node.js

ry@tinyclouds.org

April 17, 2010

node.js : Evented Server-side Javascript

Good at handling lots of different kinds of I/O at the same time.

Achieves this by all making network I/O nonblocking and all file I/O asynchronous. (Almost all.)

```
http = requrie('http');
net = requrie('net');
c = 0;

http.createServer(function(req, res){
  c++;
  res.writeHead(200);
  res.end('hello world\n');
}).listen(8000);

net.createServer(function(socket){
  socket.write('connections: ' + c);
  socket.end();
}).listen(8001);
```

For a detailed introduction see

`http://jsconf.eu/2009/video_nodejs_by_ryan_dahl.html`

Speed

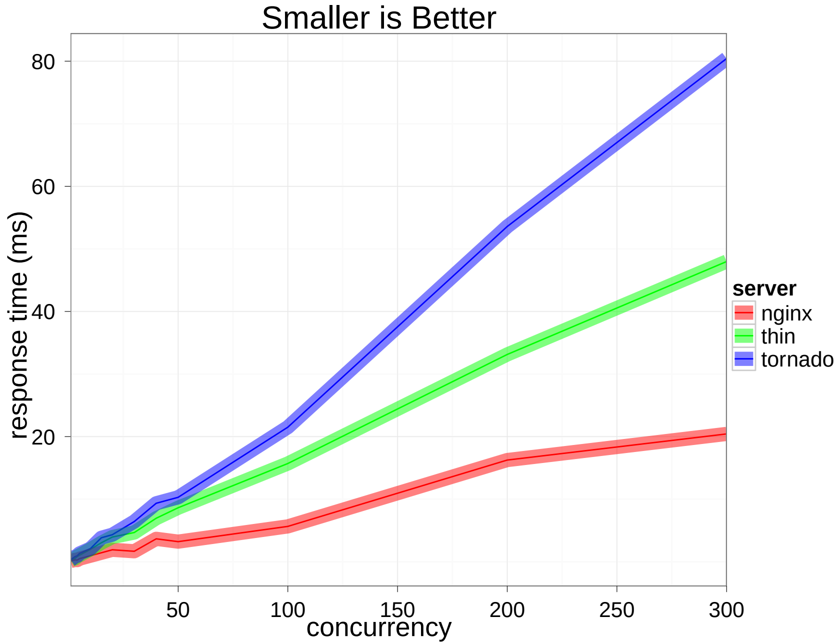
Benchmarks

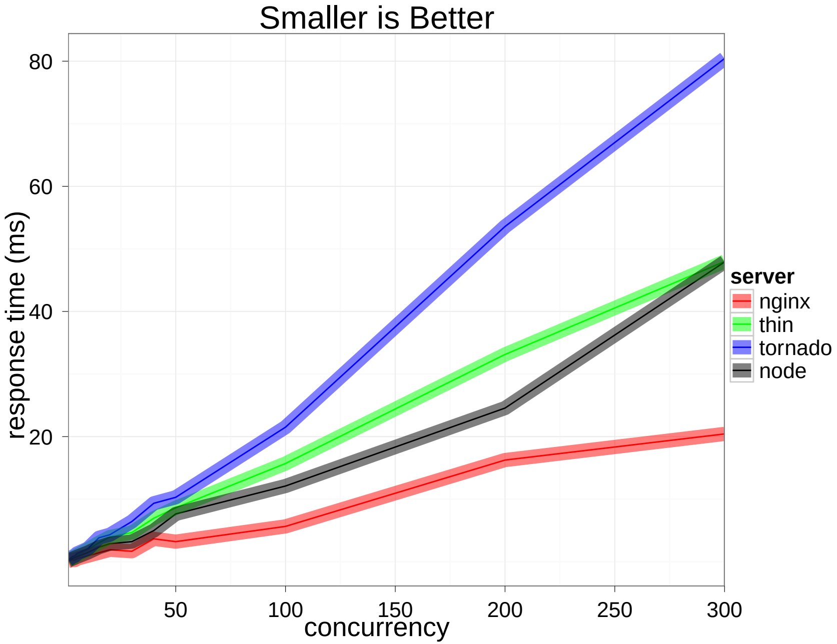
- nginx v0.7.65
- node v0.1.91
- tornado v0.2 (python 2.6.4)
- thin v1.2.7 (ruby 1.9.1-p376)

In Linux using a Intel Core 2 Due 2.53, 4 GB memory

The standard ‘hello world’ concurrency benchmark.

(Approximately 100 byte response for each.)


Smaller is Better
response time (ms)
80
60
40
20
50
100
150
200
250
300
concurrency
server
nginx
thin
tornado



Smaller is Better
response time (ms)
80
60
40
20
50
100
150
200
250
300
concurrency
server
nginx
thin
tornado
node

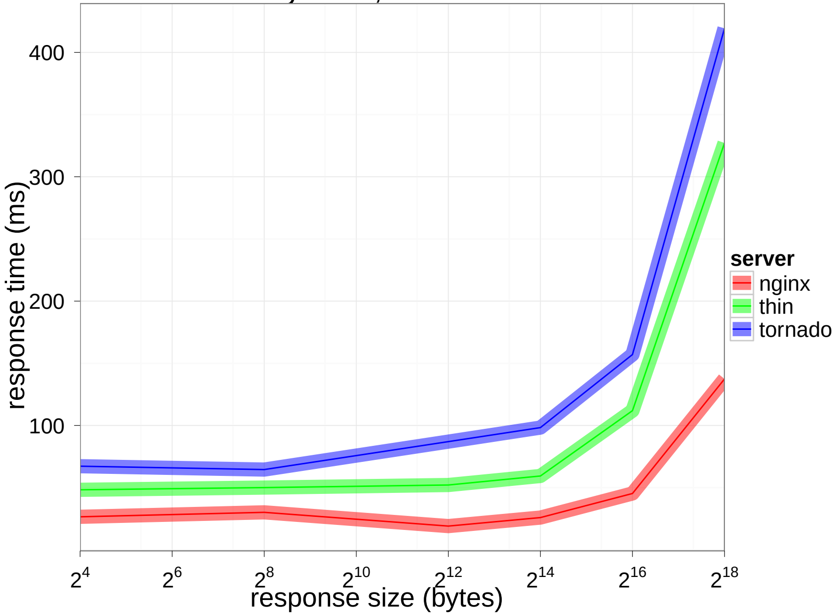
How do the servers preform when the concurrency is fixed at 300, but serve different response sizes.

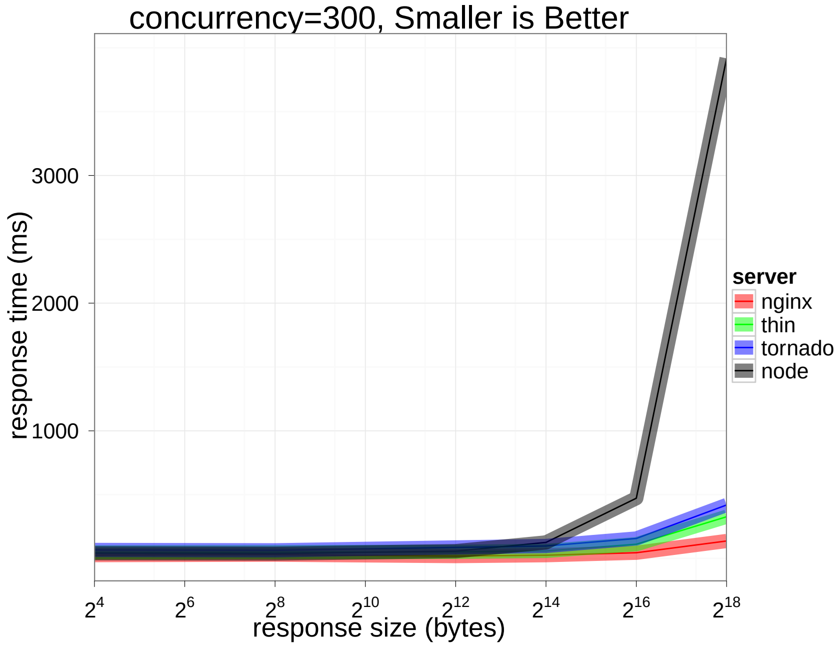
This is what the node version looks like, approximately

```
string = '';
for (i = 0; i < 16*1024; i++) {
  string += 'd';
}

http.createServer(function(req, res){
  res.writeHead(200);
  res.end(string, 'ascii');
});
```

concurrency=300, Smaller is Better




concurrency=300, Smaller is Better
response time (ms)
3000
2000
1000
2^{4}
2^{6}
2^{8}
2^{10}
2^{12}
2^{14}
2^{16}
2^{18}
response size (bytes)
server
nginx
thin
tornado
node

Wow. Node sucks at serving large files.

Well over 3 second responses for 256 kilobyte files at 300 concurrent connections.

What’s happening:

V8 has a generational garbage collector. Moves objects around randomly.

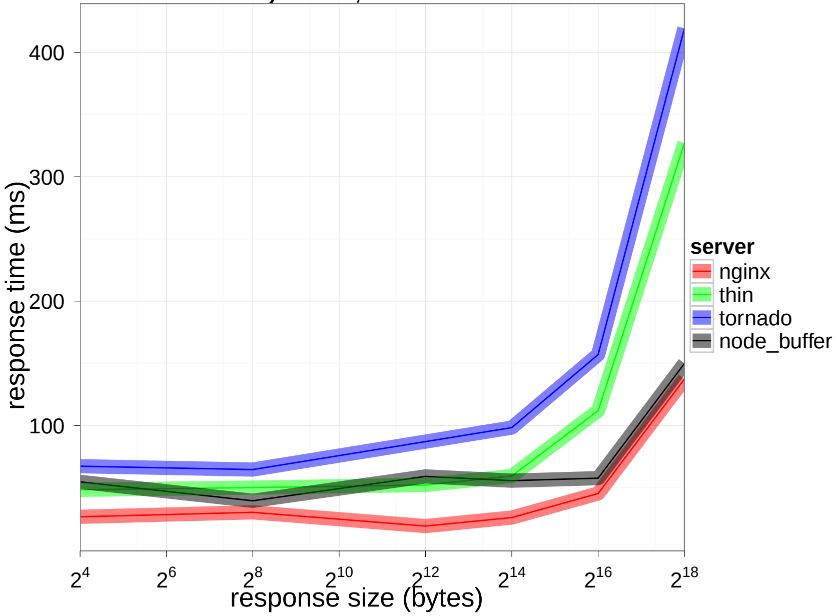
Node can’t get a pointer to raw string data to write to socket.

Using Node’s new **`Buffer`** object, the results change.

```
buffer = new Buffer(16*1024);
for (i = 0; i < buffer.length; i++) {
  buffer[i] = 100;
}

http.createServer(function(req, res){
  res.writeHead(200);
  res.end(buffer);
});
```

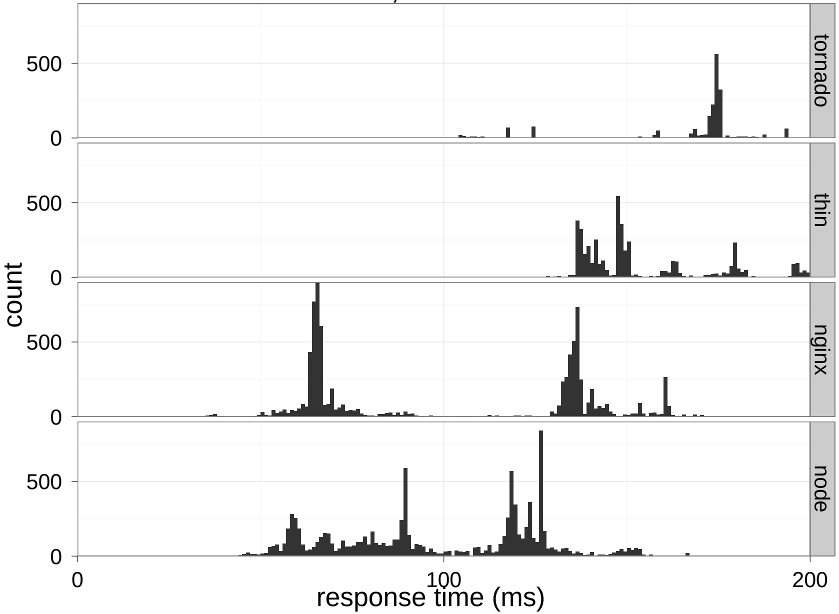
concurrency=300, Smaller is Better



Node can push a **`Buffer`** reference to socket pretty quickly.

A histogram of the 256 kilobyte case:

c=300, res=256kb



But the fact remains, pushing large strings to socket is slow.

Hopefully this can be mitigated in the future.

```
// Create
new Buffer(size);

// Modify
buffer[i] = value;

// memcpy
bufferA.copy(bufferB, offset);

// encode a string
buffer.write('string', 'utf8', offset);

// decode a string
buffer.toString('utf8', offset);
```

Since the 0.1.90 release, most of Node is written in JavaScript.

Just thin bindings.

Part of this rewrite was to expose the raw memory **`Buffer`** to users.

The other part was to unify Streams.

Node had all of these objects which would stream data, but had different interfaces.

HTTP request object: emitted a **`'body'`** event for each new chunk of data, and had a **`sendBody()`** method.

TCP socket: emitted a `'receive'` event for each new chunk of data, and had a **`send()`** method.

Stdio: emitted a `'data'` event for each chunk on stdin, and had a **`write()`** method for stdout.

`'data', 'receive', 'body'`

`sendBody(), send(), write()`

I slowly realized that I should stop making up different names for these things.

With a unified interface a general purpose ‘pumping’ function, with all the proper **throttling** and error checking is possible!

```
pump(request, stdout, callback);

pump(request, file, callback);

pump(file, response, callback);
```

Stream interface is split into two parts: Readable and writable streams.

Some streams are both readable and writable.

Readable stream.

- Event: `'error'`
- Event: `'data'`
- Event: `'end'` (EOF or FIN)
- `pause()`
- `resume()`
- `destroy()`

Writable stream.

- Event: `'error'`
- Event: `'drain'`
- `write()`
- `end()`
- `destroy()`

Readable:

- **`stdin`**
- **`ServerRequest`**
- **`childProcess.stdout`**

Writable:

- **`stdout`**
- **`ServerResponse`**
- **`childProcess.stdin`**

Probably need to expand the interface for more complex types of throttling.

(E.G. a low water mark for when to draining.)

Other Improvements.

For technical reasons:

- UDNS has been replaced with C-Ares.
- GnuTLS has been replaced with OpenSSL.

Side effect: Node no longer depends on any GPL or LGPL libraries.

The REPL library supports connecting to arbitrary sockets, not just stdio.

So you can setup a UNIX socket server at **`/tmp/my_node_app.sock`** and ‘telnet’ into it with the **`socat`** utility.

```
net.createServer(function (socket) {
  repl.start('my app> ', socket);
}).listen('/tmp/my_node_app.sock');
```

`promise.wait()` was removed.

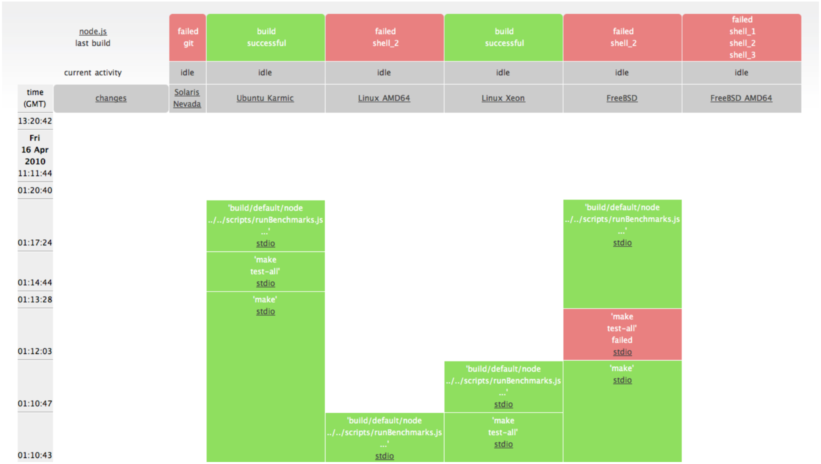
This was a kind of cooperative threading thing. Execution would jump to a new stack when called.

Coroutines complicate the mental model while adding only cheap syntactic pleasures.

Must worry about I/O occurring in all function calls. (They might call **`wait()`**.) The user needs to make their functions coroutine safe!

Cooperative threading of any sort is a bad idea.

Build Bot

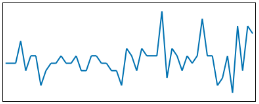


	node.js last build	failed git	build successful	failed shell_2	build successful	failed shell_2	failed shell_1 shell_2 shell_3
	current activity	idle	idle	idle	idle	idle	idle
time (GMT)	changes	Solaris Nevada	Ubuntu Karmic	Linux AMD64	Linux Xeon	FreeBSD	FreeBSD AMD64
13:20:42							
Fri 16 Apr 2010 11:11:44							
01:20:40							
01:17:24			'build/default/node ../../scripts/runBenchmarks.js ...' stdio			'build/default/node ../../scripts/runBenchmarks.js ...' stdio	
01:14:44			'make test-all' stdio				
01:13:28			'make' stdio				
01:12:03						'make test-all' failed stdio	
01:10:47					'build/default/node ../../scripts/runBenchmarks.js ...' stdio	'make' stdio	
01:10:43				'build/default/node ../../scripts/runBenchmarks.js ...' stdio	'make test-all' stdio		

Build Bot

Linux AMD64

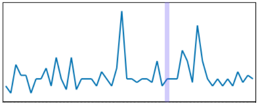
Static Http Server



Legend: t

move mouse over graph *Shift-click to place baseline*

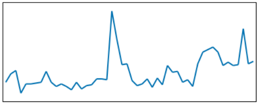
Timers



Legend: t

rb98cd6753b4: 2827.00 ms +/- *Shift-click to place*

Process Loop

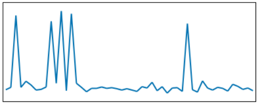


Legend: t

move mouse over graph *Shift-click to place baseline*

Linux Xeon

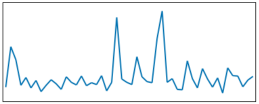
Static Http Server



Legend: t

move mouse over graph *Shift-click to place baseline*

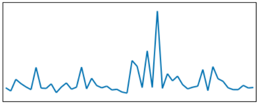
Timers



Legend: t

move mouse over graph *Shift-click to place baseline*

Process Loop



Legend: t

move mouse over graph *Shift-click to place baseline*

The Project Is Growing

42 releases. Current version: v0.1.91

63 contributors.

6000 lines of JavaScript, 11000 lines of C++

1100 people on the mailing list, 1400 watchers on GitHub.

http://nodejs.org/

http://wiki.github.com/ry/node/modules

http://howtonode.org/

ry@tinyclouds.org