CODA: Toward Automatically Identifying and Scheduling COflows in the DArk

Hong Zhang[1]

Li Chen[1], Bairen Yi[1], Kai Chen[1], Mosharaf Chowdhury[2], Yanhui Geng[3]

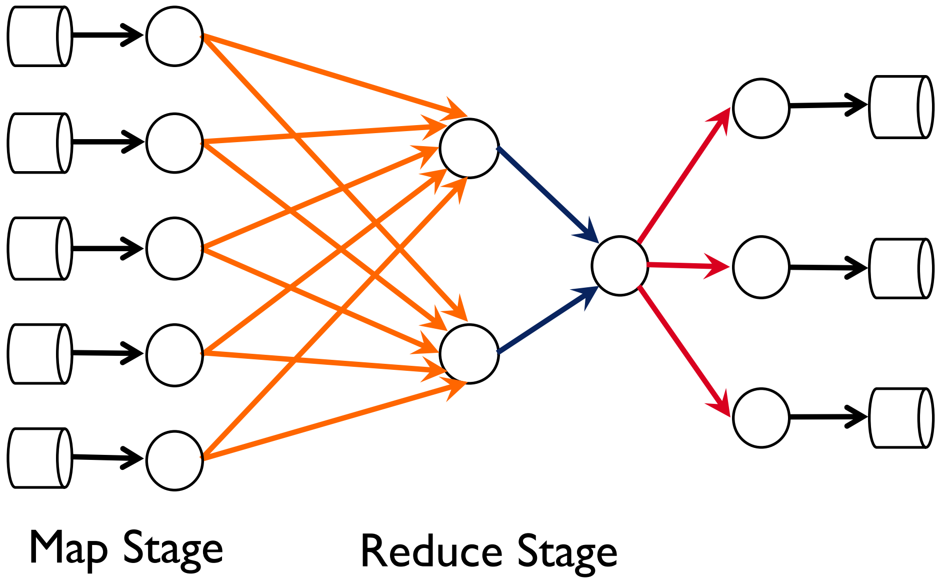
[1]SING Group, Hong Kong University of Science and Technology

[2]University of Michigan [3]Huawei

Communication is Crucial

- *Many distributed data-parallel applications involve a rich communication stage*

- *As SSD-based and in-memory systems proliferate, the network is more likely to become the bottleneck*

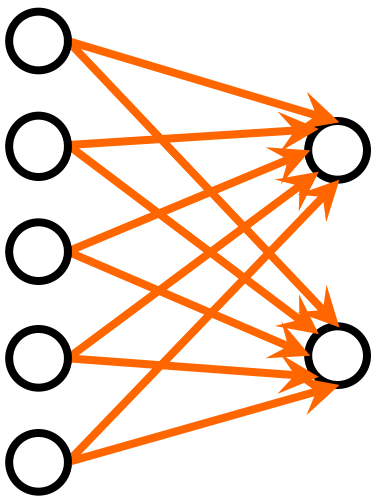


Improving the communication performance is crucial for these applications

Flow

- ***Application agnostic*** *--- schedule each flow independently*
- *Does not directly minimize the completion time of communication stages*

Coflow



- *A collection of parallel flows sharing a common application-level goal*
- *Minimizes the completion time of communication stages*

Coflow HotNets'12

Baraat SIGCOMM'14

Varys SIGCOMM'14

Rapier Infocom'14

Aalo SIGCOMM'15

Assumption: *all distributed data-parallel applications have to be modified to correctly use the same coflow API*

- **Difficulty 1:** Enable current Coflow API requires intrusive refactoring
- **Difficulty 2:** Hard to modify all applications and keep them up to date

Can we automatically identify and schedule coflows without manually modifying any data-parallel applications?

Varys

*Efficiently schedules coflows with **complete** application information*

Aalo

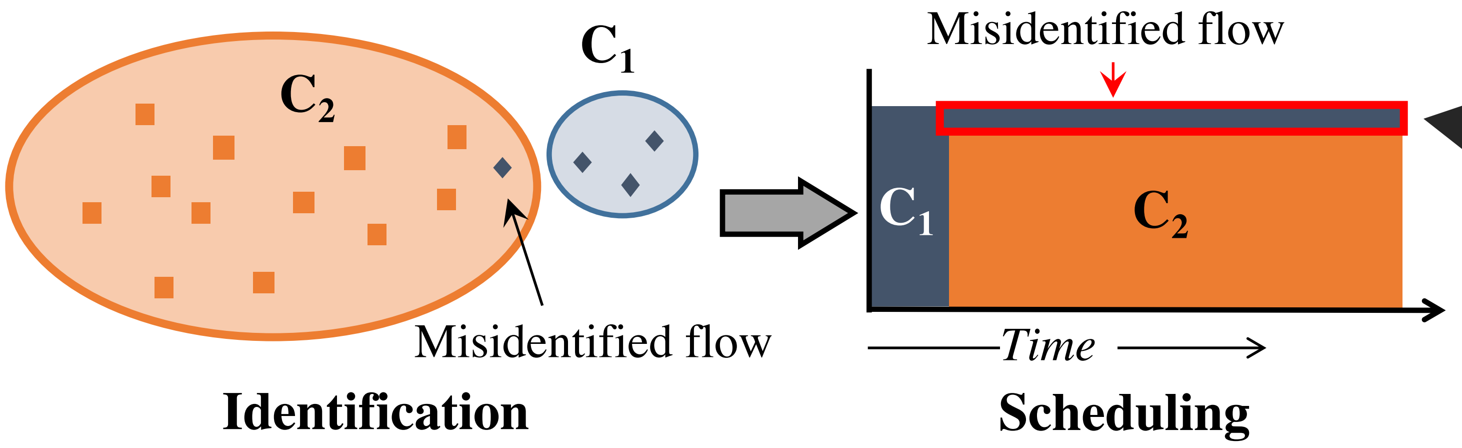
*Efficiently schedules coflows with **incomplete** application information*

CODA

*Efficiently **identifies** and schedules coflows with **no** information from application*

Challenge 1: *How to identify coflows without modifying applications?*

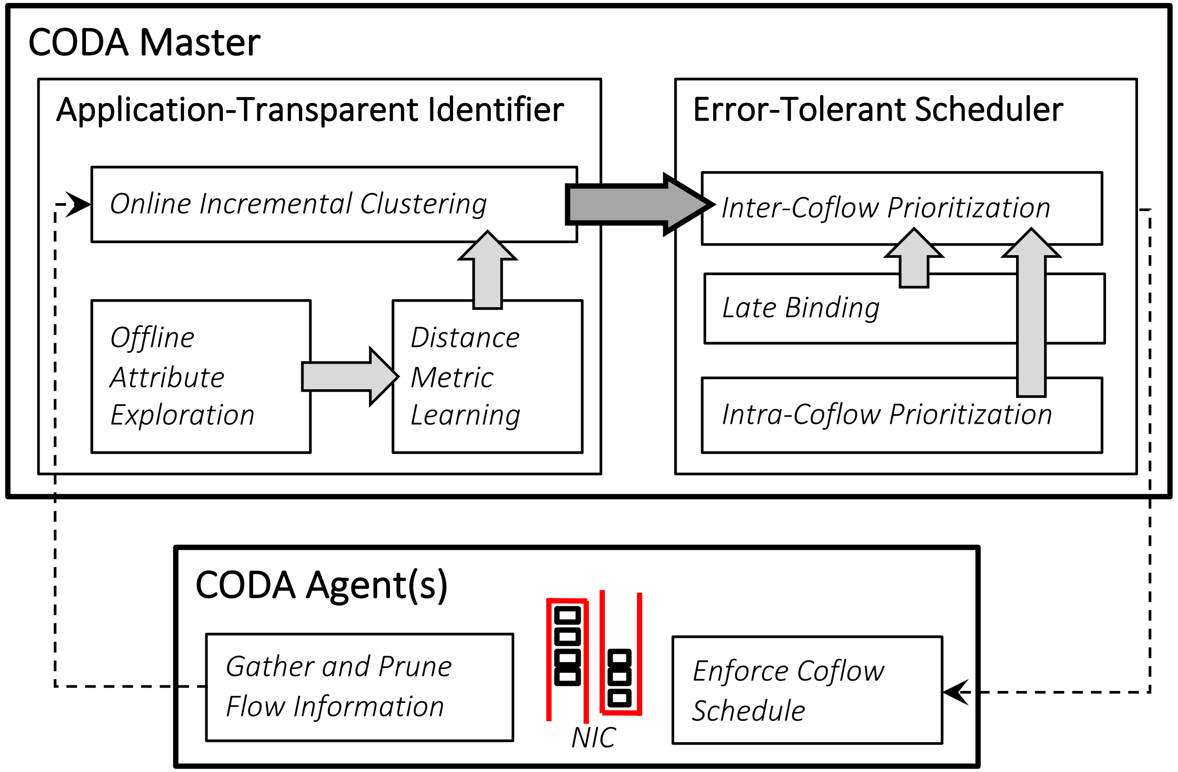
--- Transparent, accurate, fast



A mis-identified flow of C1 significantly affects the completion time of its parent Coflow C1.

Challenge 2: *How to schedule coflows with identification errors?*

CODA in One Slide



Application-Transparent Coflow Identification

Idea: a simple 3-step identification framework

Error-tolerant Coflow Scheduling

Idea: Combine inter-coflow scheduling with two novel heuristics

Application-Transparent Coflow Identification

Step 1 --- Attribute Exploration

---- *search for candidate attributes*

- *Flow start time, inter-packet arrival time, …*
- *Communication pattern*
- *Application-specific attributes (e.g., port assignment rules)*
- *….*

Step 2 --- Distance Calculation

---- *identify the importance of each attribute*

Input

- *Candidate attributes*
- *Training data*

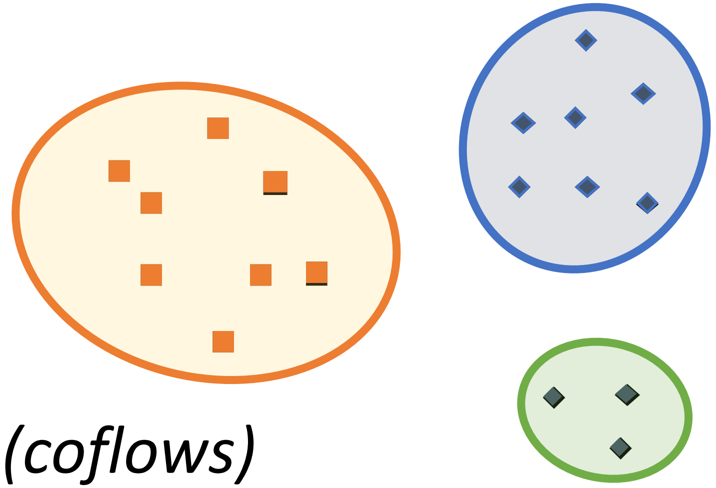
Distance Metric Learning

Output: distance function

Small/large distances between flows of same/different coflows;

Application-Transparent Coflow Identification

- **Step 3 --- Online Clustering**



- **Basic algorithm --- DBSCAN**
  - *Distance-based*
  - *Automatically determine the number of clusters (coflows)*
- **How to speed up?**
  - *Idea 1: sacrifice some accuracy for much faster speed*
  - *Idea 2: incremental identification*

----- Errors are inevitable

Error-tolerant Coflow Scheduling

D-CLAS (Aalo-SIGCOMM'15)

---inter-coflow scheduling to minimize average CCT

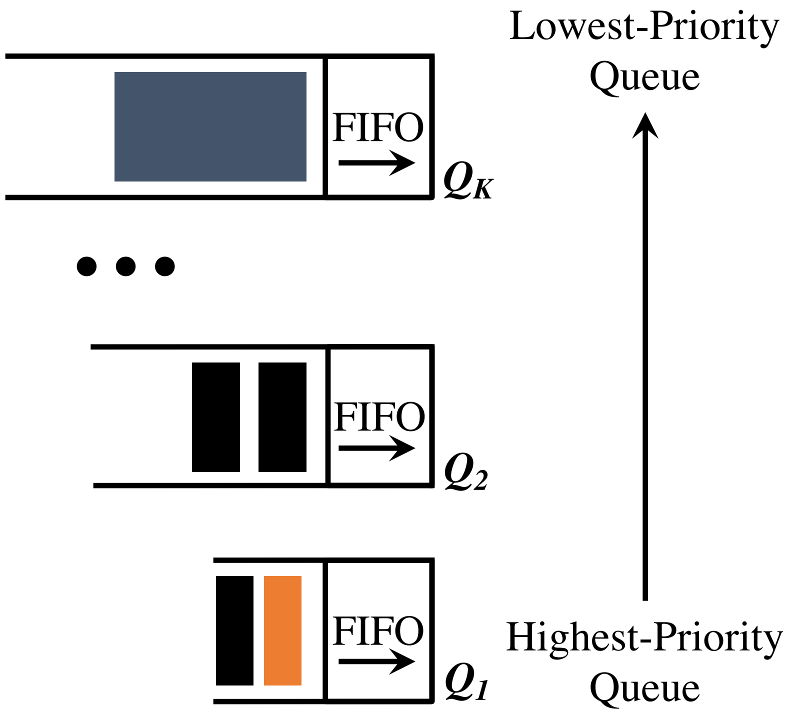
MLFQ with exponentially spaced thresholds

Priority discretization

- *Drop priority when total # of bytes sent exceeds predefined thresholds*

Scheduling policies

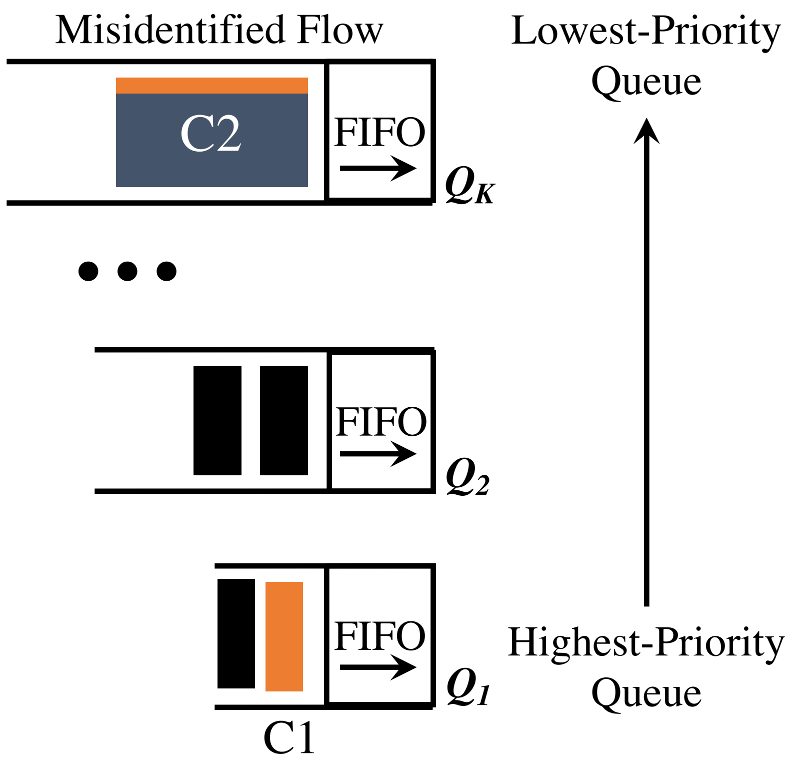
- *Prioritization across queues*
- *FIFO within the same queue*



Error-tolerant Coflow Scheduling

- **D-CLAS with identification errors**

  ---- Errors may significantly affect the performance of D-CLAS



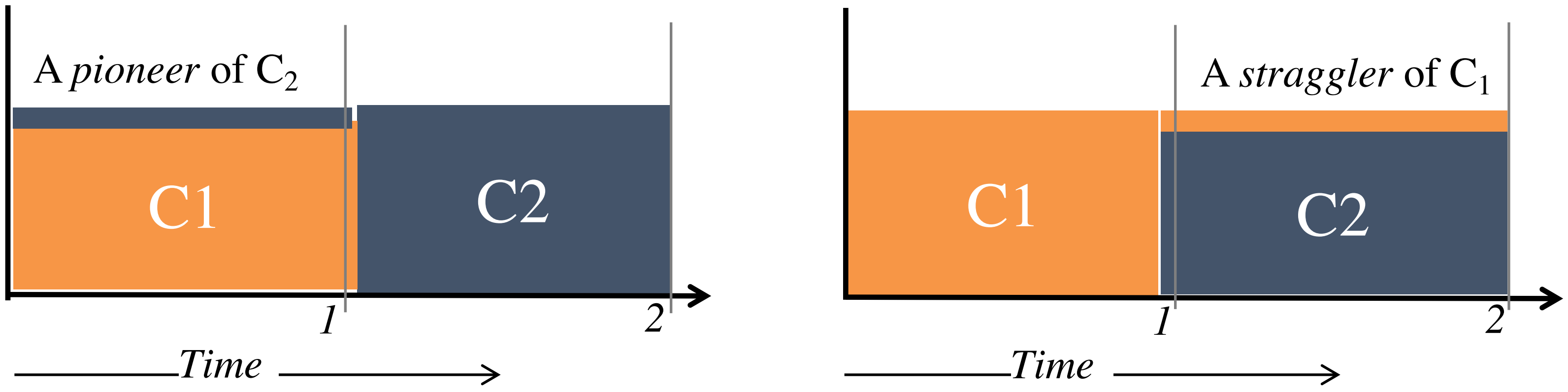
Error-tolerant Coflow Scheduling

- **Impact of different identification errors**
  - ***Pioneers***: *Flows that are misidentified into a coflow that is scheduled* ***earlier*** *than their parent coflow*

  - ***Stragglers***: *Flows that are misidentified into a coflow that is scheduled* ***later*** *than their parent coflow*

Error-tolerant Coflow Scheduling

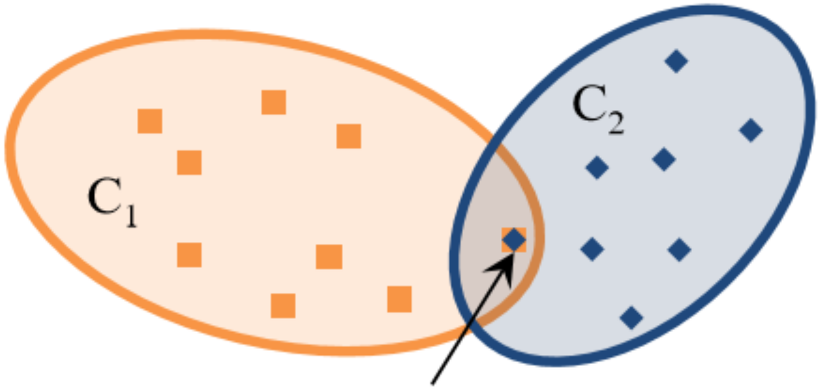
- **Impact of different identification errors**



Observation 1: *stragglers are likely to more negatively affect the average coflow completion time than pioneers*

Error-tolerant Coflow Scheduling

- **Design Principle 1: Late binding**
  ---- Reduce the number of stragglers



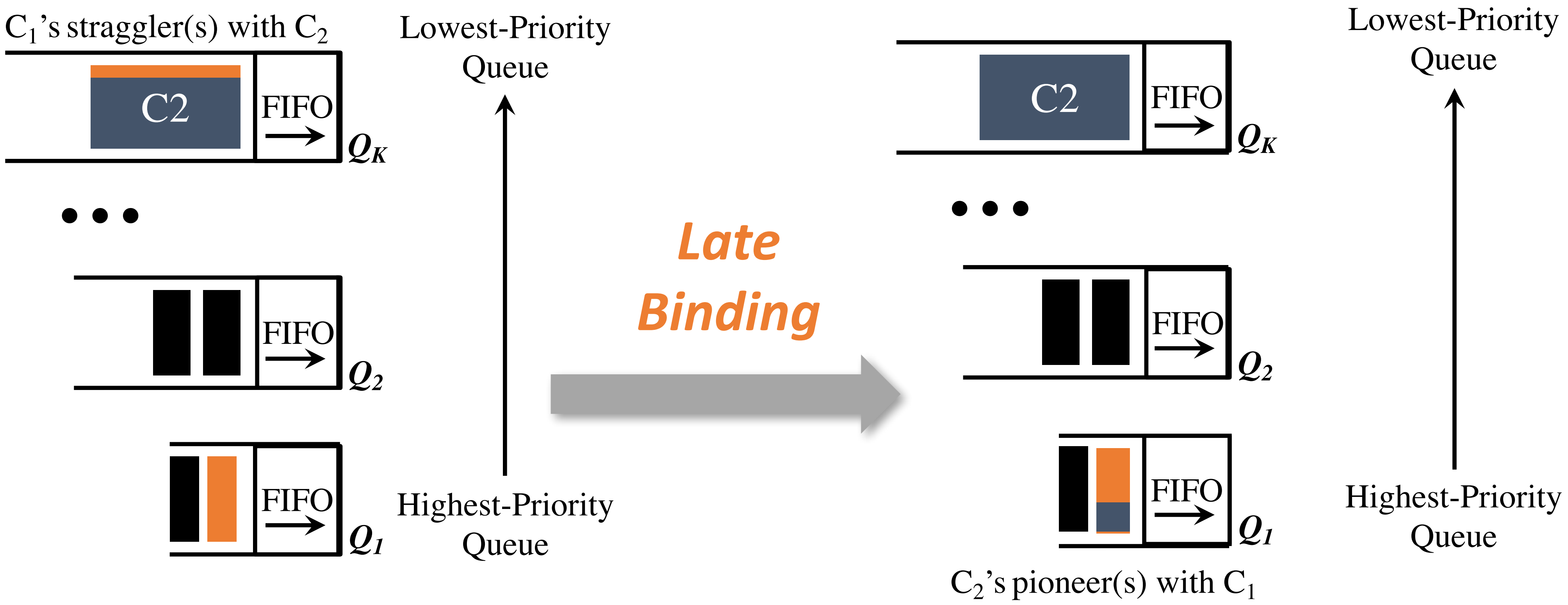
Potential source of misidentification

- *For a flow that can potentially belong to either C1 or C2*
- *Delay the decision and consider it to be in **both C1 and C2** at first*
- *Only during scheduling, assign it to the coflow with the **higher priority***

This flow does not become a straggler, no matter whether it belongs to C1 or C2!

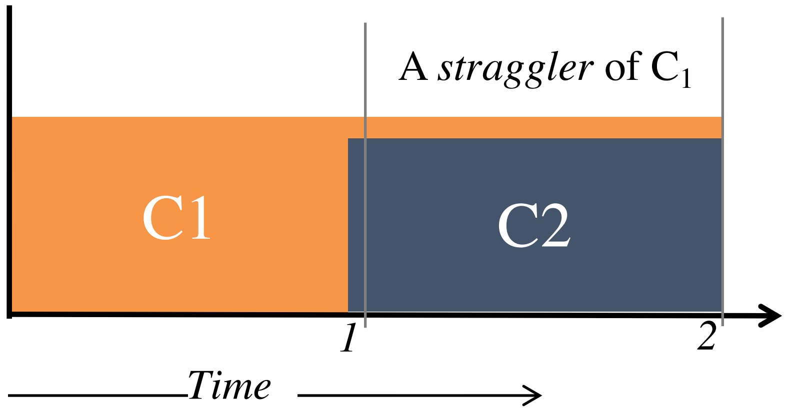
Error-tolerant Coflow Scheduling

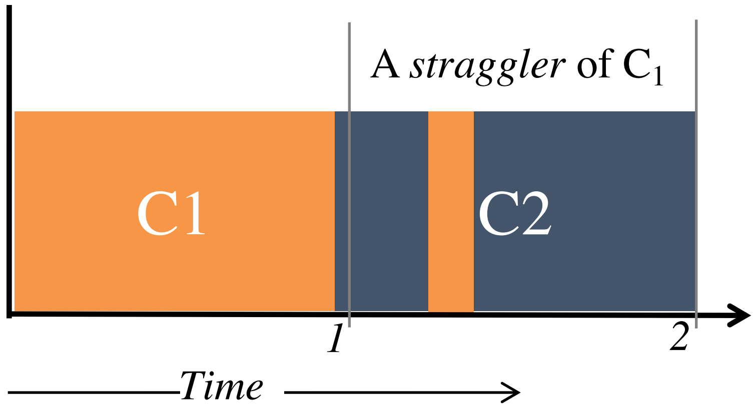
- ***Late binding**: Reduce the number of stragglers **at the cost of more pioneers***



Error-tolerant Coflow Scheduling

- ***Observation 2:*** *Intra-coflow prioritization matters*

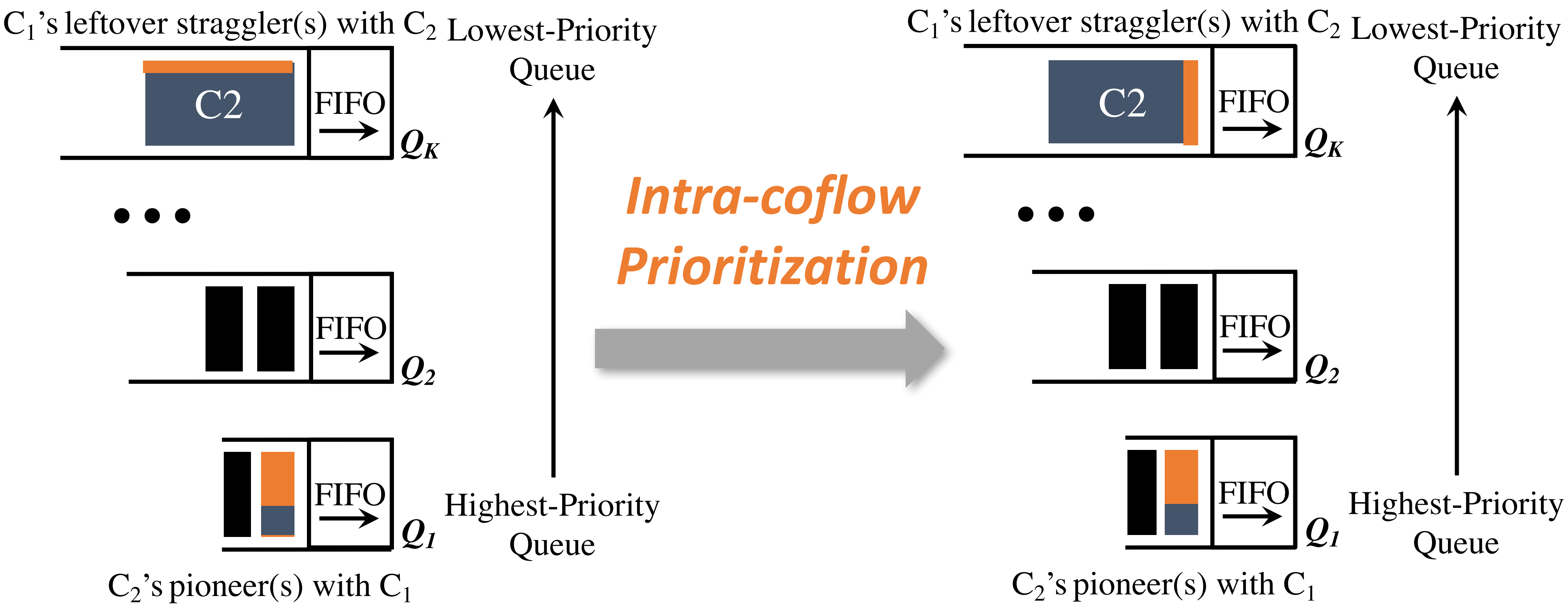




- **Design Principle 2: Intra-coflow prioritization**
  - *Idea: prioritize small flows within a coflow*

Error-tolerant Coflow Scheduling

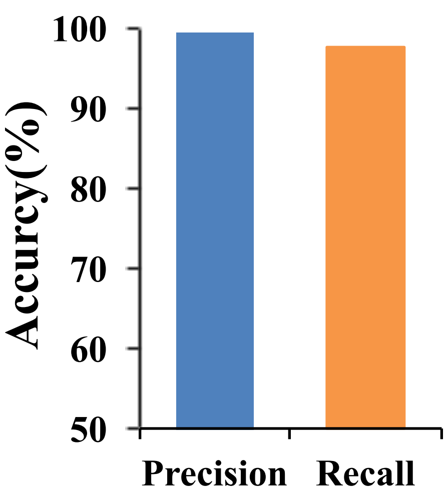
- ***Intra-coflow prioritization***: *Reduce the impact of leftover stragglers*

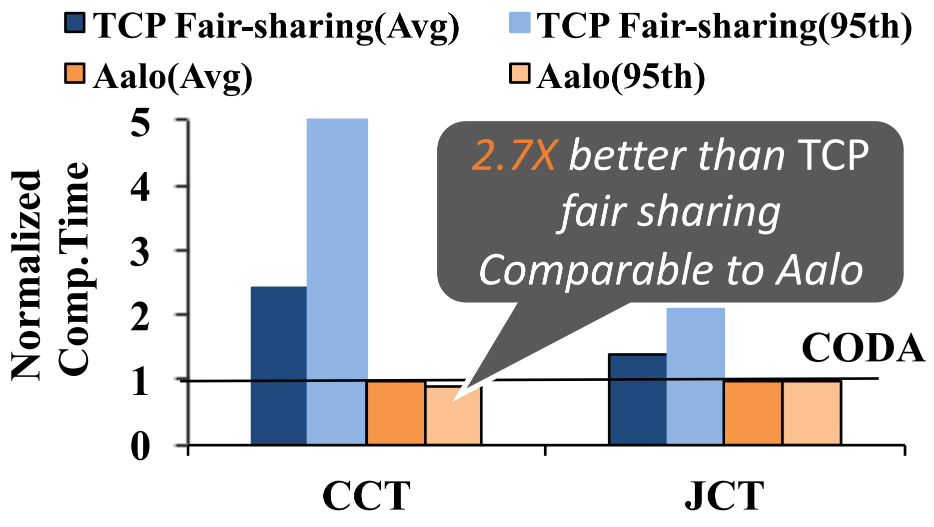


How does CODA Perform in Practice?

- **Workload**
  - *1-hour 3000-machine Mapreduce trace*
  - *500 coflows (7x10^5 flows)*
- **Settings**
  - *40-server testbed*
- **Performance Metric**
  - *Identification*
    - *Precision*
    - *Recall*
  - *Scheduling*
    - *TCP fair-sharing*
    - *Aalo (coflow-aware solution)*

Over 90% Identification accuracy





How Effective is CODA's Error-Tolerant Scheduling?

- *Creating more challenging cases*
  - *Batch arrival*
  - *Stretched arrival*

Up to 40% accuracy loss

- *CODA under more challenging cases*

	Stretched arrival	*Batch arrival*
Per-flow Fair	*2.03X*	*1.47X*
Aalo	*0.77X*	*0.56X*
CODA	***1X***	***1X***
CODA w.t. E.T.	***1.16X***	***1.04X***

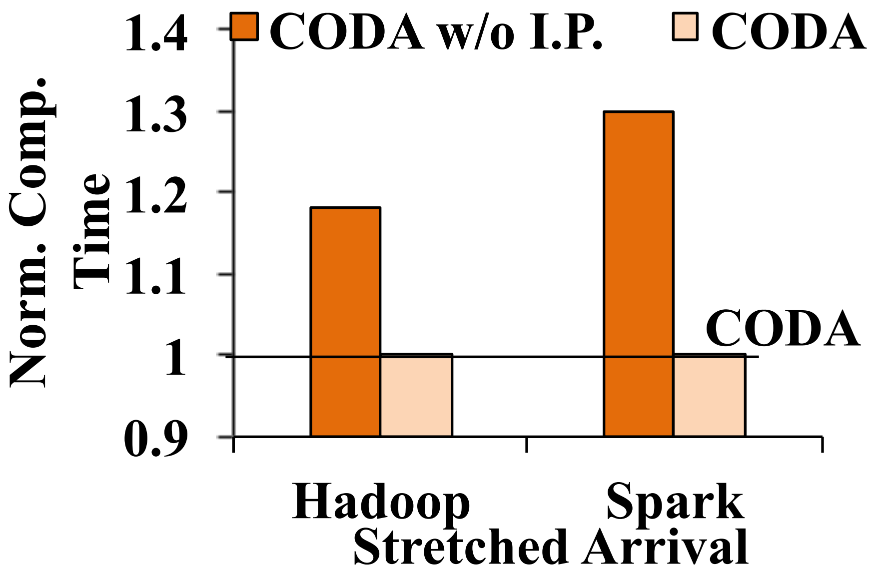
Reduce the impact of error by 40%

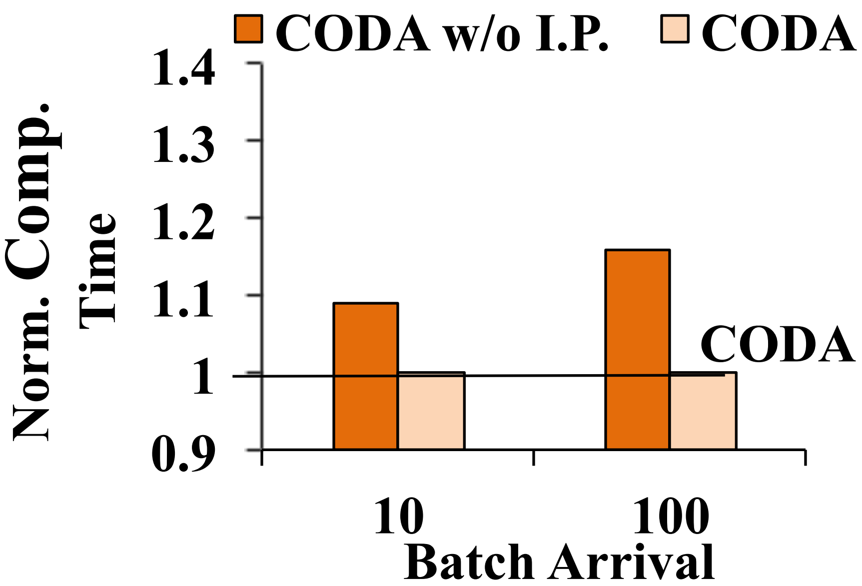
How Effective is CODA's Error-Tolerant Scheduling?

- *Benefit of Late Binding*
  - *Improve average coflow completion time by up to 10%*

Reduce the impact of error by 30%

- *Benefit of Intra-Coflow Prioritization*
  - *Improve average coflow completion time for small coflows by up to 30%*





CODA

Automatically identifies and schedules coflows ***without application modification***

Application-Transparent Coflow Identification:

-----Identify coflows without application modification

Error-Tolerant Coflow Scheduling:

-----Schedule coflows with minimal impact of identification errors

CODA, not coda

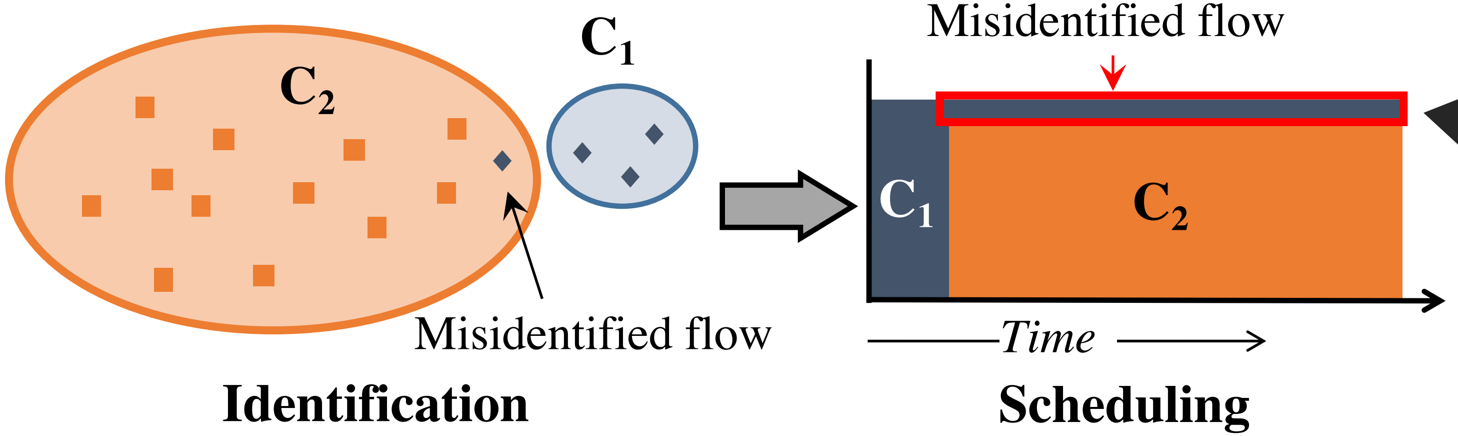
- *Apply CODA to more applications*
- *Extend CODA to coflow dependencies*
- *Perform error-tolerant ~~coflow~~ scheduling*

Thank You!

Application-Transparent Coflow Identification

Challenge 1: *How to accurately identify coflows without the help from applications in an online manner?*

Error-tolerant Coflow Scheduling



A mis-identified flow of Coflow C1 will significantly affects the CCT of its parent Coflow C1.

Challenge 2: *How to minimize the influence of identification errors?*

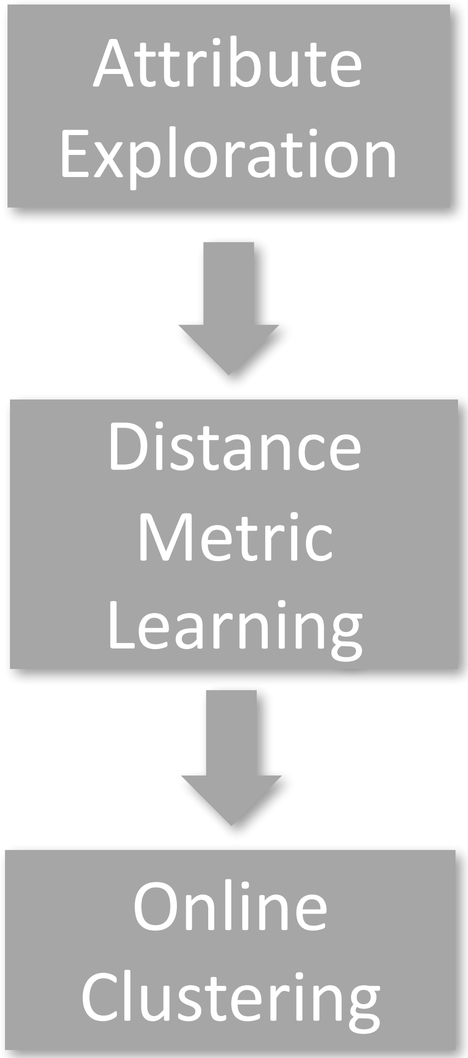
Application-Transparent Coflow Identification

Design Goals

- *Transparency: no modification to applications*
- *Accuracy: accurate for effective scheduling*
- *Speed: fast enough for timely scheduling*

3-step Learning Framework

- Attribute Exploration
  ---- *search for candidate attributes*
- Distance Calculation
  ---- *identify the importance of each attribute*
- Online Clustering
  ---- *group flows into coflows based on the distance metric*



Application-Transparent Coflow Identification

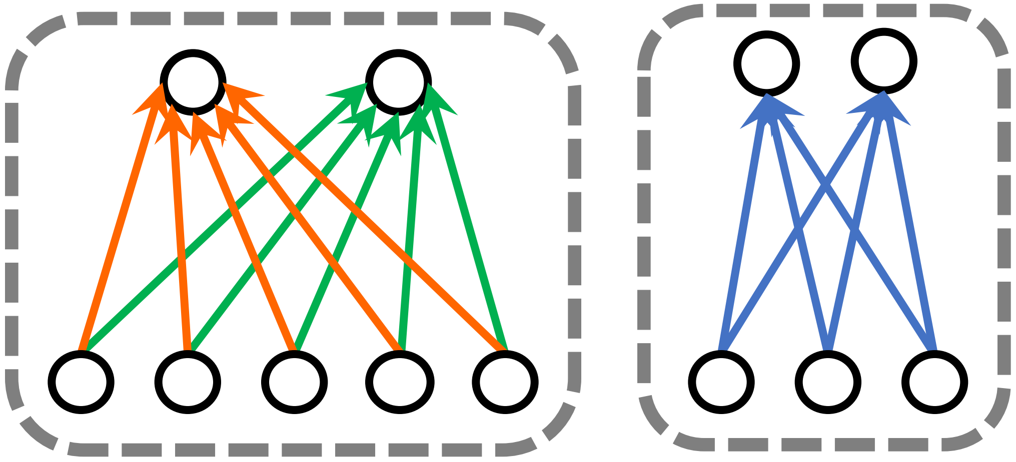
Step 1 --- Attribute Exploration

- Flow-level Attributes

Flow start time, inter-packet arrival time, …

Flow size …

- Community-Level Attributes
  - *Community Distance*
- Application-Level Attributes
  - *Port assignment of Spark*
  - *Port assignment of Hadoop*



Application-Transparent Coflow Identification

- **Step 2 --- Distance Calculation**
  - *Different attributes may have different importance*
  - *Thus need a good distance metric to reflect coflow relationships*

Input

- *Candidate Flow attributes*
- *Workloads with coflow information*

Distance Metric Learning →

Output: distance function

- *Small distances between flows within the same coflow;*
- *Large distances between flows belonging to different coflows;*

Flow arrival time and community attribute are most helpful

How Effective is CODA's Error-Tolerant Scheduling?

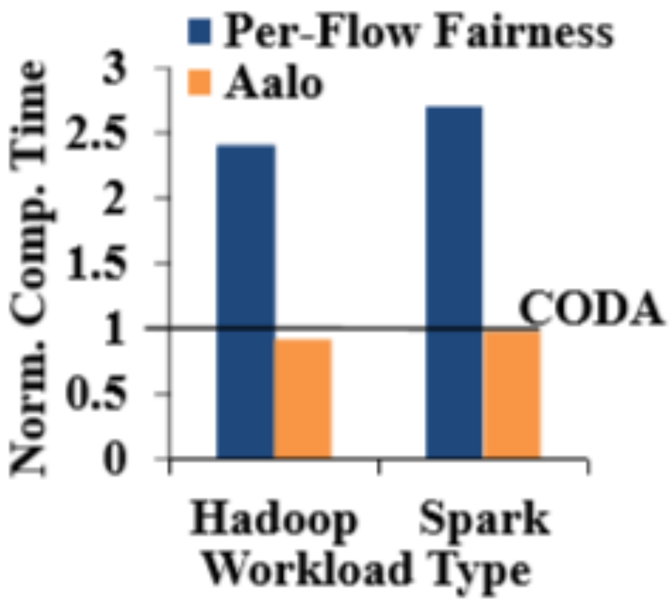
- *Benefit of Error-tolerant Scheduling*

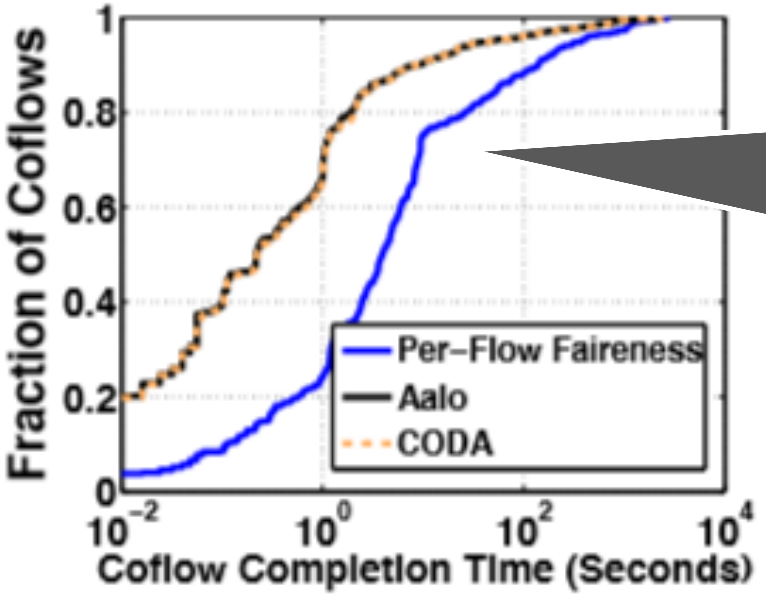
	Stretched arrival	*Batch arrival*
Per-flow	*2.03X*	*1.47X*
Aalo	*0.77X*	*0.56X*
CODA	*1X*	*1X*
CODA w.t. ET	***1.16X***	***1.04X***

Reduce the impact of error by 40%

How Effective is CODA's Error-Tolerant Scheduling?

- *Performance under Normal cases*





Almost as good as Coflow-aware solutions

Application-Transparent Coflow Identification

- **Caveat**
  - *Xxxx*
  - *Xxxx*
  - *Xxxx*

Application-Transparent Coflow Identification

- **Discussion**
  - *More than Spark/Hadoop*
  - *The need of a training step*
  - *Sensitivity to workload*

Error-tolerant ~~Coflow~~ Scheduling

- **Error-tolerant scheduling design --- A new problem beyond coflow**
  - Most of existing scheduling problems take ground-truth information as input, thus no need to consider possible input errors.
  - However, with the wide adoption of machine learning algorithms, many of the scheduling inputs are predictions/estimations based on learning results.
  - As a result, error-tolerant scheduling can be an interesting yet important research topic, which may greatly improve the scheduling performance with erroneous inputs in many different scenarios.

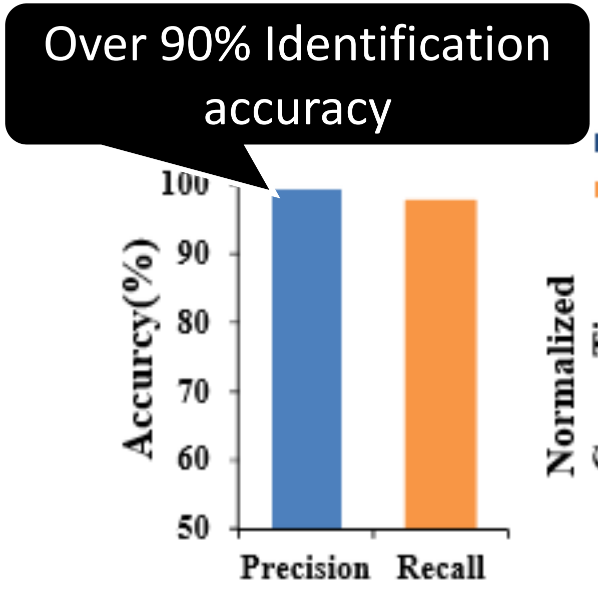
CODA

Toward Automatically Identifying and Scheduling COflows in the DArk

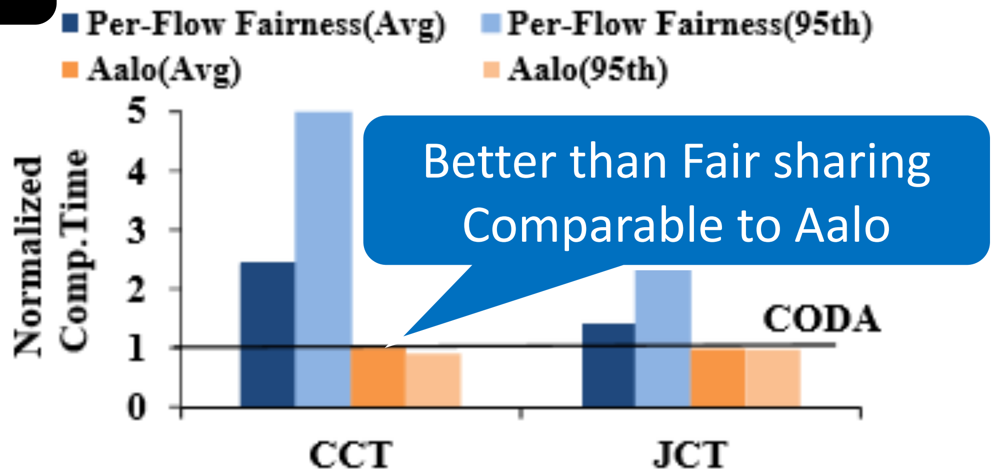
Hong Zhang, Li Chen, Bairen Yi, Kai Chen,
Mosharaf Chowdhury, Yanhui Geng

How does CODA perform in practice?

- **Can it approach xxx solutions?**

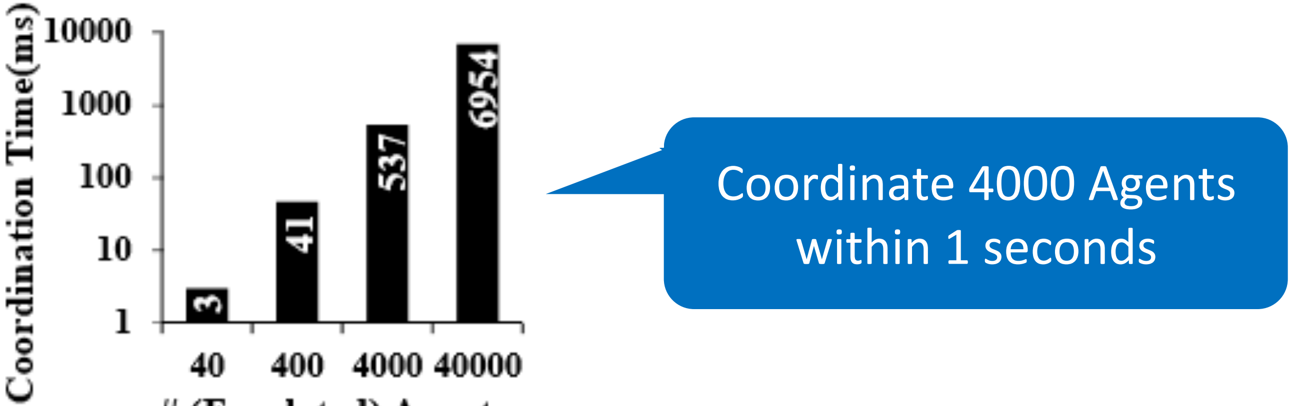


(a) Accuracy



(b) CCT and JCT

- **Can it scale gracefully?**



Application-Transparent Coflow Identification

- **Step 2 --- Distance Metric Learning**
  - Different attributes may have different importance
  - Thus need a good distance metric to reflect coflow relationships
    - Small distances between flows within the same coflow
    - Large distances between flows belonging to different coflows

tes

g

Formulation

- *Flow:*
- *Flow Distance:*

$$d(f_i, f_j) = ||f_i - f_j||_A = \sqrt{(f_i - f_j)^T A (f_i - f_j)}$$

$$\min_A \sum_{(f_i, f_j) \in S} ||f_i - f_j||_A^2$$

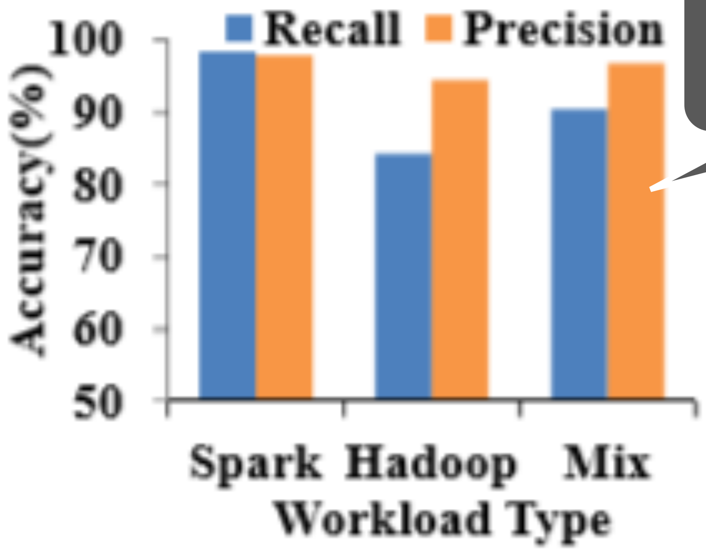
$$\text{s. t.} \sum_{(f_i, f_j) \in D} ||f_i - f_j||_A \geq 1, \quad A \succeq 0$$

Minimize the overall distance of flows within same coflows

Minimize the overall distance of flows within same coflows

How Effective is CODA's Identification?

- *How does CODA's identification perform overall?*



Over 90% accuracy

- *How does CODA's identification perform not work well?*

- *What is the speed up?*
  - *600X speed up with 2% accuracy loss*

How Effective is CODA's Error-Tolerant Scheduling?

- *Creating more challenging cases*
  - *Batch arrival*
  - *Stretch arrival*

→ *Around 40% accuracy loss*

- *CODA under more challenging cases*

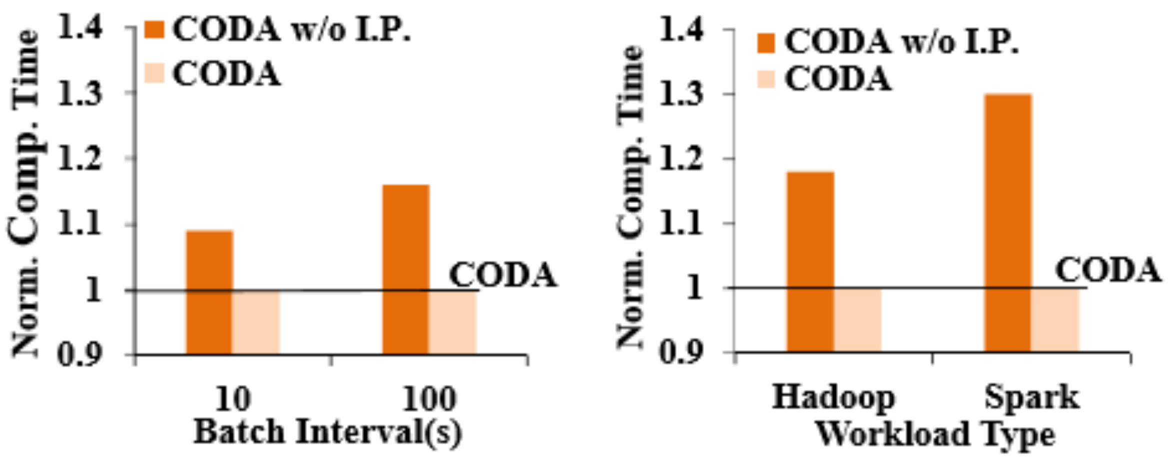
	Stretched arrival	*Batch arrival*
Per-flow	*2.03X*	*1.47X*
Aalo	*0.75X*	*0.56X*

How Effective is CODA's Error-Tolerant Scheduling?

- *Benefit of Late Binding*
  - *Improve average coflow completion time for up to 10%*

Reduce the impact of error by 30%

- *Benefit of Intra-Coflow Scheduling*
  - *Improve average coflow completion time for small flows for over 40%*



(a) Batch arrival case (Hadoop) (b) Stretched arrival case

Varys

Efficiently schedules coflows leveraging ***complete information***

- The size of each flow, the total number of flows

Not always achievable

Aalo

Efficiently schedules coflows ***without complete*** *information*

- ~~The size of each flow, and the total number of flows~~
- Which flow belongs to which coflow

Requires Application (e.g., Hadoop, Spark) Modification