JSMeter: Characterizing the Behavior of JavaScript Web Applications

Paruj Ratanaworabhan
Kasetsart University, Thailand

Ben Livshits and Ben Zorn
Microsoft Research, Redmond

in collaboration with
David Simmons, Corneliu Barsan, and Allen Wirfs-Brock
Why Measure JavaScript?

• Standardized, de facto language for the web
  – Support in every browser, much existing code

• Browser and JavaScript performance is important
  – Are current JavaScript benchmarks representative?
  – Limited understanding of JavaScript behavior in real sites

• Who cares?
  – Users, web application developers, JavaScript engine developers
### Artificial Benchmarks versus Real World Sites

#### JSMeter

<table>
<thead>
<tr>
<th>7 V8 programs:</th>
<th>8 SunSpider programs:</th>
<th>11 real sites:</th>
</tr>
</thead>
<tbody>
<tr>
<td>richards</td>
<td>3-draytrace</td>
<td>bing, Google, Gmail, Windows Live, Hotmail, amazon.com, ebay</td>
</tr>
<tr>
<td>deltablue</td>
<td>access-nbody</td>
<td></td>
</tr>
<tr>
<td>crypto</td>
<td>bitops-nsieve</td>
<td></td>
</tr>
<tr>
<td>raytrace</td>
<td>controlflow</td>
<td></td>
</tr>
</tbody>
</table>

**Goals of JSMeter Project**

- Instrument JavaScript execution and measure behavior
- Compare behavior of JavaScript benchmarks against real sites
- Consider how benchmarks can mislead design decisions
How We Measured JavaScript

Source-level instrumentation

\ie\jscript\*.cpp
custom jscript.dll

website visits
custom trace files

Offline analyzers

custom trace files

Microsoft Research
Visiting the Real Sites

• Getting past page load performance
• Attempted to use each site in “normal” way:

<table>
<thead>
<tr>
<th>Site</th>
<th>Activity Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amazon</td>
<td>Search a book, add to shopping cart, sign in, and sign out</td>
</tr>
<tr>
<td>Bing</td>
<td>Type in a search query and also look for images and news</td>
</tr>
<tr>
<td>Bingmap</td>
<td>Search for a direction from one city to another</td>
</tr>
<tr>
<td>CNN</td>
<td>Read front page news</td>
</tr>
<tr>
<td>eBay</td>
<td>Search for a notebook, bid, sing in, and sign out</td>
</tr>
<tr>
<td>Economist</td>
<td>Read front page news, view comments</td>
</tr>
<tr>
<td>Facebook</td>
<td>Log in, visit a friend pages, browse through photos and comments</td>
</tr>
<tr>
<td>Gmail</td>
<td>Sign in, check inbox, delete a mail, and sign out</td>
</tr>
<tr>
<td>Google</td>
<td>Type in a search query and also look for images and news</td>
</tr>
<tr>
<td>GoogleMap</td>
<td>Search for a direction from one city to another</td>
</tr>
<tr>
<td>Hotmail</td>
<td>Sign in, check inbox, delete a mail, and sign out</td>
</tr>
</tbody>
</table>
Understanding JavaScript Behavior

- Code
- JavaScript
- Events
- Objects
Code Behavior

- Function size
- Instructions/call
- Code locality
- Instruction mix
Static Unique Functions Executed

<table>
<thead>
<tr>
<th>Code</th>
<th>Objects</th>
<th>Events</th>
</tr>
</thead>
<tbody>
<tr>
<td>Real Sites</td>
<td>V8</td>
<td>SunSpider</td>
</tr>
<tr>
<td>amazon</td>
<td>bing</td>
<td>bingmap</td>
</tr>
<tr>
<td>cnn</td>
<td>ebay</td>
<td>economist</td>
</tr>
<tr>
<td>facebook</td>
<td>gmail</td>
<td>google</td>
</tr>
<tr>
<td>googlemap</td>
<td>hotmail</td>
<td>Richards</td>
</tr>
<tr>
<td>crypto</td>
<td>raytrace</td>
<td>earley</td>
</tr>
<tr>
<td>regexp</td>
<td>splay</td>
<td>3d-raytrace</td>
</tr>
<tr>
<td>access-nbody</td>
<td>bitops-nsieve</td>
<td>controlflow</td>
</tr>
<tr>
<td>crypto-aes</td>
<td>date-xparb</td>
<td>math-cordic</td>
</tr>
<tr>
<td>regexp-dna</td>
<td>string-tagcloud</td>
<td>code</td>
</tr>
</tbody>
</table>
Bytecodes / Call

function(a, b) {
    var i=0, elem, pos=a.length;
    if(D.browser.msie) {
        while(elem=b[i++])
            if(elem.nodeType!=8)
                a[pos++]=elem;
    } else
        while(elem=b[i++])
            a[pos++]=elem;
    return a
}
Fraction of Code Executed

Most code not executed
Object Allocation Behavior

- Allocation by types
- Live heap composition
- Lifetime distribution
Total Bytes Allocated

- Real Sites
- V8
- SunSpider

Total heap data (kilobytes)

- Code
- Objects
- Events
Heap Data by Type

Many functions
Rest are strings

Real Sites

V8

SunSpider

Few benchmarks allocate much data

Many functions
Rest are strings
Live Heap Over Time (gmail)

- Functions grow steadily
- GC reduces size of heap
- Objects grow steadily too
Live Heap over Time (ebay)

Heap contains mostly functions

Heaps repeatedly created, discarded

Heap drops to 0 on page load
2 Search Websites, 2 Architectures

You stay on the same page during your entire visit
Code loaded once
Heap is bigger

Every transition loads a new page
Code loaded repeatedly
Heap is smaller

Bing

Google
Event Handlers in JavaScript

- Number of events
- Sizes of handlers
Event-driven Programming Model

- Single-threaded, non-preemptive event handlers
- Example handlers: onabort, onclick, etc.
- Very different from batch processing of benchmarks
- Handler responsiveness critical to user experience
Total Events Handled

Real Sites V8

Almost no events
Median Bytecodes / Event Handled

<table>
<thead>
<tr>
<th>Code</th>
<th>Objects</th>
<th>Events</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>506</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>2137</td>
</tr>
</tbody>
</table>

- Median Bytecodes / Event Handled
- Code: [Code], [Objects], [Events]
Sure, this is all good, but...

- Everyone knows benchmarks are unrepresentative
- How much difference does it make, anyway?
- Wouldn’t any benchmarks have similar issues?
Cold-code Experiment

• Observation
  – Real web apps have lots of code (much of it cold)
  – Benchmarks do not

• Question: What happens if the benchmarks have more code?
  – We added extra, unused to code to 7 SunSpider benchmarks
  – We measured the impact on the benchmark performance
Cold code has non-uniform impact on execution time. Cold code makes SunSpider on Chrome up to 4.5x slower.
Impact of Benchmarks

• What gets emphasis
  – Making tight loops fast
  – Optimizing small amounts of code

• Important issues ignored
  – Garbage collection (especially of strings)
  – Managing large amounts of code
  – Optimizing event handling
  – Considering JavaScript context between page loads
Conclusions

• JSMeter is an instrumentation framework
  – Used to measure and compare JavaScript applications
  – High-level views of behavior promote understanding

• Benchmarks differ significantly from real sites
  – Misleads designers, skews implementations

• Next steps
  – Develop and promote better benchmarks
  – Design and evaluate better JavaScript runtimes
  – Promote better performance tools for JavaScript developers
Additional Resources


• **Video:** [Project JSMeter: JavaScript Performance Analysis in the Real World](Project JSMeter: JavaScript Performance Analysis in the Real World) - MSDN Channel 9 interview with Erik Meier, Ben Livshits, and Ben Zorn

• **Paper:**

• **Additional measurements:**
Additional Slides
Hot Function Distribution

80% of time in 100+ functions

80% of time in < 10 functions

Real Sites

V8 Benchmarks
Opcode Distribution

Green = SunSpider | Blue = Real Web Apps | Red = V8
Object Type Distribution

Green = SunSpider | Blue= Real Web Apps | Red = V8

economist is an outlier (arrays)
Distribution of Time in Handlers

Size of handlers (# of executed instructions)

Number of events (normalized)

- amazon
- bing
- bingmap
- cnn
- ebay
- economist
- facebook
- google
- googlemap
- gmail
- hotmail
Related Work

- **JavaScript**
  - Richards, Lebresne, Burg, and Vitek (PLDI’10)
  - Draw similar conclusions

- **Java**
  - Doufour et al. (OOPSLA’03), Dieckmann and U. Hölzle (ECOOP’99)

- **Other languages**
  - C++: Calder et al. (JPL’95)
  - Interpreted languages: Romer et al. (ASPLOS’96)