node JS® Interactive

EUROPE

Amsterdam, Netherlands

September 15–18, 2016

Node.js Interactive
EUROPE
Amsterdam, Netherlands
September 15–18, 2016

Node Community Benchmarking Efforts

Gareth Ellis, IBM

- Runtime Performance Analyst @ IBM
- Looking at Performance since 2012
- Originally solely Java
- Started on Node 2015
- Member of benchmarking workgroup

- Contact me:

  gareth.ellis@uk.ibm.com

  github.com/gareth-ellis

Agenda

- Introduction to benchmarking
- Benchmarking Node.js
- Node benchmarking workgroup

Introduction

- Change one thing and one thing only between runs
    - Application code / benchmark
    - Runtime
    - Machine
    - NPM module(s)
- Performance testing is quite different to functional testing

Key challenges

- Fundamental run-to-run variance
  - False positives
  - Collecting enough samples to be sure of the result
  - Documenting expected variance
- Consistent environment
  - Known starting machine state
  - Machine isolation
  - Interleave comparison runs
- Jumping to conclusions

Approaches

- Micro-Benchmarks
  - Measure a specific function/API
  - Ex: Buffer.new()
  - Compare key characteristics
  - Micro-benchmark improvements may not mean real world improvements
  - Risk of not measuring exactly what you expect – especially where a JIT is involved
- Whole System
  - Benchmark expected customer use case
  - Ex: AcmeAir - http://github.com/acmeair/acmeair-nodejs
  - The more you test, the more chance for variance

I’ve found a regression ….now what?

- Are you sure ?
  - Revalidate environment,
  - Expected variance
- If so, what changed ?
  - Your application
  - Node.js
  - Your environment
- Compare between good/bad cases
  - Tools
  - Binary search

Benchmarking Node.js

- Sources of regressions
  - Node.js
    - lib/*.js – buffer, cluster, etc
  - V8
  - OpenSSL
  - libuv
  - NPM Module
- Tools
  - Javascript profiler
    - V8 profiler
    - Appmetrics
  - Native profiler (ex perf, tprof, oprofile)

V8 Profiler

- Part of Node.js binary
- Turn on with
    - --prof
- Test-tick-process to post-process
    - ./node --prof-process isolate-0x2818130-v8.log
- Other helper modules like
    - https://www.npmjs.com/package/v8-profiler

Appmetrics

- npm install appmetrics
- Can provide cpu, gc, memory, profiling + lots more
- Connect into IBM healthcenter for remote monitoring
- https://www.npmjs.com/package/appmetrics

Binary chop

- Compare changes between good & bad
- git rev-list ^good bad^
- Bit more time consuming for where there’s a lot of change sets
- git bisect is quite useful though

Example - Microbench

```
var harness = require('../../common/harness.js');
var ARRAY = [1, 2, 23829, 4, 5, 7, 12312321, 2131, 434832, 43792, 23421, 65345, 132210,
77777, 322131, 1, 2, 23829, 4, 5, 7, 12312321, 2131, 434832, 43792, 23421, 65345,
132210, 77777, 322131, 1, 2, 23829, 4, 5, 7, 12312321, 2131, 434832, 43792, 23421,
65345, 132210, 77777, 322131, 1, 2, 23829, 4, 5, 7, 12312321, 2131, 434832, 43792,
23421, 65345, 132210, 77777, 322131];
var ITERATIONS = 300000;
var result;


function test() {
        for(var i=0;i<ITERATIONS;i++) {
                result = new Buffer(ARRAY);
        }
}
harness.run_test(test);
```

Example - Microbench

Node 4.3.2:

```
./node benchmark.js
total time:5.079s / iterations:54 / ops/sec:10.63 / average time:0.09s / variance:0.89%
total time:5.076s / iterations:54 / ops/sec:10.64 / average time:0.09s / variance:0.75%
```

Node 4.4.0:

```
./node benchmark.js
total time:5.131s / iterations:31 / ops/sec:6.04 / average time:0.17s / variance: 2.32%
total time:5.106s / iterations:31 / ops/sec:6.07 / average time:0.16s / variance: 0.28%
```

= ~ 40% regression

V8 profiler (--prof)

```
<    5585   23.7%   23.9%  LazyCompile: *fromObject buffer.js:121:20
<    1308    5.6%    5.6%  LazyCompile: *subarray native typedarray.js:165:28
<    1263    5.4%    5.4%  LazyCompile: *Uint8ArrayConstructByArrayBuffer native
typedarray.js:35:42
<     964    4.1%    4.1%  Builtin: JSConstructStubGeneric
<     854    3.6%    3.7%  Stub: InstanceofStub
<     677    2.9%    2.9%  LazyCompile: *test benchmark.js:7:14
<     669    2.8%    2.9%  LazyCompile: *Uint8Array native typedarray.js:122:31
---
>   15227   47.1%   47.3%  LazyCompile: *fromObject buffer.js:121:20
>    1240    3.8%    3.9%  LazyCompile: *subarray native typedarray.js:165:28
>    1166    3.6%    3.6%  LazyCompile: *Uint8ArrayConstructByArrayBuffer native
typedarray.js:35:42
>     967    3.0%    3.0%  Builtin: JSConstructStubGeneric
>     802    2.5%    2.5%  Stub: InstanceofStub
>     780    2.4%    2.4%  LazyCompile: *test benchmark.js:7:14
>     654    2.0%    2.0%  LazyCompile: *Uint8Array native typedarray.js:122:31
```

Perf – system based

- perf record -i -g -e cycles:u -- ./node --perf-basic-prof benchmark.js
- perf report

```
diff perf_good.out perf_bad.out

327c302
<     91.52%    23.43%  node     perf-16993.map       [.]
LazyCompile:*fromObject buffer.js:121
---
>     93.25%    46.56%  node     perf-16934.map       [.]
LazyCompile:*fromObject buffer.js:121
331c306
```

The result

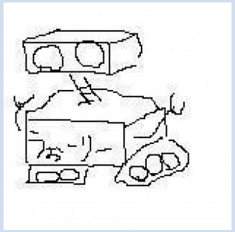
- Issue in v8 optimiser.
- Will be fixed there once TurboFan becomes default
- Until then…
- github.com/nodejs/node/pull/5819

Workgroup

- Mandate to track and evangelize performance gains between node releases
- Key goals
  - Define Use Cases
  - Identify Benchmarks
  - Run/Capture results
- 12 current members
- Meetings every month or so

http://github.com/nodejs/benchmarking
http://benchmarking.nodejs.org

Who's involved?

	Michael Dawson (@mhdawson)		Yosuke Furukawa (@yosuke-furukawa)		Michael Paulson (@michaelbpaulson)
	Trevor Norris (@trevnorris)		Yunong Xiao (@yunong)		Gareth Ellis (@gareth-ellis)
	Yang Lei (@yanglei99)		Mark Leitch (@m-leitch)		Chris Bailey (@seabaylea)
	Ali Sheikh (@ofrobots)		Surya V Duggirala (@suryadu)		David Stewart (@davidcstewart)

Current benchmarks

- Currently Running
  - Startup time
  - Footprint
  - Time to 'require' a module
  - AcmeAir – throughput, response time, footprint measurements
  - Docker

- In progress
  - URL performance
  - Additions from Node.js benchmarks directory

Use cases

Use case	Benchmark with coverage
Service oriented architectures (SOA)	
Microservice-based applications	
Generating/serving dynamic web page content	Acme Air Ops/s Acme Air Latency
Single page applications with bidirectional communication over WebSockets and/or HTTP/2	
Agents and Data Collectors	

Use cases

Use case	Benchmark with coverage	Notes
Small scripts	Require New Ops Require Cached Ops	Needs more attributes tested
startup/stop time	Start + stop Time	

Key Attributes

Key attribute	Benchmark with coverage
Memory footprint at startup	Startup footprint
Memory footprint after load	Acme air runtime footprint
Node.js process cpu usage at idle	
Throughput	Acme Air ops/sec
Install package size	
Size on disk once install	

Key Attributes

Key attribute	Benchmark with coverage
GC CPU usage under load	
GC allocation throughput under load	
GC max pause time under load	

Charts

How to get involved

- http://www.github.com/nodejs/benchmarking
- Take a look at what we're running – and the areas we're looking to get benchmarks to cover
- Something missing?
- Something you don't think is quite right?
- Open an issue!