AjaxScope: Remotely Monitoring Client-side Web-App Behavior

Emre Kıcımınan
emrek@microsoft.com

Ben Livshits
livshits@microsoft.com

Internet Services Research Center
Microsoft Research

Runtime Analysis & Design
Microsoft Research
A Web Application
A Web Application

- Talks to >14 backend services (traffic, images, search, directions, ads, ...)
- ~ 1 MB code
- 70k lines of JavaScript code downloaded to the client.
- 2855 Functions
Web 1.0 → Web 2.0

Server-side computation

Static HTML

Client-side rendering
Web 1.0 → Web 2.0

Server-side computation

Static HTML

Client-side rendering

JavaScript + DHTML

Client-side computation
Web App Challenges

• Code complexity: more client-side code
  – Ex. Maps.live.com: 1MB of code, 70k LoC
  – Bugs, race conditions, memory leaks, ...

• Non-standard execution environments
  – Many APIs differ across browsers
  – Perf of simple ops vary 10x-100x across impl.

• Third-party dependencies (e.g., mash-ups)

Missing: Visibility into behavior on clients
Outline

1. Motivation
2. AjaxScope Platform
3. Expt: Adaptive instrumentation
4. Expt: Distributed instrumentation
5. Conclusions
AjaxScope Approach

**Goal:** Detailed visibility into app behavior in the client

**Approach:** On-the-fly rewriting to add instrumentation

**Key Enabler:** Instant re-deployability of web apps
Monitoring Goals

• **Performance Optimization**
  • Performance profiling
  • String optimization; cache placement; ...
  • Code splitting and joining

• **Debugging**
  • Report function arguments, app state, errors
  • Memory leak checking
  • Statistical debugging

• **Test**
  • Code coverage
  • A/B tests

• **Operations**
  • Measure RPC network latencies

• **User interaction feedback**
  • What features are being used / discovered?
Approach: JavaScript Rewriting

- Simple but powerful monitoring
  - Inspect application state
  - Observe control flow
  - Limited only by JS sandbox

- Easy deployability
  - No changes required to original web app
  - No changes to client-side browsers
Example: Record Function Args

1. Search JavaScript AST for function definitions
2. For each function definition, add a statement to report every argument.

sendLog() queues up messages for bulk reporting to AjaxScope
Deploying AjaxScope...
Server-side Deployment

Users

Logs

AjaxScope

My Service

3rd Party Service
AjaxScope Proxy

- Rewritten Web Page
- Logs
- Log Collector
- Instrumentation #1
- Instrumentation #2
- Rewriting Engine
- JS Parser

- Pluggable policies
- Controls rewriting based on incoming logs
- Platform support for adaptive and distributed instrumentation
Rewrite “On-the-fly”

• Service has tight control over code running at client
  – Clients always download new version
  – Cache-ability controlled by service

• Enables dynamic instrumentation

• Use to reduce performance overhead
  1. Adaptive instrumentation
  2. Distributed Instrumentation

• Also enables A/B tests to compare versions
Outline

1. Motivation
2. AjaxScope Platform
3. Expt: Adaptive instrumentation
4. Expt: Distributed instrumentation
5. Conclusions
Experimental Setup

- Profile **90** web sites’ “startup”
  - Client-side AjaxScope

<table>
<thead>
<tr>
<th></th>
<th>Site</th>
<th>Code Size (kB)</th>
<th># of Functions</th>
<th>Exec Time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maps</td>
<td>maps.google.com</td>
<td>295</td>
<td>1935</td>
<td>530</td>
</tr>
<tr>
<td></td>
<td>maps.live.com</td>
<td>924</td>
<td>2855</td>
<td>190</td>
</tr>
<tr>
<td>Portals</td>
<td>msn.com</td>
<td>124</td>
<td>592</td>
<td>300</td>
</tr>
<tr>
<td></td>
<td>yahoo.com</td>
<td>278</td>
<td>1097</td>
<td>670</td>
</tr>
<tr>
<td></td>
<td>google.com/ig</td>
<td>135</td>
<td>960</td>
<td>190</td>
</tr>
<tr>
<td></td>
<td>protopages.com</td>
<td>599</td>
<td>1862</td>
<td>13780</td>
</tr>
</tbody>
</table>

+ 6 more JS-heavy news & game sites
+ 78 sites randomly chosen, weighted by popularity
Adaptation: Drill-down Perf Profiling

• Naïve: Add timestamps everywhere
  – Too expensive! (both CPU and logging BW)
• Instead, auto-drill-down based on user experience

```javascript
LogTime();
FastFunc1();
FastFunc2();
SlowFunc();
LogTime();
LogTime();
FastFunc1();
FastFunc2();
LogTime();
SlowFunc();
LogTime();
LogTime();
LogTime();
LogTime();

If it’s slow
function SlowFunc() {
  // drill-down continues
}

Found it!
```
Adaptation Results

Avg 30% reduction in CPU Overhead

95% Avg Reduction in Logging Bandwidth
Outline

1. Motivation
2. AjaxScope Platform
3. Expt: Adaptive instrumentation
4. Expt: Distributed instrumentation
5. Conclusions
Monitoring for JS Memory Leaks

• Mem leaks major problem in older browsers
  – Web apps can work-around

• E.g., Circular reference across DOM + JavaScript memory heaps

• Instrumentation looks for runtime patterns indicative of leak

• Expensive! Use distribution to reduce per-user overhead
```javascript
var pipelineContainers = document.getElementById("cnnPipelineModule").getElementsByTagName("div");
...
for (var i=0; i<pipelineContainers.length; i++){
    var pipelineContainer = pipelineContainers[i];
    if(pipelineContainer.id.substr(0,9) == "plineCntr") {
        pipelineContainer.onmouseover = function () {
            CNN_changeBackgroundColor(this,1); return false;
        }
    }
}
```

First, get DOM elements

Then, set their event handlers to a new function
Example: CNN.com

var pipelineContainers =
document.getElementById("cnnPipelineModule").
getElementsByTagNames("div");
...
for (var i=0; i<pipelineContainers.length; i++){
    var pipelineContainer = pipelineContainers[i];
    if(pipelineContainer.id.substr(0,9) == "pline") {
        pipelineContainer.onmouseover = function (){
            CNN_changeBackground(this,1); return false;
        }
    }
}
Checking for Memory Leaks

Check all object assignments for potential cycles

\[
a.b = c;
\]

\[
TraverseHeapAndCheckForCycles(c, a);
\]

- Distribute expensive traversals across users
  - Each user gets random N% of checks
  - Controls per-user overhead
Distribution Gives Fine Control of Per-User Overhead

Trade-off per-user overhead vs. detection speed
Outline

1. Motivation
2. AjaxScope Platform
3. Expt: Adaptive instrumentation
4. Expt: Distributed instrumentation
5. Conclusions
Related Work

• JavaScript rewriting for safety & security
  – BrowserShield and CoreScript
• Monitoring and tracing systems
  – E.g., Magpie, Project5
• Dynamic and adaptive instrumentation
  – In parallel computing cluster: ParaDyn
• Runtime program analysis for bug finding
  – Statistical debugging, taint analysis, ...
Future Work

• Platform improvements:
  – Integrate caching considerations into rewriting
  – Limit risk of bad rewriting with meta-monitoring
  – Improved information protection

• Collecting data and analysis:
  – Compare executions across users to find outliers
  – Collect dynamic call graphs to inform smart prefetching
Conclusions

• End-to-end visibility into client-side web app
  – Requires no client-side / server-side changes
• Distribution and adaptation controls overhead
  – While maintaining high coverage & detail
• Demonstrated variety of instrumentation policies
  – Performance profiler, memory leak checker, cache placement, ...

• Download and extend the prototype
  – http://research.microsoft.com/projects/ajaxview/
  – Supports plug-ins for new instrumentation policies