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
Flow: Transfer of data from a source to a destination

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
Aggregation
Broadcast
Shuffle
Parallel Flows
All-to-All
Single Flow
Parallel Flows
How to schedule coflows online …

… for faster completion of coflows?

… to meet more deadlines?
Benefits of Inter-Coflow Scheduling

Coflow 1 comp. time = 5
Coflow2 comp. time = 6

Coflow 1 comp. time = 5
Coflow2 comp. time = 6

Coflow 1 comp. time = 3
Coflow2 comp. time = 6

Fair Sharing
Smallest-Flow First
The Optimal

Benefits of Inter-Coflow Scheduling

**Concurrent Open Shop Scheduling**

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

---

Coflow 1

- Link 1: 3 Units

Coflow 2

- Link 2: 6 Units
- Link 1: 2 Units

---

Concurrent Open Shop Scheduling with Coupled Resources

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

### Inter-Coflow Scheduling is NP-Hard

<table>
<thead>
<tr>
<th>Link 2</th>
<th>Link 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Coflow 1</td>
<td>3 Units</td>
</tr>
<tr>
<td>Coflow 2</td>
<td>6 Units</td>
</tr>
</tbody>
</table>

Input Links | Output Links
---|---
1 | 1
2 | 2
3 | 3

Datacenter
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.
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
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], EveQ [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 [SIGCOMMM ‘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]

Widely-accepted mantras:
Network and disk I/O are bottlenecks

Disk
Themis [SoCC ’12], PACSMAN [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]
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

- network read
- compute
- disk write

Original task runtime

: time to handle one record
: time blocked on network
: time blocked on disk

Best case task runtime if network were infinitely fast
(2) Simulate how job completion time would change

![Diagram showing task completion time with and without accounting for task scheduling and network delays.]

- \( t_o \): Original job completion time
- \( t_n \): Job completion time with infinitely fast network

- Task 0: 2 slots
- Task 1: 2 slots
- Task 2: 2 slots

: time blocked on 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, Vyasa 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 wireless inter-rack link, and (b) showing the feasibility of reconfiguring the link using a wireless controller.

Integrating Microsecond Circuit Switching into the Data Center
George Porter Richard Strong Nathan Farrington Alex Forencich Pang Chen-Sun Tajana Rosing Yeshaiah 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.