Node.js Workloads and Performance Optimization Strategies

List of Applications/Workloads

- AcmeAir - an airline booking system from nodejs.org,
- Node-DC-EIS - Data Center use cases
- Lets-Chat - a chat application,
- Ghost - blogging platform,
- Many micro-benchmarks from Node.js and V8 test suite, and
- Octane, Kraken – JavaScript (V8, Chakra) specific benchmarks

External partnerships and Node.js Applications/Workloads

- Bitnami (Lets-Chat, Ghost)
- Walmart (Server Side Rending use case via Node-DC-EIS)
- Netflix (SSR via Node-DC-EIS)
- NodeSource
- NearForm

Using Node.js

- This is a tool for working in the non-blocking, event driven I/O paradigm
- Node.js is not meant to solve compute scaling problem.
- Node really shines in building fast, scalable network applications due to it’s capability to handle huge number of simultaneous connections with high throughput.

Optimizing Node.js runtime

- Find opportunities in Node.js runtime,
  - V8, libuv, native libraries (JS API interface)
- Growing Intel stickiness
  - Single core performance optimizations
  - Using IA specific optimization strategies,
    - Compiled code by native compiler,
    - Code gen in V8,
    - Use of Intel optimized libc/libm functions),
  - PGO, LTO optimizations
  - Core Scaling optimizations
  - Expose overall Intel platform capabilities
    - Crystal Ridge, FPGA, NICs, etc
  - Increasing use of Intel’s software technology
    - DPDK
- OS specific optimizations,
  - Huge page support

Optimizing Node.js Application

- Optimization opportunities in Node.js Runtime
- Application specific optimization,
  - various “node_module” usages (not all but heavily used ones from npmjs.org)
    - bcrypt.js
  - Contribution to original repository
- Performance of Node.js application in a cloud, containerized environment