Coflow
Mending the Application-Network Gap in Big Data Analytics

Mosharaf Chowdhury
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

- MapReduce
- Hadoop
- Dryad
- Hive
- Pregel
- GraphLab
- DryadLINQ
- Spark
- Dremel
- GraphX
- Spark-Streaming
- BlinkDB
- Storm

Timeline:
- 2005
- 2010
- 2015
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

Faster Communication Stages: Networking Approach

“Configuration should be handled at the system level”
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

Solution 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.

---

Per-Flow Fair Sharing

- Link to $r_1$
  - Completion Time = 5
- Link to $r_2$
  - Completion Time = 3.66

Data-Proportional Allocation

- Link to $r_1$
  - Completion Time = 4
- Link to $r_2$
  - Completion Time = 4

Slow down faster flows to accelerate slower flows.
Applications know their performance goals, but they have no means to let the network know.

“Configuration should be handled by the end users”
Faster Communication Stages:

- Systems Approach
  
  “Configuration should be handled by the end users”

- Networking Approach
  
  “Configuration should be handled at the system level”
Holistic Approach
Applications and the Network Working Together
Communication abstraction for data-parallel applications to express their performance goals

1. Minimize completion times,
2. Meet deadlines, or
3. Perform fair allocation.
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?

... for fair allocation of the network?
Varys

Enables coflows in data-intensive clusters

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

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.
Benefits of Inter-Coflow Scheduling

Coflow 1

Coflow 2

Link 2

Link 1

Fair Sharing

Smallest-Flow First\textsuperscript{1,2}

The Optimal

Coflow 1 \ comp. time = 5
Coflow 2 \ comp. time = 6

Coflow 1 \ comp. time = 5
Coflow 2 \ comp. time = 6

Coflow 1 \ comp. time = 3
Coflow 2 \ comp. time = 6

Inter-Coflow Scheduling is NP-Hard

Concurrent Open Shop Scheduling

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

---

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.
Ordering Heuristic

**Shortest-First**

(Total CCT = 35)
Ordering Heuristic

Shortest-First \((35)\)

(Narrowest-First
(Total CCT = 41)

<table>
<thead>
<tr>
<th>Width</th>
<th>(C_1)</th>
<th>(C_2)</th>
<th>(C_3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>
Ordering Heuristic

<table>
<thead>
<tr>
<th>Size</th>
<th>C_1</th>
<th>C_2</th>
<th>C_3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>9</td>
<td>10</td>
<td>6</td>
</tr>
</tbody>
</table>

Shortest-First (35)

Narrowest-First (41)

Smallest-First (34)
Ordering Heuristic

Bottleneck

<table>
<thead>
<tr>
<th>C₁</th>
<th>C₂</th>
<th>C₃</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>10</td>
<td>6</td>
</tr>
</tbody>
</table>

Smallest-First (34)

Shortest-First (35)

Narrowest-First (41)

Smallest-Bottleneck (31)
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

2. Global Coordination
   Consistent calculation and enforcement of scheduler decisions

3. The Coflow API
   Decouples network optimizations from applications, relieving developers and end users
The Need for Coordination

Scheduling with Coordination
(Total CCT = 13)
The Need for Coordination

Scheduling with Coordination
(Total CCT = 13)

Scheduling without Coordination
(Total CCT = 19)

Uncoordinated local decisions *interleave* coflows, hurting performance
Centralized master-slave architecture

• Applications use a client library to communicate with the master

Actual *timing* and *rates are* determined by the coflow scheduler

1. Download from http://varys.net
Varys

Enables coflows in data-intensive clusters

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
The Coflow API

- register
- put
- get
- unregister

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

@driver

\[ b \leftarrow \text{register}(\text{BROADCAST}) \]
\[ s \leftarrow \text{register}(\text{SHUFFLE}) \]

\[ id \leftarrow b.\text{put}(\text{content}) \]
\[ \ldots \]
\[ b.\text{unregister}() \]
\[ s.\text{unregister}() \]

@mapper

\[ b.\text{get}(id) \]
\[ \ldots \]
\[ id_{s_{1}} \leftarrow s.\text{put}(\text{content}) \]
\[ \ldots \]

@reducer

\[ s.\text{get}(id_{s_{1}}) \]
\[ \ldots \]
Evaluation

A 3000-machine trace-driven simulation matched against a 100-machine EC2 deployment

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

What About Starvation

NO

I. Managing Data Transfers in Computer Clusters with Orchestra, SIGCOMM’2011
Lack of Coordination Hurts [Sim.]

1. Managing Data Transfers in Computer Clusters with Orchestra, SIGCOMM'2011
2. Finishing Flows Quickly with Preemptive Scheduling, SIGCOMM'2012
4. Decentralized Task-Aware Scheduling for Data Center Networks, SIGCOMM'2014

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

FIFO-LM$^4$ performs decentralized coflow scheduling
- Suffers due to local decisions
- Works well for small, similar coflows
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.

- Pipelining between stages
- Speculative executions
- Task failures and restarts
How to Perform Coflow Scheduling *Without* Complete Knowledge?
### Implications

<table>
<thead>
<tr>
<th>Minimize Avg. Comp. Time</th>
<th>Flows in a Single Link</th>
<th>Coflows in an Entire Datacenter</th>
</tr>
</thead>
<tbody>
<tr>
<td>With complete knowledge</td>
<td>Smallest-Flow-First</td>
<td>Ordering by Bottleneck Size + Data-Proportional Rate Allocation</td>
</tr>
<tr>
<td><strong>Without</strong> complete knowledge</td>
<td>Least-Attained Service (LAS)</td>
<td>?</td>
</tr>
</tbody>
</table>
Revisiting Ordering Heuristics

Shortest-First (35)

Narrowest-First (41)

Smallest-First (34)

Smallest-Bottleneck (31)
Coflow-Aware LAS (CLAS)

Set priority that decreases with how much a coflow has *already* sent
- The more a coflow has sent, the lower its priority
- Smaller coflows finish faster

Use *total size* of coflows to set priorities
- Avoids the drawbacks of full decentralization
Coflow-Aware LAS (CLAS)

Continuous priority reduces to fair sharing when similar coflows coexist

- Priority oscillation

FIFO works well for similar coflows

- Avoids the drawbacks of full decentralization
Discretized Coflow-Aware LAS (D-CLAS)

Priority discretization
  • Change priority when total size exceeds predefined thresholds

Scheduling policies
  • FIFO within the same queue
  • Prioritization across queue

Weighted sharing across queues
  • Guarantees starvation avoidance
How to Discretize Priorities?

Exponentially spaced thresholds
• $K$: number of queues
• $A$: threshold constant
• $E$: threshold exponent

Loose coordination suffices to calculate global coflow sizes
• Slaves make independent decisions in between

Small coflows (smaller than $E'A$) do not experience coordination overheads!
Closely Approximates Varys [Sim. & EC2]

1. Managing Data Transfers in Computer Clusters with Orchestra, SIGCOMM'2011
2. Finishing Flows Quickly with Preemptive Scheduling, SIGCOMM'2012
4. Decentralized Task-Aware Scheduling for Data Center Networks, SIGCOMM'2014
My Contributions

Coflow

Application-Aware Network Scheduling
My Contributions

**Orchestra**
SIGCOMM'11

**Varys**
SIGCOMM'14

**Aalo**
SIGCOMM'15

Application-Aware Network Scheduling
My Contributions

- **Spark**
  - NSDI'12

- **Sinbad**
  - SIGCOMM'13

- **Orchestra**
  - SIGCOMM'11

- **Varys**
  - SIGCOMM'14

- **Aalo**
  - SIGCOMM'15

- **FairCloud**
  - SIGCOMM'12

- **HARP**
  - SIGCOMM'12

- **ViNEYard**
  - ToN’12

**Network-Aware Applications**

**Application-Aware Network Scheduling**

**Datacenter Resource Allocation**
My Contributions

- **Spark**: NSDI’12
  - Top-Level Apache Project
- **Sinbad**: SIGCOMM’13
  - Merged at Facebook
- **Orchestra**: SIGCOMM’11
  - Merged with Spark
- **Varys**: SIGCOMM’14
  - Open-Source
- **Aalo**: SIGCOMM’15
  - Open-Source
- **FairCloud**: SIGCOMM’12
  - @HP
- **HARP**: SIGCOMM’12
  - @Microsoft Bing
- **ViNEYard**: ToN’12
  - Open-Source

- **Network-Aware Applications**
- **Application-Aware Network Scheduling**
- **Datacenter Resource Allocation**
Communication-First Big Data Systems

In-Datacenter Analytics
- Cheaper SSDs and DRAM, proliferation of optical networks, and resource disaggregation will make network the primary bottleneck

Inter-Datacenter Analytics
- Bandwidth-constrained wide-area networks

End User Delivery
- Faster and responsive delivery of analytics results over the Internet for better end user experience
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@cs.berkeley.edu
http://mosharaf.com
Improve Flow Completion Times

Per-Flow Fair Sharing

Shuffle Completion Time = 5
Avg. Flow Completion Time = 3.66

Smallest-Flow First¹,²

Link to r₁
Link to r₂

Shuffle Completion Time = 6
Avg. Flow Completion Time = 2.66

¹ Finishing Flows Quickly with Preemptive Scheduling, SIGCOMM’2012.
Distributions of Coflow Characteristics

- Coflow Length (Bytes)
- Coflow Width (Number of Flows)
- Coflow Size (Bytes)
- Coflow Bottleneck Size (Bytes)
**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

---

**Percentage of Traffic by Category at Facebook**

- **46%**
- **30%**
- **14%**
- **10%**
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 ($\frac{64}{3}$)*

The need for coordination

- *Fully decentralized schedulers can perform arbitrarily worse than the optimal*
The Coflow API

- register
- put
- get
- unregister

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

@driver
b ← register(BROADCAST, numFlows)
s ← register(SHUFFLE, numFlows, {b})

id ← b.put(content, size)
...
b.unregister()
s.unregister()

@mapper
b.get(id)
...

@reducer
s.get(id_s)
...

id_s ← s.put(content, size)
...

Varys

Employs a two-step algorithm to support coflow deadlines

1. Admission control
   Do not admit any coflows that cannot be completed within deadline without violating existing deadlines

2. Allocation algorithm
   Allocate minimum required resources to each coflow to finish them at their deadlines
More Predictable

Facebook Trace Simulation

1. Finishing Flows Quickly with Preemptive Scheduling, SIGCOMM'2012

EC2 Deployment

I. Finishing Flows Quickly with Preemptive Scheduling, SIGCOMM'2012
Optimizing Communication Performance: Systems Approach

“Let users figure it out”

<table>
<thead>
<tr>
<th></th>
<th># Comm. Params</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spark-v1.1.1</td>
<td>6</td>
</tr>
<tr>
<td>Hadoop-v1.2.1</td>
<td>10</td>
</tr>
<tr>
<td>YARN-v2.6.0</td>
<td>20</td>
</tr>
</tbody>
</table>

*Lower bound. Does not include many parameters that can indirectly impact communication; e.g., number of reducers etc. Also excludes control-plane communication/RPC parameters.
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