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

Introduction

- **Context**

  – Single-threaded environment

  – Uses asynchronous requests

  – Uses an Event Loop

  - Uses a Thread Pool

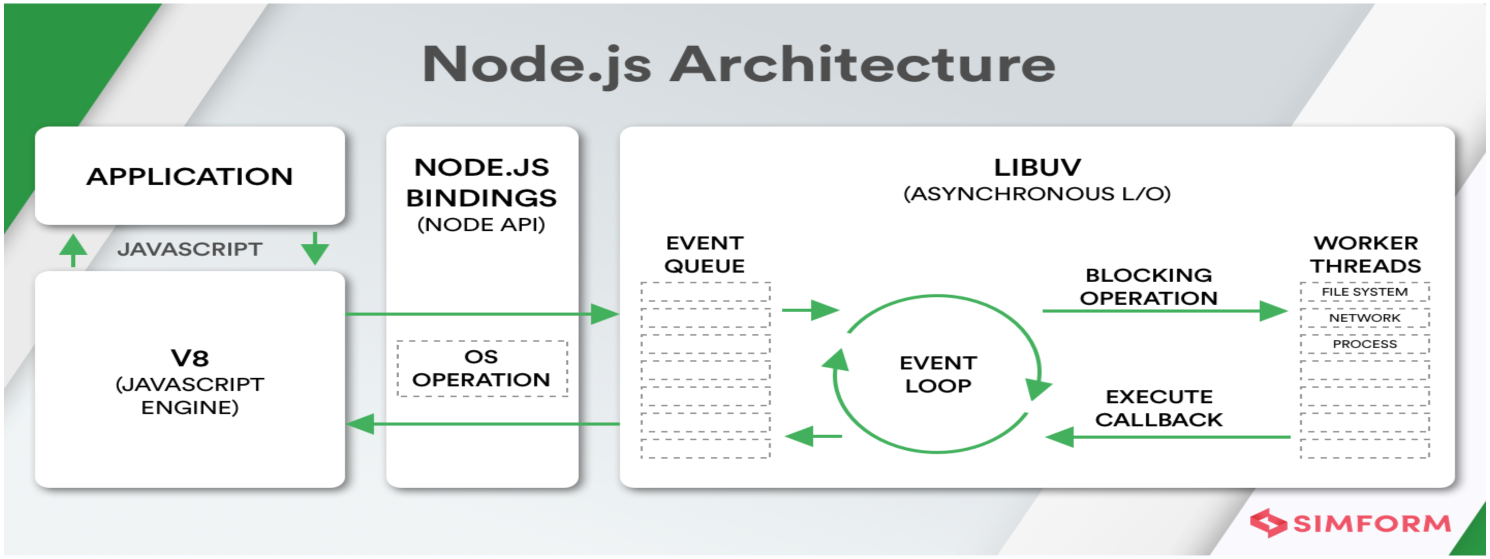
- **The asynchronous nature is managed by the Libuv library**

Introduction

- **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

Introduction

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



Nodejs Architecture [1]

Introduction

- **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

Introduction

- **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

Introduction

- **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

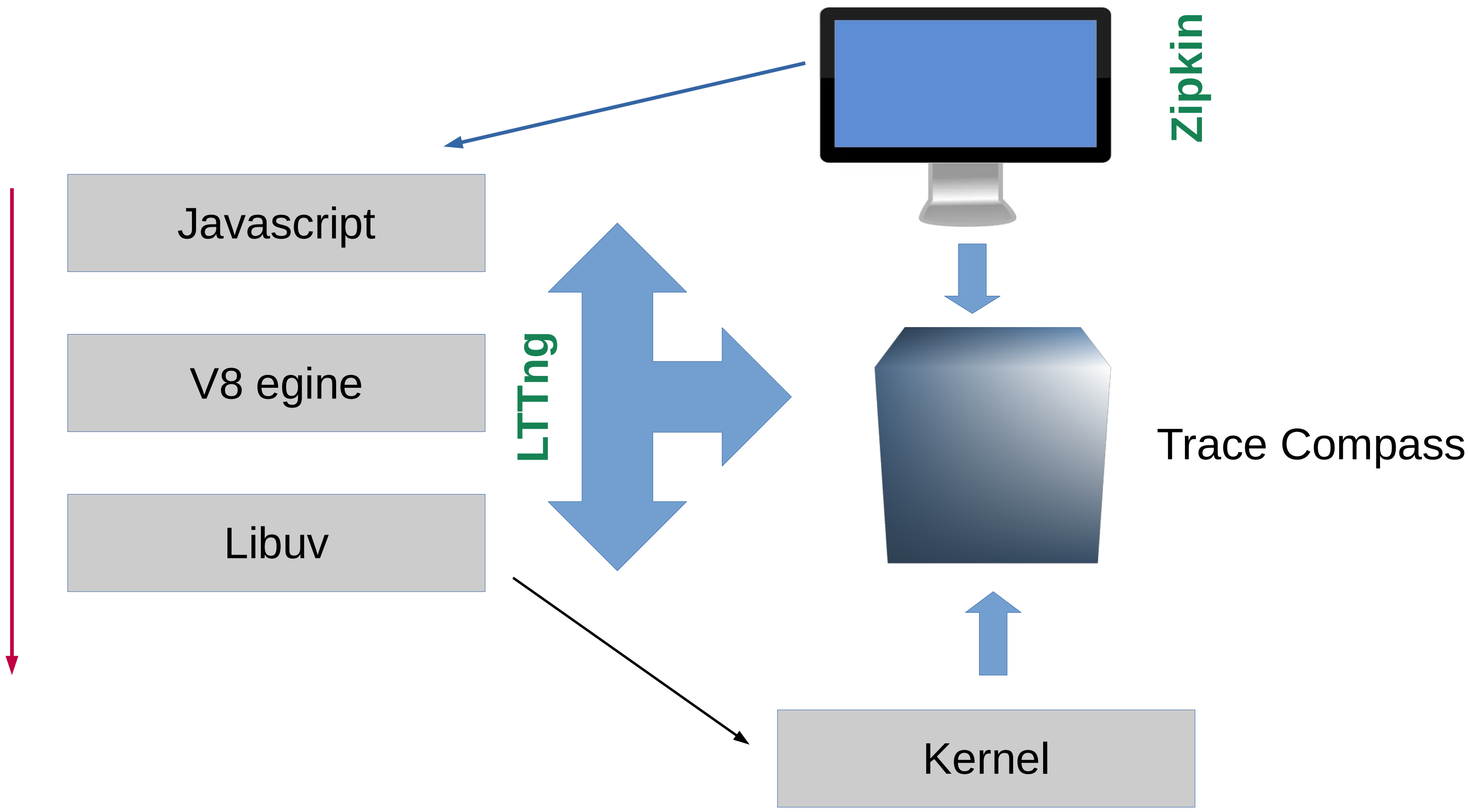
- Resulting in a synchronized LTTng trace

- 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)

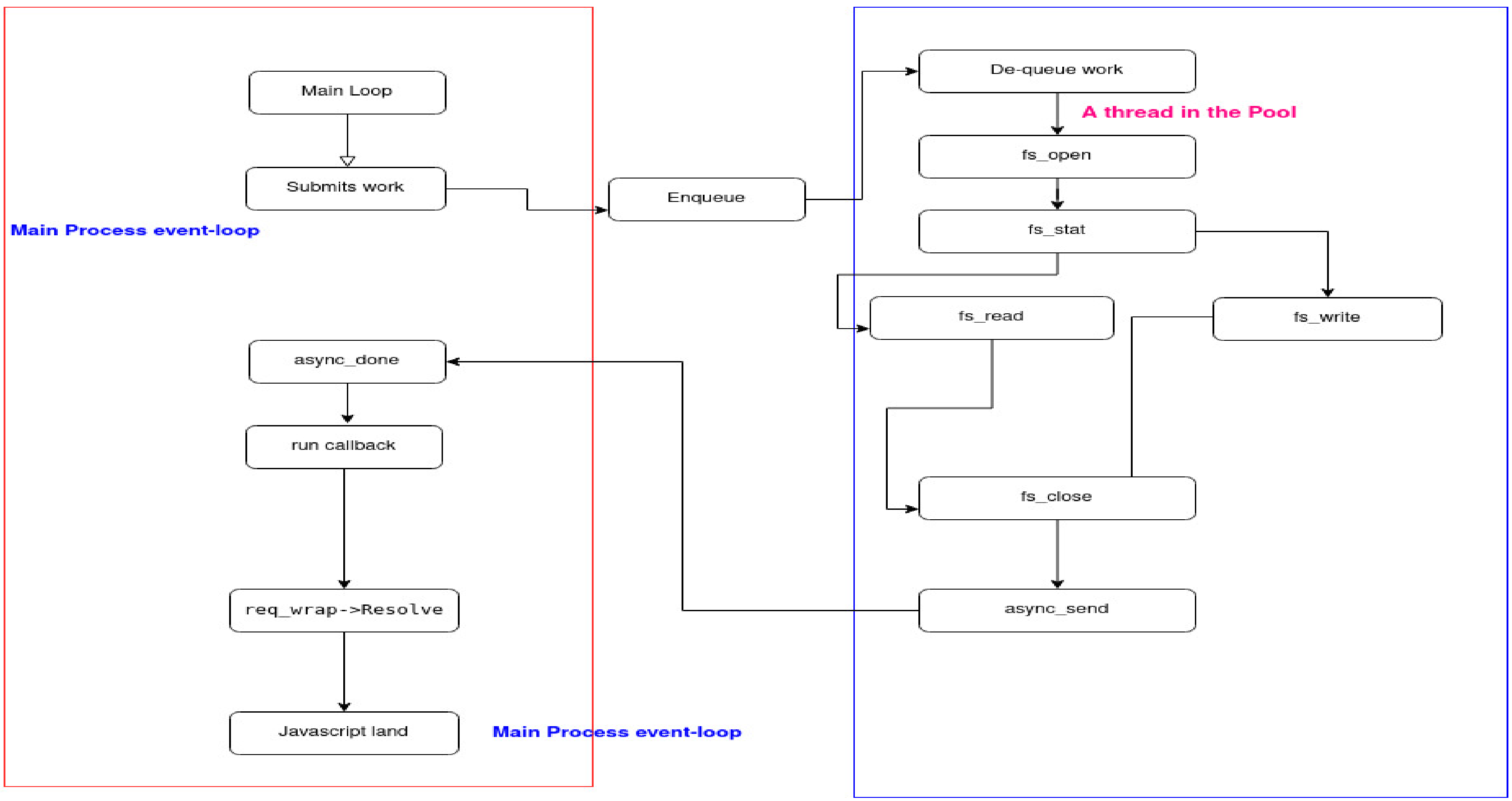
Introduction

- **System Architecture**



Introduction

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 !!!

Use case 1

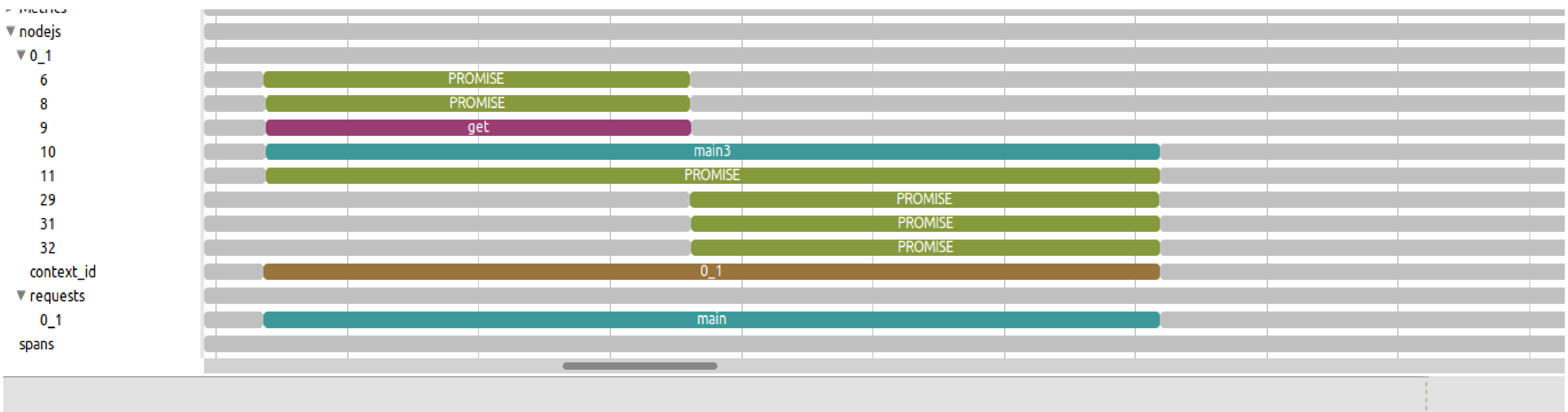
home > herve > promise > test.js > main > then() callback

```
}

var errHandler = function(err) {
    console.log(err);
}

function main() {
    const as=async_hooks.executionAsyncId();
    const ass= async_hooks.triggerAsyncId();

    calculate.send_event(as, ass, 'js_open_main');
    var userProfileURL = "https://api.github.com/users/hervekb";
    var dataPromise = getData(userProfileURL);
    // Get user details after that get followers from URL
    dataPromise.then(JSON.parse, errHandler)
                .then(function(result) {
                    userDetails = result;
                    // Do one more async operation here
                    calculate.send_event(async_hooks.executionAsyncId(), async_hooks.triggerAsyncId(), 'jsx_get');
                    var anotherPromise = getData(userDetails.followers_url).then(JSON.parse);
                    return anotherPromise;
                }, errHandler)
                .then(function(data) {
                    //console.log(data)
                    const eid=async_hooks.executionAsyncId();
                    calculate.send_event(eid, async_hooks.triggerAsyncId(), 'jsx_main3');

                }, errHandler)

                .then(function(data) {

                    calculate.send_event(as, ass, 'js_exit_main');
                }, errHandler);

```

PROBLEMS 3 OUTPUT TERMINAL DEBUG CONSOLE

Use case 1

- **Function asynchronous execution**



Use case 2

- **Root cause analysis**

–***vscode.workspace.fs.readfile*** and node ***fs.readFile***

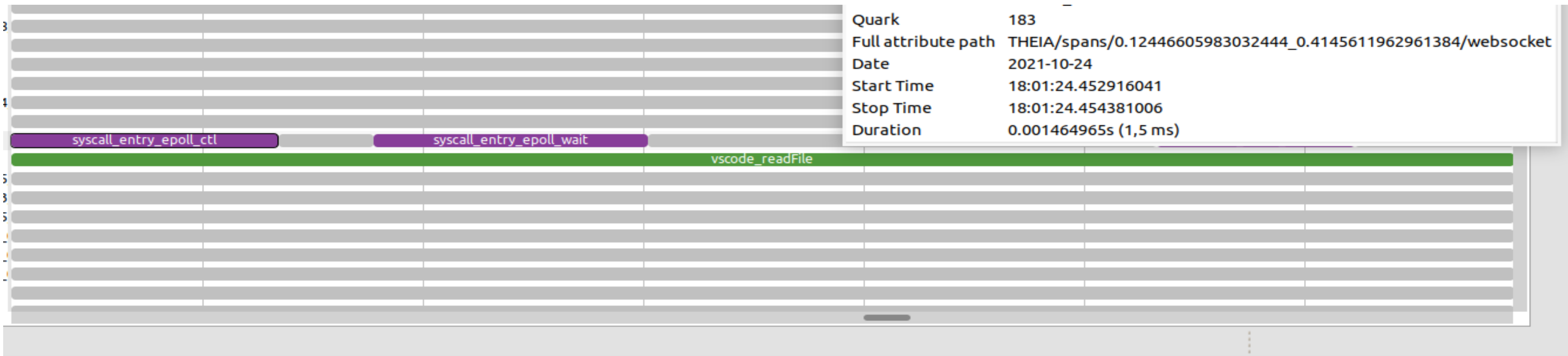
- Develop a vscode plugin that reads a file and inject it into **Theia**

- Compare the two reading latencies:

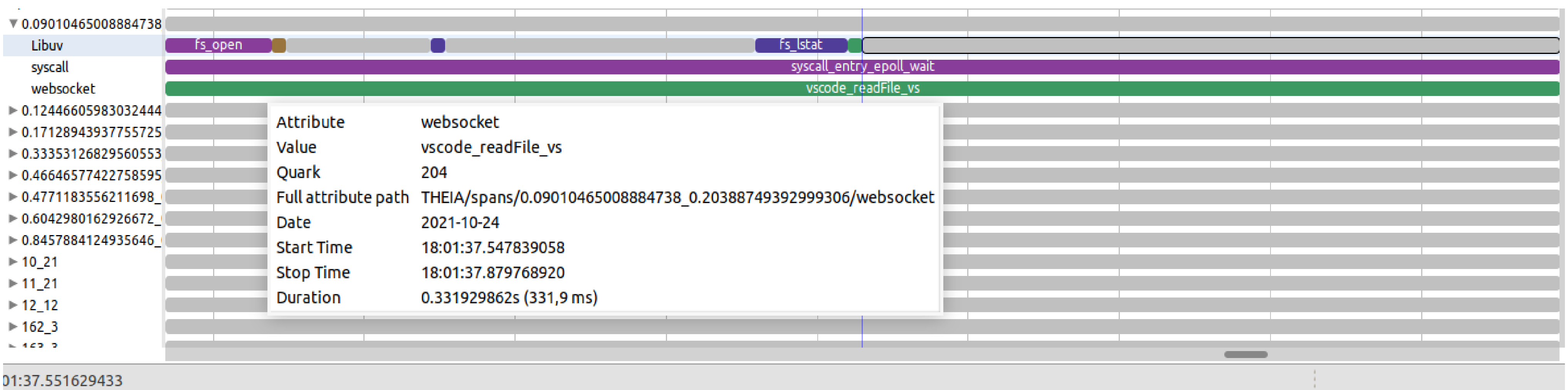
vscode.workspace.fs.readfile: arround **300 ms**
Nodejs: arround 1,5 **ms**

Use case 2

Nodejs reading operation

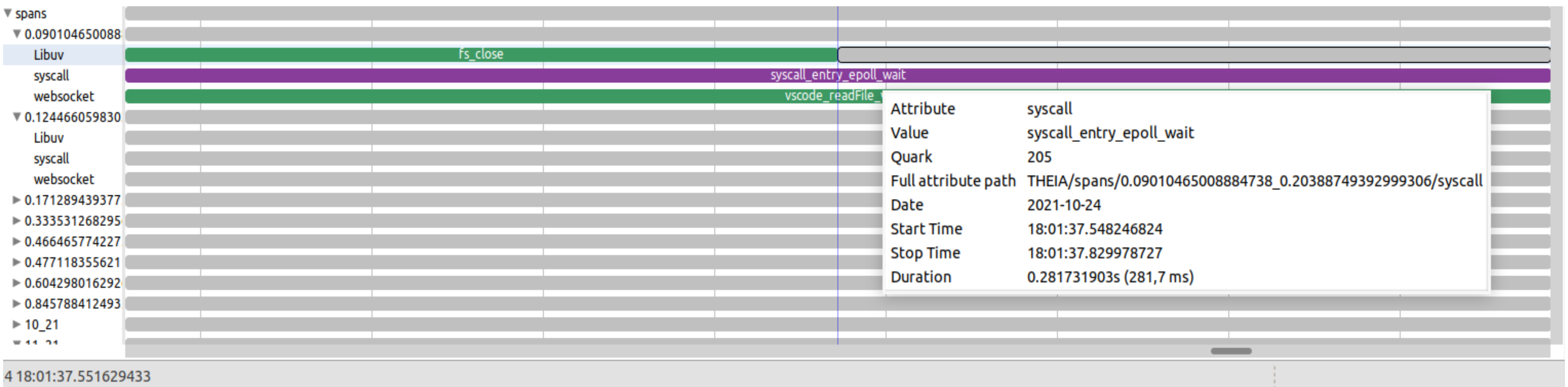


Vscode reading operation



Use case 2

- **Problem pinpointing**



Use case 3

- **Monitoring metrics**

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.