How to develop internet apps with R and Node.js

Alberto Santini
Monte dei Paschi di Siena

Outline

- What is ConPA?
- ConPA Architecture.
- Frontend with Yahoo! User Interface library.
- Backend with Node.js.
- node-rio, a Node.js module integrating R via Rserve.

ConPA: open source finance app

ConPA is an asset allocation application.

It provides the optimal weights for each assets, retrieving the historical prices from Yahoo! Finance, applying the constraints and optimizing the weights with the Markowitz algorithm.

It provides also the performances of the portfolio and the implied volatility of the assets.

Why R and JavaScript?

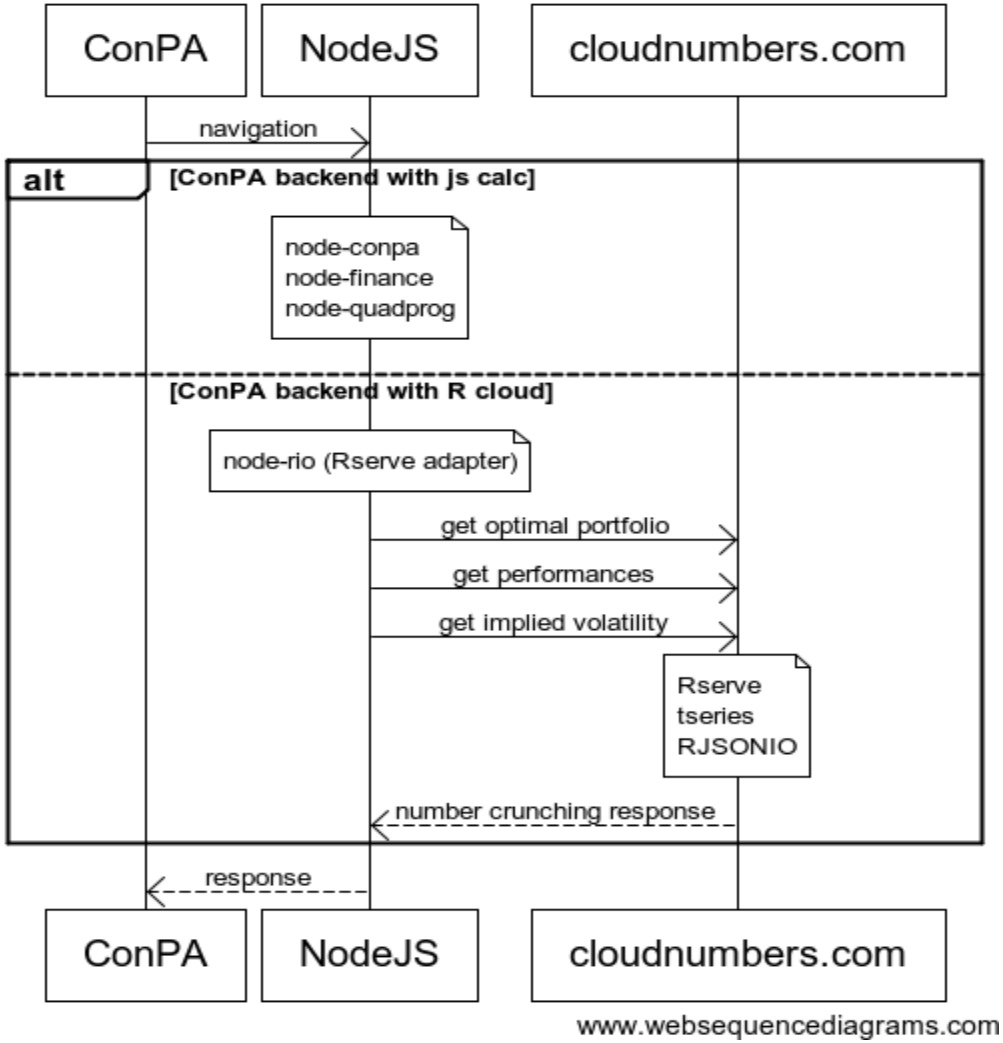
	Julia 3d331c58	NumPy 1.5.1	Matlab R2011a	Octave 3.4	R 2.14.2	JavaScript V8 3.6.6.11
fib	1.97	30.79	1360.47	2463.97	224.96	1.48
parse_int	1.27	16.58	827.13	6871.04	342.98	2.12
quicksort	1.27	59.73	135.51	3357.78	730.70	6.61
mandel	7.19	32.28	66.27	870.55	156.68	5.73
pi_sum	0.74	18.59	1.09	356.08	166.41	0.75
rand_mat_stat	4.17	39.59	12.20	57.39	22.19	8.32
rand_mat_mul	0.99	1.21	0.78	1.69	8.65	42.32

Figure: benchmark times relative to C++ (smaller is better).

C++ compiled by GCC 4.2.1, taking best timing from all optimization levels (-O0 through -O3).

source: http://julianlang.org

ConPA Architecture



Frontend with YUI

YUI is a free, open source JavaScript and CSS framework for building anything from simple non-interactive views to rich applications with URL-based routing, data binding, and full client-server synchronization.

```
Y.io("/ConPA/getOptimalPortfolio", {
        method: "POST",
        data: ptfData, // object literal containing assets, constraints, etc.
        on: {
            success: function (id, o) { // callback
                var res = JSON.parse(o.responseText);
                ... // res contains the response.
            }
           ... // handling failure callback, timeout, etc.
        }
});
```

Node.js

Node.js is a platform built on Chrome's JavaScript runtime for easily building fast, scalable network applications.

Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices.

source: http://nodejs.org

Backend with Node.js

```
 // express: high performance, high class web development for Node.js
var express = require('express'),
      finance = require('finance'); // ConPA backend module
...
var app = express.createServer();
...
// route defined in "finance" module
app.post('/ConPA/getOptimalPortfolio', function (req, res) {
  // calling the 'finance' methods using node-rio module
  // or calling persistence layer
}
...
app.listen(process.env.PORT || 8001);
```

node-rio

RIO, R Input Output, is a Node.js module: it connects an app to **Rserve**, a TCP/IP server which allows other programs to use facilities of **R**.

RIO supports double, double array, string and string array objects.

RIO supports also the plain text authentication, if Rserve is configured for that capability.

node-rio api

evaluate(command, options)

Execute a command contained in a string.

sourceAndEval(filename, options)

Read a content of a filename and execute it.

The script needs using rjsonio package to deserialize the parameter, passed as JSON object, and to serialize the response for the callback.

node-rio api options

```
options = { // evaluate
    callback: function (res) {
        if (res !== false) {
            sys.puts(res);
        } else {
            sys.puts("Rserve call failed");
        }
    },
    host = "127.0.0.1",
    port = 6311, // default Rserve port
    user = "anon",
    password = "anon"
}
options = { // sourceAndEval
    entryPoint: "main", // entryPoint to be called
    data: { foo: "bar" } // an object literal stringified and passed to entryPoint
}
```

node-rio evaluate internals

```
conn = connect(host, port, callback);
binary(conn)
      .buffer("rServeIdSignature", 4) // Rsrv
      .buffer("rServeProtocolVersion", 4) // 0103
      .buffer("rServeCommunicationProtocol", 4) // QAP1
      .skip(4)
      .buffer("rServeCapabilities", 16)
      .tap(function (vars) {
          // sending finally login and command
      .buffer("dataHeader", 16)
      .buffer("dataPacketType", 1) // 0x10
      .buffer("lenDataPacket", 3)
      .tap(function (vars) {
          var rl = int24(vars.lenDataPacket);

          this.buffer("dataPacket", rl)
              .tap(function (vars) {
                  var res;

                  log("Data packet");
                  log(vars.dataPacket);

                  res = parse_SEXP(vars.dataPacket, 0);
                  log("Response value: " + res);

                  callback(res);
              });
      });
```

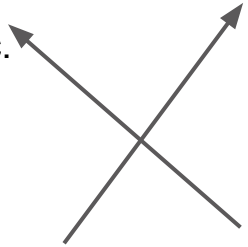
node-rio example I

```
rio = require("rio"); // loading the module rio

rio.evaluate("pi / 2 * 2");
rio.evaluate('c(1, 2)');
rio.evaluate("as.character('Hello World')");
rio.evaluate('c("a", "b")');
rio.evaluate('Sys.sleep(5); 11');
```

node-rio example II

```
// ex2.js file
function displayResponse(res) {
    ... // check error, validate input, etc.
    res = JSON.parse(res);
    ...
}
rio.sourceAndEval(__dirname + "/ex2.R", {
    entryPoint: "getOptimalPortfolio",
    data: {
        prods: ["IBM", "YHOO", "GOOG"]
        ...
    },
    callback: displayResponse
});
```



```
# ex2.R file
getOptimalPortfolio <- function (jsonObj) {
    o <- fromJSON(jsonObj)
    symbols <- o$prods
    ...
    return(toJSON(res))
}
```

Wrapping up: YUI, Node.JS & rio

- YUI frontend provides the user interface, calling the Node.js backend, using a standard RPC frontend approach: XHR, JSONP, etc.
- Node.js backend provides a flexible platform to handle finally the number crunching platform and the persistence layer.
- node-rio allows to connect a Rserve instance and a Node.js app without bindings.

Ready from prototype to large scale.

Thanks!

Alberto Santini
https://github.com/albertosantini

http://conpa.nodejitsu.com/ConPA/ConPA.html
http://conpa.nodester.com/ConPA/ConPA.html
http://as.no.de/ConPA/ConPA.html