Octopus-Man

QoS-Driven Task Management for Heterogeneous Multicore in Warehouse Scale Computers

Vinicius Petrucci (UFBA)*, Michael Laurenzano, John Doherty, Yunqi Zhang (UMich), Daniel Mossé (PITT), Jason Mars, Lingjia Tang (UMich)

* Work done while the author was a post-doc at UMich

The 21st IEEE International Symposium on High Performance Computer Architecture (HPCA)
February 2015 — Bay Area, CA
Warehouse Scale Computers (WSC)

Computation shifting to the “cloud”

Google data center in Douglas County, Georgia
Typical WSC workload

Load fluctuation and power consumption of Web-search running on Google servers *

Energy consumption is not proportional to the amount of computation!

* Meisner et al. Power management of online data-intensive services. ISCA 2011
Opportunity: heterogeneous multicore

- Heterogeneous multicore (Wimpy + Brawny cores)
  - Power efficiency improvement
  - **Real system** evaluation on Intel QuickIA (Atom + Xeon)

![Graph showing QPS/Watt comparison for different workload levels (5QPS, 10QPS, 20QPS, 30QPS, 45QPS)]

*Wimpy cores can be 7-13x more power-efficient than Brawny cores*
Opportunity: heterogeneous multicores

- What about performance (e.g., tail latency)?

Web-search running on Intel QuickIA

Brawny cores achieve lower latency at all load levels

But wimpy cores can still meet the QoS at low load using much less power!
Opportunity: heterogeneous multicores

Insight: Exploit \textit{load fluctuation} to improve energy efficiency and meet QoS

- **Low load**: Wimpy cores to reduce power with satisfactory QoS
Opportunity: heterogeneous multicores

Insight: Exploit load fluctuation to improve energy efficiency and meet QoS

- **High load**: Brawny cores to guarantee QoS
Octopus-Man: Goal

- To **guarantee quality of service** (e.g., bounding tail latency) while **maximizing energy efficiency**

... but this is **not** a trivial task!

Naive design of tasking mapping/migrations on heterogenous multicore can cause **significant QoS violations**
Octopus-Man: Challenges

• Tension between **responsiveness** and **stability**

  • **Responsiveness**
    - *react quickly* to capture load fluctuations and migrate tasks accordingly to meet QoS

  • **Stability**
    - *do not over-react* because it can cause oscillatory behavior and hurt the QoS
Responsiveness and stability

**Fast reaction**

- **Core mappings**
  - 1 wimpy
  - 2 wimpy
  - 3 wimpy

- **QoS target**
  - 1 W 2W 1W 2W 1W 2W

- **Latency**
  - t1
  - t2
  - t3

**Over-reaction**

- **Core mappings**
  - 1 W 2W 1W 2W 1W 2W

- **QoS target**
  - 1 wimpy 2 wimpy 3 wimpy

- **Latency**
  - t1
  - t2
  - t3

**Slow reaction...**

- **Core mappings**
  - 1 wimpy
  - 2 wimpy

- **QoS target**
  - t1'
  - t2'

**QoS violations!**

**Over-reaction**

- **Core mappings**
  - 1 W 2W 1W 2W 1W 2W

- **QoS target**
  - 1 wimpy 2 wimpy 3 wimpy

- **Latency**
  - t1''
  - t2''
  - t3''

**QoS violations!**
Octopus-Man: Solution

- **Octopus-Man monitor**
  - Application-level latency monitoring

- **Octopus-Man Mapper**
  - Task-to-core management for QoS guarantee and energy efficiency
Octopus-Man Mapper: Designs

1) PID control system
   • **pros**: well-known control methodology
   • **cons**: parameter tuning via extensive offline app profiling

2) Deadzone-based control system
   • **pros**: simple online scheme based on QoS thresholds
   • **cons**: sensitive to threshold parameter selection

• Can either effectively provide high QoS while maximizing energy efficiency?
  • Responsiveness and Stability
Design 1: PID control system

**GOAL:** To keep the **controlled system** running as close as possible to its specified QoS target

QoS target (e.g., 90%-tile latency)

QoS monitored

\[ e(t) = r(t) - y(t) \]

Controller

\[ u(t) = K_p e(t) + K_i \sum e(t) + K_d \frac{de(t)}{dt} \]

Controlled System

Latency-sensitive application

Het. multicore

Computing resources
PID Control Mapping

- Task-to-core mapping
  - Mapping from the continuous PID output to a discrete task-core mapping

- Parameter selection/tuning
  - Classical control system method, root locus (Hellerstein et al. 2004), is used to determine $K_p$, $K_i$, $K_d$ parameter
    - Responsiveness and stability
PID control: web-search

- QoS
- Core Mapping
- Throughput

Violations

90%-ile latency (ms)

QoS measured & QoS target

Workload Execution (s)

QPS
Design 2: Deadzone State Machine

QoS alert: QoS variable > QoS target * UP_THR
QoS safe: QoS variable < QoS target * DOWN_THR

The deadzone thresholds impact the stability of the mapping algorithm!
Stability: selecting deadzone parameters

Web-search execution with UP thr=0.8, DOWN thr=0.3

QoS
Core Mapping
Throughput

High QoS violations occur due to oscillatory behavior!
Solution: Dynamic deadzone selection

QoS target

Up\_thr = 0.8

Down\_thr = 0.6
Solution: Dynamic deadzone selection

- **QoS alert! (increase computing capacity)**
  
  - QoS target
  - $\text{Up\_thr} = 0.8$
  - $\text{Down\_thr} = 0.6$

Diagram showing the time (t1) and QoS target with thresholds $\text{Up\_thr} = 0.8$ and $\text{Down\_thr} = 0.6$. The graph indicates a QoS alert at time t1, suggesting a need to increase computing capacity.
Solution: Dynamic deadzone selection

- **QoS target**
- **Up_thr = 0.8**
- **Down_thr = 0.6**

*QoS safe! — but just after a QoS alert*
Solution: Dynamic deadzone selection

QoS alert again! — Oscillatory behavior!

QoS target

Up_thr = 0.8
Solution: Dynamic deadzone selection

Decrease $Down_{thr}$
Solution: Dynamic deadzone selection

With small probability (e.g., 1%) increase $\text{Down\_thr}$
Stability: Dealing with settling time

Websearch
QoS target = 500ms (90%-ile)

Memcached
QoS target = 1ms (95%-ile)

Do not reconfigure the system during the course of task migration (gray area)!
Evaluation
Experimental Platform: Intel QuickIA

Wimpy core socket
- Atom
- Atom
- L2 cache (1MB)

Brawny core socket
- Xeon
- Xeon
- L2 cache (6MB)

Front side bus

Memory
All-brawny (Static) baseline: Web-search

QoS

Core Mapping

2 Brawny

Throughput

Latency slack!
PID vs Deadzone: **web-search**

**PID control**

**Deadzone control**
*(adaptive threshold)*
QoS results

- Static (Brawny)
- Octopus-Man (PID)
- Static (Wimpy)
- Octopus-Man (Deadzone)

Web-search

- 100%
- 75%
- 50%
- 25%
- 0%

Memcached

- 100%
- 75%
- 50%
- 25%
- 0%
Energy reduction

Octopus-Man (Deadzone)

- Web-search: 21%
- Memcached: 31.5%

CPU

Octopus-Man (Deadzone)

- Full-system: 15%

Full-system
Conclusion

- Octopus-Man: task management solution exploring heterogeneous multicores
  - challenges addressed on responsiveness and stability
- Evaluation on real heterogeneous platform (Intel QuickIA)
  - Web-search and Memcached workloads
- Energy improvement of up to 41% (CPU) and 15% (full-system) over all-brawny homogeneous multicores
  - Batch processing throughput improvement of 34% (mean) and 50% (max)
Thanks!