Joyent

node.js

ryan@joyent.com

November 4, 2010

Joyent

node.js is a set of bindings to the V8 javascript VM.

Allows one to script programs that do I/O in javascript.

Focused on performance.

Joyent

First a silly (aging) benchmark:

```
100 concurrent clients
1 megabyte response

node     822 req/sec
nginx    708
thin      85
mongrel    4

(bigger is better)
```

Joyent

The code:

```
1 http = require('http')
2
3 b = new Buffer(1024*1024);
4
5 http.createServer(function (req, res) {
6   res.writeHead(200);
7   res.end(b);
8 }).listen(8000);
```

Joyent

Please take with a grain of salt. Very special circumstances.

NGINX peaked at 4mb of memory

Node peaked at 60mb.

Joyent

I/O needs to be done differently.

Joyent

Many web applications have code like this:

```
result = query('select * from T');
// use result
```

What is the software doing while it queries the database?

Joyent

In many cases, just waiting for the response.

Joyent

Modern Computer Latency

L1: 3 cycles

L2: 14 cycles

RAM: 250 cycles

DISK: 41,000,000 cycles

NETWORK: 240,000,000 cycles

`http://duartes.org/gustavo/blog/post/what-your-computer-does-while-you-wait`

“Non-blocking”

L1, L2, RAM

“Blocking”

DISK, NETWORK

Joyent

```
result = query('select * from T');
// use result
```

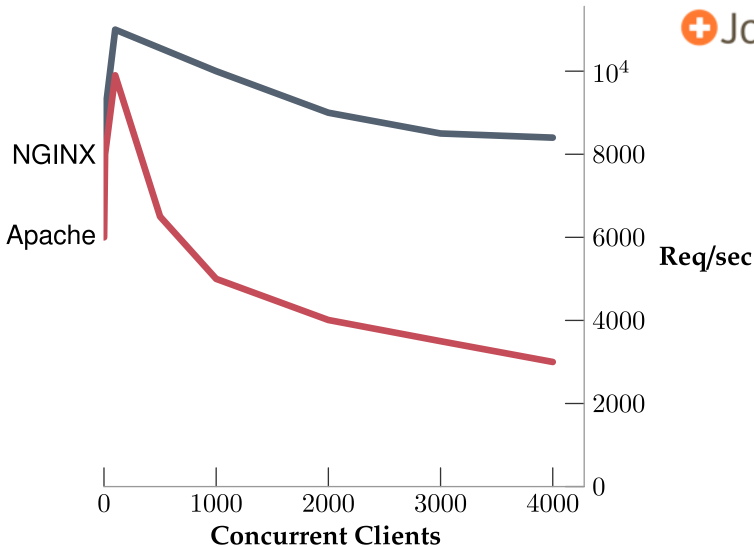
BLOCKS

Better software can multitask.

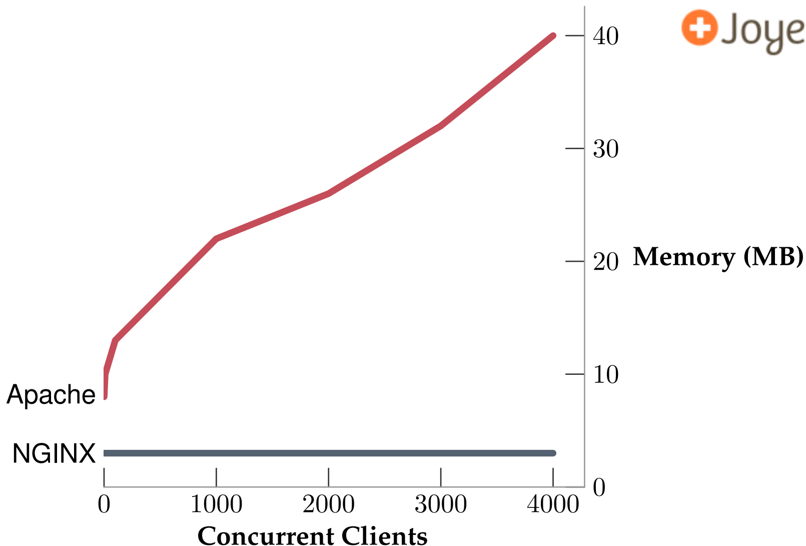
Other threads of execution can run while waiting.

Joyent

Is that the best that can be done?



http://blog.webfaction.com/a-little-holiday-present



http://blog.webfaction.com/a-little-holiday-present

Joyent

Apache vs NGINX

The difference? Apache uses one thread per connection.

NGINX doesn't use threads. It uses an **event loop**.

Joyent

- Context switching is not free
- Execution stacks take up memory

For massive concurrency, **cannot** use an OS thread for each connection.

Joyent

Green threads or coroutines can improve the situation dramatically

BUT there is still machinery involved to create the **illusion** of holding execution on I/O.

Joyent

Code like this

```
result = query('select..');
// use result
```

either **blocks the entire OS thread** or implies **multiple execution stacks**.

Joyent

But a line of code like this

```
query('select..', function (result) {
  // use result
});
```

allows the program to return to the event loop immediately.

Joyent

```
query('select..', function (result) {
  // use result
});
```

This is how I/O should be done.

- Good **servers** are written with **non-blocking I/O**.
- **Web browsers** are scripted with **non-blocking I/O**.
- Lots of interest in building good JavaScript VMs (Google, Mozilla, Apple, Microsoft)

Build the foundations of a server–allow the rest to be customized.

JavaScript

Node standard library

C

Node bindings

V8

thread pool

(libeio)

event loop

(libev)

Joyent

Examples

using node 0.2.4

Joyent

Download, configure, compile, and **`make install`** it.

`http://nodejs.org/`

No dependencies other than Python for the build system. V8 is included in the distribution.

Joyent

```
setTimeout(function () {
  console.log('world');
}, 2000);
console.log('hello');
```

A program which prints “hello”, waits 2 seconds, outputs “world”, and then exits.

```
setTimeout(function () {
  console.log('world');
}, 2000);
console.log('hello');
```

Node exits automatically when there is nothing else to do.

Joyent

```
% node hello-world.js
hello
```

2 seconds later...

```
% node hello-world.js
hello
world
%
```

Joyent

```
1 dns = require('dns');
2
3 console.log('resolving yahoo.com...');
4
5 dns.resolve('yahoo.com', function (err, addresses) {
6   if (err) throw err;
7   console.log('found: %j', addresses);
8 });
```

A program which resolves **yahoo.com**

```
% node resolve-yahoo.js
resolving yahoo.com...
found: ['67.195.160.76','69.147.125.65','72.30.2.43',
'98.137.149.56','209.191.122.70']
%
```

Joyent

Node includes support for DNS and HTTP out of the box.

Joyent

A simple HTTP server:

```
1 http = require('http');
2
3 s = http.Server(function (req, res) {
4   res.writeHead(200);
5   res.end('hello world\n');
6 });
7
8 // Listen on port 8000
9 s.listen(8000);
```

Joyent

```
% node http-server.js &
[1] 9120
% curl http://127.0.0.1:8000/
hello world
%
```

Joyent

Goal:

```
% curl http://127.0.0.1:8000/yahoo.com
query: yahoo.com
['67.195.160.76','69.147.125.65','72.30.2.43',
'98.137.149.56','209.191.122.70']
%
% curl http://127.0.0.1:8000/google.com
query: google.com
['74.125.95.103','74.125.95.104','74.125.95.105',
'74.125.95.106','74.125.95.147','74.125.95.99']
%
```

Joyent

```
1  dns = require('dns');
2  http = require('http');
3
4  var s = http.Server(function (req, res) {
5    var query = req.url.replace('/','');
6
7    res.writeHead(200);
8    res.write('query: ' + query + '\n');
9
10   dns.resolve(query, function (error, addresses) {
11     res.end(JSON.stringify(addresses) + '\n');
12   });
13 });
14
15 s.listen(8000);
```

Joyent

- the DNS resolution is async
- the HTTP server “streams” the response.

The overhead for each connection is low so the server is able to achieve good concurrency: it juggles many connections at a time.

Joyent

Abstracting Concurrency

The non-blocking *purity* allows I/O to be composable.

Joyent

As a larger example:

- Two servers in one process
- TCP server
- HTTP server
- When you connect to the TCP server, data is streamed to the HTTP server.

Joyent

First the TCP server

```
1 var net = require('net');
2 var queue = [];
3
4 var tcp = net.Server(function (socket) {
5   socket.write('waiting for req\n');
6   queue.push(socket);
7 });
8
9 tcp.listen(8000);
```

A function for pulling a socket off of the queue:

```
1  function getSocket () {
2    var socket;
3    // Get the first socket that is 'readable'
4    // I.E. Not closed.
5    while (queue.length) {
6      socket = queue.shift();
7      if (socket.readable) break;
8    }
9    return socket;
10 }
```

Now the HTTP server:

Joyent

```
1  http = require('http');
2
3  web = http.Server(function (req, res) {
4    res.writeHead(200);
5    var socket = getSocket();
6    if (socket) {
7      socket.pipe(res); // New in v0.3
8    } else {
9      res.end('No sockets\n');
10   }
11 });
12
13 web.listen(7000);
```

Joyent

Attempt to demonstrate live...

Chat

```
net = require('net');
peers = [];
s = net.Server(function (socket) {
  peers.push(socket);
  socket.on('data', function (d) {
    // write 'd' to all 'peers'
    // ...
    // ...
    // ...
    // ...
    // ...
    // ...
    // ...
    // ...
    // ...
    // ...
  });
});
s.listen(8000);
```

Joyent

Joyent

```
1  net = require('net');
2  peers = [];
3  s = net.Server(function (socket) {
4    peers.push(socket);
5    socket.on('data', function (d) {
6      // write 'd' to all 'peers'
7      var completed = [];
8      while (peers.length) {
9        var peer = peers.shift();
10       // write 'd' to 'peer'
11       // ...
12       // ...
13       // ...
14       // ...
15     }
16     peers = completed;
17   });
18 });
19 s.listen(8000);
```

Joyent

```
1  net = require('net');
2  peers = [];
3  s = net.Server(function (socket) {
4    peers.push(socket);
5    socket.on('data', function (d) {
6      // write 'd' to all 'peers'
7      var completed = [];
8      while (peers.length) {
9        var peer = peers.shift();
10       // write 'd' to 'peer'
11       if (!peer.writable) continue;
12       // ...
13       // ...
14       // ...
15     }
16     peers = completed;
17   });
18 });
19 s.listen(8000);
```

```
net = require('net');
peers = [];
s = net.Server(function (socket) {
  peers.push(socket);
  socket.on('data', function (d) {
    // write 'd' to all 'peers'
    var completed = [];
    while (peers.length) {
      var peer = peers.shift();
      // write 'd' to 'peer'
      if (!peer.writable) continue;
      peer.write(socket.remotePort + '> ');
      peer.write(d);
      completed.push(peer);
    }
    peers = completed;
  });
});
s.listen(8000);
```

Joyent

Joyent

Questions...?

http://nodejs.org/

ryan@joyent.com

Joyent

Other links

http://howtonode.com/

http://developer.yahoo.com/yui/theater/video.php?v=dahl-node

http://www.youtube.com/watch?v=F6k8lTrAE2g