CS434/534: Topics in Network Systems

CoFlow Scheduling;
Network Function Virtualization: Click

Yang (Richard) Yang
Computer Science Department
Yale University
208A Watson
Email: yry@cs.yale.edu

http://zoo.cs.yale.edu/classes/cs434/
Outline

- Admin and recap
- Cloud data center (CDC) applications/services
  - Fine-grained dataflow programming (e.g., Web apps)
  - Coarse-grained dataflow (e.g., data analytics)
  - Distributed machine learning using parameter server
  - DC cluster resource scheduling
  - DC transport scheduling
    - Overview
    - DCTCP
    - Fastpass
    - WAN scheduling
    - CoFlow scheduling

- Carrier cloud
  - overview
  - network function programming
    - click
Admin

- Instructor office hours
  - Tuesday: 1:30-2:30
  - Thursday: 3:00-4:00
  - Fridays: 1:30-2:30 pm

- Projects
  - Milestones (exactly 4 weeks left)
    - 4/17 (T+1 week): Finish reading major related work; a google doc listing related papers (at least 4 papers)
    - 4/24 (T+2 weeks): Finish architecture design (slides/write up of architecture, including all key components)
    - 5/1 (T+3 weeks): Initial, preliminary evaluations (slides/write up, about experiment/analysis setup)
    - 5/8 (T+4 weeks) 5:30 pm, final report due

- Remaining topics
Recap: WAN Scheduling

- Wide-area links are typically more expensive, less reliable and hence need better resource scheduling.
Recap: WAN Scheduling

- Google B4 and MS SWAN address quite common issues but give quite different designs
  - How to specify resource requirements?
    - Bandwidth share (B4), SWAN (priority)
  - How to allocate bandwidth, compute routes?
    - B4 progressive filling (maxmin fairness) and path generation
    - SWAN 15 paths, approximate max-min fairness
  - How to scale?
    - B4 (FG at site level -> super node level); SWAN approximate alg, pre-generate paths
  - How to update?
    - Both consider consistency, SWAN introduces a notion called scratch bandwidth
Recap: Solution Space

Flow: Transfer of data from a source to a destination in a data center

Per-Flow Fairness

Flow Completion Time

Where does B4/SWAN fit?
Outline

- Admin and recap
- Cloud data center (CDC) applications/services
  - Fine-grained dataflow programming (e.g., Web apps)
  - Coarse-grained dataflow (e.g., data analytics)
  - Distributed machine learning using parameter server
  - DC cluster resource scheduling
  - DC transport scheduling
    - Overview
    - DCTCP
    - Fastpass
    - WAN scheduling
    - CoFlow scheduling
Data Parallel Applications

Structure:
Multi-stage dataflow
  • Computation interleaved with communication

Computation Stage (e.g., Map, Reduce)
  • Distributed across many machines
  • Tasks run in parallel

Communication Stage (e.g., Shuffle)
  • Between successive computation stages

A communication stage cannot complete until all the data have been transferred
Per-flow vs Co-flow Scheduling

Coflow 1

Fair Sharing

Smallest-Flow First¹,²

The Optimal

Coflow 1 comp. time = 5
Coflow 2 comp. time = 6

Coflow 1 comp. time = 5
Coflow 2 comp. time = 6

Coflow 1 comp. time = 3
Coflow 2 comp. time = 6

Example CoFlows

Q: What is a co-flow?
CoFlow API

<table>
<thead>
<tr>
<th>VarysClient Methods</th>
<th>Caller</th>
</tr>
</thead>
<tbody>
<tr>
<td>register(numFlows, [options]) (\rightarrow) coflowId</td>
<td>Driver</td>
</tr>
<tr>
<td>put(coflowId, dataId, content, [options])</td>
<td>Sender</td>
</tr>
<tr>
<td>get(coflowId, dataId) (\rightarrow) content</td>
<td>Receiver</td>
</tr>
<tr>
<td>unregister(coflowId)</td>
<td>Driver</td>
</tr>
</tbody>
</table>

**Table 2: The Coflow API**

```scala
val cId = client.register(6)

// Read from DFS, run user-written map method, // and write intermediate data to disk. // Now, invoke the coflow API.
for (r <- reducers)
  client.put(cId, dId-m-r, content-m-r)

// Shuffle using the coflow API.
for (m <- mappers)
  content-m-r = client.get(cId, dId-m-r)
// Now, sort, combine, and write to DFS.
client.unregister(cId)
```
Centralized master-slave architecture

- Applications use a client library to communicate with the master

**Coflow transport time** and **rates are determined by the coflow scheduler**
How may you compute the rate allocation to schedule a coflow?
A two step algorithm

- **Ordering Heuristic:** Typically scheduling algorithms are based on ordering, i.e., considering the jobs in some order (called permutation scheduling)
  - Discussion: what are potential metrics to order the coflows?

- **Rate allocation algorithm:** Allocates minimum required resources to each coflow to finish in minimum time
Varys Ordering Metric: Bottleneck

- Assume bottleneck only at the (ingress or egress) edge
- Assume $d_{ij}$ for amount of data from $i$ to $j$
- An estimation of finishing time for the coflow:

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

- Given estimation of finishing time for a coflow:

\[
\Gamma = \max \left( \max_i \frac{\sum_j d_{ij}}{\text{Rem}(P_{i}^{\text{in}})}, \max_j \frac{\sum_i d_{ij}}{\text{Rem}(P_{j}^{\text{out}})} \right)
\]

- How much to allocate rate to each member flow \((d_{ij})\) of the coflow?
Algorithm 1 Coflow Scheduling to Minimize CCT

1: function ALLOC_BANDWIDTH(Coflows C, Rem(.), Bool cct)
2:     for all $C \in C$ do
3:         $\tau = \Gamma^C$ (Calculated using Equation (1))
4:         if not cct then
5:             $\tau = D^C$
6:         end if
7:         for all $d_{i,j} \in C$ do
8:             $r_{i,j} = d_{i,j}/\tau$  
9:             Update Rem($P^\text{in}_i$) and Rem($P^\text{out}_j$)
10:        end for
11:     end for
12: end function

13: function MIN_CCT_OFFLINE(Coflows C, C, Rem(.))
14:     $C' = \text{SORT_ASC}(C \cup C')$ using SEBF
15:     allocBandwidth($C'$, Rem(.), true)
16:     Distribute unused bandwidth to $C \in C'$  \text{ Work conserv. (§5.3.4)}
17:     return $C'$
18: end function

19: function MIN_CCT_ONLINE(Coflows C, C, Rem(.))
20:     if timeSinceLastDelta() < $T$ then  \text{T-interval: Decrease CCT}
21:         $C' = \text{minCCTOffline}(C, C, Rem(.))$
22:         Update $C_{\text{zero}}$, the set of starved coflows
23:     else  \text{ $\delta$-interval: Starvation freedom}
24:         $C^* = \bigcup C$ for all $C \in C_{\text{zero}}$
25:         Apply MADD on $C^*$
26:         Schedule a call to minCCTOnline(.) after $\delta$ interval
27:     end if
28: end function
Discussion

- What do you take away from the Varys design?

- What are issues of Varys?
Key Issues of Varys

- Assumes
  - Size of each flow is known
    - But pipelining may change the size
  - The total number of flows is known
    - But real system conducts speculative execution
  - The endpoints are known
    - But failure recovery may change endpoints

<table>
<thead>
<tr>
<th>Link 1</th>
<th>Link 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 Units</td>
<td>6 Units</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Coflow 1</th>
<th>Coflow 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 Units</td>
<td>2 Units</td>
</tr>
</tbody>
</table>
Outline

- Admin and recap
- **Cloud data center (CDC) applications/services**
  - Fine-grained dataflow programming (e.g., Web apps)
  - *Coarse*-grained dataflow (e.g., data analytics)
  - Distributed machine learning using parameter server
  - DC cluster resource scheduling
  - DC transport scheduling
    - Overview
    - DCTCP
    - Fastpass
    - WAN scheduling
    - CoFlow scheduling
      - Varys
      - Aalo
Aalo Goal

- Dynamic scheduling of coflows according to their current states
  - an online framework
  - non-blocking API
    - no longer wait for all flows’ info to be registered to start
Aalo Design I: Least-Attained Service (LAS) Ordering

- Prioritize coflow that has sent the least total number of bytes
  - The more a coflow has sent, the lower its priority
    => Smaller coflows finish faster

- Problems:
  - can lead to starvation
  - suboptimal for similar-size coflows
Suboptimal for Similar Coflows

Reduces to fair sharing
• Doesn't minimize average completion time

FIFO works well for similar coflows
• Optimal when coflows are identical
Between a “Rock” and a “Hard Place”

Prioritize across dissimilar (elephant and mice) coflows

FIFO schedule similar coflows (not ping-pong among them)
Aalo Idea: Discretized Coflow-Aware LAS (D-CLAS)

Priority discretization
- Change priority when total # of bytes sent exceeds predefined thresholds

Scheduling policies
- FIFO within the same queue
- Prioritization across queue

Weighted sharing across queues
- Guarantees starvation avoidance
Aalo Discretize Priorities

Exponentially spaced thresholds: $A \times E^i$
- $A, E$: constants
- $1 \leq i \leq K$: threshold constant
- $K$: number of the queues

Q: Which queue is a coflow when it just first starts?
Remaining Issue: Computing **Total** # of Bytes Sent by a CoFlow

- **Why an issue?**
  - D-CLAS requires to know total # of bytes sent over all flows of a coflow, but such distributed aggregation can be challenging over small time scales
  - D-LAS has worse performance

**Diagram:**
- **Coflow 1**
  - Link 1: 3 Units
  - Link 2: 6 Units
- **Coflow 2**
  - Link 1: 2 Units
  - Link 2: 6 Units

**Timing:**
- Coflow 1 comp. time = 6
- Coflow 2 comp. time = 6
- Coflow 1 comp. time = 3
- Coflow 2 comp. time = 6
Aalo Architecture
**Varys vs Aalo**

- **Similar for large coflows** because they are in slow-moving queues.
- **Performance loss for medium coflows** by mischeduling them.
- **Improvements for small coflows**
Offline Reading: Sincronia
Summary: Cloud Datacenter Programming and Resource Scheduling

Key issues

- Programming models
  - Acceleration/performance scaling techniques
- Control architecture
  - Resource isolation (scheduling) mechanisms
  - Security isolation
Summary: DC Programming Models

- **Noria**

```sql
/* base tables */
CREATE TABLE stories
(id int, author int, title text, url text);
CREATE TABLE votes (user int, story_id int);
CREATE TABLE users (id int, username text);
/* internal view: vote count per story */
CREATE INTERNAL VIEW VoteCount AS
SELECT story_id, COUNT(*) AS vcount
FROM votes GROUP BY story_id;
/* external view: story details */
CREATE VIEW StoriesWithVC AS
SELECT id, author, title, url, vcount
FROM stories
JOIN VoteCount ON VoteCount.story_id = stories.id
WHERE stories.id = ?;
```

- **MapReduce**

```java
Map(input_key, input_value) {
    foreach word w in input_value:
        emit(w, 1);
}

Reduce(output_key, intermediate_vals) {
    set count = 0;
    foreach v in intermediate_vals:
        count += v;
    emit(output_key, count);
}
```
Summary: DC Programming Models

- **Spark**

- **Parameter server**
Summary: Acceleration Techniques

- Local scheduling (MapReduce)
- Working set in the memory (Spark)
- Pipelining (spark)
- Caching (Noria)
- ...

36
Summary: DC Orchestration/Control Architecture

Two-level

Resource Mgr

S1  S2  S3

Shared State
Summary: DC Resource Scheduling Mechanisms

- **Cluster**
  - Delay scheduling
  - Spark stage based scheduling

- **Fairness**
  - Max-min fairness
  - Hadoop hierarchical scheduling
  - Dominant resource fairness (DRF)
  - Coflow scheduling

- **Transport scheduling**
  - DCTCP
  - Fastpass
  - WAN scheduling
  - Coflow scheduling
Summary: DC Security Isolation

- Not really covered, assuming VMs and/or containers
Outline

- Admin and recap
- Cloud data center (CDC)
- Carrier cloud
**Major Trend**

- Programming and managing carrier network infrastructures (CN) in a similar way to programming and managing DC
  - Convert from expensive, hardware-centric CN architecture to software-centric CN architecture
    - Functions deployed called network functions
  - May have a major impact on existing carrier networks
  - Essential to 5G network architecture
5G Network Architecture

Contents

1. A Cloud-Native 5G Architecture is Key to Enabling Diversified Service Requirements
   04

2. 5G Will Enrich the Telecommunication Ecosystem
   05
   2.1 The Driving Force Behind Network Architecture Transformation
   2.2 The Service-Driven 5G Architecture

3. End-to-End Network Slicing for Multiple Industries Based on One Physical Infrastructure
   08

4. Reconstructing the RAN with Cloud
   09
   3.1 Multi-Connectivity Is Key to High Speed and Reliability
   3.2 MCE

5. Cloud-Native New Core Architecture
   13
   4.1 Control and User Plane Separation Simplifies the Core Network
   4.2 Flexible Network Components Satisfy Various Service Requirements
   4.3 Unified Database Management

6. Self-Service Agile Operation
   17

7. Conclusion:
   19
   Cloud-Native Architecture is the Foundation of 5G Innovation
**Bigger Picture**

Cellular access

Residential access

Campus access, e.g., Ethernet WiFi

ISP

Backbone ISP

ISP

Data center
Discussion

- What functions/apps/services may run in carrier clouds?
Example: Cellular Architecture

- **MME/HSS**: authentication, mobility management, ...
- **PCRF**: charging instruction, QoS info, ...
- **SGW, PGW**: standard network functions, NAT, QoS, policing, firewall, content cache, parent control, transcoding, ...
Discussion

- What are some major differences between cloud data center and carrier network cloud?
Discussion: Carrier Cloud
Programming and Resource Scheduling

What may the following look like in carrier cloud?

- Programming models
  - Acceleration/performance scaling techniques
- Control architecture
  - Resource isolation (scheduling) mechanisms
  - Security isolation
Outline

- Admin and recap
- Cloud data center (CDC) applications/services
- Carrier cloud
  - overview
  - network function programming
    - click
Click

- One of the first major, modular network programming models

The Click modular router
E Kohler, R Morris, B Chen, J Jannotti… - ACM Transactions on ..., 2000 - dl.acm.org
Clicks is a new software architecture for building flexible and configurable routers. A Click router is assembled from packet processing modules called elements. Individual elements implement simple router functions like packet classification, queuing, scheduling, and …
☆ abella Cited by 2971  Related articles  All 57 versions  Web of Science: 823  ❖

The Click modular router
R Morris, E Kohler, J Jannotti… - ACM SIGOPS Operating ..., 1999 - dl.acm.org
Click is a new software architecture for building flexible and configurable routers. A Click router is assembled from packet processing modules called elements. Individual elements implement simple router functions like packet classification, queuing, scheduling, and …
☆ abella Cited by 512  Related articles  All 39 versions

- Highly influential for later designs
Click Design Goals

- **Focus on single device**
- **Flexibility**
  - easy to add new features
  - enable experimentation
- **Openness**
  - allow users/researchers to build and extend (In contrast to most commercial routers)
- **Modularity**
  - Simplify the reuse and composition of existing features
- **Speed/efficiency**
  - Run in OS
Click Programming Structure: A Graph of Network Elements

- Large number of small elements
  - Each performing a simple packet function
  - E.g., IP look-up, TTL decrement, buffering

- Connected together in a directed graph
  - Elements inputs/outputs snapped together
  - Packet flow through a graph as main organizational primitive
  - Construct different graphs using the same element as the main reusability primitive
Click Elements and Graph Specification

// Declare three elements...
src :: FromDevice(eth0);
ctr :: Counter;
sink :: Discard;
// ...and connect them together
src -> ctr;
ctr -> sink;

// Alternate definition using syntactic sugar
FromDevice(eth0) -> Counter -> Discard;

input port -----> Tee(2) -----> output ports

configuration string

FromDevice(eth0) -----> Counter -----> Discard
Exercise

- What are possible ways for two linked elements to interact (handoff of packets)?
Click Elements Interaction

Two modes

- Push processing
  - Initiated by the source end
    - E.g., when an unsolicited packet arrives (e.g., from an interface card)
  - Often push to the output card, or to a queue

- Pull processing
  - Initiated by the destination end
  - E.g., to control timing of packet processing (e.g., based on a timer or packet scheduler)
Push and Pull

- **Push connection**
  - Source pushes packets downstream
  - Triggered by event, such as packet arrival
  - Denoted by filled square or triangle

- **Pull connection**
  - Destination pulls packets from upstream
  - Packet transmission or scheduling
  - Denoted by empty square or triangle

- **Agnostic connection**
  - Becomes push or pull depending on peer
  - Denoted by double outline
Element Abstraction

- Instances of C++ classes

```cpp
class NullElement: public Element { public:
    NullElement() { add_input(); add_output(); }
    const char *class_name() const { return "Null"; }
    NullElement *clone() const { return new NullElement; }
    const char *processing() const { return AGNOSTIC; }
    void push(int port, Packet *p) { output(0).push(p); }
    Packet *pull(int port) { return input(0).pull(); }
};
```

- Ex: How to define new types of elements?
An IP Router
Discussion

- What you may take away from Click design?

- What features are missing/undesired for global carrier network cloud programming?