Modelling Systems Performance using Machine Learning Techniques

Rui Zhang
Computing Laboratory, University of Oxford
Enterprise Systems Department, IBM T.J. Watson Research Center (2005 – 2006)
Outline

• Background and Motivations
• Component-level Modelling
• Break (Questions)
• Code-level Modelling
• Conclusions and Future Work
• Questions
I. Background and Motivations
Enormous System Size

- Consumer demand has caused data-center growth

- Appliance server growth has complicated the application topology

*IDC
Complex Application Topologies
Large Code, Rich Instruction Set
Complex Machine Architecture
Performance Modeling in Complex Environments is Hard, But…

• Facilitates *autonomic* system management:
  – Dynamic resource provisioning/allocation
  – Problem determination

• Generates better code
  – Software performance debugging
  – Compiler optimization

• Ultimately brings faster response/execution time:
  – Subscribers: violating SLAs – $$$ penalty !!
  – Casual users: better experience
The Machine Learning Approach

• Analytical models:
  – Require substantial human efforts
  – Time-consuming to build
  – Difficult to acquire for certain domains
  – May not be adaptive to changes

• Machine learning to the rescue!
II. Component-level Modeling through Probabilistic Learning
Workload Monitoring for An eDiaMoND Scenario

Management Server

Internet

Remote Hospital

image_retrieve Service

DB query

Image_R

image_R

database Service

Local Hospital

image_list Service

Username

ID_L, ID_R

image_R

work_list Service

ID_L

Image_L

Globus

Tomcat

eDiaMoND Service

Radiologist

Request from previous service

Response to next service

Request to next service

Response from previous service

Request from previous service

Response to previous service
SOA Performance Can Be Hard to Understand….

- Complications
  - Complex, dynamic topology
  - Missing data
- Analytical models (e.g. queuing networks [2, 3]):
  - Can make strong assumptions
  - May not be reactive to system changes

The Machine Learning Problem

• GOAL: Mapping component behaviours to end-to-end performance states (e.g. response time)

• Challenges
  – Probabilistic SLA goals and behaviours
  – *Fast model building with small training data sets*
  – Accuracy
  – Model comprehensibility

• Existing statistically learned models[^1,^2] can be:
  – Computationally expensive
  – Data-intensive

[^1]: Cohen et. al. Correlating instrumentation data to system states: a building block for automated diagnosis and control, OSDI’04
[^2]: Chen et. al. Autonomic Provisioning of Backend Databases in Dynamic Content Web Servers, IEEE ICAC’07
A Bayes Net (BN) Mapping Per-component Behaviour to Response Time

- $X_i$ ($i = 1, 2, \ldots, n$) represents elapsed time on component $i$;
- $D$ stands for the overall response time;

Joint Prob. Dist. (JPD)

$$P(D, X_1, X_2, \ldots, X_n) = P(D \mid \text{parents}(D)) \prod_{i=1}^{n} (X_i \mid \text{parents}(X_i))$$

Conditional Prob. Dist. * (CPD)

* Retires to Prior Prob. Dist. when parent set is empty.
Efficiency-Boosting Magic I: Determining BN structure using Workflow Knowledge

- Workflow discovery techniques abundant
  - Intrusive: GWLM\textsuperscript{[1]}, EWLM\textsuperscript{[2]}, IBM Tivoli, HP Openview
  - non-intrusive: Aurora\textsuperscript{[3]}, Constellation\textsuperscript{[4]}

- Upstream services impacting direct downstream services

---

[1] Zhang et. al., OGSA-based grid workload monitoring, IEEE CCGRID’05
[2] Bari et.al., IBM Enterprise Workflow Manager Release 2.1, IBM Redbooks’07
Efficiency-Boosting Magic II: Determining BN parameters using Workflow Knowledge

• The CPD for D can be defined using workflow information as a deterministic function $f$ with leak probability $l$.

\[
P_D(D = f(X)|X) = 1 - l
\]

\[
P_D(D \neq f(X)|X) = l
\]

• Converting eDiaMoND workflow to $f$: Parallel --> max, Sequential --> +

\[
D = X_1 + X_2 + max(X_3 + X_5, X_4 + X_6)
\]
Efficiency-Boosting Magic II: Decentralizing Parameter Learning

• Exploit the locality of data required to compute CPD for $X_i$
  – Only data regarding $X_i$ and its parents are needed.
Simulation Setup

• Customized simulator in Matlab
• 4 3.0Ghz CPU Red-hat Linux machine
• Metrics
  – Construction time
  – Accuracy log(TestData|BN)
• Average efficacy over various simulated settings
Assessing the Benefit of Domain Knowledge Incorporation

- Knowledge-enhanced BN vs. Naive BN
  - KERT-BN the winner
  - Time difference widens with more data
  - Accuracy difference with more data
Assessing the Benefit of Domain Knowledge Incorporation

- Knowledge-enhanced BN vs. Naive BN
  - KERT-BN the winner
  - Difference widens as system size gets larger
Assessing the Advantages of Decentralized Parameter Learning

- Decentralized learning vs. centralized learning
  - KERT-BN the winner
  - Difference widens as system size gets larger
III. Code-level Performance Modelling through Relational Learning
# Address Generation Interlock (AGI)
## – A Famous Pipeline Stall

<table>
<thead>
<tr>
<th></th>
<th>T1</th>
<th>T2</th>
<th>T3</th>
<th>T4</th>
<th>T5</th>
<th>T6</th>
<th>T7</th>
<th>T8</th>
<th>T9</th>
</tr>
</thead>
<tbody>
<tr>
<td>D</td>
<td>AG</td>
<td>A</td>
<td>E</td>
<td>PA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>AG</td>
<td>A</td>
<td>E</td>
<td>PA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>(AG)</td>
<td>(AG)</td>
<td>AG</td>
<td>A</td>
<td>E</td>
<td>PA</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **D** – Decode, read registers for address generation
- **AG** – Address generation
- **(AG)** - AGI
- **A** – Cache access
- **E** – Execute
- **PA** – Put away.

- **L** – load memory location
- **gpr** – general purpose register

<table>
<thead>
<tr>
<th></th>
<th>L gpr7, 17(gpr6)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>L gpr3, 112(gpr4)</td>
</tr>
<tr>
<td></td>
<td>L gpr1, 96(gpr7)</td>
</tr>
</tbody>
</table>
Address Generation Interlock (AGI)

<table>
<thead>
<tr>
<th></th>
<th>T1</th>
<th>T2</th>
<th>T3</th>
<th>T4</th>
<th>T5</th>
<th>T6</th>
<th>T7</th>
<th>T8</th>
<th>T9</th>
</tr>
</thead>
<tbody>
<tr>
<td>D</td>
<td>E</td>
<td>PA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>AG</td>
<td>A</td>
<td>E</td>
<td>PA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>AG</td>
<td>A</td>
<td>E</td>
<td>PA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

LR gpr7, gpr6
L gpr3, 112(gpr4)
L gpr1, 96(gpr7)

D – Decode, read registers for address generation
AG – Address generation, send address to cache
(AG) - AGI
A – Cache access
E – Execute
PA – Put away.
### Address Generation Interlock (AGI)

<table>
<thead>
<tr>
<th>T1</th>
<th>T2</th>
<th>T3</th>
<th>T4</th>
<th>T5</th>
<th>T6</th>
<th>T7</th>
<th>T8</th>
<th>T9</th>
</tr>
</thead>
<tbody>
<tr>
<td>D</td>
<td>AG/AB</td>
<td>A</td>
<td>E</td>
<td>PA</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>AG</td>
<td>A</td>
<td>E</td>
<td>PA</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>AG</td>
<td>A</td>
<td>E</td>
<td>PA</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **D** – Decode, read registers for address generation
- **AG/AB** – Address generation, write address to bypass stack
- **(AG)** - AGI
- **A** – Cache access
- **E** – Execute
- **PA** – Put away.

Instructions:
- LA gpr7, 17(gpr6)
- L gpr3, 112(gpr4)
- L gpr1, 96(gpr7)
Address Generation Interlock (AGI)

D – Decode, read registers for address generation
AG – Address generation, send address to cache
(AG) - AGI
A – Cache access
E – Execute
PA – Put away.

<table>
<thead>
<tr>
<th></th>
<th>T1</th>
<th>T2</th>
<th>T3</th>
<th>T4</th>
<th>T5</th>
<th>T6</th>
<th>T7</th>
<th>T8</th>
<th>T9</th>
</tr>
</thead>
<tbody>
<tr>
<td>D</td>
<td>AG</td>
<td>A</td>
<td>E</td>
<td>PA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>AG</td>
<td>A</td>
<td>E</td>
<td>PA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>(AG)</td>
<td>(AG)</td>
<td>AG</td>
<td>A</td>
<td>E</td>
<td>PA</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

L gpr7, 17(gpr6)
L gpr3, 112(gpr4)
ST gpr1, 96(gpr7)
Flagging AGI is not easy…

• Complications
  – Instruction distance
  – Early execution
  – Hardware bypass
  – Many more……

• State of the Art was
  – Experienced OS core developers scan code
  – Machine experts write, debug and maintain a (incomplete) set of rules
  – Compilers throw in NOPs
The Machine Learning Problem

• GOAL: Mapping instruction sequence patterns to performance hazards (e.g. AGIs)

• Challenges:
  – Relational, multi-dimensional data
  – Data set can be large (no training time limit)
  – Accuracy
  – Model comprehensibility
Inductive Logic Programming (ILP) Preliminaries

- Models are represented as rules (*executable* logic programs).
- Models can represent arbitrarily complex structures.
- Models easy for humans to comprehend.
- *But* bad with numbers and computationally expensive.

\[
\text{GrandFather}(X, Y) :\neg \text{Father}(X, Z), \text{Parent}(Z, Y).
\]

\[
\text{Father}(X, Y) :\neg \text{Father}(X, Z).
\]
ILP in Action for AGI Modeling

Domain knowledge

Positive Sequence, Positive Sequence, Positive Sequence N

Negative Sequence, Negative Sequence, Negative Sequence M

ILP Engine

Rule 1, Rule 2, ......
Representing An Instruction in ILP

- Instruction addr. and a loop counter as unique ID
- One predicate per *relevant* inst-specific property
- Easily interpretable predicates

Encoding instruction: L gpr7, 17(gpr6)

```
instruction_1_is_previous_to_2('01-3BC9C90A', '01-3BC9C90E').
opcode('01-3BC9C90E', 1).
instruction_1_has_2_as_3('01-3BC9C90E', gpr7, r1).
instruction_1_has_2_as_3('01-3BC9C90E', gpr6, b2).
instruction_1_has_disp_2_as_3('01-3BC9C90E', 17, d2).
instruction_1_writes_into_register_2('01-3BC9C90E', gpr7).
Instruction_1_generates_an_address_using_register_2('01-3BC9C90E', gpr6).
```
Encoding An Instruction Seq. in ILP

- Pointer predicates chaining instructions together
- Reasonable seq. length to avoid state explosion

Representing sequence:

L gpr7, 17(gpr6)
L gpr3, 112(gpr4)
ST gpr1, 96(gpr7)

instruction_1_is_previous_to_2('00-00000000', '01-3BC9C90A').
opcode('01-3BC9C90A',l).

......

instruction_1_is_previous_to_2('01-3BC9C90A', '01-3BC9C90E').
opcode('01-3BC9C90E',l).

......

instruction_1_is_previous_to_2('01-3BC9C90E', '01-3BC9C912').
opcode('01-3BC9C912',st).
Incorporating Background Knowledge

**Simple Rules**

\[\text{instruction}_1\text{ executes}_\text{early}(X):=\]
\[\text{opcode}(X,\text{ltr}); \text{opcode}(X,\text{lr}); \text{opcode}(X,\text{ar}); \text{opcode}(X,\text{alr})\ldots\]

**More Complex Rules**

\[\text{instruction}_1\text{ is_a_predecessor_of}_2(X,Y):=\]
\[\quad \text{instruction}_1\text{ is_previous_to}_2(X,Y).\]

\[\text{instruction}_1\text{ is_a_predecessor_of}_2(X,Y):=\]
\[\quad \text{instruction}_1\text{ is_a_predecessor_of}_2(X,Z),\]
\[\quad \text{instruction}_1\text{ is_previous_to}_2(Z,Y).\]
Results

With No Background Knowledge:
Null

With Weak Background Knowledge:
instruction_1_agi(A):-
    instruction_1_generates_an_address_using_operand_2(A, P),
    instruction_1_is_a_predecessor_of_2(B,A),
    instruction_1_writes_into_register_2(B, P),
    instruction_1_has_2_as_3(B,X,d2).

True Pos: >90%
True Neg: >90%

With Strong Background Knowledge:
instruction_1_agi_due_to_2(A,B):-
    instruction_1_generates_an_address_using_operand_2(A, P),
    instruction_1_is_a_predecessor_of_2_with_distance_3(B,A,F),
    instruction_1_writes_into_register_2(B, P),
    instruction_1_has_agi_shadow_2(B, G),
    F <= G.

True Pos: >99.9%
True Neg: >99.9%
Performance Test

- Percentage of getting positive/negative examples right
- Average over 10 runs
Accuracy Stability Test

- Percentage of getting positive/negative examples right
- Average over 10 runs
Application I: Performance Hazard Tool for IBM OS Developers
Application II: Compiler Optimization

<table>
<thead>
<tr>
<th>Inst Address</th>
<th>Opcode</th>
<th>Instruction</th>
<th>EXEC</th>
<th>AGI</th>
<th>AGI Cause</th>
<th>Total Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000000000000001</td>
<td>LR</td>
<td>18F1</td>
<td>1</td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>0000000000000002</td>
<td>LR</td>
<td>1810</td>
<td>1</td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>0000000000000003</td>
<td>LA</td>
<td>41101000</td>
<td>1</td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>0000000000000004</td>
<td>L</td>
<td>5830F028</td>
<td>1</td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>0000000000000005</td>
<td>MH</td>
<td>4C30031A</td>
<td>2</td>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>0000000000000006</td>
<td>ALR</td>
<td>1FF3</td>
<td>1</td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>0000000000000007</td>
<td>NOP</td>
<td>A70-FF77</td>
<td>1</td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>0000000000000008</td>
<td>L</td>
<td>5830P018</td>
<td>1</td>
<td>4</td>
<td>01-0000000000000006</td>
<td>5</td>
</tr>
<tr>
<td>0000000000000009</td>
<td>L</td>
<td>5830F01C</td>
<td>1</td>
<td>4</td>
<td>01-0000000000000006</td>
<td>5</td>
</tr>
<tr>
<td>000000000000000A</td>
<td>CLR</td>
<td>1531</td>
<td>1</td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>000000000000000B</td>
<td>BRZ</td>
<td>A70-FF5B</td>
<td>1</td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>000000000000000C</td>
<td>ST</td>
<td>50301000</td>
<td>1</td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>000000000000000D</td>
<td>LA</td>
<td>41002001</td>
<td>1</td>
<td>2</td>
<td>01-0000000000000008</td>
<td>3</td>
</tr>
<tr>
<td>000000000000000E</td>
<td>CD6</td>
<td>BB20F018</td>
<td>7</td>
<td></td>
<td></td>
<td>7</td>
</tr>
<tr>
<td>000000000000000F</td>
<td>BRNZ</td>
<td>A77-FFF3</td>
<td>1</td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>0000000000000010</td>
<td>BSM</td>
<td>080E</td>
<td>11</td>
<td></td>
<td></td>
<td>11</td>
</tr>
</tbody>
</table>
Extension I: Higher-level Code

• Buffer overflow attack
• Java Byte code execution
• SQL queries
• Java/C code
  – Deadlock
  – Verification
Extension II: Meta-instructions in Distributed Systems

- Consider distributed system components as macro-instructions and use ILP to identify problematic component patterns

```
WEB property 1, property 2  L gpr7, 17(gpr6)
APP property 1, property 2  L gpr3, 112(gpr4)
DB property 1, property 2  L gpr1, 96(gpr7)
```
IV. Final Words
Conclusions

• Analyzing complex systems (performance) is important but very hard (for humans), yet machine learning can help.

• Domain knowledge can largely
  – Reduce training time
  – Improve end model accuracy

<table>
<thead>
<tr>
<th></th>
<th>BN</th>
<th>ILP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Problem nature</td>
<td>Probabilistic</td>
<td>Deterministic</td>
</tr>
<tr>
<td>Training data</td>
<td>Single table</td>
<td>Multi table</td>
</tr>
<tr>
<td>Representation</td>
<td>Graph</td>
<td>Rules</td>
</tr>
<tr>
<td>Domain knowledge support</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Future Work

• Plotting accuracy against the amount of domain knowledge.
• Autonomous domain knowledge incorporation.
• Linking low-level models with high-level ones (e.g. through probabilistic relational models[1])