Parsing Millions of URLs per Second

Yagiz Nizipli 🇺🇸

GitHub: https://github.com/anonrig

Daniel Lemire 🇨🇦

GitHub: https://github.com/lemire

Software performance

- Reduces cost (AWS, Azure)
- Improves latency
- Reduce complexity (parallelism, caching)
- Don't cause climate change

State of Node.js Performance 2023

> Since Node.js 18, a new URL parser dependency was added to Node.js — Ada. This addition bumped the Node.js performance when parsing URLs to a new level. Some results could reach up to an improvement of 400%. (State of Node.js Performance 2023)

> Just had a benchmark for a code change go from 11 seconds to complete down to about half a second to complete. This makes me very happy.

James Snell, Cloudflare

Referencing adding Ada URL to Cloudflare Workers

Structure of an URL

Example: https://user:pass@example.com:1234/foo/bar?baz#quu

- protocol
- user name, password
- hostname
- port
- pathname
- search
- hash

Examples

- non-ASCII: `http://□□□□.□□`
- File: `file:///foo/bar/test/node.js`
- JavaScript: `javascript:alert("node is awesome");`
- Percent Encoding: `https://\%E4\%BD\%A0/foo`
- Pathname with dots: `https://example.org/./a/../b/./c`
- Ipv4 address with hex/octal digits: `https://127.0.0x0.1`

WHATWG URL

input string	**`https://7-Eleven.com/Home/../P/Montréal`**
PHP	unchanged
Python	unchanged
WHATWG URL	`https://xn--7eleven-506c.com/Home/P/Montr%C3%A9al`
curl 7.87	`https://7-Eleven.com/P/Montr%C3%A9al`
Go runtime (`net/url`)	`https://7-Eleven.com/Home/../P/Montr%C3%A9al`

Assumptions

> Does URL parsing really matter? Is it bottleneck to some performance metric? Tbh i care more about JS runtimes to handle CI/CD processes faster and more parallelized.

> URLs are free, you don't gain anything by overloading them.

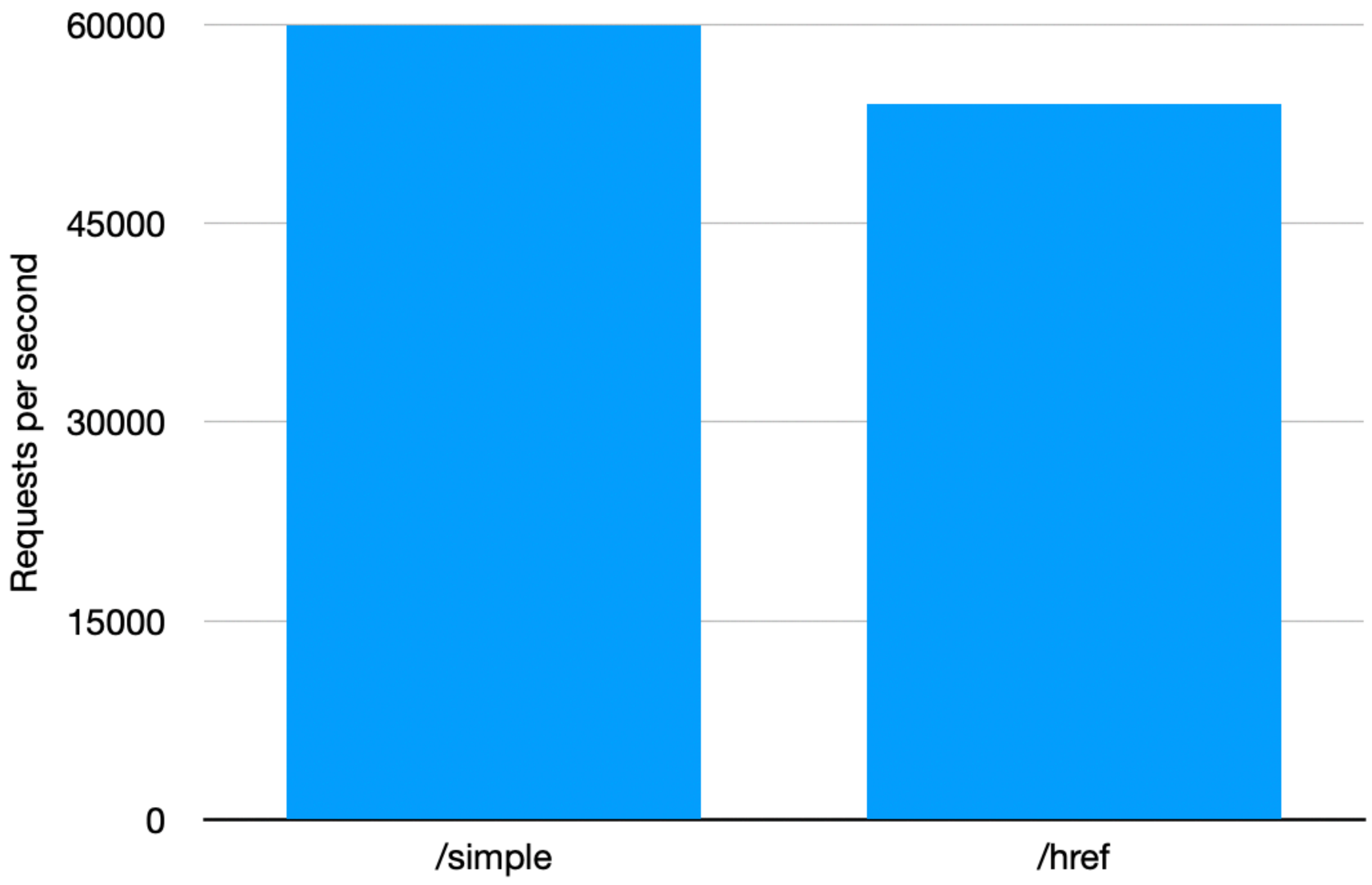
HTTP Benchmark

```
const f = require('fastify')()

f.post('/simple', async (request) => {
    const { url } = request.body
    return { parsed: url }
})

f.post('/href', async (request) => {
    const { url } = request.body
    return { parsed: new URL(url).href }
})
```

Input:

```
{ "url": "https://www.google.com/hello-world?query=search\#value" }
```

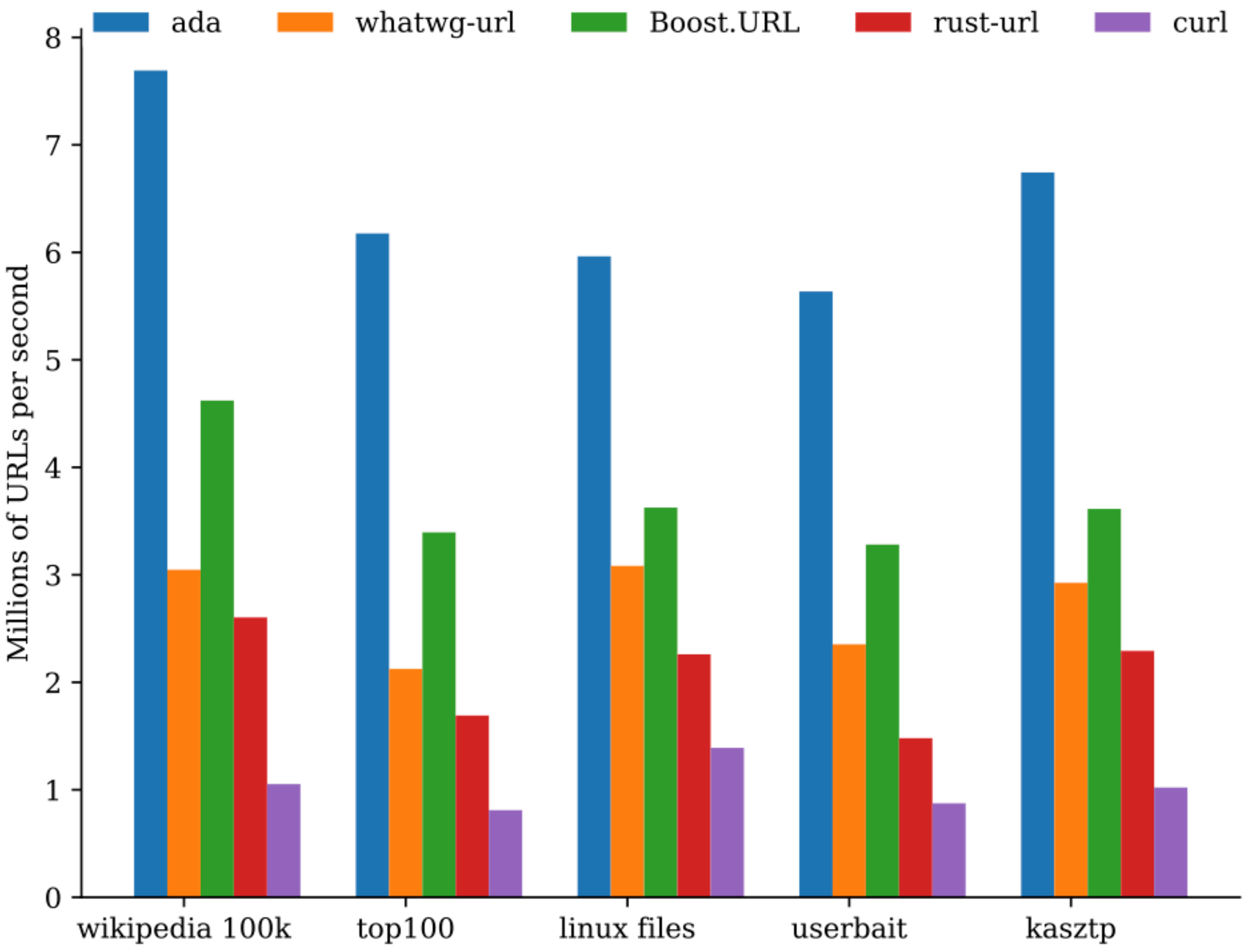
URL parsing was a bottleneck in Node 18.15



Wrote a C++ library (called Ada)

- Named after Ada Nizipli
- Full WHATWG URL support
- No dependency, full portability
- Over 20,000 lines of code
- Six months of work, 25 contributors
- Apache-2.0, MIT licensed
- Available at https://github.com/ada-url/ada

- 6 million URLs parsed/second
- Apple M2, LLVM 14
- Wide range of realistic data sources
- Faster than alternatives in C, C++, Rust
- WHATWG URL: whatwg-url and rust-ur



Trick 1: perfect hashing

```cpp
std::string_view names[] = {"http", " ",   "https", "ws",
                            "ftp",  "wss", "file",  " "};
enum type : uint8_t { HTTP, NOT_SPECIAL, HTTPS, WS, FTP, WSS, FILE};

type get_scheme_type(std::string_view scheme) noexcept {
  int hash_value = (2 * scheme.size() + scheme[0]) % 8;
  const std::string_view target = names[hash_value];
  if (target == scheme) {
    return type(hash_value);
  } else {
    return NOT_SPECIAL;
  }
}
```

Trick 2: use memoization (tables)

https://en.wikipedia.org/wiki/Memoization

```
uint8_t contains_bad_char(unsigned char* input, size_t length) {
  uint8_t accumulator = 0;
  for (size_t i = 0; i < length; i++) {
    accumulator |= is_bad_char[input[i]];
  }
  return accumulator;
}
```

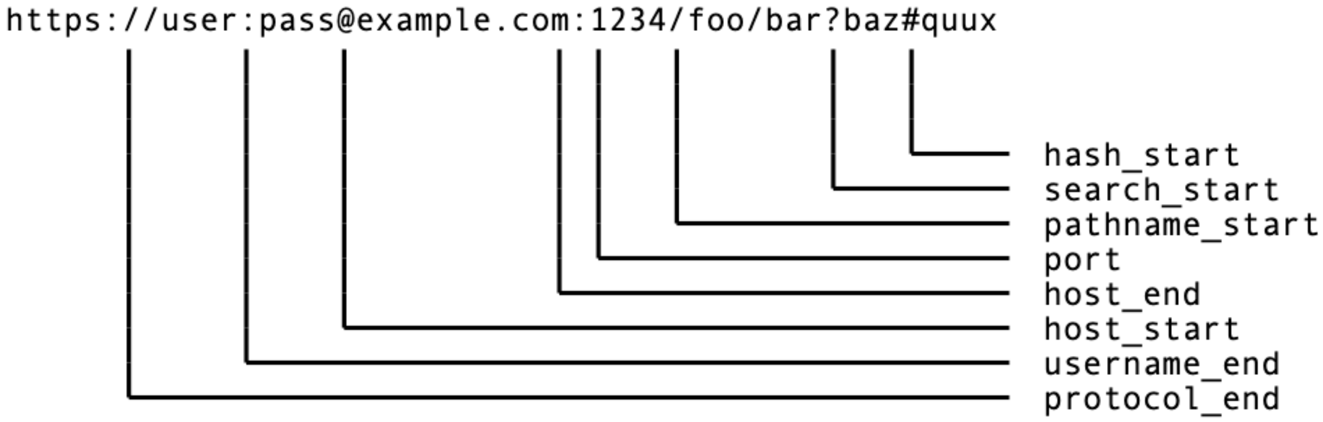
Trick 3: use vectorization

Do no process byte-by-byte when you can process 16-byte by 16-byte.

```cpp
bool has_tabs_or_newline(std::string_view user_input) {
  size_t i = 0;
  const __m128i mask1 = _mm_set1_epi8('\r');
  const __m128i mask2 = _mm_set1_epi8('\n');
  const __m128i mask3 = _mm_set1_epi8('\t');
  __m128i running{0};
  for (; i + 15 < user_input.size(); i += 16) {
    __m128i word = _mm_loadu_si128(user_input.data() + i);
    running = _mm_or_si128(
        _mm_or_si128(running, _mm_or_si128(
          _mm_cmpeq_epi8(word, mask1),
          _mm_cmpeq_epi8(word, mask2))),
          _mm_cmpeq_epi8(word, mask3));
  }
  return _mm_movemask_epi8(running) != 0;
}
```

Efficient C++/JavaScript bridge

- Passing multiple strings is expensive.
- Pass one string with offsets.



```
https://user:pass@example.com:1234/foo/bar?baz#quux
```

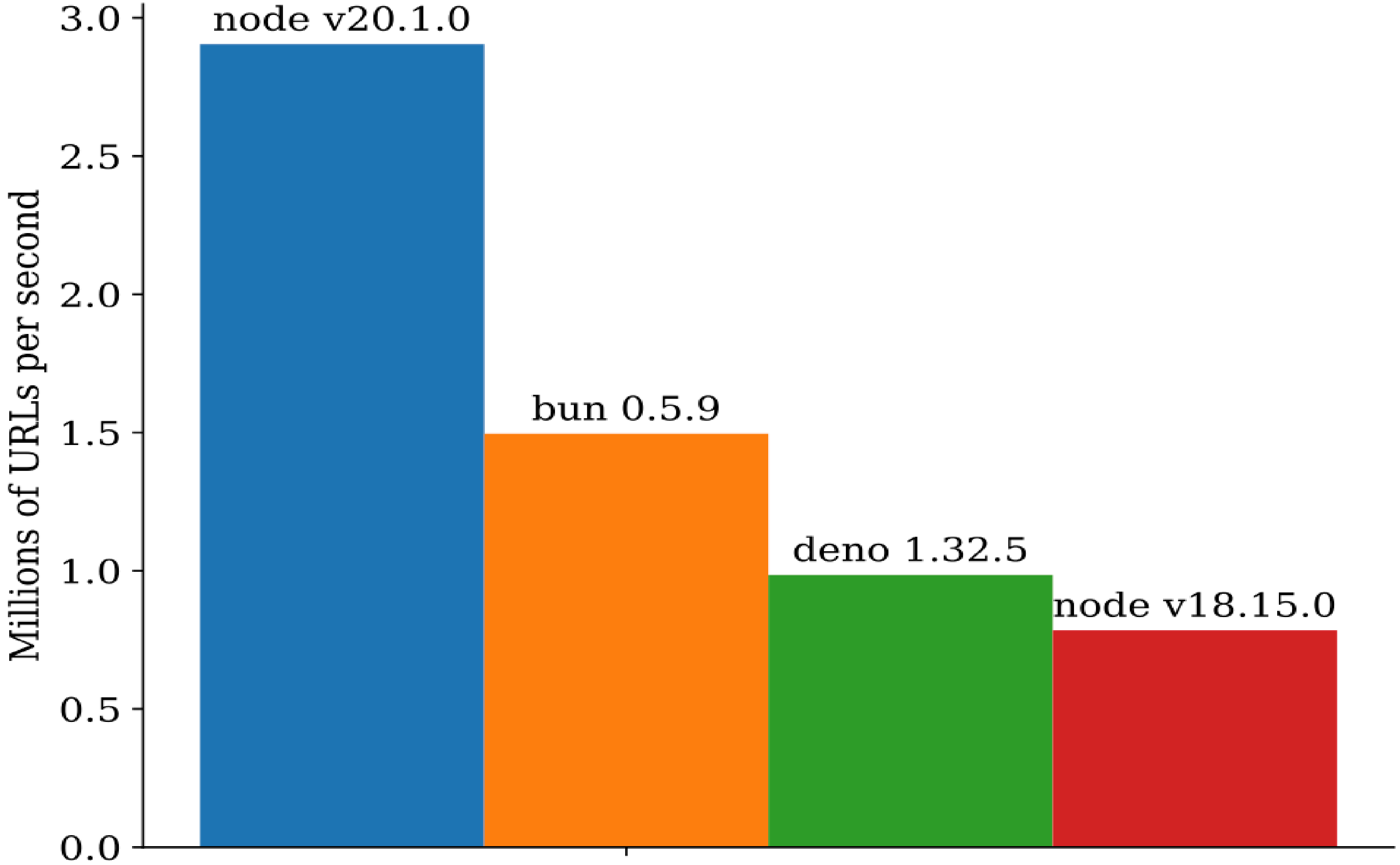
hash_start
search_start
pathname_start
port
host_end
host_start
username_end
protocol_end

JavaScript Benchmark

```
bench(filename, () => {
    for (let i = 0; i < lines.length; i++) {
      try {
        length += new URL(lines[i]).href.length;
        good_url++;
      } catch (e) {
        bad_url++;
      }
    }
    return length;
  });
```

https://github.com/ada-url/js_url_benchmark/

JavaScript Results



The Ada C++ library is safe and efficient

- Modern C++
- Sanitizers
- Fuzzing
- Unit tests

→ A few minor bugs were reported, mostly related to the standard. Quickly fixed.

Ada is available in the language of your choice

- JavaScript with Node.js
- C bindings at https://github.com/ada-url/ada
- Rust bindings at https://github.com/ada-url/rust
- Go bindings at https://github.com/ada-url/goada
- Python bindings at https://github.com/ada-url/ada-python
- R bindings at https://github.com/schochastics/adaR

Often the only way to get WHATWG URL support!

Links

- https://www.ada-url.com (includes a playground)
- @yagiznizipli's blog: https://www.yagiz.co
- @lemire's blog: https://lemire.me