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\(^2\)

\(^2\) Large-scale parallel collaborative filtering for the Netflix prize, AAIM’08.
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?
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]

Micro-management is infeasible in large clusters
  • Scaling to $O(10^4)$ 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
Coflow

Represents a collection of one or more flows

- Performance-centric allocation
- Flexibility for cluster applications

- Coordination causes complexity
- Fixed endpoints are restrictive
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
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
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
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

Optimize at the level of coflows instead of individual flows
Many-to-Many/Shuffle

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

Status Quo

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

At a sender

At a receiver

In the network

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
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
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
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, \(C_v = \frac{\text{stddev}}{\text{mean}}\).
Usher Overview

Performs network-aware replica placement
Takes online decisions

Decreases network imbalance

Does it impact the storage balance? NO
### Observations

<table>
<thead>
<tr>
<th></th>
<th>Network hotspots are stable in the short term (5-10 sec)</th>
</tr>
</thead>
</table>

### Implications

|   | 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

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

Flow-level Prioritization

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

The Optimal

Coflow 1 comp. time = 3
Coflow 2 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

<table>
<thead>
<tr>
<th>Length</th>
<th>Width</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>2</td>
<td>9</td>
</tr>
</tbody>
</table>

### Flow Scheduling Policies

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

---

### Simulation Details

- 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

---

![Graph: Comparison of scheduling policies](image-url)
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
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

---

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 depend on per-iteration communication time
- Monarch\textsuperscript{1} spends up to 40% of the iteration time in shuffle

Communication often limits scalability
- Recommendation system for the Netflix challenge\textsuperscript{2}

\begin{itemize}
  \item Iterative algorithms depend on per-iteration communication time
  \item Monarch\textsuperscript{1} spends up to 40% of the iteration time in shuffle
\end{itemize}
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

<table>
<thead>
<tr>
<th>Google Scholar Query</th>
<th>Number of Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>network sharing +&quot;internet&quot;</td>
<td>1,420,000</td>
</tr>
<tr>
<td>network sharing +&quot;mobile&quot;</td>
<td>808,000</td>
</tr>
<tr>
<td>network sharing +&quot;wireless&quot;</td>
<td>407,000</td>
</tr>
<tr>
<td>network sharing +&quot;sensor&quot;</td>
<td>140,000</td>
</tr>
<tr>
<td>network sharing +&quot;local area&quot;</td>
<td>134,000</td>
</tr>
<tr>
<td>network sharing +&quot;wide area&quot;</td>
<td>93,400</td>
</tr>
<tr>
<td>network sharing +&quot;vehicular&quot;</td>
<td>36,000</td>
</tr>
<tr>
<td>network sharing +&quot;data center&quot;</td>
<td>26,000</td>
</tr>
</tbody>
</table>
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

*Completion time depends on the last flow to complete*
Cooperative Broadcast

Send the same data to all receivers

- Fast, scalable, and resilient

Peer-to-peer mechanism optimized for cooperative environments

<table>
<thead>
<tr>
<th>Observations</th>
<th>Design Decisions</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 High-bandwidth, low-latency network</td>
<td>✔ Large block size (4-16MB)</td>
</tr>
</tbody>
</table>
Performance

1GB data to 100 receivers on EC2

Status quo

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
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

Up to 2X faster than vanilla implementation
Collaborative Filtering using Alternating Least Squares

Performance degrades with increasing parallelism due to communication overhead

Without Orchestra

With Orchestra

\~2x faster at 90 nodes
Orchestra in Action: Netflix Challenge

Without Orchestra

With Orchestra

Performance degrades with increasing parallelism due to communication overhead

~2x faster at 90 nodes
Shuffle

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

Widespread use
• 68% of the Facebook jobs use shuffle

Status Quo

R₁ and R₂ are bottlenecks: 3 time units
S₃ 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

1.75X faster high priority coflows
1.06X slower low priority coflow
Sources of Network Traffic

Facebook
- 40%
- 14%
- 46%

Bing
- 54%
- 31%
- 15%

DFS Reads
DFS Writes
Coflow Comm.
Network is Imbalanced

- 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, $C_v = \frac{\text{stddev}}{\text{mean}}$. 

---

**Facebook**

- Coeff. of Var. of Load Across Core-Rack Links
- Fraction of Time
- Coeff. of Var. of Load Across
- Down Links
- Up Links

**Bing**

- Coeff. of Var. of Load Across Core-Rack Links
- Fraction of Time
- Coeff. of Var. of Load Across
- Down Links
- Up Links
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

EC2 Deployment

Facebook Trace Simulation

Decrease in median $C_v$ for $\exp(sim)$ is 0.46(0.33)
System Architecture

Network Fabric

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

SELECT * FROM A INNER JOIN B ON A.x = B.x

create(Shuffle)

Topology Monitor

Usage Estimator

Coflow Scheduler

Master

Network Interface

Distributed File System

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

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

// Create coflow
val desc = new CoflowDescription("Broadcast-" + fileName, CoflowType.BROADCAST, numSlaves)
val coflowId = client.registerCoflow(desc)

// Put blocks
for (fromBytes <- 0L to FILE_SIZE by DEFAULT_BLOCK_SIZE) {
  val blockSize = if (fromBytes + DEFAULT_BLOCK_SIZE >= FILE_SIZE) FILE_SIZE - fromBytes else DEFAULT_BLOCK_SIZE
  val blockName = fileName + "-" + fromBytes
  // Put block
  client.putFile(blockName, pathToFile, coflowId, fromBytes, blockSize, numSlaves)
}

// Wait for all slaves to finish
// Terminate coflow
client.unregisterCoflow(coflowId)
```
Cornet\textsuperscript{1} Implementation \textit{[Slaves]}

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

- create
- put
- get
- terminate

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

@driver
\[ b \leftarrow \text{create}(\text{BCAST}) \]
\[ s \leftarrow \text{create}(\text{SHUFFLE}) \]

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

@mapper
\[ b.\text{get}(id) \]
\[ \ldots \]
\[ s.\text{put}(id_{s1}) \]
\[ \ldots \]

@reducer
\[ s.\text{get}(id_{s1}) \]
\[ \ldots \]
System Architecture

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

- Application layer overlay
- Hypervisor-based
- SDN-based
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 $L_0$:

\[
\text{Improvement} = \frac{K(K+3)}{2} + N
\]

Completion time considering all links:

\[
= N
\]

Max Improvement:

<table>
<thead>
<tr>
<th>$K$ relative to $N$</th>
<th>Improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>$K \ll N$</td>
<td>$1x$</td>
</tr>
<tr>
<td>$K = N$</td>
<td>$Kx$</td>
</tr>
<tr>
<td>$K \gg N$</td>
<td>$K^2x$</td>
</tr>
</tbody>
</table>

No change for other coflows.

The diagram illustrates the completion times for different coflows with varying $K$ values relative to $N$. The improvement in completion time is calculated based on the number of links $K$ and the total number of coflows $N$. The table provides the max improvement based on the relative size of $K$ compared to $N$.