WORKLOAD CHARACTERIZATION OF INTERACTIVE CLOUD SERVICES
ON
BIG AND SMALL SERVER PLATFORMS

Shuang Chen*, Shay Galon**, Christina Delimitrou*, Srilatha Manne**, and José Martínez*

*Cornell University
**Cavium Inc.
Executive Summary

- How to achieve low tail latency for interactive cloud services?
  - Tail latency more important and challenging
  - The entire stack from SW to HW is involved

- Understand how tail latency reacts to application and system changes
  - See how current designs work
  - Get insights on future designs
**Motivation**

- Google Translate
- Bing
- Google Maps

- NGINX
- MEMCACHED
- MongoDB®

---

**Cornell University Computer Systems Laboratory**

- Introduction
- Characterization
- Implications
**Tail latency**

- e.g., QoS defined as 99th %ile in 500 usec

\[0.99^5 = 0.95\]
The entire stack from SW to HW is involved

- Application bottleneck
- Different user cases
- Scalability

- Overhead of virtualization
- SW isolation mechanisms
- Overhead of context switching
- HW isolation mechanisms
- Hyperthreading
By requirement of tail latency
- us: memcached
- ms: web server, in-memory database
- s: persistent database

By statefulness
- Stateful: memcached
- Stateless: web server
**SELECTED LC WORKLOADS**

- **NGINX**
  - Web server
  - Stateless
  - $99^{th}$% in tens of ms

- **Memcached**
  - Key-value store
  - Stateful
  - $99^{th}$% in hundreds of us
**Server Architecture**

**Intel Xeon E5-2699 v4**
- 22 Cores
- 2 Threads/Core
- L1 I/D: 32/32KB
- L2: 256KB
- LLC: 55MB, 20 ways
- Memory: 128G DDR4
- NIC: 10Gbps
- 14nm
- $4,115

**Cavium ThunderX**
- 48 Cores
- 1 Thread/Core
- L1 I/D: 78/32KB
- L2: 256KB
- LLC: 16MB, 16 ways
- Memory: 128G DDR4
- NIC: 10Gbps
- 28nm
- $785
STUDIED PARAMETERS

- Application bottleneck
- Different user cases
- Scalability

- Overhead of virtualization
- SW isolation mechanisms
- Overhead of context switching
- HW isolation mechanisms
- Hyperthreading
**Input Load**

**Introduction**
- Characterization
- Implications

---

**Memcached**

![Graph showing latency (μsec) vs. percentage of Max RPS for Xeon and ThunderX.]

- **Xeon**: 5.2x
- **ThunderX**: 5x

---

**NGINX**

![Graph showing latency (msec) vs. percentage of Max RPS for Xeon and ThunderX.]

- **Xeon**: 5.2x
- **ThunderX**: 5x
**Memcached Latency Decomposition**

Introduction • Characterization • Implications

- **Little user-space processing**

  - At 10% of max throughput
    - Xeon: 6 3 1 1 1 5
    - ThunderX: 14 4 5 9 7 24

  - Network delay
    - 2x slower than Xeon

  - At 90% of max throughput
    - Xeon: 6 782 1,009 315
    - ThunderX: 14 1,290 1,650

- **Queuing delay**

  - At 90% of max throughput
    - Xeon: 6 782 1,009 315
    - ThunderX: 14 1,290 1,650

- **Both NICs at 10% of max throughput:**
  - Xeon: 6 31115
  - ThunderX: 14 4 5 9 7 24

- **Both NICs at 90% of max throughput:**
  - Xeon: 6 782 1,009 315
  - ThunderX: 14 1,290 1,650

- **General Observations:**
  - BM: At 10% of max throughput, Xeon is 2x slower than ThunderX.
  - BM: At 90% of max throughput, ThunderX shows a significant increase in processing time compared to Xeon.

- **IRQ, Kernel, Syscall, User**

- **Nic**

- **TX**
STUDIED PARAMETERS

- Application bottleneck
- Different user cases
- Scalability

- Overhead of virtualization
- SW isolation mechanisms
- Overhead of context switching
- HW isolation mechanisms
- Hyperthreading
Memcached Value Size

Xeon
• Memory copy
• Network processing and transmission
• ThunderX is more sensitive

ThunderX
**Number of Memcached Items**

- **Xeon**
  - Cache capacity
  - ThunderX is more sensitive

- **ThunderX**
STUDIED PARAMETERS

- Application bottleneck
- Different user cases
- Scalability

- Overhead of virtualization
- SW isolation mechanisms
- Overhead of context switching
- HW isolation mechanisms
- Hyperthreading
SCALABILITY

Memcached

• Interrupt handling
• Load imbalance
• Lock contention

NGINX

Introduction • Characterization • Implications
**STUDIED PARAMETERS**

- Application bottleneck
- Different user cases
- Scalability

- Overhead of virtualization
- SW isolation mechanisms
  - **Overhead of context switching**
- HW isolation mechanisms
- Hyperthreading
**CONTEXT SWITCHING**

- Statically spawned threads VS dynamically allocated cores
- ThunderX is more sensitive

Memcached on Xeon

Memcached on ThunderX
**STUDIED PARAMETERS**

- Application bottleneck
- Different user cases
- Scalability

- Overhead of virtualization
- SW isolation mechanisms
- Overhead of context switching
- HW isolation mechanisms
- **Hyperthreading**
- **Reduce the overhead of context switching**
  - Allocate two threads on two hyperthreads

- **Make better use of execution units**
  - Co-locate different applications

<table>
<thead>
<tr>
<th></th>
<th>10%</th>
<th>20%</th>
<th>30%</th>
<th>40%</th>
<th>50%</th>
<th>60%</th>
<th>70%</th>
</tr>
</thead>
<tbody>
<tr>
<td>10%</td>
<td>MN</td>
<td>MN</td>
<td>MN</td>
<td>MN</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20%</td>
<td>MN</td>
<td>MN</td>
<td>MN</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>30%</td>
<td>MN</td>
<td>MN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>40%</td>
<td>MN</td>
<td>N</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>50%</td>
<td>N</td>
<td>N</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>60%</td>
<td>N</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>70%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Memcached & Nginx on the same hyperthreads**

<table>
<thead>
<tr>
<th></th>
<th>10%</th>
<th>20%</th>
<th>30%</th>
<th>40%</th>
<th>50%</th>
<th>60%</th>
<th>70%</th>
</tr>
</thead>
<tbody>
<tr>
<td>10%</td>
<td>MN</td>
<td>MN</td>
<td>MN</td>
<td>MN</td>
<td>MN</td>
<td>MN</td>
<td>M</td>
</tr>
<tr>
<td>20%</td>
<td>MN</td>
<td>MN</td>
<td>MN</td>
<td>MN</td>
<td>MN</td>
<td>M</td>
<td>M</td>
</tr>
<tr>
<td>30%</td>
<td>MN</td>
<td>MN</td>
<td>MN</td>
<td>MN</td>
<td>M</td>
<td>M</td>
<td>M</td>
</tr>
<tr>
<td>40%</td>
<td>MN</td>
<td>MN</td>
<td>MN</td>
<td>MN</td>
<td>M</td>
<td>M</td>
<td>M</td>
</tr>
<tr>
<td>50%</td>
<td>MN</td>
<td>MN</td>
<td>N</td>
<td>N</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>60%</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>70%</td>
<td>N</td>
<td>N</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Memcached & Nginx on different hyperthreads**
- Reduce queuing delays
- Improve elasticity
  - Lock alternatives
  - Load balance

- Reduce the overhead of virtualization
- Avoid context switching
- Make best use of SW isolation mechanisms
- Big VS Small Cores
- Make best use of HW features