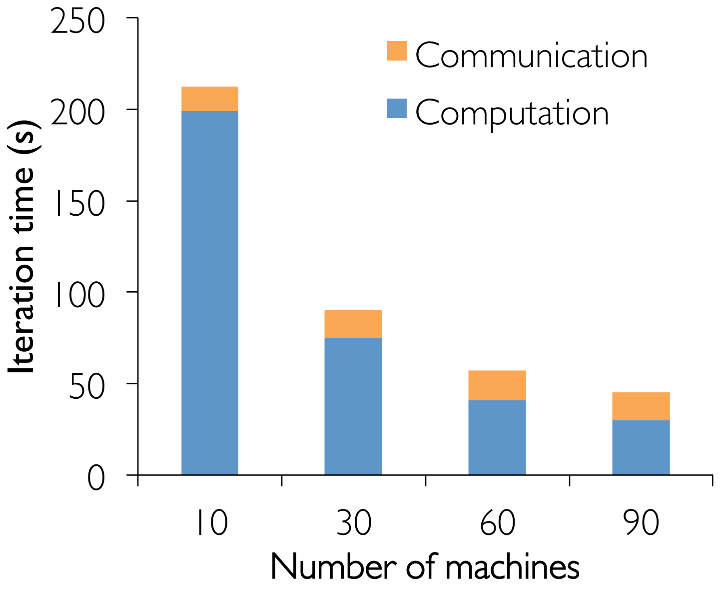
Orchestra in Action : Netflix Challenge

Without Orchestra



Performance degrades with increasing parallelism due to communication overhead

With Orchestra



~2x *faster* at 90 nodes

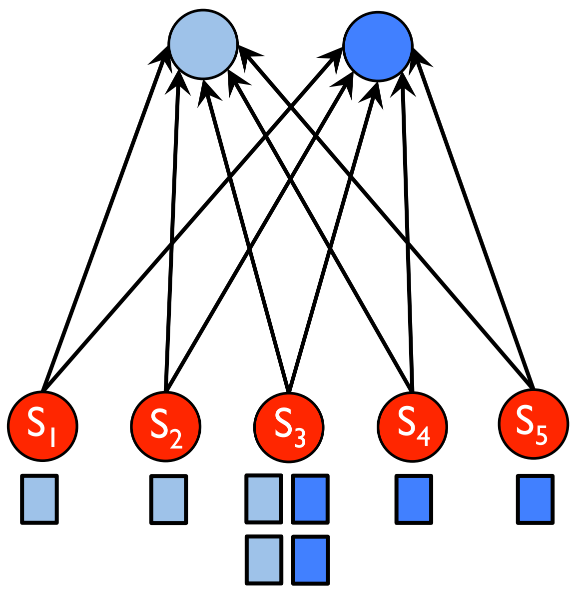
Shuffle

Status Quo

Transfers output of one stage to be used as input of the next

Widespread use

- 68% of the Facebook jobs use shuffle



R_1 and R_2 are bottlenecks:	3 time units
S_3 is the bottleneck:	2 time units
Completion time:	5 time units

Benefits of the Coordinator

Shuffle on a 30-node EC2 cluster

Two priority classes

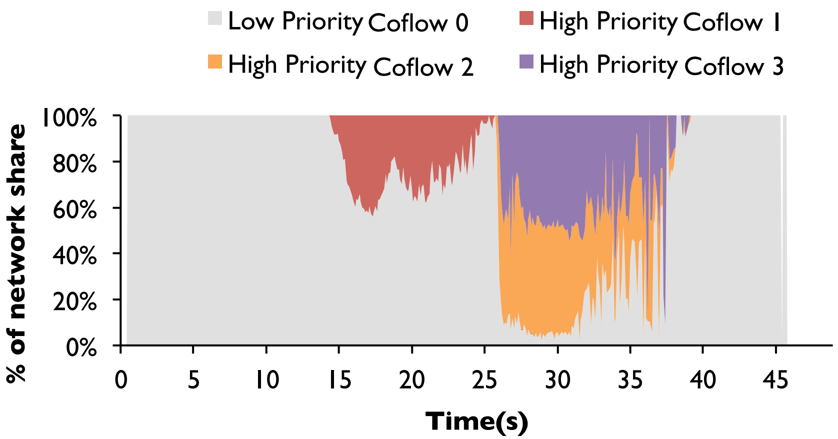
- FIFO within each class

Low priority coflow

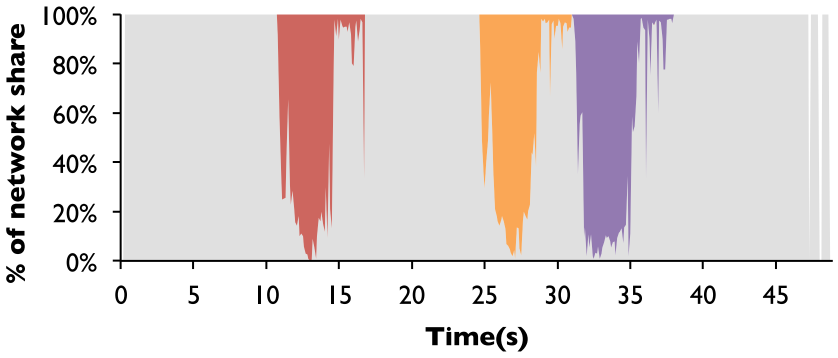
- 2GB per reducer

High priority coflows

- 250MB per reducer



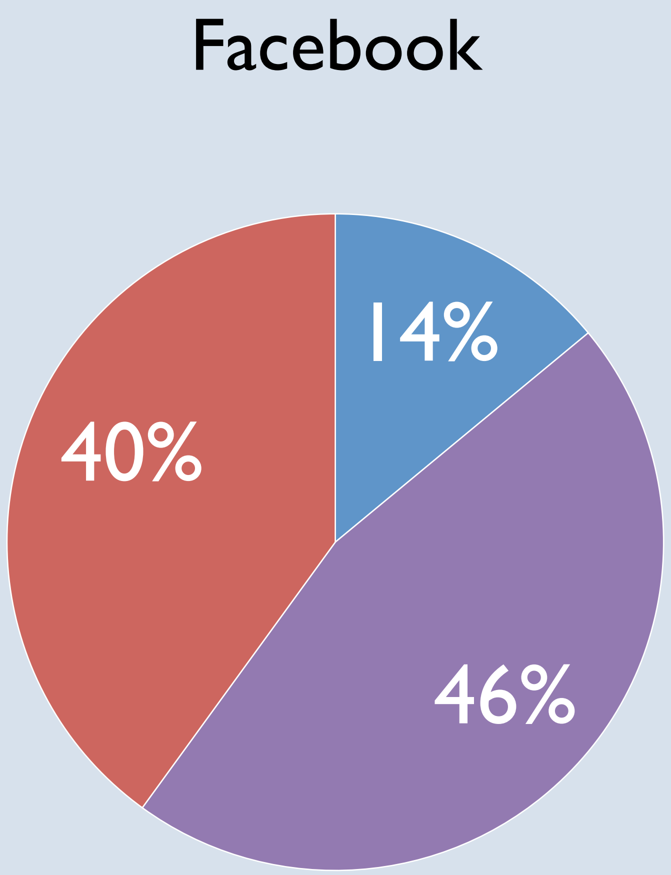
Without Inter-Coflow Scheduling

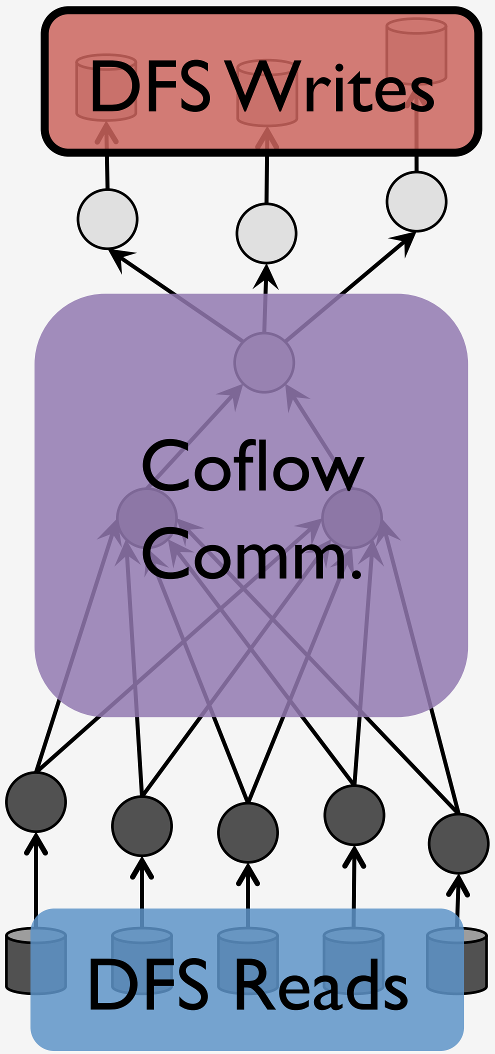


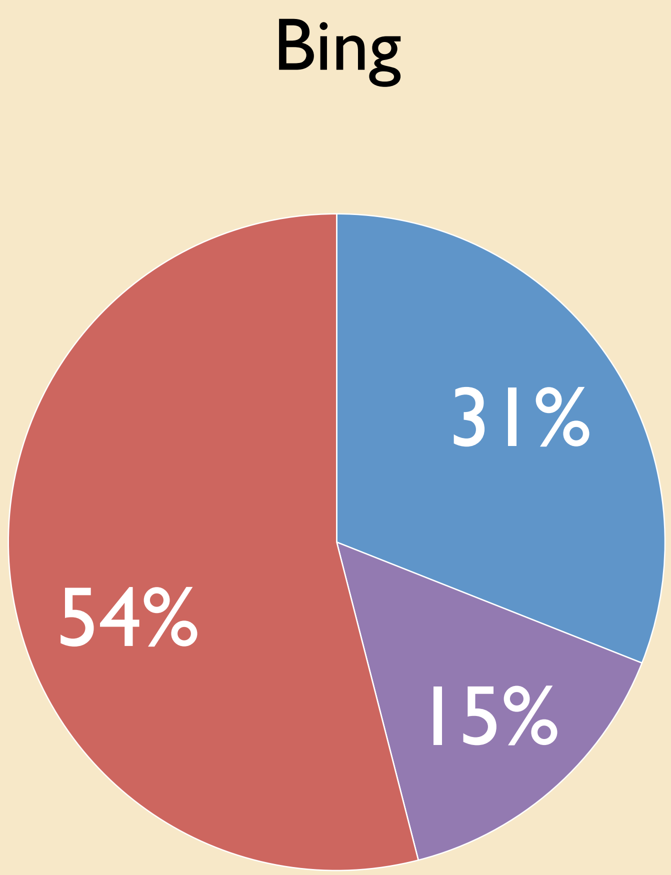
Priority Scheduling in the ICC

1.75X faster high priority coflows
1.06X slower low priority coflow

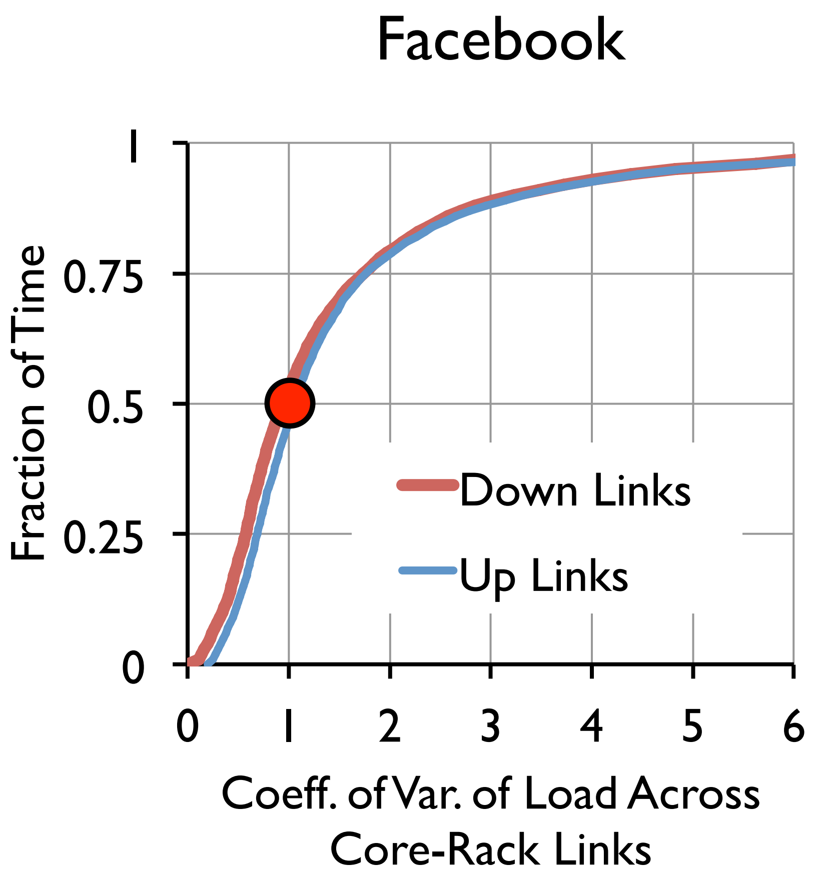
Sources of Network Traffic



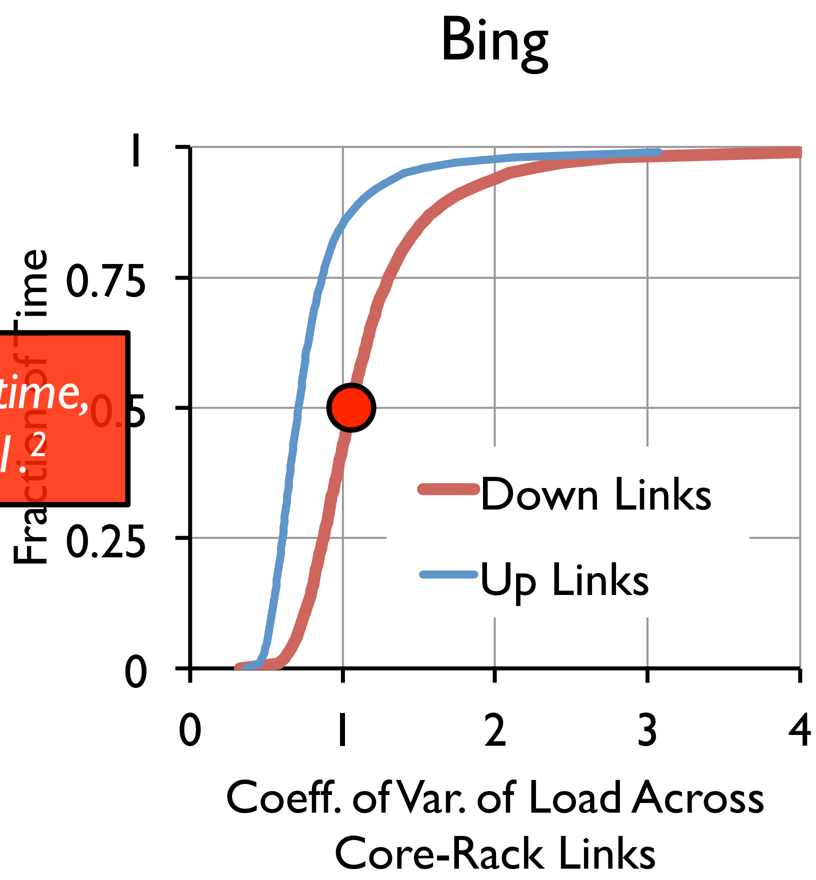




Network is Imbalanced[1]



More than 50% of the time, downlinks have $C_v > 1$.[2]



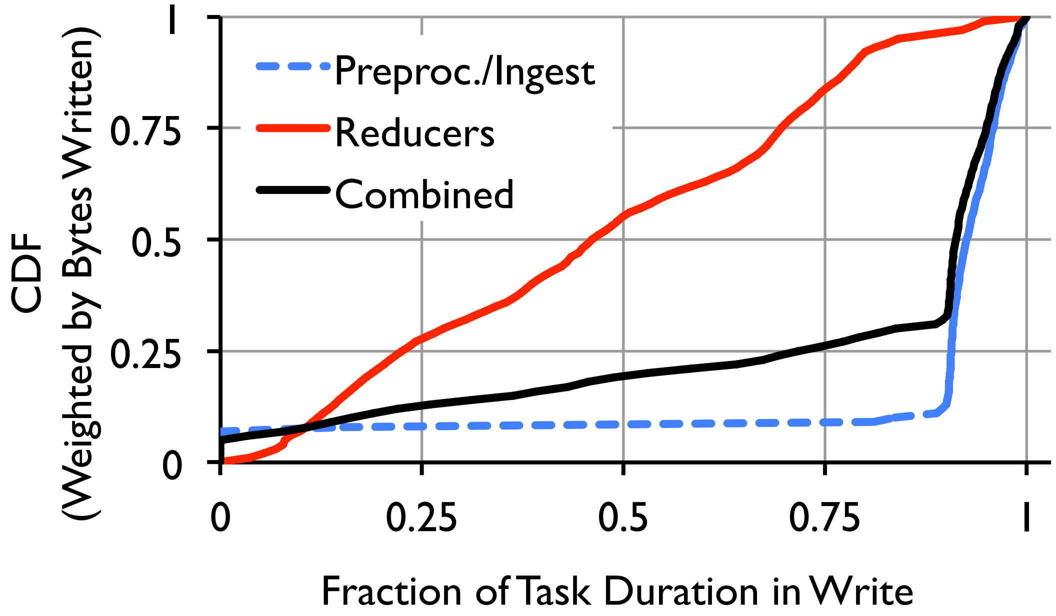
1. Imbalance considering all cross-rack bytes. Calculated in 10s bins.
2. Coefficient of variation, C_v = (stdev/mean).

Writer Characteristics

37% of all tasks write to the DFS

Two types of writers

1. Reducers
2. Ingestion/preprocessing tasks



Th1

Greedy assignment of blocks to the least-loaded-link-first order is optimal for minimizing the average block write time

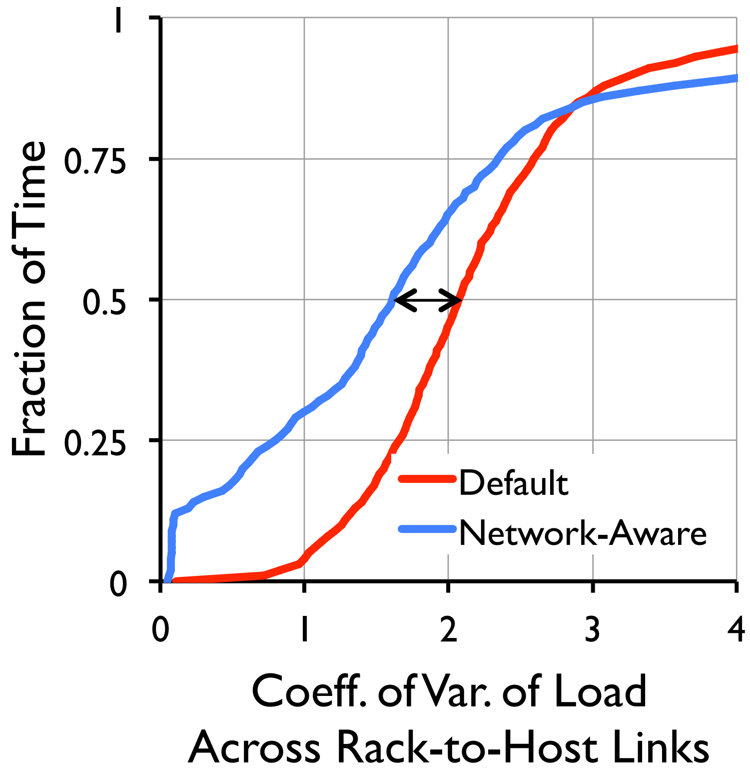
Th2

Greedy assignment of blocks to the least-loaded link in the least-remaining-blocks-first order is optimal for minimizing the average file write time

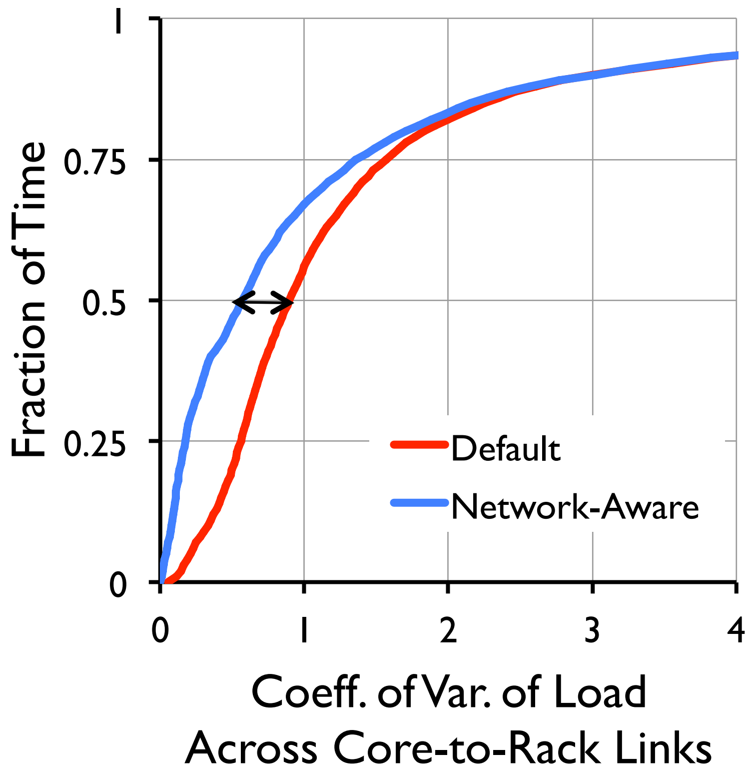
Balanced Network

Decrease in median Cv for exp(sim) is 0.46(0.33)

EC2 Deployment

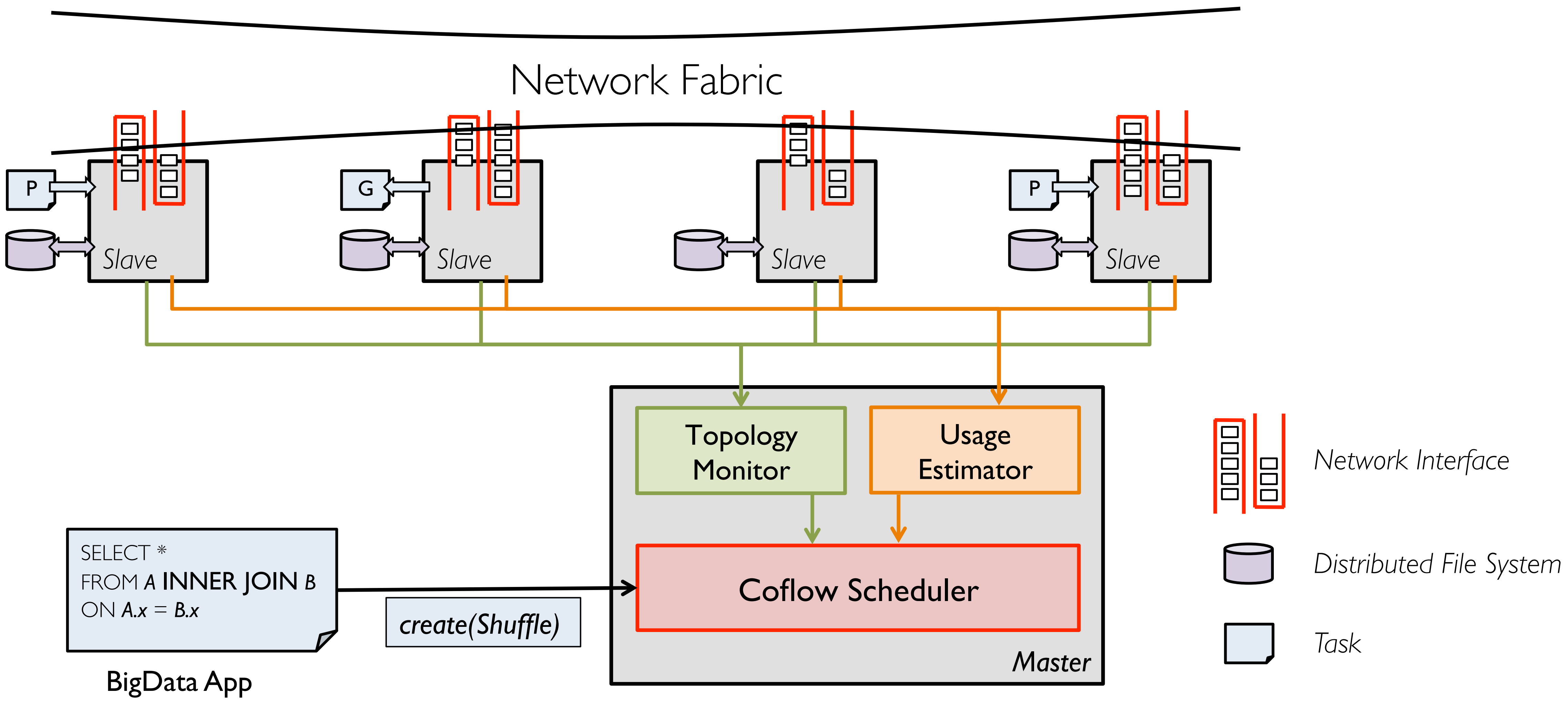


Facebook Trace Simulation

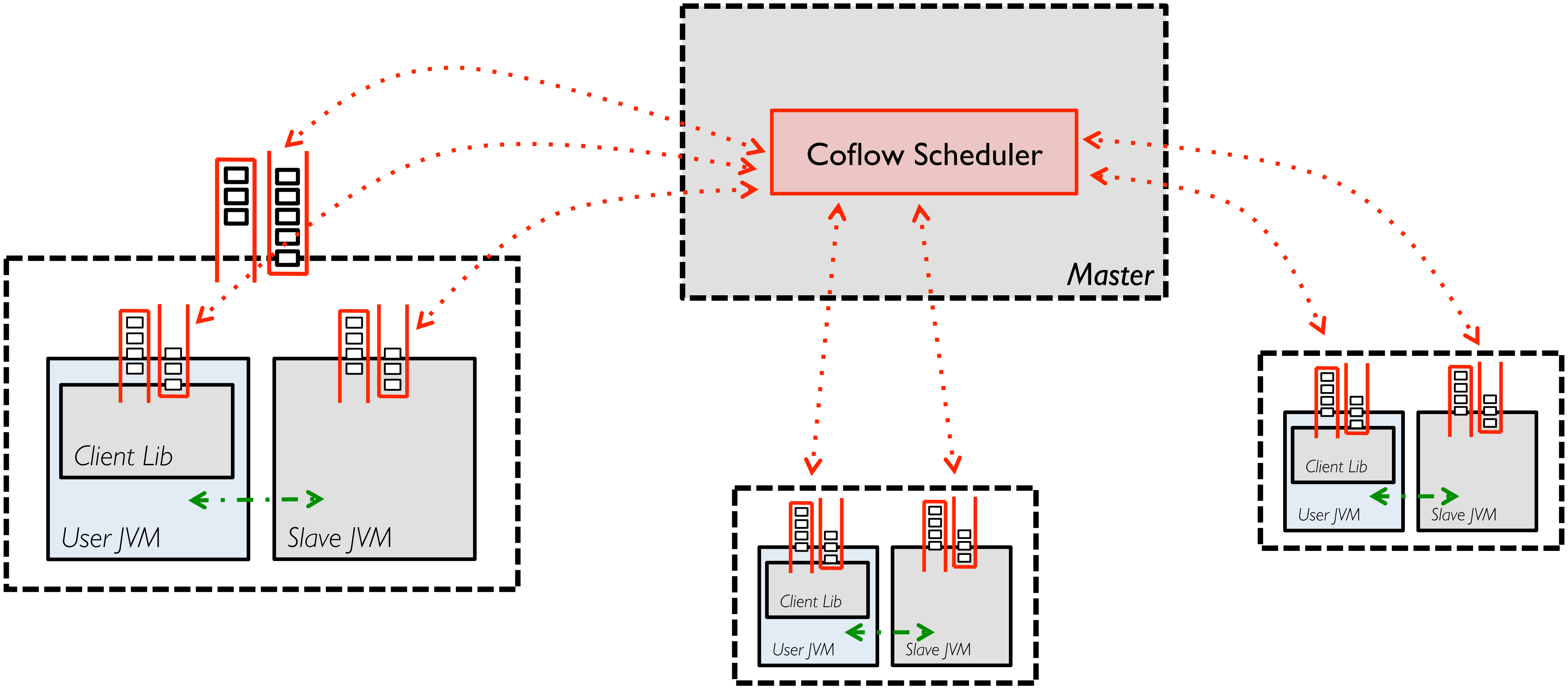


System Architecture

Actual *timing* and *order* of communication is controlled by the Coflow Scheduler



Details



Coflow Scheduler
Master
Client Lib
User JVM
Slave JVM
Client Lib
User JVM
Slave JVM
Client Lib
User JVM
Slave JVM

Current Implementation

Implemented in ~2700 lines of Scala

- Core + Framework: ~1800 lines
- Client library: ~400 lines
- Web UI: ~300 lines
- Utils: ~200 lines
- ***Scheduler does not exist yet***

Can **put** and **get**

- On-disk files,
- In-memory objects, and
- Fake data (for testing)

Sufficient to implement Orchestra

- Cornet already implemented

Includes OFS/Usher/Sinbad functionalities

- Exposes **getBest(Rx|Tx)Machines** method

Cornet[1] *Implementation* [Master]

1. Managing Data Transfers in Computer Clusters with Orchestra, SIGCOMM'11.

Cornet[1] Implementation [Slaves]

```
// Create new client
val client = new Client("BroadcastReceiver", masterUrl)
client.start()
```

1. Managing Data Transfers in Computer Clusters with Orchestra, SIGCOMM'11.

Theorems

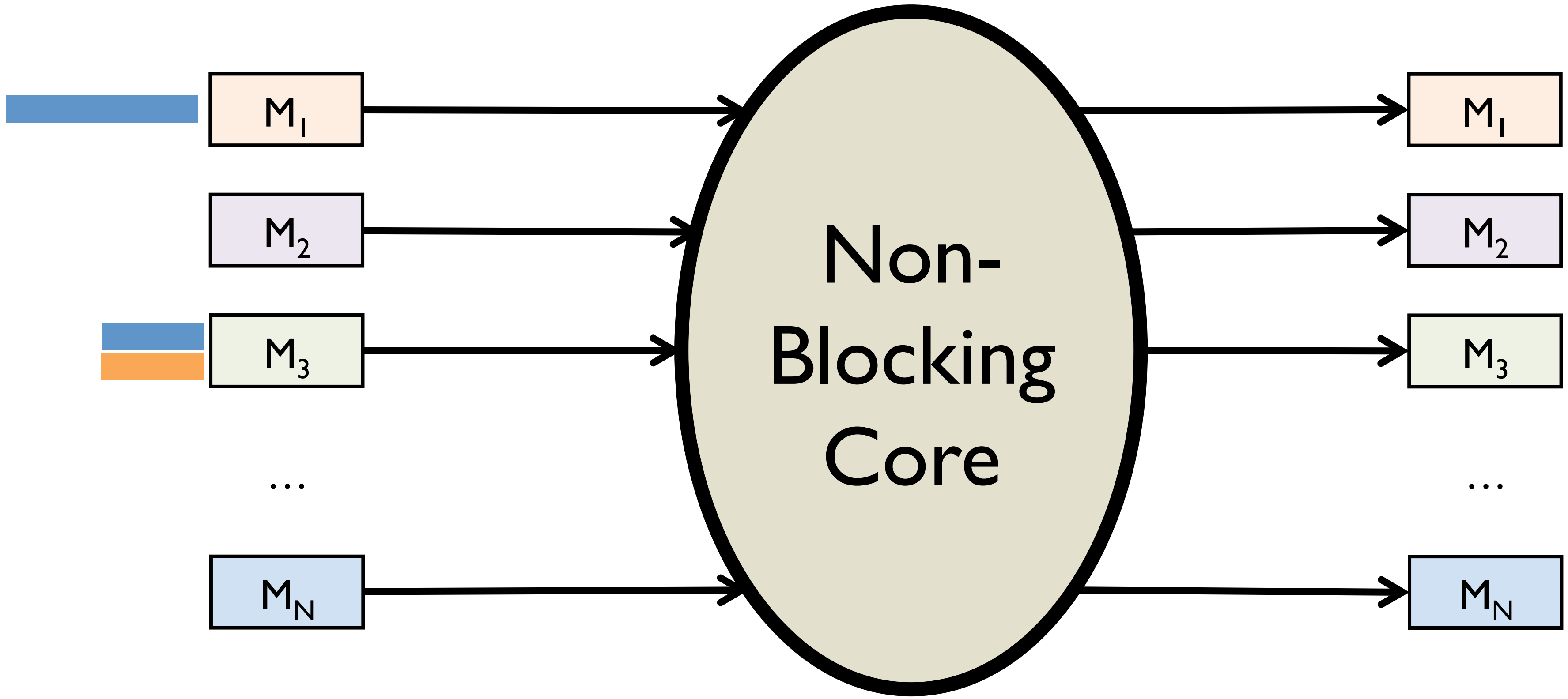
Upper Bound:

There exists an algorithm that result in completion time within 2X of the optimal

Lower Bound:

Unless P=NP, we can find completion time within, at best, 1.5X of the optimal

Two-Sided Problem [Bipartite Matching]



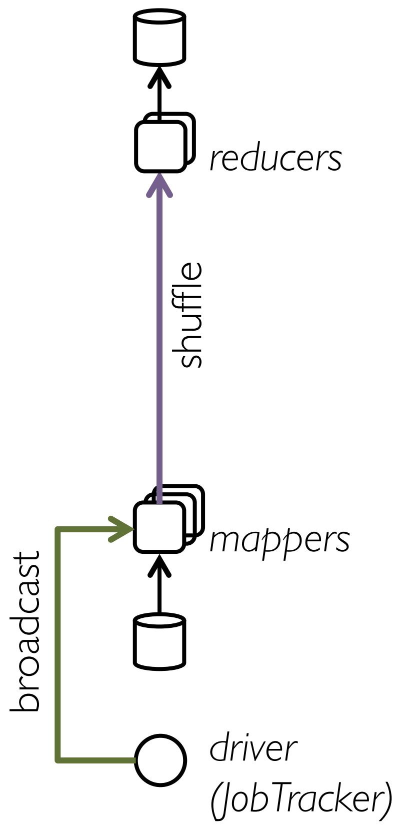
In what order?
To where?

In Progress.
Results from ordering might be useful.

Declarative API

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

- create
- put
- get
- terminate



@driver

b ← **create**(*BCAST*)
s ← **create**(*SHUFFLE*)

id ← *b*.**put**(*content*)
…

b.**terminate**( )
s.**terminate**( )

@mapper

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

@reducer

s.**get**(id_{sl})
…

System Architecture

Centralized design

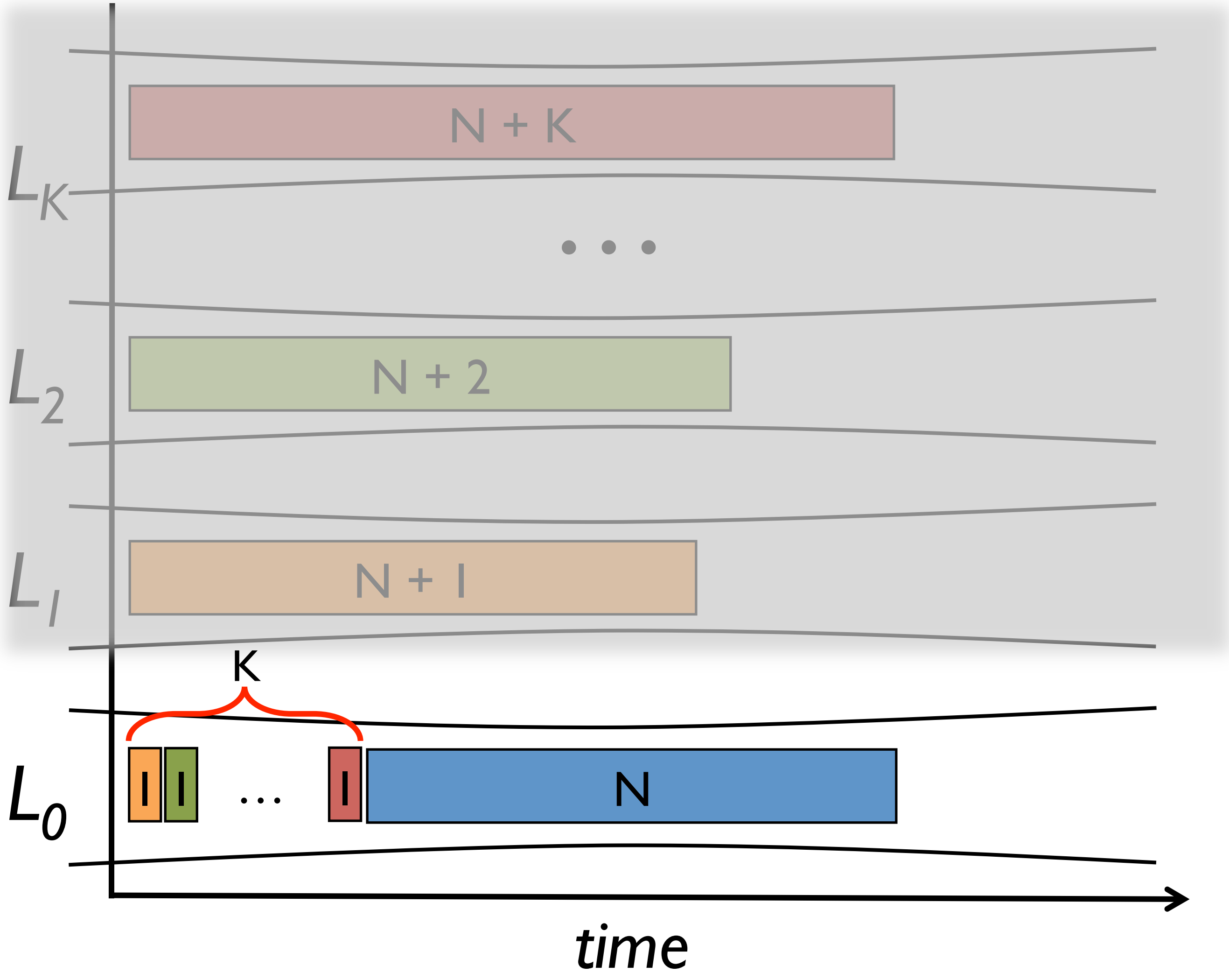
- Common architectural pattern in cluster computing
- Fall back to normal communication upon failure

Application layer overlay

Hypervisor-based

SDN-based

How Much Better Can We Do?

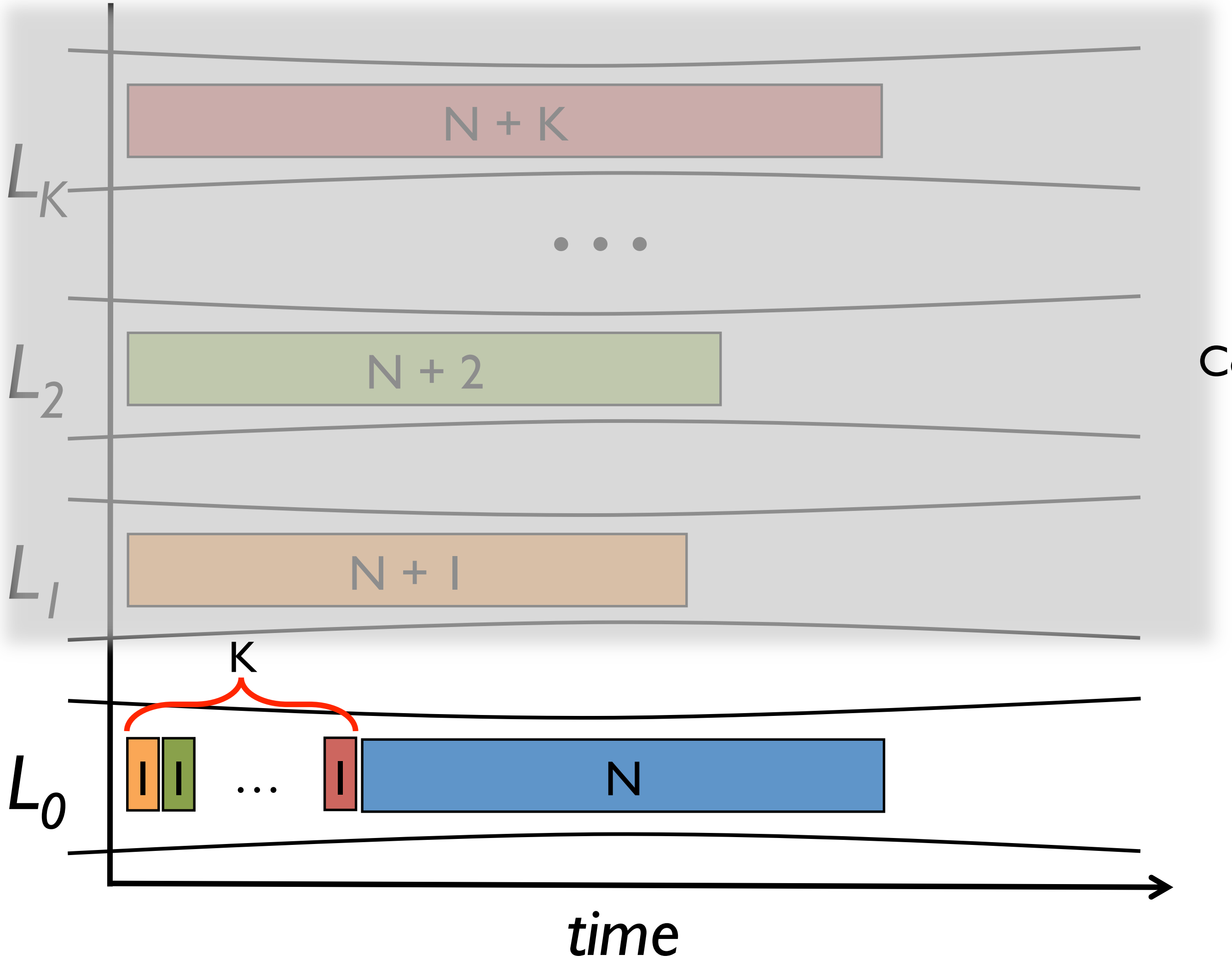


Completion time of the blue coflow considering only L_0

$$= \frac{K(K+1)}{2} + (N+K)$$

$$= \frac{K(K+3)}{2} + N$$

How Much Better Can We Do?



Completion time of the blue coflow considering only L_0 $= \frac{K(K+1)}{2} + (N+K)$

$$= \frac{K(K+3)}{2} + N$$

Completion time considering all links $= N$

Improvement $= \frac{K(K+3)}{2N} + 1$

No change for other coflows

Max Improvement

K << N	1x
K == N	Kx
K >> N	K^2x