A Framework for Benchmarking BPMN 2.0 Workflow Management Systems

Vincenzo Ferme, Ana Ivanchikj, Cesare Pautasso

Faculty of Informatics, University of Lugano (USI), Switzerland
{vincenzo.ferme,ana.ivanchikj,cesare.pautasso}@usi.ch
http://benchflow.inf.usi.ch

Abstract. The diverse landscape of Workflow Management Systems (WfMSs) makes it challenging for users to compare different solutions to identify the ones most suitable to their requirements. Thus a comparison framework that would define common grounds in many different aspects, such as price, reliability, security, robustness and performance is necessary. In this paper we focus on the performance aspect, and we present a framework for automatic and reliable calculation of performance metrics for BPMN 2.0 WfMSs. We validate the framework by applying it on two open-source WfMSs. The goal is to contribute to the improvement of existing WfMSs by pinpointing performance bottlenecks, and to empower end users to make informed decisions when selecting a WfMS.

Keywords: BPMN 2.0 · Workflow Management Systems · Benchmarking

1 Introduction

With the growth in the variety of Workflow Management Systems (WfMSs) companies face a difficult decision when selecting a suitable system for their Business Process Management (BPM) projects. The main differences among WfMSs consist of: the supported executable modeling languages (e.g., WS-BPEL, BPMN 2.0), the integration with external systems and tools (e.g., Web service APIs for monitoring and business intelligence [3]), their performance [12], robustness [6], operating costs and other non-functional requirements. In this paper, we address the need to assess alternative WfMSs based on the runtime performance of the Business Process (BP) execution by their Workflow Engine (WfE). Such assessment would empower BPM adopters to map their requirements to the available solutions and BPM vendors and developers to improve their technology offerings. Our framework for benchmarking BPMN 2.0 WfMSs ensures the reliability of the benchmarking process. It does so by structurally defining and controlling the environment under which the performance experiments are carried out and their results analyzed, while taking into account the main requirements of a good benchmark [5]. The decision to focus on BPMN 2.0 (BPMN2 hereinafter) is due to its growing support by commercial and academic WfMSs. As initial validation of the framework we use it for a simplistic performance test of two WfMSs.

2 Related Work

The need for benchmarking WfMSs has been identified many times. In 2000, Weikum et al. [4] propose a benchmark for comparing the performance of different commercial WfEs by measuring their throughput to study the impact of the database component. Ten years later, SOABench [1] defines one of the first performance comparisons for WS-BPEL WfEs. It assumes that the performance of a WS-BPEL WfE can be reduced to its response time. A recent review by Röck et al. [12], of the work undertaken so far for benchmarking WS-BPEL WfEs, highlights: the lack of an extensive evaluation of different WfEs, the unclear definition of the workload mix (i.e., the mix of process models executed by the System Under Test (SUT)) and the load functions (i.e., the functions describing how the requests to start and interact with the BP instances are sent to the SUT), as well as the narrowly focused metrics for characterizing system's performance. There exist many commercial and open-source performance measurement frameworks [11], some dedicated to generic Web Applications (e.g, Faban (`http://faban.org`), JMeter (`http://jmeter.apache.org`)), and others to middlewares [1], [8]. However, to the best of our knowledge, no ready-to-use open-source solutions exist for comprehensive benchmarking of BPMN2 WfMSs. There exist similar tools [1], [7], however they do not implement performance benchmarking (e.g., Betsy [7]), or they use virtual machines to deploy the SUT (e.g., SOABench [1]), introducing additional overhead possibly impacting the performance.

3 The BenchFlow Benchmarking Framework

As opposed to other software systems, benchmarking WfMSs introduces additional challenges derived from: 1) the system deployment complexity due to their distributed nature; 2) the high number of configuration parameters affecting their performance; 3) the absence of a standard interface to interact with the WfMSs and to access execution data; 4) the asynchronous execution of the processes; and 5) the complexity of the execution behaviours that can be expressed by modern modeling and execution languages such as BPMN2. The system deployment complexity and the high number of configuration options require to integrate the configuration and the deployment of the SUT, i.e., the WfMS, as part of the performance test definition. It is the only way to scale out the large number of tests needed to comprehensively evaluate the WfMSs performance. The absence of a standard interface makes abstractions necessary for handling many different WfMSs by the framework. The asynchronous execution of processes makes it challenging to collect the measures directly on the SUT side since the client is only aware of the response time of the requested task's queuing, whereas it remains unaware of its actual execution status and execution time. The complexity of the possible behaviours that can be expressed needs to be tackled by proposing a model-driven approach that, starting from a BPMN model and a set of WfMS configurations, has to instantiate the infrastructure necessary to perform the test.

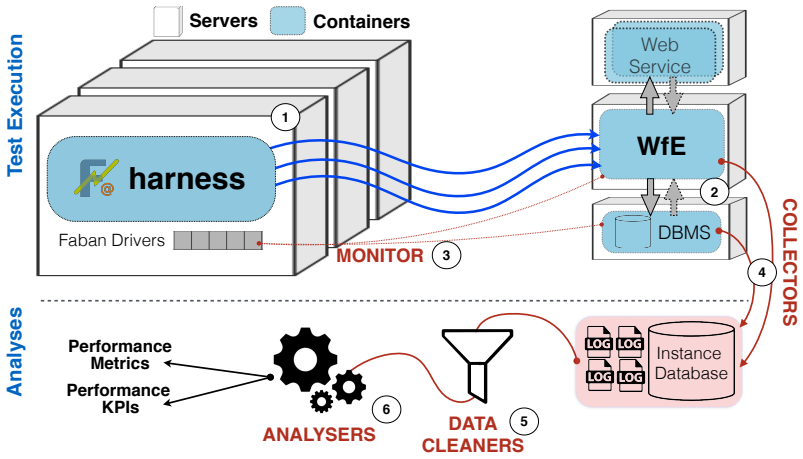

Fig. 1: BenchFlow Framework Architecture

The BenchFlow framework builds upon Faban, an established and tested "performance workload creation and execution framework", to design the load drivers and issue the load to the WfMS. The load driver, which is executed by the harness, provides the infrastructure needed to define the simulated users and their interaction with the WfE. BenchFlow also exploits Docker (`https://www.docker.com`) as a containerization technology, to enable the automatic deployment and configuration of the WfMS and to ensure that the experimental results can be reproduced. As shown in Fig.1, a WfMS can be tested with BenchFlow only if: 1) its API can be used for automation of the test execution [13]. The API should feature: deployment of BP models, start of a BP instance execution, access to the list of pending user and manual tasks and ability to complete them, and sending events to the running BP instances; 2) its logs or instance database (DB) include performance data about the BP instances' execution. These can be queried to calculate WfMS performance metrics. BenchFlow aims to minimise the effort of adding a new WfMS to the framework and executing performance tests on it. To add a new WfMS, users need to: provide a containerized version of the WfMS (this should be published in a public registry if the results are meant to be reproduced), integrate the custom WfE interfaces, provide the queries that BenchFlow can run to assess the completion of the BP instances execution and to extract the raw performance data.

3.1 Performance Test Execution

The WfMSs are automatically deployed and undeployed using Docker (Fig. 1.1 and 1.2). Each component involved in the benchmark (the WfE, the instance DB, the Web services and Faban load drivers) are packaged as Docker images to be deployed and executed on different servers connected by a dedicated local network so that interferences are minimized. Docker enables the repeatability of the benchmark execution, since it freezes the system state so that every test

runs from exactly the same initial conditions. Containerization technologies introduce some overhead in system's performance that can be detrimental for the performance tests. However a recent reliable performance analysis of Docker [2] has shown that, if carefully configured, Docker reaches near-zero overhead.

After the WfMS is deployed, the workload can be applied. The workload is defined by standard BPMN2 models and BenchFlow provides already defined workload packages and artifacts. These consist of a mix of BPMN2 models that have to be deployed in the WfE. They are characterized by their features (e.g., which BPMN2 language constructs they use), and the simulated behaviour of the users instantiating them and interacting with them, as well as the simulated behaviour of the Web Services called by the WfE. Each WfMS uses a custom mechanism for deployment, instantiation and interaction with tasks. We abstract common interaction interfaces, and then map them to the actual ones implemented by each WfMS. Faban drivers issue the load to the WfMS, and we expose its API by means of a Domain Specific Language (DSL), similarly to what has been done in other performance frameworks and generic application performance testing (e.g., `https://github.com/flood-io/ruby-jmeter`). The use of a DSL simplifies the definition of a reusable workload package encapsulating the simulated behaviour of the interacting users and external services.

3.2 Performance Analyzes

BenchFlow automatically collects all the data needed to compute performance metrics, and to check the correct execution of the tests (e.g., errors by different WfMS components). The client-side data (e.g., the response time of BP instance start requests) are collected by Faban and integrated with the server-side data collected by BenchFlow. The server-side data is collected from the execution logs from all the different containers realizing the WfMS, as well as from the instance DB populated by the WfMS during the test execution. In order to avoid interferences during the test execution, we collect (Fig. 1.4) all the data only after the WfMS completes the execution of the issued load. This is determined by first monitoring the CPU utilization of the running Docker container, and then when the container is idle, by checking if the number of completed BP instances matches the number of BP instances started by the load driver (Fig. 1.3). We exploit the logs to identify execution errors, and containers' statistics (obtained through Docker *stats* API) and the instance DB data to compute the performance metrics included in BenchFlow (Fig. 1.6). Each WfMS has its own internal representation and structure for the logs and the instance DB data. In order to define the metrics computation and the performance analyzes only once for all WfMSs we map these logs and data to a uniform representation (Fig. 1.5).

4 Evaluation: Preliminary Scalability Experiment

In this section we present the results of using BenchFlow to evaluate the scalability of two open-source WfMSs as the number of users increases. The load issued to the WfMS and the selected performance metrics are not exhaustive.

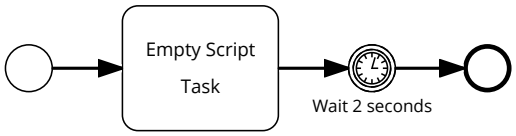

Fig. 2: Experiment Business Process Model

4.1 Experiment Description and Set-up

This performance testing experiment is based on the following elements [11]:

1) *Workload:* the instances of the BP models set executed by the WfE during the experiment. Given the limited scope of this experiment we use only one simple BP model presented in Fig. 2. The Script task is an empty script. The Timer event defines a wait time of 2s before allowing the process to continue.

2) *Load Function:* the function handling system's load. In our case the Load function determines how the BP instances are initiated by a variable number of simulated users (since we are testing system's scalability), growing from 5 to 150 concurrent users. Due to experiment's simplicity the load driver executes the Load function only once when the test starts. The duration of the load function is 300s with 30s of *ramp-up period*. The ramp-up period defines the transition from none to all simulated users being active. This means that it takes 30s before all users start issuing requests for BP instance instantiation. For example, in an experiment with 5 users, a new simulated user is created every 6s during the ramp-up period. After becoming active each user issues one BP instance start request per second.

3) *Test environment*: the characteristics of the hardware used to run the experiment. We use three servers (Fig. 1): one for the harness executing the load driver, one for the WfE and one for the Database Management System (DBMS). We deploy the WfE on the least powerful machine (12 CPU Cores at 800Mhz, 64GB of RAM) to ensure that the machine where we deploy the Load driver (64 CPU Cores at 1400MHz, 128GB of RAM) can issue sufficient load and that the DBMS (64 CPU Cores at 2300MHz, 128GB of RAM) can handle the requests from the WfE. After each test we verify the absence of measurement noise by checking the environment metrics (CPU, RAM and network usage) and the WfE logs to ensure that all the BP instances are completed.

We run the experiment on two open-source WfMSs supporting native execution of BPMN2. We test them on top of Apache Tomcat 7.0.59 using Oracle Java 8 and MySQL Community Server 5.5.42. We use the default configuration as specified on vendors' websites.

4.2 Results

The first metric we analyze is the $Throughput = \frac{\#BPInstances(bp)}{Time(s)}$ [9, ch. 11]. As per Fig. 3 Engine B does not scale well after 25 and the throughput starts degrading after 50 users. Engine A can handle a load up to 125, with the throughput decreasing abruptly with 135-150 users. Thus the Capacity of the WfEs can be estimated to less than 135 for Engine A and less than 50 users for Engine B.

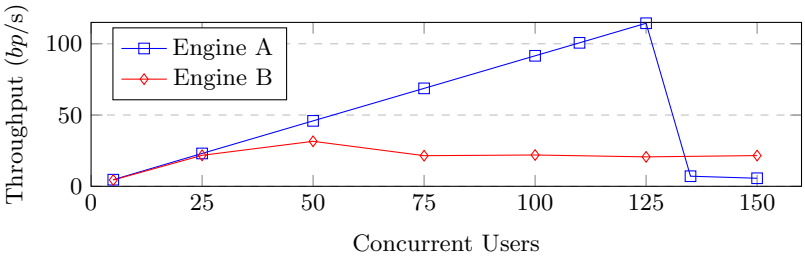

Fig. 3: Throughput

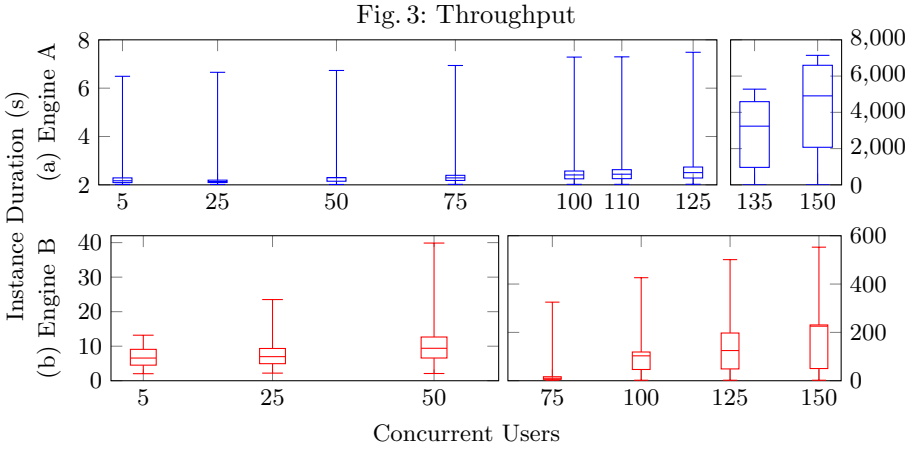

Fig. 4: Aggregated Process Instance Duration Comparison

The BP *instance duration* is the time difference between the start and the completion of a BP instance. It is presented in the box and whisker plot in Fig. 4(a) for Engine A and Fig. 4(b) for Engine B. This type of plot displays the analyzed data into quartiles where the box contains the second and third quartile, while the median is the line inside the box. The lines outside of the box, called whiskers, show the minimum and maximum value of the data [10]. The measurements show that Engine A scales better since it starts having an unexpected behaviour after 125 concurrent users, while the first execution performance problems of Engine B appear at 50 users, as evident from the instance duration increase of one order of magnitude. In Fig. 5 we report Engine A's CPU utilization for each of the tests. It is interesting to notice that while the instance duration increases substantially starting from 135 concurrent users (Fig. 4 (a)), the CPU utilization decreases, indicating that the slowdown of the WfE is not caused by lack of resources. The same has been verified by checking the CPU/RAM utilization of the DBMS.

After noticing a bottleneck in performance scaling, we investigate the causes. Since only two constructs, a Script task and a Timer event, are used in the experiment BP model, we test the WfE performance in handling each of them individually. The test processes used consist of a Start event, the tested construct and an End event. As per the previously gathered information we focus on the critical number of users (125/135 for Engine A and 25/50 for Engine B). We use the *delay* metric which compares the expected to the actual duration of the

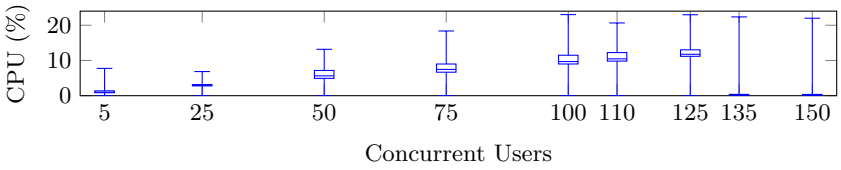

Fig. 5: Aggregated CPU Usage (Engine A)

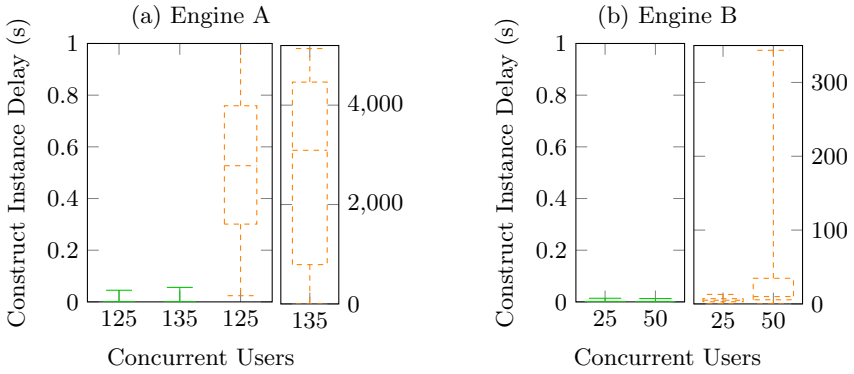

Fig. 6: Script Task (——) and Timer Event (- - - -) feature comparison

construct execution. The expected duration of the Timer is 2s, while the empty Script task should take 0s to complete. The delay measurements (Fig. 6) show that both WfEs handle the Script task efficiently with an average delay below 10ms. The same does not hold for the Timer. For Engine A, the average delay of the Timer at 135 users is by three orders of magnitude greater than at 125 users. For Engine B, the delay increases by two orders of magnitude between 25 and 50 users. The observed system behaviour could be due to an excessive overhead introduced by concurrently handling many Timers, which could cause growth in the Timers queue thus postponing their execution and increasing their delay.

5 Conclusion and Future Work

The BenchFlow framework greatly simplifies the performance benchmarking of BPMN2 WfMSs, by abstracting the heterogeneity of their interfaces and automating their deployment, the data collection and the metrics and Key Performance Indicators (KPIs) computation. It does so by relying on Faban and Docker, and by verifying the absence of noise in the performance measurements. While the complexity of BPMN2 makes it challenging to benchmark the performance of the WfMSs implementing it, the benefits of doing so are evident. The first experimental results obtained with a simple BP model running on two popular open-source WfMSs have uncovered important scalability issues. We have discussed the identified performance bottlenecks with the WfMS vendors who have clarified the probable cause. Namely, in Engine A we have used a different DBMS configuration in the setup of the system. In Engine B the goal of the

default configuration is a fast setup, not optimisation. The discussion has emphasised the need of defining a systematic methodology for obtaining a ready to use setup of the WfMS from the vendors. Developers can use these results to improve the WfMSs, while end users can decide which WfMS to deploy depending on how many concurrent users they have, and carefully set their configuration.

As a next step we plan to release the framework to the community to gain empirical evidence about its benefits and limitations. We will also continue the experiments to measure the performance of additional real-world WfEs. The ultimate goal is to give a fair comparison of commercial BPMN2 WfMSs by means of a small set of KPIs [13], which we intend to derive by defining and aggregating a set of raw metrics. We also plan to extend the BenchFlow framework towards other non-functional quality attributes, e.g., reliability, security and robustness.

Acknowledgments This work is partially funded by the Swiss National Science Foundation with the BenchFlow - A Benchmark for Workflow Management Systems (Grant Nr. 145062) project. The authors would like to thank Paul Holmes-Higgin, Frank Leymann, Sebastian Menski, Daniel Meyer, Tijs Rademakers, Bernd Rücker, and Marigianna Skouradaki for the insightful discussions and feedback.

References

1. Bianculli, D., Binder, W., et al.: Automated performance assessment for service-oriented middleware: A case study on BPEL engines. In: 19th WWW. pp. 141–150. ACM, Raleigh (2010)
2. Felter, W., Ferreira, A., Rajamony, R., Rubio, J.: An updated performance comparison of virtual machines and linux containers. Tech. rep., IBM (July 2014)
3. Garro, J.M., Bazán, P.: Constructing and monitoring processes in BPM using hybrid architectures. IJACSA 3(4), 78–85 (2014)
4. Gillmann, M., Mindermann, R., et al.: Benchmarking and configuration of workflow management systems. In: 7th CoopIS. pp. 186–197. Springer, Eilat (2000)
5. Gray, J.: The Benchmark Handbook for Database and Transaction Systems. Morgan Kaufmann, San Francisco, CA, USA, 2nd edn. (1992)
6. Harrer, S., Wirtz, G., Nizamic, F., Lazovik, A.: Towards a robustness evaluation framework for BPEL engines. In: 7th SOCA. pp. 199–206. IEEE, Matsue (2014)
7. Harrer, S., et al.: Automated and isolated tests for complex middleware products: The case of BPEL engines. In: 7th ICSTW. pp. 390–398. IEEE, Cleveland (2014)
8. IBM: IBM Rational Performance Tester. `http://www.ibm.com/software/products/en/performance`
9. Lazowska, E.D., et al.: Quantitative System Performance: Computer System Analysis Using Queueing Network Models. Prentice-Hall, Upper Saddle River (1984)
10. McGill, R., Tukey, J.W., Larsen, W.A.: Variations of box plots. The American Statistician 32(1), 12–16 (1978)
11. Molyneaux, I.: The Art of Application Performance Testing: From Strategy to Tools. O'Reilly Media, Inc., Sebastopol, CA (2014)
12. Röck, C., et al.: Performance benchmarking of BPEL engines: A comparison framework, status quo evaluation and challenges. In: 26th SEKE. Pittsburgh (2014)
13. Skouradaki, M., Roller, D.H., et al.: On the Road to Benchmarking BPMN 2.0 Workflow Engines. In: 6th ICPE. pp. 301–304. ACM, Austin, Texas (2015)