Run-time characteristics of JavaScript
Part I – Introduction
An analysis of the Dynamic Behavior of the JavaScript Programs

Published at: PLDI 2010

Gregor Richards    Brian Burg

Sylvain Lebresne    Jan Vitek
Why it is interesting

- Proposal of the GAČR and GAUK
  - “Web security” project
- Teaching “internet” lectures
  - bad practices
- Creating web pages
  - optimizations
- Quality of the JavaScript benchmark
How to get data?

- **Framework**
  - instrumented WebToolkit engine (Safari browser)
    - records executed traces (read, writes, calls, deletes, ...)
    - no visible performance impact
  - off-line trace analyzer
    - replays trace
    - gather code characteristics
    - DB used to store result
- Volunteers (colleagues) visit the pages
Selected sites

- Approx. 100 most popular sites (according www.alexa.com)
  - Google, Gmail, YouTube, Google Maps, Facebook, Twitter, eBay, Bing, Wikipedia

- Benchmark suites
  - SunSpider, Dromaeo, V8 Benchmark suite

- Selected by authors
  - Lively Kernel, 280 slides
## Selected sites

<table>
<thead>
<tr>
<th>Sites</th>
<th>Source size</th>
<th>Uniques source size</th>
<th>Trace size</th>
<th>Function count</th>
<th>Hot</th>
<th>Live</th>
</tr>
</thead>
<tbody>
<tr>
<td>280S</td>
<td>116KB</td>
<td>81KB</td>
<td>11,931K</td>
<td>4,293</td>
<td>6.8%</td>
<td>44%</td>
</tr>
<tr>
<td>BING</td>
<td>815KB</td>
<td>186KB</td>
<td>1,199K</td>
<td>2,457</td>
<td>6.4%</td>
<td>46%</td>
</tr>
<tr>
<td>BLOG</td>
<td>1,347KB</td>
<td>775KB</td>
<td>91K</td>
<td>5,087</td>
<td>11.5%</td>
<td>16%</td>
</tr>
<tr>
<td>DIGG</td>
<td>1,106KB</td>
<td>759KB</td>
<td>1,734K</td>
<td>2,957</td>
<td>8.7%</td>
<td>39%</td>
</tr>
<tr>
<td>EBAY</td>
<td>3,156KB</td>
<td>1,034KB</td>
<td>2,239K</td>
<td>10,791</td>
<td>11.7%</td>
<td>31%</td>
</tr>
<tr>
<td>FBOK</td>
<td>14,904KB</td>
<td>1,604KB</td>
<td>5,309K</td>
<td>43,469</td>
<td>5.8%</td>
<td>19%</td>
</tr>
<tr>
<td>FLKR</td>
<td>8,862KB</td>
<td>246KB</td>
<td>490K</td>
<td>19,149</td>
<td>14.0%</td>
<td>13%</td>
</tr>
<tr>
<td>GMAP</td>
<td>1,736KB</td>
<td>833KB</td>
<td>13,125K</td>
<td>5,146</td>
<td>7.8%</td>
<td>61%</td>
</tr>
<tr>
<td>GMIL</td>
<td>2,084KB</td>
<td>1,719KB</td>
<td>6,047K</td>
<td>10,761</td>
<td>7.6%</td>
<td>38%</td>
</tr>
<tr>
<td>GOGL</td>
<td>2,376KB</td>
<td>839KB</td>
<td>1,815K</td>
<td>10,250</td>
<td>15.0%</td>
<td>28%</td>
</tr>
<tr>
<td>ISHK</td>
<td>915KB</td>
<td>420KB</td>
<td>5,376K</td>
<td>2,862</td>
<td>0.6%</td>
<td>35%</td>
</tr>
<tr>
<td>LIVE</td>
<td>1,081KB</td>
<td>938KB</td>
<td>48,324K</td>
<td>2,936</td>
<td>7.4%</td>
<td>49%</td>
</tr>
<tr>
<td>MECM</td>
<td>4,615KB</td>
<td>646KB</td>
<td>14,084K</td>
<td>14,401</td>
<td>6.6%</td>
<td>24%</td>
</tr>
<tr>
<td>TWIT</td>
<td>837KB</td>
<td>160KB</td>
<td>2,252K</td>
<td>2,967</td>
<td>9.2%</td>
<td>45%</td>
</tr>
<tr>
<td>WIKI</td>
<td>1,009KB</td>
<td>115KB</td>
<td>53K</td>
<td>1,226</td>
<td>14.6%</td>
<td>24%</td>
</tr>
<tr>
<td>WORD</td>
<td>1,386KB</td>
<td>235KB</td>
<td>6,403K</td>
<td>3,118</td>
<td>1.0%</td>
<td>42%</td>
</tr>
<tr>
<td>YTUB</td>
<td>2,897KB</td>
<td>562KB</td>
<td>541K</td>
<td>11,321</td>
<td>13.0%</td>
<td>22%</td>
</tr>
<tr>
<td>ALL</td>
<td>2,544KB</td>
<td>790KB</td>
<td>4,151K</td>
<td>10,625</td>
<td>2.2%</td>
<td>26%</td>
</tr>
</tbody>
</table>
Reminder – JavaScript

- Leading “web client-side scripting language”
- “ECMAScript” standard
- Characteristics
  - imperative
  - java-like syntax
  - object oriented
  - prototype based object system
  - dynamic, weak type system
JavaScript – Know your enemy

- Flexible type system
  - “everything can change”

- Object
  - mutable set of properties and methods
  - added/removed at runtime
  - per instance

- Type hierarchy
  - `prototype` property → “predecessor”
  - assignment change hierarchy of predecessors
JavaScript – Know your enemy

- Eval
  - “ordinary” function
  - no limits on state modifications
  - breaks static analysis

- Function arity
  - not considered
    - undefined/arguments array

- Caller/arguments properties
  - modify parameters of function caller
  - transitively
JavaScript Engines

- Mozilla foundation
  - Rhiho (Java)
  - SpiderMonkey (C)
    - Interpreter, High level bytecode
  - TraceMonkey
    - JIT, trace trees

- Google
  - V8 (C++)
    - JIT, hidden classes, generational GC
Part II – Evaluation
1. Prototype hierarchy is invariant

- Prototype chain modifications
  - change of the prototype property
    - add/remove methods

- Uncommon for user-created types
  - modularity

- Used in libraries
  - Prototype library
    - Array – collection like methods
2. Adding properties

Property are added at object initialization.

- most modifications at “initialization phase”

Initialization phase – constructor?

constructor ends, normalized
2. Adding properties

“Hump” of field added after constructor

- factory methods, inheritance emulation, ...
3. Add/delete properties

- Adding properties after initialization
  - rare for the most of sites
  - poorly behaved sites exists
    - up 10% of accesses changes the protocol

- Property removal
  - non-monotonic evolution (breaks subtyping guaranties)

most sites $\rightarrow$ not used
rest $\rightarrow$ quite common
3. Add/delete properties

Average ratio of field add/delete operations to total activity.
4. Eval

- Use of `eval` is infrequent and does not affect semantics

- Recognized types of usage
  - JSON – (de)serialization
  - trivial – calculator, ...
  - arbitrary code

```javascript
if (eval("flash" + i + "Installed")===true) {
  ...
```
4. Eval
4. Eval

- Randomly selected `eval` strings

```javascript
window.dc_AdLinkBold = false

playlist[204]= new function() {
  this.album_id = 204;
  this.album_name = "[elided]";
  this.album_rating_avg = 3.9;
  this.OA = 1255051920;
  this.album_rating_user = -0.1;
};

this.load = function() {
  var a = arguments, len = a.length, s = "";
  for (var i=0 ; i<len ; i++) s += ",a[" + i + "]";
  return eval('this._processEvent("load" + s +')');
}

typeof(l[i].parent.onAfterLoad) == 'function'

objRef.onHandleInteraction = new Function()
```
4. Eval

- Most sites use `eval`

- Real usages
  - not only deserialization
  - arbitrary unpredictable behavior
  - significant for logic of JS

- Hidden evals
  - `document.write`
  - `document.createElement("script")`
5. Function arities

- Declared function signatures are indicative of types
  - assumption of type systems for JS

- Variadic functions rare in other languages
## 5. Function arities

<table>
<thead>
<tr>
<th>Site</th>
<th>Functions with N distinct arities</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
</tr>
<tr>
<td>280S</td>
<td>99.3%</td>
</tr>
<tr>
<td>BING</td>
<td>94.2%</td>
</tr>
<tr>
<td>BLOG</td>
<td>97.1%</td>
</tr>
<tr>
<td>DIGG</td>
<td>92.5%</td>
</tr>
<tr>
<td>EBAY</td>
<td>95.9%</td>
</tr>
<tr>
<td>FBOK</td>
<td>93.9%</td>
</tr>
<tr>
<td>FLKR</td>
<td>94.2%</td>
</tr>
<tr>
<td>GMAP</td>
<td>93.4%</td>
</tr>
<tr>
<td>GMIL</td>
<td>95.3%</td>
</tr>
<tr>
<td>GOGL</td>
<td>94.6%</td>
</tr>
<tr>
<td>ISHK</td>
<td>97.6%</td>
</tr>
<tr>
<td>LIVE</td>
<td>92.7%</td>
</tr>
<tr>
<td>MECM</td>
<td>91.9%</td>
</tr>
<tr>
<td>TWIT</td>
<td>90.9%</td>
</tr>
<tr>
<td>WIKI</td>
<td>96.7%</td>
</tr>
<tr>
<td>WORD</td>
<td>92.6%</td>
</tr>
<tr>
<td>YTUB</td>
<td>98.5%</td>
</tr>
<tr>
<td>All</td>
<td>93.5%</td>
</tr>
</tbody>
</table>
5. Function arities

• Approx. 8% function variadic
  - cannot be ignored
• Some functions highly variadic
  - up to 30 distinct arities
• Few sites <5% variadic functions

• Both coding styles used
  - “C++ default parameters”
  - Checks for additional parameters
    - in argument array
6. Modest program size

- Assumption
  - hand written program small → analyses need not scale

- Full program analysis → infeasible

<table>
<thead>
<tr>
<th>Sites</th>
<th>Source size</th>
<th>Uniques source size</th>
<th>Trace size</th>
<th>Function count</th>
<th>Hot</th>
<th>Live</th>
</tr>
</thead>
<tbody>
<tr>
<td>WORD</td>
<td>1,386KB</td>
<td>235KB</td>
<td>6,403K</td>
<td>3,118</td>
<td>1.00%</td>
<td>42%</td>
</tr>
<tr>
<td>YTUB</td>
<td>2,897KB</td>
<td>562KB</td>
<td>541K</td>
<td>11,321</td>
<td>13.00%</td>
<td>22%</td>
</tr>
<tr>
<td>ALL</td>
<td>2,544KB</td>
<td>790KB</td>
<td>4,151K</td>
<td>10,625</td>
<td>2.20%</td>
<td>26%</td>
</tr>
</tbody>
</table>

averages
7. Low call site dynamism

- Monomorphic call site
  - always calls same method
  - candidates for inlining
  - Java – 90% call sites

- V8 engine – hidden classes + inline caches
  - works well if behavior is similar to OO languages (Java, C++)
7. Low call site dynamism

<table>
<thead>
<tr>
<th>Site</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>&gt;5</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>280S</td>
<td>99.9%</td>
<td>0.0%</td>
<td>0.0%</td>
<td>0.0%</td>
<td>0.0%</td>
<td>1437</td>
</tr>
<tr>
<td>BING</td>
<td>93.6%</td>
<td>4.8%</td>
<td>1.0%</td>
<td>0.3%</td>
<td>0.3%</td>
<td>274</td>
</tr>
<tr>
<td>BLOG</td>
<td>95.4%</td>
<td>3.4%</td>
<td>0.5%</td>
<td>0.2%</td>
<td>0.5%</td>
<td>95</td>
</tr>
<tr>
<td>DIGG</td>
<td>95.4%</td>
<td>3.2%</td>
<td>0.4%</td>
<td>0.3%</td>
<td>0.7%</td>
<td>44</td>
</tr>
<tr>
<td>EBAY</td>
<td>91.5%</td>
<td>7.1%</td>
<td>0.5%</td>
<td>0.5%</td>
<td>0.5%</td>
<td>143</td>
</tr>
<tr>
<td>FBOK</td>
<td>76.3%</td>
<td>14.8%</td>
<td>3.7%</td>
<td>1.7%</td>
<td>3.5%</td>
<td>982</td>
</tr>
<tr>
<td>FLKR</td>
<td>81.9%</td>
<td>13.2%</td>
<td>3.6%</td>
<td>0.5%</td>
<td>0.8%</td>
<td>244</td>
</tr>
<tr>
<td>GMAP</td>
<td>98.2%</td>
<td>0.8%</td>
<td>0.4%</td>
<td>0.2%</td>
<td>0.4%</td>
<td>345</td>
</tr>
<tr>
<td>GMIL</td>
<td>98.4%</td>
<td>1.2%</td>
<td>0.2%</td>
<td>0.1%</td>
<td>0.2%</td>
<td>800</td>
</tr>
<tr>
<td>GOGL</td>
<td>93.1%</td>
<td>5.5%</td>
<td>0.6%</td>
<td>0.3%</td>
<td>0.6%</td>
<td>1042</td>
</tr>
<tr>
<td>ISHK</td>
<td>90.2%</td>
<td>8.1%</td>
<td>1.0%</td>
<td>0.0%</td>
<td>0.8%</td>
<td>42</td>
</tr>
<tr>
<td>LIVE</td>
<td>97.0%</td>
<td>1.7%</td>
<td>0.5%</td>
<td>0.3%</td>
<td>0.5%</td>
<td>115</td>
</tr>
<tr>
<td>MECM</td>
<td>94.2%</td>
<td>4.1%</td>
<td>1.2%</td>
<td>0.2%</td>
<td>0.4%</td>
<td>106</td>
</tr>
<tr>
<td>TWIT</td>
<td>89.5%</td>
<td>7.2%</td>
<td>1.7%</td>
<td>0.3%</td>
<td>1.3%</td>
<td>60</td>
</tr>
<tr>
<td>WIKI</td>
<td>87.9%</td>
<td>6.7%</td>
<td>1.9%</td>
<td>0.2%</td>
<td>3.2%</td>
<td>32</td>
</tr>
<tr>
<td>WORD</td>
<td>86.8%</td>
<td>7.9%</td>
<td>2.7%</td>
<td>1.9%</td>
<td>0.6%</td>
<td>106</td>
</tr>
<tr>
<td>YTUB</td>
<td>83.6%</td>
<td>10.6%</td>
<td>5.4%</td>
<td>0.1%</td>
<td>0.4%</td>
<td>183</td>
</tr>
<tr>
<td>All</td>
<td>81.2%</td>
<td>12.1%</td>
<td>3.0%</td>
<td>1.2%</td>
<td>2.5%</td>
<td>1437</td>
</tr>
</tbody>
</table>
7. Low call site dynamism

- 81% sites monomorphic
  - chance to inline calls

- “megamorphic” call sites (>200 targets)
  - quite often – Bing, Facebook, Flicker, Gmail, ..

- Static analysis
  - high enough to reduce static analysis precision
8. Execution time / hot loops

- Execution time is dominated by hot loops
  - Important for trace based engines (TraceMonkey)

- 90/10 rule (90% of time spent in 10% of code)
  - 6 - 15% hot (8% median)

- 50% function is never called
  - due to libraries
Summary

• Assumptions are (often) violated
  ▪ no site violates all assumptions
    → optimizations can work in many cases

• Evals are common

• Whole-program analysis – infeasible

• Rigidly type systems – not usable
9. Benchmarks

- Object kinds
  - a few type heavily stressed (#instances, #arrays)

- Evals
  - SunSpider fits

- Function variadicity

- Call site dynamism

<table>
<thead>
<tr>
<th></th>
<th>SunSpider</th>
<th>V8 BS</th>
<th>Dromaeo</th>
<th>RL -ALL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eval</td>
<td>2785</td>
<td>0</td>
<td>32</td>
<td>-</td>
</tr>
<tr>
<td>Eval – JSON</td>
<td>33</td>
<td>0</td>
<td>1</td>
<td>-</td>
</tr>
<tr>
<td>Variadicity</td>
<td>5%</td>
<td>2%</td>
<td>5%</td>
<td>7.5%</td>
</tr>
<tr>
<td>Monomorphic call sites</td>
<td>98%</td>
<td>99%</td>
<td>97%</td>
<td>81.2%</td>
</tr>
</tbody>
</table>
9. Benchmarks – Add/delete properties

SunSpider

V8 Benchmark suite
Thank you