

Tracing Micro-services and Modular IDEs: Performance evaluation in asynchronous requests context

Hervé KABAMBA Michel Dagenais June 11, 2021

Polytechnique Montréal Laboratoire **DORSAL**

Agenda

- Introduction
- Objective
- Methodology
- Current Results
- Ongoing work



THEIA: The framework is developed in Typescript

- Frontend runs on the browser
- Backend runs on Node.js

An interaction of the frontend with the backend:

- Is mainly a communication through a websocket connection channel that carries json encoded messages containing the data.
- Vscode libraries are used by the backend to listen to the socket to retrieve the data
- The data is mainly the service that must be invoked remotely, sometimes with arguments

Backend:

- Operations are executed by invocation on the backend, and the returned results are sent back to the frontend
- The backend is therefore mainly responsible for low level operations with the OS.
- Node.js is single threaded and uses an event loop to handle asynchronous operations.

Intuitively:

- Evaluating the performance of applications running on Node.js brings complexity
- High level tracing of distributed operations can only expose their latency in a global point of view.

Definition of the problem

- Most operations and interactions from the frontend and the backend in Theia are executed by Node.js in asynchronous ways.
- Asynchronous callbacks such as Promises, SetTimeout, etc. in Javascript are used to handle the operations.
- Internally, there is a lot of activities responsible for delivering the final result that increase the complexity of the performance analysis in such context.

High level trace information view

sb1.jsonstat04:00:10.146 254 000924 µs393c280da73ab711server{}{internal.span.format=zipkin, spanld=393c280da73ab73absb1.jsonstat04:00:10.147 453 000925 µs71650fa820b8738bserver{}{internal.span.format=zipkin, spanld=71650fa820b873sb1.jsonstat04:00:10.148 651 000878 µse87ed36b8cb1d36bserver{}{internal.span.format=zipkin, spanld=87ed36b8cb103sb1.jsonstat04:00:10.149 827 000881 µs4eb42df36af35e7dserver{}{internal.span.format=zipkin, spanld=27ed36b8cb106sb1.jsongethostlogpath04:00:10.150 948 0005,717 ms7c5da2fcfa10608aserver{}{internal.span.format=zipkin, spanld=27ed47f89ae33b7sb1.jsongetconfigdiruri04:00:10.152 532 000686 µs27f6447f89ae31b7server{}{internal.span.format=zipkin, spanld=27ef6447f89ae33b7sb1.jsongethoststoragepath04:00:10.161 277 0001,504 ms36013debc9896dbbserver{}{internal.span.format=zipkin, spanld=36013debc9896sb1.jsonstat04:00:10.161 277 0001,504 ms36013debc9896dbbserver{}{internal.span.format=zipkin, spanld=36013debc9896	Trace	Name	Timestamp	Duration	ID	Process	Process tags	Tags
sb1.json stat 04:00:10.146 254 000 924 µs 393c280da73ab711 server {} {Internal.span.format=zipkin, spanId=393c280da73ab8 sb1.json stat 04:00:10.147 453 000 925 µs 71650fa820b8738b server {} {Internal.span.format=zipkin, spanId=71650fa820b873 sb1.json stat 04:00:10.149 827 000 878 µs e87ed36b8cb1d36b server {} {Internal.span.format=zipkin, spanId=7647659a320b87 sb1.json stat 04:00:10.149 827 000 871 µs e87ed36b8cb1d36b server {} {Internal.span.format=zipkin, spanId=764764769a420f36a735 sb1.json stat 04:00:10.150 948 000 5,717 ms 7c5da2fcfa10608 server {} {Internal.span.format=zipkin, spanId=27647769a47769a47769a47769a4769a475 sb1.json gethostboragepath 04:00:10.152 8227 000 4,248 ms 05038145dc8212 server {} {Internal.span.format=zipkin, spanId=36013debc98964b server {}	<pre><srch></srch></pre>	<srch></srch>	<srch></srch>	<srch></srch>	<srch></srch>	<srch></srch>	<srch></srch>	<srch></srch>
sb1.json stat 04:00:10.147 453 000 925 µs 71650fa820b8738b server {} {internal.span.format=zipkin, spanld=71650fa820b873 sb1.json stat 04:00:10.148 651 000 878 µs e87ed36b8cb1d36b server {} {internal.span.format=zipkin, spanld=71650fa820b873 sb1.json stat 04:00:10.149 827 000 881 µs 4eb42df36af35e7d server {} {internal.span.format=zipkin, spanld=7650fa820b8736 sb1.json gethostlogpath 04:00:10.159 948 000 5,717 ms 7c5da2fcfa10608 server {} {internal.span.format=zipkin, spanld=7c5da2fcfa1066 sb1.json gethoststoragepath 04:00:10.158 927 000 4,248 ms 05b338145dc8e212 server {} {internal.span.format=zipkin, spanld=7c5da2fc4106698 sb1.json stat 04:00:10.161 277 000 1,504 ms 36013debc9896db server {} {internal.span.format=zipkin, spanld=7c5da2fc4166628212 server {} {internal.span.format=zipkin, spanld=7c5da2fc4166628212 server {} {internal.span.format=zipkin, spanld=7c5da2fc4166628212 server {} {internal.span.format=zipkin, spanld=7c5da2fc4166268212 server {} {internal.span.format=zipkin, spanld=7c5da2fc416626826147	sb1.json	stat	04:00:10.145 053 000	935 µs	d6310d38d8a55608	server	8	{internal.span.format=zipkin, spanId=d6310d38d8a55608
sb1.json stat 04:00:10.148 651 000 878 µs e87ed36b8cb1d36b server 0 (internal.span.format=zipkin, spanld=e87ed36b8cb1d36b sb1.json gethostlogpath 04:00:10.149 827 000 881 µs 4eb42df36af35e7d server 0 (internal.span.format=zipkin, spanld=e87ed36b8cb1d66b sb1.json gethostlogpath 04:00:10.150 948 000 5,717 ms 7c5da2fcfa10608a server 0 (internal.span.format=zipkin, spanld=27f6447f89ae31b sb1.json gethoststoragepath 04:00:10.158 927 000 4,248 ms 05b338145dc8e212 server 0 (internal.span.format=zipkin, spanld=27f6447f89ae31b sb1.json stat 04:00:10.161 277 000 1,504 ms 36013debc9896dbb server 0 (internal.span.format=zipkin, spanld=36013debc9896db sb1.json stat 04:00:10.161 277 000 1,504 ms 36013debc9896db server 0 (internal.span.format=zipkin, spanld=36013debc9896db state System / Attribute 04:00:10.130 04:00:10.140 04:00:10.150 04:00:10.160 04:00:10.170 04:00:10.180 203 204 205 206 9 9 9 9 9 9 9	sb1.json	stat	04:00:10.146 254 000	924 µs	393c280da73ab711	server	8	{internal.span.format=zipkin, spanId=393c280da73ab711,
sb1.json stat 04:00:10.148 651 000 878 µs e87ed36b8cb1d36b server {} {internal.span.format=zipkin, spanld=e87ed36b8cb1d36b sb1.json stat 04:00:10.149 827 000 881 µs 4eb42df36af35e7d server {} {internal.span.format=zipkin, spanld=e4b42df36af355 sb1.json gethoststoragepath 04:00:10.150 948 000 5,717 ms 7c5da2fcfa10608a server {} {internal.span.format=zipkin, spanld=27f6447f89ae31b7 sb1.json gethoststoragepath 04:00:10.158 927 000 4,248 ms 05b338145dc8212 server {} {internal.span.format=zipkin, spanld=27f6447f89ae31b7 sb1.json stat 04:00:10.158 927 000 4,248 ms 05b338145dc8212 server {} {internal.span.format=zipkin, spanld=27f6447f89ae35b sb1.json stat 04:00:10.161 277 000 1,504 ms 36013debc9896db server {} <	sb1.ison	stat	04:00:10.147 453 000	925 µs	71650fa820b8738b	server	0	{internal.span.format=zipkin.spanId=71650fa820b8738b.
sb1.json stat 04:00:10.149 827 000 881 µs 4eb42df36af35e7d server {} {internal.span.format=zipkin, spanid=4eb42df36af35 sb1.json gethostlogpath 04:00:10.150 948 000 5,717 ms 7c5da2fcfa10608a server {} {internal.span.format=zipkin, spanid=4eb42df36af35 sb1.json gethoststoragepath 04:00:10.152 532 000 686 µs 27f6447f89ae31b7 server {} {internal.span.format=zipkin, spanid=27f6447f89ae3 sb1.json gethoststoragepath 04:00:10.152 927 000 4.248 ms 05b338145dc8e212 server {} {internal.span.format=zipkin, spanid=27f6447f89ae3 sb1.json stat 04:00:10.151 8927 000 4.248 ms 05b338145dc8e212 server {} {internal.span.format=zipkin, spanid=05b338145dc8e394 sb1.json stat 04:00:10.161 277 000 1,504 ms 36013debc9896db server {} {internal.span.format=zipkin, spanid=36013debc9896 sb1.json stat 04:00:10.161 00:00:10.140 04:00:10.150 04:00:10.160 04:00:10.170 04:00:10.180 state System / Attribute 04:00:10.130 04:00:10.140 04:00:10.150 04:00:10.160 04:00:10.160 04:00:10.170 <td></td> <td></td> <td>04:00:10.148 651 000</td> <td>878 us</td> <td>e87ed36b8cb1d36b</td> <td>server</td> <td>0</td> <td></td>			04:00:10.148 651 000	878 us	e87ed36b8cb1d36b	server	0	
sb1json gethostlogpath 04:00:10.150 948 000 5,717 ms 7c5da2fcfa10608a server {} {internal.span.format=zipkin, spanl=7c5da2fcfa10608a server {} sb1json gethoststoragepath 04:00:10.150 948 000 5,717 ms 7c5da2fcfa10608a server {} {internal.span.format=zipkin, spanl=7c5da2fcfa10608a server {} sb1json gethoststoragepath 04:00:10.150 948 000 5,717 ms 7c5da2fcfa10608a server {} {internal.span.format=zipkin, spanl=27f6447f89ae31b sb1json gethoststoragepath 04:00:10.158 927 000 4,248 ms 05b338145dc8e212 server {} (internal.span.format=zipkin, spanl=05b338145dc8e312 sb1json stat 04:00:10.161 277 000 1,504 ms 36013debc9896dbb server {} (internal.span.format=zipkin, spanl=36013debc9896 sbt te System Attribute 04:00:10.130 04:00:10.140 04:00:10.150 04:00:10.160 04:00:10.170 201 203 04:00:10.130 04:00:10.140 04:00:10.150 04:00:10.160 04:00:10.170 203 204 205 9 9 9 9 9 9 204 205 9 9 9 9 9 9 205 206 9 9 9 9 9 9 206 9 9 9 9 9 9 9 207 9 9 9 9 9 9 9 206 9 9 9 9 9 <td< td=""><td>-</td><td></td><td></td><td></td><td></td><td></td><td>n</td><td></td></td<>	-						n	
sb1.json getconfigdiruri 04:00:10.152 532 000 686 µs 27f6447f89ae31b7 server {} {internal.span.format=zipkin, spanId=27f6447f89ae31b7 sb1.json gethoststoragepath 04:00:10.158 927 000 4.248 ms 05b338145dc8e212 server {} {internal.span.format=zipkin, spanId=27f6447f89ae31b7 sb1.json stat 04:00:10.158 927 000 1,504 ms 36013debc9896dbb server {} {} {internal.span.format=zipkin, spanId=36013debc9896 sb1.json stat 04:00:10.161 277 000 1,504 ms 36013debc9896dbb server {} <	-						U D	
sb1,son gethoststoragepath 04:00:10.158 927 000 4,248 ms 05b338145dc8e212 server {} {internal.span.format=zipkin, spanid=05b338145dc8e398 sb1,son stat 04:00:10.161 277 000 1,504 ms 36013debc9896dbb server {} {internal.span.format=zipkin, spanid=05b338145dc8e398 Li Histogram Properties E State System Explorer Image: Theia Data Provider Time Graph View Error Log Image: Theia Data Provider Time Graph View Image: Theia Data Provider Time Graph View 04:00:10.160 04:00:10.170 04:00:10.180 201 04:00:10.130 04:00:10.140 04:00:10.150 04:00:10.160 04:00:10.170 04:00:10.180 201 203 Image: Theia Data Provider Time Graph View Image: Theia Data Provider Time Graph View Image: Theia Data Provider Time Graph View 04:00:10.160 04:00:10.170 04:00:10.180 201 04:00:10.130 04:00:10.140 04:00:10.150 04:00:10.160 04:00:10.170 04:00:10.180 203 Image: Theia Data Provider Time Graph View Image: Theia Data Provider Time Graph View </td <td></td> <td></td> <td></td> <td></td> <td></td> <td></td> <td>ប</td> <td></td>							ប	
sb1.json stat 04:00:10.161 277 000 1,504 ms 36013debc9896dbb server {} {internal.span.format=zipkin, span!d=36013debc9896db Image: Histogram Properties E State System Explorer 20 Theia Data Provider Time Graph View 9 Error Log Image: Histogram Properties Image: Histogram Properis Image: Histogram Propertis </td <td></td> <td></td> <td></td> <td></td> <td></td> <td>1</td> <td>{}</td> <td></td>						1	{}	
Histogram Properties E State System Explorer State System / Attribute 04:00:10.130 04:00:10.140 04:00:10.150 04:00:10.160 04:00:10.170 201 203 204 205 206 207 209 210 211 210 211 211 212 213	sb1.json	gethoststoragepath	04:00:10.158 927 000) 4,248 ms	1	1	8	{internal.span.format=zipkin, spanId=05b338145dc8e212,
State System / Attribute 04:00:10.130 04:00:10.140 04:00:10.150 04:00:10.160 04:00:10.170 04:00:10.180 201	sb1.json	stat	04:00:10.161 277 000) 1,504 ms	36013debc9896dbb	server	8	{internal.span.format=zipkin, spanId=36013debc9896dbb
State System / Attribute 04:00:10.130 04:00:10.140 04:00:10.150 04:00:10.160 04:00:10.170 04:00:10.180 201	Histogra	m 🔲 Properties 🔚 State System	Explorer 🛛 🗖 Theia D	ata Provider	Time Graph View 🧕	Error Log		<u>≖∷</u> ∰ ⊊ ⊒ ⋩ ⋩ ≠ ⋳₽€€ ≝ ≠ 8 ⊓ ⊡
201 \$, 203 \$, 204 \$, 205 \$, 206 \$, 207 \$, 209 \$, 210 \$, 211 \$, 212 \$, 213 \$, \$, \$,	State Syste	m / Attribute 04:	00:10.130 04	4:00:10.140	04:00:10.15		04:00:10.16	0 04:00:10.170 04:00:10.180
204 205 206 207 209 210 211 212 213 gethoststoragepath on client	201		1					• •
205 206 207 209 210 211 212 213 gethoststoragepath on client	203						<u>st</u>	
206 207 209 210 211 212 213 gethoststoragepath on client	204							
207 209 210 211 212 213	205							
209 getschemaattributes on client 210 getschemaattributes on client 211 gethoststoragepath on client 213 gethoststoragepath on client	206							
210 getschemaattributes on client 211								
211 212 213								
212 gethoststoragepath on client gethoststogpat					getsche	maattribute	s on client	
213 gethoststoragepath on client gethostlogpat gethostst								
			asthectsteesessthe					
		vete	gernösistoragepath o	in cuenc	gethostio	gpal	gernostst	
► Spans								
▼ spails	Double							

Fig1: high level trace

Definition of the problem

- In the previous figure, the higher view of the latency of each operation executed in Theia is generally exposed in the way that distributed tracers works and do not tell much except latency.
- Although the latency represents the real time the operation took to complete, pinpointing the source of the problem is something else in Node.js.
- Tracing the application with distributed tracers, when such a problem happens most of the time, cannot help pinpointing the operation responsible of the fault propagation.
- A non optimized code, or an operation can slow down or block the internal event-loop and consequently delay all pending operations in the stack.

Objective

Asynchronous operations

- Tracking asynchronous resources and their respective callbacks in Node.js is very important
- Asynchronous operations go through phases in the event-loop and, at each phase, their respective callback are executed.
- The way the event-loop will behave is a function of the operations that it has enqueued.
- In this context, a real performance analysis of Node.js applications must involve collecting lower internal information on its functioning for correlation with the higher level ones.
- Such approach should result in accurate tools for performance analysis in the Node.js environment.

Event-loop at

Glance

Abstraction of the Event-loop

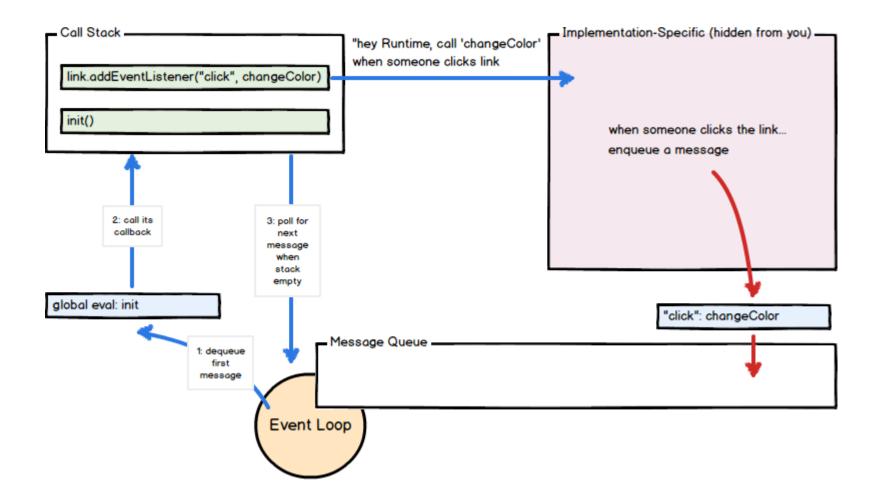


Fig 2: event-loop abstraction[2]

Event-loop at

Glance

Delegation of tasks to Workers

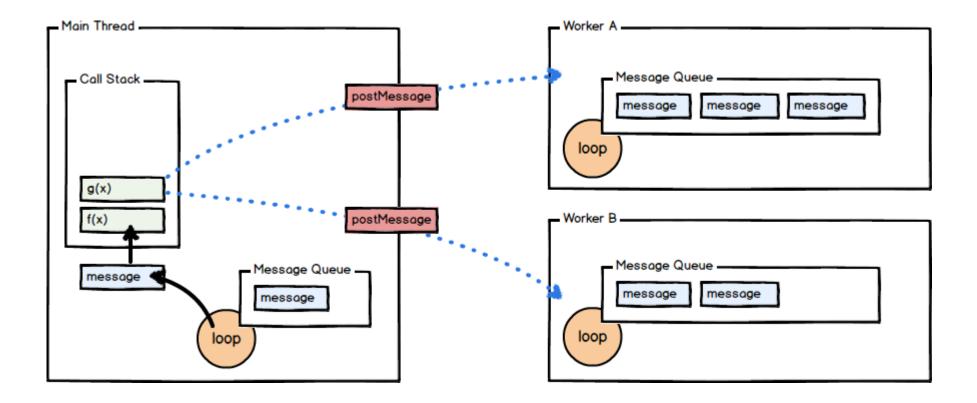


Fig 3:Tasks delegation to workers[2]

Workers view

The communication event between the workers are captured in this view (postMessage method) of the trace.

		04:00:38.4	90 04:00:38.500	04:00:38.510	04:00:38	520 04:00:38.5	04:00:38.54	0 04:00:38.550	04:00:38.560	04:00:38.570		
E Experiment	Е грс4	.json i≡ rpca	ll ¤ ⊫ rpc.json	≣ sb2.json	≡ sb1.js	on ≣ rpcall	≣ rpc5.json	≣ sb1.json				
stamp	Duration	ID	Process	Proce	ess tags T	gs						
>	<srch></srch>	<srch></srch>	<srch></srch>	<srch:< td=""><td>> P</td><td>stMessage</td><td></td><td></td><td></td><td></td></srch:<>	> P	stMessage						
:38.484 635 000	218 µs	3006ca97a3c90f	fc rpc	{}	{i	{internal.span.format=zipkin, method=log, span.kind=client, payload.id=182, dir=clsend}						
38.486 193 000	734 µs	033ae0264043a	26c server	{}	{i	(internal.span.format=zipkin, spanid=033ae0264043a26c, span.kind=server, payload.id=178, dir=srv,						
38.487 461 000	605 µs	f15e13bb577ee2	2f4 server	{}	{i	{internal.span.format=zipkin, spanId=f15e13bb577ee2f4, span.kind=server, payload.id=179, dir=srv,						
38.498 187 000	21,228 ms	dfaedd8a69c0cl	07 server	{}	{i	{internal.span.format=zipkin, spanId=dfaedd8a69c0cb07, span.kind=server, payload.id=180, dir=srv						
38.518 670 000	289 µs	a20ec3acd4f48e	42 грс	rpc {} {internal.span.format=zipkin, method=log, span.kind=client, payload.id=18						id=180, dir=clsend}		
38.520 311 000	4,228 ms	6ed0fed47a05b	Odc server	{}	{i	{internal.span.format=zipkin, spanId=6ed0fed47a05b0dc, span.kind=server, payload.id=181, dir=srv,						
38.523 033 000	1,005 ms	e6a310124033f1	1f грс	{}	{i	{internal.span.format=zipkin, method=log, span.kind=client, payload.id=181, dir=clsend}						
38.541 835 000	36,787 ms	2d8982536214e	3d9 грс	{}	{i	ternal.span.forn	nat=zipkin, metho	od= <mark>postMessage</mark> ,	span.kind=clien	t, payload.id=29, dir=lspsend}		
38.542 156 000	41,018 ms	59034b87a4a64	1с9 грс	{}	{i	ternal.span.forn	nat=zipkin, metho	od= <mark>postMessage</mark> ,	span.kind=clien	t, payload.id=30, dir=lspsend}		
38.546 154 000	33,022 ms	a83f066badd07	аа7 грс	{}	{i	ternal.span.forn	nat=zipkin, metho	od= <mark>postMessage</mark> ,	span.kind=clien	t, payload.id=31, dir=lspsend}		
38.562 206 000	918 µs	0dad51eef0a094	1е3 грс	{}	{i	ternal.span.forn	nat=zipkin, metho	od=stat, span.kino	d=client, payload	d.id=142, dir=clsend}		
38.562 460 000	410 µs	b32697974dfbc	50b грс	{}	{i	ternal.span.forn	nat=zipkin, metho	od=stat, span.kino	l=client, payload	d.id=141, dir=clsend}		
:38.567 402 000	2,995 ms	920f8a1be51af1	8d server	{}	{i	ternal.span.forn	nat=zipkin, spanlo	d=920f8a1be51af	18d, span.kind=s	server, payload.id=182, dir=srv,		
:38.583 963 000	2,294 ms	608417924c68ff	62 server	{}	{i	{internal.span.format=zipkin, spanId=608417924c68ff62, span.kind=server, payload.id=141, dir=srv,						
:38.588 373 000	1,988 ms	d6186090928bb	bac server	{}	{i	ternal.span.forn	nat=zipkin, spanlo	d=d6186090928bb	bac, span.kind=	server, payload.id=142, dir=srv		

Fig 4: Intercepting workers communication

Event-loop at

Glance

Event-loop phases

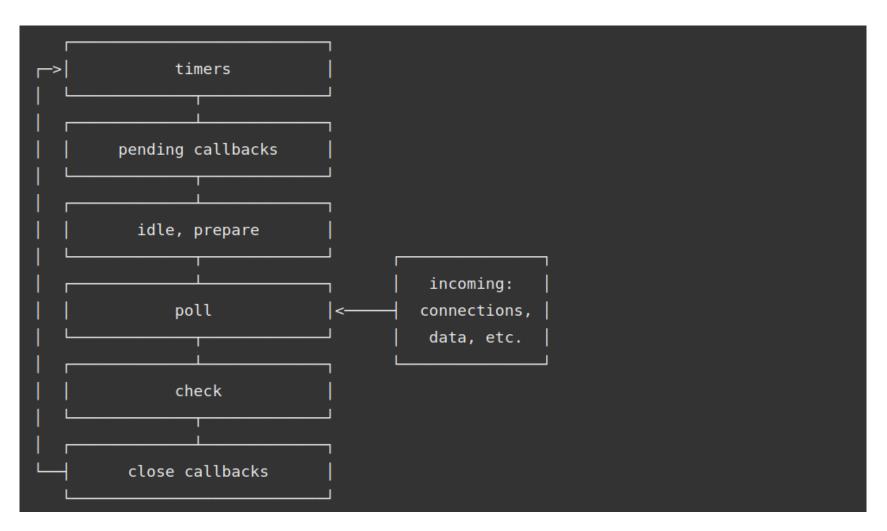


Fig 5: event-loop phase[5]

Methodology

- Tracing Node.js internals should give more insights on application bottlenecks
- Internal queues are sometimes responsible for high latency propagation at higher level operations
- A common problem with asynchronous operations is that they may enqueue other operations as a tree of operations.
- In such scenario, it results in the event-loop stucking in the same phase until it completes all operations.
- This results in increasing the latency of other pending operations in queues of other phases.

Methodology

Our Approach

- From high level information, track down the different operations at lower layers and reconstruct a vertical sequence of the request

- 3 layers are considered: The application layer, the Node.js layer and the kernel layer

- In a vertical request sequencing, latency at each layer can be identified

Problem

How to vertically inject the context of the trace to reconstruct the sequences of the execution?

Methodology

Libuv

- Node.js relies on this library to manage asynchronous operations
- It clearly is a core Node.js IO operations library and is responsible for interacting with the OS

Instrumentation

- Libuv and Node.js internals are instrumented to obtain low level information at the intermediate layer
- Application (Theia) is instrumented with Zipkin to obtain high level information
- Kernel trace is collected and correlation algorithms are designed for vertical context injection and sequence reconstruction

Current

Results

- Algorithms for heterogeneous traces correlation and vertical context injection
- Correlation between high level information and intermediate layer (Node.js), and intermediate layer with Kernel layer reconstruction of the execution sequence)

Current

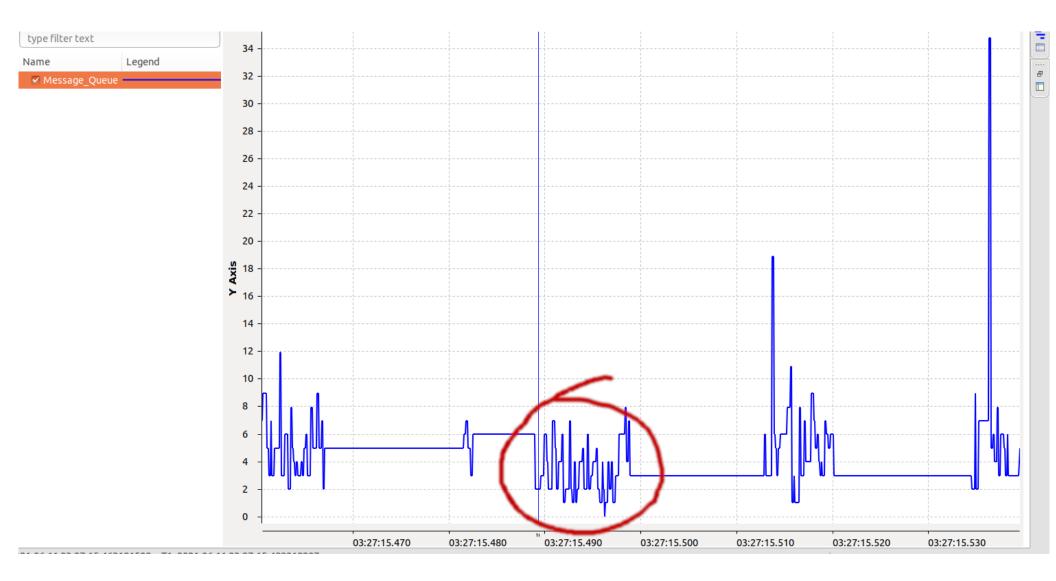
Async Operations times in the event-loop **Results** type filter text 120 k Name Legend 110 k 100 k 90 k 80 k 70 k Y Axis 60 k 50 k 40 k 30 k 20 k 10 k a a dh 03:27:15.050 03:27:15.100 03:27:15.150 03:27:15.200 03:27:15.250 03:27:15.300 03:27:15.350 03:27:15.400 10 k type filter text Name Legend 9,5 k 🗹 Async 9 k 8,5 k 8 k 7,5 k Y Axis 7 k 6,5 k 6 k 5,5 k 5 k 4,51 4 k 03:27:15.150 03:27:15.200 03:27:15.250 03:27:15.300 03:27:15.350 03:27:15.400

POLYTECHNIQUE MONTREAL – *Hervé Kabamba*

Current

Results

Number of operations enqueued / 100 ms



Ongoing Work

- Intensive work is being done on the development of views tailored to the performance analysis of Theia, based on the preliminary results
- Work on identifying the critical path is also ongoing

References

[1] Piero Borrelli, https://blog.logrocket.com/a-complete-guide-to-the-node-js-event-loop/

[2] Erin Swenson-Healey, https://blog.carbonfive.com/the-javascript-event-loop-explained/

[3] Aman Agrawal,

https://www.loginradius.com/blog/async/understanding-event-loop/

[4] Tania Rascia,

https://www.digitalocean.com/community/tutorials/understanding-the-event-l oop-callbacks-promises-and-async-await-in-javascript

[5] https://nodejs.org/en/docs/guides/event-loop-timers-and-nexttick/

Questions?

herve.kabamba-mbikayi@polymtl.ca



POLYTECHNIQUE MONTREAL – *Hervé Kabamba*