UDC 681.5

Y. Timoshin, B. Barna

NATIVE PROMISES VS BLUEBIRD PROMISES IN NODE.JS

Abstract: In this article the main point is in comparing of two ways to work with an asynchronous code in Node.js. Because Node.js is native asynchronous this problem is very important. In high load applications which are built on Node.js server every second is important. So, very long time there are many discussions between different programmers: what we should to use? Going through every version of Node.js we will compare and calculate which way has more advantages in speed. All results will be displayed in understandable graphics and tables.

Key word: Node.js, Bluebird, native, Promise, asynchronous, callback

Introduction

The Promise object represents the eventual completion of failure of an asynchronous operation, and it resulting value [1] (Fig. 1). It came from well-known callback function. The main problem with callback was creating new scope, so we couldn't get the result from asynchronous operation in the same scope. And it was much harder to get result, when there were a lot of asynchronous calls (Fig. 2).

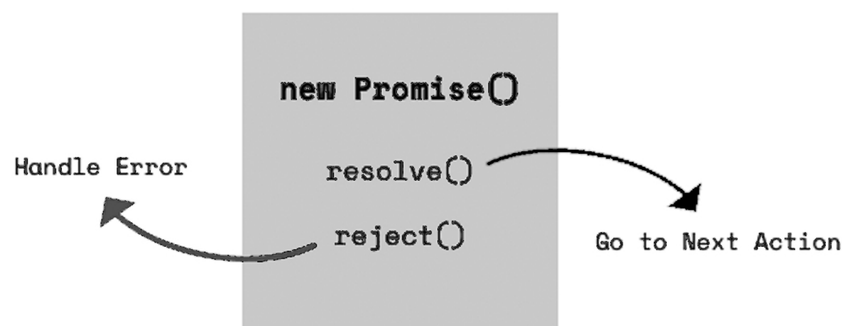

Figure 1. Promise model

Promises first became mainstream in Javascript when jQuery introduced Deferred Objects in jQuery 1.5 in 2011. Alternative promise libraries started gaining ground soon afterwards after You're Missing the Point of Promises post which was critical of jQuery's implementation of Promises via *Deferred Objects*, specifically in regards to exception handling [2].

© Y. Timoshin, B. Barna

```
a(function (resultsFromA) {
  b(resultsFromA, function (resultsFromB) {
    c(resultsFromB, function (resultsFromC) {
      d(resultsFromC, function (resultsFromD) {
        e(resultsFromD, function (resultsFromE) {
          f(resultsFromE, function (resultsFromF) {
            console.log(resultsFromF);
          })
        })
      })
    })
  })
});
```

Figure 2. Callback hell

The main principle of Promise work is in being in some state and call function depends on this states. A Promise can be in of these states:

- *pending* - initial state, neither fulfilled nor rejected.
- *fulfilled*: meaning that the operation completed successfully.
- *rejected*: meaning that the operation failed.

A pending promise can either be *fulfilled* with a value, or *rejected* with a reason (error). When either of these options happens, the associated handlers queued up by a promise's then method is called. If the promise has already been fulfilled or rejected when a corresponding handler is attached, the handler will be called, so there is no race condition between an asynchronous operation completing and its handlers being attached (Fig. 3).

In other definitions there is another promise meaning. A Promise is a proxy for a value not necessarily known when the promise is created. It allows you to associate handlers to an asynchronous action's eventual success value or failure reason. This lets asynchronous methods return values like synchronous methods: instead of the final value, the asynchronous method returns a *promise* for the value at some point in the future [3]. From this, Promise allows customer to return value in the same place, where it was called and this is main advantage of promise. In conclusion, Promise is a design pattern to remove callback usage.

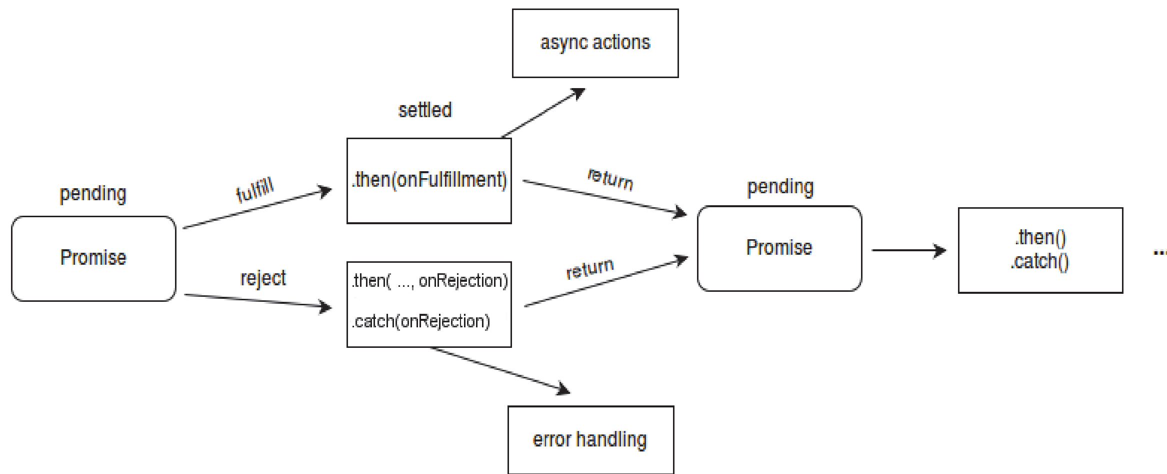

Figure 3. Promise in work

Problem Statement

Now, there are two ways to use Promises: native and Bluebird. Native Promise is a promise, which Node.js support from the box. So, logically, this the best choice - to use it. But, it is not almost like this. Many of Node.js software engineer prefers Bluebird. Bluebird is a npm package, which allows us to use fully featured promise library with focus on features and performance. Bluebird has a big variety of asynchronous methods, such as cancellation, iteration, warnings methods and other useful methods. But it is easy to determine your goal for using native or Bluebird: if you want speed than use native, if you want features – use Bluebird. But it is not so easy.

Bluebird has many features that native don't – so we will compare only base promise features. We will calculate performance of both ways on different version of Node.js.

An Overview of Existing Solutions

Some of authors make comparison between them only in some specific cases. For example, features in Bluebird, that are absent in native or some cases with better error catching and handling. They look on this problem very simple. They show only quality different, but not quantity. This is the case, where the second one is more important. When engineer choose a pattern, which he will be involve in his project, his code, he should know what he wants from working with promises.

Some authors calculate performance only on specific Node.js version, so we can't understand if there was any improvement or performance rise. So we need to

 ISSN 1560-8956

find different authors, where they describe different Node.js version Promise flows, but they not clear, because they used various asynchronous operation and the experiment is not clear.

So, there is a possibility to include all Node.js versions with equal asynchronous operation in one experiment.

Problem Solution

Some of applications use now 6 Node.js version. In order to this we will start calculate metrics from 6 version up to latest 12.

For experiment we will use native Node.js. We will test 4 versions: v6.17.1(6), v8.16.2(8), v10.17.0(10), v.12.13.0(12). All these versions are LTS supported. We will also use Bluebird library v.3.7.2.

For time metric we will use native Node.js Date object.

To unify asynchronous operation, we will wait response from https://www.google.com. All data we will be entered in Excel and will create absolute graphics.

Results

All data will be represented in tables and based on it graphics for each Node.js version. Number in cell represent time in ms (milliseconds).

All data is represented as an average values of 10 iterations. All data, which has a big delay in series wasn't included. So, all data represent averaged values of asynchronous HTTP calls.

Table 1. **Metrics for Node.js v6**

Native	Bluebird
237	259
237	265
237	266
238	265
237	270
238	265
239	265
240	265
244	266
244	266

From table 1 we can see that values are pretty similar, average for native calls is 239.1 and for bluebird calls is 265.2.

Table 2. **Metrics for Node.js v8**

Native	**Bluebird**
173	235
208	242
212	241
213	241
215	246
217	241
217	241
222	242
219	241
219	256

From table 2 we can see that values are pretty similar, average for native calls is 211.5 and for bluebird calls is 242.6.

Table 3. **Metrics for Node.js v10**

Native	**Bluebird**
253	222
258	226
260	225
259	225
260	227
259	227
265	232
260	227
292	256
296	260

From table 3 we can see that values are pretty similar, average for native calls is 266.2 and for bluebird calls is 232.7.

Table 4. **Metrics for Node.js v12**

Native	**Bluebird**
257	213
258	214
260	234
262	239

 ISSN 1560-8956

Endid tab. 4

Native	Bluebird
263	240
263	241
263	242
264	243
264	253
264	297

From table 4 we can see that values are pretty similar, average for native calls is 261.8 and for bluebird calls is 400.7.

Based on this data sets we will create graphics.

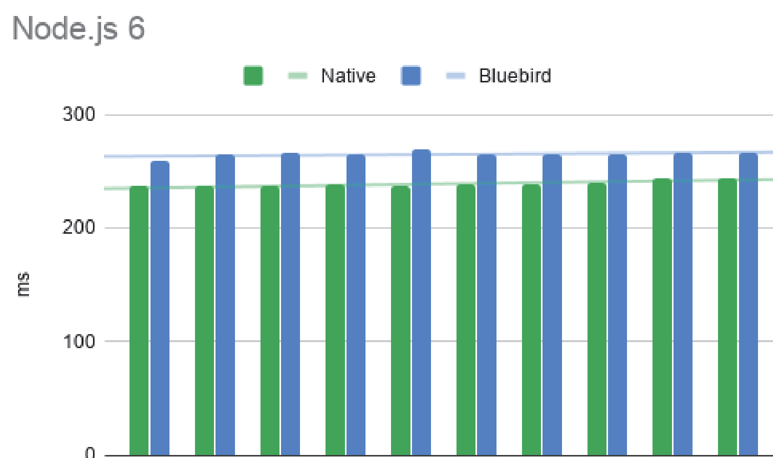

Figure 4. Graphic for Node.js v6

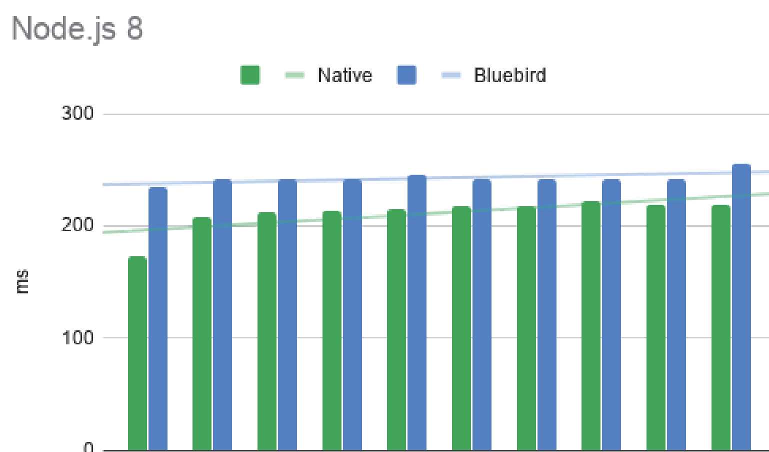

Figure 5. Graphic for Node.js v8

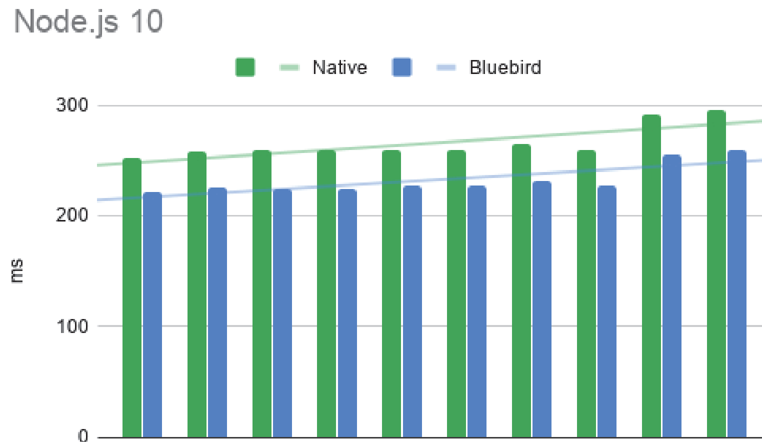

Figure 6. Graphic for Node.js v10

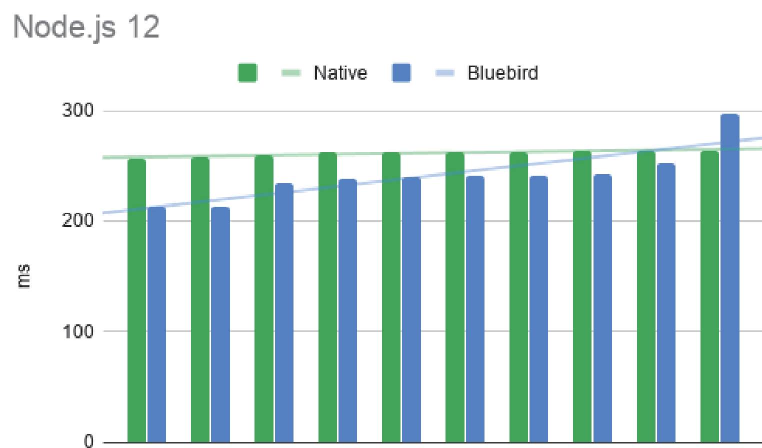

Figure 7. Graphic for Node.js v12

So, there are some interesting facts from this data sets:

1. Native Promise is faster than Bluebird only on v6, v8 and v12.
2. Bluebird is faster than only on v10.
3. On v6 Bluebird is slower than native in about 10%.
4. On v8 Bluebird is slower than native in about 13%.
5. On v10 Bluebird is faster than native in about 10-13%.
6. On v12 Bluebird is faster than native in about 7%.

Summary

The work considers a model of the environment in which applications under Node.zhs for its various versions work. It is shown that for applications using Node.js v.6 and v.8, it is better to use the native Promysh, and for applications using Node.js v10 and

 ISSN 1560-8956

v12 it is better to use Bluebird. It is also important to say that the simulation results on v12 were pretty similar: Bluebird has a better average, and native has a smaller range.

But you must keep in mind the unique features of Bluebird and their faster updates. If features and performance are important, use Bluebird on v12, v6, and v.8, but not on om v10. And if you need to use only simple asynchronous operations, use the native Promise for any version of Node.js.

REFERENCES

1. Promise MDN [Електронний ресурс] - https://developer.mozilla.org/ru/docs/Web/Java Script/Reference/Global_Objects/Promise

2. Promises vs Bluebird [Електронний ресурс] - https://pub.clevertech.biz/native-javascript-promises-vs-bluebird-9e58611be22

3. Asynchronous code with promises [Електронний ресурс] https://medium.com/dev-bits/writing-neat-asynchronous-node-js-code-with-promises-32ed3a4fd098