Datacenter Resource Scheduling:
Transport Scheduling

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

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

Acknowledgement: slides contain content from presentations by authors of DC TCP and fastpass.
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
Admin

- Project meetings
  - Tuesday: 1:30-2:30
  - Thursday: 3:00-4:00
Recap: DC Cluster Scheduling

- Problem: fine-grained sharing of the same cluster by multiple jobs

- Requirements focused on
  - Extensibility
  - Scalability
  - Fairness/isolation
Recap: YARN Architecture

Application Layer

Resource Layer

Client

Resource Manager

Node Manager

App Mstr

Container

Client

MapReduce Status

Job Submission

Node Status

Resource Request
Recap: Mesos Architecture

- **MPI job**
  - **MPI scheduler**
  - **Mesos master**
- **Hadoop job**
  - **Hadoop scheduler**
  - **Resource offer**
  - **Allocating**
- **Mesos slave**
  - **MPI executor**
  - **Hadoop executor**
  - **Task**

- **Framework-specific scheduling**
- **Pick framework to offer resources to**
- **Launches and isolates executors**
Recap: Omega Architecture

- Multiple controllers, shared state
  - Each controller can independently read/write shared state
Recap: Summary of Architectures

Question to think about: other potential architectures?
Recap: Fairness/Isolation Properties

- Efficiency of dynamic allocation depends on that some users do not use up all of their fair-share resources

- Max-min fairness, realized in various forms
  - Progressive filling, e.g.,
    - 1, 2, 3, 4, 8; total capacity 15
  - Virtual clock with equal share, e.g.,
    - 5 queues, with arrival rates 1, 2, 3, 4, 8/sec; system can process 15 requests/sec; compute the departure time of each arrival using its virtual share
      \[ D_i(n+1) = \max(A_i(n+1), D_i(n)) + L_i(n+1)/\text{share} \]
  - Simple longest not served, e.g.,
    - 4 queues, with arrival rates 1, 2, 3, 4, 8/sec; system can process 15 requests/sec;
    - Each queue has a timestamp of last served time; pick the longest not served
Recap: Fairness/Isolation Properties

- **Dominant resource fair**

**Algorithm 1** DRF pseudo-code

\[
R = \{r_1, \ldots, r_m\} \quad \triangleright \text{total resource capacities}
\]

\[
C = \{c_1, \ldots, c_m\} \quad \triangleright \text{consumed resources, initially 0}
\]

\[
s_i \quad (i = 1..n) \quad \triangleright \text{user } i\text{'s dominant shares, initially 0}
\]

\[
U_i = \{u_{i,1}, \ldots, u_{i,m}\} \quad (i = 1..n) \quad \triangleright \text{resources given to user } i, \text{initially 0}
\]

1. **pick** user \(i\) with lowest dominant share \(s_i\)
2. \(D_i \leftarrow \text{demand of user } i\text{'s next task}\)
3. **if** \(C + D_i \leq R\) **then**
   - \(C = C + D_i\) \quad \triangleright \text{update consumed vector}
   - \(U_i = U_i + D_i\) \quad \triangleright \text{update } i\text{'s allocation vector}
   - \[s_i = \max_{j=1}^{m} \{u_{i,j}/r_j\}\]
4. **else**
   - **return** \(\triangleright \text{the cluster is full}\)
5. **end if**

- Some tasks are CPU-intensive
- Some tasks are memory-intensive
- Most tasks need \(<2 \text{ CPU, 2 GB RAM}>\)

2000-node Hadoop Cluster at Facebook (Oct 2010)
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
      - basic challenges
Discussion

- What are potential problems and requirements in DC transport scheduling?
INTERNET

Fabric

Diverse workload sharing the same transport infrastructure

web  app  cache  database  map-reduce  HPC  monitoring
Different Workload w/ Different Requirements

- Short messages [50KB-1MB] (e.g., coordination, short query, control state) 
- Large flows [1MB-50MB] (Data update)
Example Delay-Sensitive Framework: Partition and Aggregation

- The foundation for many large-scale web applications: Web search, Social network composition, Ad selection, etc.

- Tree-like structure
  - Root node sends query
  - Leaf nodes respond with data
  - Parent aggregates results

- Deadline budget split among nodes
  - Missed deadlines
    - incomplete responses
1. Art is a lie...
2. The chief...
3. ...

• Strict deadlines (SLAs)

• Lower quality result if missed deadlines

Deadline = 250ms

Deadline = 50ms

Deadline = 10ms
Challenging Traffic: Incast

Worker 1

Worker 2

Worker 3

Worker 4

• Synchronized mice collide.
  ➢ Caused by Partition/Aggregate.

Aggregator

RTO_{min} = 300 ms

TCP timeout
A Real Incast Event

Figure 7: A real incast event measured in a production environment. Timeline shows queries forwarded over 0.8ms, with all but one response returning over 12.4ms. That response is lost, and is retransmitted after $RTO_{\text{min}}$ (300ms). RTT+Queue estimates queue length on the port to the aggregator.
Incast: Cluster-based Storage Systems

Synchronized Read

Client

Switch

Server Request Unit (SRU)

Data Block

Client now sends next batch of requests
Q: How does TCP maintain high throughput despite cutting rate to half?

Q: What is the “cost” of TCP achieving high throughput?
Queue Buildup

- Flows targeting high put buildup queues for rate match.
  - Increased latency for short flows.
RTT affects soft-real-time apps

Page Load Time As RTT Decreases

R. Peon & W. Chan, SPDY Essentials, Google Tech Talk, 12/8/11
Summary: DC Transport Challenges

- **Challenging setting:**
  - Mixed elephants and mice traffic with potential high burst traffic

- **Challenging requirement:**
  - Achieving high throughput for elephants and low latency for mice, despite potential high burst
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
      - challenges
      - solution space
Solution Space

Controller implements transport scheduling

Switches implement transport scheduling

Hosts implement transport scheduling
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
    - DC TCP
Recap Current Switch Mechanism: Network Feedback (ECN: Explicit Congestion Notification)

Sender 1

Sender reduces rate if ECN received.

Receiver

Receiver bounces marker back to sender in ACK msg

Sender 2

Network marks ECN Mark (1 bit) on pkt according to local condition, e.g., queue length > K
Recap Current Host Mechanism: TCP

- AIMD to respond (in each RTT) to network feedback (ECN), according to one of two states:
  - cong: cut rate to half
  - ! cong: increase rate by 1

- Cutting to 1/2 is drastic, and hence need large buffer to absorb the large oscillation.
Q: Is it possible to obtain more information to more fine-grained adjust?
DC TCP Basic Idea

- Use ECN and extract multi-bit feedback from single-bit stream of ECN marks.
  - Reduce window size based on fraction of marked packets.

<table>
<thead>
<tr>
<th>ECN Marks</th>
<th>TCP</th>
<th>DCTCP</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 0 1 1 1 1 0 1 1 1</td>
<td>Cut window by 50%</td>
<td>Cut window by 40%</td>
</tr>
<tr>
<td>0 0 0 0 0 0 0 0 0 1</td>
<td>Cut window by 50%</td>
<td>Cut window by 5%</td>
</tr>
</tbody>
</table>

**Graphs:**
- **TCP:** Variations in window size (Bytes) over time (sec).
- **DCTCP:** Smoother window size (Bytes) over time (sec).
DC TCP Algorithm

**Switch side:**
- Mark packets when Queue Length > K.

**Sender side:**
- Maintain running average of *fraction* of packets marked ($\alpha$).

\[
each \text{RTT} : F = \frac{\text{# of marked ACKs}}{\text{Total # of ACKs}} \implies \alpha \leftarrow (1 - g)\alpha + gF
\]

- Adaptive window decreases: $W \leftarrow (1 - \frac{\alpha}{2})W$
**DCTCP vs TCP**

**Experiment:** 2 flows (Win 7 stack), Broadcom 1Gbps Switch

DCTCP mitigates Incast by creating a large buffer headroom

Queue Length (Packets)

Buffer is mostly empty

Time (seconds)
Intuition: Analysis

- N infinitely long-lived, synchronized flows
- Flows with identical round-trip times RTT
- Flows share a single bottleneck link with capacity C.
- Assume that

\[ Q(t) = NW(t) - C \times RTT \]
Exercise

Let $S(W_1, W_2)$ be the packets sent by one sender, when its window grows from $W_1$ to $W_2$.

$$S(W_1, W_2) = \frac{(W_2^2 - W_1^2)}{2}$$
Intuition: Analysis

\[ Q(t) = NW(t) - C \times RTT \]

\[ W^* = \frac{(C \times RTT + K)}{N} \]

\[ Q_{\text{max}} = N(W^* + 1) - C \times RTT \]

\[ = N + K \]

Figure 11: Window size of a single DCTCP sender, and the queue size process.
**Intuition: Analysis**

\[ S(W_1, W_2) = (W_2^2 - W_1^2)/2 \]

---

**Figure 11:** Window size of a single DCTCP sender, and the queue size process.

Consider \( W^* = (C \times RTT + K)/N \)

**Loss \( \alpha \):**

\[
\alpha = \frac{S(W^*, W^*+1)}{S((W^*+1)(1-\alpha/2), W^*+1)}
\]

\[
\alpha = \frac{2W^*+1}{(W^*+1)^2(\alpha-\alpha^2/4)}
\]

\[
\alpha^2(1-\alpha/4) = \frac{2W^*+1}{(W^*+1)^2} \approx \frac{2}{W^*}
\]

\[
\alpha \approx \sqrt{2/W^*}
\]
**Intuition: Analysis**

Figure 11: Window size of a single DCTCP sender, and the queue size process.

\[
S(W_1, W_2) = \frac{(W_2^2 - W_1^2)}{2} \\
W^* = \frac{(C \times RTT + K)}{N} \\
\alpha \approx \frac{\sqrt{2}}{W^*}
\]

\[
D = (W^* + 1) - (W^* + 1)(1 - \frac{\alpha}{2})
\]

\[
A = ND = N(W^* + 1) \frac{\alpha}{2} \approx \frac{N}{2} \sqrt{2W^*} = \frac{1}{2} \sqrt{2N(C \times RTT + K)}
\]
Intuition: Analysis

\[ Q(t) = NW(t) - C \times RTT \]

\[ S(W_1, W_2) = \frac{(W_2^2 - W_1^2)}{2} \]

\[ W^* = \frac{(C \times RTT + K)}{N} \]

\[ \alpha \approx \frac{\sqrt{2}}{W^*} \]

\[ A = \frac{1}{2} \sqrt{2N(C \times RTT + K)} \]

\[ Q_{\text{min}} = Q_{\text{max}} - A \]

\[ = K + N - \frac{1}{2} \sqrt{2N(C \times RTT + K)} \]

\[ K > \frac{(C \times RTT)}{7} \]
Why it Works

- **Low Latency**
  - Small buffer occupancies $\rightarrow$ low queuing delay

- **High Throughput**
  - Keep queue non empty to fully utilize

- **High Burst Tolerance**
  - Large buffer headroom $\rightarrow$ bursts fit
  - Aggressive marking $\rightarrow$ sources react before packets are dropped
Discussion

- Limitations of DCTCP
Other Example Network Mechanism
Supporting Low Latency and High Tput

Priority-based flow control (PFC)

- Hop-by-hop flow control, with eight priorities for HOL blocking mitigation
- The priority in data packets is carried in the VLAN tag
- PFC pause frame to inform the upstream to stop

Component for RDMA
“Congestion Spreading” in PFC
**Ideal Datacenter Transport Properties**

Scaling Memcache at Facebook, Fine-grained TCP retransmissions

Burst Control

Datacenter

TDMA, Tail at scale, pFabric, PDQ, DCTCP, D3, Orchestra

Low Tail Latency

EyeQ, Seawall, Oktopus, Hedera, VL2, Mordia, SWAN, MATE, DARD

Multiple Objectives
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
Fastpass Goals

Design a network that provides

1. Zero network queues
2. High Utilization
3. Multiple app and user objectives

Approach: Centralized arbiter schedules and assigns paths to all packets
Discussion

- What may such a centralized system look like?

- What are key concerns of a central arbiter?
When an application calls send() or sendto() on a socket, the operating system sends this demand in a request message to the Fastpass arbiter, specifying the destination and the number of bytes.

The controller computes schedule and path.
Example: Packet from A to B

5µs  A → Arbiter  "A has 1 packet for B"
1-20µs  Arbiter  timeslot allocation & path selection
15µs  Arbiter → A  "@t=107: A → B through R1"
no queuing  A → B  sends data
**Fastpass Scheduling and Selecting Paths**

- Assume rearrangeably non blocking (RNB) topology, and hence can decompose the problem into 2 steps:

  **Step 1: Timeslot Allocation**
  Choose a matching (how?)

  **Step 2: Path selection**
  Map matching onto paths
Timeslot Allocation for Max-min Fairness

- active flows
  - src: 1 → dst: 1
  - src: 2 → dst: 2
  - src: 3 → dst: 3
  - src: 4 → dst: 4

- src → dst:
  - 1 → 3
  - 4 → 2
  - 2 → 3
  - 2 → 1
  - 3 → 4
Timeslot Allocation for Max-min Fairness

<table>
<thead>
<tr>
<th>active flows</th>
<th>src</th>
<th>dst</th>
<th>last allocation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>45</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>2</td>
<td>47</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>3</td>
<td>48</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>1</td>
<td>48</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>4</td>
<td>51</td>
</tr>
</tbody>
</table>

allocated srcs & dsts

- Circled in purple: ✓
- Circled in red: ✗
- Circled in blue: ✓
- Circled in orange: ✓
- Circled in green: ✓
Exercise: Timeslot Allocation for Minimizing Flow Completion Time (FCT)

```
active flows

<table>
<thead>
<tr>
<th>src</th>
<th>dst</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>src</th>
<th>dst</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>
```
Scaling

2211.8 Gbits/s on 8 cores
Path Selection
Path Selection

Figure 5: Path selection. (a) input matching (b) ToR graph (c) edge-colored ToR graph (d) edge-colored matching.
Fastpass Base Performance

Queue length (KB)

Time (seconds)

4351 KB

18 KB

Density

Ping time (milliseconds)

.23 ms

3.56 ms

TCP ping

fastpass

baseline
Convergence to network share

![Graph showing per-connection throughput (Gbits/s) over time for different senders.](image)

- Sender 1
- Sender 2
- Sender 3
- Sender 4
- Sender 5

**Per-connection throughput (Gbits/s)**

**Time (seconds)**

- Baseline
- Fastpass

5200x std dev

5200x standard deviation
Discussion

- What do you like about fastpass?

- What are issues of fastpass?
Solution Space

**Flow:** Transfer of data from a source to a destination

Independent flows cannot capture the collective communication behavior common in data-parallel applications.