Is advance knowledge of flow sizes a plausible assumption?

Vojislav Dukic, Sangeetha Abdu Jyothi, Bojan Karlas, Muhsen Owaida, Ce Zhang, Ankit Singla
Flow size information can increase efficiency

Performance

FS *not known* in advance  FS *known* in advance
Flow size information can increase efficiency

...fine-grained circuit scheduling

...packet scheduling within switches

...deadline-aware prioritization

...adaptive routing

...congestion control schemes
Packet scheduling at a switch queue

Typically, first-in first-out
Packet scheduling at a switch queue

Typically, first-in first-out
Packet scheduling at a switch queue

But could apply shortest job first idea!
Packet scheduling at a switch queue

Least remaining bytes first

100 MB

100 KB

[*pFabric: Minimal Near-Optimal Datacenter Transport; SIGCOMM '13]*
Packet scheduling at a switch queue

100 MB

100 KB

Least remaining bytes first

[*pFabric: Minimal Near-Optimal Datacenter Transport; SIGCOMM ’13*]
Packet scheduling at a switch queue

Simple, but problematic: do we know the shortest flow?
Assumption:
Flow size is know in advance
“In many datacenter applications flow sizes or deadlines are known at initiation time and can be conveyed to the network stack (e.g., through a socket API)...”

“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 sender must specify the size of a message when presenting its first byte to the transport... Knowledge of message sizes is particularly valuable because it allows transports to prioritize shorter messages.”

**pFabric** – SIGCOMM ’13
Improves FCT by **4x**

**FastPass** – SIGCOMM ’14
Improves FCT by **15x**

**Homa** – SIGCOMM ’18
Reduces tail latency by **100x**
Assumption:
Flow size is known in advance
Assumption: Flow size is not known in advance
In this paper, we question the validity of this assumption, and point out that, for many applications, such information is difficult to obtain, and may even be unavailable.

“...A number of applications are unable to provide size/deadline information at the start of their flows, e.g. database access and HTTP chunked transfer.”

“We ignore flow size and duration as they cannot be acquired until a flow finishes.”

PIAS – NSDI ‘15

Karuna – SIGCOMM ‘16

CODA – SIGCOMM ‘16
Example alternative: Flow aging

Packets sent = packets remaining
Example alternative: Flow aging

100 MB

100 KB

5 4

Packets sent = packets remaining
Possible alternative: flow aging

Works well for long tail flow size distributions
Possible alternative: flow aging

Works well for long tail flow size distributions

Works only when relative flow size is needed

Doesn't work for equally sized flows
Is advance knowledge of flow sizes a plausible assumption?
Is advance knowledge of flow sizes a plausible assumption?

100% knowledge is likely intractable*

* or at least too expensive
Is advance knowledge of flow sizes a plausible assumption?

100% knowledge is likely intractable*

* or at least too expensive

... but partial knowledge is plausible and useful
Sources of flow size information?

How useful is imprecise / partial knowledge?

Incentives for operators and users?
Sources of flow size information?

How useful is imprecise / partial knowledge?

Incentives for operators and users?
Flow size estimation: design space

Exact sizes given by application

TCP buffer occupancy

Monitoring system calls

Learning from past traces

Many apps know flow sizes

Doable in private DCs
Flow size estimation: design space

Exact sizes given by application

TCP buffer occupancy

Monitoring system calls

Learning from past traces

Many apps know flow sizes

Doable in private DCs

Some apps don’t

Change a lot of applications

Change network API
Flow size estimation: design space

Exact sizes given by application

TCP buffer occupancy

Monitoring system calls

Learning from past traces
Flow size estimation: design space

Exact sizes given by application

TCP buffer occupancy

Monitoring system calls

Learning from past traces
Flow size estimation: design space

Exact sizes given by application

TCP buffer occupancy

Monitoring system calls

Learning from past traces
Flow size estimation: design space

Exact sizes given by application

TCP buffer occupancy

Monitoring system calls

Learning from past traces
Flow size estimation: design space

Exact sizes given by application

TCP buffer occupancy

Monitoring system calls

Learning from past traces
Flow size estimation: design space

Exact sizes given by application

TCP buffer occupancy

Monitoring system calls

Learning from past traces

```python
while(data in buffer):
    y = write(
        socket_desc,
        buffer+x,
        100KB
    )
```
Flow size estimation: design space

Exact sizes given by application

TCP buffer occupancy

Monitoring system calls

Learning from past traces

\[
\text{flow\_size} = f(\text{disk I/O, memory I/O, past network traffic, computation})
\]
# Learning flow size

<table>
<thead>
<tr>
<th>Workload</th>
<th>Trace</th>
<th>Model</th>
<th>Performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Web server</td>
<td>Disk I/O</td>
<td>GBDT</td>
<td>0.96</td>
</tr>
<tr>
<td>TensorFlow</td>
<td>Memory I/O</td>
<td>FFNN</td>
<td>0.97</td>
</tr>
<tr>
<td>PageRank</td>
<td>CPU cycles</td>
<td>LSTM</td>
<td>0.83</td>
</tr>
<tr>
<td>KMeans</td>
<td>Past traffic</td>
<td></td>
<td>0.88</td>
</tr>
<tr>
<td>SGD</td>
<td></td>
<td></td>
<td>0.79</td>
</tr>
</tbody>
</table>

*Values are in $R^2$  
1 – perfect prediction  
0 – mean value prediction
Sources of flow size information?

How useful is imprecise / partial knowledge?

Incentives for operators and users?
Flow scheduling using imprecise knowledge

Mean FCT (ms)

<table>
<thead>
<tr>
<th></th>
<th>pFabric</th>
<th>¹pHost</th>
<th>²FastPass</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean FCT (ms)</td>
<td>0.2</td>
<td>0.15</td>
<td>0.1</td>
</tr>
</tbody>
</table>

Perfect
Fifo

[1. pHost: Distributed Near-Optimal Datacenter Transport Over Commodity Network Fabric; CoNEXT '15]
[2. Fastpass: A Centralized “Zero-Queue” Datacenter Network; SIGCOMM ’14]
Flow scheduling using imprecise knowledge

Mean FCT (ms)

1. pHost: Distributed Near-Optimal Datacenter Transport Over Commodity Network Fabric; CoNEXT '15
2. Fastpass: A Centralized "Zero-Queue" Datacenter Network; SIGCOMM '14

Least remaining bytes first

[TensorFlow workload]
Flow scheduling using imprecise knowledge

![Graph showing Mean FCT (ms) for different network architectures: Perfect, Fifo, pFabric, pHost, FastPass.]

Mean FCT (ms)

- Perfect
- Fifo
- pFabric
- pHost
- FastPass

[1. pHost: Distributed Near-Optimal Datacenter Transport Over Commodity Network Fabric; CoNEXT '15]
Flow scheduling using imprecise knowledge

Mean FCT (ms)

- Perfect
- Fifo
- Prediction

- pFabric
- pHost
- FastPass

1. pHost: Distributed Near-Optimal Datacenter Transport Over Commodity Network Fabric; CoNEXT ’15
2. Fastpass: A Centralized “Zero-Queue” Datacenter Network; SIGCOMM ’14
Flow scheduling using imprecise knowledge

Mean FCT (ms)

- Perfect
- Fifo
- Prediction

- pFabric
- pHost
- FastPass

[TensorFlow workload]

[1. pHost: Distributed Near-Optimal Datacenter Transport Over Commodity Network Fabric; CoNEXT ’15]
[2. Fastpass: A Centralized “Zero-Queue” Datacenter Network; SIGCOMM ’14]
Flow scheduling using imprecise knowledge

Mean FCT (ms)

- Perfect
- Fifo
- Prediction
- Aging

<table>
<thead>
<tr>
<th></th>
<th>pFabric</th>
<th>¹pHost</th>
<th>²FastPass</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flow scheduling using imprecise knowledge</td>
<td><img src="image" alt="Graph showing Mean FCT for pFabric, ¹pHost, and ²FastPass" /></td>
<td><img src="image" alt="Graph showing Mean FCT for pFabric, ¹pHost, and ²FastPass" /></td>
<td><img src="image" alt="Graph showing Mean FCT for pFabric, ¹pHost, and ²FastPass" /></td>
</tr>
<tr>
<td>[TensorFlow workload]</td>
<td><img src="image" alt="Graph showing Mean FCT for pFabric, ¹pHost, and ²FastPass" /></td>
<td><img src="image" alt="Graph showing Mean FCT for pFabric, ¹pHost, and ²FastPass" /></td>
<td><img src="image" alt="Graph showing Mean FCT for pFabric, ¹pHost, and ²FastPass" /></td>
</tr>
</tbody>
</table>

[1. pHost: Distributed Near-Optimal Datacenter Transport Over Commodity Network Fabric; CoNEXT '15]
Coflow scheduling using imprecise knowledge (Sincronia*)

Slowdown

- PageRank
- KMeans
- SGD
- TensorFlow
- Web Server

[*Sincronia: Near-Optimal Network Design for Coflows; SIGCOMM '18]
Coflow scheduling using imprecise knowledge (Sincronia*)

![Graph showing slowdown for different workloads: PageRank, KMeans, TensorFlow, SGD, and Web Server. The R² values are as follows: Web server 0.96, TensorFlow 0.97, PageRank 0.83, KMeans 0.88, SGD 0.79.]

[*Sincronia: Near-Optimal Network Design for Coflows; SIGCOMM '18*]
Coflow scheduling using imprecise knowledge (Sincronia*)

The chart shows the relative performance degradation of various workloads: Web server, TensorFlow, PageRank, KMeans, and SGD. The R² values for each workload are as follows:

- Web server: 0.96
- TensorFlow: 0.97
- PageRank: 0.83
- KMeans: 0.88
- SGD: 0.79

[*Sincronia: Near-Optimal Network Design for Coflows; SIGCOMM '18]
Coflow scheduling using imprecise knowledge (Sincronia*)

Workload

<table>
<thead>
<tr>
<th>Workload</th>
<th>$R^2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Web server</td>
<td>0.96</td>
</tr>
<tr>
<td>TensorFlow</td>
<td>0.97</td>
</tr>
<tr>
<td>PageRank</td>
<td>0.83</td>
</tr>
<tr>
<td>KMeans</td>
<td>0.88</td>
</tr>
<tr>
<td>SGD</td>
<td>0.79</td>
</tr>
</tbody>
</table>

[*Sincronia: Near-Optimal Network Design for Coflows; SIGCOMM '18]
Fast enough, deployable learning?

CDF

Latency (µs)

CPU
Fast enough, deployable learning?

CDF

Latency (µs)

CPU
Fast enough, deployable learning?

CDF

Latency (µs)

CPU

FPGA
Flow size estimation: design space

Exact sizes given by application

TCP buffer occupancy

Monitoring system calls

Learning from past traces

Works only for repetitive workloads

Must identify the application

Substantial implementation effort.
Every technique provides some knowledge

- Exact sizes given by application
- TCP buffer occupancy
- Monitoring system calls
- Learning from past traces
Every technique provides some knowledge

- Exact sizes given by application
- TCP buffer occupancy
- Monitoring system calls
- Learning from past traces
Sources of flow size information?

How useful is imprecise / partial knowledge?

Incentives for operators and users?
What we want:

More knowledge $\Rightarrow$ better performance

Knowledge = Effort

Performance

0% 100%
Q1: Can this flow’s performance deteriorate?
Q2: Can the whole system’s performance deteriorate?
Q1: Can this flow’s performance deteriorate?
Q2: Can the whole system’s performance deteriorate?
Q1: Can this flow’s performance deteriorate?
Q2: Can the whole system’s performance deteriorate?

Old scheduling problem, but novel, interesting questions raised by our unique angle!
Q1: Can this flow’s performance deteriorate?
Q1: Can this flow's performance deteriorate?

Least remaining bytes first
Q1: Can this flow’s performance deteriorate?

Least remaining bytes first

Flow aging
Q1: Can this flow’s performance deteriorate?

Least remaining bytes first

A flow’s performance cannot deteriorate!

Flow aging
Q1: Can this flow’s performance deteriorate?

For coflow scheduling with mean flow size used for unknowns, performance can in fact deteriorate.
Q2: Can the whole system's performance deteriorate?
Q2: Can the whole system’s performance deteriorate?

Average co/flow completion time system-wide can deteriorate!
Q2: Can the whole system’s performance deteriorate?
Q2: Can the whole system’s performance deteriorate?

Open question: positive results with some restrictions?
Is advance knowledge of flow sizes a plausible assumption?

Sources of flow size information?
Partial knowledge available from many sources

How useful is imprecise / partial knowledge?
Can still provide performance improvements

Incentives for operators and users?
Need better guarantees on value of investment
Thank you for your attention

vojislav.dukic@inf.ethz.ch
Performance improvement

The graph shows the performance improvement for different sizes of known flows for different algorithms:

- SGD
- PageRank
- TensorFlow

Performance is measured on the y-axis, and the size of known flows is on the x-axis. The graph indicates that performance improves as the size of known flows increases for all algorithms.
<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start time, $tf$</td>
<td>Start time of $f$ relative to job start time</td>
</tr>
<tr>
<td>Flow gap</td>
<td>Time since the end of the previous flow</td>
</tr>
<tr>
<td>First call</td>
<td>Size of the first system call $tf$</td>
</tr>
<tr>
<td>Network in</td>
<td>Data received until $tf$</td>
</tr>
<tr>
<td>Network out</td>
<td>Data sent until $tf$</td>
</tr>
<tr>
<td>Network in($d$)</td>
<td>Data received from flow’s dest. $d$ until $tf$</td>
</tr>
<tr>
<td>Network out($d$)</td>
<td>Data sent by this host to $d$ until $tf$</td>
</tr>
<tr>
<td>CPU</td>
<td>CPU cycles used until $tf$</td>
</tr>
<tr>
<td>Disk I/O</td>
<td>Total disk I/O until $tf$</td>
</tr>
<tr>
<td>Memory I/O</td>
<td>Total memory I/O until $tf$</td>
</tr>
<tr>
<td>Previous flows</td>
<td>Flow sizes for last $k$ flows</td>
</tr>
</tbody>
</table>