PYMMA

OpenESB

An Advanced Study on Leaky and Token bucket Algorithms with Respect to the JMS BC

Version 1.0
July 2020
Please send feedback to:
contact@pymma.com

About Pymma Consulting

Pymma Consulting, an EU company, founded in 1999, is a software system technical architecture bureau. Pymma provides expertise in the conception and execution of integration projects and programs. Working since its inception with top 500 companies and government entities, Pymma has evolved unique skills in the integration and services marketplace. Proactive in the open-source landscape, Pymma supports, develops and sponsors open-source projects such as OpenESB and is the architect of the integration platform *OpenESB Enterprise Edition*.

Pymma is based in London, UK, with satellite activity in France, Israel and Canada.

To reach Pymma: contact@pymma.com or www.pymma.com

About the authors

Pymma's development team has written this tutorial. Please feel free to send us your feedback, your questions, and comments.

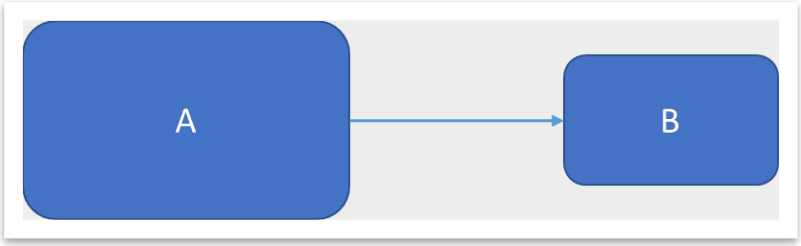
Contents

Introduction 4
The OpenESB Case 5
Token Bucket and Leaky Bucket algorithms 7
Token bucket configuration 7
Leaky bucket configuration 8
BPEL atomic process 8
More on JMS and atomic BPEL 10
Simulate Leaky Bucket with JMS Endpoint. 11
Conclusion 12

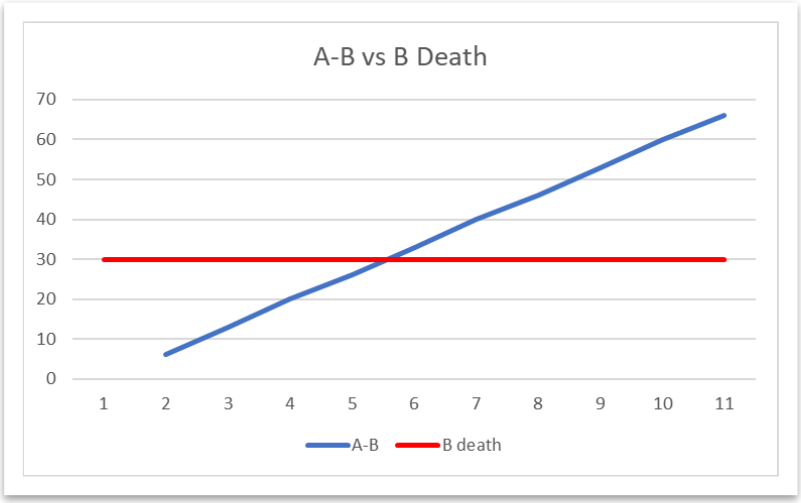
Introduction

This short tutorial explains how to limit the message flow between two services within OpenESB. It is a common requirement in integration when a fast and powerful consumer generates many invocations which intensively require services from a slower provider.

Before starting, let us define the context of this case.

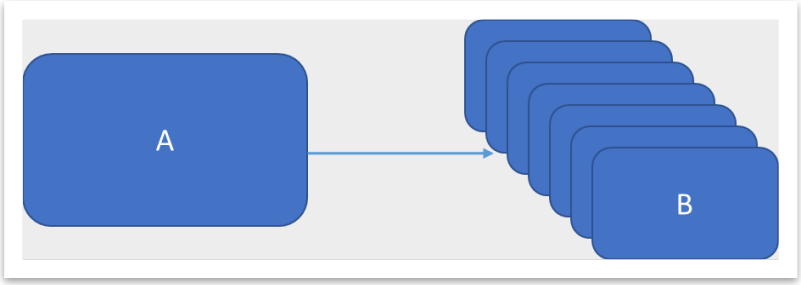


A service A invokes a provider B, and the invocation number is beyond B's capability. Let us suppose (these numbers being intentionally ridiculously low) that A generates ten invocations per second and B can process only three invocations per second. B shutdowns if more than 30 invocations are waiting to be processed

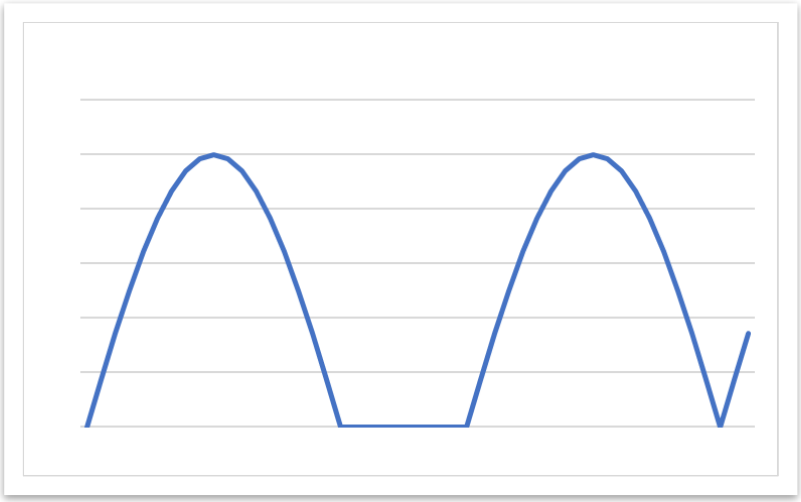


After 5 seconds B shuts down and the process stops.

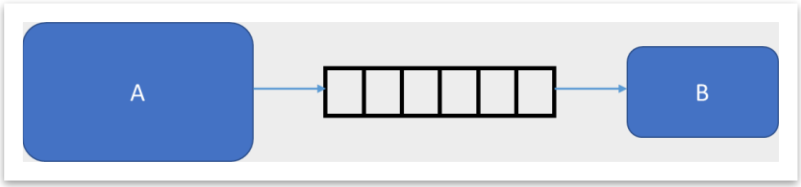
If A generates a constant flow of invocations, the only way to solve this issue is to duplicate B in a cluster and spread the invocation between B instances.



If, however, the message (invocation) flow generated by A is not constant and has a periodic form we can define a configuration that allows A and B to work together.



This is achieved by placing a FIFO (First In First Out) queue between A and B with the queue acting as a buffer between the components.

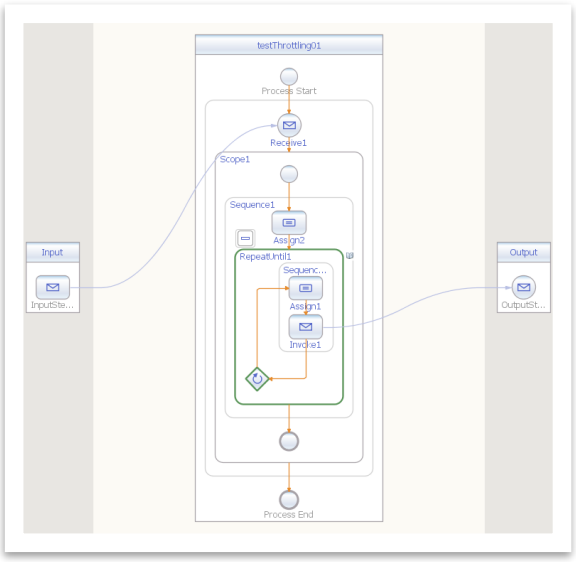


A generates a burst of invocations and puts them in the queue, B receives the invocations when it is able. If the delay between two bursts is long enough, A and B can work harmoniously.

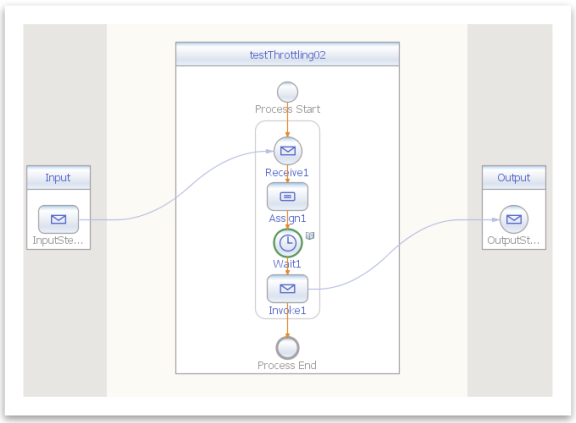
The OpenESB Case

Let us implement the queue within OpenESB.

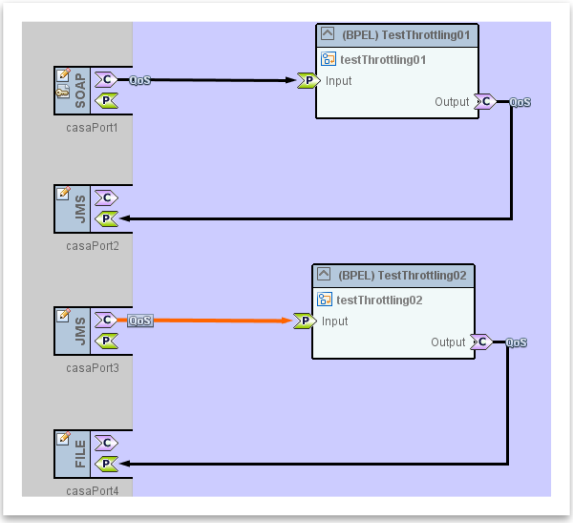
We have two services, named TestThrottling01 and TestThrottling02. The first service generates 100 messages for each invocation and puts them in a JMS Queue. The second services get the message from the queue, waits for a second, concatenates the message with the current time and then sends it to a file.



Service A



Service B



Composite Application

If we test this application, the file contains the data below:

Simple test 100 11:12:49.01+01:00	Simple test 11 11:12:49.30+01:00
Simple test 99 11:12:49.01+01:00	Simple test 10 11:12:49.30+01:00
Simple test 98 11:12:49.02+01:00	Simple test 9 11:12:49.30+01:00
Simple test 97 11:12:49.02+01:00	Simple test 8 11:12:49.30+01:00
Simple test 96 11:12:49.02+01:00	Simple test 7 11:12:49.30+01:00
Simple test 95 11:12:49.03+01:00	Simple test 6 11:12:49.31+01:00
Simple test 94 11:12:49.04+01:00	Simple test 5 11:12:49.31+01:00
Simple test 93 11:12:49.05+01:00	Simple test 4 11:12:49.31+01:00
Simple test 92 11:12:49.05+01:00	Simple test 3 11:12:49.31+01:00
Simple test 91 11:12:49.06+01:00	Simple test 2 11:12:49.31+01:00
Simple test 90 11:12:49.06+01:00	Simple test 1 11:12:49.32+01:00
Simple test 89 11:12:49.07+01:00	Simple test 0 11:12:49.32+01:00

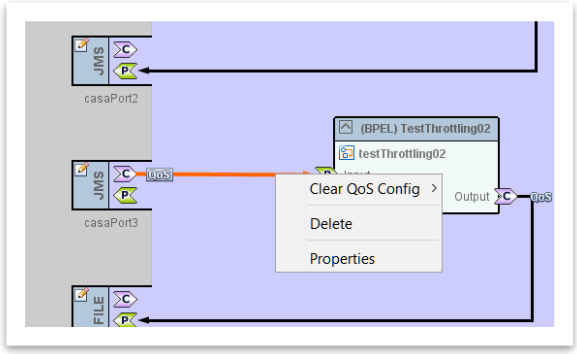
With this configuration, all the messages in the queue are read within a second.

Token Bucket and Leaky Bucket algorithms

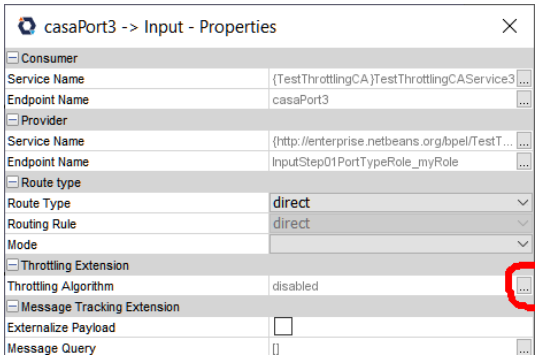
OpenESB implements two algorithms named “token bucket[1]” and “leaky bucket[2]”, to limit the number of messages between two services (Two endpoints). Both algorithms have the same purpose, but each one relies on different parameters. The token bucket algorithm defines the number of messages a service can process concurrently, and the leaky bucket algorithm limits the messages sent through the connection.

Token bucket configuration

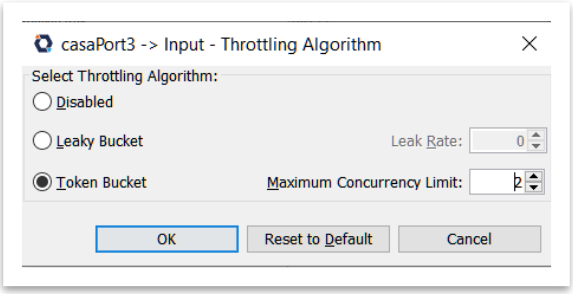
The bucket algorithms are both configured in the CASA Editor:



Configure a connection



Select the throttling Algorithm



Select the Token Bucket algorithm, set the Maximum Concurrency Limit to 2, click OK and then close the properties windows.

Build and deploy the CASA application and test it again with the text “Token Bucket 2 “.

Token bucket 2 100 15:28:09.85+01:00	Token bucket 2 11 15:28:54.02+01:00
Token bucket 2 99 15:28:09.86+01:00	Token bucket 2 10 15:28:55.02+01:00
Token bucket 2 97 15:28:10.86+01:00	Token bucket 2 9 15:28:55.02+01:00
Token bucket 2 98 15:28:10.86+01:00	Token bucket 2 8 15:28:56.02+01:00
Token bucket 2 96 15:28:11.87+01:00	Token bucket 2 7 15:28:56.02+01:00
Token bucket 2 95 15:28:11.87+01:00	Token bucket 2 6 15:28:57.02+01:00
Token bucket 2 94 15:28:12.87+01:00	Token bucket 2 5 15:28:57.03+01:00
Token bucket 2 93 15:28:12.87+01:00	Token bucket 2 4 15:28:58.03+01:00
Token bucket 2 91 15:28:13.88+01:00	Token bucket 2 3 15:28:58.03+01:00
Token bucket 2 92 15:28:13.88+01:00	Token bucket 2 2 15:28:59.03+01:00
Token bucket 2 90 15:28:14.88+01:00	Token bucket 2 1 15:28:59.03+01:00
Token bucket 2 89 15:28:14.88+01:00	Token bucket 2 0 15:29:00.03+01:00

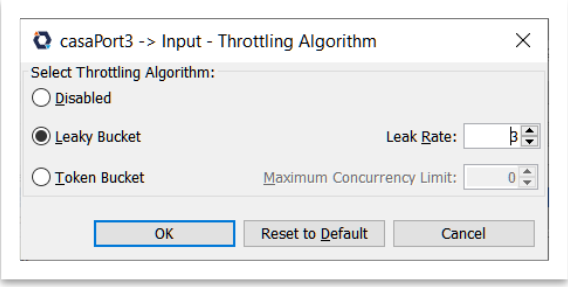
[1] https://en.wikipedia.org/wiki/Token_bucket

[2] https://en.wikipedia.org/wiki/Leaky_bucket

Note that the service provider processes two messages per second only.

Leaky bucket configuration

Change the throttling algorithm to leaky Bucket and set the leak rate to 3.



Build and deploy the Composite Application. Test it with the test "Leaky Bucket 3 ".

Leaky bucket 3 100 15:52:47.58+01:00	Leaky bucket 3 11 15:52:47.80+01:00
Leaky bucket 3 99 15:52:47.58+01:00	Leaky bucket 3 10 15:52:47.80+01:00
Leaky bucket 3 98 15:52:47.59+01:00	Leaky bucket 3 9 15:52:47.80+01:00
Leaky bucket 3 97 15:52:47.59+01:00	Leaky bucket 3 8 15:52:47.81+01:00
Leaky bucket 3 96 15:52:47.59+01:00	Leaky bucket 3 7 15:52:47.81+01:00
Leaky bucket 3 95 15:52:47.59+01:00	Leaky bucket 3 6 15:52:47.81+01:00
Leaky bucket 3 94 15:52:47.59+01:00	Leaky bucket 3 5 15:52:47.81+01:00
Leaky bucket 3 93 15:52:47.59+01:00	Leaky bucket 3 4 15:52:47.81+01:00
Leaky bucket 3 92 15:52:47.59+01:00	Leaky bucket 3 3 15:52:47.82+01:00
Leaky bucket 3 91 15:52:47.60+01:00	Leaky bucket 3 2 15:52:47.82+01:00
Leaky bucket 3 90 15:52:47.60+01:00	Leaky bucket 3 1 15:52:47.82+01:00
Leaky bucket 3 89 15:52:47.60+01:00	Leaky bucket 3 0 15:52:47.83+01:00

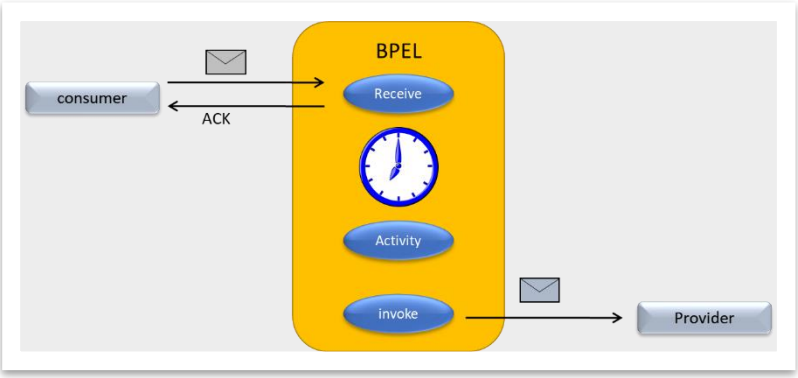
No that the leaky bucket algorithm does not provide the expected results. In order to understand why the leaky Bucket is not working, we need to understand what an atomic process is.

BPEL atomic process

OpenESB, as defined by the Java Business Integration architecture, requires that each endpoint that sends a message to a partner and the partner must compete a message exchange by returning an acknowledgement whatever the request type.

- If the message is a request-response request, the response is considered as an acknowledgement.
- If the process generates a fault, the fault is seen as an acknowledgement.
- If the message is a one-way request, the service provider MUST send back an acknowledgement to the service provider, even if there is no business meaning in the acknowledgement.

The OpenESB BPEL engine has a sophisticated design to optimise exchanges with a service consumer. When a message is received, the activity "Receive" immediately returns an acknowledgement and does not wait for the process end.

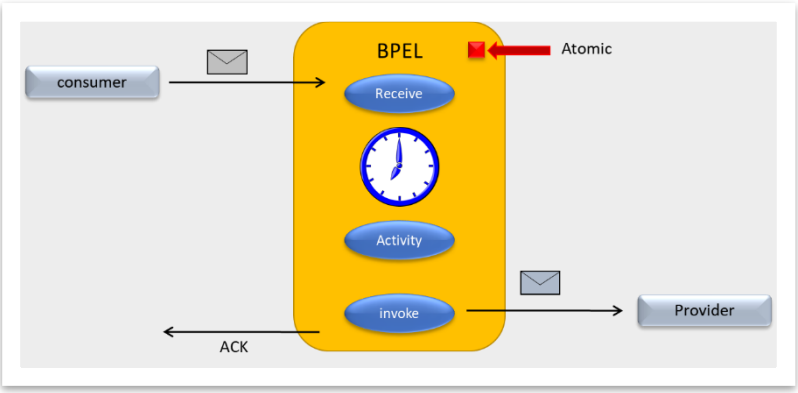


BPEL Acknowledgment

When we use the Leaky Bucket algorithm with the BPEL, the connection sends a new message when it receives an acknowledgement from the BPEL. When the Receive activity returns the ACK, the next message can arrive before the end of the process that uses the previous request (or message).

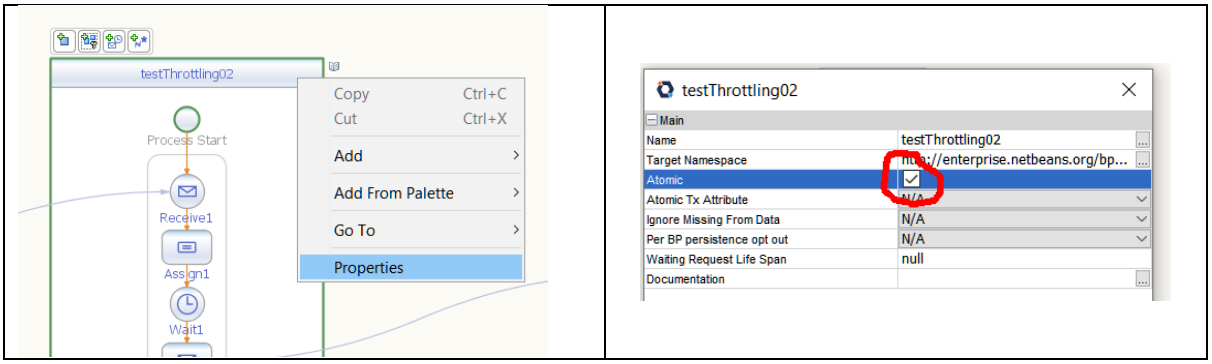
In our test, the JMS BC does not wait for the process to complete before sending a new message. This BPEL feature defeats the Leaky Bucket algorithm.

To avoid this unexpected result, we define the BPEL as “Atomic”. An atomic process returns an acknowledgement only at the end of the process.



Atomic BPEL Acknowledgment

To set up a BPEL as atomic, click right on the BPEL of the testThotlling02 project and select the Atomic property.



Build the BPEL project and build and deploy the Composite Application.

Replace the test input value with “Leaky bucket atomic 3” and run the test.

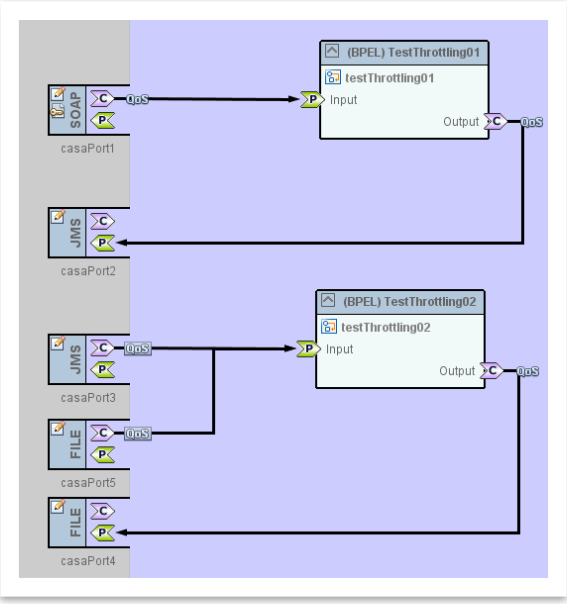
Leaky bucket atomic 3 100 22:36:49.43+01:00	Leaky bucket atomic 3 11 22:38:18.68+01:00
Leaky bucket atomic 3 99 22:36:50.43+01:00	Leaky bucket atomic 3 10 22:38:19.69+01:00
Leaky bucket atomic 3 98 22:36:51.43+01:00	Leaky bucket atomic 3 9 22:38:20.69+01:00
Leaky bucket atomic 3 97 22:36:52.43+01:00	Leaky bucket atomic 3 8 22:38:21.69+01:00
Leaky bucket atomic 3 96 22:36:53.44+01:00	Leaky bucket atomic 3 7 22:38:22.70+01:00
Leaky bucket atomic 3 95 22:36:54.44+01:00	Leaky bucket atomic 3 6 22:38:23.70+01:00
Leaky bucket atomic 3 94 22:36:55.44+01:00	Leaky bucket atomic 3 5 22:38:24.70+01:00
Leaky bucket atomic 3 93 22:36:56.45+01:00	Leaky bucket atomic 3 4 22:38:25.70+01:00
Leaky bucket atomic 3 92 22:36:57.45+01:00	Leaky bucket atomic 3 3 22:38:26.71+01:00
Leaky bucket atomic 3 91 22:36:58.45+01:00	Leaky bucket atomic 3 2 22:38:27.71+01:00
Leaky bucket atomic 3 90 22:36:59.45+01:00	Leaky bucket atomic 3 1 22:38:28.71+01:00
Leaky bucket atomic 3 89 22:37:00.46+01:00	Leaky bucket atomic 3 0 22:38:29.72+01:00

We notice that the result does not behave as intended. We expected to have three logs in the same second but instead get just one log. The following explains the reason for this result.

More on JMS and atomic BPEL

The JMS Binding component and the BPEL both support transactions between components. When a BPEL is define to be atomic, JMS BC and the BPEL share the transaction – with the JMS expecting an acknowledgement from the BPEL to send the next message - therefore only one transaction can be present at the connection (OpenESB bus) at a time. As a consequence, the Leaky Bucket algorithm cannot be applied in that case.

If we use another component such as the file BC, there is no shared transaction between the component and the BPEL and the Leaky Bucket algorithm can be used. Let’s test it by adding a new connector to feed the BPEL TestThrottling02.



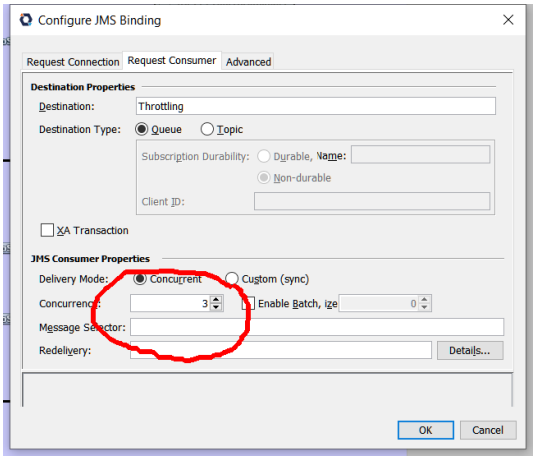
File Input data	File Output data
Leaky bucket atomic 3 with file BC	Leaky bucket atomic 3 with file BC 23:06:03.28+01:00
Leaky bucket atomic 3 with file BC	Leaky bucket atomic 3 with file BC 23:06:03.78+01:00
Leaky bucket atomic 3 with file BC	Leaky bucket atomic 3 with file BC 23:06:04.28+01:00
Leaky bucket atomic 3 with file BC	Leaky bucket atomic 3 with file BC 23:06:04.78+01:00
Leaky bucket atomic 3 with file BC	Leaky bucket atomic 3 with file BC 23:06:05.28+01:00
Leaky bucket atomic 3 with file BC	Leaky bucket atomic 3 with file BC 23:06:05.78+01:00
Leaky bucket atomic 3 with file BC	Leaky bucket atomic 3 with file BC 23:06:06.28+01:00
Leaky bucket atomic 3 with file BC	Leaky bucket atomic 3 with file BC 23:06:06.78+01:00
Leaky bucket atomic 3 with file BC	Leaky bucket atomic 3 with file BC 23:06:07.28+01:00
Leaky bucket atomic 3 with file BC	Leaky bucket atomic 3 with file BC 23:06:07.78+01:00
Leaky bucket atomic 3 with file BC	Leaky bucket atomic 3 with file BC 23:06:08.28+01:00
Leaky bucket atomic 3 with file BC	Leaky bucket atomic 3 with file BC 23:06:08.78+01:00
Leaky bucket atomic 3 with file BC	Leaky bucket atomic 3 with file BC 23:06:09.28+01:00
Leaky bucket atomic 3 with file BC	Leaky bucket atomic 3 with file BC 23:06:09.78+01:00
Leaky bucket atomic 3 with file BC	Leaky bucket atomic 3 with file BC 23:06:10.28+01:00
Leaky bucket atomic 3 with file BC	Leaky bucket atomic 3 with file BC 23:06:10.78+01:00
Leaky bucket atomic 3 with file BC	Leaky bucket atomic 3 with file BC 23:06:11.28+01:00
Leaky bucket atomic 3 with file BC	Leaky bucket atomic 3 with file BC 23:06:11.78+01:00
Leaky bucket atomic 3 with file BC	Leaky bucket atomic 3 with file BC 23:06:12.28+01:00
Leaky bucket atomic 3 with file BC	Leaky bucket atomic 3 with file BC 23:06:12.78+01:00
Leaky bucket atomic 3 with file BC	Leaky bucket atomic 3 with file BC 23:06:13.28+01:00
Leaky bucket atomic 3 with file BC	Leaky bucket atomic 3 with file BC 23:06:13.78+01:00
Leaky bucket atomic 3 with file BC	Leaky bucket atomic 3 with file BC 23:06:14.28+01:00
Leaky bucket atomic 3 with file BC	Leaky bucket atomic 3 with file BC 23:06:14.78+01:00
Leaky bucket atomic 3 with file BC	Leaky bucket atomic 3 with file BC 23:06:15.28+01:00
Leaky bucket atomic 3 with file BC	Leaky bucket atomic 3 with file BC 23:06:15.78+01:00
Leaky bucket atomic 3 with file BC	Leaky bucket atomic 3 with file BC 23:06:16.28+01:00
Leaky bucket atomic 3 with file BC	Leaky bucket atomic 3 with file BC 23:06:16.78+01:00
Leaky bucket atomic 3 with file BC	Leaky bucket atomic 3 with file BC 23:06:17.28+01:00
Leaky bucket atomic 3 with file BC	Leaky bucket atomic 3 with file BC 23:06:17.78+01:00
Leaky bucket atomic 3 with file BC	Leaky bucket atomic 3 with file BC 23:06:18.28+01:00
Leaky bucket atomic 3 with file BC	Leaky bucket atomic 3 with file BC 23:06:18.78+01:00

The leaky bucket algorithm works as expected when the data comes from the File Binding Component.

In conclusion, we see that when a JMS BC endpoint is connected with a BPEL endpoint, the Leaky Bucket algorithm is not applicable.

Simulate Leaky Bucket with JMS Endpoint.

The Leaky Bucket can be simulated with the JMS BC options by increasing the concurrency setting, which can be seen as an equivalent of the leak rate.



Replace the test input value with “Concurrency 3 simulate Leaky bucket atomic” and run the test.

Concurrency 3 simulate Leaky bucket atomic 100 00:10:49.53+01:00	Concurrency 3 simulate Leaky bucket atomic 11 00:12:18.77+01:00
Concurrency 3 simulate Leaky bucket atomic 99 00:10:50.54+01:00	Concurrency 3 simulate Leaky bucket atomic 10 00:12:19.78+01:00
Concurrency 3 simulate Leaky bucket atomic 98 00:10:51.54+01:00	Concurrency 3 simulate Leaky bucket atomic 9 00:12:20.78+01:00
Concurrency 3 simulate Leaky bucket atomic 97 00:10:52.54+01:00	Concurrency 3 simulate Leaky bucket atomic 8 00:12:21.78+01:00
Concurrency 3 simulate Leaky bucket atomic 96 00:10:53.55+01:00	Concurrency 3 simulate Leaky bucket atomic 7 00:12:22.78+01:00
Concurrency 3 simulate Leaky bucket atomic 95 00:10:54.55+01:00	Concurrency 3 simulate Leaky bucket atomic 6 00:12:23.79+01:00
Concurrency 3 simulate Leaky bucket atomic 94 00:10:55.55+01:00	Concurrency 3 simulate Leaky bucket atomic 5 00:12:24.79+01:00
Concurrency 3 simulate Leaky bucket atomic 93 00:10:56.56+01:00	Concurrency 3 simulate Leaky bucket atomic 4 00:12:25.79+01:00
Concurrency 3 simulate Leaky bucket atomic 92 00:10:57.56+01:00	Concurrency 3 simulate Leaky bucket atomic 3 00:12:26.80+01:00
Concurrency 3 simulate Leaky bucket atomic 91 00:10:58.56+01:00	Concurrency 3 simulate Leaky bucket atomic 2 00:12:27.80+01:00
Concurrency 3 simulate Leaky bucket atomic 90 00:10:59.56+01:00	Concurrency 3 simulate Leaky bucket atomic 1 00:12:28.80+01:00
Concurrency 3 simulate Leaky bucket atomic 89 00:11:00.57+01:00	Concurrency 3 simulate Leaky Bucket atomic 0 00:12:29.81+01:00

The result is identical to having a leaky bucket set to three.

Conclusion

OpenESB implements Leaky Bucket and token bucket algorithms to limit message flows between endpoints. Nevertheless, the leaky Bucket does not work when JMS consumer endpoint is connected with a BPEL provider. Using concurrency number with a BPEL in the atomic mode can replace the leaky bucket algorithm