Evaluation of Asynchronous Server Technologies

Rocco Schulz, Max Vökler, Joe Boden,

Robert Wawrzyniak, Can Paul Bineytioglu

Corporate State University

Baden-Wuerttemberg - Stuttgart

January 16, 2013

This paper evaluates asynchronous server technologies. Strenghts and weaknesses of asynchronous programming models are elaborated and a proof of concept based on node.js and vert.x is used to evaluate non-functional attributes such as maintainability. ...

Contents

Nomenclature

JS .... JavaScript

List of Tables

List of Figures

1 Introduction

In traditional web application development data is transmitted synchronously, i.e. upon a GET/POST request the result can be displayed only after transmission and processing are finished, as highlighted in figure 1[1]. While maintaining simplicity and predictability this can cause serious latency when uploading large pieces of data most commonly complex forms for registration. Naturally rich content such as images and videos causes even more waiting.

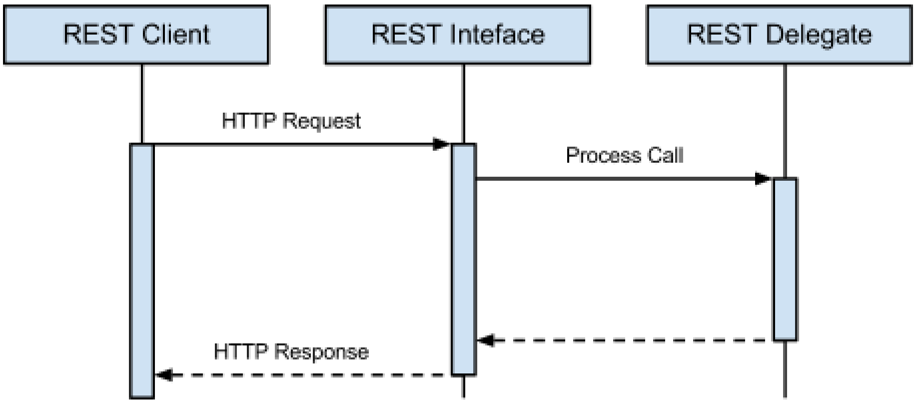

Figure 1: REST request-response sequence diagram

As demands around collaborative access and media richness evolved, this became a serious bottleneck, essentially preventing these types of applications. On the client-, i.e. browser-, side developers were able to work around the issue of synchronous transmission using the XmlHttpRequest object which allows to request resources programmatically (using JS) while deferring handling of the response to a callback (see figure 2[2]) thus enabling much more responsive software.

Although this addressed the issue on the client-side, server-side request were still handled very much in a synchronous fashion. For example the popular Apache web server forks a new process for each incoming request[3]. As popular applications have to cope with unprecedented amounts of concurrent users in conjunctions with massive request

[1] w.a (w.y)
[2] Matejka, J. (2012)
[3] TODO: find source

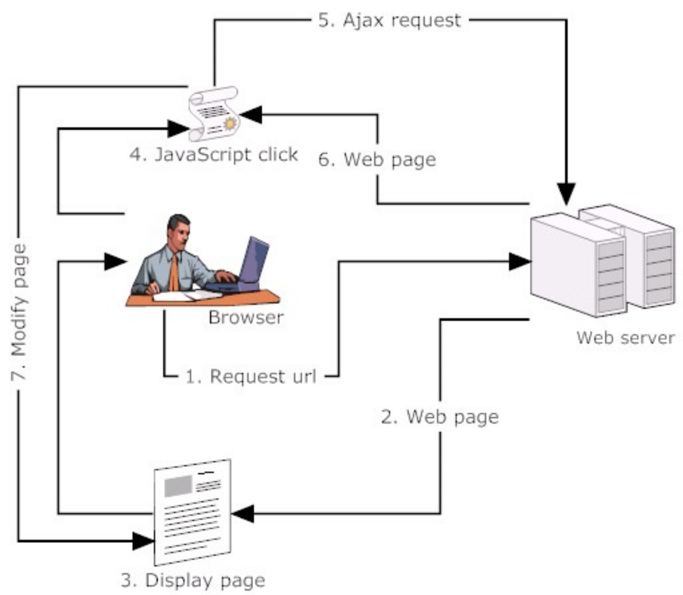

Figure 2: AJAX Diagram

counts, this obviously causes performance issues.

Relatively young frameworks such as Node.js and Vert.x try to address this issue by providing a completely asynchronous programming model which allows associating multiple simultanious connections with a single thread by using an event-driven approach.

This paper elaborates the concepts behind these young frameworks and analyses their technical strengths and weaknesses. Furthermore non-functional attributes will be evaluated based on two sample implementations in Node.js and Vert.x.

2 Setting the context

2.1 Comparison between Asynchronous and Synchronous Processing

A common case in programming is access to I/O. In synchronous processing a running thread needs to wait for the completion of the I/O operation before it can continue. The thread is in an idle state while it is waiting which allows another process or thread to occupy the CPU in the meanwhile.

In multithreaded applications several threads can run simultaniously within one process. Several threads can access shared memory concurrently which can cause inconsistent states. This can be avoided by synchronizing threads - e.g. with locks. This means that programmers need to take into account every possible execution order to effectively avoid program defects such as data races and deadlocks.[4] This can be time consuming and potentially results in error-prone code.

A typical synchronous call is provided in listing 1. The contents of a file are read and displayed afterwards. The programm is blocked until the read operation has finished.

```
reader = new FileReader();
content = reader.readAsText("input.txt");
printContent(content);
```

Listing 1: Pseudocode: Synchronously reading and displaying a file's contents

Asynchronous programming style uses a different concept. The flow of an application is determined by events, which is why this style is also called event-driven programming.[5] In listing 2 the call to the *readAsText* function is done asynchronously. There is no return value - instead an event handler is provided as a second argument. This function is also referred to as a callback function. It is called as soon as the read operation has completed.

```
read_completed = function(content) {
  printContent(content);
}

reader = new FileReader();
reader.readAsText("input.txt", read_completed);
```

[4] Cf. Breshears, C. (2009), p. 10
[5] Cf. Teixeira, P. (2012), p. 16

Listing 2: Pseudocode: Asynchronously reading and displaying a file's contents

This concept is coupled with an event loop, which is a single thread that is running inside the main process. The loop constantly checks for new events. When an event is detected, the loop invokes the corresponding callback function. The callback is processed in the same thread which means that there is at most one callback running at a time. The event loop continues when the callback has completed. As a result the developer does not need to take care of concurrency issues during development. But the developer's task is to write light event handlers that can be processed quickly as every callback is an interruption of the event processing in the event loop.[6] Memory or processor intense callbacks can lead to growing queues of unserved events which eventuelly results in a slow application or service[7].

2.2 Existing Asynchronous Frameworks

2.2.1 Overview

[6]Cf. Hughes-Croucher, T. (2010)
[7]Cf. Teixeira, P. (2012), p. 48
[8]Cf. therve (2012)
[9]Cf. Fettig, A. (2005), p. 12
[10]Klishin, M. (2012)
[11]Cf. O'Dell, J. (2011)
[12]Cf. w.a. (2012a)

Name	**Language/s**	**Description**
Twisted	Python	"Twisted is an event-driven networking engine written in Python and licensed under the open source MIT license"[8]. Twisted is a mature framework with a large number of supported networking protocols. Its development is backed by an active community.[9]
EventMachine	Ruby	"EventMachine is a library for Ruby, C++, and Java programs. It provides event-driven I/O using the Reactor pattern."[10]
Node.js	JavaScript	Node is based on Chrome's JavaScript runtime *V8.* The platform is fully event-driven and offers core functionalities for network programming. Its functionality can be extended with a large number of modules using an integrated package management system. Node.js started development in 2009 and the community is growing since that.[11]
Vert.x	JavaScript, Java, Python, Groovy, Ruby, Coffeescript	A JVM based platform for network applications which is inspired by Node.js. Vert.x comes with its own event bus system that allows distributing applications among multiple network nodes. Support for the languages Scala and Clojure is scheduled for future releases.[12]

Table 1: Existing asynchronous programming frameworks

2.2.2 Node.js

To put it in a nutshell one can say, that Node.js is JavaScript on a server.

Besides, Node.js is a young platform with a lot of buzz around it. Due to the rising of the Web 2.0 and widely accessible internet through smartphones, demanding users expect more complex and more interactive forms of application usage. The challenge even gets harder considering the steep number of devices that are interacting with online services.To overcome those problems Node.js lays it's foundations on an event driven computing architecture for web servers.

Node.js doesn't try to make you perform undoable things. It rather lets events drive the action, so that it is single-threaded and only one thing happens at once. This is why an event loop is a fundamental part of Node.js. It includes the concept of nonblocking I/O activities. A result is that actions that cause the program to wait like database requests and file I/O do not halt execution until they return data. In contrast they process independently and raise an event when the data is accessible. It is therefore necessary to use callbacks for dealing with different kinds of I/O.

An exemplary code for a basic HTTP server in Node.js is shown in listing 3 to deepen the understandin gof the event loop and callback in Node.js.

```
var http = require("http");

http.createServer(function(request, response) {
    response.writeHead(200, {"Content-Type": "text/plain"});
    response.write("Hello World");
    response.end();
}).listen(8888);
```

Listing 3: The simplest way to programm a server in Node.js

The code uses a factory method to create a new HTTP server and attaches the argument of the createServer function as a callback to the request event. The first run of this code is also called setup. When a HTTP request arrives the anonymous callback function is processed and "Hello World" appears on the browser.

That the basic code above isn't the most sophisticated way to write Node.js code explains the following thought experiment: Assuming that the Hello World page would be popular

and had a lot of requests from different devices to handle and in addition the callback processing would take one second, it is obvious that the second request would already have to wait for one second until it gets served. This is far away from the near-real-time requirement Node.js is confronted with.
Two programming rules in Node.js can be inferred from the basic server and it's event loop blocking problem, which is described in section 2. First, once the setup is in place all actions should be programmed event-driven. Second, if a workload requires Node.js to process something for a long time, it should be outsourced to web workers.[13]

2.2.3 Vert.x

Vertx is a polyglot that runs on the JVM (Java Virtual Machine). It is hence possible to scale over available cores without manually forking multiple servers.
The application API is exposed in multiple programming lanugages (see table 1).

In Vert.x the smallest available deployment unit is a verticle, which runs inside an instance. Each instance runs singlethreaded. Multiple verticles can be run inside one instance as depicted in figure 3.

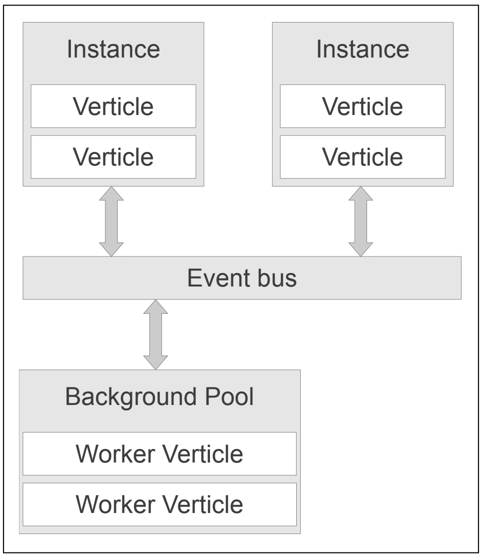

Figure 3: Abstracted deployment units of Vert.x

[13] Cf. Hughes-Croucher, T. (2012)

When multiple instances are run on one machine, Vert.x automatically distributes incoming requests among all running instances in a round-robin fashion, so that each vert.x verticle instance remains single threaded.

Vert.x also includes a distributed event bus, which enables verticles to communicate with each other, either within the same instance or across different instances. The event bus allows direct communication with in-browser JavaScript as well.
Vert.x allows to run IO heavy tasks in separate worker threads that reside in a so-called background pool as these tasks would otherwise block the event loop as described in section 2.1.

The core functionality of Vert.x covers basic networking tasks and protocols, web servers, clients, access to the file system, shared maps and the event bus. The core libraries of Vert.x can be embedded in any JVM program for reuse in larger projects.
As opposed to Node.js, the core functionality and API can be considered quite static as changes need to be done in all supported languages.[14]
The core can be exctended with additional features that are provided by optional modules that can be obtained over a public git-based module repository.[15]. The repository currently contains 16 distinct modules in different versions.[16]

An extensive online documentation is available for all supported languages. Additionally, code examples for most features are available for all supported languages in a public repository[17].
Vert.x is open source and licensed under the Apache Software License 2.0[18], so that commecial redistribution in closed source projects should not be an issue.

[14] Cf. w.a. (2012a)
[15] Cf. w.a. (2012b)
[16] Cf. Fox, T. et al. (2012)
[17] Cf. Fox, T. (2013)
[18] See `http://www.apache.org/licenses/LICENSE-2.0.html`

3 Areas of Application

3.1 Use Cases

The non-blocking nature of asynchronous calls is important in all types of applications that need to handle a large number of requests in real time.
Some success stories are available on `http://nodejs.org/`.

Could be used for:

- networked applications that tend to keep many inactive connections
- web trackers
- web servers
- lightweight json APIs (non-blocking I/O model combined with JavaScript make it a great choice for wrapping other data sources such as databases or web services and exposing them via a JSON interface)
- proxies
- email and messaging systems
- authorization processors
- Streaming data (e.g. file uploads in real time)

3.2 Don't use cases and anticipated issues

As with all technologies one has to carefully decide whether or not it suits the requirements of a project.

- Systems that do a lot of computing / IO access
- Datawarehousing with analytics / analytical tasks on large filesets

- CRUD / HTML applications*: At present node and vertx do not provide additional benefits to acalability for these types of web applications. Powerfull frameworks like Ruby On Rails, Grails and Django are currently better suited. Providing an app that is suited for millions of requests does not automatically increase the number of requests.
- Desktop applications*: Some frameworks are suited for this, but not node.js. JavaScript is missing operators for good OS integration.

Anticipated issues

- running worker threads does introduce the concurrency issues that were avoided with the event loop again as they could concurrently access IO
- ...

4 Exemplary Implementations

A simple web form application has been implemented in Node.js and Vert.x to further analyze non-functional requirements and collect practical experience with these frameworks.

4.1 Software Description

Brief description of the insurance fee calculator

4.2 Software Design

High level design

Interface description, and differences between Node.js and Vert.x

4.3 Software Implementation

Complications or any other notes on the implementation process that might be of importance for the evaluation.

5 Evaluation of Non-functional Attributes

5.1 Maintainability

Language: JavaScript is wide spread, same for Java. However JavaScript offers better flexibility and native constructs for these types of applications. (Java 8 might improve things a bit). Node's API is undergoing backwards incompatible changes from time to time. It is desired to updated some components of a Node based application from time to time.

5.2 Integration

Vert.x and Node.js are supposed to be used as fully event driven standalone applications that can be extended with event-driven modules. However it might be desired to reuse existing software that is not fully event driven. Dont make use of blocking apis. Consider connecting that software with a message bus. Separate IO intense tasks into "web workers" or similar constructs.

5.3 Scalability

Support for single machine scaling using multiple threads exists. Deployment on distributed systems differs in node.js and vert.x. Vert.x. uses its own communication bus to share information between verticles.

6 Conclusion

7 Lists of References

List of Literature

Breshears, C. (2009)	The Art of Concurrency: A Thread Monkey's Guide to Writing Parallel Applications, O'Reilly Media.
Fettig, A. (2005)	Twisted Network Programming Essentials, O'Reilly Media.
Hughes-Croucher, T. (2012)	Node Up and Running, O'Reilly Media.
Teixeira, P. (2012)	Professional Node.js : Building Javascript Based Scalable Software, Wrox.

List of Internet and Intranet Resources

Fox, T. (2013)	Vert.x code examples, https://github.com/vert-x/vert.x/tree/master/vertx-examples/src/main, retrieval: 01/10/2013
Fox, T. et al. (2012)	Vert.x Module Repository on Github, https://github.com/vert-x/vertx-mods, retrieval: 01/11/2013
Hughes-Croucher, T. (2010)	Understanding event loops, http://developer.yahoo.com/blogs/ydn/posts/2010/10/understanding-the-event-loops-and-writing-great-code-for-node-js-part-1/, retrieval: 01/01/2013
Klishin, M. (2012)	EventMachine Readme, https://github.com/eventmachine/eventmachine/blob/b79d508532e210084a7fcda31f330a8a81bf78aa/README.md, retrieval: 01/12/2013

Matejka, J. (2012)	w.t., http://jirimatejka.cz/ajax-diagram, retrieval: 01/12/2013
O'Dell, J. (2011)	Why Everyone Is Talking About Node, http://mashable.com/2011/03/10/node-js/, retrieval: 01/09/2013
therve (2012)	Twisted Project Page, http://twistedmatrix.com/trac/, retrieval: 01/01/2013
w.a. (2012a)	Vert.x Main Manual, http://vertx.io/manual.html, retrieval: 01/10/2013
w.a. (2012b)	vert.x Module Manual, http://vertx.io/mods_manual.html, retrieval: 01/10/2013
w.a (w.y)	Figure 1.1 - REST request-response sequence diagram, http://windyroad.org/static/rest4BW-User-Guide.html, retrieval: 01/12/2013