COFLOW CHAPTER 4 INTRA-COFLOW SCHEDULING

Author: Mosharaf Kabir Chowdhury

Presenter: Yuwei Jiao

UNIVERSITY OF WATERLOO

Outline

- Background
- Coflow
- Two Examples
  - Logistic Regression
  - Collaborative Filtering
- Broadcast Coflow
- Shuffle Coflow
- Experiment & Evaluation

Background

- Communication is crucial:
  - Facebook analytics jobs spend 25% of the running time in communication
- Network is likely to become the primary bottleneck

Background

- High cost of clusters → Maximize the cluster utilization
- Previous solutions focus on:
  - scheduling and managing computation and storage resources
  - ignoring the network

Background

- Overlook application-level requirements
- Existing approaches improving communication performance:
  - Increasing datacenter bandwidth
  - Decreasing flow completion time
- Lack of job-level semantics
- Hurt application-level performance

Background

- Optimizing communication performance
  - System approach: let users figure it out
  - Networking approach: let systems figure it out

	# Comm. Params*
Spark 1.0.1	6
Hadoop 1.0.4	10
YARN 2.3.0	20

Coflow

- Flow:
  - A sequence of packets between two endpoints
  - Independent unit of allocation, sharing, load balancing, prioritization
- Coflow:
  - A collection of flows that share a common performance goal
  - all-or-nothing property:
    - “ a communication stage cannot complete until all its flows have completed”

Coflow

- Two objectives:
  - Improve application-level performance by minimizing CCTs(completion time of a coflow)
  - Guarantee predictable completions within coflow deadlines
- NP-hard
  - Scheduler decide when to start and at what rate
- Focus on developing effective heuristics

Coflow

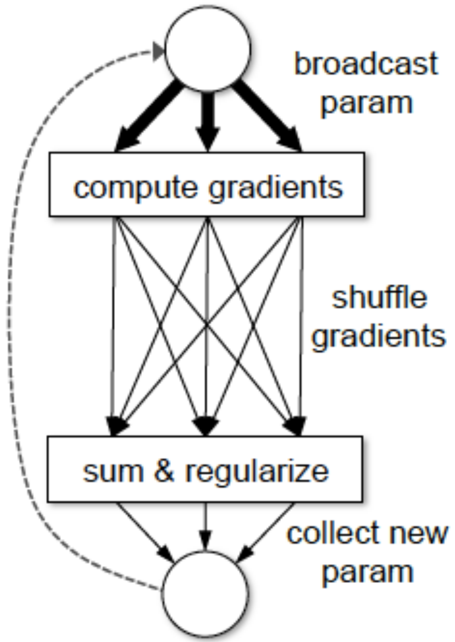
- Broadcast
  - One-to-many communication pattern
  - BitTorrent(Cornet)
- Shuffle
  - Many-to-many communication pattern
  - MADD(Minimum Allocation for Desired Duration)

Coflow

- Appropriate and attractive
  - Easy to implement into high-level frameworks
  - Faster deployment without modifying routers and switches
- Cornet:
  - 4.5x faster than default Hadoop
- MADD:
  - 29% speed up shuffles

Two Examples: Logistic Regression

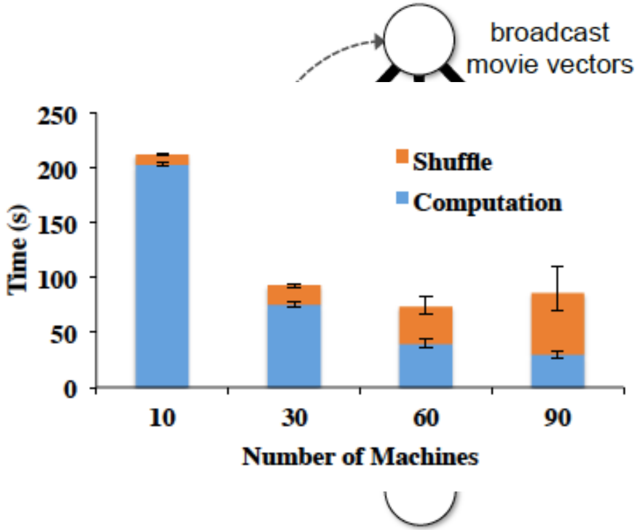
- Problem:
  - 55 GB of data collected about 345,000 tweets with links
  - 1000 – 2000 features
  - Identify which feature correlate with links to spam
- Workload
  - 100 iterations to converge
  - Broadcast(300MB) and shuffle(190MB per reducer) for each iteration
- Communication cost(30-machine)
  - 42% of the iteration time
  - 30% broadcast, 12% shuffle



(a) Logistic Regression

Two Examples: Collaborative Filtering

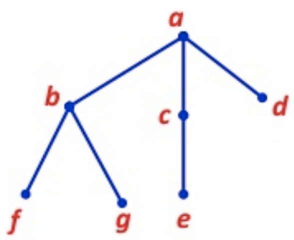
- Problem:
  - Predict users' ratings for movies
  - ALS(alternating least squares)
- Workload:
  - 385 MB broadcast
- Communication cost(60-machine)
  - 45% broadcast
  - Over 60 machines: stop scaling

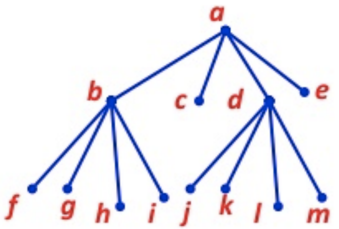


(b) Collaborative Filtering

Broadcast Coflow

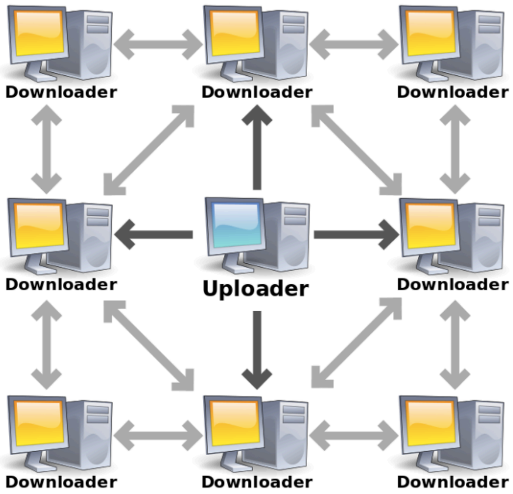
- Solutions:
  - Shared file system
    - Centralized storage system quickly become a bottleneck as receivers grows
  - d-ary distribution trees
    - Every vertex has no more than d children
    - Data is divided into blocks
    - Limitations:
      - Sending capacity at leaf machines not utilized
      - Slow machine will slow down its entire sub-tree





Broadcast Coflow

- Nature of a cluster:
  - High speed and low latency connections
  - Absence of selfish peers
  - No malicious data corruption
- BitTorrent protocol:
  - Communication protocol of peer-to-peer sharing
  - Used to distribute data and files over the Internet
  - Use BitTorrent client to send or receive files
- Cornet is a BitTorrent-like protocol optimized for datacenters



Broadcast Coflow

	BitTorrent	Coflow
Block Size	Small(256 KB)	Large(4 MB)
Peer	Can leave anytime	Full capacity over the full duration
Data integrity	SHA1 for each block	Single check over whole data

Broadcast Coflow

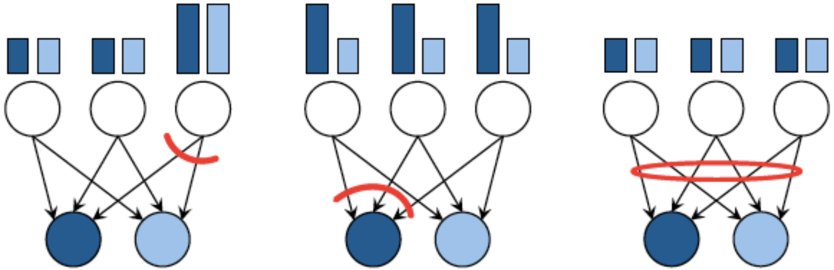
- Two extensions:
  - Cornet Topology
    - Assume the network topology is known in advance
    - Prioritize machines on the same rack as the receiver
  - Cornet Clustering
    - Infer and exploit the underling network topology automatically

Shuffle Coflow

- Solutions:
  - Hadoop:
    - Receiver opens connections to multiple random senders
    - Rely on TCP fair sharing among these flows
    - Close to optimal when data sizes are balanced
    - 1.5x worse than optimal with unbalanced data

Shuffle Coflow

- Bottlenecks:
  - Sender-side
  - Receiver-side
  - In-network

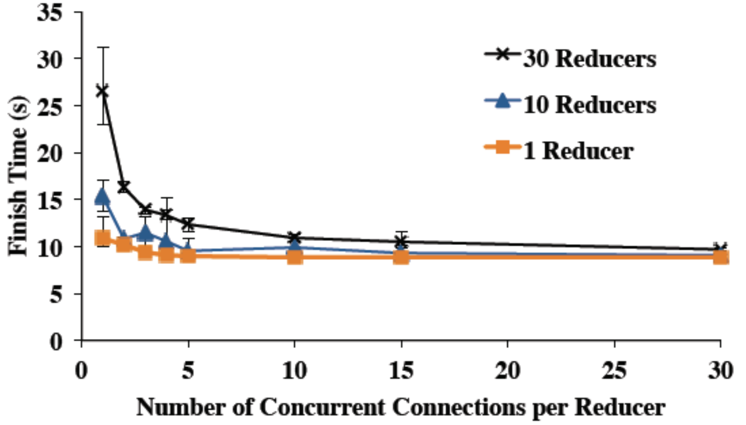

(a) Sender-side (b) Receiver-side (c) In-network

- The minimum completion time:

$$\Gamma = \max\left(\max_i \frac{\sum_j d_{ij}}{Rem(P_i^{\text{in}})}, \max_j \frac{\sum_i d_{ij}}{Rem(P_j^{\text{out}})}\right)$$

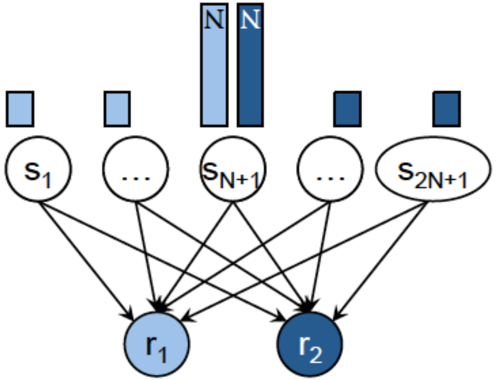
Shuffle Coflow

- Experiment:
  - 30 senders and 1-30 receivers
  - 1 GB of data for each receiver
  - Random connection
- Two trends:
  - The power of 2:
    - single fetch connection leads to poor performance, but improves quickly even with 2 connections
  - With enough connections, transfer time reaches the lower bound
- Reason:
  - Reduce collisions
  - Reduce the effect of imbalances



Shuffle Coflow

- MADD
  - Minimize completion time
  - Finish before its bottleneck
  - Guarantee by ensure rates is at least $\frac{d_{ij}}{\Gamma}$



Experiment & Evaluation

- In general
  - Cornet performs 4.5x better than default Hadoop and BitTorrent
  - Further 2x improvement with Cornet Topology Awareness
  - MADD can improve shuffle by 29%
  - Taken together
    - Reduce application communication times by up to 3.6x
    - Speed up jobs by up to 1.9x

Experiment & Evaluation

- Broadcast
- Cornet remains within 33% of the theoretical lower bound
- Structured mechanisms works well only for small scale
- HDFS performs well only for small amount of data. Trade-off between creating and reading replicas

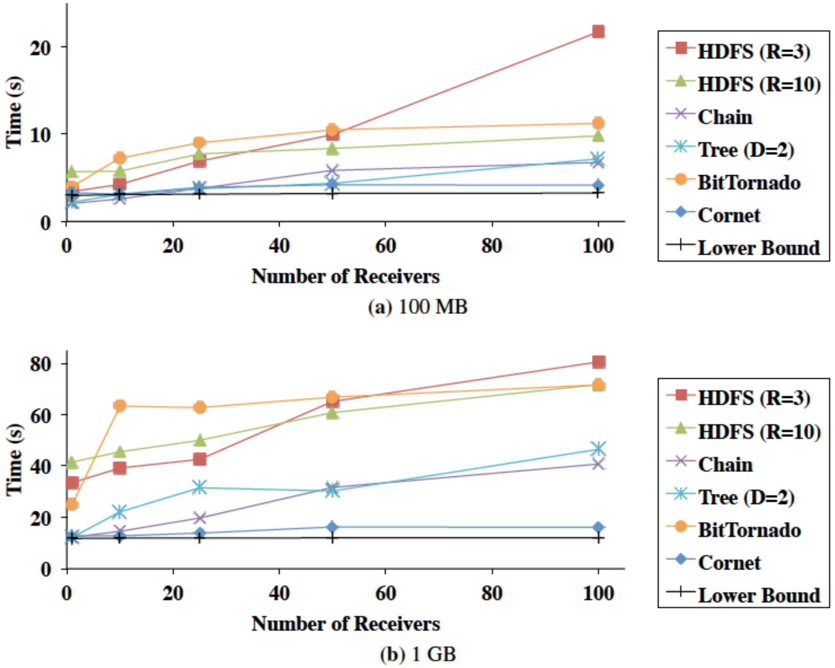


Figure 4.7: [EC2] Completion times of different broadcast mechanisms for varying data sizes.

Experiment & Evaluation

- Per-machine completion times
- All receivers finished simultaneously in Cornet
- BitTorrent is similar except variation in individual completion time
- Chain and Tree is horizontally segmented because of stragglers
- HDFS-10 starts later but finishes faster than HDFS-3 because of more replicas

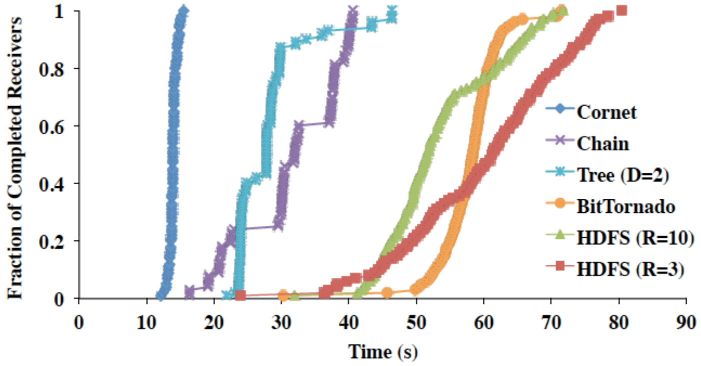
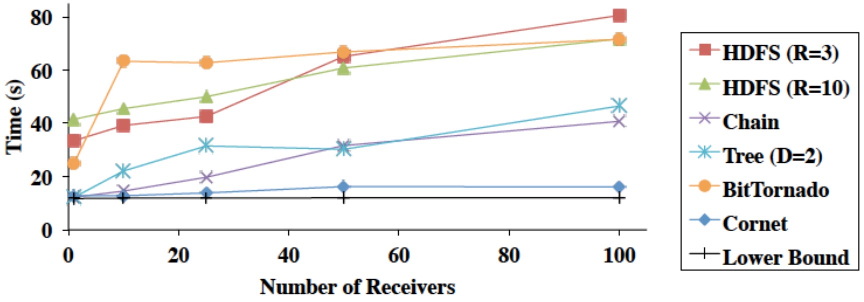


Figure 4.8: [EC2] CDF of completion times of individual receivers while transferring 1 GB to 100 receivers using different broadcast mechanisms. Legend names are ordered by the topmost points of each line, i.e., when *all* the receivers completed receiving.

Experiment & Evaluation

- Chain and tree based approaches are faster than Cornet for small number of machines and small data set
- Block sizes and polling intervals in Cornet prevent from utilizing bandwidth



(b) 1 GB

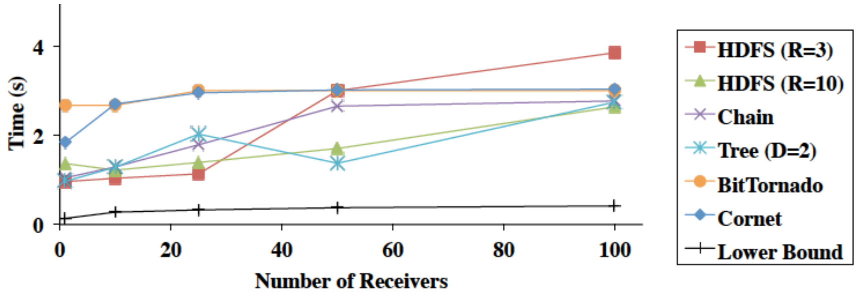


Figure 4.9: [EC2] Broadcast completion times for 10 MB data.

Experiment & Evaluation

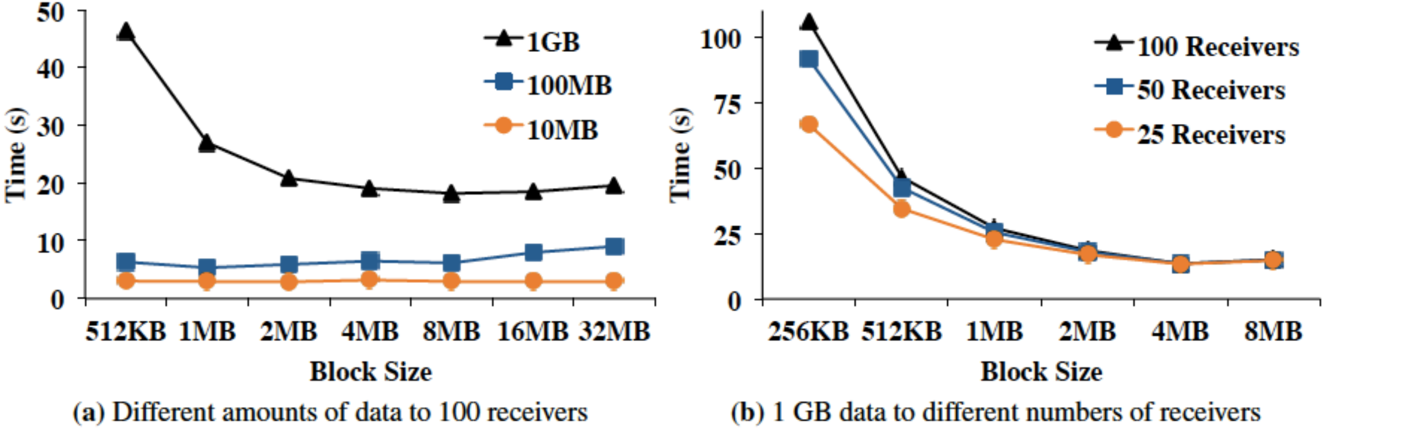


Figure 4.10: [EC2] Cornet completion times for varying block sizes.

- Impact of block size
- Too large block size limits sharing between peers
- Small size increases overheads

Experiment & Evaluation

- Hypothesis: there is a significant difference between block transfer within a rack or between racks
- Cornet: any receiver randomly contact any other receiver
- CornetTopology: disallow communications across partitions given the topology information
- CornetClustering: dynamically inferred partitioning

Experiment & Evaluation

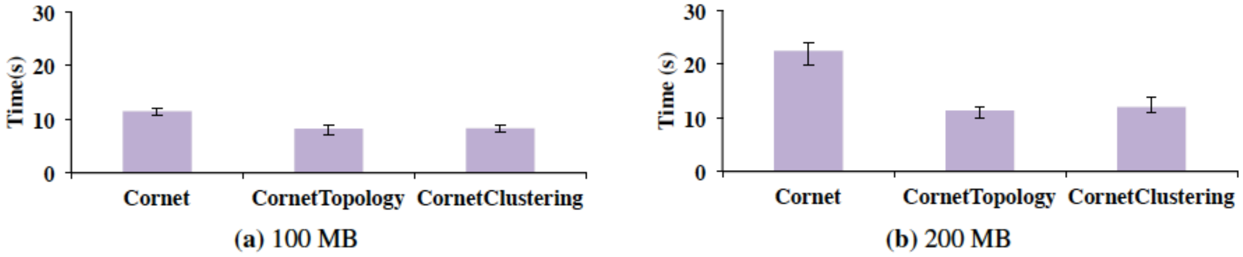
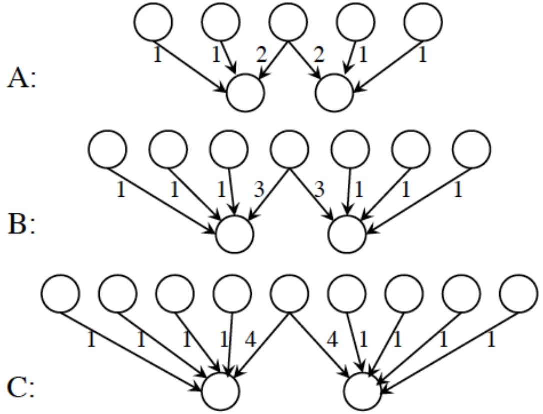


Figure 4.11: [DETERlab] Cornet completion times when the rack topology is unknown, given, and inferred using clustering.

- Average completion time to transfer to 30 receivers over 10 runs
- 200 MB:
  - CornetTopology decreased by 50%
  - CornetClustering reduces 47%

Experiment & Evaluation

- Standard shuffle(each reducer simultaneously connects to at most 3 mappers) and MADD



Topology	Standard Shuffle	MADD	Speedup	Theoretical Speedup
A	83.3 (1.1)	70.6 (1.8)	18%	25%
B	131 (1.8)	105 (0.5)	24%	33%
C	183 (2.6)	142 (0.7)	29%	38%

Experiment & Evaluation

- Communication overhead decreased from 42% to 28%, 22% faster overall
- 2.3x speedup in broadcast, 1.23x in shuffle

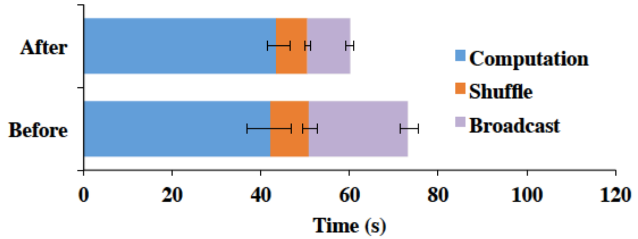


Figure 4.14: [EC2] Per-iteration completion times for the logistic regression application before and after using Cornet and MADD.

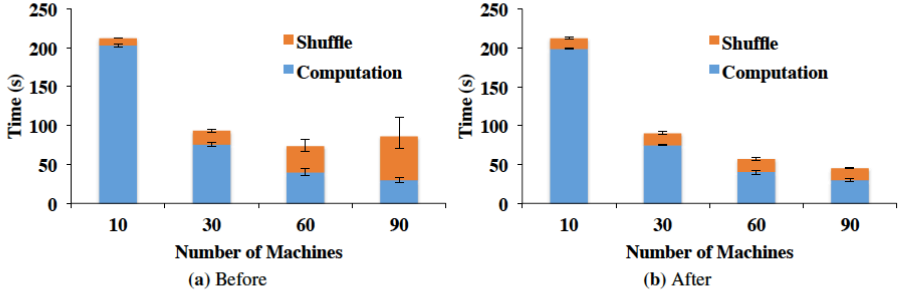


Figure 4.15: [EC2] Per-iteration completion times when scaling the collaborative filtering application using MADD.

Thanks!

QA?