Flowtune

Flowlet Control for Datacenter Networks

Jonathan Perry, Hari Balakrishnan and Devavrat Shah
Software in the Datacenter
Software in the Datacenter

• Response Time: Productivity, Revenue, Reputation

• microservices → develop → network is central deploy scale
Traditional approach is packet-centric

Switch Mechanisms

Server Mechanisms

Implicit Allocation
Several RTT to converge
Changes many components
Allocate network resources

- Explicitly (maximize utility)
- Quickly, Consistently
- Flexibly (in software)
Flowtune’s approach

1. Flowlet control
   
   send() ⇔ flowlet

2. Logically centralized
   • Reduce RTT dependence
Example

Hadoop on Server A has data for B:
Example

Hadoop on Server A has data for B:

A $\rightarrow$ Allocator

“Hadoop on A has data for B”
Example

Hadoop on Server A has data for B:

A → Allocator "Hadoop on A has data for B"

Allocator

Assign rates
Example

Hadoop on Server A has data for B:

A $\rightarrow$ Allocator

“Hadoop on A has data for B”

Allocator

Assign rates

Allocator $\rightarrow$ A

“Send at 40Gbps”
Example

Now say ad_server on Server C has data for B:
Example

Now say ad_server on Server C has data for B:

C $\rightarrow$ Allocator

"ad_server on C has data for B"
Example

Now say ad_server on Server C has data for B:

C → Allocator

"ad_server on C has data for B"

Allocator

Assign rates
Example

Now say ad_server on Server C has data for B:

C $\rightarrow$ Allocator

"ad_server on C has data for B"

Allocator

Assign rates

Allocator $\rightarrow$ A

"Send at 5Gbps"

Allocator $\rightarrow$ C

"Send at 35Gbps"
Network Utility Maximization (NUM)

Hadoop flowlet:

Ads flowlet:

Hadoop flowlet rate 2x → $0.05
Ads flowlet rate 2x → $0.20

Kelly et al., Journal of the Operational Research Society, 1998
NUM Iterative Optimizer

1. Each link $\ell$ chooses price $p_\ell$ using $\sum$ flow rates on $\ell$ − link capacity

2. Each flow $s$ chooses rate $x_s$

3. Goto 1
Adjusting prices
Adjusting prices

Newton Exact Diagonal (NED)
Increasing responsiveness

Solution 1: Update inputs → Run 100 iterations → Output rates

But: too slow!
Increasing responsiveness

Solution 1:

- Update inputs
- Run 100 iterations
- Output rates

But: too slow!

Solution 2:

- Update inputs
- Run 1 iteration
- Output rates

$0.01
$0.05
$0.09
Increasing responsiveness

Solution 1:

- Update inputs
- Run 100 iterations
- Output rates

But: too slow!

Solution 2:

- Update inputs
- Run 1 iteration
- Output rates

But: queueing, packet drops!
Increasing responsiveness

Solution 1:
- Update inputs
- Run 100 iterations
- Output rates

But: too slow!

Solution 2:
- Update inputs
- Run 1 iteration
- Output rates

But: queueing, packet drops!

Solution 3:
- Update inputs
- Run 1 iteration
- Normalize rates
- Output rates
Flowtune normalizes rates
Flowtune normalizes rates

\[ r_\ell = \frac{\text{allocation}}{\text{capacity}} \]
Flowtune normalizes rates

\[ r_\ell = \frac{\text{allocation}}{\text{capacity}} \]

\[ \hat{x}_S = \frac{x_S}{\max(r_\ell)} \]
Flowtune normalizes rates

\[ r_\ell = \frac{\text{allocation}}{\text{capacity}} \]

\[ \hat{x}_s = \frac{x_s}{\max(r_\ell)} \]

99.7% of optimal throughput
Multicore

Core 1: $p_1$, $p_2$, $p_3$, $p_4$

Core 2: $p_5$, $p_6$, $p_7$, $p_8$
Multicore

Core 1: $p_1, p_2, p_3, p_4$

Core 2: $p_5, p_6, p_7, p_8$
Multicore

Core 1: $p_1 \quad p_2 \quad p_3 \quad p_4$

Core 2: $p_5 \quad p_6 \quad p_7 \quad p_8$
Multicore

For each link $\ell$ compute $f$ (flows on $\ell$)
For each flow $x$ compute $g$ (links $x$ traverses)
Multicore

```
core  core  core  core
0     1     2     3

core  core  core  core
4     5     6     7

core  core  core  core
8     9     10    11

core  core  core  core
12    13    14    15
```
Multicore

Block 1  Block 2  Block 3  Block 4

Source

Block 1
Block 2
Block 3
Block 4

Destination
Multicore

Block 1
Block 2
Block 3
Block 4

Source

Block 1  Block 2  Block 3  Block 4

Destination

<table>
<thead>
<tr>
<th>Block 1</th>
<th>Block 2</th>
<th>Block 3</th>
<th>Block 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>core</td>
<td>core</td>
<td>core</td>
<td>core</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>core</td>
<td>core</td>
<td>core</td>
<td>core</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>core</td>
<td>core</td>
<td>core</td>
<td>core</td>
</tr>
<tr>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
</tr>
<tr>
<td>core</td>
<td>core</td>
<td>core</td>
<td>core</td>
</tr>
<tr>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
</tr>
</tbody>
</table>
Multicore

Block 1

Block 2

Block 3

Block 4

Source

Block 1 | Block 2 | Block 3 | Block 4
---|---|---|---
core | core | core | core
0 | 1 | 2 | 3

core | core | core | core
4 | 5 | 6 | 7

core | core | core | core
8 | 9 | 10 | 11

core | core | core | core
12 | 13 | 14 | 15

Destination
Multicore
Multicore

Block 1  Block 2  Block 3  Block 4

Source
Block 1  core  core  core  core
Block 2  core  core  core  core
Block 3  4    5    6    7
Block 4  8    9    10   11

Destination
Block 1  Block 2  Block 3  Block 4
12 13 14 15
Multicore

Block 1
Block 2
Block 3
Block 4

Source
Block 1
Block 2
Block 3
Block 4

Destination

Core 0 -> Core 3
Core 4 -> Core 7
Core 8 -> Core 11
Core 12 -> Core 15
Multicore
Multicore

Block 1: Core 0, Core 1, Core 2, Core 3
Block 2: Core 4, Core 5, Core 6, Core 7
Block 3: Core 8, Core 9, Core 10, Core 11
Block 4: Core 12, Core 13, Core 14, Core 15

Source:
Block 1: Core 0, Core 1, Core 2, Core 3
Block 2: Core 4, Core 5, Core 6, Core 7
Block 3: Core 8, Core 9, Core 10, Core 11
Block 4: Core 12, Core 13, Core 14, Core 15

Destination:
Block 1: Core 0, Core 1, Core 2, Core 3
Block 2: Core 4, Core 5, Core 6, Core 7
Block 3: Core 8, Core 9, Core 10, Core 11
Block 4: Core 12, Core 13, Core 14, Core 15
Multicore

Block 1 | Block 2 | Block 3 | Block 4
--- | --- | --- | ---
0 | 1 | 2 | 3
4 | 5 | 6 | 7
8 | 9 | 10 | 11
12 | 13 | 14 | 15

Source

Destination
Multicore

Block 1  Block 2  Block 3  Block 4

Source

Block 1  Block 2  Block 3  Block 4

Destination

Block 1  Block 2  Block 3  Block 4
Multicore

Block 1  Block 2  Block 3  Block 4

Source

Block 1  core  0  core  1  core  2  core  3
Block 2  core  4  core  5  core  6  core  7
Block 3  core  8  core  9  core 10  core 11
Block 4  core 12  core 13  core 14  core 15

Destination
In the paper...

Flow view

Link view
4608 servers in $<31\mu s$

<table>
<thead>
<tr>
<th>Cores</th>
<th>Nodes</th>
<th>Flows</th>
<th>Cycles</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>384</td>
<td>3072</td>
<td>19896.6</td>
<td>8.29 $\mu s$</td>
</tr>
<tr>
<td>16</td>
<td>768</td>
<td>6144</td>
<td>21267.8</td>
<td>8.86 $\mu s$</td>
</tr>
<tr>
<td>64</td>
<td>1536</td>
<td>12288</td>
<td>30317.6</td>
<td>12.63 $\mu s$</td>
</tr>
<tr>
<td>64</td>
<td>1536</td>
<td>24576</td>
<td>33576.2</td>
<td>13.99 $\mu s$</td>
</tr>
<tr>
<td>64</td>
<td>1536</td>
<td>49152</td>
<td>40628.5</td>
<td>16.93 $\mu s$</td>
</tr>
<tr>
<td>64</td>
<td>3072</td>
<td>49152</td>
<td>57035.9</td>
<td>23.76 $\mu s$</td>
</tr>
<tr>
<td>64</td>
<td>4608</td>
<td>49152</td>
<td>73703.2</td>
<td>30.71 $\mu s$</td>
</tr>
</tbody>
</table>

Communication $>\frac{1}{2}$ of time
EC2: Resource Allocation

8 senders, every 50 milliseconds

![Diagram showing resource allocation](image)
Ns-2: Flowtune converges quickly to a fair allocation

Every 10 milliseconds:

senders

receiver
Overhead is low

99% of links < 10% utilized

Inside the Social Network’s (Datacenter) Network, Roy et al., SIGCOMM’15
Open Questions

• Handling mice
  • Bypass the allocator? Fastpass?

• External traffic
  • Measure & react?

• Deadlines, Co-flow
  • Market?

• Multicore: 3-tier Clos, WAN
Give application developers control over network transport