

# Tracing Nodejs Applications in a Low Level Context

**Progress Report Meeting** 

Hervé KABAMBA

January 14, 2022

Polytechnique Montréal Département de Génie Informatique et Génie Logicielle

# Agenda

- Introduction
- Use cases
- Conclusion
- Ongoing work
- Bibliography



#### Context

- Single-threaded environment
- Uses asynchronous requests
- Uses an Event Loop
- Uses a Thread Pool

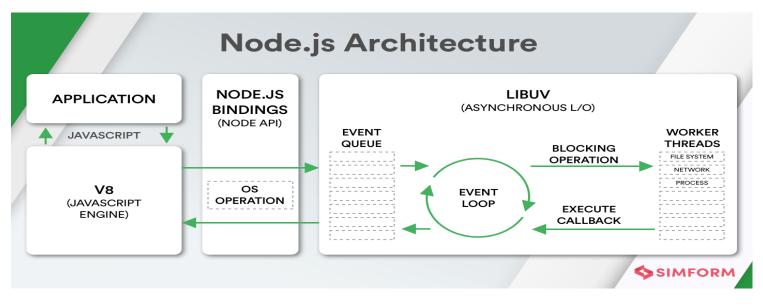
#### The asynchronous nature is managed by the Libuv library

#### Objectives

- Track the high-level asynchronous requests
- Track the underlying low-level operations
- Reconstruct the request life-cycle
- Obtain a vertical span representation of the asynchronous request

#### Problem

- NodeJs architecture is very complex
- It is built-in with many independent components
- An asynchronous operation vertically spans many layers



Nodejs Architecture [1]

#### Problem 2

- High level latency do not give information on the root cause
- A high latency may results from a blocked event loop

- High latency may result from the threads being busy executing other tasks

This results in the propagation of the problem to the other pending requests, increasing their execution latency

Problem 3

At the origin, the request is high-level

Vertically tracking it in the underlying sub-layers raises 2 problems:

- Synchronization of trace
- Context management

#### Proposed solution

- The synchronization is addressed by implementing a Native Module:

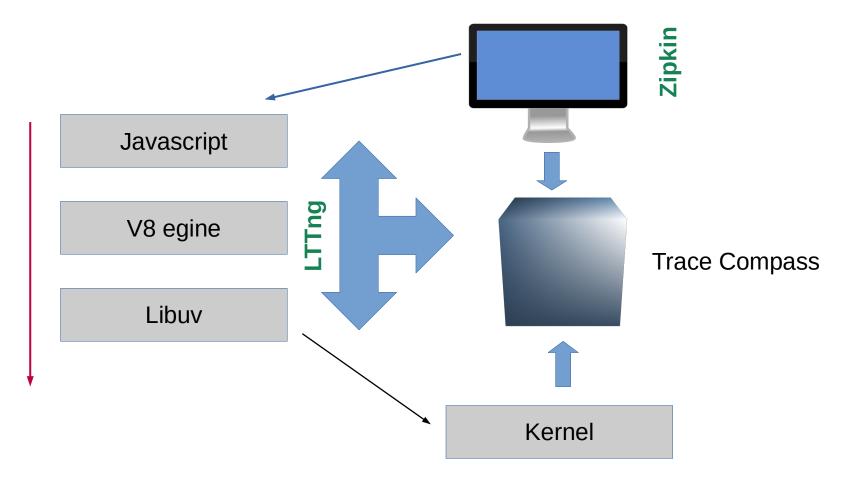
-Typescript and JavaScript applications are instrumented with the latter

- The module is called from Nodejs application, and LTTng functions are invoked from Nodejs internals
- Resulting in a synchronized LTTng trace

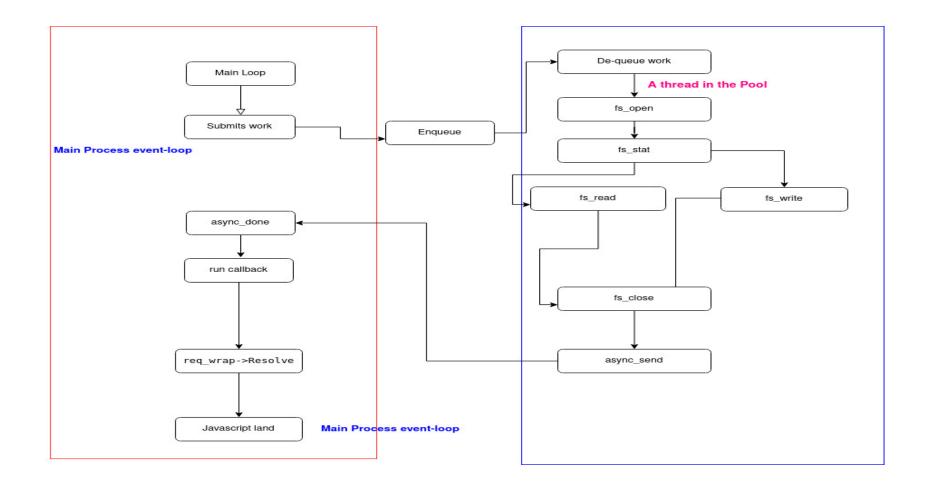
- The trace context is managed by a new proposed algorithm called the **Bounded Context Tracking Algorithm (BCTA):** 

A four-layer event context tracking algorithm that vertically reconstructs a request sequence (Javascript land, V8, Libuv and Kernel)

#### System Architecture



#### I/O Abstraction model



## **Use Cases**

#### Use case 1: Tracing Asynchronous functions

- Load the aync\_hooks module from nodejs
- Load the calculate module to instrument your code

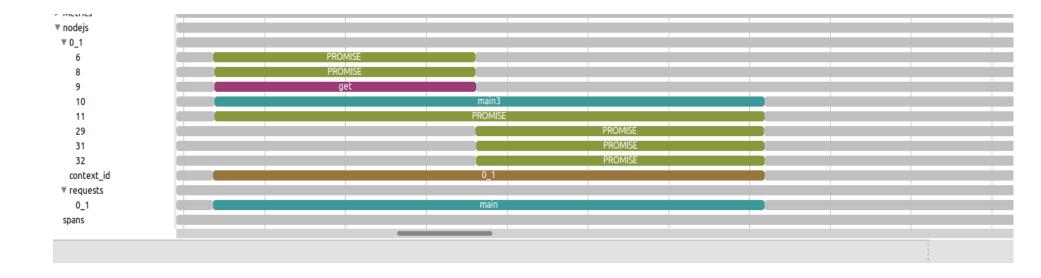
const async\_hooks = require('async\_hooks');

const calculate = require("./build/Release/calculate");

Everything is setup !!!

52	}			> async	Aa <u>ab</u> <b>_*</b>
53		orrlland	<pre>ller = function(err) {</pre>		
54 55	var I		e.log(err);		
55 56	1	CONSOLE			
57					
58	fun	ction ma	in() {		
59			is=async hooks.executionAsyncId();		
60			<pre>async hooks.triggerAsyncId();</pre>		
61					
62		calcula	<pre>ite.send_event(as, ass, 'js_open_main');</pre>		
63			<pre>rProfileURL = "https://api.github.com/users/hervekb";</pre>		
64			aPromise = getData(userProfileURL);		
65		// Get	user details after that get followers from URL		
66		dataPro	<pre>mise.then(JSON.parse, errHandler)</pre>		
67			.then(function(result) {		
68			userDetails = result;		
69	• • •		········//·Do·one·more·async·operation·here		
70			calculate.send_event(async_hooks.executionAsyncId(), async_hooks.triggerAsyncId(), 'j	<pre>sx_get');</pre>	
71			<pre>var anotherPromise = getData(userDetails.followers_url).then(JSON.parse);</pre>		
72			return anotherPromise;		
73			<pre>}, errHandler)</pre>		
74			.then(function(data) {		
75			//console.log(data)		
76 77			<pre>const eid=async_hooks.executionAsyncId(); calculate.send event(eid, async hooks.triggerAsyncId(), 'jsx main3');</pre>		
77			catcutate.send_event(erd, async_nooks.triggerAsyncid(), 'jsx_mains');		
78 79			}, errHandler)		
80			j, criticald corp		
81			.then(function(data) {		
82					
83			calculate.send event(as, ass, 'js exit main');		
84			<pre>}, errHandler);</pre>		
85					

#### Function asynchronous execution



- Root cause analysis
  - -vscode.workspace.fs.readfile and node fs.readFile
  - Develop a vscode plugin that reads a file and inject it into **Theia**
  - Compare the two reading latencies:

vscode.workspace.fs.readfile: arround **300 ms** Nodejs: arround 1,5 **ms** 

#### Nodejs reading operation

					_
3				Quark	183
1				Full attribute path	THEIA/spans/0.12446605983032444_0.4145611962961384/websocket
				Date	2021-10-24
				Start Time	18:01:24.452916041
1					
1				Stop Time	18:01:24.454381006
				Duration	0.001464965s (1,5 ms)
	syscall_entry_epoll_ctl	syscall_entry_epoll_wait			
			vscode_readFile		
5					
- 1			1		
2					
5					
- (E				1	
_					

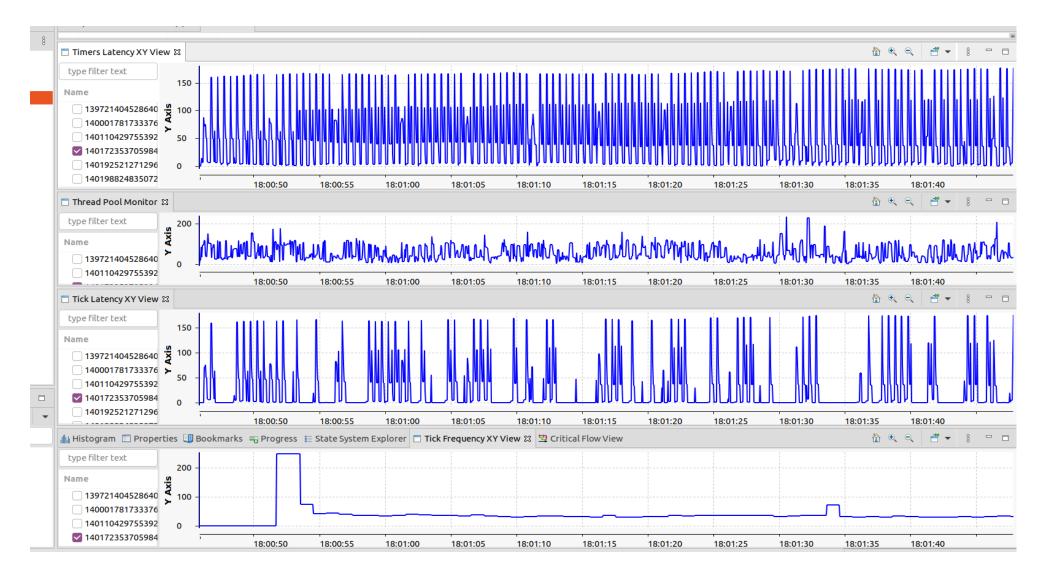
#### Vscode reading operation

▼ 0.09010465008884738	fc open	fs_lstat		
Libuv	fs_open			
syscall			y_epoll_wait	
websocket		vscode_r	eadFile_vs	
0.12446605983032444	Attribute	vebsocket		
0.17128943937755725				
▶ 0.33353126829560553	Value	scode_readFile_vs		
0.46646577422758595	Quark	04		
▶ 0.4771183556211698_	Full attribute path	HEIA/spans/0.09010465008884738_0.20388749392999306/	websocket	
▶ 0.6042980162926672	Date	021-10-24		
▶ 0.8457884124935646				
▶ 10_21	Start Time	8:01:37.547839058		
▶ 11_21	Stop Time	8:01:37.879768920		
▶ 12_12	Duration	.331929862s (331,9 ms)		
► 162_3				
► 162_3				
01:37.551629433				

#### Problem pinpointing

▼ spans				
v 0.090104650088				
Libuv	fs_close			
syscall	sys	call_entry_epoll_	wait	
websocket 🛛		vscode_readFile_1	Athelbuche	
▼ 0.124466059830				syscall
Libuv			Value	syscall_entry_epoll_wait
syscall			Quark	205
websocket			Full attribute path	THEIA/spans/0.09010465008884738_0.20388749392999306/syscall
▶ 0.171289439377				2021-10-24
▶ 0.333531268295			i	18:01:37.548246824
▶ 0.466465774227				
▶ 0.477118355621			- ·	18:01:37.829978727
▶ 0.604298016292			Duration	0.281731903s (281,7 ms)
▶ 0.845788412493				
▶ 10_21				
w 11 01			1	
4 10:01:27 551620422				
4 18:01:37.551629433				

#### Monitoring metrics



#### **POLYTECHNIQUE** MONTREAL – Hervé Kabamba

# **Ongoing work**

- Completing the critical path analysis model
- Developing views for the critical path analysis
- Defining more monitoring metrics

### Thank you

# **Bibliography**

[1] I. Beschastnikh, P. Wang, Y. Brun, M. D. Ernst, Debugging distributed systems, ACM-Queue (2015).

[2] J. Hoglund, An analysis of a distributed tracing systems effect on performance. jaeger and opentracing api, UMEA University (2020).

[3] S. Tilkov, S. Vinoski, Node.js: Using javascript to build high-performance network programs, IEEE INTERNET COMPUTING (2010).

- [4] Cloud desktop ide platform.
- URL https://kubernetes.io/fr/
- [5] Visual studio code.
- URL https://code.visualstudio.com/

[6] Y. Geng, S. Liu, Z. Yin, A. Naik, B. Prabhakar, M. Rosenblum, A. Vahdat, Exploiting a natural network effect for scalable, fine-grained clock synchronization, 2018.

2007, pp. 171-180.