Joyent

DTrace, Node.js, and Flame Graphs

NodeConf 2012
David Pacheco (@dapsays)
Joyent

Node.js for systems programming

Joyent

- At Joyent, we've found Node.js to be a great environment for systems programming
  - Unix abstractions (e.g., streams)
  - Event-oriented architecture limits impact of latency bubbles
- We see Node displacing C for **core infrastructure software**:
  - HTTP, obviously
  - Also: DHCP, DNS, LDAP, SNMP, HTTP proxies, key-value stores, ...
  - The whole SmartDataCenter stack (provisioning, real-time analytics, monitoring, ...)
- Node makes it very fast to do new things with these technologies (e.g., ldapjs), but we need modern tools for performance analysis and postmortem debugging to make these services as fast and reliable as we expect from core infrastructure.

DTrace Review

Joyent

- Don't guess. **Measure everything**.
- Allows performance investigations to be **data-driven**.
- Comprehensive tracing of both kernel and application-level events in real-time.
- Scales arbitrarily with the number of traced events.
  (first class *in situ* data aggregation)
- Suitable for production systems because it's safe, has minimal overhead (usually no disabled probe effect), and can be enabled/disabled dynamically (no application restart required).
- Open-sourced in 2005. Available on illumos (and Solaris-derived systems), BSD, and MacOS (Linux ports in progress).

Two types of performance problems

Joyent

- Case 1: asynchronous (your program is **not** on CPU):
  - Problem is with something else (filesystem, database, remote web service).
  - (May also be an architectural problem with your program.)
- Case 2: synchronous (your program is on CPU too much)
  - Problem is with your program.

Measuring asynchronous operations

Joyent

- Want to see the *latency* of asynchronous operations.
- Date.now() (or process.hrtime()) around asynchronous operations (works okay, but ad-hoc, not comprehensive, cannot be easily correlated with other system activity, ...).
- You can use DTrace with system probes (e.g., syscall provider) or custom USDT probes to measure latency of any operation.
  - Example: read syscall (filesystem latency)
  - Example: gc-start, gc-done (USDT probes built into Node.js)
  - Example: restify probes (automatically-generated probes for each stage of your HTTP request pipeline)
  - Example: add your own probes with node-dtrace-provider.
- Demo.

Measuring synchronous operations

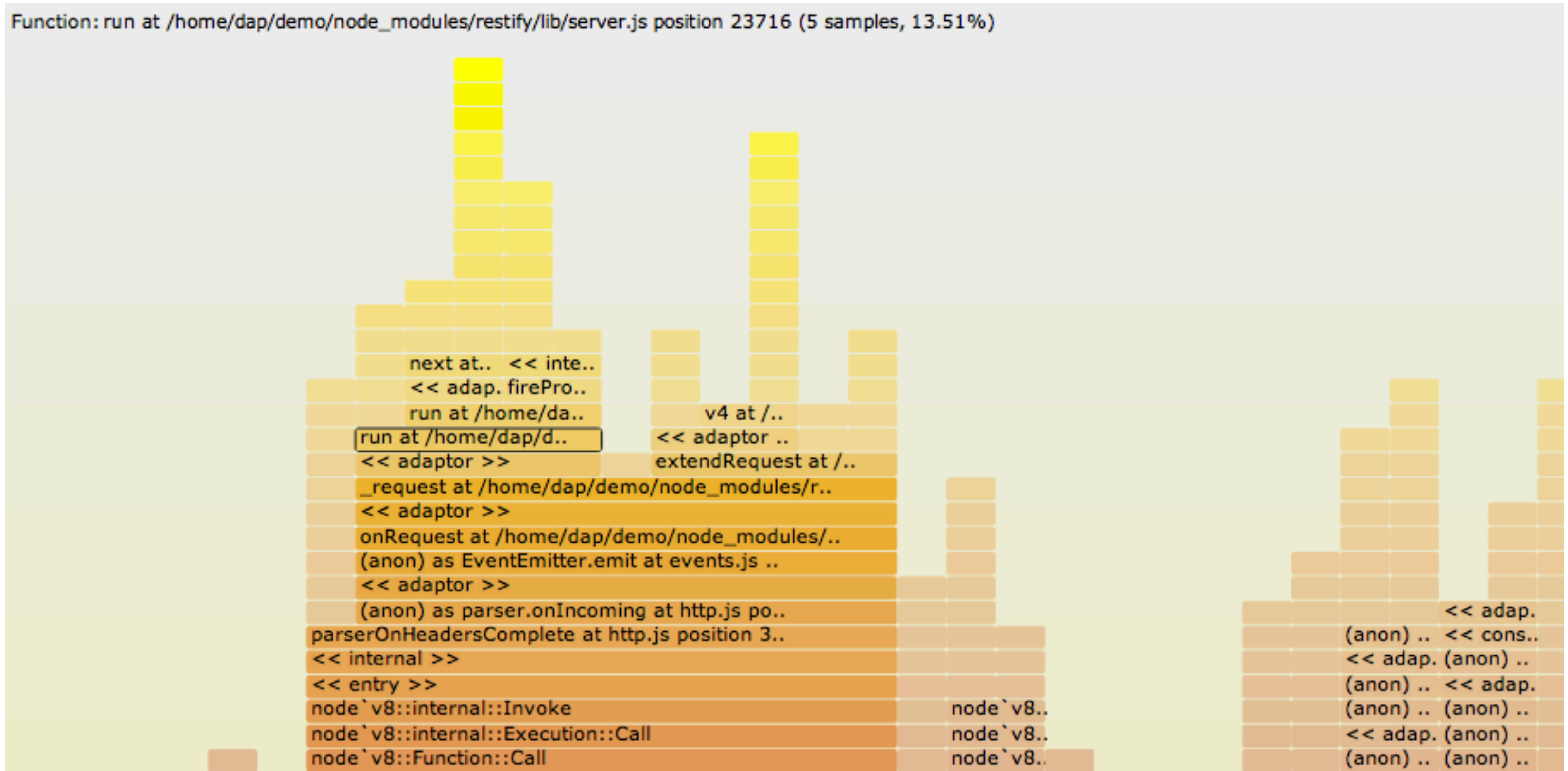
Joyent

- If your program is not waiting for something else, it must be doing something itself (i.e. running on CPU). But what?
- System probes (e.g., syscall provider) may still be useful.
- Use DTrace profile provider + `jstack()` action to sample stacks and see what your program is actually doing.
  - Includes JavaScript, C++, libraries, kernel (if desired)
  - Very little runtime overhead (can be run in **production**)
  - Can be turned on and off dynamically (no need to restart your app)
- Visualize output with a **flame graph**. See node-stackvis.

Node.js Flame Graph

Joyent

- Visualizing profiling output:



- Full, interactive version:
  http://www.cs.brown.edu/~dap/restify-flamegraph.svg

Wrap-up

Joyent

- DTrace helps with **data-driven** performance analysis. Measure performance of the **whole system** in both dev and **production**.
  - For asynchronous operations: use system probes and USDT probes.
  - For synchronous operations: use system and USDT probes, or stack sampling with flame graphs.
- What kinds of operations are interesting for your app?
  - Add custom instrumentation to your Node app: https://github.com/chrisa/node-dtrace-provider
  - Restify example: http://mcavage.github.com/node-restify/#DTrace http://mcavage.github.com/presentations/dtrace_conf_2012-04-03/
- What would you expect to see in a flame graph of your app?
  - Make your own flame graphs: http://dtrace.org/blogs/dap/2012/04/25/profiling-node-js/

Joyent

DTrace, Node.js, and Flame Graphs

NodeConf 2012
David Pacheco (@dapsays)
Joyent