# Chapter 1

# Introduction of Node.js and its web frameworks

*Kuan-Lin Chiu*

Node.js, created by Ryan Dahl, is aimed for "providing an easy way to build scalable network programs." In a short time since its initial release in 2009, it has quickly gathered much attention in the area of web application development and captured the interest of thousands of experienced developers. A package manager (Node Packaged Modules, NPM) was soon established and thousands of modules and interesting applications have been created. Even a number of innovative startups were spawned from this heat. Some people deem node as the next generation revolution of web technology.

What is new in this technology that attracts developer's eyes? How it fits into current growing need of web applications? What's the different between node and other long existing server-side JavaScript? This article will introduce its features, and then discuss where it excels other traditional approaches on certain application domain, and what kind of application it fits in. Later in the article will talk about some web frameworks of Node.js, introduce the innovative ideas and key aspects which benefit from the design of Node.js.

## 1.1 Node.js

### 1.1.1 What is Node.js?

Node.js, or just "node", is a Server-side JavaScript platform, which allows you to run JavaScript programs, without the browser, to do IO and system works on the server. Powered by Google V8 JavaScript engine, it's a set of libraries and bindings to the V8 virtual machine. The V8 engine itself and the core of node are implemented in C/C++, whereas the standard libraries and APIs on top are written in JavaScript. For extensibility, node follows CommonJS standard so that modules can be shared between other JavaScript platforms. It even allows developer to write C/C++ add-ons for performance critical component.

Node uses an event-driven, non-blocking I/O model, which makes it good at handling high concurrency, data IO intensive and real-time applications. The lightweight and efficient design is focus on performance and low memory consumption to support long-running, scalable network programs.

### 1.1.2 Server-Side JavaScript

One of the features that attract developers is it allows you to use one language, JavaScript, from browser to backend. Code can be reused on both side.

One of the features that attract developers is that it allows you to use one language, JavaScript, from browser to backend.

JavaScript is on its way to become a universal language and even dominate the frontend development. Frontend developer can easily get used to node without much learning curve since the API of node is designed to be familiar to client-side JS programmers. Node is a candy for frontend developers to set foot in the server-side. For the backend developers, no matter you use PHP/Ruby/Java as backend, you will probably pick up some JavaScript code to deal with your client. Modules and features such as Ajax or data validation are common in both sides. Code and module can be reuse for both browser and Server-side is obvious bene-

fits.

Server-Side JavaScript (SSJS) is not actually a novel technology. It was introduced by Netscape since 1994. There are several long existing SSJS solutions based on Rhino, SpiderMonkey¡Ketc, but none of them makes SSJS so well-known and acknowledged until node.js.

One of the reasons is timing. JavaScript was once treated as not a serious language but nowadays it's popular than ever. Since early 2000's, AJAX technology arising has changed the trend of web applications. Later JavaScript technologies such as jQuery further extend the use of JavaScript in frontend development. They allow developers to build rich UI and provide user experience mimic desktop rich client.

Another reason is performance. The high demand on speed of these JS front-end technologies exploded the new browser wars on their JavaScript engine. Google Chrome, one of the best competitors in this speed race, outperformed others on its release in 2008. Its V8 JavaScript engine is famous as one of chrome's feature but it's actually a standalone open source project. Ryan thus decided to build node on top of it. The V8 engine is constantly pushing the boundaries in being one of the fastest dynamic language interpreters on the planet. No other language is being pushed for speed as aggressively as JavaScript is right now. Compared to the interpreters for other server-side dynamic languages like Ruby, Python, PHP and Perl, JavaScript has incredibly fast runtimes.

The key difference that makes node success but other SSJS remain silent is that node focus on building a different kind of application. While other SSJS can do well on most dynamic web application, there're already lots of well-proofed and matured platform such as Python, Java, Ruby, PHP out there and solved the problems even better. On comparison, node is design to use an evented, non-blocking IO model to handle high concurrency applications, which traditional approaches may suffered. Choosing JavaScript is because it fits perfectly in this design. We will elaborate this point in the following section.

### 1.1.3 Evented, Asynchronous, Non-blocking IO model

The design of node is all about evented asynchronous, non-blocking IO model. Hence it's perfect for most IO-bound web applications.

The design of node is all about evented asynchronous, non-blocking IO model. Node is fast. But not only because of the superb performance of google V8 engine, it's fast in nature of its design. Most web applications are IO bound. The bottleneck of speed is the latency of IO. Accessing L1, L2, RAM are non-blocking. Disk and Network IO are definitely blocking. In most traditional synchronized programming, the code is written like:

```
var result = db.query("select.."); //wait
doSomething(result);
nextTask();
```

Here the query blocks the program from doing anything else until the query is returned. For an asynchronous, non-blocking design such as Node, the code is as following:

```
db.query("select..", function (result) {
  doSomething(result);
  });
nextTask();
```

The query takes an anonymous function as a Callback. The program doesn't wait for the query to be finished. It just executes nextTask() directly. When the query is done, node will then fire an event. The main thread then executes the callback function to deal with the result.

In comparison, in a single thread blocking program, to execute A, B, C, three IO operations, the executing time will be Sum(opA, opB, opC). But in a non-blocking design, it's Max(opA, opB, opC).

To achieve this design, node runs an evented model. The core of event system is implement using libev in Unix-like system and IOCP in windows. The life cycle is as following: The event loop first Initialize as an empty event loop. Executes non-IO code and add every I/O call to the event loop until reach the end of source code. Then the event loop starts iterating over a list of events and callbacks. Perform IO using non-blocking kernel facilities. Kernel noti?es

the Event Loop when done. Event Loop executes and removes a callback. Finally, program exits when Event Loop is empty. In this manner, we can use single thread to deal with high concurrency, IO intensive applications.

It's worth noticed that how JavaScript fits into this design. First, all the IO needs to be non-blocking. JavaScript at first can't actually do IO itself since it's not necessary its job in the browser. Ryan doesn't choose any other language which already has built-in blocking IO. Since IO is not a fundamental part of JavaScript, nothing had to be taken away or to add. He chose JavaScript and start from scratch with an all non-blocking IO system and libraries based on event loops. Second, JavaScript well supports callbacks. It's like functional language which has first-order function and lambda. It's easy to pass around anonymous callbacks. Finally, JavaScript is in nature single thread.

### 1.1.4 Event loop and single thread

Compare to multi-threading, the design of single thread with event loop is very efficient with massive concurrency.

The design of single thread with event loop is very efficient with massive concurrency.

Traditional programming designs that block on IO usually try to deal with these problems by spawning new process or thread. By using multi-tasking, other threads of execution can run while waiting. For example, Apache uses one thread per connection. But obviously this does not scale well for massive concurrency. Hundreds of concurrent connections mean Hundreds of threads. It costs huge memory and context switch overhead. In additional, locks, dead locks and race problem increase the complexity of program.

Of course there's other programing language can use event and asynchronous IO libraries to achieve the similar design, such as Twisted for Python, EventMachine for Ruby¡Ketc. They provide very good event loop platform and can create efficient servers. The problem is, most of the programming language itself, pretty much entire original libraries are mostly synchronous and blocking. In the event loop, one blocked then everything halts thus you can't make any blocking code. Therefore great care must be

taken when using a not native evented, non-blocking platform. Users must be very careful of every method they call and libraries they use to avoid blocking IO.

There's a good comparison between multi-thread approach and event loop, single thread approach. Apache use multi-threading while nginx runs an event loop thus it needs only small memory allocation.

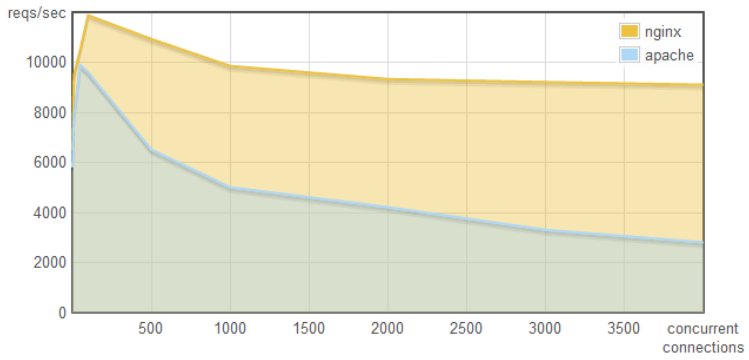


**Figure 1.1:** Performance reqs/sec

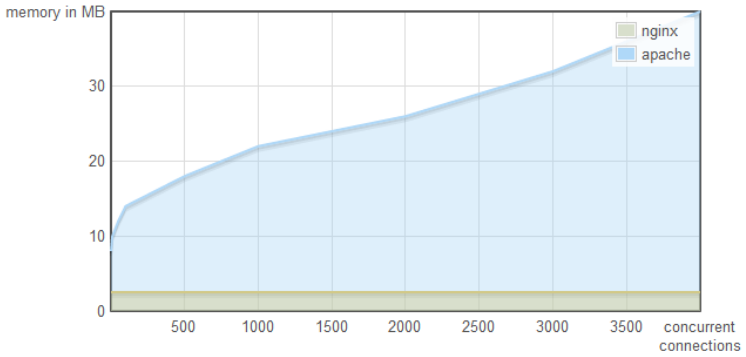


**Figure 1.2:** Memory consumption

### 1.1.5 Summary of Node.js

Although there're many interesting features in Node.js, there're still many criticisms and concerns remain to be clear.

1. It's still young, not mature yet.

Some might say it's not battle tested yet. But actually there're already many companies include node in their pioneer product. For example, LinkedIn (Mobile Web App), Yahoo! (Manhattan), Ebay (Data retrieval gateway), GitHub (for Downloads) and Palm/HP (in WebOS). And the popularity and community of node is rapidly increasing. The number of modules in Node Packaged Modules (NPM) is now reached more than 13000. The node in GitHub is also the second most popular Starred project, below bootstrap but above jQuery and Rails.

2. The design is not suitable for CPU intensive work.

3. The support to multi-core processor

There's a module cluster add the support of multi-tasking for node. Cluster is included since version 0.6 but still in the experimental phase. But in the latest 0.8 version it basically get completely rewrite and has large improvement.

4. Asynchronous programming

5. Difficult to debug

Most developers are used to synchronous programming style. When there're lots of IOs in order, it results deeply nested callbacks as callback hell. For some developers, the logic and flow control are somehow complex and confused thus hard to debug. This is not necessary true if you're familiar with functional programming and recursive calls. But it can still be solved by naming the callback and dividing functions into modules. There're also some modules support to arrange the control flow.

## 1.2 Web frameworks of Node.js

### 1.2.1 Express

Express is the most popular framework of node. (GitHub starred more than 7000.) It's already version 3.0 thus rather mature than most of frameworks of Node. Inspired by Ruby Sinatra, it is a minimal and flexible framework builds on top of Connect middleware. Connect is itself a framework of many middlewares wrapped together. Express expose them as its build-in functions for convenience.

Features include: routing, HTTP helpers, view (support many template engines, by default, jade) and Content negotiation. There's no model included in Express. You will have to build that yourself such as using Mongoose. It's not necessary MVC but can easily be combined with any known MVC library. It's also RESTful.

Express provides only basic foundation and tools you need but give you the utmost control over your application. It lacks of built in support for many things but is very extensible as it is so popular that most modules support it. So you can build almost everything based on it but would have to spend a lot of time to find appropriate packages and set them up to work with Express.

### 1.2.2 MVC frameworks

#### Geddy

Geddy is a modular, full-service MVC framework similar to Merb, Rails, Pylons, or Django. Compared to Express, it targets the same domain of Express but more robust with MVC. It's in incompatible with Express and can't use Connect middleware. Good for a yet still minimal basic MVC framework.

**TowerJS**

Tower is a Full Stack MVC framework for node.js and the browser. It's similar to Ruby on Rails and build on top of Express and Connect. Components includes: MongoDB (database), Redis (background jobs), CoffeeScript, Stylus (or LESS), Jasmine (tests), jQuery and Mocha. Noticed that it even include client side component, which is good for unified code on both side. Almost everything included, it might save you a lot of time.

### 1.2.3 Real-time synchronized models frameworks

Clients, Servers, and database all share a common model state and synchronize the changes. Thus the model-bound view can update itself in real-time.

Node.js makes building real-time web application easier than most other platforms. Some take the benefit of it and build innovative real-time frameworks which consist of features such as: Model/view bindings on the client, synchronized model state, subscription to database changes from the server and JavaScript code sharing on both client and the server. Meteor and Derby are the most popular frameworks of this sort.

The common features of them include:

1. JavaScript code sharing on both client and the server

Meteor can even access database on client side using the same API on server side.

2. Full development framework

All the core packages are included and integrated for you. They pretty much include all you need to build a real-time synchronization application between client and server.

3. Model/view bindings on the client

HTML generated from template. Variable bound to model. View updates automatically as the model changed.

4. Synchronized Model State

Clients, Servers, and database all share a common state and synchronize the changes. Subscriptions manage what data to publish where.

5. Live Rendering Update in real-time based on the synchronized model.

On the client side, view is binding with models. Models are synchronized across client, server and database. Subscriptions decide filter which models publish to where. Model-bound views recognize changes in their models. Thus the view is updated in real-time with the model changed anywhere.

Traditional way to write a real-time synchronized web application across multiple clients, server and database may take days even weeks. With the power of Meteor or Derby, it could be done within hours.

**Meteor**

Meteor is the most popular frameworks of its kind. It recently received 11M+ dollars in funding which somehow promising that it will remain supported and probably faster to become a stable framework. Meteor is an end-to-end framework which include everything you need. Meteor doesn't really follow Node.js community standards thus has more limitations. It doesn't use NPM. It has its full build environment and only uses node and node packages internally. It allows you to bundle the whole application altogether easily and even provide a hosting service. Meteor has same API across server and client, even access database on the client using the same API. Latency compensation allows client side to react immediately then later patched up if the server rejects the result. Hot Code Pushes allows you to hot update application while users connected to it. Meteor works closely to MongoDB and use the same API as mongoDB on both server and client side. Client side code

can have direct access database. One of its biggest problem of Meteor is: Unlike Derby, it doesn't support server-side rendering, instead it pushes view generating code into client, resulting search engine optimizing (SEO) problem. Thus search engine can't index your content and also no REST applications.

#### Derby

Derby uses Express thus REST is possible. It has server-side rendering so that no search engine issue and works with JavaScript disabled. Derby is now rather unstable and has fewer followings, publicity and contributors.

## 1.3 Summary

Given the fact that JavaScript is the common language of web applications, it's expected that Node will attract more and more developers and become one of the most rapid growing server-side platform. Node.JS not even reaches its 1.0 version yet. There're still promising potentials inside wait to be discovered. We can see the prototype of next generation web applications from novel web framework such as Meteor and Derby. In the trend of growing number of mobile devices, Node, with its nature to support real-time, high concurrency application while remain lightweight and high performance, will definitely affect the way how web can take part in our life.

# Bibliography

Typeset August 20, 2012