Oracle Service Bus Thread Model

Introduction

This subject has been misunderstood by some Oracle Fusion Middleware's developers over the last several years. I can tell you that understanding this topic is of the utmost importance because we need to ensure the stability of the production environment, especially when your services begin to experience a high volume of consumption. It's about performance and how it can be improved with a simple understanding of the Oracle Service Bus architecture and following some best practices.

In this post, you will learn how the OSB Thread Model works and how you can make good use of it during the development of your project. The following subjects will be explained: Service Callout, Routing, Publish, Quality of Service (Qos), Work Manager and a few differences between blocking and non blocking architectures.

Objective

Understanding the difference between Routing, Service Callout and Publish and how their internal threading structure works throughout development of a practical use case in which a partner WebService (that contains an intentional sleep of 5 seconds) will be consumed in a high volume way using the Apache Jmeter tool and its stress features. Look at the following picture:

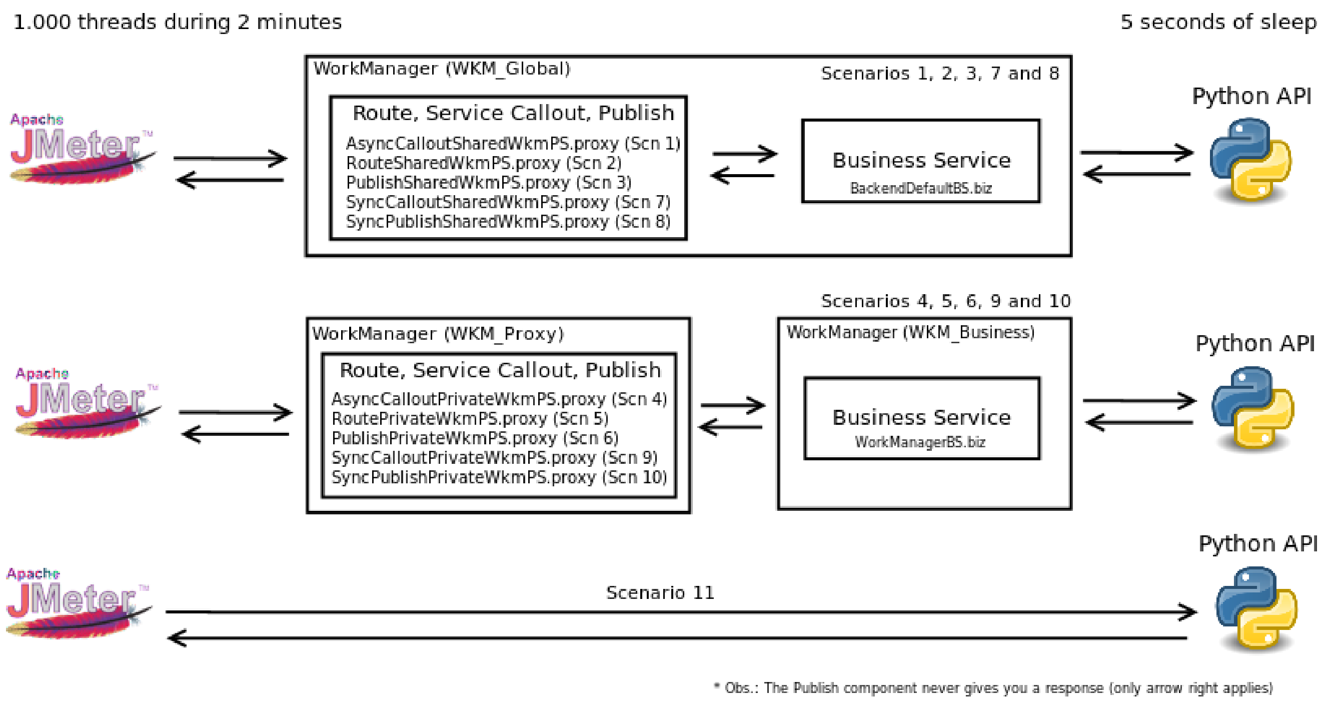

1.000 threads during 2 minutes
5 seconds of sleep
WorkManager (WKM_Global)
Scenarios 1, 2, 3, 7 and 8
Route, Service Callout, Publish
AsyncCalloutSharedWkmPS.proxy (Scn 1)
RouteSharedWkmPS.proxy (Scn 2)
PublishSharedWkmPS.proxy (Scn 3)
SyncCalloutSharedWkmPS.proxy (Scn 7)
SyncPublishSharedWkmPS.proxy (Scn 8)
Business Service
BackendDefaultBS.biz
Python API
Scenarios 4, 5, 6, 9 and 10
WorkManager (WKM_Proxy)
Route, Service Callout, Publish
AsyncCalloutPrivateWkmPS.proxy (Scn 4)
RoutePrivateWkmPS.proxy (Scn 5)
PublishPrivateWkmPS.proxy (Scn 6)
SyncCalloutPrivateWkmPS.proxy (Scn 9)
SyncPublishPrivateWkmPS.proxy (Scn 10)
WorkManager (WKM_Business)
Business Service
WorkManagerBS.biz
Python API
Scenario 11
Python API
* Obs.: The Publish component never gives you a response (only arrow right applies)

Figure 1 – Practical experience mixing OSB scenarios

As you can see above, three main groups were created:

1. **Global WorkManager** – In this group Proxy and Business Service makes use of a shared Weblogic WorkManager called WKM_Global which was created manually in WebLogic console and specified on PS and BS dispatcher configuration. This is the most common scenario found because when you don't set any value in dispatcher policy then the Weblogic defaukt WorkManager will be used. It is dangerous because if your PS receive a high load rate of requests you can ended up with an unstable environment.
2. **Private WorkManager** – In this case, I created two WorkManagers (WKM_Proxy and WKM_Business) the first was used on PS and the last in BS. You will understand why later in this post.
3. **Point to Point** – Here, you can see the client (Jmeter) invoking directly the exposed service. This scenario was created to collect the benchmark in order to see what is the maximum throughput of the Python API. It will be useful for the results comparison section later.

Practical Experience

In this experience, we built a Python Api that contains an intentional 5 sleep seconds which was consumed through Oracle Service Bus using as much combinations of scenarios as possible, mixing Service Callout, Route and Publish with private and shared WorkManagers. To add some emotion to this experiment the Jmeter tool was used in order to stress the OSB with 1.000 threads during 2 minutes, as you can see in the following strategy:

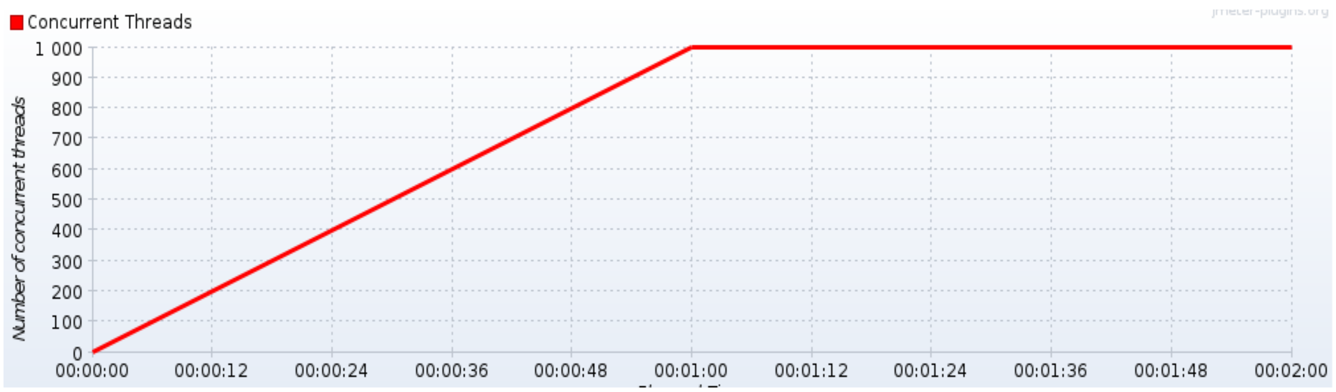


Figure 2 – Apache Jmeter stress test strategy

Environment used in this experiment:

- Oracle Fusion Middleware version 11.1.1.7 (Oracle Service Bus)
- Operating System: Linux Mint 18.1 Cinnamon 64-biy
- Cinnamon Version: 3.2.7
- Linux Kernel: 4.4.0-66-generic
- Processor: Intel Core i7-3520M CPU 2.90GHz x 2
- Memory Ram: 16 Gb
- Hard Drive: 250 Gb – Samsung 850 Evo (SSD)
- Graphics Card: NVIDIA Corporation GF108M [Geforce GT 635 M]

Source Code Available (GitHub)

You will find the whole files that you need to run this experiment locally on your computer here:

https://github.com/groundswellgroup/osbthreadmodel

Main folders explanation:

- **article** – this article in an ODT format (Libre Office)
- **Diagrams** – the diagrams built in this post
- **images** – all images used in this post
- **jmeter_project** – this is the xml project file of the Apache Jmeter Tool
- **jmeter_results_csvs** – here are all jmeter csv result files generated on my experiment
- **python** – here is the python source code to run a http multi-threaded server, which is the api used in this experiment as an endpoint webservice
- **results** – there is a sheet (libre office format – ODS) with result tables data
- **sourcecode_osb** – here you will find the full source code project to be imported to your Oepe (Eclipse IDE)

Differences between Service Callout, Route and Publish

There were quite a few differences between these components and it is pretty easy to choose what component should be used in a project. Follow some decision points:

Route

- You can only execute one route in your Proxy Service
- Last node in request processing
- It can only be created in a route node
- Default behaviour is synchronous for external callers and asynchronous internally in Osb (non blocking architecture)

Service Callout

- It can have multiple Service Callout nodes in a Proxy services
- It can be invoked from the request and/or response pipelines
- Used to enrich the incoming request or outgoing response
- Default behaviour is synchronous for external and internal calls (it can degrade performance under heavy loads)

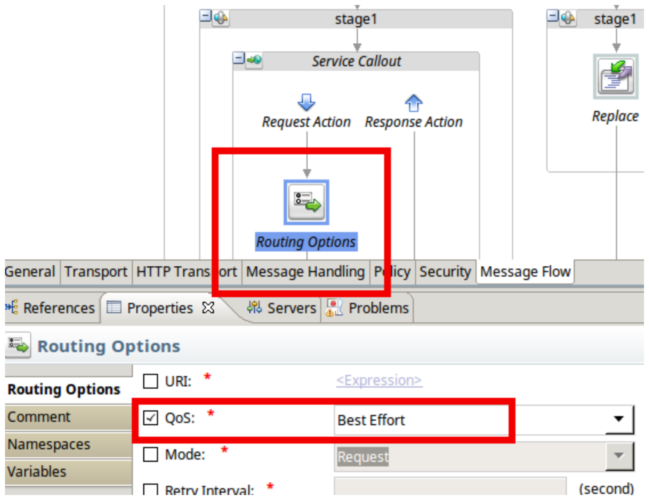
Publish

- It is a fire forget request and you never will receive a response
- It can be invoked from the request and/or response pipelines
- Default behaviour is asynchronous for external and internal calls (the most fast component due to lack of response)

*** Important:**

There is a little trick to modify the normal behaviour that can be used for any of those 3 components. All you need to do is to add a “Routing Options” box as the first element (preferably) in your service callout or routing pipeline. Following these instructions:

Insert Into -> Communication -> Routing Options



☐ URI: * <Expression>
☑ QoS: * Best Effort
☐ Mode: * Request
☐ Retry Interval: * (second)

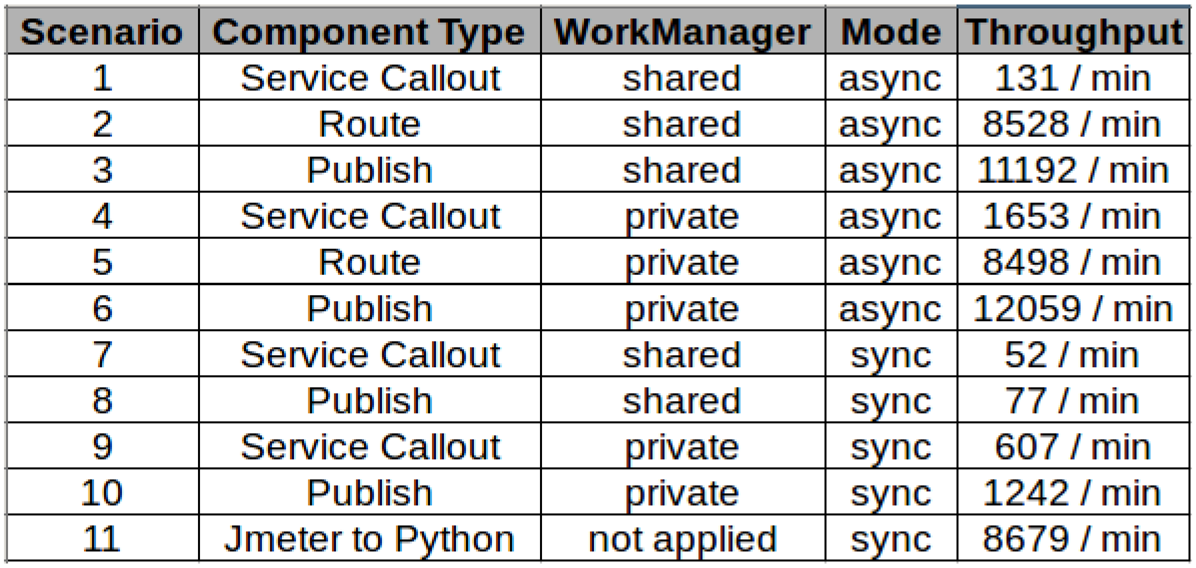
Figure 3 – Routing Options Box (Modifying the normal behaviour)

In the QoS select box, you have two options to choose:

- **Best Effort**: transforms the request into asynchronous mode. It makes sense to be used in Service Callout only because it is synchronous and this way you can modify the default behaviour to increase the throughput (see the result tables).
- **Exactly once**: transforms the request into synchronous mode. It makes sense to be used in Publish because it is asynchronous and this way you can modify the default behaviour. If you are considering to use it on a Route, please, don’t do that, replace the Route by a Service Callout component because it is synchronous by default.

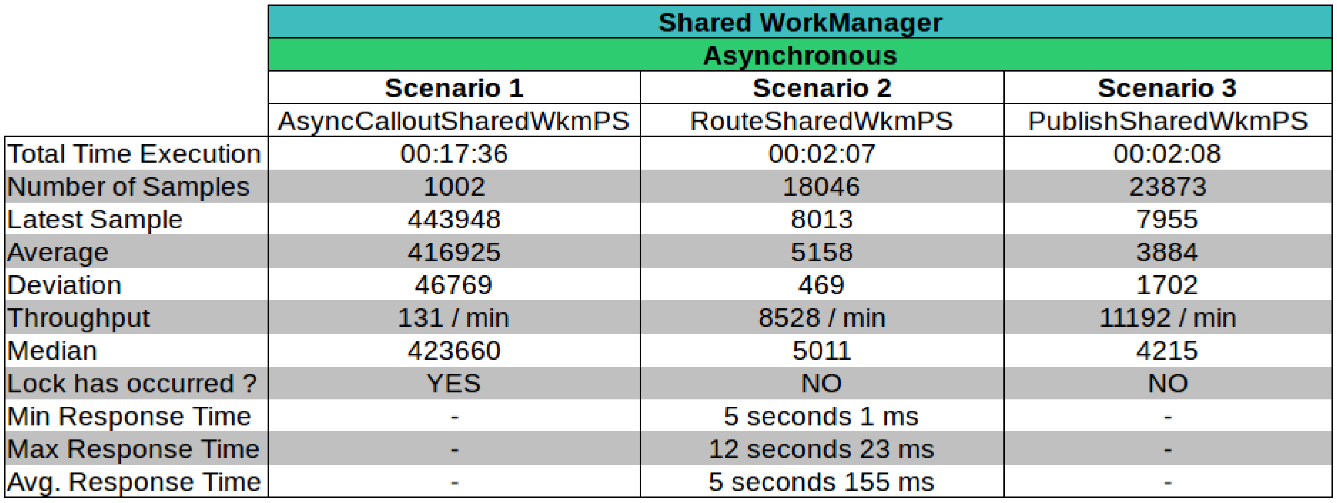
Benchmark results for OSB Scenarios

In this section you have some tables with the overall benchmark results. Three components were used, Service Callout, Route and Publish in a combination between WorkManager (shared or private) and Mode (sync or assync). You can see a detailed result analysis in next the section, but I can tell you that these results are impressive, because I've always heard some comments about what would be the best way to use these components in some scenarios. Based on these results you can have a better idea of how to use these components consciously in your project.



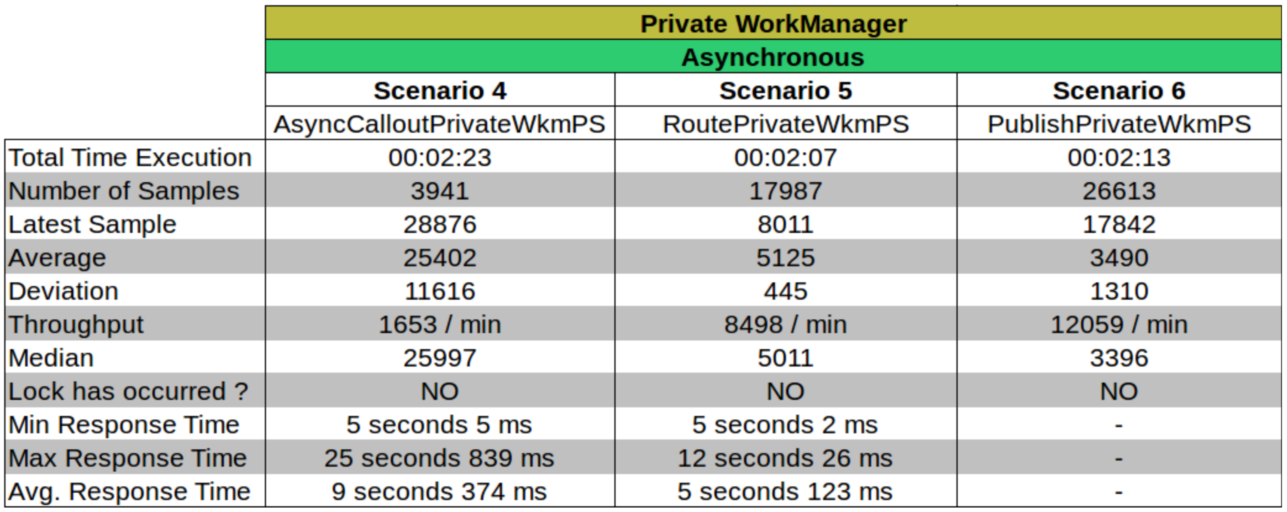
Scenario	Component Type	WorkManager	Mode	Throughput
1	Service Callout	shared	async	131 / min
2	Route	shared	async	8528 / min
3	Publish	shared	async	11192 / min
4	Service Callout	private	async	1653 / min
5	Route	private	async	8498 / min
6	Publish	private	async	12059 / min
7	Service Callout	shared	sync	52 / min
8	Publish	shared	sync	77 / min
9	Service Callout	private	sync	607 / min
10	Publish	private	sync	1242 / min
11	Jmeter to Python	not applied	sync	8679 / min

Figure 4 – Main results and aggregated throughput rates



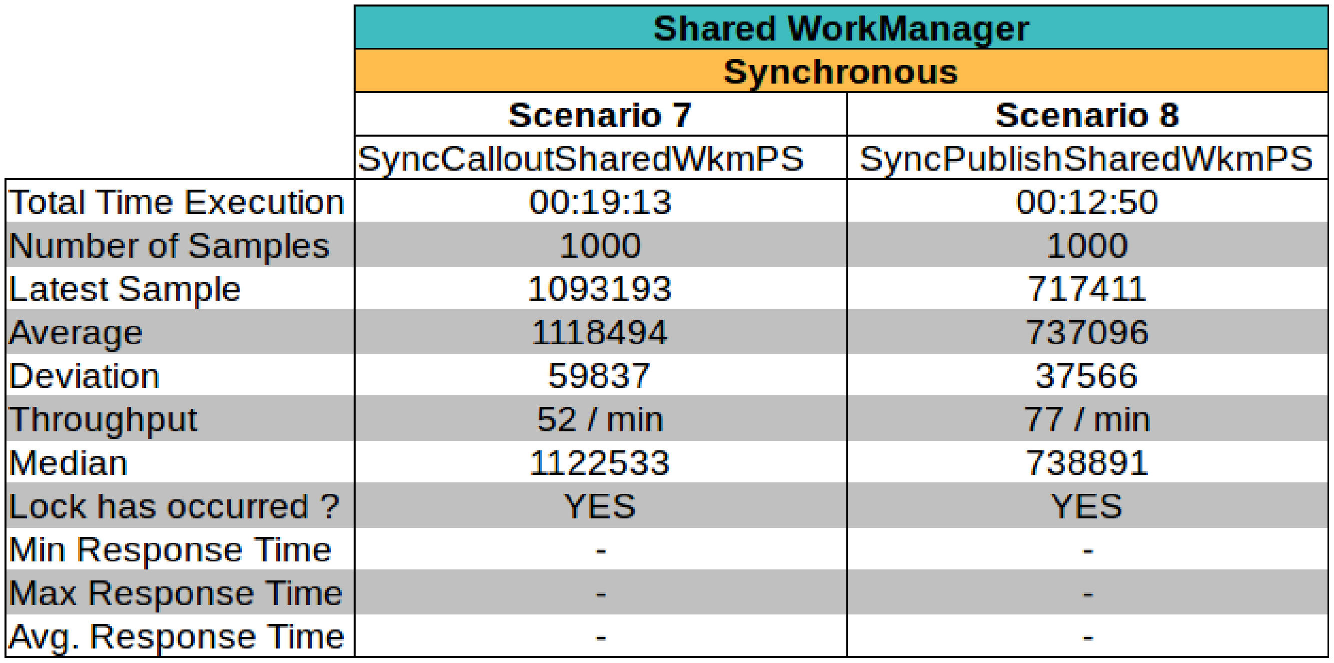
	Shared WorkManager		
	Asynchronous		
	Scenario 1	Scenario 2	Scenario 3
	AsyncCalloutSharedWkmPS	RouteSharedWkmPS	PublishSharedWkmPS
Total Time Execution	00:17:36	00:02:07	00:02:08
Number of Samples	1002	18046	23873
Latest Sample	443948	8013	7955
Average	416925	5158	3884
Deviation	46769	469	1702
Throughput	131 / min	8528 / min	11192 / min
Median	423660	5011	4215
Lock has occurred ?	YES	NO	NO
Min Response Time	-	5 seconds 1 ms	-
Max Response Time	-	12 seconds 23 ms	-
Avg. Response Time	-	5 seconds 155 ms	-

Figure 5 – Benchmark results for scenarios 1, 2 and 3



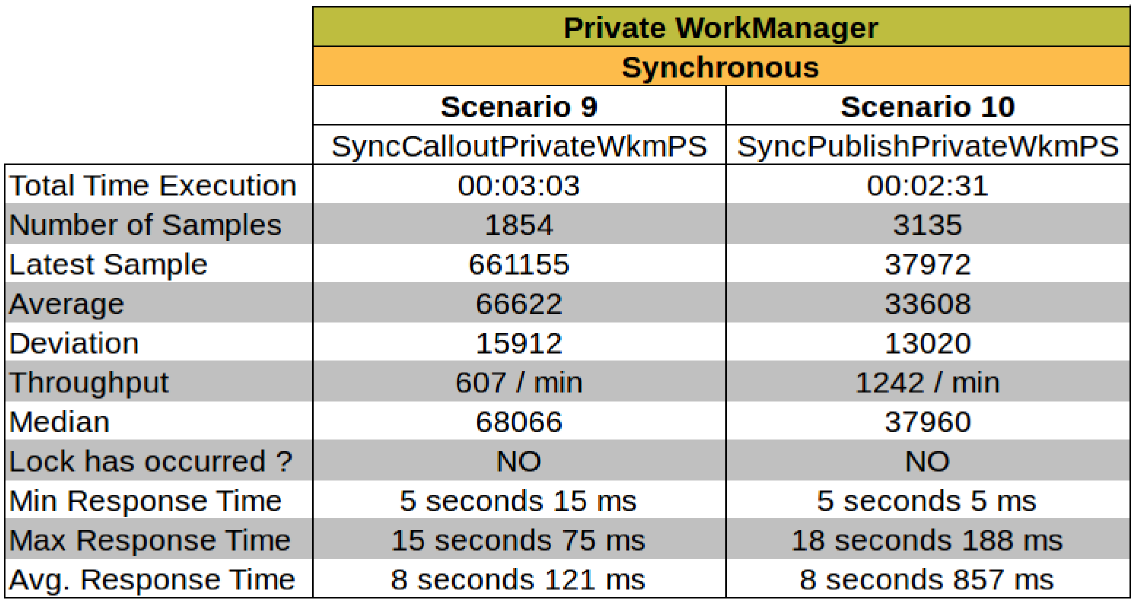
	Private WorkManager		
	Asynchronous		
	Scenario 4	Scenario 5	Scenario 6
	AsyncCalloutPrivateWkmPS	RoutePrivateWkmPS	PublishPrivateWkmPS
Total Time Execution	00:02:23	00:02:07	00:02:13
Number of Samples	3941	17987	26613
Latest Sample	28876	8011	17842
Average	25402	5125	3490
Deviation	11616	445	1310
Throughput	1653 / min	8498 / min	12059 / min
Median	25997	5011	3396
Lock has occurred ?	NO	NO	NO
Min Response Time	5 seconds 5 ms	5 seconds 2 ms	-
Max Response Time	25 seconds 839 ms	12 seconds 26 ms	-
Avg. Response Time	9 seconds 374 ms	5 seconds 123 ms	-

Figure 6 – Benchmark results for scenarios 4, 5 and 6



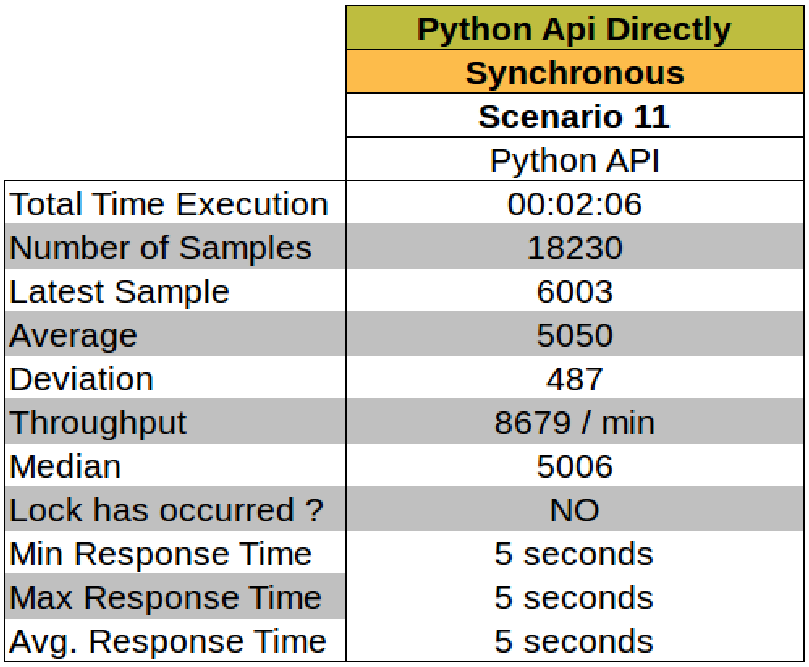
	Shared WorkManager	
	Synchronous	
	Scenario 7	Scenario 8
	SyncCalloutSharedWkmPS	SyncPublishSharedWkmPS
Total Time Execution	00:19:13	00:12:50
Number of Samples	1000	1000
Latest Sample	1093193	717411
Average	1118494	737096
Deviation	59837	37566
Throughput	52 / min	77 / min
Median	1122533	738891
Lock has occurred ?	YES	YES
Min Response Time	-	-
Max Response Time	-	-
Avg. Response Time	-	-

Figure 7 – Benchmark results for scenarios 7 and 8



	Private WorkManager	
	Synchronous	
	Scenario 9	Scenario 10
	SyncCalloutPrivateWkmPS	SyncPublishPrivateWkmPS
Total Time Execution	00:03:03	00:02:31
Number of Samples	1854	3135
Latest Sample	661155	37972
Average	66622	33608
Deviation	15912	13020
Throughput	607 / min	1242 / min
Median	68066	37960
Lock has occurred ?	NO	NO
Min Response Time	5 seconds 15 ms	5 seconds 5 ms
Max Response Time	15 seconds 75 ms	18 seconds 188 ms
Avg. Response Time	8 seconds 121 ms	8 seconds 857 ms

Figure 8 – Benchmark results for scenarios 9 and 10



	Python Api Directly
	Synchronous
	Scenario 11
	Python API
Total Time Execution	00:02:06
Number of Samples	18230
Latest Sample	6003
Average	5050
Deviation	487
Throughput	8679 / min
Median	5006
Lock has occurred ?	NO
Min Response Time	5 seconds
Max Response Time	5 seconds
Avg. Response Time	5 seconds

Figure 9 – Benchmark results for scenario 11

Result Analysis

After collecting the benchmark results and fulfill the tables showed on last section I realized that some results had the expected behaviour while others were a surprise to me. After this experiment, I could have a better comprehension of how the OSB internally works and certainly it will help me in future projects. Let's go to the analysis:

- The fastest component was the Publish (scenarios 3 and 6) and it is an expected behaviour since it is an asynchronously fire forget interaction. However, it was roughly 30% faster than the scenario 11 where the Python Api was called directly from Jmeter without involving the OSB in the flow and it was a little surprise to me.
- The most surprising discovery was the evidence that the route component was 5 times faster in comparison to Service Callout (scenarios 2 and 5). I know that it was faster because it is asynchronous, but 5 times?
- I didn't know that the Service Callout / Publish could cause a starvation or deadlock (scenarios 1, 7 and 8) during heavy loads when it uses a shared WorkManager between PS and BS. I never even considered to use different Workmanagers between them. I read about it on Antony's Blog (referenced at the end) and I decided to try and for my surprise it proved to be true. In fact, many Osb production environments are configured with this dangerous scenario right now because many developers don't know about it as well, but maybe they didn't realize it because a huge load is needed in order to cause a deadlock.
- An uncommon thing to be done is to add asynchronous behaviour to the Service Callout through "Routing Option Box" but I did it on scenario 4 and I realized that it caused the throughput to double in comparison to the scenario 9 which is the normal scenario. Some people told me that this way it would be impossible to grab the response, but it worked in this experiment.
- For asynchronous scenarios with Route (2 and 5) and for Publish (3 and 6), it makes no really difference to have a private or shared WorkManager since the throughput was practically the same. So, from this lesson, we learned that it only matters to use different WorkManagers when we are dealing with synchronous operations (blocking architecture).

Conclusion

From this article, I conclude that we don’t have to change the normal behavior of the OSB components (Route, Callout and Publish) using the Routing Options box in a normal daily projects, but you can take some advantages from this article using routing options when you need to do it. The most important lesson that I am teaching here is:

“Always use different WorkManagers for PS and BS to handle synchronous requests, instead of it could cause a deadlock in your environment when your services begin to experience a high volume of consumption. So, because you will never know when it could happen, always do it”.

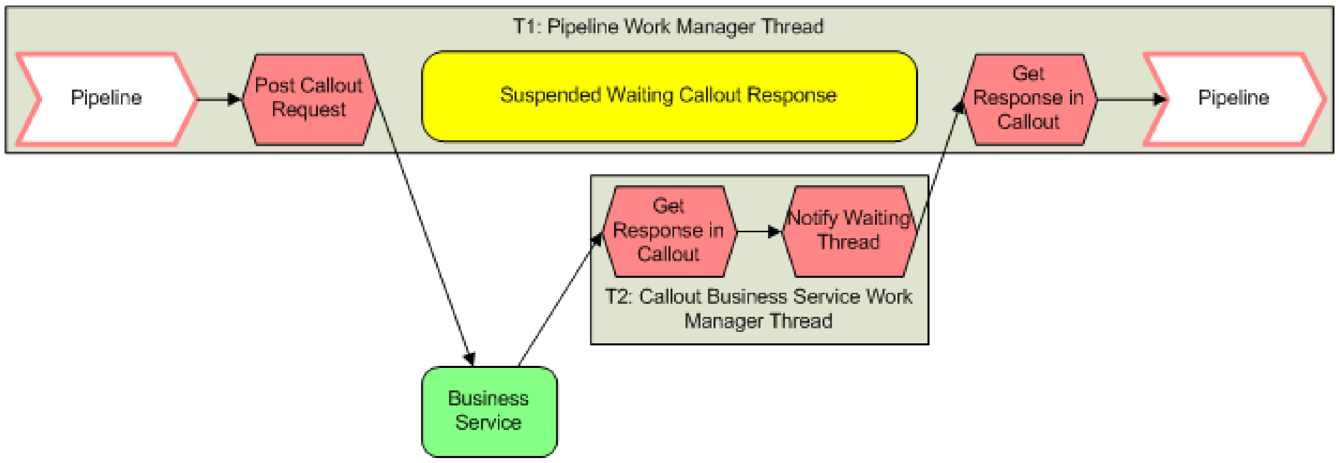


Figure 10 – Threads are locked (suspended) while the main pipeline wait for response

References:

REYNOLDS, ANTONY - Following the Thread in OSB – Oracle Blog – October 2012
https://blogs.oracle.com/reynolds/entry/following_the_thread_in_osb

ORACLE – Official OFMW Developer's Guide 11g - 2017
http://docs.oracle.com/cd/E23943_01/dev.1111/e15866/architecture.htm#i1035778

ORACLE – Modeling Message Flow in Oracle Service Bus
Official Service Bus Documentation – 10gR3
http://docs.oracle.com/cd/E13159_01/osb/docs10gr3/userguide/modelingmessageflow.html

ATEAM – Specialized Blog about OFMW
http://www.ateam-oracle.com/wp-content/uploads/2013/09/OSBThreadingModelHTTPTransport_1.1.pdf

SHERRILL, JARED – Choosing Between Route, Service Callout and Publish – January 2013
http://jaredsoablogaz.blogspot.ca/2013/01/choosing-between-route-service-callout.html

KS - MY OSB 123 BLOGSPOT – June 2012
http://myosb123.blogspot.ca/2012/06/different-between-routing-service.html

MITRA, ABHIJIT – Midleware Blog – June 2013
http://soatransientprocess.blogspot.ca/2013/06/osb-route-publish-and-service-callout.html

JAN, VICTOR – Victor's Blog – June 2012
http://victor-jan.blogspot.ca/2012/06/osb-publish-routing-and-service-callout.html

COMMUNITY, ORACLE – Discussion about Routing, Callout and Publish – October 2010
https://community.oracle.com/thread/1774545

OVERTON, ANDY – Oracle Weblogic Work Managers – A Practical Overview
http://www.c2b2.co.uk/middleware-blog/oracle-weblogic-work-managers-a-practical-overview.php

SHAIK, KHASIM – Using Work Managers in Oracle Service Bus – April 2016
https://khassoablog.wordpress.com/2016/04/23/using-work-managers-in-oracle-service-bus/

WIJK, RENÉ VAN – Working with Work Managers – June 2016
http://middlewaresnippets.blogspot.ca/2015/06/working-with-work-managers.html