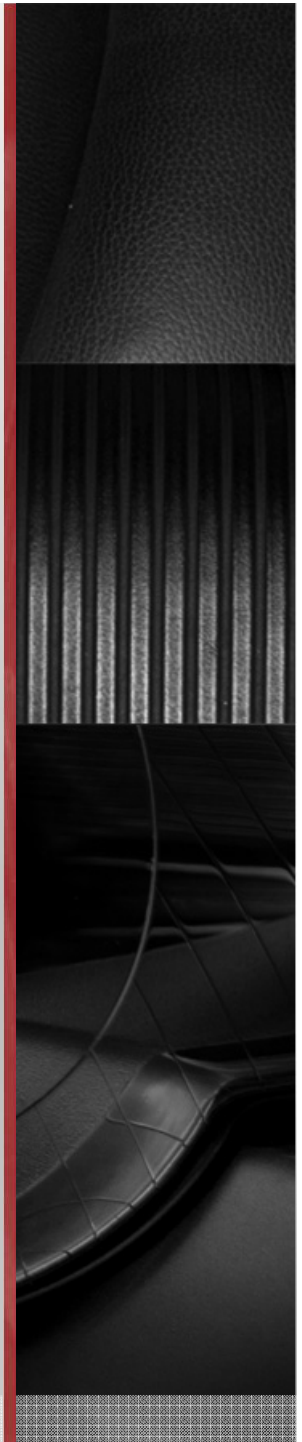


Automated construction of JavaScript benchmarks

*Gregor Richards, Andreas Gal,
Brendan Eich, Jan Vitek*

OOPSLA '11

Kamil Hajduczenia





Presentation plan

1. Introduction
2. JavaScript in modern web applications
3. Issues related to JavaScript testing
4. JavaScript benchmark tools
5. Requirements
6. JSBench model
7. Testing mechanisms in JSBench
8. Formalisms
9. Results
10. Summary
11. Discussion



JavaScript language in modern web applications

1. JavaScript is a leading client-side scripting programming language used in modern web applications
2. Ask Google*:
 1. Java job offers: 16,000,000 results
 2. Python job offers: 23,600,000 results
 3. PHP job offers: 42,600,000 results
 4. JavaScript job offers: **101,000,000** results
3. JavaScript framework at Google: 12,200,000 results

* Google queries' results on 20th May 2012, site: www.google.pl



Issues related to JavaScript testing

- Execution nondeterminism
 - Presentation - DOM objects and Listener functions
 - Network - XMLHttpRequest objects (send), Listener functions (receive)
 - File System - DOM objects (cookies)
 - Time - Date, setTimeout
 - User input - DOM objects and Listener objects
 - Nondeterministic functions - Math.random
 - Environment queries - document.location, navigator
- Dynamic scripts loading
- Locations of JavaScript code – script tags, inline code, eval(), ajax-loaded code
- Experience with Java benchmarks: SPECjvm98 vs. DaCapo



JavaScript benchmark tools

- SunSpider
- Google V8 Benchmark Suite
- Dromaeo
- Benchmark.js
- Kraken

JavaScript execution comparison

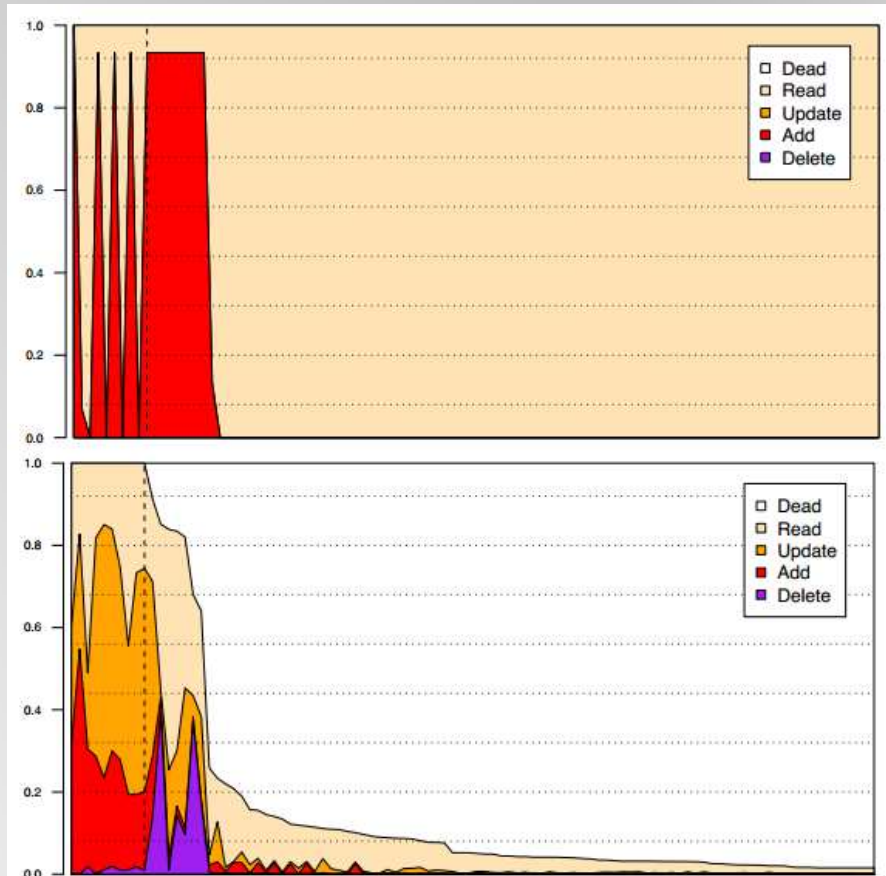


Figure 1. Object timelines. Comparing the operations performed on objects in industry standard benchmarks and web applications. Above, SunSpider. Below, Google.



Requirements

- **Deterministic replay:** multiple runs of a benchmark should display the same behavior.
- **Browser-independence:** a benchmark's behavior should not be affected by browser-specific features and should execute on all browsers.
- **Fidelity:** benchmarks should correctly mimic the behavior of live interactions with a web application. As there is no result in a web page, we focus on the execution of events and changes to the web page
- **Accuracy:** benchmarks should be representative of the performance and non-functional characteristics of the original web applications.



JSBench

- JavaScript benchmark tool
- Available at: http://hg.mozilla.org/users/gkrichar_purdue.edu/jsbench/
- Created in cooperation by Purdue University and Mozilla Foundation

In Dionne's taxonomy, JSBENCH is a data-based automatic replay system as it records data exchanged between the program and its environment and requires no human-written changes to the source of the monitored program.

JSBench model

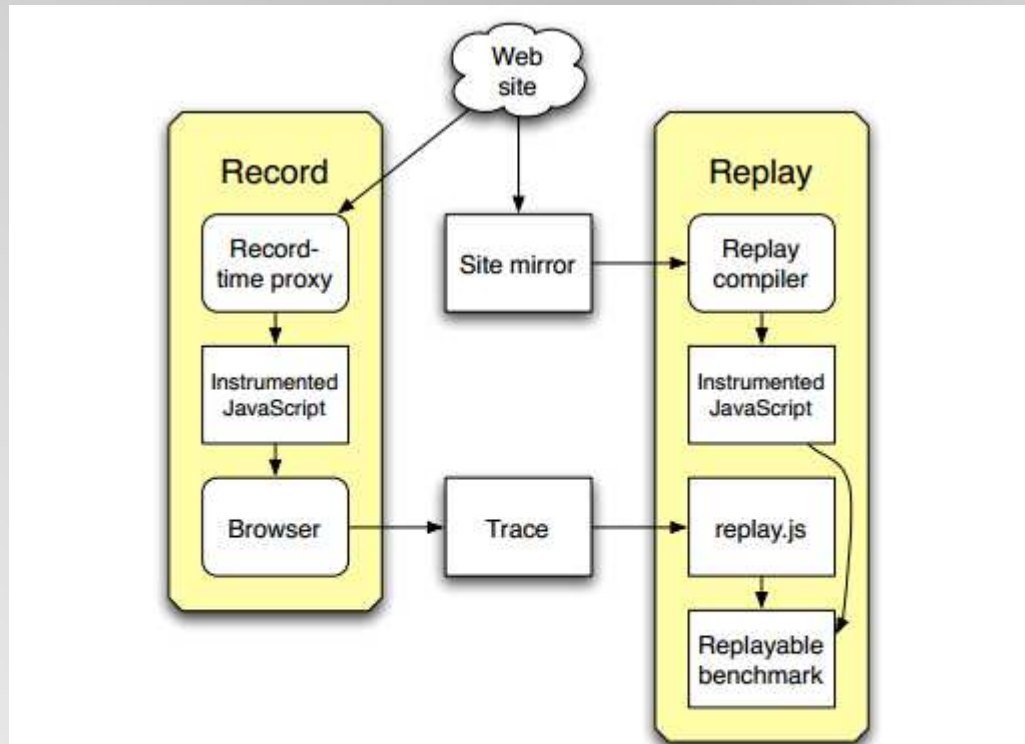


Figure 4. System architecture. JSBENCH acts as a proxy between the browser and the server, rewriting JavaScript code on the fly to add instrumentation and finally creating an executable stand-alone application.



Testing mechanisms in JSBench

- Record/replay testing approach:
 - A client-side proxy instruments the original web site's code to emit a trace of JavaScript operations performed by the program
 - The trace is filtered to get rid of unnecessary information
 - A replayable JavaScript program, with all non-determinism replaced, that will reproduce the behavior of the trace is generated from the trace
 - The program is recombined with HTML from the original web site.
- Recording traces by rewriting the execution model – filtering non-deterministic elements, memoizing function calls, remembering attributes access

Record/replay example

```
1 function onloadHandler() {  
2   myvalue = document  
3     .getElementById("input")  
4     .value;  
5 }
```

(a) Source code

```
1 function replay() {  
2   var o1 = {};  
3   window.document = o1;  
4   var f1 =  
5     function() { return lookupCase(f1, this, arguments); }  
6   o1.getElementById = f1;  
7   var o2 = {};  
8   f1.cases[2][o1]["input"] = o2;  
9   o2.value = "Hello!";  
10  onloadHandler.call(window);  
11 }
```

Figure 7. Example replay function.

- Example shows the general mechanism of replaying the event handler from JavaScript source code, recorded by JSBench and executed in browser-independent model.

Record/replay example

```
1 function onloadHandler() {  
2   myvalue = document  
3     .getElementById("input")  
4     .value;  
5 }
```

(a) Source code

```
1 function replay() {  
2   var o1 = {};  
3   window.document = o1;  
4   var f1 =  
5     function() { return lookupCase(f1, this, arguments); }  
6   o1.getElementById = f1;  
7   var o2 = {};  
8   f1.cases[2][o1]["input"] = o2;  
9   o2.value = "Hello!";  
10  onloadHandler.call(window);  
11 }
```

Figure 7. Example replay function.

- Example shows the general mechanism of replaying the event handler from JavaScript source code, recorded by JSBench and executed in browser-independent model.

Definitions

Definition 1. *The execution state of a web application consists of the state of a JavaScript engine P and an environment E . A step of execution is captured by a transition relation $P|E \xrightarrow{\alpha}_t P'|E'$ where α is a label and t is a time stamp.*

Definition 2. *A trace T is a sequence $(\alpha_1, t_1), \dots, (\alpha_n, t_n)$ corresponding to an execution $P|E \xrightarrow{\alpha_1}_{t_1} \dots \xrightarrow{\alpha_n}_{t_n} P'|E'$. We write $P|E \vdash T$ when execution of a configuration $P|E$ yields trace T .*

Definition 3. *Two traces are DOM-equivalent, $T \cong_D T'$, if*

$$(\text{SET}_{\rightarrow}, p_i, v_i, t_i) \in T|_{\text{SET/DOM}} \wedge (\text{SET}_{\rightarrow}, p'_i, v'_i, t'_i) \in T'|_{\text{SET/DOM}}$$

and

$$\forall i : p_i = p'_i \wedge v_i = v'_i$$

Requirements – formally defined

Property 1. [Determinism] $P_R|E$ always yields the same trace \bar{T} and for any environment E' , $P_R|E'$ yields a trace \bar{T}' that is DOM-equivalent to \bar{T} , such that $\bar{T} \cong_D \bar{T}'$.

Property 2. [Fidelity] If $R(P)|E \vdash \bar{T}^R$, and $P_R|E \vdash \bar{T}$ then $\bar{T}^R \cong_D \bar{T}$.

Property 3. [Accuracy] If $P|E \vdash T$ and $P_R|E \vdash \bar{T}$, there is some environment E' such that $P|E' \vdash T'$ and the distance between the traces $\delta(T, \bar{T}) < \delta(T, T')$.

Results

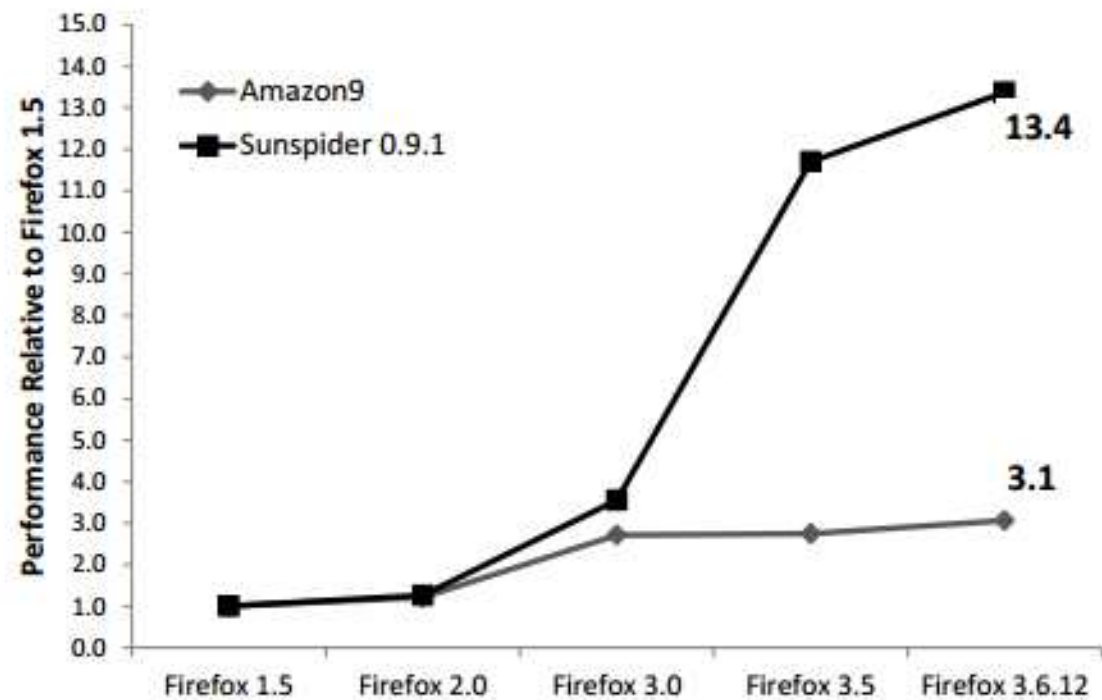


Figure 18. Speed ups. Throughput improvements of different versions of Firefox. (sunspider, amazon; FF1.5 - FF 3.6.12) Measurements on an HPZ800, dual Xeon E5630 2.53Ghz, 24GB memory, Windows 7 64-bit Enterprise. Numbers normalized to FF 1.5.

Results

Benchmark	IE8	IE9	FF3	FF4	Chrome5	Chrome6	Opera10	Safari5
Sibelius	684.4	117.8	315.6	285.6	82.4	82	89.4	74.2
Amazon	342.6	63.4	104	85.8	110.4	111.4	147.2	62.8
Microsoft	42.8	4	14	11.8	12.4	10.4	58.6	5
Bing	87	11.4	44.2	51.8	30	27.6	9.8	9
Economist		81.6	103	124.4	48	39.4	40.6	57.6
MSNBC		32.2	172.2	85.2	32.6	31.8	43.6	48.4
JSMIPS		7,773.4			4,935.4	3,480.6		12,596.2

Figure 22. Cross-browser running time comparison. Times are in milliseconds. An empty cell indicates that the benchmark could not produce results on the JavaScript engine we used for measurements, due either to insufficient feature support or taking too long to execute.




Summary

- JavaScript is commonly used programming language in web applications
- Optimizing JavaScript engines in web browsers is a necessity
- Profiling optimizations according to market needs leads to users' satisfaction
- Using static JavaScript test suites unrelated to real-world websites' scripts usage might produce optimizations that are off the mark
- Building benchmark tools' scripts based on market JavaScript executions is a way to improve engines' optimizations
- JSBench is a tool that shows basic mechanisms for recognizing bottlenecks of JavaScript engines in market-environment



Discussion



Thank you for your attention!

