Micro-Benchmarking BPMN 2.0 Workflow Management Systems with Workflow Patterns

Marigianna Skouradaki*,[1], Vincenzo Ferme*,[2] Cesare Pautasso[2], Frank Leymann[1], and André van Hoorn[3]

[1] Institute of Architecture of Application Systems (IAAS), University of Stuttgart, Germany
[2] Faculty of Informatics, University of Lugano (USI), Switzerland
[3] Institute of Software Technology (ISTE), University of Stuttgart, Germany

Abstract. Although Workflow Management Systems (WfMSs) are a key component in workflow technology, research work for assessing and comparing their performance is limited. This work proposes the first micro-benchmark for WfMSs that can execute BPMN 2.0 workflows. To this end, we focus on studying the performance impact of well-known workflow patterns expressed in BPMN 2.0 with respect to three open source WfMSs. We executed all the experiments under a reliable environment and produced a set of meaningful metrics. This paper contributes to the area of workflow technology by defining building blocks for more complex BPMN 2.0 WfMS benchmarks. The results have shown bottlenecks on architectural design decisions, resource utilization, and limits on the load a WfMS can sustain, especially for the cases of complex and parallel structures. Experiments on a mix of workflow patterns indicated that there are no unexpected performance side effects when executing different workflow patterns concurrently, although the duration of the individual workflows that comprised the mix was increased.

Keywords: Benchmarking · Micro-benchmark · BPMN 2.0 · Workflow Patterns · Workflow Management Systems

1 Introduction

Despite the current trend of utilizing BPMN 2.0 as a common modeling and execution language for business processes [17], there are no means to measure and compare the performance of Workflow Management Systems (WfMSs). However, the need for a benchmark is regularly affirmed by the literature [21]. Before proceeding with the development of a standard complex benchmark one needs to understand the individual characteristics of the workload components. As a first approximation, the workload of a WfMS benchmark mainly consists of the workflow models to be executed and the frequency of their execution. However,

* Corresponding authors

we are currently lacking any information regarding the impact of individual BPMN 2.0 constructs on the performance of a WfMS. Micro-benchmarks aim to stress fundamental concepts of a system such as single operations or target narrow aspects of more complex systems. Therefore, we consider a micro-benchmark the appropriate tool for our goal as it targets the specific performance evaluation of atomic operations [20]. Workflow patterns can be seen as generic, recurring concepts and constructs that should be implemented by any workflow language [19]. In our context and given the complexity of the BPMN 2.0 language, we focus on the basic control-flow workflow patterns that apply on the core of the BPMN 2.0 language. Targeting to the simple workflow patterns we follow the assumption that these are the simplest and more frequent atomic operations that a WfMS would use. The main contribution of this paper is thus the first micro-benchmark for BPMN 2.0 WfMSs based on the following workflow patterns: sequence flow, exclusive choice and simple merge, explicit termination, parallel split and synchronization, as well as arbitrary cycle. Similar efforts for different systems [12,1] or languages (e.g., WS-BPEL [2]) have revealed fundamental bottlenecks in the corresponding engines, and have therefore been proven beneficiary to improve the tested systems. The main goal of this work is to enable further research in the performance engineering of the BPMN 2.0 WfMSs, by examining three state-of-the-art open-source WfMSs and providing the first insight on which BPMN 2.0 language factors impact the WfMSs performance.

This work focuses on studying the performance of the Process Navigator, a core WfMS component responsible for driving the execution of the tasks of each workflow instance with respect to the semantics of BPMN 2.0. More particularly, the research questions that our work aims to answer are: *i)* what is the impact of individual or a mix of workflow patterns on the performance of each one of the benchmarked BPMN 2.0 WfMSs? *ii)* are there performance bottlenecks in the selected WfMSs? We consider it important to understand the performance behaviour of the WfMS fundamental components before proceeding to more complex performance measurements that will also include external interactions. To do so, BPMN 2.0 workflows that implement the selected workflow patterns are given as input to two sets of experiments. The first set of experiments aims to execute a large load of workflow instances for each workflow pattern and investigate the behavior of the WfMSs. The second set of experiments studies the behavior of the WfMSs when they execute a uniformly distributed mix of all workflow patterns. For all the experiments we have calculated the throughput, the process execution time, and resource utilization from raw measurements, obtained using a reliable benchmarking environment presented in previous work [6]. The results revealed bottlenecks on architectural design decisions, wasteful resource utilization, and load limits for specific workflow patterns.

To summarize, the original, scientific contributions of this work are: *i)* providing the first micro-benchmark for BPMN 2.0 WfMS; *ii)* analyzing the effect of selected core BPMN 2.0 language constructs on the WfMS performance; *iii)* defining meaningful candidate constructs for BPMN 2.0 complex benchmarks; *iv)* running experiments on a reliable environment; *v)* conducting a thorough analysis on the results of the performance evaluation of the selected WfMSs to reveal performance bottlenecks. The remainder of this paper is structured as follows: Sec. 2 presents the workload mix of the experiments and Sec. 3 explains the setup of the benchmark environment and of the experiments. The analysis of the results as well as possible threats to validity are discussed in Sec. 4. Sec. 5 overviews the related work and Sec. 6 concludes and presents our plans for future work. Moreover, supplementary material of the raw data and aggregated metrics can be found at: `http://benchflow.inf.usi.ch/results/2015/caise-microbenchmark.tgz`

2 Experiments Workload Mix

The workflows making up the workload of the micro-benchmark are designed to comply with these constraints: *i)* Maximize the simplicity of the model expressing the workflow pattern; *ii)* Omit the interactions with external systems. All tasks are implemented as script tasks, while human tasks and Web service invocations are excluded. This way we stress mainly the Process Navigator, since script tasks are fully automated and only use embedded application logic that is co-located with the engine. *iii)* Most script tasks are empty. Only the ones required to implement the workflow pattern semantics contain the minimal amount of code and produce the minimum amount of data to do so. *iv)* Define equal probability of passing the control flow to any outgoing branch of the gateways. *v)* As it is recommended by the BPMN 2.0 Standard [9, p. 90], the exclusive choice is combined with the simple merge ([EXC]) workflow pattern and the parallel split is combined with the synchronization ([PAR]) workflow pattern.

In the scope of this work, we focus on the basic control flow and structural workflow patterns that can be expressed by BPMN 2.0 [22]. We have excluded the deferred choice, multiple instances without synchronization, and synchronization merge because the BPMN 2.0 elements that are used to implement them are not widely used in practice [15]. The workflows designed for our experiments are shown in Fig. 1. In the rest of this section we present the workflow models that comprise the workload mix of the micro-benchmark and define our hypotheses concerning their expected performance.

Sequence Flow [SEQ] - This workflow consists of two sequential empty script tasks. Since this is the simplest structure a workflow model may have, we expect that the execution times should be similar and stable on all three WfMSs [HYP1]. **Exclusive Choice and Simple Merge [EXC]** - The first script task randomly generates with uniform probability the numbers 1 or 2, according to which the upper or the lower branch is chosen. In both cases an empty script task is executed. The evaluation of the condition of the exclusive choice is expected to have an impact on the performance [HYP2]. **Parallel Split and Synchronization [PAR]** - This workflow executes in parallel two empty script tasks. As parallelism generally demands more CPU power we expect this to reflect on the performance measurements [HYP3]. **Explicit Termination Pattern [EXT]** - This workflow executes two branches concurrently and according to the BPMN 2.0 language semantics when one of these branches ends it will also

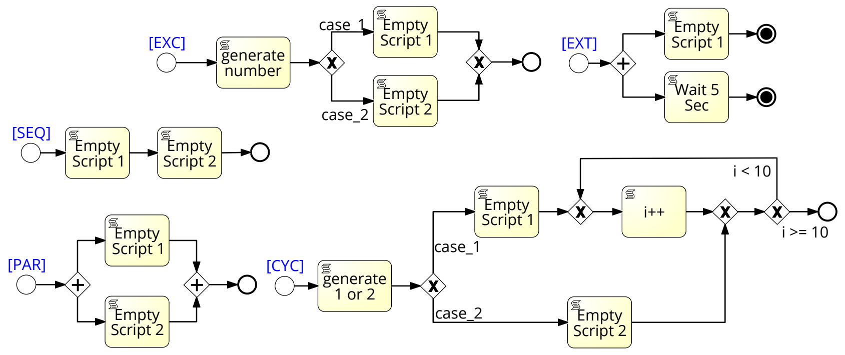

Fig. 1. Defined Patterns for the Workload Mix

terminate the rest of the executing branches and the overall workflow instance will be completed successfully. The "Empty Script 1" is an empty script task, while the "Wait 5 Sec" task waits for five seconds. The value of five seconds was chosen to guarantee that the lower branch will be slower than the upper one. As the "Empty Script 1" is the fastest, we expect that the workflow will be completed with the completion of the path containing the "Empty Script 1", and then the terminate event will interrupt the "Wait 5 Sec" task. The workflows in [PAR] and [EXT] have very similar structure, although they represent different workflow patterns. We expect that the concurrent execution of tasks should demonstrate similar performance behavior [HYP4]. **Arbitrary Cycle [CYC] -** Cycles are not expressed through any specific BPMN 2.0 construct but through a combination of exclusive gateways that form a cyclic structure that has at least two entries or two exits. The [CYC] is implemented with two entry points at the second and third exclusive gateways and starts by a script task that randomly generates the integer number $x = 1$ or $x = 2$ and initializes a variable $i = 0$. With respect to the value of the x variable the upper or the lower branches are followed (cf. [EXC]). The lower branch executes the "Empty Script 2". The upper branch executes an "Empty Script 1" and then increases the variable i. This path will be followed until the variable $i == 10$. To have a different but deterministic behavior of the branches we have implemented the "Empty Script 2" to assign $i = 5$. In this case the cycle will be repeated fewer times, until the variable $i == 10$. In terms of size [CYC] represents a slightly more complex structure than the other structures defined under the scope of this work. This might contribute towards revealing performance bottlenecks due to the usage of nested exclusive gateways, or sequential decision points [HYP5].

3 Experiments Setup

3.1 WfMS Configuration

The complex architecture of the WfMS introduces a set of challenges to be addressed by the design of the benchmark: *i)* controlling the initial condition of the experiments is difficult due to the distributed nature of the WfMS; *ii)* variable combinations of configuration parameters may have a significant impact on the performance; *iii)* a standard API to interact with the WfMSs and to access the execution data is not available; and *iv)* the asynchronous execution of processes introduces additional challenges for handling the performance data collection.In previous work we have therefore introduced the BenchFlow framework [6], that addresses the above challenges and provides a complete solution for benchmarking WfMS. Moreover, BenchFlow is compliant with the main requirements of a benchmark: portability, scalability, simplicity, vendor neutrality, repeatability, and efficiency [10].

To automate the configuration and deployment of the WfMS before the execution of benchmark, BenchFlow [7] uses the lightweight containerization technology Docker [13]. The execution of the benchmark is driven by Faban [4], a framework for performance workload creation and execution, used in industry benchmarks such as SPECjEnterprise2010 and SPECjms [18]. The load drivers provide the infrastructure needed to issue the load to the WfMS. To the rest of this paper the load drivers are also referred to as instance producers, as they are sending the requests for the workflow instance initiation. In order to ensure the reproducibility of the benchmark results one needs to explicitly describe the benchmark environment and configurations. Thus, in the following we provide this information.

The BenchFlow framework is used in this work for benchmarking three open-source WfMSs: WfMS A, WfMS B and Camunda 7.3.0 [3][4]. These WfMSs are widely used in industry and have a large user community according to the vendors' websites. The selected engines are also already tested against conformance to the BPMN 2.0 standard [8]. This makes them a suitable starting point for our defined workload and ensures the possibility to execute more diverse, complex workload in future versions of the benchmark. Moreover, WfMS B and Camunda are provided in Docker containers with vendor-suggested configurations, a fact that improves the reproducibility of our benchmark.

We benchmark these WfMSs on top of Ubuntu 14.04.01, using Oracle Java Server 7u79. WfMS A and Camunda were deployed on top of Apache Tomcat 7.0.62, while WfMS B was deployed on top of Wildfly 8.1.0.Final. All these WfMS utilise a MySQL Community Server 5.6.26 as Database Management System (DBMS), installed in a Docker container[5]. For WfMS B and Camunda[6] we have used the official Docker images, and we have followed the vendor-suggested configurations. We configured WfMS A as suggested from the vendor's website, and we deployed it using the most popular Docker image. We have updated the dependencies on the operating system and Java to be identical to the other two WfMS, to reduce possible discrepancies introduced by using different versions.

[4] At the time of publication some of the vendors we contacted did not explicitly agree to be named when presenting the results of the benchmark

[5] `https://hub.docker.com/_/mysql/`

[6] `https://hub.docker.com/r/camunda/camunda-bpm-platform/`

Every WfMS was given a maximum Java heap size of 32 GB, and the connection to the DBMS uses the MySQL Connector/J 5.1.33 with 10 as value for initial thread pool size, 100 as maximum number of connections, and 10 minimum idle connections. For WfMS A, we enabled the "Async executor" as suggested on the vendor's website. The other configurations are as provided in the mentioned Docker images. In particular, all the WfMSs log a complete history of the workflows execution to the database (i.e., details on the execution of the workflow instances as well as all the initial business process models). The containers are run by using the *host* network option of Docker. This option enables the containers to directly rely on the network interfaces of the physical machine hosting the Docker Engine, and has been proven not to add performance overhead in the network communications [5].

The benchmark environment is distributed on three servers: one for Faban that executes the instance producers, one for the WfMS, and one for the database of the WfMS that maintains the execution information of the workflows. All the servers use Ubuntu 14.04.3 LTS (GNU/Linux 3.13.0-33-generic x86_64) as operating system and the Docker Engine version 1.8.2. The WfMS is deployed on a 12 CPU Cores at 800Mhz, 64GB of RAM. In this way we ensure that the machine where we deploy the instance producer (64 CPU Cores at 1400MHz, 128GB of RAM) can issue sufficient load to the WfMS and the database (64 CPU Cores at 2300MHz, 128GB of RAM) and handle the requests from the WfMS. For the interaction of the WfMS with the DBMS and of the Instance Producers (IP), with the WfMS we use two different dedicated networks of 10Gbit/s. Since the BenchFlow environment guaranties a repeatable benchmark, any test that follows the suggested configuration should reproduce the same results.

3.2 Experiments Methodology

We define two scenarios for the micro-benchmark. *Scenario 1* issues a large load to the WfMSs and investigates their behavior for individual patterns ([SEQ], [EXC], [EXT], [PAR], [CYC]). In some cases a WfMS does not sustain the predefined load. Then we re-execute the experiments with a lower load and observe the WfMS behavior for this execution. *Scenario 2* studies the performance behavior when different workflow patterns run concurrently ([MIX]). More particularly, we test if the performance of a workflow pattern is affected when it runs concurrently with other types of workflow patterns. For this purpose, we benchmark a mix of all the workflow patterns distributed uniformly (i.e., 20% of instances for each workflow pattern). The load of this experiment corresponds to the large load defined in Scenario 1. Both scenarios are executed three times for each WfMS to verify that the behavior is similar among the runs. The maximum standard deviation allowed among the repetitions was set to 5%, but it was approximately 3.5% on average.

For each benchmark run we collect the raw execution data for each workflow instance execution, and with these we calculate meaningful statistical data on the workflow instance duration ($WIDuration$), which is defined as the time difference between the start and the completion of a workflow instance; the resource utilization, in terms of *CPU utilization* and *Memory utilization*; the absolute number of the executed workflow instances by the WfMS per benchmark run ($\#WorkflowInstances(wi)$); and the number of executed workflow instances per time unit ($Throughput = \frac{\#WorkflowInstances(wi)}{Time(sec)}$) [11].

The load function (T_l) we use consists of an experiment duration time of 10 minutes (cf., Experiment Duration of Table 1) with 30 seconds of ramp-up period (T_r). During the experiment the instance producers (u) perform up to 1 request per second (req/sec) when the response time of the WfMS is low, and comprises the variable for which we execute the performance test. Whereas a load time of 10 minutes might not be representative of a real execution time, we consider it adequate for the micro-benchmark, as bottlenecks are already revealed within this time period (cf. Sec. 4). Given the used load function and the workload mix with only one workflow, the expected number of started workflows (S) is computed as $S = \sum_{j=1}^{u-1} \frac{T_r}{u} rj + (T_l - T_r)ru$ where T_r is the ramp-up period, T_l is the load time, and r is the user requests/sec. The actual number can be less or equal than the expected one, since it depends on the resource availability of the servers where the instance producers are deployed, and the response time of the WfMS. We have also set a connection time-out period T_o of 20 seconds. According to our experiments it is an adequate time to indicate that the WfMS cannot handle the issued load. At the end of the run we are collecting the data, and analyze them to compute the relevant performance metrics. For all the data that are collected for the statistical analysis we have removed the first one minute ($2 * T_r$). This way we make sure that the analyzed results correspond to a stable state of the WfMS.

4 Evaluation

4.1 Results

For each workflow pattern and each WfMS, we show the duration (milliseconds, Fig. 2(a)), the CPU utilization (%, Fig. 2(b)), and the mean amount of RAM that was allocated by the engine (MB, Fig. 3(a)). Table 1 shows the statistics [14] of the duration computed for each workflow pattern and for every WfMS. The data provided in Table 1 correspond to the means of measurements obtained under the maximum load each WfMS could sustain, shown in terms of the number of concurrent instance producers. In some cases the WfMS could not handle the maximum load ($1,500$ concurrent instance producers), and we had to reduce the number of concurrent instance producers. These cases and the resulting data are discussed in detail in the following subsections. For every experiment, the total number of completed workflow instance requests from all WfMSs is listed in Table 1 in column $\#Workflow(wi)$. We also include the total duration of each experiment (in seconds) and the average throughput in terms of workflow instances per second.

Similar statistics have been respectively calculated for CPU and RAM usage but they are omitted for space reasons, and they are provided with the supple-

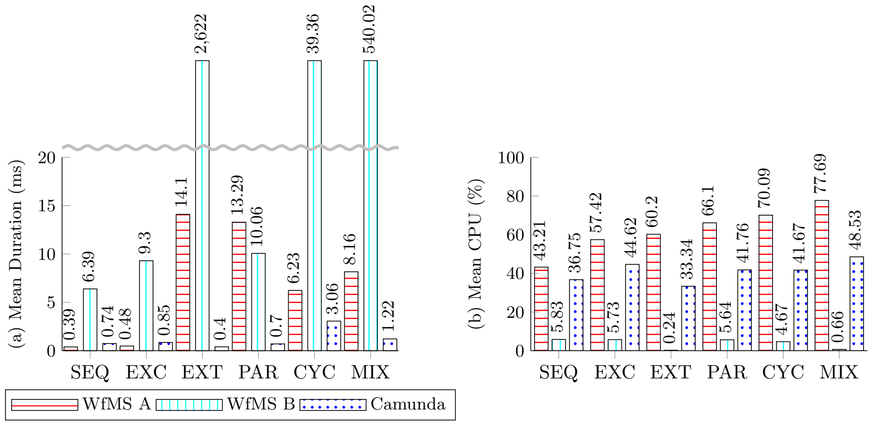

Fig. 2. (a) Mean Duration (ms) per workflow pattern and (b) Mean CPU (%) Usage per workflow pattern

mentary material. The behaviour of all the WfMSs is discussed thoroughly in Sec. 4.3.

4.2 Results Analysis

Sequence Flow Pattern [SEQ] The [SEQ] workflow pattern lasted on average 0.39 ms for WfMS A, 6.39 ms for WfMS B and 0.74 ms for Camunda. The short duration of this workflow pattern justifies the low mean CPU usage which is 43.21% for WfMS A, 5.83% for WfMS B and 36.75% for Camunda. WfMS B also has a very low average throughput of 63.31 wi/s while for the other two WfMS the average throughput is similar. Concerning the memory utilization under the maximum load WfMS A needed in average 12, 074, WfMS B 2, 936 and Camunda 807.81 MB of RAM respectively. As observed from the Table 1 [SEQ] is the workflow pattern with the highest throughput for all the WfMS under test.

Exclusive Choice & Simple Merge Patterns [EXC] Before proceeding to the results analysis of the [EXC], we should consider that the first script task of the workflow pattern generates a random integer, which is given as an input to the very simple evaluation condition of the exclusive choice gateway. This was expected to have some impact on the performance. However, Fig. 2(a) shows that the duration times are not notably affected as the values are close to those of [SEQ]. More particularly, we have a mean of 0.48 ms for WfMS A, 9.30 ms for WfMS B and 0.85 ms for Camunda. Concerning the CPU and RAM utilization, we see a slight increase with respect to the [SEQ]. WfMS A uses an average of 57.42% CPU and 12, 215 MB RAM for executing 775, 455 workflow instances in 540 sec, WfMS B takes approximately the same amount of time (562 sec) to execute 27, 805 workflow instances. For this, it utilizes a mean of 5.73% CPU and 2, 976.37 MB of RAM, and Camunda 43.21% of CPU and 824.96 MB of RAM for executing 765, 274 workflow instances in 540 sec.

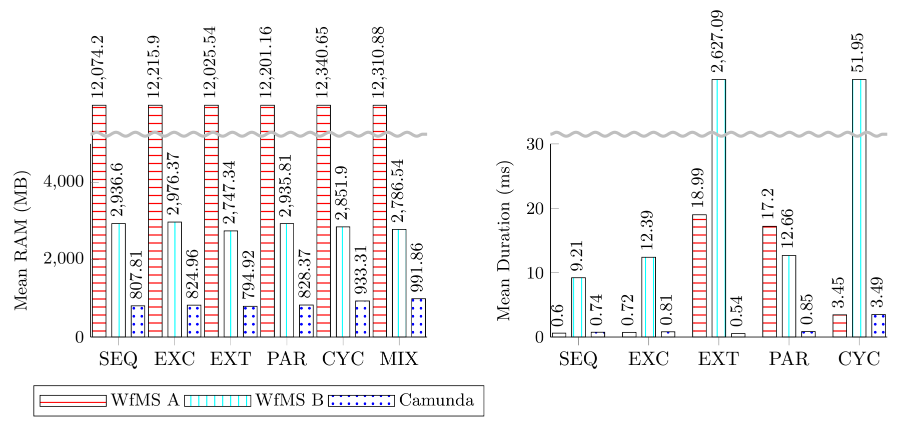

Fig. 3. (a) Mean RAM (MB) Usage per workflow pattern and (b) Mean Duration (ms) per workflow pattern in [MIX]

Explicit Termination Pattern [EXT] As discussed in Sec. 2 the [EXT] executes concurrently an empty script and a script that implements a five seconds wait. According to the BPMN 2.0 execution semantics, the branch of the [EXT] that finishes first terminates the rest of the workflow's running branches. We have therefore designed the model considering that the fastest branch (empty script) will complete first, and stop the slow script on the other branch when the terminate end event following the empty script is activated. This was the case for WfMS A and Camunda, which executed the workflow patterns in an average of 14.11 ms and 0.4 ms respectively. The resource utilization of these two WfMSs also increases in this workflow pattern, i.e., we have 60.20% mean CPU usage and $12,025$ MB mean RAM usage for WfMS A and 33.34% mean CPU usage and 794.92 MB mean RAM usage for Camunda. We can already see an interesting difference on the performance of the two WfMS as [EXT] constitutes the slowest workflow pattern for WfMS A and the fastest for Camunda.

As seen in Fig. 2(a), WfMS B has very high duration results for this workflow pattern. We have investigated this matter in more detail and we have observed that over the executions WfMS B chooses the sequential execution of each path with an average percentage of 52.23% for following the waiting script first and 47.77% for following first the empty script. Since the waiting script takes five seconds to complete, every time it is chosen for execution it adds a five seconds overhead, and thus the average duration time is so high. This alternate execution of the two branches also explains the rest of the statistics. For example, we observe a very high standard deviation of 2500.44 that indicates that there is a very large spread of the values around the mean duration. Concerning the resource utilization we can observe a very low average usage of CPU at 0.24% and a mean RAM usage similar to the rest of the workflow patterns at $2,747.34$ MB. In Sec. 4.3 we attempt to give an explanation of this behavior for WfMS B.

Table 1. Workflow Instance Duration and Experiment Execution Statistics

		Workflow Instance Execution Duration Statistics (ms)								**Experiment Execution Statistics**			
		Mean & CI T ($\alpha = 0.95$)	**Mode**	**Min**	**Max**	**Sd**	**Q1**	**Q2**	**Q3**	**Max Load (IPs)**	**#Workflow (wi)**	**Experiment Duration (s)**	**Throughput (wi/s)**
SEQ	*A.*	0.39 ± 0.01	0	0	561	1.70	0	0	1	1,500	781,736	540	1,447.66
	B.	6.39 ± 0.43	6	4	82	1.21	6	6	7	1,500	35,516	561	63.31
	C.	0.74 ± 0.01	1	0	682	2.29	0	1	1	1,500	786,664	540	1,456.79
EXC	*A.*	0.48 ± 0.01	0	0	485	2.07	0	0	1	1,500	775,455	540	1,436.03
	B.	9.30 ± 0.05	9	6	131	2.11	9	9	10	1,500	27,805	567	49.04
	C.	0.85 ± 0.01	1	0	627	2.51	0	1	1	1,500	765,274	540	1,417.17
EXT	*A.*	14.10 ± 0.06	10	5	858	13.45	10	11	14	1,500	770,229	540	1,426.35
	B.	$2,622.00 \pm 237.68$	11	8	5,047	2,500.44	13	5,012	5,016	1,500	1,703	4,498	0.38
	C.	0.40 ± 0.01	0	0	74	1.03	0	0	1	1,500	784,614	539	1,455.68
PAR	*A.*	13.29 ± 0.06	8	4	456	11.99	9	10	13	1,500	772,013	540	1,429.65
	B.	10.06 ± 0.06	10	7	145	2.22	9	10	10	1,500	27,718	567	48.89
	C.	0.70 ± 0.01	1	0	691	2.10	0	1	1	1,500	773,883	540	1,433.12
CYC	*A.*	6.23 ± 0.13	2	0	478	18.68	1	2	3	800	347,770	540	644.02
	B.	39.36 ± 0.40	50	25	146	9.52	30	43	47	1,500	8,695	646	13.46
	C.	3.06 ± 0.04	2	0	353	4.43	2	2	3	600	177,770	542	327.99
MIX	*A.*	8.16 ± 0.07	0	0	663	14.65	1	2	12	1,500	758,659	541	1,402.33
	B.	540.02 ± 122.3	11	6	5,195	1,525.27	10	12	38	1,500	2,392	1,343	1.78
	C.	1.22 ± 0.02	0	0	434	4.21	0	1	1	1,500	575,210	542	1,061.27

WfMS A: *A.*, **WfMS B:** *B.*, **Camunda:** *C.*

Parallel Split & Synchronization Patterns [PAR] The [PAR] executes two empty scripts concurrently. For WfMS A and WfMS B we observe an increase in the duration times to 13.30 ms for WfMS A and 10.07 ms for WfMS B. Camunda handles parallelism very fast, with a mean duration of 0.71 ms. Although WfMS B seems faster by looking the duration results, we should take into consideration that it has a total execution of $27,718$ workflow instances in 567 sec while WfMS A executed $772,013$ workflow instances in 540 sec. Moreover, it is noteworthy that WfMS A has a standard deviation of 11.99 which indicates that there were executions for which the parallelism introduced more overhead in duration than the average value. WfMS B has a 5.64% mean CPU and $2,935.81$ MB mean RAM usage and Camunda has a 41.67% mean CPU and 828.37 MB mean RAM usage. For both WfMSs these values are in the same range as the values resulted for the execution of the other workflow patterns. WfMS A utilizes in average 66.10% of CPU and $12,201.16$ MB of RAM. For WfMS A the values of utilized resources are relatively higher than these obtained from the other workflow patterns.

Arbitrary Cycle Pattern [CYC] The performance of the [CYC] workflow pattern cannot be directly compared to the other workflow patterns, because it contains a higher number of language elements and demonstrates a more complex structure. The [CYC] is also expected to have some extra overhead because of the number generation and the script that increases the value of the variable. Finally, the duration of this workflow pattern is dependent on the generated number, as in the one case it executes 10 cycles while in the other it will execute 5 cycles.

During the execution of [CYC], Camunda showed connection timeout errors for a load greater than 600 instance producers. For this reason, we had to reduce the load to 600 instance producers for testing the other two WfMS. The load for the results shown in Figures 2(a), 2(b) and 3(a) for this workflow pattern is thus 600 instance producers. Table 1 shows the results for the maximum load each WfMS could sustain: 800 instance producers for WfMS A, 1500 instance producers for WfMS B and 600 for Camunda,. As expected, the mean [CYC] execution duration is higher than the other workflow patterns. WfMS A has a mean duration of 6.23 ms and Camunda a marginally bigger mean duration of 3.06 ms for this number of instance producers. WfMS B has a mean duration of 39.36 ms for approximately 600 instance producers.

Concerning the resource utilization, WfMS B and Camunda remain stable to the same range of mean CPU usage (4.67% for WfMS B and 41.67% for Camunda) as with the other workflow patterns. WfMS B remains on the same range of mean RAM usage (2, 851.9 MB), while we observe an increase for Camunda to an average of 933.31 MB. Concerning WfMS A's resource utilization, we observe a tendency to increase in comparison with the rest of the workflow patterns. For approximately 600 instance producers, WfMS A uses in average 70.09% of CPU and 12, 201.16 MB RAM. We consider it also interesting to report how the results evolved for WfMS A and WfMS B when we increased the load to the maximum (1500 and 800 instance producers respectively). Then, we observe WfMS A doubling the mean duration time from 2.92 ms to 6.23 ms. The CPU is also more stressed reaching 83.93% while the mean memory usage is only slightly increased to 12, 429.67 MB. WfMS B remains in the same range of the previous values with scarcely any increase to its performance. It uses in average 4.59% of CPU and 2, 897.72 MB of RAM. This is because its response time increases while adding instance producers.

Mix [MIX] By a quick overview of the [MIX] statistics, one could conclude that they express the mean duration times of the individual workflow patterns shown in Fig. 3(b). The throughput of the mix, is also a bit smaller for all the WfMSs, although WfMS A keeps it on the same range as the previous values at 1, 402.33 wi/s. In Fig. 3(b) we can observe the separate duration times of the workflow patterns for the case that they are executed in the uniformly distributed mix. As seen in Fig. 3(b), all workflow patterns have a slight increase in their duration times with respect to the execution as a single workflow pattern.

4.3 Discussion

As reported in Sec. 4.2, the BPMN version of WfMS B presents some peculiarity in its behaviour. This was also noticed by Bianculli et al. [2] on their performance measurements on the WS-BPEL version of the WfMS B. According to the WfMS B documentation the REST API calls to the execution server will block until the process instance has completed its execution. We observed the effects of this synchronous API in our experiments. All instance producers send requests to the WfMS using the REST API with a think time of 1 sec. The instance producers need to wait for the completion of their previous request before sending a new one, but in the case of WfMS B the clients that are waiting for the entire execution of the workflow instance to finish introduce a high overhead. This overhead causes a delay that burdens the WfMS's performance. In order to investigate this further we have executed a scalability test to analyze the WfMS behavior under different load intensity levels. The goal of this experiment was to examine, whether by increasing significantly the number of instance producers we could achieve a number of executed workflow instances that can be more comparable to those of WfMS A and Camunda. We executed the experiment for 500, 1000, 1500, and 2000 instance producers and observed a mean response time of 7.15, 15.19, 22.58 and 30.89 seconds respectively, while the throughput remained stable to an average of 62.23 workflow instances per second. These data basically show that *i)* it is pointless to increase the number of instance producers and target to the execution of more workflow instances; and that *ii)* the fact that WfMS B is the only WfMS of the three under test using a synchronous REST API does not impact the comparability of the measurement.

Another issue discussed concerning WfMS B was the inconsistent execution behaviour of the [EXT]. Although the expected execution of [EXT] is that when the path with the empty script ends the execution of the path with the 5 sec script will also be terminated, we have observed many executions with the opposite behavior. The path with the wait script was executing "first" and then, after 5 sec, followed the execution of the empty script. In this case, the end event that corresponded to the empty task was never executed. This behavior of WfMS B was also explained in their documentation. WfMS B basically chooses to dedicate a single thread to the parallel execution of scripts, leading to a non-deterministic serialization of the parallel paths. Indeed data showed that in about 50% of the cases, the fast path is chosen to be executed first (cf., Sec. 2). When the branch with the 5 sec waiting script is chosen then as expected the execution of the WfMS needs to wait 5 sec until this branch is completed. This explains the very high duration of the [EXT], as half of the executions have the 5 sec duration.

At this point we can draw some conclusions. Regarding the behavior of WfMS B on the duration of the workflow execution we observe much higher values for all the workflow patterns. The CPU and memory utilization of WfMS B is always on much lower limits when compared to the other two WfMSs because of the lower throughput. However, this is reasonable since every workflow instance is executed sequentially and the actual executed load is lower compared to the other two WfMSs, because of the higher response time. WfMS A and Camunda share many architectural similarities because Camunda was originally a fork of WfMS A. Still their behaviour is not identical and leads to some interesting points. Camunda kept the duration values low for all the workflow patterns, but for [SEQ] and [EXC] WfMS A executed slightly better. However, we note large differences in the duration values for [EXT], [PAR], and [MIX], that indicate an impact of parallelism on the performance of the WfMS A, and increased resource utilization. The parallelism does not seem to have much impact on Camunda, as it remained relatively stable in all tests. Concerning the resource utilization in general we observe WfMS B and Camunda having a more stable behaviour, while WfMS A shows a direct increase when it is more stressed. In general, we may conclude that Camunda performed better and more stable for all metrics when compared with WfMS A and WfMS B.

Finally, concerning our hypotheses (cf., Sec. 2), [SEQ] resulted in the workflow pattern with the lowest and most stable performance for all WfMSs [HYP1]. Also it was the workflow pattern with highest throughput for all tested WfMSs. Concerning the [EXC], our hypothesis was affirmed (cf., [HYP2]) as there is a slight impact on the performance, which we connect with the evaluation of the condition. The [HYP3] and [HYP4] that the [PAR] and [EXT] will have similar impacts on performance holds basically for WfMS A and Camunda. Our [HYP4] and [HYP5] for parallelism and complex structures having an impact on the performance seems to hold for WfMS A, while for Camunda no conclusions can be drawn with respect to this point. These results indicate that sequential workflows (i.e., [SEQ]) may help towards discovering the maximum throughput of the WfMSs. Parallelism (i.e., [PAR][EXT]) may affect the WfMSs in terms of throughput and resource utilization, while more complex structures (i.e., [CYC]) are better candidates for stressing the WfMSs in terms of resource utilization. These conclusions should be considered when designing the workload for more complex, realistic cases and macro-benchmarks.

4.4 Threats to Validity

A threat to external validity is that we evaluate three WfMSs, which is the minimum number of WfMS for drawing initial conclusions. For generalizing our conclusions more WfMS are needed, and for this we are designing the BenchFlow environment to allow the easy addition of more WfMSs. Moreover, our simple workload models threaten the construct validity. Although the micro-benchmark does not correspond to real-life situations, we consider it fundamental for the purposes of our work. Using this knowledge as a basis, we plan to test more complex, realistic structures such as nested parallelism and conditions, as well as combinations of them in diverse probabilistic workload mixes [17].

5 Related Work

Röck et al. [16] conduct a systematic review on approaches that test the performance of WS-BPEL WfMSs, and stress the need for improvement on WfMSs baseline tests. Micro-benchmarks target to test the performance of atomic operations and assists performance engineers towards a deep comprehension of the evaluated system, in order to assure correct and reliable results. Especially in the case of modern, complex middleware systems, micro-benchmarks are usually preferred for satisfying this goal [20]. For example Mendes et al. [12] apply several micro-benchmarks on event processing systems to answer fundamental questions on their performance concerning scalability and bottlenecks. Another micro-benchmark is introduced by Angles et al. [1] based on social networks, and define the best candidates for macro-benchmarks. Another case of micro-benchmarking is proposed by Waller and Hasselbring [20] for measuring the overhead of application-level monitoring. The proposed micro-benchmark identifies three causes of monitoring overhead, and sets the basis for a reliable macro-benchmark. Regarding WfMSs Bianculli et al. [2] ran a micro-benchmark for WS-BPEL. In this work, the goal is to rank the WfMS with respect to their performance. To the extent of our knowledge we propose the first micro-benchmark for BPMN 2.0 WfMS, with respect to the language characteristics and we are confident that it is a strong contribution for more reliable, complex macro-benchmarks in the field of WfMSs.

6 Conclusion and Future Work

In this work we have presented a micro-benchmark for BPMN 2.0 WfMSs. To the extent of our knowledge this is the first attempt to investigate the impact of BPMN 2.0 language constructs on the WfMSs performance. We ran a set of experiments on a reliable benchmarking environment and among many important observations our results showed important bottlenecks due to architectural design decisions for WfMS B and that resource utilization can be a potential issue for WfMS C. We also discovered load bottlenecks for Camunda during the execution of the arbitrary cycle pattern. Consequently, despite the simplicity of the micro-benchmark we argue that it is a potentially suitable choice for benchmarking fundamental behavior or complex real-world WfMS.

Regarding individual workflow patterns we observed that the sequential workflow pattern revealed the maximum throughput for all of the WfMSs. Parallelism (i. e., explicit termination and parallel pattern) affected two of the three WfMSs in terms of throughput and resource utilization. More complex structures, such as the arbitrary cycle, also seem impact the resource utilization, thus they can be better candidates to stress the WfMS. Finally, the mix execution helped us conclude that there are no adverse performance effects when executing different workflow patterns concurrently. While there are no side-effects when executing different types of workflows concurrently, we did observe a slight increase on individual performance metrics when compared to the homogeneous experiments with individual patterns. The above results provide the first insights on which constructs constitute meaningful candidates for building more complex benchmarks. For example, a test aiming to measure the throughput or resource utilization of the WfMS should preferably choose complex, parallel structures.

We are currently working towards a public release of the BenchFlow environment in the near future. In future work we plan to exploit the conclusions of this work to execute macro-benchmarks on BPMN 2.0 WfMSs with more complex and realistic workflows that will also contain events and web service invocations.

Acknowledgements

This work is funded by the "BenchFlow" project (DACH Grant Nr. 200021E-145062/1) project and is supported by the SPEC research group.

References

1. Angles, R., Prat-Pérez, A., et al.: Benchmarking database systems for social network applications. In: Proc. GRADES'13. pp. 15:1–15:7. ACM (2013)
2. Bianculli, D., Binder, W., Drago, M.L.: Automated performance assessment for service-oriented middleware: A case study on BPEL engines. In: Proc. WWW '10. pp. 141–150 (2010)
3. Camunda Services GmBH: Camunda BPM. https://camunda.org/ (October 2015)
4. Faban: Performance framework. `http://faban.org` (December 2014)
5. Felter, W., Ferreira, A., et al.: An updated performance comparison of virtual machines and Linux containers. Tech. rep., IBM (July 2014)
6. Ferme, V., Ivanchikj, A., Pautasso, C.: A framework for benchmarking BPMN 2.0 workflow management systems. In: Proc. BPM '15. Springer (2015)
7. Ferme, V., Pautasso, C.: Integrating faban with docker for performance benchmarking. In: Proc. of the 7th ACM/SPEC International Conference on Performance Engineering. Delft, The Netherlands (March 2016)
8. Geiger, M., Harrer, S., Lenhard, J., Casar, M., Vorndran, A., Wirtz, G.: BPMN conformance in open source engines. In: Proc. of the 9th IEEE International Symposium on Service-Oriented System Engineering (SOSE 2015). SOSE 2015, IEEE, San Francisco Bay, CA, USA (March 30 – April 3 2015)
9. Jordan, D., Evdemon, J.: Business Process Model And Notation (BPMN) Version 2.0. Object Management Group, Inc (2011), `http://www.omg.org/spec/BPMN/2.0/`
10. von Kistowski, J., Arnold, J.A., et al.: How to build a benchmark. In: Proc. ICPE '15. pp. 333–336 (2015)
11. Lazowska, E.D., Zahorjan, J., et al.: Quantitative System Performance: Computer System Analysis Using Queueing Network Models. Prentice-Hall (1984)
12. Mendes, M.R., Bizarro, P., Marques, P.: A performance study of event processing systems. In: Performance Evaluation and Benchmarking, pp. 221–236. Springer (2009)
13. Merkel, D.: Docker: Lightweight Linux containers for consistent development and deployment. Linux J. 2014(239) (Mar 2014)
14. Montgomery, D.C., Runger, G.C.: Applied Statistics and Probability for Engineers. John Wiley and Sons (2003)
15. Muehlen, M.Z.: Workflow-based process controlling. Logos (2004)
16. Röck, C., Harrer, S., Wirtz, G.: Performance benchmarking of BPEL engines: A comparison framework, status quo evaluation and challenges. In: Proc. SEKE 2014. pp. 31–34 (2014)
17. Skouradaki, M., Roller, D.H., et al.: On the road to benchmarking BPMN 2.0 workflow engines. In: Proc. ICPE '15. pp. 301–304 (2015)
18. Standard Performance Evaluation Corporation: SPEC's Benchmarks (SPEC), `https://www.spec.org/benchmarks.html`
19. Van Der Aalst, W.M.P., Hofstede, T., et al.: Workflow patterns. Distrib. Parallel Databases 14(1), 5–51 (Jul 2003)
20. Waller, J., Hasselbring, W.: A benchmark engineering methodology to measure the overhead of application-level monitoring. In: KPDAYS. CEUR Workshop Proceedings, vol. 1083, pp. 59–68. CEUR-WS.org (2013)
21. Wetzstein, B., Leitner, P., et al.: Monitoring and analyzing influential factors of business process performance. In: Proc. EDOC '09. pp. 141–150 (2009)
22. Wohed, P., van der Aalst, W., Dumas, M., et al.: On the Suitability of BPMN for Business Process Modelling. In: Business Process Management, LNCS, vol. 4102, pp. 161–176. Springer (2006)