6.888
Lecture 8:
Networking for Data Analytics

Mohammad Alizadeh

✧ Many thanks to Mosharaf Chowdhury (Michigan) and Kay Ousterhout (Berkeley)

Spring 2016

“Big Data”

Huge amounts of data being collected daily

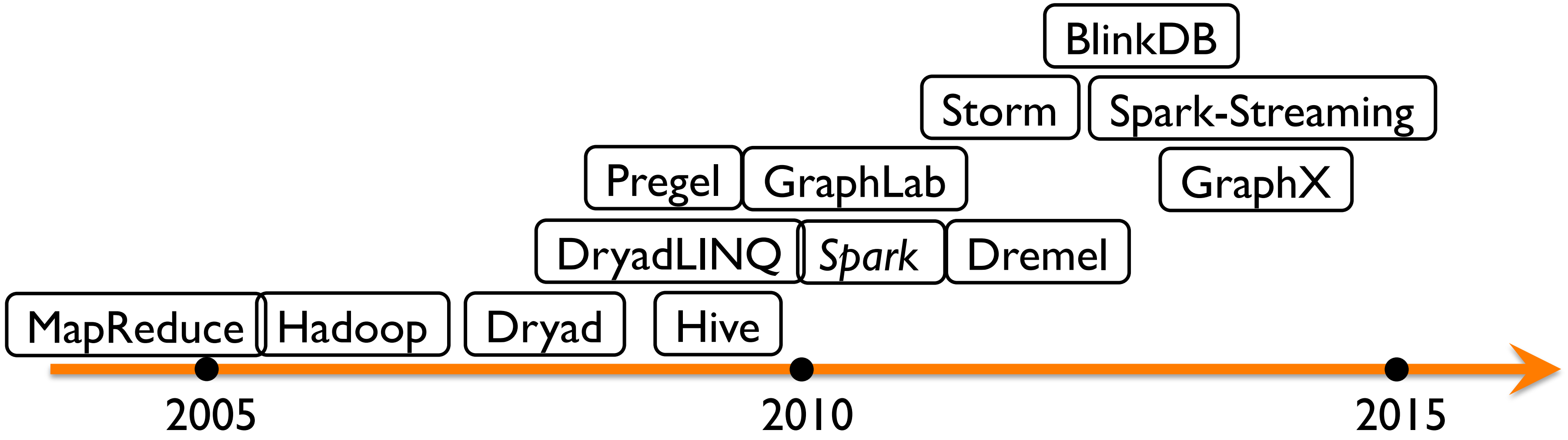
Wide variety of sources

- Web, mobile, wearables, IoT, scientific
- Machines: monitoring, logs, etc

Many applications

- Business intelligence, scientific research, health care

Big Data Systems



Data Parallel Applications

Multi-stage dataflow

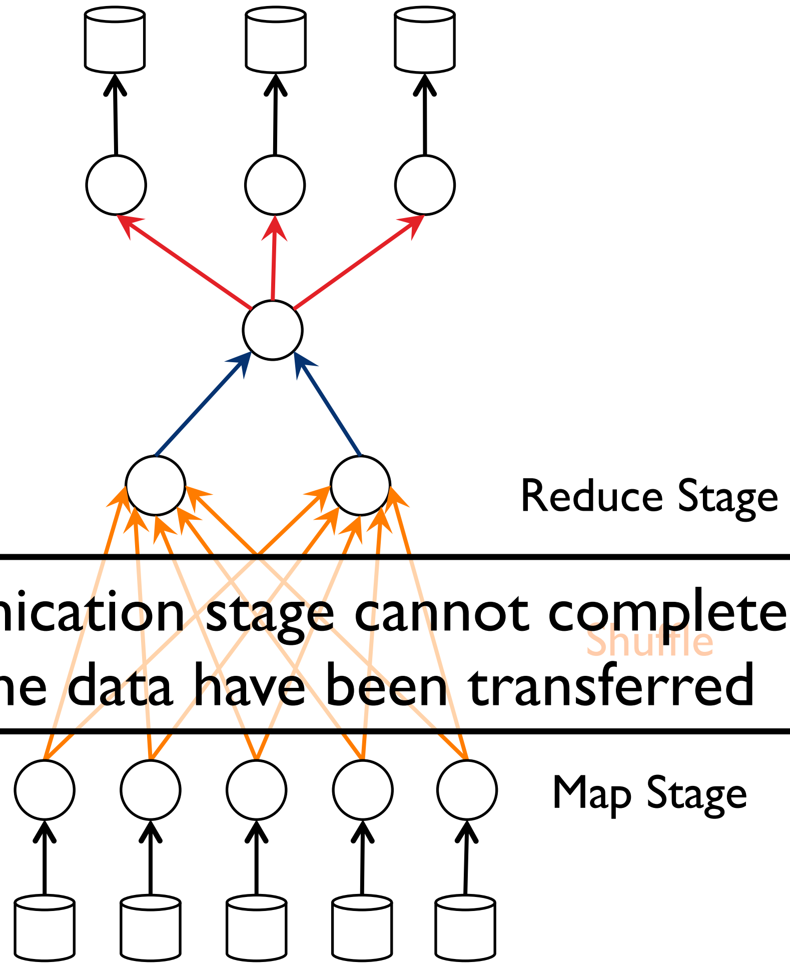
- Computation interleaved with communication

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

- Distributed across many machines
- Tasks run in parallel

Communication Stage (e.g., Shuffle)

- Between successive computation stages



A communication stage cannot complete until *all* the data have been transferred

Questions

How to design the network for data parallel applications?

- What are good communication abstractions?

Does the network matter for data parallel applications?

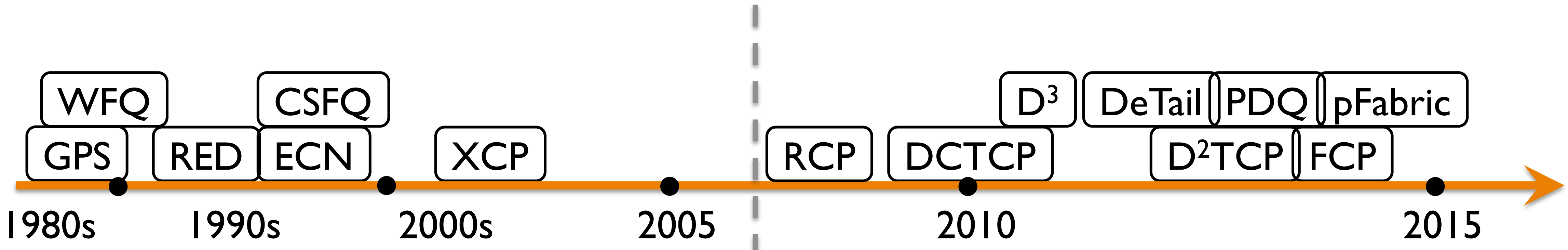
- What are the bottlenecks for these applications?

Efficient Coflow Scheduling with Varys

✧ Slides by Mosharaf Chowdhury (Michigan), with minor modifications

Existing Solutions

Flow: Transfer of data from a source to a destination



Per-Flow Fairness

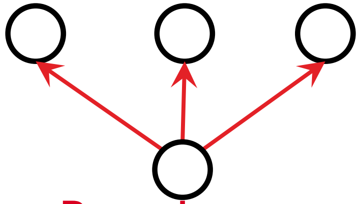
Flow Completion Time

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

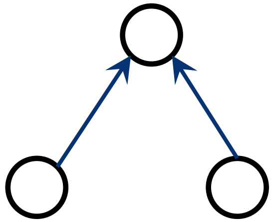
Coflow

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

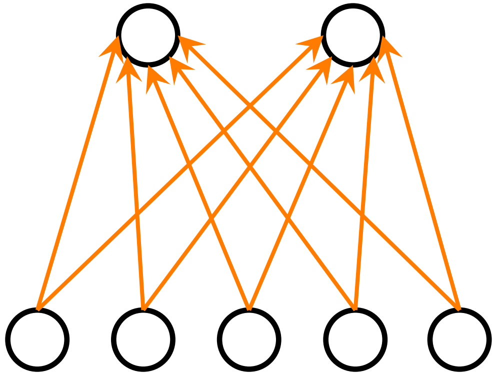
1. Minimize completion times,
2. Meet deadlines



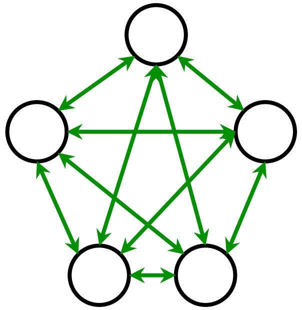
Broadcast



Aggregation

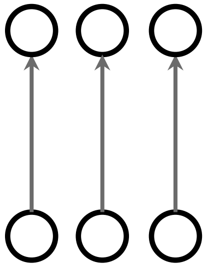


Shuffle



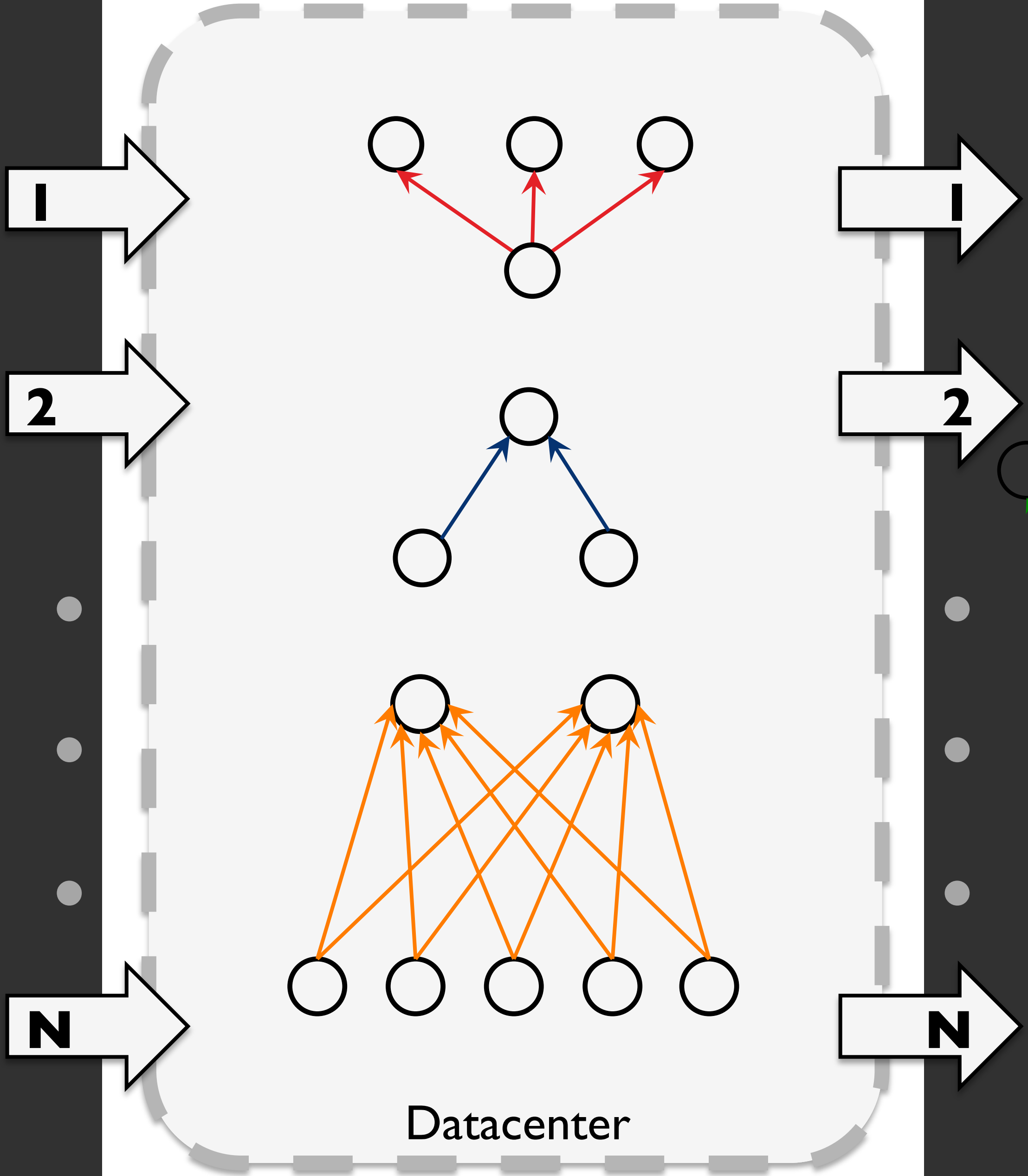
All-to-All

Single Flow

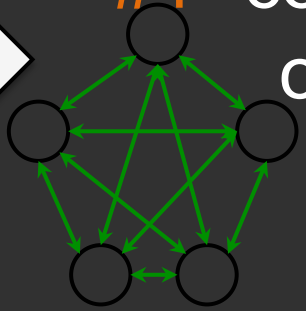


Parallel Flows

How to schedule coflows online …

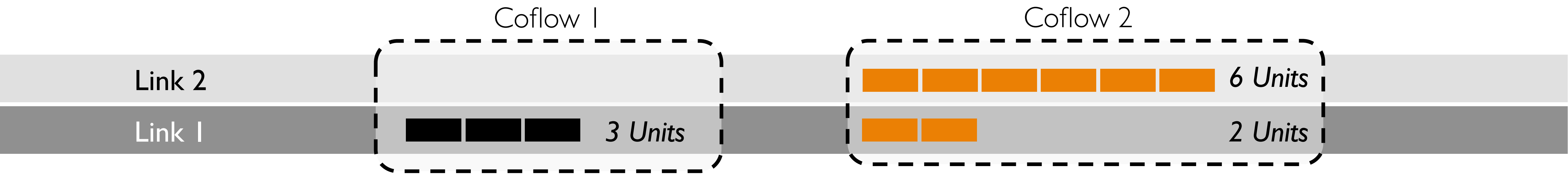


#1 … for faster completion of coflows?

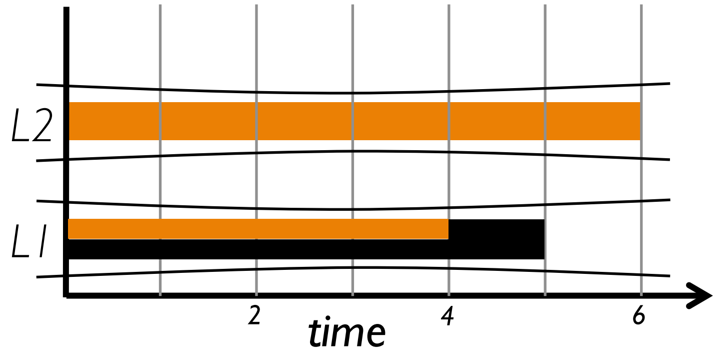


#2 … to meet more deadlines?

Benefits of Inter-Coflow Scheduling

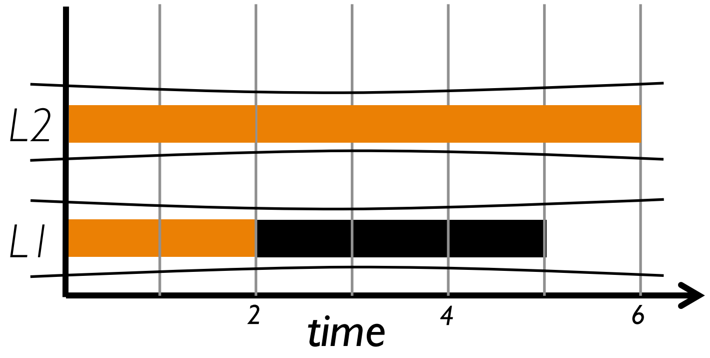


Fair Sharing



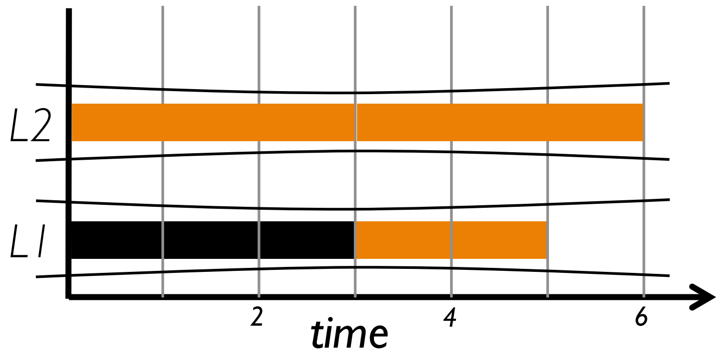
Coflow1 comp. time = 5
Coflow2 comp. time = 6

Smallest-Flow First[1,2]



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

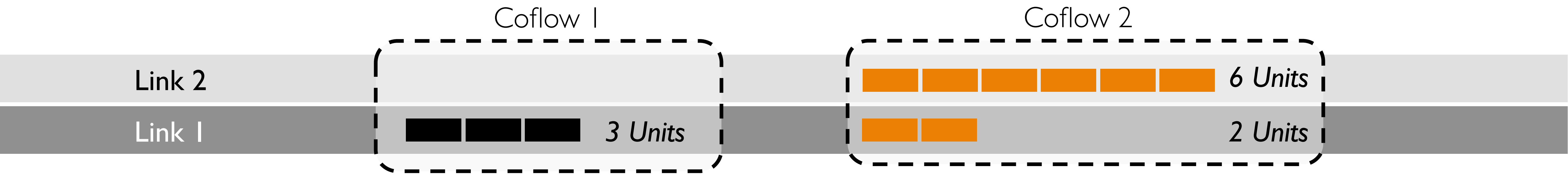
The Optimal



*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.

Benefits of Inter-Coflow Scheduling is NP-Hard

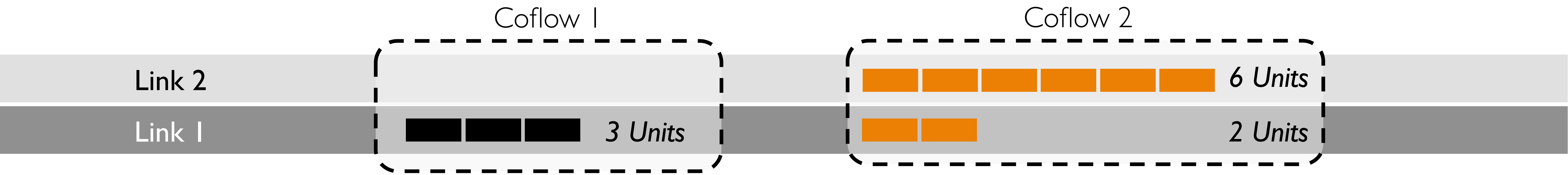


Concurrent Open Shop Scheduling[1]

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

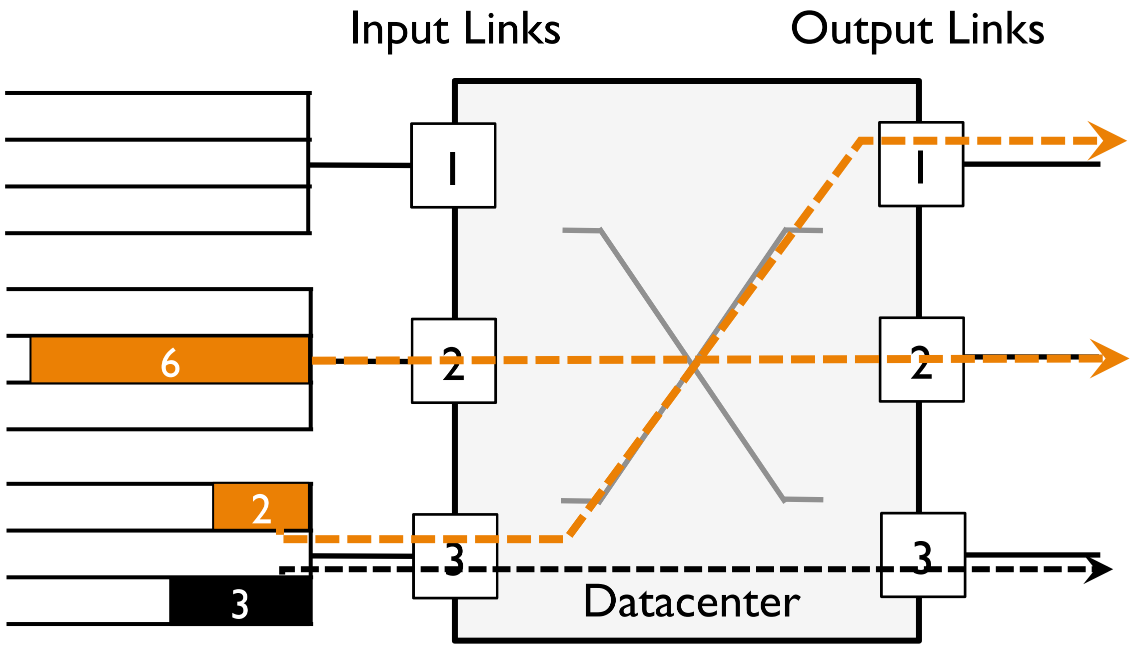
1. Finishing Flows Quickly with Preemptive Scheduling, SIGCOMM'2012.
2. pFabric: Minimal Near-Optimal Datacenter Transport, SIGCOMM'2013.

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*

Allocation Algorithm

A coflow cannot finish before its very last flow

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

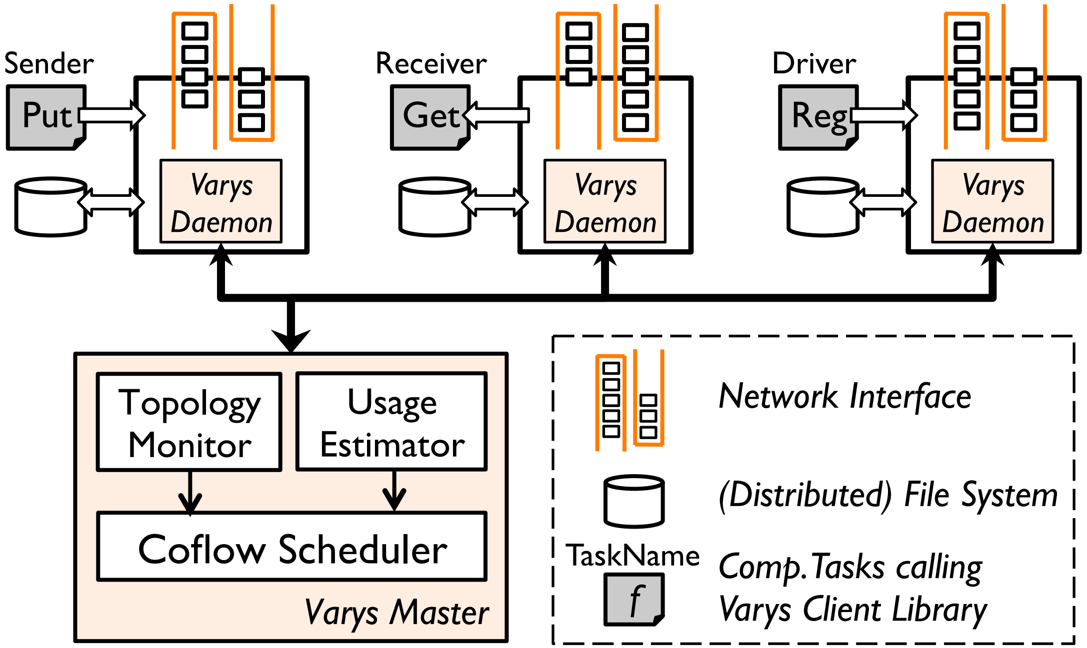
Allocate minimum flow rates such that all flows of a coflow finish together on time

Varys 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



Discussion

Making Sense of Performance in Data Analytics Frameworks

✧ Slides by Kay Ousterhout (Berkeley), with minor modifications

Network

Load balancing: VL2 [SIGCOMM ‘09], Hedera [NSDI ’10], Sinbad [SIGCOMM ’13]
Application semantics: Orchestra [SIGCOMM ’11], Baraat [SIGCOMM ‘14], Varys [SIGCOMM ’14]
Reduce data sent: PeriSCOPE [OSDI ‘12], SUDO [NSDI ’12]
In-network aggregation: Camdoop [NSDI ’12]
Better isolation and fairness: Oktopus [SIGCOMM ’11], EyeQ [NSDI ‘12], FairCloud [SIGCOMM ’12]

Disk

Themis [SoCC ‘12], PACMan [NSDI ’12], Spark [NSDI ’12], Tachyon [SoCC ’14]

Stragglers

Scarlett [EuroSys ‘11], SkewTune [SIGMOD ‘12], LATE [OSDI ‘08], Mantri [OSDI ‘10], Dolly [NSDI ‘13], GRASS [NSDI ‘14], Wrangler [SoCC ’14]

Network

Load balancing: VL2 [SIGCOMM '09], Hedera [NSDI '10], Sinbad [SIGCOMM '13]

Application semantics: Orchestra [SIGCOMM '11], Baraat [SIGCOMM '14], Varys [SIGCOMM '14]

Reduce data sent: PeriSCOPE [OSDI '12], SUDO [NSDI '12]

In-network aggregation: Camdoop [NSDI '12]

Better isolation and fairness: Oktopus [SIGCOMM '11]), EyeQ [NSDI '12], FairCloud [SIGCOMM '12]

Missing: what's most important to end-to-end performance?

Disk

Themis [SoCC '12], PACMan [NSDI '12], Spark [NSDI '12], Tachyon [SoCC '14]

Stragglers

Scarlett [EuroSys '11], SkewTune [SIGMOD '12], LATE [OSDI '08], Mantri [OSDI '10], Dolly [NSDI '13], GRASS [NSDI '14], Wrangler [SoCC '14]

Network

Load balancing: VL2 [SIGCOMM ‘09], Hedera [NSDI ’10], Sinbad [SIGCOMM ’13]
Application semantics: Orchestra [SIGCOMM ’11], Baraat [SIGCOMM ‘14], Varys [SIGCOMM ’14]
Reduce data sent: PeriSCOPE [OSDI ‘12], SUDO [NSDI ’12]
In-network aggregation: Camdoop [NSDI ’12]
Better isolation and fairness: Oktopus [SIGCOMM ‘11]), EyeQ [NSDI ‘12], FairCloud [SIGCOMM ’12]

Disk

Themis [SoCC ‘12], PACMan [NSDI ’12], Spark [NSDI ’12], Tachyon [SoCC ’14]

Stragglers

Scarlett [EuroSys ‘11], SkewTune [SIGMOD ‘12], LATE [OSDI ‘08], Mantri [OSDI ‘10], Dolly [NSDI ‘13], GRASS [NSDI ‘14], Wrangler [SoCC ’14]

Widely-accepted mantras:

Network and disk I/O are bottlenecks

Stragglers are a major issue with unknown causes

This work

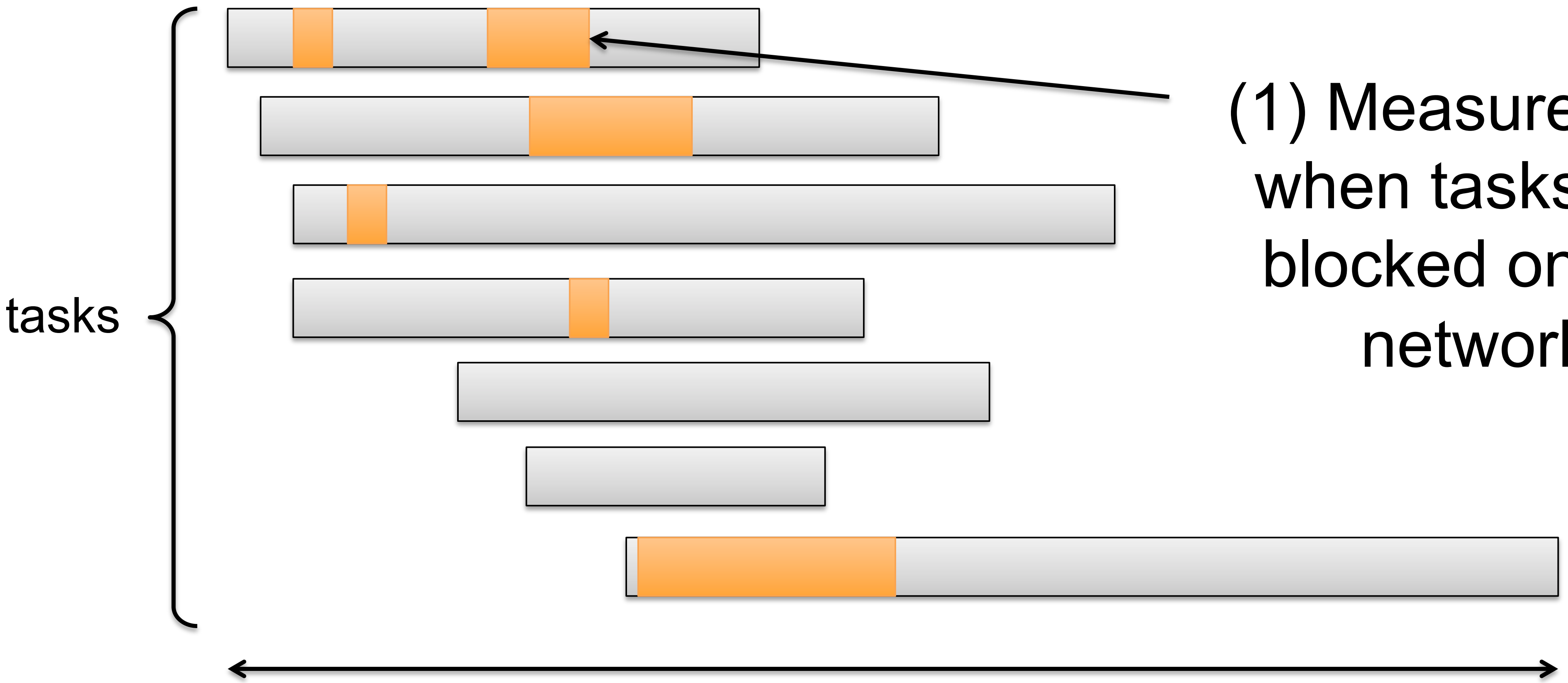
(1) How can we quantify performance bottlenecks?

Blocked time analysis

(2) Do the mantras hold?

Takeaways based on three workloads run with Spark

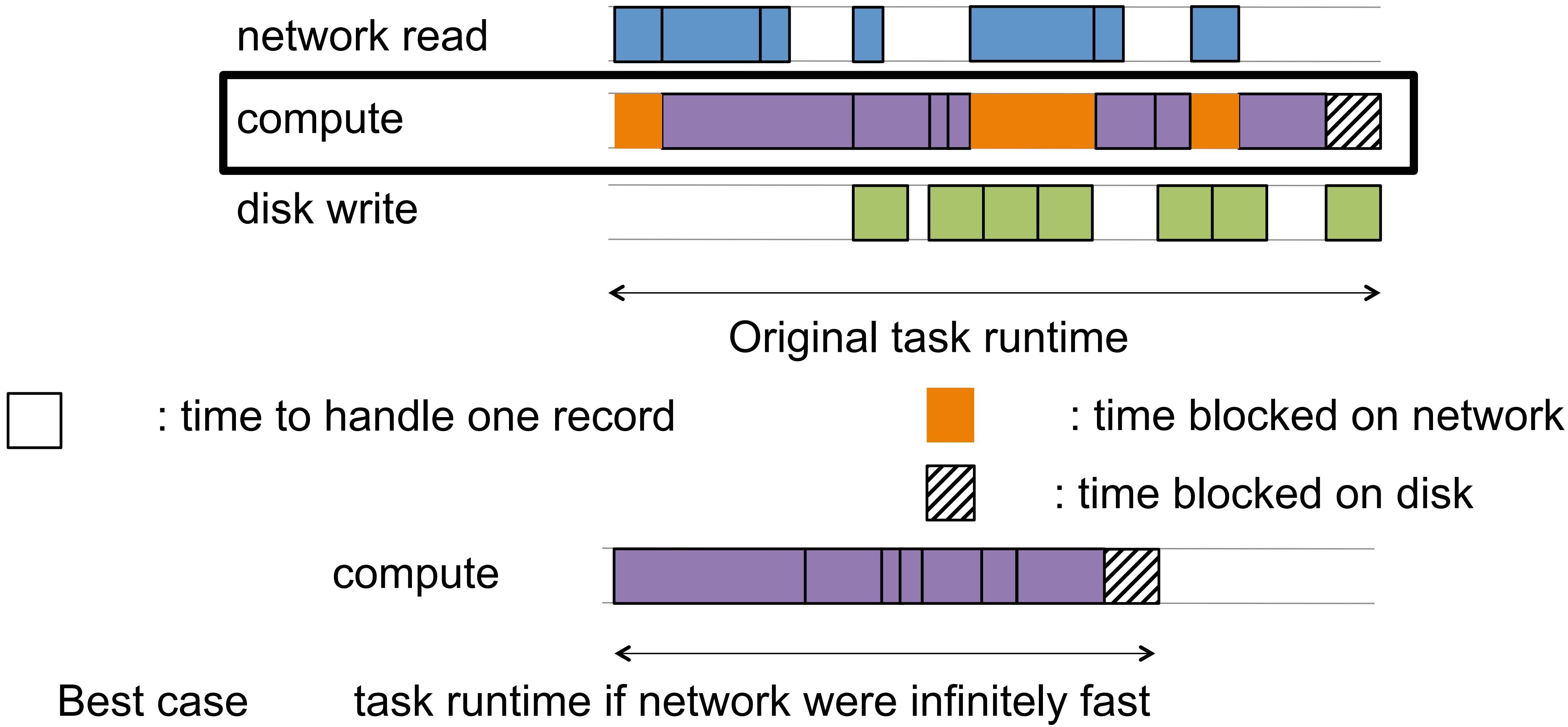
Blocked time analysis



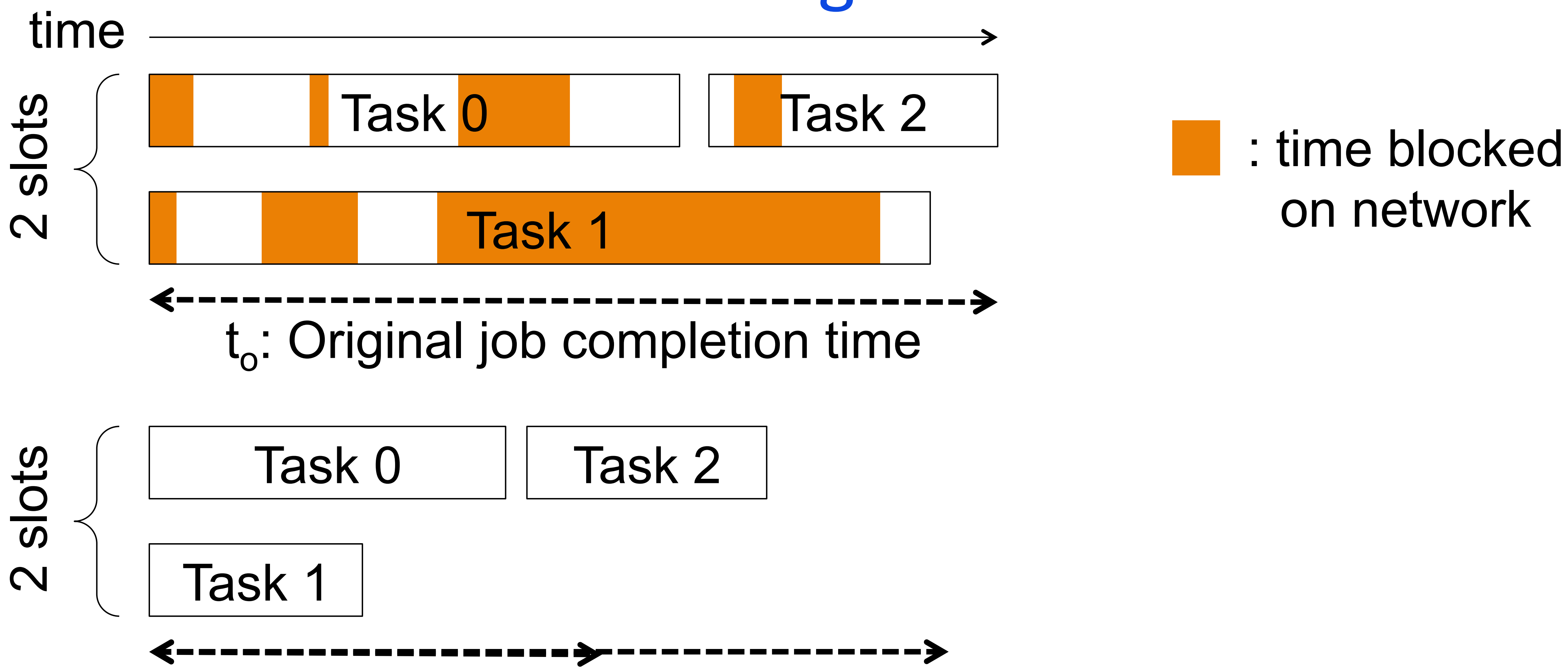
(1) Measure time when tasks are blocked on the network

(2) Simulate how job completion time would change

(1) Measure the time when tasks are blocked on the network



(2) Simulate how job completion time would change



t_n: Job completion time with infinitely fast network

Incorrectly computed time: doesn't account for task scheduling

Takeaways based on three Spark workloads:

Network optimizations

can reduce job completion time by at most 2%

CPU (not I/O) often the bottleneck

<19% reduction in completion time from optimizing disk

Many straggler causes can be identified and fixed

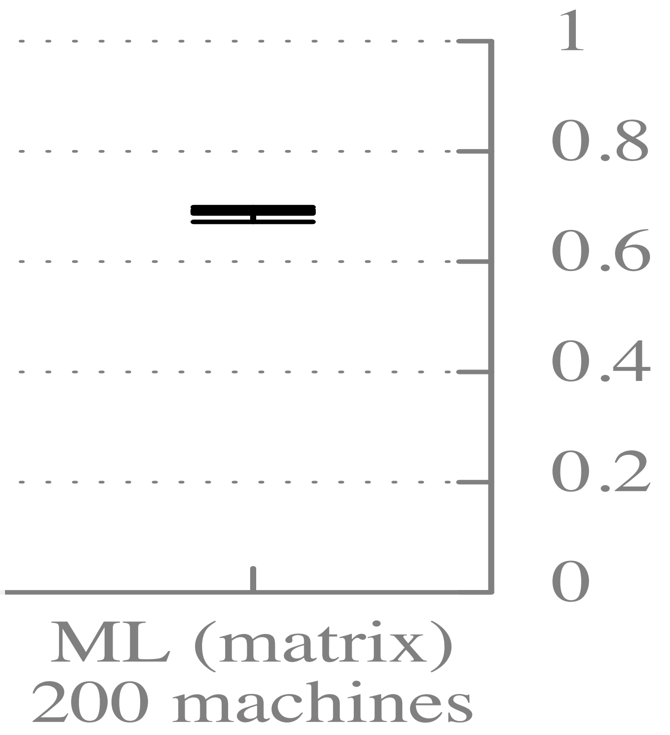
When does the network matter?

Network important when:

(1) Computation optimized

(2) Serialization time low

(3) Large amount of data sent over network



Discussion

What You Said

“I very much appreciated the thorough nature of the "Making Sense of Performance in Data Analytics Frameworks" paper.”

“I see their paper as more of a survey on the performance of current data analytics platforms as opposed to a paper that discusses fundamental tradeoffs between compute and networking resources. I think the question of whether current “data-analytics platforms” are network bound or CPU bound depends heavily on the implementation, and design assumptions. As a result, I see their work as somewhat of a self-fulfilling prophecy.”

What You Said

"The paper admits its bias in primarily studying instrumented Spark servers. It uses traces from real-world services to back up its conclusions across other types and scales of services, and is reasonably convincing in this analysis. It is easy to agree with the conclusion that services should be more heavily instrumented."

Next Time:
Wireless/Optical Data Centers

FireFly: A Reconfigurable Wireless Data Center Fabric Using Free-Space Optics

Navid Hamedazimi,[†] Zafar Qazi,[†] Himanshu Gupta,[†] Vyas Sekar,[*] Samir R. Das,[†] Jon P. Longtin, Himanshu Shah,[†] and Ashish Tanwer[†]
[†]Stony Brook University [*]Carnegie Mellon University

ABSTRACT

Conventional *static* datacenter (DC) network designs offer extreme cost vs. performance tradeoffs—simple leaf-spine networks are cost-effective but oversubscribed, while "fat tree"-like solutions offer good worst-case performance but are expensive. Recent results make a promising case for *augmenting* an oversubscribed network with reconfigurable inter-rack wireless or optical links. Inspired by the promise of reconfigurability, this paper presents *FireFly*, an inter-rack network solution that pushes DC network design to the extreme on three key fronts: (1) all links are reconfigurable; (2) all links are wireless; and (3) non top-of-rack switches are eliminated altogether. This vision, if realized, can offer significant benefits in terms of increased flexibility, reduced equipment cost, and minimal cabling complexity. In order to achieve this vision, we need to look beyond traditional RF wireless solutions due to their interference footprint which limits range and data rates. Thus, we make the case for using free-space optics (FSO). We demonstrate the viability of this architecture by (a) building a proof-of-concept prototype of a

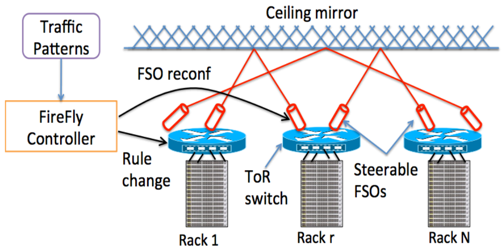


Figure 1: High-level view of the FireFly architecture. The only switches are the Top-of-Rack (ToR) switches.

Recent work suggests a promising middleground that augments an oversubscribed network with a few reconfigurable links, using either 60 Ghz RF wireless [26, 52] or optical switches [48]. Inspired by the promise of these flexible DC designs,[1] we envision a radically different DC architecture that pushes the network design to the *logical extreme* on three dimensions: (1) All inter-rack links

Integrating Microsecond Circuit Switching into the Data Center

George Porter Richard Strong Nathan Farrington Alex Forencich Pang Chen-Sun
Tajana Rosing Yeshaiahu Fainman George Papen Amin Vahdat[†]
UC San Diego UC San Diego and Google, Inc.[†]

ABSTRACT

Recent proposals have employed optical circuit switching (OCS) to reduce the cost of data center networks. However, the relatively slow switching times (10–100 ms) assumed by these approaches, and the accompanying latencies of their control planes, has limited its use to only the largest data center networks with highly aggregated and constrained workloads. As faster switch technologies become available, designing a control plane capable of supporting them becomes a key challenge.

In this paper, we design and implement an OCS prototype capable of switching in 11.5 μs, and we use this prototype to expose a set of challenges that arise when supporting switching at microsecond time scales. In response, we propose a microsecond-latency control plane based on a circuit scheduling approach we call Traffic Matrix Scheduling (TMS) that proactively communicates circuit assignments to communicating entities so that circuit bandwidth can be used efficiently.

supporting high bisection bandwidth is important, since ultimately application performance, and hence overall server utilization, may suffer if insufficient bandwidth is available. The result is that network complexity and expense are increasing.

To meet the required bandwidth demands, data center operators have adopted multi-layer network topologies [14] (e.g., folded Clos, or "FatTrees" [1, 16]), shown in Figure 1(a). While these topologies scale to very high port counts, they are also a significant source of cost, due in part to the large amount of switches, optical transceivers, fibers, and power each of their layers requires. Recent efforts have proposed [6, 8, 25] using optical circuit switches (OCS) to deliver reconfigurable bandwidth throughout the network, reducing some of the expense of multi-layer scale-out networks, shown in Figure 1(b). A key challenge to adopting these proposals has been their slow reconfiguration time, driven largely by existing 3D-MEMS technology limitations. Two components dominate this reconfiguration time: (1) the hardware switching time of the