Institute of Architecture of Application Systems

A Container-centric Methodology for Benchmarking Workflow Management Systems

Vincenzo Ferme[2], Ana Ivanchkikj[2], Cesare Pautasso[2], Marigianna Skouradaki[1], Frank Leymann[1]

[1]Institute of Architecture of Application Systems, University of Stuttgart, Germany,
{lastname}@iaas.uni-stuttgart.de
[2]Faculty of Informatics, University of Lugano (USI), Switzerland
{firstname.lastname}@usi.ch

BibTeX:

@inproceedings{INPROC-2016-14,
author = {Vincenzo Ferme and Ana Ivanckikj and Cesare Pautasso and Marigianna Skouradaki and Frank Leymann},
title = {{A Container-centric Methodology for Benchmarking Workflow Management Systems}},
booktitle = {Proceedings of the 6th International Conference on Cloud Computing and Service Science, (CLOSER 2016)},
publisher = {SciTePress},
pages = {74--84},
month = {April},
year = {2016}
}

This publication and contributions were presented at CLOSER 2016
CLOSER 2016 Web site: http://closer.scitevents.org

Universität Stuttgart
Germany

A Container-centric Methodology for Benchmarking Workflow Management Systems

Vincenzo Ferme[1], Ana Ivanchikj[1], Cesare Pautasso[1], Marigianna Skouradaki[2], Frank Leymann[2]
[1]*Faculty of Informatics, University of Lugano (USI), Switzerland*
[2]*Institute of Architecture of Application Systems, University of Stuttgart, Germany*
{*vincenzo.ferme,ana.ivanchikj,cesare.pautasso*}*@usi.ch,* {*skourama,leymann*}*@iaas.uni-stuttgart.de*

Keywords: Benchmarking, Docker Containers, Workflow Management Systems, Cloud Applications

Abstract: Trusted benchmarks should provide reproducible results obtained following a transparent and well-defined process. In this paper, we show how Containers, originally developed to ease the automated deployment of Cloud application components, can be used in the context of a benchmarking methodology. The proposed methodology focuses on Workflow Management Systems (WfMSs), a critical service orchestration middleware, which can be characterized by its architectural complexity, for which Docker Containers offer a highly suitable approach. The contributions of our work are: 1) a new benchmarking approach taking full advantage of containerization technologies; and 2) the formalization of the interaction process with the WfMS vendors described clearly in a written agreement. Thus, we take advantage of emerging Cloud technologies to address technical challenges, ensuring the performance measurements can be trusted. We also make the benchmarking process transparent, automated, and repeatable so that WfMS vendors can join the benchmarking effort.

1 Introduction

Performance benchmarking contributes to the constant improvement of technology by clearly positioning the standards in measuring and assessing performance. Benchmark results conducted under the umbrella of research are recognized as a great contribution to software evolution (Sim et al., 2003). For example, the well-known and established benchmark, TPC-C (Transaction Processing Council (TPC), 1997) had a meritorious contribution to the performance of Database Management Systems (DBMSs), which improved from 54 transactions/minute in 1992 to over 10 million transactions/minute in 2010. The importance of benchmarking is also recognized by the emergence of organisations, such as the Transaction Processing Performance Council (TPC)[1] and the Standard Performance Evaluation Corporation (SPEC)[2], that are responsible for ensuring the integrity of the benchmarking process and results. They maintain steering committees, involving both industry and academic partners.

Widely accepted and trustworthy benchmarks should demonstrate relevance, portability, scalability, simplicity, vendor-neutrality, accessibility, repeatability, efficiency, and affordability (Gray, 1992; Huppler, 2009; Sim et al., 2003). These requirements impose significant technical challenges (Pautasso et al., 2015) regarding the automation of the benchmarking process, in order to facilitate efficiency and scalability, i.e., the benchmarking of different systems; the repeatability of tests under the same initial conditions; the reduction of noise from the environment, etc. Benchmarks also often face logistic challenges, since many vendors[3] use software licence agreements to tightly control how benchmark results, obtained using their products, are published. Addressing such challenges is only possible with a carefully designed methodology to manage the entire benchmarking process, from the deployment of the targeted system, through its testing, and up till the publishing of the results.

In this paper, we propose a methodology for benchmarking the performance of Workflow Management Systems (WfMSs). As defined by the Workflow Management Coalition (WfMC), a WfMS is "a system that completely defines, manages and executes

[1] https://www.tpc.org/
[2] https://www.spec.org/
[3] Like Oracle: http://www.oracle.com/technetwork/licenses/standard-license-152015.html and Microsoft: http://contracts.onecle.com/aristotle-international/microsoft-eula.shtml

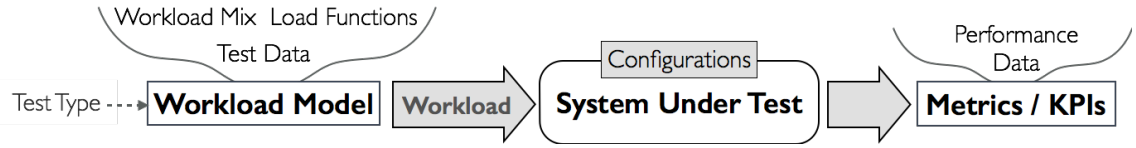

Figure 1: Benchmarking Input/Process/Output (IPO) Model

'workflows' through the execution of software whose order of execution is driven by a computer representation of the workflow logic" (Hollingsworth, 1995). The WfMSs can use different languages to execute the modeled workflows (e.g., WS-BPEL, BPMN 2.0). Even though the proposed methodology is not limited to any specific execution language, we use the Business Process Model and Notation (BPMN) 2.0 (Jordan and Evdemon, 2011) as a use case for describing its feasibility. There are several reasons behind such decision: the growing usage of BPMN 2.0 and the WfMSs that support it (Skouradaki et al., 2015) (e.g., Camunda, Activiti, jBPM, Bonita BPM[4]), the ISO standardization (ISO/IEC 19510:2013) of the language and above all the fact that BPMN 2.0 supports a superset of the elements supported by BPEL (Leymann, 2011).

Our methodology aims to offer solutions to the aforementioned: 1) technical challenges by defining a benchmarking framework which benefits from the emerging containerization technology, initially used for deployment of Cloud applications (Turnbull, 2014); and 2) logistic challenges by formalising the interaction between the conductor of the benchmarking process (hereinafter the BenchFlow team) and the vendors of the WfMSs which are being benchmarked. We believe that such formalisation will foster vendor collaboration and public access to the benchmarking results. In our approach, we treat the WfMSs as a black-box, as the proposed methodology is applicable to any WfMS released in a Docker container. Our current goal is to benchmark the performance of the WfMSs under different load and observe their behaviour in normal conditions (e.g., we do not inject fault during the workload execution). Furthermore, we assume that the execution bottlenecks lie with the WfMS, and not with the external services invoked during the process instance execution. We ensure this assumption holds when designing the workload, or when running the experiments.

Our work is conducted under the scope of the BenchFlow project[5] which has the aim of defining a standard way to assess and compare WfMSs performance (Skouradaki et al., 2015).

[4]More detailed list is provided at: https://en.wikipedia.org/wiki/List_of_BPMN_2.0_engines

[5]http://benchflow.inf.usi.ch

The remainder of the paper is organized as follows: Section 2 introduces the benchmarking framework, Section 3 presents the process of formal interaction with WfMS vendors and the used artifacts, Section 4 examines the related work, and Section 5 concludes and presents the planned future work.

2 Benchmarking Process

To ensure benchmark's transparency and repeatability, it is crucial to clearly define the benchmarking process, its input and output artifacts, as well as the implementation of the benchmarking framework.

2.1 Benchmarking Input/Process/Output Model

Regardless of the targeted System Under Test (SUT), the benchmarking process always uses as an input a workload, i.e, an instance of the workload model, whose parameters are influenced by the test type. It is a good practice to explore SUT performance using different SUT configurations and different computational resources. The output of the benchmarking process is the data for calculating different metrics and/or Key Performance Indicators (KPIs). Such data should be recorded only after a warm-up period, necessary to remove system initialization bias. Before the data can be used for computing metrics and KPIs, it must be validated, by running the same tests multiple times, to ensure that the standard deviation in the results is below a predefined limit. In the following subsections we define the elements of the Input/Process/Output (IPO) model (Fig. 1) in the WfMSs context.

2.1.1 Workload Model

The parameters of the workflow model (workload mix, load functions, and test data) vary depending on the objective of the test type (e.g., load test, stress test, soak test (Molyneaux, 2014)). The interactions between the parameters of the workload model are presented in Fig. 2.

The *workload mix* is the mix of process models to be deployed during a given test. Each process model defines the activities, the order in which they

are performed, as well as any external interactions. It is crucial for benchmark's validity that the workload mix is representative of real-world models and the set of language constructs that they use. Collecting and synthesizing real-world models is one of the challenges faced when benchmarking WfMS performance, to be addressed by static and dynamic analysis of real-world model collections (Pautasso et al., 2015). Given the fact that the workflow execution languages are very rich and complex, the constructs included in the workload mix models need to be carefully selected. They should: 1) stress the performance of the WfMS; 2) be supported by the WfMS (Geiger et al., 2015); 3) be frequently used in real-world processes; and 4) be in-line with the targeted performance tests and KPIs. To make sure that our workload mix is representative, we will follow the iterative approach for its development and evaluation (Skouradaki et al., 2015). Since each WfMS uses different serialization of executable process models, every time we add a new WfMS to the framework, our workload mix needs to be adapted to a compatible executable format.

The *load functions* describe how the workload is issued to the SUT. For example, in the case of Online Transaction Processing (OLTP) systems, the workload mix consists of transactions against a database (DB) and a load function describes the number of queries that will be executed against the DB per time unit (Transaction Processing Council (TPC), 1997; Transaction Processing Performance Council, 2007). In more complex examples, such as the session-based application systems, the workload mix is defined through different types of transactions, and different roles of users that execute these transactions. In these systems, the load functions are more complex as they have to describe the behavior of the users, i.e., the frequency and sequence in which they execute different types of transactions (van Hoorn et al., 2014). The workload model for benchmarking WfMSs can be seen as an extension of the workload model for session-based applications, since the WfMS executes blocks of language constructs as transactions which can be compensated in case of an error. When

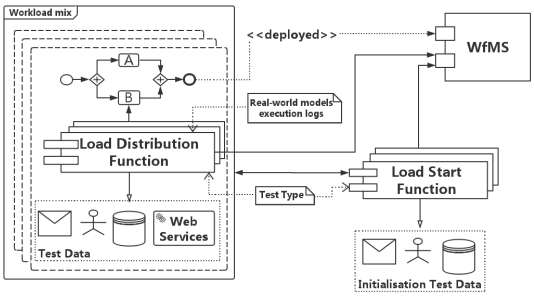

Figure 2: Workload Model Parameters' Interactions

benchmarking WfMSs we distinguish two types of load functions: 1) *Load Start Functions* - determined based on the performance test type, they define how often process instances are being initiated during the test. Even though a Load Start Function needs to be defined for each process model in the workload mix, it is possible that multiple or all process models share the same Load Start Function; 2) *Load Distribution Functions* - contain rules to define the distribution of the executed paths of a given process model in the workload mix and the distribution of the interactions. Examples of such interactions are the completion of a human (user or manual) task and the call of an external Web service. The rules will be derived from the analysis of real-world execution logs and based on the test type. There is mutual dependency between all the instances of a given process model, and the Load Start Function for that process model, marked with bidirectional arrow in Fig. 2. Namely the Load Start Function determines the number of instances for a given process model in the workload mix, but at the same time, the Load Start Function needs to know the process model to be able to generate the test data necessary to instantiate a process instance.

The *test data* are used as an input for starting a process instance, or during the execution of the process instance as part of the performance test. They depend on the definition of the load functions and can refer to, for example, results of evaluating gateway rules, messages for invoking Web services, persistent data required for completing a task, the number of users, etc.

To summarise: the *workload* applied to the SUT is determined by the load functions, used to generate process instances of all the process models included in the workload mix. The process instances are instantiated and executed using the test data selected in accordance with the load functions. For instance, we could use the process model shown in Fig. 2 as a simplistic workload mix. Consider Task A a human task and Task B a Web service task. We can define the Load Start Function, such that, 100 process instances are started with the initialization event and then executed by the WfMS. The test data in this case, would be the input data that the users are entering into the WfMS for Task A. The Load Distribution Function would define users' think times, and the time it takes for the Web service to complete.

2.1.2 System Under Test

The SUT refers to the system that is being tested for performance. In our case a WfMS which, according to the WfMC (Hollingsworth, 1995), includes the Workflow Enactment Service (WES) and any envi-

		Functionality	Min Response Data
Core APIs	**Initialisation APIs**	Deploy a process Start a process instance	Deployed process ID Process instance ID
Non-core APIs	**User APIs**	Create a user Create a group of users Access pending tasks Claim a task* Complete a task	User ID User group ID Pending tasks IDs
	Event APIs	Access pending events Issue events	Pending events IDs
	Web service APIs	Map tasks to Web service endpoints	

*Optional depending on the WfMS implementation

Table 1: Summary of Core and Non-core APIs to be implemented by the WfMS

ronments and systems required for its proper functioning, e.g., application servers, virtual machines (e.g., Java Virtual Machine) and DBMSs. The WES is a complex middleware component which, in addition to handling the execution of the process models, interacts with users, external applications, and Web services. This kind of SUT offers a large set of configuration options and deployment alternatives. The DBMS configuration as well as the WES configurations (e.g., the granularity level of history logging, the DB connection options, the order of asynchronous jobs acquisition), may affect its performance. Moreover, WfMSs often offer a wide range of deployment alternatives (e.g., standalone, in a clustered deployment, behind a load balancer in an elastic cloud infrastructure) and target different versions of application server stacks (e.g., Apache Tomcat, JBoss).

To add a WfMS in the benchmark, we require the availability of certain Core and Non-core APIs. A summary of the same is presented in Table 1.

The ***Core APIs*** are *Initialisation APIs* necessary for automatic issuing of the simplest workload to the SUT in order to test SUT's performance. They include: 1) deploy a process and return as a response an identifier of the deployed process (*PdID*); and 2) start a process instance by using the *PdID* and return as a response the new instance identifier (*PiID*).

Depending on the execution language of the WfMS and the constructs that it supports, other ***Non-core APIs*** might be necessary for testing more complex workloads. For instance, if we are targeting BPMN 2.0 WfMSs we might also require the following APIs:

For applying workloads involving human tasks, the following *User APIs* are necessary: 1) create a user and return the identifier of the created user (*UsID*); 2) create a group of users, return the created group identifier (*UgID*), and enable adding users by using their *UsIDs*; 3) pending user/manual tasks: access all the pending user/manual task instances of a given user/manual task identified by its *id* (Jordan and Evdemon, 2011, sec. 8.2.1) as specified in the model serialization. We want to obtain all the pending tasks with the given *id* of all the process instances (*PiIDs*) of a given deployed process (*PdID*). The API has to respond with data, enabling the creation of a collection that maps the process instances to the list of their pending tasks <*PiID*, *UtIDs*> and <*PiID*, *MtIDs*>; 4) claim a user/manual task identified by *UtID*/*MtID*, if tasks are not automatically assigned by the WfMS; and 5) complete a user/manual task identified by *UtID*/*MtID* by submitting the data required to complete the task.

To issue a workload containing process models with catching external events, the following *Event APIs* are necessary: 1) pending catching events/receive tasks: access the pending catching event/receive task instances of a given event/task identified by its *id* (Jordan and Evdemon, 2011, sec. 8.2.1) specified in the model serialization. We want to obtain all the pending catching events/receive tasks with the given *id* of all the process instances (*PiIDs*) of a given deployed process (*PdID*). The API has to respond with data enabling the creation of a collection that maps the process instances to the list of their pending catching events/receive tasks <*PiID*, *CeIDs*> and <*PiID*, *RtIDs*>; and 2) issue an event to a pending catching event/receive task identified by using *CeID*/*RtID*. We require the APIs to accept the data necessary to correlate the issued event to the correct process instance, e.g., a correlation key.

Finally, to be able to issue a workload defining interaction with Web services and/or containing throwing events, the WfMS has to support a binding mechanism to map each Web service task/throwing event to the corresponding Web service/throwing event endpoint. The WfMS should preferably allow to specify the mapping in the serialized version of the model, so that the binding can be added before deploying the process.

Since many WfMSs are offered as a service, it is safe to assume that many WfMSs expose, what we call, the Core APIs. In our experience with systems we have evaluated so far (e.g., Activiti, Bonita BPM, Camunda, Imixs Workflow, jBPM), they support not only the core APIs, but also the non-core APIs. The exact API may differ among systems, however the necessary API features were always present.

2.1.3 Metrics and Key Performance Indicators

The output of the performance testing is collected and later used to calculate a set of metrics and KPIs.

To make performance testing meaningful, they must be carefully selected in order to capture the differences among configurations and systems. A metric is defined as a "quantitative measure of the degree to which a system, component or process possesses a given attribute" (Fenton and Bieman, 2014). A KPI on the other hand is a "a set of measures" (Parmenter, 2010, ch. 1) which focus on the critical aspects of SUT's performance. As opposed to metrics, the number of KPIs should be limited, in order to clearly focus the improvement efforts. We propose grouping the metrics and KPIs in three different levels, based on the requirements of their expected users: 1) Engine-level: to help end-users select the most suitable WfMS as per their performance requirements; 2) Process-level: suitable for WfMS vendors to position and evaluate their offerings in different use-case scenarios; and 3) Feature-level: fine-grained measures allowing WfMS developers to deal with system's internal complexity and to explore its bottlenecks. For each level, we enumerate a non-exhaustive set of metrics to be calculated using process execution, environment monitoring and resource utilization data. Which metrics will be calculated when running a specific performance test, will depend on the objective of the test. To increase transparency, a detailed definition of each of the used metrics will be provided to the WfMS vendors, together with the benchmark results.

The *Engine-level* metrics measure the WfMS performance based on the execution of the entire test workload. Most metrics defined in the literature for assessing WfMS performance, refer to this level and include throughput, latency, resource utilization (e.g., RAM, CPU, disk and network utilization) (Röck et al., 2014). In addition to these metrics, we propose to also add metrics for assessing the scalability, the capacity (Jain, 1991, p. 123), and the flexibility of the SUT. For example, response time (ms) - what is the time it takes for the SUT to respond to the service request (Barbacci et al., 1995, ch. 3), endurance - can the SUT sustain a typical production load (Molyneaux, 2014, p. 51), flexibility to spike - how does the SUT react to a sudden increase of the load (Molyneaux, 2014, ch. 3).

The *Process-level* metrics measure the WfMS performance based on the execution of instances of a specific process model in the test workload. The resource utilization metric is interesting at the process-level as well, to analyse how much resource consumption is needed per single process instance execution. We also propose metrics for the duration of a process instance, and the delay introduced by the WfMS in executing the process instance. The delay is computed as a difference between the actual and the expected duration of the process instance, where the expected duration is determined as the aggregation of the expected duration of the constructs included in the executed path of the process instance.

Different languages can support different concepts (e.g., time, parallelism, control and data flow), thus impacting the design of the WfMSs executing them. The *Feature-level* fine-grained metrics are necessary to determine the impact of specific language constructs or patterns on WfMS performance, and to isolate possible sources of performance issues. We propose measuring the resource usage, the duration, and the delay, for each language construct as well as for frequently used patterns[6]. For instance, in BPMN 2.0, such constructs can include user task, timer event, exclusive gateway, etc. The ultimate goal is to help developers determine the performance bottlenecks of their WfMS, so that they can work on removing them, and thus improving the WfMS performance in general. We have used some of the mentioned metrics to evaluate the BenchFlow framework in (Ferme et al., 2015).

However, being able to calculate the above mentioned metrics imposes certain requirements on the granularity and the type of data to be saved in the DBMS. The DBMS persists and recovers the process instance execution status and history. The most important data to be recorded are the following timestamps: start time and end time of each executed process instance (identified by its *PiID*); as well as start time and end time of specific language constructs that are part of the process instances, e.g., activities or events identified by their *id* with a possibility to map them to the process instance they belong to. The timestamps should preferably have a precision up to milliseconds.

2.2 Benchmarking Framework Implementation

The goal of the benchmarking framework is twofold: 1) to make our research publicly available, and foster collaboration; and 2) to enable reproducibility and full transparency in the benchmarking process. To do so we have released the framework as open source[7]. The second goal is very important for inducing WfMS vendors to join our efforts, since it enables them to verify the benchmark results and use the framework with their own infrastructure.

The framework builds on: 1) Faban (Faban, 2014), an established and tested "performance work-

[6] http://www.workflowpatterns.com

[7] https://github.com/benchflow

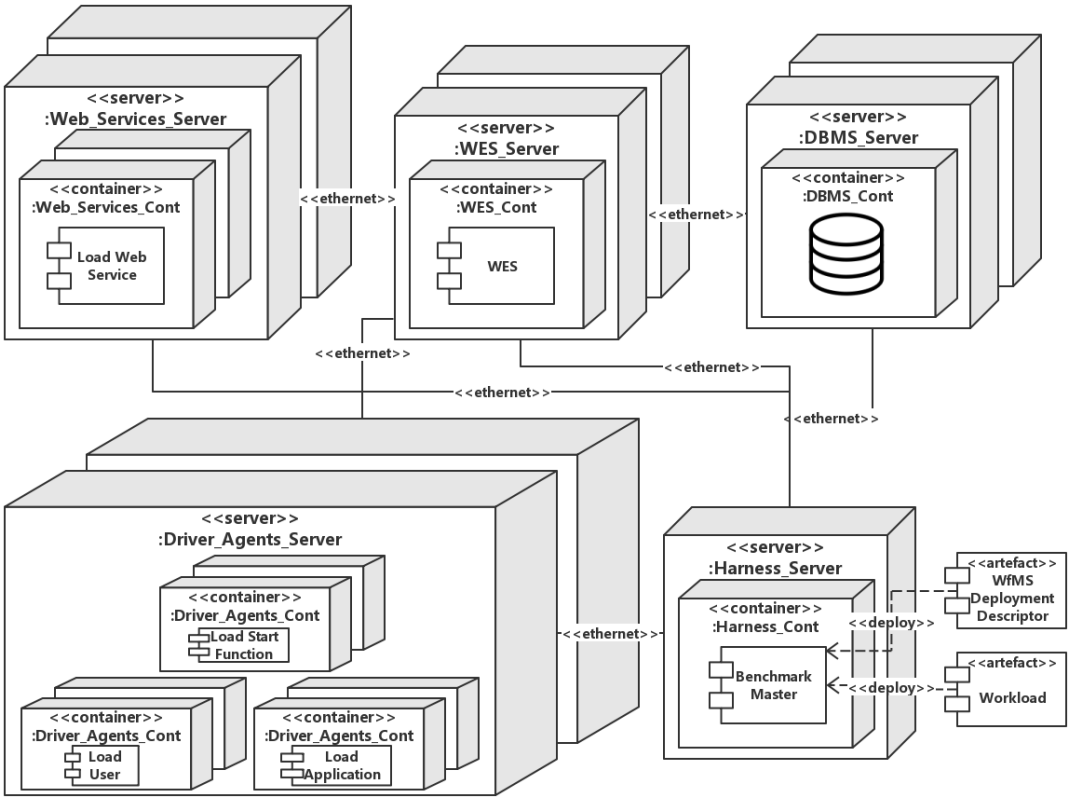

Figure 3: Framework Deployment View

load creation and execution framework", to design the load drivers and issue the load; and 2) lightweight software containerization technologies, precisely Docker (Turnbull, 2014) in the current version of the framework, in order to obtain, from the vendors, a ready to use setup of the WfMSs to be tested. We have decided to adopt Docker since it is a widely used and accepted containerization solution (Merkel, 2014). Moreover, if carefully configured, it has proved to have a negligible impact on the performance of the containerized application (Felter et al., 2014). The benefits of using containerization technologies are: 1) avoiding the highly error-prone task of installing the WfMS that can lead to a non optimal setup impacting its performance results; 2) ensuring the reproducibility of the benchmark results by saving system's initial status (Boettiger, 2014); and 3) collecting detailed resources usage statistics for each Container by means of the Docker *stats* API.

The architecture of the framework has been described in details in (Ferme et al., 2015), while Fig. 3 presents the framework components and their deployment. Each of the framework's components is deployed inside a different Container. Although we use Docker as a containerization technology, the proposed methodology and framework can be applied to any other containerization technology as well. To scale with the amount of performance tests to be included in the benchmark, in order to provide for a reliable comparison of WfMSs, we integrate the deployment of the WfMS in the setup of the performance test. We do so by using a *SUT Deployment Descriptor* and by extending the *Faban Master* in the *Benchmark Master*. The SUT Deployment Descriptor defines the servers on which to deploy the WfMS's Containers. The Benchmark Master automatically deploys the SUT as well as the workload, and all the related entities defined in the load functions, i.e., the Load Start Functions, users and applications which are simulated by means of Faban Driver Agents; and the Web services which are simulated by using techniques for automatic testbed generation, e.g., (Juszczyk et al., 2008). The main deployment requirement for a reliable benchmark is to isolate, as much as possible, the WES from other components, by deploying them on different physical machines on the same local network, with the purpose of minimising the possible noise introduced by their concurrent execution on the same machine (e.g., the DBMS running on the same machine as the WES). To do so, the framework enables the user to select the servers where to deploy the different components of the SUT,

so that there are enough computational resources to generate a load that stresses the WfMS performance, and enough computational resources for the WES to sustain that load. For example, less powerful servers can be used to run the WES, while the load functions and the DBMS can be assigned to more powerful servers. While enhancing the reliability of the obtained results, deploying the components on different physical machines adds network costs to the systems. We mitigate these costs by connecting the different servers on a high performance local network.

The framework also deals with the asynchronous execution of process models, by gathering performance data directly from the DB used by the WES. Data cannot be gathered from the client side since the starting of a new process instance from the Core API is done in an asynchronous way. This means that the Core API call to the WfMS finishes as soon as the instance is created, and the Load Start Function is only aware of the start time of a process instance, while the end time needs to be obtained directly from the DB. By using measurements collected from all the WfMS Containers, the framework ensures that the conditions of the environments that host the WfMS are suitable for benchmark execution (e.g., there are neither interferences from other running applications nor bottlenecks in the underlying hardware). This safeguards the reliability of the environment in which the performance tests are executed.

3 Formal Interaction and Artifacts

Although a carefully designed framework can lead to a benchmark that systematically satisfies the aforementioned desired properties, its industry acceptance is not always a straightforward process. To address this issue, in addition to the technical framework, as part of our methodology, we also propose a process of formal interaction with WfMS vendors (Fig. 4). Its main purpose is to enhance the transparency of the overall benchmarking process, while promoting the engagement of WfMS vendors. The interaction can be initiated either by the BenchFlow team or by the WfMS vendor. Fig. 4 depicts only the desired path. If the vendor decides not to participate in the benchmark, we can still run tests on its WfMS and publish only anonymised results in research papers, provided it is not prohibited by its licence agreement. The rest of this section describes in detail the artifacts exchanged or produced during the benchmarking process.

3.1 Agreement with Vendors

After the vendor becomes familiar with the proposed benchmarking methodology, its WfMS can only be included in the benchmark after a written agreement is signed. The agreement precisely defines the rights and obligations of both parties: the vendor and the BenchFlow team. The main concerns the vendor needs to agree on are:

- defining which versions of the WfMS will be included in the benchmark. Such versions need to be a production stable release of the system and provide at least the Core APIs described in Subsection 2.1.2.
- providing the BenchFlow team with a containerized WfMS for each version to be included in the benchmark, or providing WfMS's installation guide and configuration details to enable the BenchFlow team to prepare the containerized WfMS;
- making the containerized WfMS publicly available on a Container registry or providing the BenchFlow team access to a private registry.
- authorizing the BenchFlow team to publish the obtained results on its website and in research papers using WfMS's name, after the vendor has verified their correctness.

The framework will be publicly available for free use for non commercial purposes, however with limited analytics functionalities. Full analytics and different workload models will be available to vendors which sign the agreement and include their WfMS in the benchmark. Thus they would benefit from detailed insights to possible causes of performance bottlenecks, and performance comparisons to competitors' products or to previous versions of their own product. Namely they could integrate the benchmarking framework in their performance regression testing framework, in addition to their continuous improvement efforts.

3.2 Containerized WfMS

As discussed in Section 2.2, we deploy the WfMS components in different Containers. To isolate the WES from other components, to access relevant data, and to formalize WfMS configuration parameters, we define some requirements for the containerized WfMS. The vendor should provide at least two separate Containers, one for the WES and one for the DBMS. Separate Containers can be provided for other WfMS components as well (e.g., load balancers, workload predictors and autonomic controllers) depending on the WfMS's architecture. The DBMS

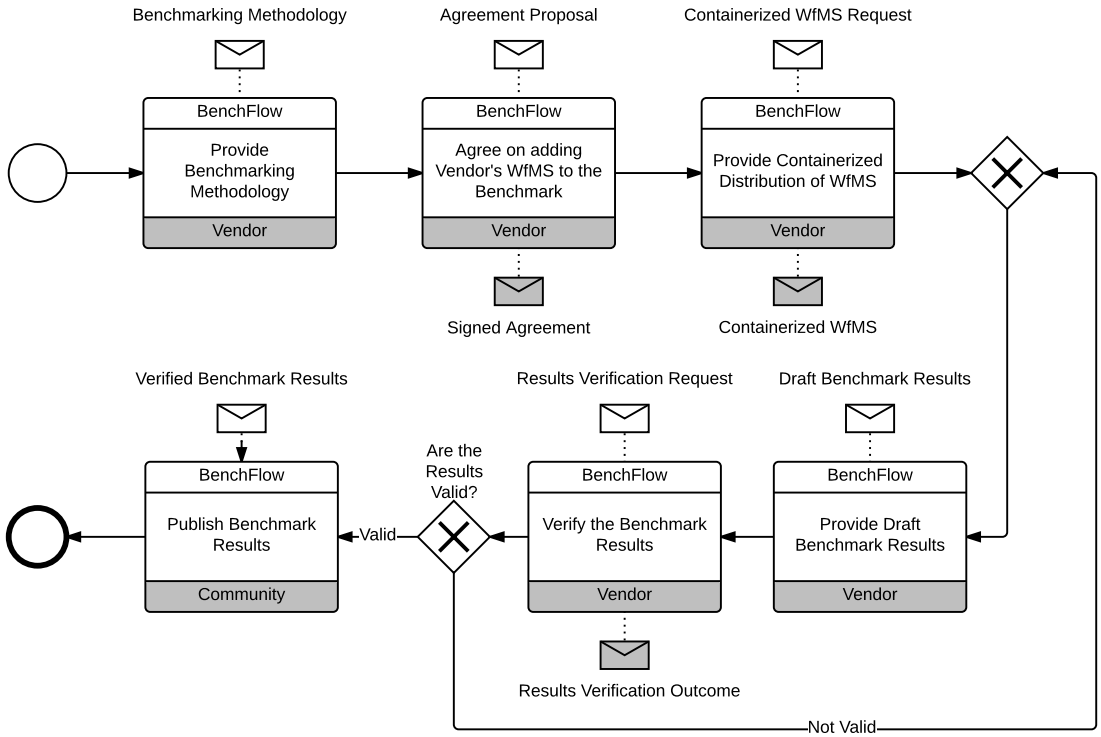

Figure 4: Benchmarking Methodology Choreography

Container can refer to an existing publicly available Container distributions. The containerized WfMS should be publicly available (e.g., at the Docker Hub registry[8]), or the Benchflow team should be granted access to a private registry used by the vendor. The same applies to the Containers' definition file, i.e., the Dockerfile (Turnbull, 2014, ch. 4). While private registries are a solution that can work with vendors of closed-source systems, they impact the reproducibility of the results. For each WfMS version to be included in the benchmark, there must be a default configuration, i.e., the configuration in which the WfMS Containers start without modifying any configuration parameters, except the ones required in order to correctly start the WfMS, e.g., the database connection. However, if vendors want to benchmark the performance of their WfMS with different configurations, for example, the configuration provided to users as a "getting started" configuration, or production-grade configurations for real-life usage, they can also provide different configurations. To do that, the Containers must allow to issue the WfMS configurations through additional environment variables[9], and/or by specifying the volumes to be exposed[10] in order to access the configuration files. Exposing the volumes allows to access the files defined inside the Containers, on the host operating system. Precisely, the WES Container has to, at least, enable the configuration of: 1) the used DB, i.e., DB driver, url, username and password for connection; 2) the WES itself; and 3) the logging level of the WES, and the application stack layers on which it may depend on. Alternatively, instead of providing configuration details, the vendors can provide different Containers for different configurations. However, even in that case enabling the DB configuration is required.

In order to access relevant data, all WfMS Containers have to specify the volumes to access the WfMS log files, and to access all the data useful to setup the system. For instance, if the WfMS defines examples as part of the deployment, we may want to remove those examples by overriding the volume containing them. Moreover, the WES Container (or the DBMS Container) has to create the WfMS's DB and populate it with data required to run the WfMS. The vendors need to provide authentication configuration details of the WfMS components (e.g., the user with admin privileges for the DBMS). Alternatively the vendor may simply provide the files needed to create and populate the DB.

[8] https://hub.docker.com/

[9] https://docs.docker.com/reference/run/#env-environment-variables

[10] https://docs.docker.com/userguide/dockervolumes/#mount-a-host-directory-as-a-data-volume

3.3 Draft Benchmark Results

The draft benchmark results delivered to the WfMS vendor consist of two important sets of information: 1) information necessary for replicating the performance tests, i.e., input and configuration data. They include description of the type of tests that have been run accompanied by all the necessary input for running them, i.e., the parameters of the workload model and the workload as described in Subsection 2.1.1. They also include data regarding the environment setup and deployment, and WfMS configuration; and 2) information and data regarding the metrics and KPIs calculated with the framework, both in terms of precise definition of their calculation and purpose, as well as, in terms of the results of their calculation and possible interpretation of the same. If the calculation of certain metrics/KPIs is not possible, due to lack of data or system support of given language constructs, the status of the metric calculation attempt will be provided in the draft results, together with an explanation of its meaning. The raw performance data, as obtained during the execution, will also be included in the draft results, to enable WfMS vendors to calculate additional metrics not provided in the framework, should it be necessary.

If the WfMS vendors have agreed on including different versions or configurations of their product in the benchmark, they would receive draft benchmark results for each version/configuration. This would enable them, for example, to draw conclusions on how different configurations impact the performance of their product and, if necessary, change their default configuration with a better performing one. They could also control benchmark's website for published results from other WfMS vendors to understand their product's market position.

3.4 Verified Benchmark Results

After receiving the draft benchmark results, the vendors can validate their correctness by replicating the tests and comparing the obtained results. Ideally, no significant differences should be noticed in the results when replicating the tests using the same environment and workload the BenchFlow team had used to perform the initial tests. However, should such differences be identified, the vendor has to report them to the BenchFlow team who is obliged, by the agreement, not to publish results unless they have been verified for correctness by the vendor. Iterative approach will be followed, as evident in Fig. 4, until the results have been verified and thus authorised for publishing at benchmark's website and in research papers. Reasonable time for verification of draft results will be defined in the agreement as per discussion with the vendors.

4 Related Work

To the best of our knowledge, this is the first work proposing a Container-centric benchmarking framework, and providing a full methodology for benchmarking WfMSs. There have been various efforts to outline structured frameworks for the description of benchmarking methodologies applied to diverse applications and environments. Iosup et al. introduce a generic approach for Cloud benchmarking, discuss the challenges in benchmarking Cloud environments, and summarize experiences in the field (Iosup et al., 2014). They also propose a benchmarking tool, SkyMark, for workloads based on the MapReduce model (Dean and Ghemawat, 2008). Baruwal Chhetri et al. propose Smart CloudBench (Chhetri et al., 2015), a platform for automated performance benchmarking of the Cloud. Both of these references propose Infrastructure-as-a-Service (IaaS) benchmark, thus they do not deal with the challenges introduced by benchmarking applications (e.g., the need of an agreement with vendors). Puri presents the challenges in benchmarking applications, and acknowledges that the benchmarking process requires extensive planning and thus a well defined methodology (Puri, 2008). While the aim of the mentioned papers is similar to ours, i.e., to formalize a benchmarking process for emerging technologies and attract industry attention to give feedback and share experiences, the focus is different. In our work we focus on benchmarking a precise middleware, the WfMS.

SPEC and TPC propose a different approach in benchmarking software systems. Their approach can be seen as complementary to the one proposed by us, since it allows the vendors to execute the benchmarks on their own hardware and send back the results. The SPEC/TPC committees verify internally the correctness of the results before publication on their website. Their approach has demonstrated to be effective when a standard benchmark is well defined. Currently we are working on defining such a standard benchmark for WfMSs, thus at this phase, we believe that feedback from different vendors through validation of internally calculated results, might be more effective.

The need to benchmark WfMSs is frequently discussed in literature (Wetzstein et al., 2009; Russell et al., 2007). Gillmann et al. compare the performance of a commercial WfMS with the one developed by the authors, by using a simple e-commerce

workflow (Gillmann et al., 2000). The benchmark measures the throughput of each of the systems as well as the impact of the DB work that is forwarded to a dedicated server. Bianculli et al. propose a more systematic approach in the SOABench (Bianculli et al., 2010b), where they tackle the automatic generation, execution and analysis of testbeds for testing the service-oriented middleware performance. That framework is used to compare the response time of several Web Services Business Process Execution Language (WS-BPEL) WfMSs, i.e., ActiveVOS, jBPM, and Apache ODE (Bianculli et al., 2010a), when using a different number of clients and different think times between subsequent requests. The results have pointed to some scalability limitations of the tested systems.

The limitations of the above projects, and other work in the area (Röck et al., 2014; Skouradaki et al., 2015; Daniel et al., 2011) lie in the fact that: 1) they target a small number of WfMSs; 2) their workload mix consists of simple, non-representative processes; 3) their performance tests are limited to, e.g., response time, or load testing; 4) the performance metrics and KPIs used are not focused on WfMS, but on software systems in general; and 5) they focus on benchmarking framework architecture, while ignoring the methodological issues discussed in this paper. Thus they do not satisfy the requirements of standard benchmarks, but can be seen more as custom benchmarks. Our methodology aims at addressing these limitations, while being applicable on different industry use cases.

There are many commercial and open-source performance framework solutions (Molyneaux, 2014), some dedicated to generic web applications (Faban, 2014; The Apache Software Foundation, 2015; SmartBear, a), and others to specific middleware performance testing (Li et al., 2009; SmartBear, b). However, none of them refers specifically to WfMSs. Hence, this paper aims at filling that gap by offering an open-source solution.

5 Conclusion and Future Work

Benchmarking is a widely accepted approach to assess the current state of a technology and drive its performance improvement (Sim et al., 2003), provided that it meets the main requirements of relevance, trustworthiness, and reproducibility. To address these requirements, in this paper, we propose a framework which takes advantage of Docker Containers and Faban. They enable the automatic and reliable benchmarking of different WfMSs, as long as the latter expose suitable APIs for process model deployment, instance execution, and monitoring. The actual performance metrics are collected and processed offline, after the workload has been completely issued in accordance with the corresponding workload model. By exploiting containerization technologies, we also deal with some of the logistic challenges in handling the interaction with WfMSs vendors. To enhance the transparency, our methodology: 1) clearly defines the interaction between the BenchFlow team and the vendors, and formalises it by means of a written agreement; 2) allows the vendors to provide configured WfMS Containers for each version to be included in the benchmark, as well as for each configuration alternative; 3) defines a verification process in which the vendors can access the draft benchmark results, as well as all the configuration parameters for automatic execution of the tests, in order to validate their correctness.

We have started applying the methodology with two open-source WfMSs (Ferme et al., 2015), which is driving the further development of the complete benchmarking framework that will be released as an open-source. This will enable interested vendors to apply the framework following the methodology defined in this paper to obtain their performance benchmark results. Only then we will be able to provide a full validation of proposed methodology's capabilities for real-world benchmarking and a definition of KPIs capable of rightfully reflecting the differences in performance. For the time being, the approach described in this paper has been approved as sound by the management of the Activiti BPM Platform[11].

6 ACKNOWLEDGMENTS

This work is funded by the Swiss National Science Foundation and the German Research Foundation with the BenchFlow project (DACH Grant Nr. 200021E-145062/1).

REFERENCES

Barbacci, M., Klein, M., et al. (1995). Quality attributes. Technical report, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA.

Bianculli, D., Binder, W., and Drago, M. L. (2010a). SOABench: Performance evaluation of service-oriented middleware made easy. In *Proc. of the ICSE '10*, pages 301–302.

[11] http://activiti.org

Bianculli, D., Binder, W., et al. (2010b). Automated performance assessment for service-oriented middleware: A case study on BPEL engines. In *Proc. of the WWW '10*, pages 141–150.

Boettiger, C. (2014). An introduction to Docker for reproducible research, with examples from the R environment. *ACM SIGOPS Operating Systems Review, Special Issue on Repeatability and Sharing of Experimental Artifacts*, 49(1):71–79.

Chhetri, M. B., Chichin, S., et al. (2015). Smart CloudBench – a framework for evaluating Cloud infrastructure performance. *Information Systems Frontiers*, pages 1–16.

Daniel, F., Pozzi, G., and Zhang, Y. (2011). Workflow engine performance evaluation by a black-box approach. In *Proc. of the International Conference on Informatics Engineering & Information Science (ICIEIS '11)*, ICIEIS '11, pages 189–203. Springer.

Dean, J. and Ghemawat, S. (2008). MapReduce: Simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113.

Faban (2014). Faban. `http://faban.org`.

Felter, W., Ferreira, A., Rajamony, R., and Rubio, J. (2014). An updated performance comparison of virtual machines and linux containers. Technical report, IBM.

Fenton, N. and Bieman, J. (2014). *Software metrics: a rigorous and practical approach*. CRC Press, 3rd edition.

Ferme, V., Ivanchikj, A., and Pautasso, C. (2015). A framework for benchmarking BPMN 2.0 workflow management systems. In *Proc. of the 13th International Conference on Business Process Management*, BPM '15. Springer.

Geiger, M., Harrer, S., et al. (2015). BPMN conformance in open source engines. In *Proc. of the SOSE 2015*, San Francisco Bay, CA, USA. IEEE.

Gillmann, M., Mindermann, R., et al. (2000). Benchmarking and configuration of workflow management systems. In *Proc. of the CoopIS '00*, pages 186–197.

Gray, J. (1992). *The Benchmark Handbook for Database and Transaction Systems*. Morgan Kaufmann, 2nd edition.

Hollingsworth, D. (1995). Workflow management coalition the workflow reference model. *Workflow Management Coalition*, 68.

Huppler, K. (2009). The art of building a good benchmark. In *Performance Evaluation and Benchmarking (TPCTC 2009)*, pages 18–30. Springer.

Iosup, A., Prodan, R., et al. (2014). IaaS Cloud benchmarking: Approaches, challenges, and experience. In *Cloud Computing for Data-Intensive Applications*, pages 83 –104. Springer New York.

Jain, R. K. (1991). *The art of computer systems performance analysis: techniques for experimental design, measurement, simulation and modeling*. Wiley, NY.

Jordan, D. and Evdemon, J. (2011). Business Process Model And Notation (BPMN) version 2.0. Object Management Group. `http://www.omg.org/spec/BPMN/2.0/`.

Juszczyk, L., Truong, H.-L., et al. (2008). GENESIS - a framework for automatic generation and steering of testbeds of complex Web Services. In *Proc. of the ICECCS 2008*, pages 131–140.

Leymann, F. (2011). BPEL vs. BPMN 2.0: Should you care? In *Proc. of the BPMN '10*, volume 67, pages 8–13. Springer.

Li, X., Huai, J., et al. (2009). SOArMetrics: A toolkit for testing and evaluating SOA middleware. In *Proc. of SERVICES-1 '09*, pages 163–170.

Merkel, D. (2014). Docker: Lightweight linux containers for consistent development and deployment. *Linux J.*, 2014(239).

Molyneaux, I. (2014). *The Art of Application Performance Testing: From Strategy to Tools*. O'Reilly Media, Inc., 2nd edition.

Parmenter, D. (2010). *Key Performance Indicators (KPI): developing, implementing, and using winning KPIs*. John Wiley & Sons.

Pautasso, C., Ferme, V., et al. (2015). Towards workflow benchmarking: Open research challenges. In *Proc. of the BTW 2015*, pages 331–350.

Puri, S. (2008). Recommendations for performance benchmarking. `http://www.infosys.com/consulting/architecture-services/white-papers/Documents/performance-benchmarking-recommendations.pdf`.

Röck, C., Harrer, S., et al. (2014). Performance benchmarking of BPEL engines: A comparison framework, status quo evaluation and challenges. In *Proc. of the SEKE 2014*, pages 31–34.

Russell, N., van der Aalst, W. M., and Hofstede, A. (2007). All that glitters is not gold: Selecting the right tool for your BPM needs. *Cutter IT Journal*, 20(11):31–38.

Sim, S. E., Easterbrook, S., et al. (2003). Using benchmarking to advance research: A challenge to software engineering. In *Proc. of the ICSE '03*, pages 74–83.

Skouradaki, M., Roller, D. H., et al. (2015). On the road to benchmarking BPMN 2.0 workflow engines. In *Proc. of the ICPE '15*, pages 301–304.

SmartBear. LoadUI. `http://www.loadui.org/`.

SmartBear. SoapUI. `http://http://www.soapui.org`.

The Apache Software Foundation (2015). JMeter. `http://jmeter.apache.org`.

Transaction Processing Council (TPC) (1997). TPC Benchmark C (Online Transaction Processing Benchmark) Version 5.11.

Transaction Processing Performance Council (2007). TPC-E. `http://www.tpc.org/tpce`.

Turnbull, J. (2014). *The Docker Book: Containerization is the new virtualization*.

van Hoorn, A., Vøgele, C., et al. (2014). Automatic extraction of probabilistic workload specifications for load testing session-based application systems. In *Proc. of the VALUETOOLS 2014*, pages 139–146. ICST.

Wetzstein, B., Leitner, P., et al. (2009). Monitoring and analyzing influential factors of business process performance. In *Proc. of EDOC '09*, pages 141–150.