WebProphet: Automating Performance Prediction for Web Services

Zhichun Li, Ming Zhang, Zhaosheng Zhu, Yan Chen, Albert Greenberg and Yi-min Wang
Lab of Internet and Security Technology (LIST)
Northwestern University
Microsoft Research
Web Services Are Prevalent

• Almost everything is related to Web
  – Web search
  – Web mail
  – Online shopping
  – Online Social network
  – Calendar
Performance Is Important

- Amazon: 100ms extra delay $\rightarrow$ 1% sale loss
- Google search results: 500 ms extra delay $\rightarrow$ reduce display ads revenues by up to 20%
Web Services Are Complicated

- Example of Yahoo Maps
  - 110 embedded objects
  - Complex object dependencies
  - 670KB JavaScript
  - Hosted by multiple data-centers around the world
Performance Optimization is Hard

A large number of possible optimization strategies

User perceived PLT: whole page or the portion with most visual effects
Limitations with Existing Techniques

• A/B test (controlled experiments)
  – Idea: set up an experiment setting and try on a group of users
  – Problems with A/B test
    • Hard to fully automated
    • Expensive to set up
    • Quite slow!
Limitations with Existing Techniques

• Service provider based techniques (WISE SIGCOMM2008)
  – Problems
    • multiple data sources
    • Object dependencies
    • Client side delays, e.g. JavaScript execution time

• Regression based techniques (LinkGradient INFOCOM2009)
  – Usually require the independence assumption on delay factors of each object. **Problematic!**
Our Contributions

• A tool for automated performance prediction
  – Fast prediction on the user perceived performance
  – Timing perturbation based dependency discovery
  – Dependency driven page load simulation
Outline

• Motivation & Design
• **Dependency Extraction**
• Performance Prediction
• Implementation
• Evaluation
• Conclusion
Why Are Dependency Discovery Difficult?

• Simple HTML parsing/DOM traversal is not enough
  – Object requests generated by JavaScript depend on the corresponding .JS files
  – Event triggers, such as when image B trigger “onload” event, then image A will be load by JavaScript

• Extensive browser instrumentation is heavy-weight and browser dependent
Our Approach

- **Goal:**
  - Light-weight black box based approach
  - Browser independent

- **Timing perturbation based technique**
  - Inject delay
  - See how delay propagate.

Objects depend on X
Take Care HTML Objects

• Regular Objects
  – Regular objects have to be fully loaded before their descendants

X --- Y

• HTML Objects are special
  – HTML is stream objects, allowing incremental rendering

X --- Y
Measure the Offset
Outline

• Motivation
• Design
• Dependency Extraction
• **Performance Prediction**
• Implementation
• Evaluation
• Conclusion
Performance Prediction Problem

- Evaluate different new scenarios
Performance Prediction Procedure

1. Packet Trace
2. Dep. Graph
3. Extract Object timing information
4. Annotate client delay
5. Adjust each of object according to new scenario
6. New Scenario
7. Dep. Graph
8. Simulate the page load process
Extract Object Timing information

- Extract Timing from packet traces
- Basic object timing info

- **DNS**
  - DNS lookup time

- **TCP**
  - TCP handshaking time

- **HTTP**
  - Request transfer time
  - Reply transfer time
Annotate client delay

- Browser processing time after dependency solved
Adjust Object Timing Info

- Consider four delay factors: client delay, server delay, RTT and DNS lookup time
- Adjust timing
  - Adjust Client delay, DNS lookup time, and server response time directly
  - RTT: adjust $\Delta RTT \times$ number of round trips
Factors Affected Object Loading

- Add DNS lookup time based on DNS cache
- Add TCP handshaking time for new connections
- Add TCP waiting time when all connections are not available
Simulate Page Load Process

Object Queue

A

A

B

C

D

E

F
Simulate Page Load Process

Object Queue

A

B

C

D

E

F

A

B

C

D

E

F
Simulate Page Load Process

Object Queue

A

B

C

D

A

B

C

D

E

F
Simulate Page Load Process

Object Queue

A

D

E

B

C

A

B

C

D

E

F
Simulate Page Load Process

Object Queue

A
B
C
D
E

A → B → C → D → E

A
B
C
D
E
F
Simulate Page Load Process

Object Queue

A
B
C
D
E
F

A -> B -> C -> D -> E -> F

A
B
C
D
E
F
Simulate Page Load Process

Object Queue

New page load time
Outline

• Motivation
• Design
• Dependency Extraction
• Performance Prediction
• Implementation
• Evaluation
• Conclusion
WebProphet Framework

Agent network → Traces

Performance Predictor
- Trace Analyzer
- Annotate object timing info
- Page simulator

Results

New scenario input

Dep graphs

Dependency Extractor
- Web Agent
- Web Proxy

Application transaction script snippet
- Web robot
- Scripting API
- Control plug-in
- Browser

Pcap trace logger
Outline

• Motivation
• Design
• Dependency Extraction
• Performance Prediction
• Implementation
• Evaluation
• Conclusion
Dependency Extraction Results

• Google and Yahoo Search

  ![Diagram showing dependency for Google]

  ![Diagram showing dependency for Yahoo]

• Validation: manual code analysis
Dependency Extraction Results

• Google and Yahoo Maps

• Validation: create pages with the same dep. graph and validate the crafted pages
Prediction Experiment Setup

• Reduce latency see the improvement on PLT
• Controlled experiments
  – Baseline: high latency
  – New Scenario: low latency
  – Use control gateway to inject and remove delays
• Planetlab experiments
  – Baseline: International nodes
  – New scenario: US nodes
  – Improve all delay factors to be the same as the US node.
Controlled Experiment

- Setup: visit Yahoo Maps from Northwestern
- Baseline: inject 100ms RTT to one DC
- New Scenario: removing the 100ms RTT injected

<table>
<thead>
<tr>
<th>DC</th>
<th>Err (median)</th>
<th>Err (P95)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Akamai</td>
<td>16.0%</td>
<td>11.8%</td>
</tr>
<tr>
<td>YDC1</td>
<td>6.5%</td>
<td>9.7%</td>
</tr>
<tr>
<td>YDC2</td>
<td>14.8%</td>
<td>6.0%</td>
</tr>
</tbody>
</table>
Planetlab Experiment

- Baseline: A International node with relative poor performance
- New Scenario: a US node

<table>
<thead>
<tr>
<th>Service</th>
<th>Baseline</th>
<th>New</th>
<th>Err(median)</th>
<th>Err(P95)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gsearch</td>
<td>Singapore</td>
<td>US</td>
<td>2.0%</td>
<td>10.7%</td>
</tr>
<tr>
<td>Ysearch</td>
<td>Japan</td>
<td>US</td>
<td>6.1%</td>
<td>0.3%</td>
</tr>
<tr>
<td>Gmap</td>
<td>Sweden</td>
<td>US</td>
<td>1.2%</td>
<td>1.8%</td>
</tr>
<tr>
<td>Ymap</td>
<td>Poland</td>
<td>US</td>
<td>0.7%</td>
<td>1.3%</td>
</tr>
</tbody>
</table>
Usage Scenarios

• Analyze how to improve *Yahoo Maps*
  – Only want to optimize a small number of objects
  – Use a greedy based search
  – Evaluate 2,176 hypothetical scenarios in 20 secs, find that
    • Move 5 objects to CDN: 14.8%
    • Reduce client delays of 14 objects to half: 26.6%
    • Combine both: 40.1% (4secs to 2.4secs)
Outline

• Motivation
• Design
• Dependency Extraction
• Performance Prediction
• Implementation
• Evaluation
• Conclusion
Conclusions

• Web service performance prediction is hard
  – Modern web services are complicated
  – Object dependencies are very important

• Design an automated tool for performance prediction
  – Dependency discovery
  – Dependency driven performance predication
  – Evaluation on the accuracy and usefulness of our tool
Q & A

Thanks!