Instrumenting the real-time web: Node.js, DTrace and the Robinson Projection

Bryan Cantrill
VP, Engineering

bryan@joyent.com
@bcantrill
node.js is a JavaScript-based framework for building event-oriented servers:

```javascript
var http = require('http');

http.createServer(function (req, res) {
    res.writeHead(200, {'Content-Type': 'text/plain'});
    res.end('Hello World
');
}).listen(8124, "127.0.0.1");

console.log('Server running at http://127.0.0.1:8124!');
```
The energy behind Node.js

- node.js is a confluence of three ideas:
  - JavaScript’s rich support for asynchrony (i.e. closures)
  - High-performance JavaScript VMs (e.g. V8)
  - The system abstractions that God intended (i.e. UNIX)
- Because everything is asynchronous, node.js is ideal for delivering scale in the presence of long-latency events
• In August of last year, Joyent hosted “Node Knockout”, a programming competition for the nascent node.js environment

• Weekend-long competition in which teams of one to four endeavored to build something complete with node

• For Joyent, this presented an opportunity to understand and observe the new environment in the wild

• What could we learn about these systems and what could we convey to the contestants in real-time?
A Node Knockout leaderboard?

• Even though the contest was judged, could we provide a real-time leaderboard?

• @ryah’s idea: instrument incoming connections, trace the remote IP address and then geo-locate in real-time

• Would allow a leaderboard to reflect number of unique IPs per contestant -- and where they’re coming from

• Would need instrumentation to be entirely transparent; log analysis and other offline techniques are both suboptimal and overly invasive

• These constraints are a natural fit for DTrace...
DTrace

- Facility for dynamic instrumentation of production systems originally developed circa 2003 for Solaris 10
- Open sourced (along with the rest of Solaris) in 2005; subsequently ported to many other systems
- Support for arbitrary actions, arbitrary predicates, *in situ* data aggregation, statically-defined instrumentation
- Designed for safe, *ad hoc* use in production: concise answers to arbitrary questions
- But how to use DTrace to instrument contestants?
Node + DTrace

- DTrace instruments the system *holistically*, which is to say, from the kernel, which poses a challenge for interpreted environments.
- User-level statically defined tracing (USDT) providers describe semantically relevant points of instrumentation.
- Some interpreted environments e.g., Ruby, Python, PHP) have added USDT providers that instrument the interpreter itself.
- This approach is very fine-grained (e.g., every function call) and doesn’t work in JIT’d environments.
- We decided to take a different tack for Node.
• Given the nature of the paths that we wanted to instrument, we introduced a function into JavaScript that Node can call to get into USDT-instrumented C++

• Introduces disabled probe effect: calling from JavaScript into C++ costs even when probes are not enabled

• Use USDT is-enabled probes to minimize disabled probe effect once in C++

• If (and only if) the probe is enabled, prepare a structure for the kernel that allows for translation into a structure that is familiar to node programmers
• Example one-liners:

```bash
  dtrace -n 'node*:::http-server-request{
    printf("%s of %s from %s\n", args[0]->method,
      args[0]->url, args[1]->remoteAddress)}'

  dtrace -n http-server-request'{@[args[1]->remoteAddress] = count()}'

  dtrace -n gc-start'{self->ts = timestamp}' \n  -n gc-done'/self->ts/{@ = quantize(timestamp - self->ts)}'
```

• A more interesting script:

```c
  http-server-request
  {
    self->ts[args[1]->fd] = timestamp;
  }

  http-server-response
  /self->ts[args[0]->fd]/
  {
    @[zonename] = quantize(timestamp - self->ts[args[0]->fd]);
  }
```
Instrumenting Node Knockout

• With a USDT provider in place for Node, we could instrument contestants in a meaningful way

• But how can contestants be instrumented given that each is executing in their own virtualized environment?
The Joyent cloud uses OS virtualization to achieve high levels of tenancy without sacrificing performance:

- Allows for transparent instrumentation of all virtual OS instances from the global zone via DTrace
• Define connection establishment/teardown to be “ticks”
• Have a daemon instrument all virtual OS instances from each compute node’s global zone, recording remote IP address and collecting ticks in a ring buffer
• Poll the data periodically from a centralized server, pulling together a merged stream of ticks and geo-locating IPs
• Have HTTP clients periodically poll the server, and rendering new connections on a world map
Leaderboard architecture
Leaderboard architecture

- 1,000 tick ring buffer
- 10,000 tick ring buffer
- every 500 ms
- every 100 ms
- every 100 ms
- 700 ms latency

HTTP

LB

leaderd

leaderd

DTrace

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS

Virtual OS
Virtual OS
Virtual OS
Virtual OS
• Necessitated a libdtrace add-on for node for tickerd:
  https://github.com/bcantrill/node-libdtrace

• Used existing node-geoip add-on for leaderd, but ultimately wrote a (much) simpler add-on:
  https://github.com/bcantrill/node-libgeoip

• Used HTTP + Keep-alive for leaderd/tickerd

• Simple architecture; very quick to build: ~400 lines of node for leaderd, ~500 lines of node for tickerd

• Surprisingly, most time-consuming and brittle part was adding git statistics to tickerd!
• How to present the geo-located IP connection information (latitude and longitude) visually?

• When a sphere is projected onto a flat surface, something has to give: distance, shape, size, bearing

• The two projections most commonly used to visualize location are both undesirable...
Equirectangular Projection
Mercator Projection
Robinson Projection FTW!
You’d be forgiven for assuming that the Robinson is actually a projection; quite the contrary:

“I started with a kind of artistic approach. I visualized the best-looking shapes and sizes. I worked with the variables until it got to the point where, if I changed one of them, it didn't get any better. Then I figured out the mathematical formula to produce that effect. Most mapmakers start with the mathematics.”

- Arthur H. Robinson

Not surprisingly, implementing this is a mess...

...and if you get it only slightly wrong, it’s obvious

But Joyent’s @rob_ellis stepped up and pulled it off:

http://github.com/silentrob/Robinson-Projection
Robinson-based Leaderboard!
Experiences

• Leaderboard very quickly got 1,000+ active users

• CPU utilization remained negligible (< 6% of one CPU), but network utilization became “interesting”

• Over the 48 hours of the contest (and for the week afterward), no tickerd failed; leaderd died twice due to memory leaks in Node (since fixed)

• Most significant issue was a per-contestant graph updating in real-time that caused the browser to crash after ~15 minutes (graph was removed Sat. AM)

• Interesting (mesmerizing?) to watch real-time geo-located connection data as contestants’ entries went globally viral
Epitome of a broader shift?

- As the competition unfolded, it became clear that the leaderboard typified the entrants: many were data-intensive real-time systems.

- Also typified many of the early adopters of node.js: many came from environments that had unacceptable outliers when used in data-intensive real-time systems.

- Acronym clearly called for; CRUD, ACID, BASE, CAP: meet DIRT!

- That node.js is such a fit for DIRT highlights that *long latency events* (and *not* CPU time) are the impediment to web-facing real-time systems.
The primacy of latency

- As a reminder, a *real-time system* is one in which the correctness of the system is relative to its timeliness.

- In such a system, *it does not make sense to measure operations per second!*

- The *only* metric that matters is latency.

- This is dangerous to distill to a single number; the *distribution* of latency over time is essential.

- This poses both instrumentation and visualization challenges!
Instrumenting the real-time web

- We’ve taken a swing at this with the new cloud analytics facility in our no.de environment, a public node.js PaaS:

- ...but there’s much more to be done to understand the coming breed of DIRTy applications!
Thank you!

- Node Knockout Leaderboard shout-outs: @rob_ellis, @jahoni, @yoheis and @brianleroux
- Node Knockout guys: @visnup and @gerad
- Node DTrace USDT integration: @ryah and @rmustacc
- no.de cloud analytics: @dapsays, @rmustacc, @rob_ellis and @notmatt