Performance Benchmarking of BPEL Engines: A Comparison Framework, Status Quo Evaluation and Challenges

Cedric Röck, Simon Harrer and Guido Wirtz

Distributed Systems Group, University of Bamberg
An der Weberei 5, 96047 Bamberg, Germany
E-mail: {cedric.roeck | simon.harrer | guido.wirtz}@uni-bamberg.de

Abstract

Despite the popularity of BPEL engines to orchestrate complex and executable processes, there are still only few approaches available which help to find the most appropriate engine for individual requirements. One of the more crucial comparison factors for middleware products in industry are the performance characteristics. There exist multiple studies in both industry and academia testing the performance of BPEL engines, which differ in focus and method. We aim to compare the methods used in these approaches and provide guidance for further research in this area. Based on the related work in the field of performance testing, we created a process engine specific comparison framework, which we used to evaluate and classify nine different approaches that were found via a systematical literature survey. With the results of the status quo analysis in mind, we derived directions for further research in this area.

Keywords: SOA, BPEL, engines, performance testing

1. Introduction

Over the past few years, the research concerning the Web Services Business Process Execution Language (BPEL) [21] made huge progress and focused on arising chances and challenges for businesses [5]. Based on service-oriented architectures (SOAs), one of the major trends in the development of business information systems, BPEL steadily became *the* standard for Web Service based business orchestrations [15]. Strongly connected to the growing popularity is the development of more complex systems, which also leads to an increased error-proneness of the developed software [3]. Therefore, the need to intensively test those SOAs also gains more importance. However, there still is a tremendous deficit in terms of proper testing tool support, which has been considered to be one of the major problems of SOAs [6].

A classic goal of performance testing has always been to compare competing platforms, allowing the selection of the best fitting product for particular needs [28]. Despite the grown acceptance of SOAs, performance testing in this area still lacks some major aspects. While the number of approaches steadily grows, the majority focuses solely on the evaluation of single Web services instead of middleware components or even complete systems [13]. However, "the middleware used to build a distributed application often determines the overall performance of the application" [8, p. 2] and should therefore be considered at least as carefully as the choice of partner services involved in a process. Looking at performance testing of BPEL processes and their engines, a few studies have been conducted in industry and academia. But there is no standardized benchmark, let alone a commonly agreed upon testing methodology [7]. As a consequence, the current view on the status quo and further research topics is clouded. In this work, we aim to provide a clear view on both, the current state in performance testing of process engines and further challenges in this area.

Our contribution[1] comprises three parts. First, the creation of a framework to compare performance benchmarking approaches of process engines in Section 2. Second, a literature survey of the status quo in performance benchmarking of BPEL engines reusing the previously created comparison framework in Section 3. Third, directions on improving performance benchmarking of process engines are given based on the comparison results in Section 4. Our work concludes in Section 5, which also outlines future work.

2. Comparison Framework and its Criteria

Performance testing refers to the usually technical examination of a system under test (SUT) with the aim to analyse and validate a product's characteristics. It is known for allowing a measurement-based comparison of different products and platforms under similar conditions, hence offering valuable information for purchase decisions [28]. Furthermore, it is also used to identify bottlenecks and verify the requested quality of service (QoS) attributes in nearly finished software, i.e., whether nonfunctional performance requirements are met [29]. According to Koziolek [14], the performance of a software component is influenced by five factors: its implementation, the usage profile, deployment platform, required services, and

[1] For more details, see the accompanying technical report [22].

resource contention. Based on these influences, we created a framework to enable the classification of BPEL engine performance tests. In the following, we present four primary and three secondary criteria, by which performance tests can be analysed and made comparable.

Measurement based performance tests can be executed in several ways depending on the intended purpose of the test. These different strategies establish the **types of performance tests**, namely *baseline*, *stress* and *load* tests [18]. *Baseline tests* measure the response time behaviour of an application with only a single request or user, hence represent the best-case scenario. It can be used for product comparison or as a benchmark baseline for other test types, e.g., for load tests [20, pp. 38-40]. A *load test* verifies the application behaviour in typical and expected situations, including load peaks with multiple concurrent users. It is often used to assure performance related QoS attributes [18] [19, p. 291]. Pushing an application beyond the expected level of load and peak conditions is called *stress testing*. Its goal lies in revealing bugs and unexpected behaviour that only appear in extreme conditions [18]. The results reflect the worst-case scenario and unearths the capacity limits of the system under test [20, pp. 38-40] [19, p. 291].

Workloads are defined as the sum of all inputs that are received by the SUT. They are considered to be one of the key aspects for assuring the validity of performance test results [19,25,28,29]. In the context of BPEL engines, workloads consist of the invoked processes and their dependent Web services, but also include the request messages and the strategy for sending these requests. According to [19, pp. 265-266], the workload of software performance benchmarks can be categorised into four *workload types*. *Basic operations* refer to the smallest operations, i.e., all supported and standardized activities in the context of a BPEL engine, and supply fine-grained performance characteristics. *Toy-benchmarks*, usually implementing classic puzzles or being proof-of-work concepts, are of not much use for performance tests. *Kernels* are core parts of real programs. They represent the most important or time consuming parts and provide an intermediate level in terms of granularity and realism. In our context, kernel processes implement patterns, e.g., the workflow patterns [27] that were extracted from a large corpus of real world processes. *Real programs*, or real processes in our case, are often seen as the most important workload type as their results are most accurate for estimating the performance of production systems [2, 25]. Therefore, real workloads are also used in other domains, e.g., for benchmarking SQL databases with TPC-C.

Workload injection strategies can be subdivided into a) *continuously* injecting the workload and b) injecting the workload with an *arrival rate*. *Continuously* injected workloads test the engine with a constant load, once reaching the plateau phase. This plateau, however, can either be reached with a big bang (default approach), or *stepwise*, by slowly increasing the number of concurrent users over a specific period [20, p. 41]. The *arrival rate* can be either fixed or dynamic to simulate even more realistic peaks [25].

If the SUT and load generating clients are co-located, one has to closely observe the system for overloads. Therefore, they are often installed on separate systems, i.e., *distributed*. Moreover, a separation allows the load to be created from distributed clients, providing more realistic situations and assuring the saturation of the SUT.

The evaluation of software performance usually considers multiple metrics. In this study, we use three **performance metrics**: *latency*, *throughput* and *utilization*. The *latency* or response time [8] defines the time span between sending a request and having received the full response. It is influenced by multiple factors, e.g., the execution of partner services, network delays and message parsing [24]. *Throughput* is the most commonly used metric for software performance, defining the number of transactions that are completed within a period. In the context of BPEL engines, the term transaction can refer either to the completion of requests or process instances. The *utilization* reflects the degree to which a capacity is used and refers to many parts of the testbed, e.g., the network, database, server, or load-generating clients [20, p. 29]. As metrics differ in semantics and focus, they depend upon the test types. For example, a baseline test using any throughput metric is meaningless as there is only one active request at a time.

In addition to the primary criteria, we distinguish between three **secondary criteria**. First, the **number of the engines and their license**, being either open source or proprietary. Second, the **setup of the test bed**, being either automated or done manually, which has effects on the experiment's reproducibility. And third, as BPEL engines provide plenty of **configuration options**, we distinguish whether the processes are executed in-memory or not, which we denote as *persistence*.

3. Literature Study

The results of our systematic literature study[2], i.e., the nine approaches and their classifications, are shown in Table 1. Each row and each column refers to an approach and a comparison criterion, respectively. Cells represent the findings, with empty cells denoting the absence of a criterion and cells marked *n/a* denoting that the approach did not provide any data regarding this criterion.

The benchmark conducted by *SOABench* [4] includes three engines and performs load tests using four BPEL processes. Two processes are built with the `flow` activity, the other two use either the `sequence` or the `while` activity. All four processes invoke mocked external services, thus, use the `invoke` activity as well. The performance is measured via latency and throughput metrics. Moreover, *SOABench* features the automated generation, execution and analysis of testbeds, allowing reproducing the test results. Workloads are either injected based on a given arrival rate, or delayed by a *thinking time* between consecutive invocations.

The load test of *OpenESB* [26] is focused on throughput metrics. Its workload is a single process, using all supported BPEL activities, i.e., a toy benchmark, and is injected stepwise.

[2]The method is detailed in the accompanying technical report [22].

Table 1. Literature analysis, comparing most important performance test factors

Criteria	Test Type			Workload			Workload Injection				Metrics			Engines		Config	Testbed
Approaches	Baseline	Load	Stress	Type	# of Processes	Ext. Service	Distributed	Continuously	Stepwise	Arrival Rate	Throughput	Latency	Utilization	Open Source	Proprietary	Persistence	Automated
SOABench [4]		x	x	basic	4	x	x	x		x	x	x		2	1		x
OpenESB [26]		x		toy	1	n/a			x		x			1			
ActiveVOS [1]		x		real	2		n/a	n/a	n/a	n/a	x				1	x	
Sliver [10]	x			kernel	12	x		x				x	x	2			
Workload model [9]		x		toy	1					x		x		1			
Intel & Cape Clear [12]		x		real	2	x	x	x			x	x	x		1	x	
Roller [23]		x		real	1	x		x			x				1		
SWoM [17]		x		basic	2	x		x			x	x			3		
FACTS [16]		x		real	1	x	n/a	n/a	n/a	n/a		x		1			

Similarly, throughput metrics are measured in the load test of *ActiveVOS* [1]. The test uses two functional processes, with one being analysed with persistence enabled or disabled.

In [10], *Sliver* is compared with *ActiveBPEL*, using a baseline test measuring request latency and memory utilization. The workload consists of twelve workflow patterns [27], i.e., kernels, realized as BPEL processes, which utilize external services and are invoked in consecutive requests.

Validating their workload model, Din et al. [9] have performed a load test that focused only on the response time behaviour of the used *ActiveBPEL* engine. The tested process uses correlations but does not call external services. The workload is injected by several virtual users over a total duration of two minutes, injecting 20 new processes per second.

The load test of Intel and Cape Clear [12] focuses on latency, throughput and utilization measures of the tested *Cape Clear 7* engine. The two processes implement a typical industrial functionality and invoke external services, while the workload is continuously injected from a set of distributed clients Additionally, the test focuses on the effect of persistence on the performance metrics.

Verifying Roller's [23] proposals, he has conducted a load test on one proprietary engine and measured throughput metrics. The tested workload is a single realistic BPEL process, including the invocation of external services, which is continuously called by the testing clients.

Benchmarking three BPEL engines, Längerer et al. [17] have conducted a load test focusing on throughput and latency metrics. The workload, which is continuously injected, consists of two processes. One uses the `assign` activity, the other one calls an external service with the `invoke` activity.

The load test of Liu et al. [16] tests the response time behaviour of a single realistic process, which includes external services and was deployed on the *ActiveBPEL* engine.

4. Discussion and Further Suggestions

This section analyses the findings of our evaluation and discusses which parts of BPEL performance testing should be strengthened in future work. As the approaches differ in many aspects and follow no common schema, we focus on patterns per criterion, i.e., compare the results column-wise.

The approaches mainly execute load tests, whereas, in addition, Bianculli et al. [4] also applies stress testing. Solely Hackmann et al. [10] perform baseline tests, leaving room for further evaluations, by means of baseline and stress tests.

The workload types differ for all approaches, with four using real workloads, one using kernels, and two using basic and toy each. For each workload, the number of processes also varies, with four approaches using only a single process, three using two, one using four and one using twelve processes. The majority (6/9) uses processes that invoke external services, thus the test always includes the performance characteristics of the `invoke` activity. For the remaining approaches, two do not use external services, while it is unknown for the last one. For all workload types, a larger corpus of processes would help to improve the meaningfulness of the test results. Regarding the basic category, we propose to have a process per feature to be able to compare their performance characteristics. For kernel processes, we suggest to cover more pattern catalogues, whereas for real processes, we propose to use distinct real processes from various use cases in different domains. As external services are a crucial part of BPEL processes, they should not be neglected in further studies as well.

Concerning the workload injection, two approaches did not mention it at all. For some approaches, it is neither stated, nor can it be deducted. As it is a crucial part of a performance evaluation, we advise to explicitly state the chosen strategy.

Regarding the metrics, latency and throughput are used by six approaches each, whereas only two measure utilization. In this context, Intel and Cape Clear [12] provide the most complete approach as they measure all three metric types. Three approaches use two metric types, while the majority (5/9) measures only metrics of a single type. Hence, we propose to focus on the neglected utilization metric, which can reveal interesting characteristics of the engines as well as ensure that there are no system overloads falsifying any results.

The number of engines under test range from one up to three per approach, limiting their relevance for buying decisions. Three approaches compare the performance of multiple

engines, while the remaining six evaluate the performance of a single engine. Only SOABench [4] compares open source with proprietary engines. But as the proprietary engine ActiveVOS incorporates the open source engine ActiveBPEL, SOABench basically test the open source one. Hence, there is no proper performance comparison of open source with proprietary engines, leaving room for further work in this area.

Regarding the configuration opportunities of the BPEL engines, only two approaches [1, 12], both evaluating only a single engine, tested their engines in different configurations, namely either execute their processes in-memory or not. As most engines offer multiple options, it shows that this has been neglected in research, despite its importance. However, when comparing more than one engine, it has to be ensured that all engines equally support the capability.

With only Bianculli et al. [4] allowing to automatically setup the testbed and execute the tests, it is very hard to redo the experiment for all other approaches. Moreover, only [4, 12, 17, 23] published their detailed test setup, processes and tools, which are essential for the repeatability of these tests.

None of the approaches allow analysing the influence of environmental aspects, for instance the system's hardware, database or influences of long-running transactions on the engines' performance. However, modern multi-core systems and solid-state drives provide new challenges and opportunities for differentiation among middleware products.

One additional problem is that none of the approaches takes into account that BPEL engines greatly vary in their degree of support of the BPEL specification [11], i.e., they implement different subsets of the BPEL features. We propose to take these results into account, creating and selecting appropriate workloads for the engines to be compared.

5. Conclusion and Future Work

In our work, we created a comparison framework with which existing performance benchmarking approaches of process engines, and BPEL engines in particular, can be classified. We applied our comparison framework onto nine methodically found approaches, revealing their differences and similarities. Based on the findings, we derived guidance for further research in the areas which have been neglected so far.

In future work, we want to apply our comparison framework and method onto other studies targeting other process engines and their languages, and fill in the open gaps that were revealed during this study.

References

[1] Active Endpoints Inc. Assessing ActiveVOS Performance. `http://bit.ly/R60NPy`. 2014-01-30.

[2] A. Avritzer, J. Kondek, D. Liu, and E. J. Weyuker. Software Performance Testing Based on Workload Characterization. In *WOSP*, 2002.

[3] V. R. Basili and B. T. Perricone. Software Errors and Complexity: An Empirical Investigation. *CACM*, 1984.

[4] D. Bianculli, W. Binder, and M. L. Drago. Automated Performance Assessment for Service-oriented Middleware: A Case Study on BPEL Engines. In *WWW*, 2010.

[5] M. Bozkurt, M. Harman, and Y. Hassoun. Testing and Verification in Service-Oriented Architecture: A Survey. *Software Testing, Verification and Reliability*, 2012.

[6] G. Canfora and M. D. Penta. Testing Services and Service-Centric Systems: Challenges and Opportunities. *IT Professional*, 2006.

[7] S. Chen, L. Bao, and P. Chen. OptBPEL: A Tool for Performance Optimization of BPEL Process. In *Softw. Comp.*, 2008.

[8] G. Denaro, A. Polini, and W. Emmerich. Early Performance Testing of Distributed Software Applications. In *WOSP*, 2004.

[9] G. Din, K.-P. Eckert, and I. Schieferdecker. A Workload Model for Benchmarking BPEL Engines. In *ICSTW*, 2008.

[10] G. Hackmann, M. Haitjema, C. Gill, and G.-C. Roman. Sliver: A BPEL Workflow Process Execution Engine for Mobile Devices. In *ICSOC*. 2006.

[11] S. Harrer, J. Lenhard, and G. Wirtz. BPEL Conformance in Open Source Engines. In *SOCA*, 2012.

[12] Intel and Cape Clear. BPEL scalability and performance testing. Technical report, Intel and Cape Clear, 2007.

[13] L. Juszczyk and S. Dustdar. Script-Based Generation of Dynamic Testbeds for SOA. In *Socially Enhanced Services Computing*. Springer, 2011.

[14] H. Koziolek. Performance Evaluation of Component-based Software Systems: A Survey. *Performance Evaluation*, 2010.

[15] T. v. Lessen, D. Lübke, and J. Nitzsche. *Geschäftsprozesse automatisieren mit BPEL [Automating Business Processes with BPEL]*. dpunkt.verlag, 2011.

[16] A. Liu, Q. Li, L. Huang, and M. Xiao. Facts: A framework for fault-tolerant composition of transactional web services. *Services Computing, IEEE Transactions on*, 2010.

[17] C. Längerer, J. Rutschmann, and F. Schmitt. Performance-Vergleich von BPEL-Engines [Performance Comparison of BPEL Engines]. Technical report, 2006.

[18] J. Meier, C. Farre, P. Bansode, S. Barber, and D. Rea. *Performance Testing Guidance for Web Applications: Patterns & Practices*. Microsoft Press, 2007.

[19] D. A. Menascé. *Capacity Planning for Web Services: Metrics, Models, and Methods*. Prentice Hall, 2002.

[20] I. Molyneaux. *The Art of Application Performance Testing*. O'Reilly Media, 2009.

[21] OASIS. *Web Services Business Process Execution Language*, 2007. v2.0.

[22] C. Röck, S. Harrer, and G. Wirtz. Testing BPEL Engine Performance: A Survey. Technical report, Univ. of Bamberg, 2014.

[23] D. Roller. *Throughput Improvements for BPEL Engines: Implementation Techniques and Measurements Applied to SWoM*. PhD thesis, IAAS, Stuttgart, Germany, 2013.

[24] F. Rosenberg, C. Platzer, and S. Dustdar. Bootstrapping Performance and Dependability Attributes of Web Services. In *ICWS*, 2006.

[25] A. J. Smith. Workloads (Creation and Use). *Com. ACM*, 2007.

[26] Sun Microsystems. Benchmarking BPEL Service Engine. `http://bit.ly/1jkssHd`. 2014-01-30.

[27] W. van der Aalst, A. ter Hofstede, B. Kiepuszewski, and A. Barros. Workflow Patterns. *Distributed and Parallel Databases*, 2003.

[28] E. J. Weyuker and F. I. Vokolos. Experience with Performance Testing of Software Systems: Issues, an Approach, and Case Study. *IEEE Trans. Softw. Eng.*, 2000.

[29] M. Woodside, G. Franks, and D. C. Petriu. The Future of Software Performance Engineering. In *FOSE*, 2007.