Ether: Providing both Interactive Service and Fairness in Multi-Tenant Datacenters

Mojtaba Malekpourshahraki
Brent Stephens
Balajee Vamanan

BITS
Networked Systems Laboratory

mmalek3@uic.edu
Modern datacenter

• Datacenters host multiple applications with different requirements
  • Memcache (delay)
  • Web search (delay)
  • Spark (throughput)

• Datacenters host multiple competing tenants
  • Private datacenters
    • Example: Facebook
    • Tenants → Product and applications groups
  • Public datacenters
    • Example: Amazon EC2, Microsoft Azure
    • Tenants → Users renting virtual machines

Modern datacenters = Multiple tenants × Diverse applications per tenant
Network sharing

• Datacenter has lots of
  • Applications → Different requirements, different protocols
  • Flows → different size, traffic pattern
  • Tenant → same or different priority

How to handle this complexity?

Datacenter must meet three main requirements
Multi-tenant datacenter requirements

• Isolation among tenants
  • Example: Each tenant must have the fair share
Multi-tenant datacenter requirements

- Isolation among tenants
  - Example: each tenant must have the fair share

- Low latency for high priority applications
  - Co-located memcache and spark leads to a high latency
Multi-tenant datacenter requirements

- Isolation among tenants
  - Example: Each tenant must have the fair share

- Low latency for high priority applications
  - Co-located memcached and spark leads to a high latency

- Utilization
  - The bottleneck capacity must be fully utilized

How to address all of these requirements together?
• A scheduler could address all these network requirements
  • Isolation among tenants
  • Low latency for high priority applications
  • Utilization

• Scheduler Goals
  • Implementable in limited available network resource
    • Unlimited resources could do anything
  • Provides a set of useful scheduling algorithm

A scheduler that achieve all these goals is hard
Limitations on designing schedulers

• The number of scheduling queues is limited

• Two types of schedulers
  • End-host based or end-to-end schedulers
  • Switch based schedulers
End-host schedulers

• End-host
  • has many queues → only at the end host doesn’t need our requirement

• Shortcomings
  • Waste of resource
    • pHost (Sending RTS)
    • Silo (Limiting burst size)
  • High computational overhead
    • Fastpass (Centralized)
  • Slow to adapt network changes
    • Trinity (ECN mark)

In switch approaches perform faster than end-to-end schedulers
PIFO/PIEO:
- Can implement complex hierarchical programmable scheduling policies

PIFO/PIEO resources on a switch is limited (less than 100 queues)
- Cannot use PIFO to implement the full scheduling policies in switches
- You cannot have all possible scheduler
- The number of required queues increases with the number of traffic class
Can we implement a useful set of scheduling policies within the constant number of scheduling queues?
Ether overview

Contributions
• Decoupling fair queueing (fairness) from priority queue (FCT)
• Variety of scheduling
  • Fair queueing, Priority queue, SJF, LSTF
  • Any combination of them
• Ether requires a fixed number of scheduling queues

Implementation:
• the implementation of two-sided-queue in programmable switches

Key insight:
• Trade-off fairness in short time intervals in bounded intervals
• Steal capacity from a tenant in short period of times to optimize tail FCT in others
Outline

• Datacenter network
• Existing proposals

• Design
  • Ether High level
  • Design potential

• Results
Ether framework

- Ether uses two set of queues and two hashing functions

- Queue operations
  - Enqueue
    - $H_t(Tenant\ ID)$  \(\xrightarrow{\text{Fairness optimizer}}\)
    - $H_f(flow\ ID) \xrightarrow{\text{Tail optimizer}}$
  - Dequeue
    - Fairness optimizer \(\xrightarrow{\text{in each round}}\)
    - Tail optimizer \(\xrightarrow{\text{based on priority}}\)

<table>
<thead>
<tr>
<th>Packet Format</th>
<th>Priority</th>
<th>Packet</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Fairness optimizer

Enqueue based on $H_f(id_{tenant})$

Tail optimizer

Enqueue based on $H_t(id_{flow})$

Dequeue based on the priority

$q_{min}$
Ether framework – fairness optimizer

• Fairness optimizer
  • In each round, $q_{\text{min}}$ bytes dequeue from fairness to tail optimizer
  • $q_{\text{min}}$ is the smallest nonzero queue length
  • Packets distributes to any of $n$ queues in fairness $H_f(id_{\text{tenant}})$

\[
q_{\text{min}} = \min_i \text{len}(Q_i)
\]
Ether framework – tail optimizer

- Tail optimizer
  - Enqueue packets based on the flow ID, $H_f(id_{flow})$
  - Dequeue the packets based on their priority
  - Priority is slack time

![Diagram showing fairness optimizer and tail optimizer](image-url)
• Two tenants
  • Tenant 1 (Memcached, Websearch)
  • Tenant 2 (Spark)

Spark < Memcached < Websearch
How it works?

- Two tenants
  - Tenant 1 (Memcached, Websearch)
  - Tenant 2 (Spark)

The window is important in accuracy of approximation

Spark < Memcached < Websearch

Order of dequeuer is different

Ether

Fairness

Priority

0
1
2
9
Discussion – windowing limits

• Some tenants generate few packets
  • **Issue**: adversely affect the tail optimization
  • **Reason**: few packets in the tail optimizer
  • **Solution**: limit minimum window size ($w_{\text{min}}$)

\[ w_{\text{min}} \text{ controls FCT performance} \]

• All tenants generates too many packets
  • **Issue**: too many flows in the window
  • **Reason**: Too many hash collisions for flows
  • **Solution**: limit maximum window size ($w_{\text{max}}$)

\[ w_{\text{max}} \text{ controls collisions on the FCT optimizer and the number of queues in FCT optimizer} \]

**Best window size**: 
\[ w = \min(w_{\text{max}}, \max(w_{\text{min}}, q_{\text{min}})) \]
Outline

• Datacenter network
• Existing proposals

• Design
  • Ether High level
  • Design potential

• Results
Ether with Programmable Switches

- Programmable switches
  - There is no two layers of queues
- Solution:
  - Divide the queue space into two
  - Use different hashing functions
  - Implement two layers of switches using packet resubmit

Implement a two steps of multilevel queue using PSA
Ether could provide a set of scheduling algorithms

<table>
<thead>
<tr>
<th>WFQ</th>
<th>SJF</th>
<th>Strict Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>$w_{\text{min}} \rightarrow q_{\text{min}}$</td>
<td>$w_{\text{min}} \rightarrow \infty$</td>
<td>$w_{\text{min}} \rightarrow \infty$</td>
</tr>
<tr>
<td>$w_{\text{max}} \rightarrow q_{\text{min}}$</td>
<td>$w_{\text{max}} \rightarrow \infty$</td>
<td>$w_{\text{max}} \rightarrow \infty$</td>
</tr>
<tr>
<td>priority $\rightarrow 0$</td>
<td>priority $\rightarrow \text{flow size}$</td>
<td>priority $\rightarrow \text{priority}$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>LSTF</th>
<th>LSTF + WFQ</th>
<th>LSTF + WFQ + Strict Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>$w_{\text{min}} \rightarrow \infty$</td>
<td>$w_{\text{min}} \rightarrow \text{short packet size}$</td>
<td>$w_{\text{min}} \rightarrow \text{short packet size}$</td>
</tr>
<tr>
<td>$w_{\text{max}} \rightarrow \infty$</td>
<td>$w_{\text{max}} \rightarrow \text{average size of flows}$</td>
<td>$w_{\text{max}} \rightarrow \text{number of distinct flows}$</td>
</tr>
<tr>
<td>priority $\rightarrow \text{slack time}$</td>
<td>$w_{\text{max}} \rightarrow \frac{\text{average size of flows}}{\text{number of FCT queues}}$</td>
<td>priority $\rightarrow \left{ \begin{array}{l} \text{low priority: slack time} \ \text{high priority: 0} \end{array} \right.$</td>
</tr>
</tbody>
</table>
Evaluation

• Goals
  • Can Ether achieve fairness among all tenant and improve the FCT per tenant?

• How Ether works compared to
  • Ideal fair queuing (FQ)
  • Ideal FCT optimizer (pFabric)

• Measured parameters
  • Jain’s index fairness
  • 99 percentile tail flow completion time
Methodology

• Network simulator (ns-3 v3.28)
  • Workloads
    
    | Short flows | 8 - 32 KB |
    | Long flows  | 1 MB      |
    | Number of tenants | 10 |

• Parameters

| \( w_{\text{min}} \) | 1  |
| \( w_{\text{max}} \) | 570(pkt) |
Ether outperforms pFabric’s fairness by 18%, Ether outperforms FQ tail FCT by 25%
Sensitivity to number of queues

For the workload we evaluated, the number of required queues between 24 to 32

Variable fairness optimizer
Fixed = 16 FCT optimizer

Fixed = 16 fairness optimizer
Variable FCT optimizer

For the workload we evaluated, the number of required queues between 24 to 32
Conclusion

• We proposed Ether
  • Ensuring fairness over longer timescales
  • Provide short tail FCT over shorter timescales

• We observed that Ether
  • Ether outperforms pFabric’s fairness by 18%
  • Ether outperforms FQ tail FCT by 25%

• Future work:
  • Implement Ether on programmable switches
  • Generalize the architecture to support other scheduler types
  • Generalize the architecture to support hierarchy
Thanks for the attention

Mojtaba Malekpourshahraki

**Email:** mmalek3@uic.edu
**Website:** cs.uic.edu/~mmalekpo