TOWARD PREDICTABLE PERFORMANCE IN SOFTWARE PACKET-PROCESSING PLATFORMS

Mihai Dobrescu, EPFL
Katerina Argyraki, EPFL
Sylvia Ratnasamy, UC Berkeley
Programmable Networks

- Industry/research community efforts
  - Easily deploy new services
  - Test research ideas

- Software packet processing
  - General purpose hardware
  - Familiar programming environment

Extensible network functionality
Problem: Unpredictable Performance

- Resource contention
  - Caches, memory controllers, buses
  - Performance interference

- Software packet-processing systems [Dobrescu’09, Han’10]
  - High performance
  - Same processing for all packets

Goal: software packet processing with predictable performance
System Overview

- IP forward
- Filtering
- Encryption
- Statistics

Contestation for shared resources

General purpose server

Input traffic

Packet processing

Output traffic
Is This Hard?

- Yes, in general-purpose context
  - Math models to predict contention
  - Contention-aware job placement

- In packet-processing context?
Our Contribution

1. It is feasible to build a packet-processing platform with predictable performance using simple techniques.

2. Contention-aware job placement does not bring significant benefit to the overall performance.
Outline

- System overview
- Contention factors
- Observations on application behavior
- A simple prediction method
- Intuition
System Overview

General purpose server

Input traffic → packet processing → output traffic

- IP forward
- Filtering
- Encryption
- Statistics
## Workloads

<table>
<thead>
<tr>
<th>Application</th>
<th>Main functionality</th>
<th>Characteristics</th>
</tr>
</thead>
<tbody>
<tr>
<td>IP</td>
<td>IP routing, 128k entries</td>
<td>L3 cache intensive</td>
</tr>
<tr>
<td>MON</td>
<td>Monitoring, 100k flows</td>
<td>L3 cache intensive</td>
</tr>
<tr>
<td>FW</td>
<td>Firewall, 1000 rules</td>
<td>L2 cache intensive</td>
</tr>
<tr>
<td>RE</td>
<td>Redundancy elimination</td>
<td>Memory intensive</td>
</tr>
<tr>
<td>VPN</td>
<td>Encryption</td>
<td>CPU intensive</td>
</tr>
<tr>
<td>Synthetic</td>
<td>Random cache reads</td>
<td>Cache/memory/CPU</td>
</tr>
</tbody>
</table>

Representative set of realistic applications
Setup

- Linux + Click
- Commodity Intel Xeon server
Basic Configuration

- One application per core
- NUMA-aware memory allocation

Contended resources: cache and memory controller
Resource Contention Effects

- IP competitors
- MON competitors
- FW competitors
- RE competitors
- VPN competitors

Performance Drop (%)

- IP
- MON
- FW
- RE
- VPN
Outline

- System overview
- Contention factors
- Observations on application behavior
- A simple prediction method
- Intuition
Contestion Factors

- **5 synthetic competitors**

Graph showing performance drop with different factors:

- **Cache + Memory Controller Contention**
- **Cache Contention**
- **Memory Controller Contention**

Legend:
- IP
- MON
- FW
- RE
- VPN

Cache is the dominant contention factor
Outline

- System overview
- Contention factors
- Observations on application behavior
- A simple prediction method
- Intuition
Characterize Application Behavior

Continuous curves: synthetic competitors

- Performance Drop (%)
- Competitors' L3 refs/sec (M)
Characterize Application Behavior

Performance Drop (%) vs. Competitors’ L3 refs/sec (M)

Continuous curves: synthetic competitors
Individual points: realistic competitors
Characterize Application Behavior

Obs. #1: competitors’ cache refs/sec determine drop
Characterize Application Behavior

Obs. #1: competitors’ cache refs/sec determine drop
Characterize Application Behavior

Obs. #2: drop curve grows slowly after certain point
Outline

- System overview
- Contention factors
- Observations on application behavior
- A simple prediction method
- Intuition
Contestion Effects Prediction

- **Step #1**: performance drop curve for each app
  - Synthetic competitors – random cache reads
  - Vary competitors’ cache refs/sec

- **Step #2**: cache refs/sec for each app running alone

- **Step #3**: predicted drop equals the value of the drop curve corresponding to the competing cache refs/sec

**Simple offline profiling**
Step by Step Prediction

Simple offline profiling
Evaluation

Contension effects are predictable
Outline

- System overview
- Contention factors
- Observations on application behavior
- A simple prediction method
- Intuition
**The Intuition**

- **Obs. #1**: competitors’ cache refs/sec determine drop
  - Aggregate data exceeds cache size
  - 3MB shared cache/core
The Intuition

- Obs. #1: competitors’ cache refs/sec determine drop
  - Aggregate data exceeds cache size
  - 3MB shared cache/core
- Obs. #2: drop curve grows slowly after certain point
  - Most damage happens early on
  - Simple probabilistic analysis
Conclusion

- It is feasible to build a packet-processing platform with predictable performance using simple techniques
  - 3% prediction error

- Contention-aware job placement does not bring significant benefit to the overall performance
  - 2% potential improvement