Performance Comparison Between BPMN 2.0 Workflow Management Systems Versions

Vincenzo Ferme*,[1], Marigianna Skouradaki[2], Ana Ivanchikj[1], Cesare Pautasso[1], and Frank Leymann[2]

[1] Faculty of Informatics, USI Lugano, Switzerland
[2] Inst. of Architecture of Application Systems (IAAS), Univ. of Stuttgart, Germany

Abstract. Software has become a rapidly evolving artifact and Workflow Management Systems (WfMSs) are not an exception. WfMSs' changes may impact key performance indicators or resource consumption levels may change among different versions. Thus, users considering a WfMS upgrade need to evaluate the extent of such changes for frequently issued workload. Deriving such information requires running performance experiments with appropriate workloads. In this paper, we propose a novel method for deriving a structurally representative workload from a given business process collection, which we later use to evaluate the performance and resource consumption over four versions of two open-source WfMSs, for different numbers of simulated users. In our case study scenario the results reveal relevant variations in the WfMSs' performance and resource consumption, indicating a decrease in performance for newer versions.

Keywords: Performance Testing · Performance Regression · BPMN · Workflow Management Systems · Workflow Engine

1 Introduction

In the era of rapidly evolving software, semantic versioning has introduced a standardized meaning for version numbers. While this versioning style conveys the extent of changes introduced with the new version, in terms of backward compatibility, new features or bug fixes; it fails to provide information regarding changes in system's performance. This holds for Workflow Management Systems (WfMSs) as for any other software. As the number of proprietary and open-source WfMSs increases, measuring and comparing their performance between different versions becomes imperative for continuously improving them [18]. Information on the performance and/or resource consumption differences between versions is not only relevant for the developers, but is also important for the decision-making regarding the upgrade or selection of a WfMS by the end users. Especially, when the WfMS is running in the Cloud.

However, WfMSs' performance does not only depend on its version, but also on the workload (i.e., the workload mix, load function and test data) applied to

* Corresponding author

the System Under Test (SUT). The end user requires performance information related to a given workload mix, i.e., the mix of Business Process (BP) models, which is structurally representative of a possibly large BP collection and related to a representative load function, which describes how the workload mix is issued to the WfMS. To address this challenge, we define and follow a novel method for deriving a synthetic workload mix from a given BP models collection. We focus on the Business Process Model and Notation (BPMN 2.0) [8] as a common modeling and execution language, since it allows to use a uniform standard representation for the workload mix.

As a case study, we apply the proposed method to derive a structurally representative workload mix from a real-world collection, which we then use to execute performance tests on the process navigator of two different popular open-source WfMSs. The process navigator is a core component of the WfMSs, that is responsible for navigating through the control flow of the BP models [7]. For each of the WfMSs, we test and compare the last four minor versions using the reliable BenchFlow environment [4]. Hence, the scientific contributions of this work are: 1) a method for deriving a structurally representative workload mix, and 2) an extensive analysis of the results from applying the proposed method on a case study scenario, providing insights on the evolution of two WfMSs in terms of performance and resource consumption.

The rest of this paper is structured as follows: Sect. 2 presents the method for generating a representative workload model; Sect. 3 refers to the configuration of the performance testing environment and the WfMSs; Sect. 4 presents and discusses the results from the case study; threats to validity are addressed in Sect. 5 and related work in Sect. 6. We conclude the paper and present plans for future work in Sect. 7.

2 Defining a Representative Workload Model

In previous work [5], we have identified the WfMS workload model components and their interactions. Failing to derive a workload model which is representative of a given initial BP model collection might produce misleading results and hence inappropriate decisions [3]. Therefore, in this section we present a parametric method for generating such workload model. Consequently, instead of using arbitrary BP models in performance tests, companies may generate synthetic BP models that reflect the essence of their BP models collection. The advantage of applying a parametric method for the workload mix generation, is that it enables a future re-application on diverse BP collections and/or domain-specific requirements. Then, we also discuss the parametric definition of the load functions.

2.1 Workload Mix Generation Method

For deriving BPs with representative structures we propose the following four phased *workload mix generation method*: Phase 1) Analyze the initial BP model collection; Phase 2) Discover the reoccurring structural patterns; Phase 3) Synthesize the BPs of the workload mix with respect to user defined parameters; and Phase 4) Partition the BPs into workload classes. The *workload mix generation method* takes as an input a collection of real-world BPs. In Phase 1 we apply statistical analysis on that collection to identify its main structural characteristics. Moreover, we execute clustering analysis based on the BPs' static metrics (e.g., number of activities, number of gateways) to obtain additional insight on the collection's characteristics. Clustering analysis is a grouping method that places similar objects in the same group (i.e., cluster). In Phase 2 we proceed to the detection and extraction of the most frequently reoccurring structural patterns in the BPs of the collection via the RoSE algorithm [16]. The detected patterns are extracted and annotated with their frequency of appearance in the original collection, as well as other metadata regarding their structure. The extracted structures and their metadata are then used in Phase 3 for recognizing and synthesizing [14] representative BPs, in accordance with user-defined parameters, such as the size and the control flow characteristics of the synthesized BPs, which can be derived from the statistical and clustering analysis. Graph synthesizing has been previously used (e.g., by Gupta [6]). However, this is the first time that parametric generation of representative BPs out of reoccurring structures is used for performance testing. Finally, in Phase 4 we divide the BPs into workload classes. A *workload class* is the pair of BP and intensity with which each BP participates in the workload mix. Thus, the workload mix is comprised of the different workload classes used as input to the SUT [5]. Each class participates in the workload mix with a different intensity, which corresponds to the degree of the model's representativeness of the collection.

Let us assume a set $C = \{c_1, c_2, ..., c_k\}, k \in \mathbb{N}$ of BP models that we include in the classes of the workload mix. In our case the set C maps to the BP models shown in Fig. 1 to 5. For calculating the representativeness *repr* of a BP model ($c_k \in C$) to the collection we define the following formula:

$$repr(c_k) = \frac{1}{2\,|S_k|} \sum_{s_i \in S_k} \left(\frac{t(s_i)}{|Sc|} + \frac{m(s_i)}{|M|} \right) \tag{1}$$

where:

$M = \{m_1, m_2, ..., m_j\}$ is the set of BP models in the original collection.

$Sc = \{s_1, s_2, ..., s_n\}$ is the collection of all the reoccurring structures s_i detected in the original BP models collection M. A given structure s_i can reoccur multiple times within the same $m_i \in M$, and/or in different models in M, and thus multiple times in the collection Sc.

$S_k \subset Sc$ is the set of reoccurring structures participating in the BP model c_k.

$t: \ Sc \to \mathbb{N}$ is a function counting how many times a structural pattern $s_i \in Sc$ is present in all $m_i \in M$, counting each time s_i is found in the same m_i.

$m: \ Sc \to \mathbb{N}$ is a function returning the number of BPs in the set M, in which the structural pattern $s_i \in Sc$ is present at least once.

The *intensity* is then computed as the normalized representativeness ($repr(c_k)$, Eq. (1)) with respect to the whole collection.

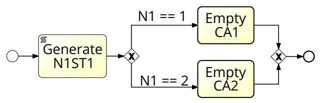

Fig. 1: Class 1: s_1, Cluster 1

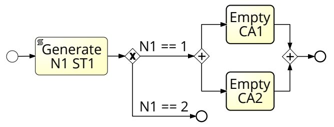

Fig. 2: Class 2: s_2, Cluster 1

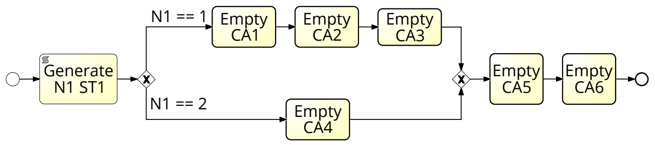

Fig. 3: Class 3: s_3, Cluster 2

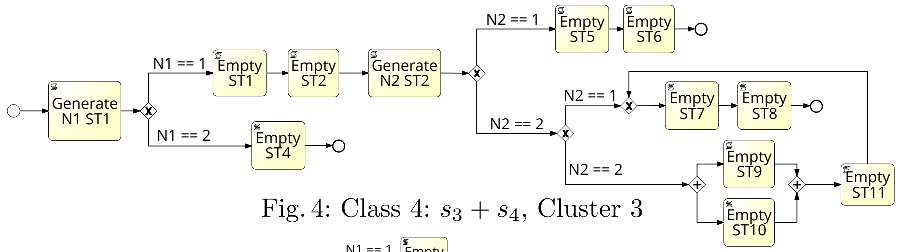

Fig. 4: Class 4: $s_3 + s_4$, Cluster 3

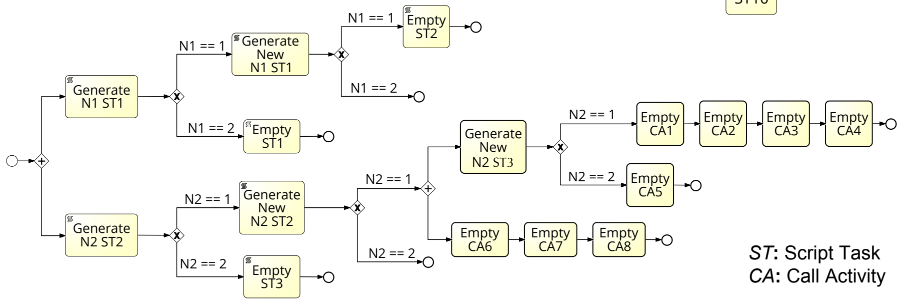

Fig. 5: Class 5: $s_5 + s_6$, Cluster 4

2.2 Applying the Workload Mix Generation Method

Phase 1: To create a case study scenario, we use a collection of 3'247 valid and complete real-world BPMN 2.0 BPs originating from: i) IBM Industry Process and Service Models[3], ii) sample models provided by the BPMN 2.0 standard, iii) the research by Pietsch et al. [12], iv) the BPM Academic Initiative[4] (invalid and incomplete BP models were removed) and v) other research and industrial partners. The diversity of our collection reflects companies with a big portfolio of different processes and results in a more "general" synthetic workload. Since event logs or real data to simulate the BP execution were not shared with us, a behavioral analysis of the example collection was not possible at this point. In the collection, the BP size ranges from 3 to 120 nodes. Models with size $5 \leqslant size \leqslant 32$ represent 82% of the collection. Despite BPMN 2.0's expressiveness, the detected reoccurring structures contain only a small subset of the BPMN 2.0

[3] `http://www-01.ibm.com/software/data/industry-models/`
[4] `http://bpmai.org/`

Table 1: Occurrences of appearance (t within the structures, m within the BPs) of the reoccurring structures and their intensity (c.f., Eq. (1))

	Class 1	**Class 2**	**Class 3**	**Class 4**		**Class 5**	
$\boldsymbol{s_i}$	s_1	s_2	s_3	s_3	s_4	s_5	s_6
$\boldsymbol{t(s_i)}$	1'602	953	640	640	309	130	30
$\boldsymbol{m(s_i)}$	1'731	2'303	1'710	1'710	635	157	30
Intensity	39%	27%	19%	13%		2%	

constructs. More than 95% of the elements are one of the following: Call Activity, Exclusive/Parallel/Inclusive Gateway, Task (Script, User, Receive, Send), or Start and End Event. This confirms the findings of earlier literature studies [11]. The performed clustering analysis resulted in six clusters of gradual complexity. This result is important to better characterize the collection, and in our use case it is used as input parameter for Phase 3. The first four clusters represent 94% of the collection, while the last two the remaining 6%. Therefore, we consider the first four clusters to be the most representative of the collection's structure:

Cluster 1: 1 Start, 2 End Events, 4 Activities, 1 Ex. Gateway
Cluster 2: 1 Start, 2 End Events, 6 Activities, 2 Ex. Gateways
Cluster 3: 1 Start, 3 End Events, 11 Activities, 4 Ex. Gateways, 1 Par. Gateway
Cluster 4: 1 Start, 3 End Events, 16 Activities, 5 Ex. Gateways, 1 Par. Gateway

Phase 2: The RoSE algorithm [16] is a novel algorithm that applies techniques of sub-graph isomorphism to detect reoccurring structural patterns in a collection of BP models. For the aforementioned collection of BP models it detected 143 structural patterns appearing more than once in the collection.

Phase 3: The clusters indicate the structural attributes a BP should have, while the detected recurring structural patterns indicate their control flow. The combination of these results helps us to parametrize the workload mix generation by controlling the structural characteristics of the produced synthetic BP model (e.g., the number of constructs per type of construct). Namely, the parameters are formed with respect to the results of the clustering analysis. After the synthesis we obtain the representative BPs shown in Fig. 1 to 5. The times ($t(s_i)$) and BPs ($m(s_i)$) of occurrence of the participating structures are shown in Table 1. To obtain fully automated executable models, we implement all tasks as *script tasks* or *call activities*. We omit external interactions (i.e., human tasks and web service invocations) in order to focus on the observation of the performance of one of WfMS's critical components, i.e., the process navigator. Additionally, the original collection had low ratio of intermediate and boundary events (i.e., at most 5 occurrences in the whole collection), thus they are not included in the synthesised BPs. Empty *script tasks* are used, except when they precede an exclusive gateway, in which case the script task generates random numbers producing a uniform probability of taking any outgoing control flow branch of the exclusive gateway. Call activities call an empty BP (*Start event - Empty Script Task - End Event).*

Phase 4: Finally, we divide the derived BPs into classes (c_k) and compute their corresponding *intensity* (cf. Table 1).

2.3 Load Function

The interactions with the WfMS needed to start new BP instances follow the load function defined by different parameters: the set of workload classes to be started with the given *intensity*; a ramp-up period (30 sec) during which the number of instantiated BP instances is gradually increased, followed by a steady state (10 min), where the number of instantiated BP instances remains stable, and a ramp-down period (30 sec), during which the number of instantiated BP instances is gradually decreased; a variable number of simulated users; and a think time [9] of 1 sec. Such short think time leads to a load function that stresses the WfMSs depending on the number of concurrent users. To reflect realistic number of users interacting with a WfMS in differently sized companies[5], we parametrize and issue the defined load function by setting 50, 500 and $1'000$ simulated users in three different experiments. All parameters of the load function, are configurable by the designer of the performance test.

3 Case Study Settings

The WfMSs' configuration and the setup of the performance testing environment aim at reducing, as much as possible, the noise in the measurements. To that end we isolate the individual testing components through containers [5] and control the testing environment to ensure the absence of interferences in the measurement. For the tests, we used the same methodology as in [15] and run the experiments with the BenchFlow environment [4], an end-to-end framework for WfMSs' performance testing relying on Docker[6]. BenchFlow aims at ensuring reliable and reproducible results, that can be verified by means of dedicated statistics, reported in Sect. 4.2.

The WfMSs we tested are two widely used open source engines[7]. They are widely used in industry and have a large user community as per vendors' websites. We cover their last two years of development (2014-2016), i.e., versions 7.2.0, 7.3.0, 7.4.0, 7.5.0 for WfMS A and versions 5.18.0, 5.19.0.2, 5.20.0, 5.21.0 for WfMS B. Comparing versions provides insights on how system's evolution impacts its performance. For WfMS A we used the official Docker images and vendor suggested configurations. WfMS B's default configuration has been reported as insufficient for the deployment of realistic loads, thus we used the one suggested at the vendor's website. We deployed it using the most popular Docker image, since no official Docker image is currently available. The configurations of WfMS A and WfMS B are comparable in terms of the connection pool, the Java Virtual Machine and the Application server settings they rely on, as well as, in terms of the BP execution logging level, which we set to log the full history. Both WfMSs utilize MySQL Community Server 5.7.15 as Database Management System (DBMS), installed in a Docker container[8].

[5] `http://www.gartner.com/it-glossary/smbs-small-and-midsize-businesses/`
[6] `https://www.docker.com`
[7] We do not have an explicit vendors' consent to publish WfMSs' names
[8] `https://hub.docker.com/_/mysql/`

The WfMS and the DBMS run on two exclusively dedicated servers connected via a dedicated 10 Gbit/s network, without relying on the Docker network bridge (i.e., we use the Docker's *host* network option). The WfMS dedicated server has 64 Cores (2 threads) and a clock speed of 1'400MHz mounting 128GB of RAM and a magnetic disk with 15'000 rpm. The DB dedicated server has 64 Cores (2 threads) and a clock speed of 2'300MHz mounting 128GB of RAM and a SSD SATA disk. The aim of such resource allocation is to avoid the DBMS becoming a performance bottleneck. Different machines, interacting with the WfMS on a second dedicated 10 Gbit/s network, are allocated for the simulation of the users, thus ensuring sufficient resources for simulating the defined load.

4 Evaluation

4.1 Performance Metrics

We characterize WfMSs' performance, using metrics that represent the performance from different points of view: 1) the client (i.e., representing the users starting BP instances), 2) the BP execution behavior, and 3) the system's resource consumption. To ensure results' reliability, given the non-determinism in WfMS's performance, we perform three rounds of executions for each experiment. Out of these multiple rounds, we compute aggregated metrics representing WfMS's observed behavior and the performance variability across different rounds.

On the client-side we include the *number of requests per second* - $\#REQ/s$ issued by the simulated users. The maximum expected $\#REQ/s$ equals the number of simulated users defined in the load function. The actual obtained value is impacted by the WfMS's response time. We report the $\#REQ/s$ metric aggregated using the average (avg) of the metric across the different rounds, as well as the 95% T-based confidence interval (ci) [10, Chap. 8]. The ci sets up a range of values for the analyzed metric in which we can be 95% confident.

A second set of metrics are computed starting from the server-side performance data logged by the WfMSs. In this work we include the *BP instance (bp_i) duration in milliseconds* - D $[ms]$ and the *throughput* - T $[\#bp_i/s]$. The duration D is defined as the time interval between the start and the completion of a BP instance. We report the weighted average ($wavg(D)$) of D aggregated among all the executed BP instances and for each single BP in the workload mix, where we compute the weights based on the number of executed BP instances in each round. As throughput T we define the number of executed BP instances per second. We report its average ($avg(T)$) along with the corresponding ci.

We also compute resource consumption metrics, based on data with a sampling interval of 1 second. In this work, we include the *weighted average of CPU, RAM consumption* - $wavg(CPU)$ [%], $wavg(RAM)$ $[MB]$ among different rounds. The weights are based on the number of CPU and RAM data points in each experiment round. Given the logged execution data, when necessary other metrics can be defined by the performance test designer depending on its goal.

Table 2: Performance and Resource Consumption Metrics - WfMS A

			WfMS A			
		Load	**7.2.0**	**7.3.0**	**7.4.0**	**7.5.0**
Client-side	$avg(\#REQ/s) \pm ci$	**50**	49.13±0.04	49.17±0.03	49.16±0.02	49.09±0.01
		500	484.87±0.39	486.44±0.10	484.84±0.82	482.91±2.20
		1'000	890.84±4.82	879.15±9.94	859.81±3.42	763.46±2.17
Server-side	$avg(T) \pm ci$ [#bp_i/s]	**50**	118.23±0.21	119.72±0.17	120.08±0.79	120.05±0.42
		500	1'185.12±0.45	1'185.56±0.33	1'180.80±4.34	1'175.10±0.58
		1'000	2'121.35±6.23	2'130.90±9.50	2'087.26±2.72	1'849.88±5.17
	$wavg(D)$ [ms]	**50**	1.03	1.08	1.12	1.24
		500	0.87	0.91	1.07	1.14
		1'000	0.93	0.99	1.05	1.15
Resource Consumption	$wavg(CPU)$ [%]	**50**	1.66	1.33	1.33	1.35
		500	8.07	6.26	7.03	6.95
		1'000	11.39	8.33	8.81	8.79
	$wavg(RAM)$ [MB]	**50**	637.28	590.82	634.79	648.67
		500	885.10	860.70	886.53	866.37
		1'000	970.61	957.47	978.97	971.43

		bp_i				
Server-side	$wavg(D)$ [ms]	**Class 1**	1.09	1.14	1.24	1.35
		Class 2	1.34	1.41	1.54	1.67
		Class 3	2.31	2.43	2.64	2.88
		Class 4	1.13	1.20	1.30	1.42
		Class 5	1.93	2.03	2.21	2.40

4.2 Reliability of Results

To ensure reliable results, we use *Little's Law* to verify that the BenchFlow environment was able to simulate the number of defined users. It compares the number of defined users versus the number of actually simulated users. BenchFlow was able to simulate the number of users defined in the load functions, with an acceptable small variation of less than 1% (at maximum).

Every software system experiences a warm-up time during which its transient behaviour differs from the one in the steady-state [9]. To account for it, we identify the outlier BP three instances during the ramp-up phase of the load function, and remove them from the analyzed data set. To verify that the three rounds obtain similar, and reliable results, we compute the *coefficient of variation - cv* [%]. The *cv* is the ratio between the standard deviation of the means of the rounds and the mean of all the rounds, expressed as a percentage. The *cv* resulted always below 3.5%. This indicates a stable behavior across the different rounds.

4.3 Results

The performance metrics (cf. Sect. 4.1) for all tested versions of WfMS A and WfMS B, with different number of users are reported in Table 2 and 3 respectively.

WfMS A's ability to handle incoming requests, depicted by the average requests per second, is close to the expected for 50 and 500 simulated users, with actual numbers of over 49 and over 482 requests per second respectively. With 1'000 users the client-side performance drops from 890.84 REQ/s in v7.2.0 to 763.46 REQ/s in v7.5.0. There is a slight increase in throughput from 118.23 bp_i/s to 120.05 bp_i/s from the oldest to the newest version analyzed for 50 users. For

Table 3: Performance and Resource Consumption Metrics - WfMS B

			WfMS B			
		Load	**5.18.0**	**5.19.0.2**	**5.20.0**	**5.21.0**
Client-side	$avg(\#REQ/s) \pm ci$	**50**	48.84±0.02	48.72±0.05	48.66±0.03	48.65±0.04
		500	488.61±0.12	487.72±0.13	487.34±0.15	487.71±0.55
		1'000	906.10±3.81	900.14±4.09	891.62±1.76	885.80±2.97
Server-side	$avg(T) \pm ci$ [$\#bp_i$/s]	**50**	119.87±0.07	119.82±0.09	119.99±0.09	119.82±0.14
		500	1'161.88±0.98	1'160.46±1.13	1'160.96±1.67	1'132.92±2.43
		1'000	2'182.78±8.25	2'202.37±6.37	2'189.36±4.83	1'974.01±10.59
	$wavg(D)$ [ms]	**50**	6.53	6.75	6.90	7.19
		500	5.08	5.27	5.56	5.65
		1'000	5.18	5.34	5.39	5.43
Resource Consumption	$wavg(CPU)$ [%]	**50**	3.01	1.59	1.63	1.66
		500	8.82	7.55	9.59	9.50
		1'000	12.05	12.74	12.78	11.84
	$wavg(RAM)$ [MB]	**50**	1'764.83	2'430.69	2'620.57	2'455.11
		500	8'686.66	8'653.84	8'633.66	8'583.11
		1'000	9'549.54	9'749.74	9'509.81	9'350.82

		bp_i				
Server-side	$wavg(D)$ [ms]	**Class 1**	9.28	9.52	9.79	10.02
		Class 2	6.09	6.24	6.42	6.57
		Class 3	16.73	17.16	17.64	18.06
		Class 4	3.53	3.62	3.73	3.81
		Class 5	22.49	23.07	23.72	24.28

500 and 1'000 users the best throughput of 1'185.56 bp_i/s and 2'130.90 bp_i/s respectively is achieved with v7.3.0. There is a reduction to 1'175.10 bp_i/s and 1'849.88 bp_i/s respectively in v7.5.0, the newest version. Given that most of the BPs used in the workload mix contain *call activities* which instantiate a globally defined BP, the throughput is much higher than the actual number of requests per second sent by the users, because it considers the instantiated BP instances as well. The average duration of the execution of one BP instance is the lowest in v7.2.0 for 500 users with 0.87 ms execution time, and the highest in v7.5.0 for 50 users with 1.24 ms execution time as it increases with newer versions of the WfMS. The same applies to the average duration at BP level (rolled-up by the number of users), with lowest duration of 1.09 ms for Class 1 in v7.2.0, and highest of 2.88 ms for Class 3 in v7.5.0. The duration of the empty BP instantiated by the *call activities*, omitted in the table, is on average 0.31 ms. A lower number of started and completed BP instances require less CPU (from an average of 1.42% across all versions for 50 users, to 9.33% with 1'000 users) and less RAM (from an average of 627.89 MB across all versions for 50 users to 969.62 MB with 1'000 users).

Comparable tendencies are noticeable for **WfMS B** which handles over 48 and over 487 requests per second for 50 and 500 users respectively. Greater variation among versions is present when 1'000 users are simulated with actual number varying from 906.10 in v5.18.0 down to 885.80 requests in v5.21.0. Similar trends are evident in the throughput, which is relatively stable for 50 users and amounts to an average of 119 completed BP instances per second and an average of 1'160 for 500 users, except for v5.21.0 where the throughput drops to 1'132.92 bp_i/s. As with the number of requests per second, the throughput also decreases from 2'182.78 in v5.18.0 to 1'974.01 bp_i/s in v5.21.0 for 1'000 users. The average

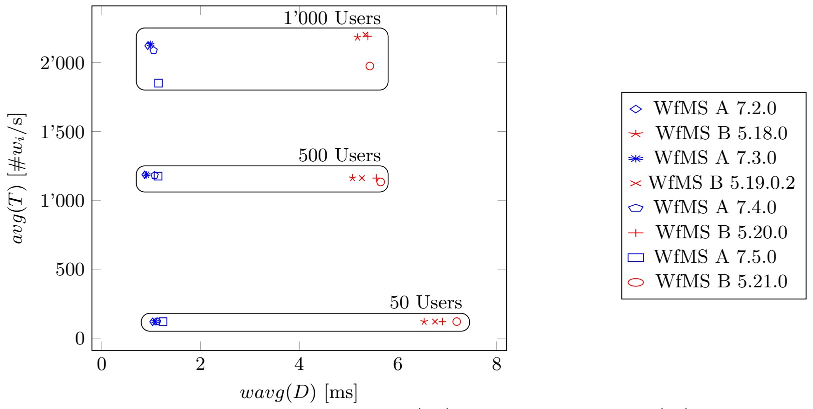

Fig. 6: Instance Duration (D) vs. Throughput (T)

duration of a single BP instance follows the same trends as in WfMS A, but with much higher absolute values of 5.08 ms in v5.18.0 for 500 users as the lowest duration, to a value of 7.19 ms in v5.21.0 for 50 users as the highest duration. When analyzed at BP level, the lowest average duration of 3.53 ms is observed for Class 4 in v5.18.0, while the highest of 24.28 ms for Class 5 in v5.21.0. The duration of the empty BP instantiated by the *call activities*, omitted in the table, is on average 1.17 ms. Usage of CPU is comparable to WfMS A, with an average of 1.97% across versions for 50 users increasing to 12.35% for 1'000 users. RAM usage, on the other hand, is much higher than in WfMS A with an average of 2'317.80 MB across versions for 50 users and 9'539.98 MB for 1'000 users.

4.4 Discussion

In Table 2 and Table 3 we can see the detected differences in performance and resource consumption between the two WfMSs, as well as among different versions of the same WfMS. Some of these differences only become obvious when a higher number of users is interacting with the system, making it relevant to have parametrized load functions representing both small and large companies.

The number of client requests per second shows the average performance of the system from the user point of view. It is relatively stable between all versions of both WfMSs when tested with 50 and 500 users. However, there is a more substantial decrease when tested with 1'000 expected users, especially in WfMS A. The expected maximum is the number of simulated users. The actual value depends mainly on the WfMS's response time and it gets more distant from the expected value as the number of simulated users increases (cf. Table 2 and 3). The resource consumption metrics in Table 2 and 3 verify that this behavior does not emerge due to unavailability of resources. The actual resource consumption in all versions is far from the theoretical maximum of the servers (see Sect. 3). The DBMS resource consumption data did not point to any bottlenecks in the communication with the DBMS. They showed that the DBMS had enough resources to handle the issued load. While WfMS A slightly outperforms WfMS B in this metric with a load of 50 users, it falls behind with a load of 500 and 1'000 users. WfMS B experiences 5.65% better $\#REQ/s$ on average than WfMS A, mainly due to the last version of WfMS A being c.a. 15% slower than the last version of WfMS B in starting new BP instances with 1'000 users. If one expects performance improvements with new system releases, these results could be surprising given that, with respect to these metrics, older versions show better performance than newer versions.

The performance decrease with newer versions is made even more evident by the BP instance duration metric (D). As we can see from Table 2, Table 3 and Fig. 6, the average duration of a BP instance increases as new versions are introduced for both WfMSs, regardless of the number of users. When looking at the columns of Table 2 and Table 3, in the last two versions of each system the average instance duration decreases as the number of users increases. This decrease is especially noticeable for WfMS B when going from 50 to 500 users, and less when going from 500 to 1'000 users (cf. Fig. 6). It is also interesting to point out that WfMS B performs worse than WfMS A with respect to the average duration for the single BP instance. WfMS B is on average 5-6 times slower than WfMS A in executing the BP instances. As previously noticed, this is not due to the unavailability of resources. After all WfMS B uses more resources, especially RAM, than WfMS A to obtain lower performance.

The throughput is relatively stable between all versions of both WfMSs when 50 users are involved, with WfMS A showing a slight increase in more recent versions. However, a relatively substantial decrease in throughput is observed in the newest version of both WfMS A and WfMS B when the load is raised to 1'000 users (cf. Fig. 6). When looking at both systems, their throughput is comparable with 50 users. Then WfMS A outperforms WfMS B by 2% to 4% in newer versions with 500 users, but WfMS B takes the lead when there are 1'000 users by 3% to 6% in newer versions. This might be unexpected given that WfMS B's average BP instance duration is 5 times greater than WfMS A's. It might be due to the fact that WfMS B accepts 15% more instances per second with 1'000 users by exploiting parallelism when executing the instances, thus balancing the longer instance duration.

We also show the average BP instance duration for each model class among all loads in Table 2 and Table 3 (bottom). When looking at the data, care needs to be taken as the averages for the different BPs are calculated using different numbers of data points. This is due to the different intensities in the execution of the different models as presented in Table 1. Table 2 and Table 3 show that the mentioned increase in average BP instance duration in newer WfMS versions is not caused from one particular BP, but is noticeable in all BPs. However, different models perform differently for the two WfMSs. While Class 1 (cf. Fig. 1) is the fastest in WfMS A with an average duration of 1.21 ms across all loads and versions, in WfMS B Class 4 (cf. Fig. 4) is the fastest with an average of 3.67 ms. And while Class 5, the model with the greatest size (cf. Fig. 5), has the longest duration in WfMS B (avg. 23.39 ms), this is not the case with WfMS A, where Class 3 (cf. Fig. 3) is the slowest one (avg. 2.56 ms). Having noticed such differences, we examined the execution data at construct level. While WfMS B is on average 7 times slower than WfMS A in executing the *call activities* (avg. 5.06 ms vs. 0.70 ms respectively), it actually performs better in executing parallel gateways (avg. 0.01 ms vs. 0.11 ms). WfMS B executes parallel gateways faster than exclusive gateways (avg. 0.01 ms vs. 0.04 ms) which might explain the faster execution of Class 2 vs. Class 1. The slower execution of instances in WfMS B also partially results from its slow instance start-up, with an average duration of the start event of 0.56 ms vs. 0.04 ms in WfMS A. When analyzing the execution duration of individual constructs at model level, stable behaviour is noticed for start and end events, while the greatest variation between models is observable for the *call activity* and *script task* in WfMS B and for the gateways in WfMS A.

Last, but not least, the resource consumption metrics verify that the WfMSs' performance behavior is not caused by lack of resources. Average CPU and RAM consumption is relatively stable between versions for both systems. In fact, the average CPU consumption is at most 12.78% across the different experiments, while the maximum (not reported in the table) is 85% for WfMS B, and 82% for WfMS A. The average RAM consumption is at most 9'749.74 MB, out of the maximum available of 128 GB. These metrics are also a powerful indication of the required resources for obtaining the results produced by the other performance metrics. The CPU consumption is comparable between the two systems, with slightly higher values for WfMS B especially for 1'000 users for all versions. Regarding the RAM consumption, WfMS B always needs more RAM than WfMS A, up to $10x$ times more for certain experiments.

5 Threats to Validity

The complexity of running performance experiments on WfMSs, contributes to the following threats to validity.

Construct validity is threatened from the fact that the extracted workload mix depends on the collection used as an input to the analysis of recurring structures. We mitigate the resulting generalization limitations by using a large and heterogeneous collection. Moreover, we provide a parametric method to derive the workload mix, so that it can be applied to other collections (e.g., domain-specific or customer-specific). Another threat to the construct validity is the unavailability of log data, which has resulted with a randomly generated artificial data for evaluating the path to follow in conditional path decisions. Moreover, all the scripts not used for data generation are empty, thus limiting the evaluation of the performance impact of data stored by tasks in the BP instances. The workload mix has not been experimentally compared to any similar baseline approach, leaving a degree of uncertainty regarding the level of accuracy. Finally, the generated workload mix remains unchanged between the experiments, thus hindering a profound comparison between the WfMSs and their versions with regard to the characteristics of the workload mix. We mitigate this by a short discussion about the performance differences between the classes of models used in the workload mix.

A threat to the external and construct validity is that specific settings of our load functions (e.g., the number of users and their think time) could be more realistic. However, the parametric method can still be applied upon availability of additional real-world data and execution logs. Nonetheless, the load functions used were sufficient to stress the system and obtain initial performance insights.

The use of different servers for the WfMS and the DBMS connected through a dedicated 10 Gbit/s network is also a potential threat to external and construct validity, because of the network latency that might impact the communication between the WfMS and the DBMS. However, we are confident that our set-up mitigates the impact of network latency and congestion on the attained measurements. Related to this, results may differ when using a system with different (e.g., higher) hardware specifications than the ones used in our experiments or, for example, when using another DBMS. Additional threat is the use of a single configuration per WfMS. Performance differences may be noticed when changing the configuration.

6 Related Work

Parametric Workload Generation Graph-based workloads have been applied for performance testing applications that model data as graphs. In addition to size, Duan et al. [2] introduce the metric of "structuredeness" of RDF datasets generated for performance testing purposes. Vicknair et al. [17] compare the performance of a graph DB and a relational DB. The defined workload is divided into structural and data queries. The structural queries address the storage of data provenance information as Directed Acyclic Graphs. The data queries use payload data, with artificial provenance information. Similar to both, for the definition of our representative workload mix, we combined structural characteristics of the BPs by considering size and other statistical information. Gupta [6] stresses the need for a parametric method to allow the user to dynamically generate workload with respect to structural parameters. Our method offers generation and parametrization of the workload mix, when the original collection or the performance goals vary. The importance of the graph's structure is most of the times ignored [2], and the structural information is represented by empirical and/or artificial data [17]. In our case, the structural information of the workload is defined by extracting the reoccurring structural patterns in a collection. The same method can be applied to any BPMN 2.0 collection, due to its parametric nature. By combining the derived information with other statistical data, such as descriptive statistics, clustering and popularity metrics [3], we are able to define different classes in the workload mix [14]. Overall, in the area of big and linked data, the generation of realistic graphs for performance testing purposes is well established. However, to the extent of our knowledge, this is the first time that an approach for BP synthesis is proposed for WfMSs performance testing. To this end, the experimental comparison of the proposed approach to a less advanced approach is currently not possible.

WfMSs Performance Testing In the literature we can find work on characterizing WfMSs' performance. As reported by Röck et al. [13], who conducted a systematic review on approaches that test the performance of WS-BPEL WfMSs, all performance tests are custom or micro-benchmarks. The most prominent one is SOABench [1], which uses a simple workload mix composed by basic BPEL structures. SOABench is used to compare the response time of three open-source WfMSs (ActiveVOS, jBPM, and Apache ODE) using a different number of clients with variable think times and has identified some scalability limitations of the tested systems. To the best of our knowledge, our existing work on micro-benchmarking [15], is the first effort to propose a performance testing method for BPMN 2.0 WfMSs. The results in [15] showed bottlenecks in architectural design decisions and resource consumption, as well as limits on the load the WfMS can sustain. The results presented in this paper extend our earlier observations with experiments based on a more complex, structurally representative, parametric workload and by comparing different WfMSs versions.

7 Conclusion and Future Work

In this paper we presented a novel parametric method for deriving a structurally representative workload mix from a potentially large BPMN 2.0 collection, to respond to the need of evaluating the performance of different WfMSs versions with an appropriate workload. We illustrated how to obtain the BP models of a workload mix, using the proposed method for distilling the most prominent control-flow characteristics of the given collection. We also proposed a method for using such collection to determine the proportion of instantiated BPs in the workload mix. We then parametrized the load function for different numbers of simulated users (50, 500, and 1'000) representing differently sized companies. We issued the derived workload model to four different versions of two widely used open-source BPMN 2.0 WfMSs. After validating the reliability of the obtained results we computed diverse performance metrics. While WfMS A demonstrated significantly lower average duration of a single BP instance and lower RAM usage, WfMS B had the lead in throughput. Furthermore, over both system releases within the past two years, it appears that priority was given to adding new features as opposed to improving the performance. The obtained results justify the need for performance testing of different WfMSs versions using a workload mix representative of user's needs and BP collections.

As future work we plan to apply the proposed method to domain-specific model collections. In such settings we also plan to extend our method to support events, human tasks, and WfMS interaction with external systems, such as Web Service APIs. We also aim at the experimental comparison of the synthetic workload mixes to real-world process models, in order to identify the accuracy of our workload mix generation method. Moreover, we have started analyzing execution logs of real world BP instances to define even more realistic load functions. We plan to add more WfMSs and run the experiments with different configuration settings.

Acknowledgements

This work is partially funded by the Swiss National Science Foundation and the German Research Foundation through the BenchFlow: A Benchmark for Workflow Management Systems (DACH Grant Nr. 200021E-145062/1) project.

References

[1] Bianculli, D., et al.: Automated performance assessment for service-oriented middleware: A case study on BPEL engines. In: Proc. of WWW. pp. 141–150 (2010)

[2] Duan, S., et al.: Apples and oranges: a comparison of RDF benchmarks and real RDF datasets. In: Proc. of SIGMOD. Association for Computing Machinery (2011)

[3] Feitelson, D.G.: Workload modeling for computer systems performance evaluation. Cambridge University Press (2015)

[4] Ferme, V., Ivanchikj, A., Pautasso, C.: A framework for benchmarking BPMN 2.0 workflow management systems. In: Proc. of BPM '15. pp. 251–259. Springer (2015)

[5] Ferme, V., et al.: A container-centric methodology for benchmarking workflow management systems. In: Proc. of CLOSER '16. pp. 74–84. Springer (2016)

[6] Gupta, A.: Generating large-scale heterogeneous graphs for benchmarking. In: Specifying Big Data Benchmarks, pp. 113–128. Springer (2014)

[7] Hollingsworth, D.: The workflow reference model. WfMC 68 (1995)

[8] Jordan, D., et al.: Business Process Model And Notation (BPMN) version 2.0. Object Management Group, Inc (2011), `http://www.omg.org/spec/BPMN/2.0/`

[9] Molyneaux, I.: The Art of Application Performance Testing: From Strategy to Tools. O'Reilly Media, 2nd edn. (2014)

[10] Montgomery, D.C., Runger, G.C.: Applied Statistics and Probability for Engineers. John Wiley and Sons (2003)

[11] Muehlen, M., Recker, J.: How much language is enough? Theoretical and practical use of the business process modeling notation. In: CAiSE 2008. pp. 465–479 (2008)

[12] Pietsch, P., Wenzel, S.: Comparison of BPMN2 diagrams. In: Mendling, J., Weidlich, M. (eds.) Business Process Model and Notation, Lecture Notes in Business Information Processing, vol. 125, pp. 83–97. Springer (2012)

[13] Röck, C., et al.: Performance benchmarking of BPEL engines: A comparison framework, status quo evaluation and challenges. In: Proc. of SEKE. pp. 31–34 (2014)

[14] Skouradaki, M., Andrikopoulos, V., Leymann, F.: Representative BPMN 2.0 process model generation from recurring structures. In: Proc. of ICWS '16 (2016)

[15] Skouradaki, M., et al.: Micro-benchmarking BPMN 2.0 workflow management systems with workflow patterns. In: Proc. of CAiSE '16. pp. 67–82. Springer (2016)

[16] Skouradaki, M., et al.: RoSE: Reoccurring structures detection in BPMN 2.0 process models collections. In: Proc. of CoopIS. Springer Berlin Heidelberg (2016)

[17] Vicknair, C., et al.: A comparison of a graph database and a relational database. In: Proc. of ACM SE'10. Association for Computing Machinery (ACM) (2010)

[18] Wetzstein, B., et al.: Monitoring and analyzing influential factors of business process performance. In: Proc. of EDOC '09. pp. 141–150 (2009)