Sincronia:
Near-Optimal Network Design for Coflows

Shijin Rajakrishnan

Joint work with

Saksham Agarwal  Akshay Narayan  Rachit Agarwal  David Shmoys  Amin Vahdat
The Flow Abstraction

**Traditional Applications:** Care about performance of individual flows

Good Match

Optimized for **Flow-level** performance
Is Flow Still the Right Abstraction?

Traditional Applications:
Care about performance of individual flows

Distributed Applications:
Care about performance for a group of flows

Mismatch

Optimized for Flow-level performance
The *Coflow* abstraction

*Collection of semantically related flows* [Chowdhury & Stoica, 2012]

 Allows applications to more precisely express their performance goals
Network and Coflow Model

- Big-switch model

- Clairvoyant scheduler
  - Coflow details known at arrival time:
    - Source-destination for each flow
    - Size of each flow
    - Coflow weight

- Metric – coflow completion time: Time when all flows complete

**Goal:** Minimize Average Weighted Coflow Completion Time (CCT)
Prior Results

**Impossibility Results**

- NP-hard
- <2x approximation hard

<table>
<thead>
<tr>
<th>Systems/ Theory</th>
<th>State-of-the-art</th>
<th>Performance Guarantees</th>
<th>Runs on Existing Transport</th>
<th>Work Conserving</th>
<th>Starvation Avoiding</th>
</tr>
</thead>
<tbody>
<tr>
<td>Systems</td>
<td>Varys [SIGCOMM ’14]</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Theory</td>
<td>On Scheduling Coflows [IPCO ’17]</td>
<td>✓ (4-apx)</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
</tbody>
</table>

Practical, Near-Optimal Network Design for Coflows?
Sincronia: Two key results

#1 Guarantees 4-approximation for (weighted) average CCT

#2 Given a set of coflows and a “right” ordering, ANY per-flow rate allocation mechanism that is work-conserving & order-preserving produces average CCT within 4x of optimal

- Per-flow rate allocation irrelevant
- Transport layer agnostic
## Sincronia – Near-Optimal Network Design

<table>
<thead>
<tr>
<th>Systems/Theory</th>
<th>Name</th>
<th>Performance Guarantees</th>
<th>Runs on Existing Transport</th>
<th>Work Conserving</th>
<th>Starvation Avoiding</th>
</tr>
</thead>
<tbody>
<tr>
<td>Systems</td>
<td>Varys</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Theory</td>
<td>On Scheduling Coflows</td>
<td>✓ (4-apx)</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Systems</td>
<td>Sincronia</td>
<td>✓ (4-apx)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

Also outperforms state-of-the-art across evaluated workloads
Sincronia Design

- Algorithm – BSSI
  - Bottleneck, Select, Scale, Iterate
  - SRPT-first style algorithm

- Priorities set from order
- Flows offloaded to transport layer
- No explicit per-flow rate allocation
Bottleneck-Select-Scale-Iterate (BSSI)

- Find **BOTTLENECK** port

- **SELECT** (weighted) largest job
  - Ordered last

- **SCALE** weights of remaining jobs

- **ITERATE** on unscheduled jobs

Ordering not important
BSSI in Action

- Bottleneck
- Select
  - Ordered Last
- Scale
- Iterate

Order:

Weights:

Scale weight from third flow (last scheduled to go out)
End-to-End Design (Offline)

- Each host knows ordering
- Flows get priority of coflow
- Offloads to priority enabled transport layer
Per-flow Rate Allocation is Irrelevant

• Intuition: Sharing bandwidth does not help CCT

• Order-preserving schedule:
  Flow blocked iff ingress or egress port serving higher-ordered flow

Given the BSSI ordering, ANY per-flow rate allocation mechanism that is work conserving & order-preserving produces average CCT within 4x of optimal
Avoiding per-flow rate allocation: Implications

• Implement on top of any transport layer
  ▪ E.g. pFabric, pHost, TCP

• Design and implementation independent of
  ▪ Network Topology
  ▪ Location of Congestion
  ▪ Paths of Coflows

• More scalable
  ▪ No reallocations upon coflow arrivals/departures

Details in paper
Handling Arbitrary Arrival Times

- Framework: Khuller, Li, Sturmfels, Sun, Venkat, ‘18
- Time divided into epochs
- In each epoch
  - Choose subset of unscheduled jobs
  - Schedule in next epoch using offline alg.

Provides 12-competitive performance (details in paper)
Evaluation Overview

• **Testbed implementation on top of TCP**
  - Evaluate impact of in-network congestion, and hardware constraints

• **Simulations**
  - Coflows arrive at time 0
  - Coflows arrive at arbitrary times
  - Sensitivity analysis
    - Coflow sizes, structure, # of coflows
    - Network topologies, Oversubscription ratios, Network load
    - ...

All simulations, workloads, and implementations are open-sourced on Sincronia website
Simulation Results

Offline

Sincronia not only provides near-optimal guarantees, but also improves upon state-of-the-art design in practice.
Even at such high network loads, Sincronia achieves CCT close to that of an unloaded network.
Implementation Results

Implemented on top of TCP

• 16-server Fat tree topology
  ▪ Full bisection bandwidth
  ▪ 20 PICA8 switches
    ➢ Supports 8 priority levels

• DiffServ for priority scheduling
Implementation Results

- Unfair Evaluation
  - TCP not designed for coflows
  - TCP not designed to minimize CT

+ Compare against existing designs
  - E.g. Varys reports 1.85x improvement at mean and at tails

Sincronia achieves significant improvements over existing network designs even with a small number of priority levels
Summary

- **Sincronia** – a network design for coflows
  - 4x within optimal
  - No per-flow rate allocation

<table>
<thead>
<tr>
<th>Name</th>
<th>Performance Guarantees</th>
<th>Run on existing Transport</th>
<th>Work Conserving</th>
<th>Starvation Avoiding</th>
</tr>
</thead>
<tbody>
<tr>
<td>Varys</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>On Scheduling Coflows</td>
<td>✓ (4-apx)</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Sincronia</td>
<td>✓ (4-apx)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

- Paper discusses number of open problems
Thanks!