Coflow
Recent Advances and What’s Next?

Mosharaf Chowdhury

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

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

"Configuration should be handled at the system level"

---

**Existing Solutions**

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>GPS</td>
<td>RED</td>
<td>ECN</td>
<td>XCP</td>
<td>RCP</td>
<td>D²TCP</td>
</tr>
</tbody>
</table>

- GPS
- RED
- ECN
- XCP
- RCP
- D²TCP
- D³TCP
- FCP

**Why Do They Fall Short?**

Independent flows cannot capture the collective communication behavior common in data-parallel applications.
Why Do They Fall Short?

![Diagram showing network and datacenter connections]

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

![Diagram showing network and datacenter connections with flow proportional allocation]

Data-Proportional Allocation

Shuffle Completion Time = 4

Avg. Flow Completion Time = 4

Slow down faster flows to accelerate slower flows.

Coflow

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

1. Minimize completion times,
2. Meet deadlines, or
3. Perform fair allocation.
How to schedule coflows online ...

Varys

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

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.

1. Efficient Coflow Scheduling with Varys, SIGCOMM’14
Benefits of Inter-Coflow Scheduling

- **Concurrent Open Shop Scheduling with Coupled Resources**
  - Examples include job scheduling and caching blocks
  - Solutions use an *ordering* heuristic
  - Consider matching constraints

Inter-Coflow Scheduling is NP-Hard

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

2. **Allocation algorithm**
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
   Consistent calculation and enforcement of scheduler decisions
   Decouples network optimizations from applications, relieving developers and end users

2. Global Coordination

3. The Coflow API

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
2. Global Coordination
3. The Coflow API

The Coflow API

• register
• put
• get
• unregister

Evaluation

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

YES
Better than Per-Flow Fairness

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>EC2</td>
<td>3.16X</td>
<td>2.50X</td>
</tr>
<tr>
<td>Sim.</td>
<td>4.86X</td>
<td>3.39X</td>
</tr>
</tbody>
</table>

Preemption is Necessary [Sim.]

NO

Starvation

Better than Per-Flow Fairness

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>EC2</td>
<td>1.85X</td>
<td>1.25X</td>
</tr>
<tr>
<td>Sim.</td>
<td>3.21X</td>
<td>1.11X</td>
</tr>
</tbody>
</table>

Preemption is Necessary [Sim.]

NO

Starvation
Lack of Coordination Hurts [Sim.]

Smallest-flow-first (per-flow priorities)
- Minimizes flow completion time

FIFO-LM\textsuperscript{4} performs decentralized coflow scheduling
- Suffers due to local decisions
- Works well for small, similar coflows

<table>
<thead>
<tr>
<th>Varys</th>
<th>Per-Flow Fairness</th>
<th>FIFO</th>
<th>Per-Flow Prioritization</th>
<th>FIFO-LM\textsuperscript{4}</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.00</td>
<td>3.21</td>
<td>5.65</td>
<td>5.33</td>
<td>22.97</td>
</tr>
</tbody>
</table>

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

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

\rightarrow Pipelining between stages
\rightarrow Speculative executions
\rightarrow Task failures and restarts

Aalo\textsuperscript{1}

Efficiently schedules coflows without complete and future information

1. Current size is a good predictor of actual size
2. Set priority that decreases by how much a coflow has sent
3. Discretize priority levels to blend in FIFO within each level

1. Efficient Coflow Scheduling Without Prior Knowledge, SIGCOMM’2015

How to Perform Coflow Scheduling Without Complete Knowledge?

1. Current size is a good predictor of actual size
2. Set priority that decreases by how much a coflow has sent
3. Discretize priority levels to blend in FIFO within each level
How to Perform Coflow Scheduling Without Changing the Applications?

1. Learn coflows online from traffic patterns
2. Error-tolerant scheduling to survive learning errors
3. Limited to jobs with single coflows

CODA

CODA: Toward Automatically Identifying and Scheduling Coflows in the Dark, SIGCOMM’2016

What About Fair Coflow Scheduling?

1. Multi-resource fairness with high utilization
2. Fairness-utilization tradeoff results in prisoner’s dilemma

HUG

HUG: Multi-Resource Fairness for Correlated and Elastic Demands, NSDI’2016
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@umich.edu
http://www.mosharaf.com/

Improve Flow Completion Times

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

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 (64/3)

The need for coordination
- Fully decentralized schedulers can perform arbitrarily worse than the optimal

Varys

Employs a two-step algorithm to support coflow deadlines

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

Do not admit any coflows that cannot be completed within deadline without violating existing deadlines
Allocate minimum required resources to each coflow to finish them at their deadlines
More Predictable

Facebook Trace Simulation

EC2 Deployment

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