PARTIES:

**QoS-Aware Resource Partitioning for Multiple Interactive Services**

*Shuang Chen, Christina Delimitrou, José F. Martínez*

Cornell University
COLOCATION OF APPLICATIONS

Best-effort

Private caches

Private caches

Private caches

Private caches

Last-level Cache

QoS violations 😞

Latency -critical

Motivation • Characterization • PARTIES • Evaluation • Conclusions
PRIOR WORK

- Interference during colocation
- Scheduling [Nathuji’10, Mars’13, Delimitrou’14]
  - Avoid co-scheduling of apps that may interfere
    - May require offline knowledge
    - Limit colocation options
- Resource partitioning [Sanchez’11, Lo’15]
  - Partition shared resources
    - At most 1 LC app + multiple best-effort jobs
TRENDS IN DATACENTERS

1. LC + many BE
   - Best-effort
   - Latency-critical
   - More LC jobs
   - Many LC + many BE
   - All have QoS targets

Monolith

Microservices
Main Contributions

- **Workload characterization**
  - The impact of resource sharing
  - The effectiveness of resource isolation
  - Relationship between different resources

- **PARTIES:** First QoS-aware resource manager for colocation of many LC services
  - Dynamic partitioning of 9 shared resources
  - No a priori application knowledge
  - 61% higher throughput under QoS constraints
  - Adapts to varying load patterns
## Interactive LC Applications

<table>
<thead>
<tr>
<th>Application</th>
<th>Memcached</th>
<th>Xapian</th>
<th>NGINX</th>
<th>Moses</th>
<th>MongoDB</th>
<th>Sphinx</th>
</tr>
</thead>
<tbody>
<tr>
<td>Domain</td>
<td>Key-value</td>
<td>Web</td>
<td>Web</td>
<td>Real-time</td>
<td>Persistent</td>
<td>Speech</td>
</tr>
<tr>
<td>Target QoS</td>
<td>600us</td>
<td>5ms</td>
<td>10ms</td>
<td>15ms</td>
<td>300ms</td>
<td>2.5s</td>
</tr>
<tr>
<td>Max Load</td>
<td>1,280,000</td>
<td>8,000</td>
<td>560,000</td>
<td>2,800</td>
<td>240</td>
<td>14</td>
</tr>
<tr>
<td>User / Sys / IO CPU%</td>
<td>13 / 78 / 0</td>
<td>42 / 23 / 0</td>
<td>20 / 50 / 0</td>
<td>50 / 14 / 0</td>
<td>0.3 / 0.2 / 57</td>
<td>85 / 0.6 / 0</td>
</tr>
<tr>
<td>LLC MPKI</td>
<td>0.55</td>
<td>0.03</td>
<td>0.06</td>
<td>10.48</td>
<td>0.01</td>
<td>6.28</td>
</tr>
<tr>
<td>Memory Capacity</td>
<td>9.3 GB</td>
<td>0.02 GB</td>
<td>1.9 GB</td>
<td>2.5 GB</td>
<td>18 GB</td>
<td>1.4 GB</td>
</tr>
<tr>
<td>Memory Bandwidth</td>
<td>0.6 GB/s</td>
<td>0.01 GB/s</td>
<td>0.6 GB/s</td>
<td>26 GB/s</td>
<td>0.03 GB/s</td>
<td>3.1 GB/s</td>
</tr>
<tr>
<td>Disk Bandwidth</td>
<td>0 MB/s</td>
<td>0 MB/s</td>
<td>0 MB/s</td>
<td>0 MB/s</td>
<td>5 MB/s</td>
<td>0 MB/s</td>
</tr>
<tr>
<td>Network Bandwidth</td>
<td>3.0 Gbps</td>
<td>0.07 Gbps</td>
<td>6.2 Gbps</td>
<td>0.001 Gbps</td>
<td>0.01 Gbps</td>
<td>0.001 Gbps</td>
</tr>
</tbody>
</table>
## Interactive LC Applications

<table>
<thead>
<tr>
<th>Application</th>
<th>Memcached</th>
<th>Xapian</th>
<th>NGINX</th>
<th>Moses</th>
<th>MongoDB</th>
<th>Sphinx</th>
</tr>
</thead>
<tbody>
<tr>
<td>Domain</td>
<td>Key-value store</td>
<td>Web search</td>
<td>Web server</td>
<td>Real-time translation</td>
<td>Persistent database</td>
<td>Speech recognition</td>
</tr>
<tr>
<td>Target QoS</td>
<td>600us</td>
<td>5ms</td>
<td>10ms</td>
<td>15ms</td>
<td>300ms</td>
<td>2.5s</td>
</tr>
<tr>
<td>Max Load</td>
<td>1,280,000</td>
<td>8,000</td>
<td>560,000</td>
<td>2,800</td>
<td>240</td>
<td>14</td>
</tr>
<tr>
<td>User / Sys / IO CPU%</td>
<td>13 / 78 / 0</td>
<td>42 / 23 / 0</td>
<td>20 / 50 / 0</td>
<td>50 / 14 / 0</td>
<td>0.3 / 0.2 / 57</td>
<td>85 / 0.6 / 0</td>
</tr>
<tr>
<td>LLC MPKI</td>
<td>0.55</td>
<td>0.03</td>
<td>0.06</td>
<td>10.48</td>
<td>0.01</td>
<td>6.28</td>
</tr>
<tr>
<td>Memory Capacity</td>
<td>9.3 GB</td>
<td>0.02 GB</td>
<td>1.9 GB</td>
<td>2.5 GB</td>
<td>18 GB</td>
<td>1.4 GB</td>
</tr>
<tr>
<td>Memory Bandwidth</td>
<td>0.6 GB/s</td>
<td>0.01 GB/s</td>
<td>0.6 GB/s</td>
<td>26 GB/s</td>
<td>0.03 GB/s</td>
<td>3.1 GB/s</td>
</tr>
<tr>
<td>Disk Bandwidth</td>
<td>0 MB/s</td>
<td>0 MB/s</td>
<td>0 MB/s</td>
<td>0 MB/s</td>
<td>5 MB/s</td>
<td>0 MB/s</td>
</tr>
<tr>
<td>Network Bandwidth</td>
<td>3.0 Gbps</td>
<td>0.07 Gbps</td>
<td>6.2 Gbps</td>
<td>0.001 Gbps</td>
<td>0.01 Gbps</td>
<td>0.001 Gbps</td>
</tr>
</tbody>
</table>

Max load: max RPS under QoS target when running alone
## Interference Study

### Table 1: Latency-Critical Applications

<table>
<thead>
<tr>
<th>Application</th>
<th>Memcached</th>
<th>Xapian</th>
<th>NGINX</th>
<th>Moses</th>
<th>MongoDB</th>
<th>Sphinx</th>
</tr>
</thead>
<tbody>
<tr>
<td>Domain</td>
<td>Key-value</td>
<td>Web</td>
<td>Web</td>
<td>Web</td>
<td>Persistent</td>
<td>Speech</td>
</tr>
<tr>
<td>Target QoS</td>
<td>600us</td>
<td>5ms</td>
<td>10ms</td>
<td>15ms</td>
<td>300ms</td>
<td>2.5s</td>
</tr>
<tr>
<td>Max Load</td>
<td>1,280,000</td>
<td>8,000</td>
<td>560,000</td>
<td>2,800</td>
<td>240</td>
<td>14</td>
</tr>
<tr>
<td>User / Sys / IO</td>
<td>13 / 78 / 0</td>
<td>42 / 23 / 0</td>
<td>20 / 50 / 0</td>
<td>50 / 14 / 0</td>
<td>0.3 / 0.2 / 57</td>
<td>85 / 0.6 / 0</td>
</tr>
<tr>
<td>LLC MPKI</td>
<td>0.55</td>
<td>0.03</td>
<td>0.06</td>
<td>10.48</td>
<td>0.01</td>
<td>6.28</td>
</tr>
<tr>
<td>Memory Capacity</td>
<td>9.3 GB</td>
<td>0.02 GB</td>
<td>1.9 GB</td>
<td>2.5 GB</td>
<td>18 GB</td>
<td>1.4 GB</td>
</tr>
<tr>
<td>Memory Bandwidth</td>
<td>0.6 GB/s</td>
<td>0.01 GB/s</td>
<td>0.6 GB/s</td>
<td>26 GB/s</td>
<td>0.03 GB/s</td>
<td>3.1 GB/s</td>
</tr>
<tr>
<td>Disk Bandwidth</td>
<td>0 MB/s</td>
<td>0 MB/s</td>
<td>0 MB/s</td>
<td>0 MB/s</td>
<td>5 MB/s</td>
<td>0 MB/s</td>
</tr>
<tr>
<td>Network Bandwidth</td>
<td>3.0 Gbps</td>
<td>0.07 Gbps</td>
<td>6.2 Gbps</td>
<td>0.001 Gbps</td>
<td>0.01 Gbps</td>
<td>0.001 Gbps</td>
</tr>
</tbody>
</table>

### Table 2: Impact of resource interference. Each row corresponds to a type of resource. Values in the table are the maximum percentage of max load for which the server can satisfy QoS when the LC application is running under interference. Cells with smaller numbers and darker colors mean that applications are more sensitive to that type of interference.

<table>
<thead>
<tr>
<th>Application</th>
<th>Memcached</th>
<th>Xapian</th>
<th>NGINX</th>
<th>Moses</th>
<th>MongoDB</th>
<th>Sphinx</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hyperthread</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CPU</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Power</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LLC Capacity</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LLC Bandwidth</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Memory Bandwidth</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Memory Capacity</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Disk Bandwidth</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Network Bandwidth</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Applications are sensitive to resources with high usage
- Applications with strict QoS targets are more sensitive
**ISOLATION MECHANISMS**

- Core mapping
  - Hyperthreads
  - Core counts
- Memory capacity
- Disk bandwidth
- Core frequency
  - Power
- LLC capacity
  - Cache capacity
  - Cache bandwidth
  - Memory bandwidth
- Network bandwidth
• **Resources are fungible**
  - More flexibility in resource allocation
  - Simplifies resource manager
PARTIES: DESIGN PRINCIPLES

PARTIES

• PARTitioning for multiple InteractivE Services

Design principles

• LC apps are equally important
• Allocation should be dynamic and fine-grained
• No a priori application knowledge or offline profiling
• Recover quickly from incorrect decisions
• Migration is used as a last resort
PARTIES

- 5 knobs organized into 2 wheels
- Start from a random resource
- Follow the wheels to visit all resources

Unallocated pool

Client side

Latency Monitor

Poll latency every 100ms

Main Function

QoS violations?
- Upsize!
- Excess resources?
- Downsize!

App 1

App 2

Computation

Storage

Unallocated pool

Server side

Motivation • Characterization • PARTIES • Evaluation • Conclusions

Page 11 of 15
**Methodology**

- **Platform: Intel E5-2699 v4**
  - Single socket with 22 cores (8 IRQ cores)

- **Virtualization**
  - LXC 2.0.7

- **Load generators**
  - Open loop
  - Request inter-arrival distribution: exponential
  - Request popularity: Zipfian

- **Testing strategy**
  - Constant load: 30s warmup, 1m measurement (x5)
  - Varying load simulates diurnal load patterns
**CONSTANT LOADS: MEMCACHED, XAPIAN & NGINX**

**Oracle**
- Offline profiling
- Always finds the global optimum

**Heracles**
- No partitioning between BE jobs
- Suspend BE upon QoS violation
- No interaction between resources
Motivation• Characterization• PARTIES• Evaluation• Conclusions

MORE EVALUATION

Constant loads
- All 2- and 3-app mixes under PARTIES
- Comparison with Heracles for 2- to 6-app mixes

Diurnal load pattern
- Colocation of Memcached, Xapian and Moses

PARTIES overhead
- Convergence time for 2- to 6-app mixes
CONCLUSIONS

- Need to manage multiple LC apps
- Insights
  - Resource partitioning
  - Resource fungibility
- PARTIES
  - Partition 9 shared resources
  - No offline knowledge required
  - 61% higher throughput under QoS targets
  - Adapts to varying load patterns
PARTIES:
QoS-Aware Resource Partitioning for Multiple Interactive Services

http://tiny.cc/parties

Shuang Chen, Christina Delimitrou, José F. Martínez
Cornell University