An Empirical Model for Predicting Cross-Core Performance Interference on Multicore Processors

Jiacheng Zhao
Institute of Computing Technology, CAS

In Conjunction with Prof. Jingling Xue,
UNSW, Australia
Sep 11, 2013
Problem – Resource Utilization in Datacenters

- How?

Figure 1: Server Utilization Histogram. Real data centers are under 20% utilized.
Problem – Resource Utilization in Datacenters

- Co-located applications
  - Contention for shared cache, shared IMC, etc.
  - Negative and unpredictable interference

- Two types of applications
  - Batch – No QoS guarantees
  - Latency Sensitive - Attain high QoS

- Co-location is disabled
  - Low server utilization

- Lacking the knowledge of interference

2013/9/11
Problem – Resource Utilization in Datacenters

- Co-located applications
  - Contention for shared cache, shared IMC, etc.
  - Negative and unpredictable interference

- Two types of applications
  - Batch – No QoS guarantees
  - Latency Sensitive - Attain high QoS

- Co-location is disabled
  - Low server utilization

- Lacking the knowledge of interference

2013/9/11
Problem – Resource Utilization in Datacenters

- Co-located applications
  - Contention for shared cache, shared IMC, etc.
  - Negative and unpredictable interference

- Two types of applications
  - Batch – No QoS guarantees
  - Latency Sensitive - Attain high QoS

- Co-location is disabled
  - Low server utilization

- Lacking the knowledge of interference

2013/9/11

Figure: Task placement in datacenters
Our Goals: Predicting the interference

- **Quantitatively** predict the cross-core performance interference
- Applicable for *arbitrarily* co-locations
- Identify any “safe” co-locations
- Deployable for datacenters
Our Intuition – Mining a model from large training data

<table>
<thead>
<tr>
<th>Application</th>
<th>Co-Runners</th>
<th>$A_i$'s Performance Degradation</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A_1$</td>
<td>$W_{A_1,1}$</td>
<td>$PDA_{A_1, W_{A_1,1}}$</td>
</tr>
<tr>
<td>$A_1$</td>
<td>$W_{A_1, Q}$</td>
<td>$PDA_{A_1, W_{A_1, Q}}$</td>
</tr>
<tr>
<td>$A_2$</td>
<td>$W_{A_2, 1}$</td>
<td>$PDA_{A_2, W_{A_2, 1}}$</td>
</tr>
<tr>
<td>$A_2$</td>
<td>$W_{A_2, Q}$</td>
<td>$PDA_{A_2, W_{A_2, Q}}$</td>
</tr>
</tbody>
</table>

- Using machine learning approaches

Training Set
Motivation example

\[
PD_{mcf} = \begin{cases} 
0.485P_{bw} + 0.183P_{cache} - 0.138, & \text{if } P_{bw} < 3.2 \\
0.706P_{bw} + 1.725P_{cache} - 0.220, & \text{if } 3.2 \leq P_{bw} \leq 9.6 \\
0.907P_{bw} + 3.087P_{cache} - 0.561, & \text{if } P_{bw} > 9.6 
\end{cases}
\]
Outline

- Introduction
- Our Key Observations
  - Our Approach – Two-Phase Approach
  - Experimental Results
- Conclusion
Observation 1: The function depends only on the pressure on shared resources, regardless of individual pressures from one co-runner.

For an application A, \( PD_A = f(P_{cache}, P_{bw}) \)

\( (P_{cache}, P_{bw}) = g(A_1, A_2, ..., A_m) \)
Our Key Observations

- **Observation 2:**
  - The function $f$ is piecewise.
Our Key Observations

- Naively, we can create A’s prediction model using brute-force approach

- **BUT**, we can **NOT** apply brute force approach for each application!
  - Thousands of applications in one datacenter
  - Frequent software updates
  - Different generations of processors
  - Even steps for one application is expensive

- **Observation 3:**
  - The function **form** is platform-dependent and application independent
  - Only the coefficients are application-dependent

2013/9/11
Outline

- Introduction
- Our Key Observations
- Our Approach - Two-Phase Approach
- Experimental Results
- Conclusion
Our Approach - Two-Phase Approach

Phase 1: Get the abstract model

- Find a function form best suitable for all applications on a given platform
- Heavy – many training workloads
- Run once for one platform

\[ PD = \begin{cases} a_{11}P_{bw} + a_{12}P_{cache} + a_{13}, \text{subdomain1} \\ a_{21}P_{bw} + a_{22}P_{cache} + a_{23}, \text{subdomain2} \\ a_{31}P_{bw} + a_{32}P_{cache} + a_{33}, \text{subdomain3} \end{cases} \]

Phase 2: Instantiate the abstract model

- Determine the application-specific coefficients (a11, etc.)
- Light-weighted, with a small number of trainings
- Run once for one application

\[ PD_{mcf} = \begin{cases} 0.49P_{bw} + 0.18P_{cache} - 0.13, P_{bw} < 3.2 \\ 0.71P_{bw} + 1.73P_{cache} - 0.22, \text{others} \\ 0.91P_{bw} + 3.09P_{cache} - 0.56, P_{bw} > 9.6 \end{cases} \]
Our Approach - Two-Phase Approach
Our Approach - Two-Phase Approach

Q1: What are selected as application features

Q2: How?

Q3: What’s the cost of the training?
Our Approach – Some Key Points

Q1: What are selected as application features?

- Runtime profiles
- Shared cache consumption
- Bandwidth consumption
Our Approach – Some Key Points

Q2: How to create the abstract model?

- Regression analysis
- Configurable
  - Each configuration binding to a function form
- Searching for the best function form for all applications in the training set

<table>
<thead>
<tr>
<th>Application</th>
<th>Co-Runners</th>
<th>$A_i$'s Performance Degradation</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A_1$</td>
<td>$W_{A_1,1}$</td>
<td>$PDA_{A_1, W_{A_1,1}}$</td>
</tr>
<tr>
<td>$A_1$</td>
<td>$W_{A_1,Q}$</td>
<td>$PDA_{A_1, W_{A_1,Q}}$</td>
</tr>
<tr>
<td>$A_2$</td>
<td>$W_{A_2,1}$</td>
<td>$PDA_{A_2, W_{A_2,1}}$</td>
</tr>
<tr>
<td>$A_2$</td>
<td>$W_{A_2,Q}$</td>
<td>$PDA_{A_2, W_{A_2,Q}}$</td>
</tr>
</tbody>
</table>

#Aggregation
#Pre-Processing: none/exp(p)/log(p)/pow(p)
#Mode: add/mul
#Domain Partitioning: (shared-resource$_1$, condition$_1$), …
#Function: linear/polynomial(p)/user-defined
Our Approach – Some Key Points

Q3: What’s the cost of the training when instantiation

- Cover all sub-domains of the piecewise function, say S
- Constant points for each sub-domain, say C
  - The constant depends on the form of abstraction model
- C*S training runs in total

- Usually C and S are small, our experience: C=4, S=3
Outline

- Introduction
- Our Key Observations
- Our Approach - Two-Phase Approach
- Experimental Results
- Conclusion

2013/9/11
Experimental Results

- Accuracy of our two-phase regression approach
  - Prediction precision
  - Error analysis
- Deployment in a datacenter
  - Utilization gained
  - QoS enforced and violated
Experimental Results

- **Benchmarks:**
  - SPEC2006
  - Nine real-world datacenter applications
    - Nlp-mt, openssl, openclas, MR-iindex, etc.
- **Platforms:**
  - Intel quad-core Xeon E5506 (main)
- **Datacenter:**
  - 300 quad-core Xeon E5506
Some Predictor Function

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Predictor Function</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>400.perlbench</td>
<td>$0.108P_{bw} + 0.484P_{cache} + 0.003$</td>
<td>$P_{bw} &lt; 3.2$</td>
</tr>
<tr>
<td></td>
<td>$0.115P_{bw} + 0.460P_{cache} + 0.001$</td>
<td>$3.2 \leq P_{bw} \leq 9.6$</td>
</tr>
<tr>
<td></td>
<td>$0.176P_{bw} + 0.336P_{cache} - 0.026$</td>
<td>$P_{bw} &gt; 9.6$</td>
</tr>
<tr>
<td>401.bzip2</td>
<td>$0.422P_{bw} + 1.337P_{cache} - 0.007$</td>
<td>$P_{bw} &lt; 3.2$</td>
</tr>
<tr>
<td></td>
<td>$0.438P_{bw} + 0.714P_{cache} + 0.018$</td>
<td>$3.2 \leq P_{bw} \leq 9.6$</td>
</tr>
<tr>
<td></td>
<td>$0.445P_{bw} + 1.240P_{cache} - 0.046$</td>
<td>$P_{bw} &gt; 9.6$</td>
</tr>
<tr>
<td>433.milc</td>
<td>$0.403P_{bw} + 0.752P_{cache} - 0.154$</td>
<td>$P_{bw} &lt; 3.2$</td>
</tr>
<tr>
<td></td>
<td>$0.935P_{bw} + 1.124P_{cache} - 0.719$</td>
<td>$3.2 \leq P_{bw} \leq 9.6$</td>
</tr>
<tr>
<td></td>
<td>$P_{bw} &gt; 9.6$</td>
<td></td>
</tr>
<tr>
<td>435.gromacs</td>
<td>$0.093P_{bw} + 0.430P_{cache} - 0.015$</td>
<td>$P_{bw} &lt; 3.2$</td>
</tr>
<tr>
<td></td>
<td>$0.129P_{bw} + 0.405P_{cache} - 0.028$</td>
<td>$3.2 \leq P_{bw} \leq 9.6$</td>
</tr>
<tr>
<td></td>
<td>$0.154P_{bw} + 0.297P_{cache} - 0.033$</td>
<td>$P_{bw} &gt; 9.6$</td>
</tr>
<tr>
<td>471.omnetpp</td>
<td>$0.355P_{bw} + 2.044P_{cache} - 0.080$</td>
<td>$P_{bw} &lt; 3.2$</td>
</tr>
<tr>
<td></td>
<td>$0.648P_{bw} + 1.280P_{cache} - 0.126$</td>
<td>$3.2 \leq P_{bw} \leq 9.6$</td>
</tr>
<tr>
<td></td>
<td>$0.843P_{bw} + 1.012P_{cache} - 0.222$</td>
<td>$P_{bw} &gt; 9.6$</td>
</tr>
</tbody>
</table>
Prediction precision for SPEC Benchmarks

Prediction Error: Average 0.2%, from 0.0% to 8.6%.
Prediction precision for datacenter applications

- 15 workloads for each datacenter applications

- Prediction Error: Average 0.3%, from 0.0% to 5%.
Error Distribution

Error Distribution

2013/9/11
Prediction Efficiency

- **Precision**
  - Two-Phase: 0.0~11.7%, Average: 0.40%
  - Brute-Force: 0.0~10.1%, Average: 0.23%

- **Efficiency**
  - co-running: ~200 → 12
Benefits of piecewise predictor functions
Benefits of piecewise predictor functions
Deployment in a datacenter

- 300 quad-core Xeon
  - 1200 tasks when fully occupied
- Applications
  - Latency sensitive: Nlp-mt
    - machine translation
    - 600 dedicated cores, 2/chip
  - Batch job
    - 600 tasks, kmeans, MR
- Our Purpose
  - QoS policy
  - Issue batch jobs to idle cores
Cross-platform applicability

- Six-core Intel Xeon

- Prediction Error: Average 0.1%, range from 0.0% to 10.2%
Cross-platform applicability

- Quad-core AMD

Prediction Error: Average 0.3%, range from 0.0% to 5.1%
Outline

- Introduction
- Our Key Observations
- Our Approach - Two-Phase Approach
- Experimental Results
- Conclusion
Conclusion

- An empirical model, based on our key observations
- Using aggregated resource consumptions to create the predictor function, thus working for *arbitrarily* co-locations
- Piecewise is reasonable and effective
- Breaking the model creation into two phases, for efficiency
Thanks
How to make the training set representative?

- Partition the space into grids
- Sample for each grid
How to do domain partitioning?

- Specified in configuration file
- Syntax: \((\text{shared resource}_i, \text{condition}_i)\), e.g. \((P_{bw}, \text{equal}(4))\)
- Empirical knowledge to perform this task

### Aggregation
- #Pre-Processing: none, exp(2), log(2), pow(2)
- #mode: add, mul
- #Domain Partitioning: \{\(\{(P_{bw}, \text{equal}(4))\), \(\{(P_{cache}, \text{equal}(4))\), \(\{(P_{cache, P_{bw}}, \text{equal}(4, 4))\}\}\)
- #Function: linear, polynomial(2)
Backup slides

- Two sources of error:
  - Estimation for shared resources consumption
  - L2 LinesIn
  - Phase behavior of applications