CODA: Towards Automatically Identifying and Scheduling COflows in the DArk

Hong Zhang, Li Chen, Bairen Yi, Kai Chen, Mosharaf Chowdhury, Yanhui Geng
SIGCOMM, 2016
Coflow: A Networking Abstraction for Cluster Applications

Mosharaf Chowdhury and Ion Stoica
Hotnets, 2012
Problem

• Flow-level optimizations (e.g. FCT, fairness) do not do well with data-parallel applications
  • Traditional flows are too fine-grained

• Need to enforce policy on groups of flows
  • Optimal completion
  • Strict deadlines

• Depends on application semantics
Observations

• Machines organized by functionality
• Communication between groups follows patterns
  • Shuffle
  • Broadcast
  • Aggregate
Solution

- Coflow: semantically related group of flows between machine groups
- Ordering of coflows
  - Finishes before
  - Starts after
- Intent-driven API between driver (coordinator), sender, receiver

<table>
<thead>
<tr>
<th>Operation</th>
<th>Caller</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>create(pattern, [options])</code> → <code>handle</code></td>
<td>Driver</td>
</tr>
<tr>
<td><code>update(handle, [options])</code> → <code>result</code></td>
<td>Driver</td>
</tr>
<tr>
<td><code>put(handle, id, content, [options])</code> → <code>result</code></td>
<td>Sender</td>
</tr>
<tr>
<td><code>get(handle, id, [options])</code> → <code>content</code></td>
<td>Receiver</td>
</tr>
<tr>
<td><code>terminate(handle, [options])</code> → <code>result</code></td>
<td>Driver</td>
</tr>
</tbody>
</table>
Coflow Scheduling Problem

• When to start flows?
• What rate should each flow progress at?
• Optimize for
  • Coflow Completion Time (CCT)
  • Deadlines
• NP-Hard
Coflow Scheduling Problem

(a) Datacenter fabric

(b) Coflow arrival times

Coflow Arrival Time

0  1  0

(c) Per-flow fairness

Time

3  6

(d) Decentralized LAS

Time

3  6

(e) CLAS

Time

3  6

(f) The optimal schedule
Coflow Scheduling Problem

(a) Per-flow fairness

(b) Per-flow prioritization

(c) WSS [15]

(d) The optimal schedule
Coflow Scheduling Solutions

• Varys (2014) ~ 3x speed up compared with per-flow
  • Smallest Effective Bottleneck First
  • Minimum Allocation for Desired Duration

• Aalo (2015)
  • Non-clairvoyant
  • Discretized Coflow Least Attained Service (by total bytes sent per coflow)
  • Weighted fair queuing across FIFOs

• CODA (2016)
  • Application transparent coflow identification
  • Error tolerant coflow scheduler
CODA: Towards Automatically Identifying and Scheduling COflows in the DArk

Hong Zhang, Li Chen, Bairen Yi, Kai Chen, Mosharaf Chowdhury, Yanhui Geng
SIGCOMM, 2016
Problems

• Infeasible to enforce correct usage of same coflow API in real clusters
• Evidence from Spark and Hadoop
  • Intrusive refactoring
  • Blocking, non-blocking I/O mismatches
  • Third-party libraries
• Want to identify coflows for any data-parallel application
Goals

• Application-transparent coflow identification
  • No application modification
• Error-tolerant coflow scheduler
• Deployable / compatible
COflows in the DArk

• Use machine learning to identify coflows from flow attributes
  • Explicit and implicit attributes
  • Incremental Rough-DBSCAN for identification

• Scheduler which tolerates errors
  • Late binding
  • Intra-coflow prioritization
Design

CODA Master

Application-Transparent Identifier

Online Incremental Clustering

Offline Attribute Exploration

Distance Metric Learning

Error-Tolerant Scheduler

Late Binding

Inter-Coflow Prioritization

Intra-Coflow Prioritization

CODA Agent(s)

Gather and Prune Flow Information

Enforce Coflow Schedule
Identification-Scheduling Cycle

• Gather and prune information on Agents
  • Per-flow IP and port
  • TCP only
  • Size threshold

• Agents periodically export to Master
  • Identify coflows
  • Generate schedule to minimize CCT

• Schedule returned to Agents for enforcement
  • tc
Identification: Goals

• Not require any modification to applications
• Accurate
• Fast
Identification: Method

- Multi-level attribute exploration
- Flow distance calculation
- Clustering
Identification: Attributes

• Explicit (flow-level)
  • Start time ($S_{time}$)
  • Mean packet size ($M_{size}$)
  • Packet size variance ($V_{size}$)
  • Average packet inter-arrival time ($M_{int}$, $V_{int}$)

• Implicit (community-level)
  • Community distance $\in \{0,1\}$ ($D_{com}$)
    • via spectral clustering community detection
  • Port distance $\in \{0,1\}$ ($D_{prt}$)
    • 1 iff same destination port and IP
Identification: Unused Attributes

• Flow-level
  • Flow size
  • Duration

• OS-level
  • PID
Identification: Distance

• Run typical benchmarks to get labeled data
  • Traces (data)
  • Coflow data annotated by applications (labels)
• Optimize attribute weights using Newton Raphson method
  • Prune out near zero weight attributes
Identification: Distance

• Start time is only useful flow-level attribute
• Community distance is useful
• Port distance utility depends on application

\[ A_s = \begin{bmatrix}
S_{time} & M_{size} & V_{size} & M_{int} & V_{int} & D_{com} & D_{prt} \\
3.825 & 0.000 & 0.000 & 0.000 & 0.000 & 5.431 & 0.217
\end{bmatrix} \]

\[ A_h = \begin{bmatrix}
S_{time} & M_{size} & V_{size} & M_{int} & V_{int} & D_{com} & D_{prt} \\
3.472 & 0.000 & 0.000 & 0.000 & 0.000 & 3.207 & 0.000
\end{bmatrix} \]

\( A_s \) : Spark weights, \( A_h \) : Hadoop weights
Identification: Clustering

• DBSCAN (O(n^2))
  • Don’t know the number of clusters in advance
  • Handles outliers well

• Rough-DBSCAN (O(nk + k^2))
  • Select k leaders, run DBSCAN on leaders, replace with followers

• Incremental Rough-DBSCAN (O(mk + k^2))
  • Only do leader selection with m newly started/finished flows
  • For most intervals m is small
Identification: Clustering

**Algorithm 1 Incremental R-DBSCAN**

1: **procedure** CLUSTERING(Previous leader-follower structure \( L \) (initially \( \emptyset \)), New flows \( F_{\text{new}} \), Flows left \( F_{\text{left}} \), range \( \tau \))

2: \hspace{1em} for each Flow \( f \in F_{\text{new}} \) do  \hspace{1em} \textbf{▷} Add new flows

3: \hspace{2em} Find a leader \( l \in L \) such that \( d(f, l) < \tau \)

4: \hspace{2em} if no such leader exists then

5: \hspace{3em} \( L = L \cup \{f\} \)  \hspace{1em} \textbf{▷} Create a new leader

6: \hspace{3em} \( f.\text{followers} = \{f\} \)

7: \hspace{2em} else

8: \hspace{3em} \( l.\text{followers} = l.\text{followers} \cup \{f\} \)  \hspace{1em} \textbf{▷} Add to an old leader

9: \hspace{2em} end if

10: end for

11: \hspace{1em} for each Flow \( f \in F_{\text{left}} \) do  \hspace{1em} \textbf{▷} Delete left flows

12: \hspace{2em} Find its leader \( l \)

13: \hspace{2em} if \( f = l \) then

14: \hspace{3em} Delete \( l \) from \( L \) if \( l.\text{followers} = \{l\} \)

\hspace{3em} \textbf{▷} A leader is deleted only when it has no other followers

15: \hspace{3em} else

16: \hspace{4em} \( l.\text{followers} = l.\text{followers} \setminus \{f\} \)

17: \hspace{4em} end if

18: \hspace{2em} end if

19: \hspace{1em} end for

20: \hspace{1em} Run \( DBSCAN(L, \epsilon, 1) \) and get \( \mathcal{C}' \) (cluster of leaders)

21: \hspace{1em} Obtain \( \mathcal{C} \) by replacing each leader by its followers

22: \hspace{1em} return cluster of flows \( \mathcal{C} \)

23: **end procedure**
Scheduling: Errors

- Pioneers: flows scheduled into earlier coflows
- Stragglers: flows scheduled into later coflows
- Stragglers are worse in terms of CCT

(a) A pioneer increases the average CCT to \( (1.1+2)/2 = 1.55 \)

(b) A straggler increases average CCT to \( (2+2)/2 = 2 \)
Scheduling: Late Binding

• Defer identification of flows lying on boundaries
• Assign these flows to the higher priority coflow later
  • Favors creation of pioneers over stragglers
Scheduling: Intra-flow prioritization

• Per-flow prioritization
  • Based on bytes sent within each identified coflow
• Small flows straggling in large flows get priority
• Mitigates falsely merged coflows
Scheduling: Implementation

- Extension
  - Extend each identified coflow by \( d \)
  - Duplicates are later scheduled with higher priority

- Inter-coflow prioritization
  - Discretized Coflow Least Attained Service
  - Prioritized coflow queues

- Intra-coflow prioritization
  - Smallest-first using MLFQ
Scheduling: Implementation

Algorithm 2 CODA's Error-Tolerant Scheduler

1: procedure COFLOWEXTENSION((Identified) Coflows C, diameter d)
2:   \[ C^* = \emptyset \] \Comment{Set of extended coflows to be returned}
3:   for all Coflow \( C \in \mathcal{C} \) do
4:     \[ G = \{(\text{Flows}) f_i | d(f_i, C) \leq d\} \]
5:     \[ C^* = C^* \cup \{C \cup G\} \] \Comment{Extend coflow and add}
6:   end for
7:   return \( C^* \)
8: end procedure

9: procedure INTERCOFLOW(Extended Coflows \( C^* \), Coflow Queues \( Q^C \))
10: for all \( i \in [1, |Q^C|] \) do
11:     for all \( C^* \in Q^C_i \) do \Comment{\( Q^C_i \) sorted by arrival time}
12:         IntraCoflow(\( C^* \), \( Q^F \))
13:     end for
14: end for
15: end procedure

16: procedure INTRACOFLOW(Extended Coflow \( C^* \), Flow Queues \( Q^F \))
17:   for all \( j \in [1, |Q^F|] \) do
18:       for all Flows \( f \in C^* \cap Q^F_j \) and not yet scheduled do
19:         \( f\text{'s rate} = \text{Max-min fair share rate} \)
20:         Mark \( f \) as scheduled \Comment{Binds \( f \) to the highest priority coflow among all it belongs to}
21:       end for
22:   end for
23: end procedure

24: procedure CODASCHEDULER(\( C \), \( Q^C \), \( Q^F \), d)
25:   \( C^* = \text{CoflowExtension}(C, d) \)
26:   InterCoflow(\( C^* \), \( Q^C \))
27: end procedure
Implementation: for validation

• Implement Aalo’s coflow API in Hadoop
  • Instrumented RPC message format
  • Modified networking library to push coflow data from RPC to TCP layer
  • Lookup coflow information in binary messages

• Instrumented java byte code to capture coflows at runtime
  • Job IDs in Hadoop
  • Spark shuffles
Implementation: CODA

• Agent could be kernel module
• Reused java bytecode instrumentation instead to collect
  • Flow start time
  • Source / destination IP / port
• Two-level HTB in tc for enforcement
• ~1% CPU overhead
Evaluation Details

• Facebook Hive/MR trace from 3000-node cluster
• 500 coflows, $7 \times 10^5$ flows
• Scaled down to 40 Gbps testbed bisection bandwidth
• Synthesized start times from uniform and exponential distributions
  • Based on experience with other benchmarks

(a) Inter coflow arrival time  (b) No. of concurrent coflows
Results

• Coflow Completion Time compared with per-flow fairness
  • 2.4x improvement average
  • 5.1x improvement 95-th percentile

• Coflow identification accuracy
  • 90% under normal production work loads
  • 60% under challenges
  • Attribute weight learning is critical (40% improvement)
  • Iterative R-DBSCAN is 600x faster than traditional DBSCAN with negligible loss
Results: Testbed Performance

(a) Accuracy

(b) CCT and JCT

Normalized Comp. Time

CCT

JCT

CODA
Results: Simulation Scalability

(a) Overheads at scale

(b) Impact of $\Delta$

Coeflow Comp. Time ($\times 10^4$ s)
Results: Different Workloads

(a) Normal workloads
(b) Batch arrival
(c) Stretched arrival

Synthetic challenges
Results: Effectiveness and Sensitivity

(d) Effectiveness of DML

(Duration Metric Learning)

(e) Sensitivity to $\epsilon$

in Iterative-R-DBSCAN classifier
Results: Scheduler (Normal)

(a) Normalized CCT

(b) CCT distribution
Results: Scheduler (Challenge)

(a) Batch arrival case (Hadoop)

(b) Stretched arrival case