Load Balancing Concurrent BPEL Processes by Dynamic Selection of Web Service Endpoints

Marvin Ferber*, Sascha Hunold*, Thomas Rauber
Department of Computer Science
University of Bayreuth, Germany
Email: {marvin.ferber,hunold,rauber}@uni-bayreuth.de

Abstract

Business workflows implemented as BPEL processes play an important role for many business applications. BPEL is used to orchestrate a series of Web service calls. Which provider is used for a specific Web service request is statically defined as endpoint of a Web service call within the BPEL code. If a Web service is offered by more than one provider an intelligent choice of a Web service provider can improve the throughput of a BPEL engine.

In this article we study how a scheduling mechanism can be injected into a standard BPEL process by an automatic transformation of the BPEL process. The scheduling mechanism helps the BPEL engine to select between different Web service providers, e. g., according to their current workload. Three different strategies for scheduling Web service calls are examined in practical experiments, considering a homogeneous and a heterogeneous collection of Web service providers. We show that in most cases the dynamic scheduling approach leads to a smaller average execution time (makespan) of concurrently executed BPEL processes.

1. Introduction

Today many business applications are built upon or integrate Web services. Major software companies offer Web services for public use, e. g., Web services from Yahoo, Google, or Amazon. Aside from applications of the web 2.0 domain, Web services are key components for larger business software systems. In order to implement a business workflow, each functional node (activity) is mapped onto the Web service that provides the required functionality. In the business domain, the Business Process Execution Language (BPEL) [1] is often used to implement workflows and to orchestrate the relations between Web services. Most service-oriented architectures are tightly coupled to BPEL as execution language for business workflows [2]. Besides BPEL, other workflow description languages have been proposed such as XPDL [3]. However, to date, BPEL is the de-facto standard for workflow execution [4].

A BPEL process denotes a workflow modeled with BPEL that is executed by a so-called BPEL (execution) engine. Various execution engines for BPEL are available, proprietary ones as well as free ones, e. g., activeBPEL, Intalio server, or the Sun BPEL engine.

In a real world scenario, different BPEL engines call several Web services which can be spread all over the world. A Web service can be provided by multiple servers, and many BPEL processes of the same type can run simultaneously on the same or on different BPEL engines. That means that a request to a Web service can be satisfied by different servers on the Internet, and the workload of these servers may change over time. However, the problem of standard BPEL processes is that the provider of a specific Web service is hard wired into the BPEL code. Thus, a BPEL engine always maps a process activity to the previously configured Web service provider. In a dynamically changing environment it is therefore desirable to select a Web service provider dynamically, taking the current workload into consideration.

The main objectives of the herein described approach are: 1) A BPEL process should be transformed automatically to enable a dynamic scheduling decision. 2) The transformed BPEL process should also be standard BPEL compliant, i. e., the process should not be dependent on a specific engine. 3) The scheduler should distribute the Web service calls efficiently so that the total execution time is reduced compared to static BPEL processes.

The contribution of the article is a novel (transformation) approach for adding a scheduling decision into a standard BPEL process to enable the BPEL engine to dynamically assign a Web service provider at runtime. Several strategies for Web service assignment are discussed and compared in practical experiments using the activeBPEL community engine [5] and the Apache Tomcat Applications server [6].

**Supported by the German Federal Ministry of Education and Research (BMBF), 01 ISF 10 B.*

The article is organized as follows. Section 2 discusses the need for dynamic BPEL processes and shows the proposed BPEL transformation by example. Section 3 summarizes which information is available to the BPEL engine and how it can be used to select a Web service provider at runtime. The Section 4 introduces several scheduling strategies which are experimentally evaluated in Section 5. Section 6 discusses related work and Section 7 concludes the article.

2. Dynamic BPEL processes

2.1. Properties of static BPEL processes

The purpose of the Business Process Execution Language (BPEL) is to let programs interact with Web services in a unified way. A BPEL process consists of variables, partner links, and activities. A BPEL process is instantiated by the BPEL engine, normally through an external request. A Web service call is represented as activity, and with BPEL different activities can be modeled as a workflow. In a *sequence* block the activities are executed one after another. A *flow* block allows the parallel execution of activities. An assign activity is used to manipulate variables, e. g., initialize a variable or store the result of a calculation. An invoke activity is used to call an external provider like a Web service. The partner link contains all information required to call a specific endpoint, e. g., a Web service. Thus, an invoke activity always corresponds with a partner link.

BPEL processes normally orchestrate one or more Web service activities. As stated above, invoke activities are implemented statically, i. e., the corresponding Web service endpoint is defined at build time. In case that a Web service is provided by several servers, a dynamic adaptation of the service endpoint can lead to a faster response time of a Web service call.

2.2. From static to dynamic BPEL processes

Usually, the BPEL process and its endpoints are statically defined. To overcome this limitation, we extend the activeBPEL community engine by a custom scheduling module that enables a dynamic endpoint selection at runtime. Additionally, the static BPEL process has to be adapted to support the dynamic selection of an endpoint.

We make the former static invoke activity aware of the scheduling mechanism by wrapping it into a pre-invoke activity (`getEndpoint`) and a post-invoke activity (`endpointReturned`) as shown below:

```
EP = getEndpoint( Web service name )
invokeOnEndpoint( EP )
endpointReturned( EP )
```

In the first step, the BPEL process calls the scheduling module of the engine and requests an endpoint for the current Web service. After that, the BPEL process can call the Web service on the given endpoint. When the Web service call is completed, the BPEL process notifies the scheduler about the completion by calling `endpointReturned()`. So, the scheduler can keep track of the call duration time of Web services for each provider. This information is necessary for the scheduler to choose the next endpoint upon request.

For a BPEL process, a call to the scheduler can be modeled the same way as a call to any Web service using the `invoke` activity. To invoke an activity it needs to be defined within a WSDL document (Web Service Description Language [7]). Hence, the scheduling activity and its interfaces are also declared in a WSDL document. In addition to that, a deployment descriptor is needed to deploy the BPEL process to the BPEL engine. With the information from the deployment descriptor the engine assigns the partner links to real endpoints. In this deployment descriptor the partner link of the scheduler is assigned to the scheduler module in the engine. For illustration, a BPEL process comprised of two Web service calls is shown together with the corresponding transformation result in Figure 1. Although the implementation of the scheduler depends on the BPEL engine, the transformed BPEL process is conform to the BPEL standard. Thus, the resulting BPEL process is executable on other BPEL engines.

3. Dynamic endpoint selection

We now want to discuss how the dynamic endpoint selection can reduce the execution time of concurrently executed BPEL processes.

In this work, we consider a system $\mathcal{H}$ which consists of a host that runs a BPEL engine and n Web service providers (servers). All servers provide the same type of Web service. The Web service provided by server $i, 0 \leq i < n$ is called w_i. We also assume that m BPEL processes are executed simultaneously. Each of the m concurrently running BPEL processes consists of k Web service calls. Hence, in total $m \cdot k$ Web service calls have to be distributed across the Web service providers.

The goal of the dynamic endpoint selection is to distribute the Web service calls over the available Web service endpoints w_i in such way that the average makespan of the concurrently executed BPEL processes is minimized. We consider one scheduler per BPEL engine that is aware of the requested Web services and the corresponding Web service endpoints. The list of available Web service providers and their endpoints hast be specified by the administrator of the BPEL engine. A Web service is exposed by the information given

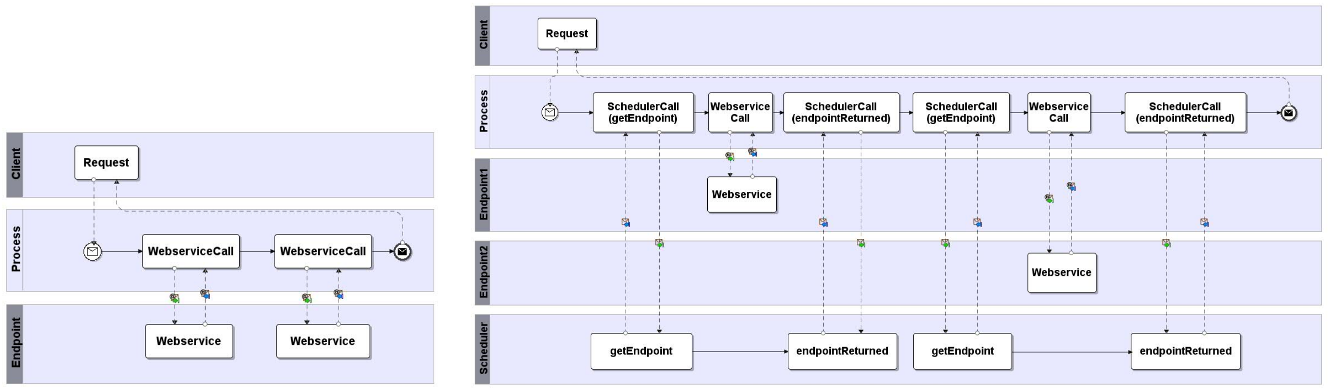

Figure 1: The original BPEL process (left) and the transformed, scheduling enabled BPEL process (right).

in the WSDL document. We consider a Web service that does not provide any state information at runtime. Moreover there is no common interface to request state information. Thus, our scheduler can only rely on its own collected information about recent requests. It has to make an online decision based on the history of previous Web service calls. In particular, the scheduler keeps track of the call duration of all Web service endpoints. Additionally, the current number of pending calls is known to the scheduler. The expected response time of the next call to a specific Web service endpoint is estimated by considering the current load and the duration of the last call to this endpoint. The scheduling instance maps each Web service call request to an appropriate Web service provider. At a particular time the scheduler must select an endpoint w_i from the set of available endpoints $w_0, \ldots, w_{n-1}$. In general, we assume that the available Web service endpoints are distributed onto a set of heterogeneous machines that leads to different response times for a Web service call on each machine.

4. Strategies for endpoint selection

After showing how to integrate a scheduling decision of Web service calls into a BPEL process, we now introduce several scheduling strategies for selecting an appropriate Web service endpoint at runtime. We focus on the general case in which the Web service calls have to be distributed among a heterogeneous set of Web service endpoints. For each strategy, the predicted performance is assessed. For rating the estimated execution time of the sample BPEL process we consider the performance ratio $F = \frac{t_{strategy}}{t_{opt}}, \quad F \geq 1$ where $t_{strategy}$ denotes the execution time of the BPEL process using the examined scheduling strategy and t_{opt} denotes the execution time of the BPEL process in the best case. Hence, the closer the F ratio to one the better is the scheduling strategy for the specific case. The speed $S(w_i)$ specifies the execution time of one Web service invocation on endpoint w_i.

We consider a sample BPEL process consisting of a sequence of Web service calls to a single type of Web service. We assume that two Web service endpoints are available: w_{fast} responds twice as fast as w_{slow} when called with the same parameters ($S(w_{slow}) = 2 \cdot S(w_{fast})$). Two scenarios are examined for each strategy. In the first scenario E_1 only one instance of the sample BPEL process is executed. In the second scenario E_∞ many BPEL processes are executed in parallel. The performance $P(example)$ denotes the execution time for a specific example and depends on the distribution of the number of Web service calls to w_{slow} and w_{fast}. In the following we present strategies to select an appropriate Web service endpoint for a specific Web service request at runtime, considering to have only information about the current number of calls and the duration of recent requests to a Web service. We compare the estimated execution time of an example to the theoretical best execution time which is determined manually.

4.1. Round robin scheduling

In a homogeneous environment, where (almost) all Web service providers have a similar processing power, every Web service endpoint should get nearly the same amount of calls. Distributing the number of calls in equal shares to the Web service providers can be achieved with a *round robin* (RR) strategy.

If an endpoint is requested, the scheduler returns the next endpoint cyclically from the array `EP` of available endpoints. The variable `idx` points to the next EP and is initialized with zero at startup. The RR strategy is easy to implement and entails a small overhead since the next endpoint can be obtained in $O(1)$. The function to select the next endpoint with the RR strategy is shown below:

```
function getEndpoint()
  idx =(idx+1) mod n
  return EP[idx]
```

When n endpoints are available, every endpoint is assigned to one n-th of the calls. Considering the previously mentioned two Web service providers (w_{slow}, w_{fast}), each Web service endpoint is assigned to 50 % of the calls. But if only *one* sequential BPEL process is executed, the best strategy would be to call only w_{fast}. So, the performance ratio of the time of the RR strategy and the best strategy can be given as:

$$F_{\mathsf{RR}} = \frac{P_{\mathsf{RR}}(E_1)}{P_{OPT}(E_1)} = \frac{\frac{1}{2}S(w_{fast}) + \frac{1}{2}S(w_{slow})}{S(w_{fast})} = 1.5\,.$$

For a larger number of BPEL processes the best strategy is to call w_{fast} twice as often as w_{slow}. The performance ratio of strategy RR for multiple concurrently running BPEL processes is:

$$F_{\mathsf{RR}} = \frac{P_{\mathsf{RR}}(E_\infty)}{P_{OPT}(E_\infty)} = \frac{\frac{1}{2}S(w_{fast}) + \frac{1}{2}S(w_{slow})}{\frac{2}{3}S(w_{fast}) + \frac{1}{3}S(w_{slow})} = 1.125\,.$$

It can be seen that the RR scheduling is especially useful for (almost) homogeneous environments. With a heterogeneous set of Web service providers the RR strategy will not balance the workload according to the processing power of a provider. Therefore, we need to include the current workload into the scheduling decision.

4.2. Lowest counter first scheduling

A strategy to reflect the processing power of a Web service provider is the *lowest counter first* (LCF) strategy. The LCF scheduler uses a counter for each endpoint, which denotes the current number of pending calls to the particular endpoint. Because the counters reflect the current workload of the Web service providers, the LCF scheduler chooses the Web service endpoint with the lowest counter. The decision which Web service endpoint should be selected can be made in $O(n)$ where n is the number of available Web service endpoints. The implementation of the LCF strategy is shown below:

```
function getEndpoint()
  /* current min is first EP */
  min = counter[0]
  idx = 0
  for i=1 to n:
    if counter[i]<min:
      min = counter[i]
      idx = i
  return EP[idx]
```

When all Web service endpoints have almost the same call duration time (homogeneous environment), the LCF strategy degenerates to a RR strategy. However, if the Web service endpoints differ in speed the LCF strategy is better suited than the RR strategy. If a Web service endpoint is faster than the others, the counter of the faster is decremented earlier than the counters of the slower ones. Hence, the faster Web service endpoint obtains more Web service calls than the slower Web service endpoints.

With respect to the example Web service providers w_{fast} and w_{slow}, scheduling one BPEL process with the LCF strategy will not always lead to the best result. All counters are initialized with zero. Thus, the LCF strategy always selects the first Web service endpoint from the list which may always be the slower Web service provider (w_{slow}). Therefore, the worst-case performance ratio of the LCF scheduler for one BPEL process is:

$$F_{\mathsf{LCF}} = \frac{P_{\mathsf{LCF}}(E_1)}{P_{OPT}(E_1)} = \frac{S(w_{slow})}{S(w_{fast})} = 2\,.$$

For executing multiple BPEL processes concurrently, the number of Web service calls is better distributed among the available endpoints. In the sample scenario, the Web service calls are distributed equally over the two endpoints. But since one Web service endpoint returns twice as fast as the other it is assigned to twice as many calls as the slower one. Thus, for many BPEL process we get a performance ratio of:

$$F_{\mathsf{LCF}} = \frac{P_{\mathsf{LCF}}(E_\infty)}{P_{OPT}(E_\infty)} = \frac{\frac{2}{3}S(w_{fast}) + \frac{1}{3}S(w_{slow})}{\frac{2}{3}S(w_{fast}) + \frac{1}{3}S(w_{slow})} = 1\,.$$

The LCF is a good scheduling strategy when many concurrent BPEL processes have to be executed by a BPEL engine. It can deal with homogeneous and heterogeneous environments equally well. Unfortunately this method does not work well when less BPEL processes are executed than Web service providers are available. In this case, the LCF strategy does not consider the speed of the Web service providers and will therefore fail to efficiently distribute the workload.

4.3. Weighted random scheduling

In order to overcome the disadvantages of the LCF scheduler when executing a smaller number of BPEL processes, we introduce the *weighted random* (WR) scheduling strategy. The WR strategy determines the next Web service endpoint by randomly selecting an available Web service endpoint. To reflect the current load and thus the response time of a Web service provider in the selection process, the probability to select an endpoint is scaled according to the speed of a Web service provider. The probability to select an Web service endpoint w_i is weighted by the inverse of the response time of the last call to Web service endpoint w_i. As a result, faster Web service endpoints are more likely to be selected than slower ones. Since the speed of a Web service provider can change, the speed information has to be updated periodically, which is guaranteed by the random selection strategy. The scheduling decision to select a Web service endpoint can be made in $O(n)$ where n is the number of available Web service endpoints.

The function to select an endpoint accounting the current workload for strategy WR is given below (`duration(i)` returns the duration time of the last call to the Web service endpoint w_i):

```
function getEndpoint()
  M = new SetOfRandomVariables()
  /* insert idx as random variable &
   * inverse of the duration as prob */
  for i=0 to n-1:
    M.insertVarWithProb(i,1.0/duration(i))
  /* get index according to probability*/
  idx = M.getVarRandomly()
  /* return endpoint */
  return EP[idx]
```

If a Web service endpoint has never been called before, the last known duration time of this Web service endpoint is 0. To estimate the performance ratio we assume that there have been calls to each Web service endpoint before and the duration is already known. Our estimations are based on the stochastic distribution of an infinite number of Web service calls. If a specific Web service endpoint is chosen with a probability of $\frac{1}{2}$, then half of the Web service calls are assigned to this Web service endpoint for a sufficiently large number of requests. In homogeneous environments the probabilities to select an endpoint are equally distributed. Thus, the scheduler will choose each endpoint $\frac{1}{n}$ times. With one BPEL process and a heterogeneous set of Web service endpoints the following performance ratio can be estimated:

$$F_{\mathsf{WR}} = \frac{P_{\mathsf{WR}}(E_1)}{P_{OPT}(E_1)} = \frac{\frac{2}{3}S(w_{fast}) + \frac{1}{3}S(w_{slow})}{S(w_{fast})} = 1.33.$$

Also with many parallel BPEL processes the distribution stays the same and can be estimated as follows:

$$F_{\mathsf{WR}} = \frac{P_{\mathsf{WR}}(E_\infty)}{P_{OPT}(E_\infty)} = \frac{\frac{2}{3}S(w_{fast}) + \frac{1}{3}S(w_{slow})}{\frac{2}{3}S(w_{fast}) + \frac{1}{3}S(w_{slow})} = 1.$$

The WR strategy is well adapted to support differences in processing power of endpoints. It should also work well in a homogeneous environment since all Web service endpoints are assigned the same share of Web service calls. In contrast to the LCF strategy, the WR strategy also selects a fast Web service provider even when only a few BPEL processes are executed concurrently.

5. Experimental results

In a practical experiment we also examined the behavior of the presented scheduling strategies in combination with the transformed BPEL process. To measure the performance gain of the scheduling strategies we created several Web service endpoints with adaptable response times. The experiments were conducted on a homogeneous and heterogeneous set of Web service providers. The Web service providers and the BPEL engine were running Linux 2.6 and Sun Java 1.6.

Table 1: Overview of the experimental setup, homogeneous and heterogeneous environment.

	Homogeneous	Heterogeneous
BPEL engine	Intel Xeon MP 2.2 GHZ (4 Cores + HT) 4 GB Memory	Intel Xeon MP 2.2 GHZ (4 Cores + HT) 4 GB Memory
Web service provider 1	Intel Pentium 4 3.0 GHz (HT) 4 GB Memory	Intel Pentium 4 3.0 GHz (HT) 4 GB Memory
Web service provider 2	Intel Pentium 4 3.0 GHz (HT) 1 GB Memory	Intel Pentium 4 3.0 GHz (HT) 1 GB Memory
Web service provider 3	Intel Pentium 4 3.0 GHz (HT) 2 GB Memory	Intel Core2 Duo 2.4 GHz (2 Cores) 4 GB Memory
Web service provider 4	Intel Pentium 4 3.0 GHz (HT) 1 GB Memory	Intel Core2 Duo 3.0 GHz (2 Cores) 4 GB Memory

We implemented a synthetic Web service endpoint whose response time can be varied by different arguments of the Web service call. The synthetic Web service function takes an integer argument and performs a prime number test for this argument. Thus, the call duration time of the Web service call scales with the integer argument. Moreover, a call to this Web service function increases the work load of the Web service provider which was a main objective of the synthetic function. To obtain a specific call duration time for a Web service endpoint the number passed as argument to the Web service has to be calibrated for a given machine.

For the experiments the activeBPEL community engine (version 5.0.2) was used to execute BPEL processes. The BPEL engine was deployed to an Apache Tomcat Application server (version 5.0.28). The same version of Tomcat was used for the Web service providers. Since all BPEL processes running on the activeBPEL community engine must have access to the scheduler, the scheduler was implemented as Java package and deployed to the Application server. The deployment descriptor of the BPEL process contains the necessary information how the scheduler invoke activities are redirected to the responsible Java class.

Table 1 gives an overview of the machines used in the experiments. The Xeon and Pentium 4 machines run a 32 bit Linux, the others run in 64 bit mode. All machines are connected via Gbit Ethernet with a peer-

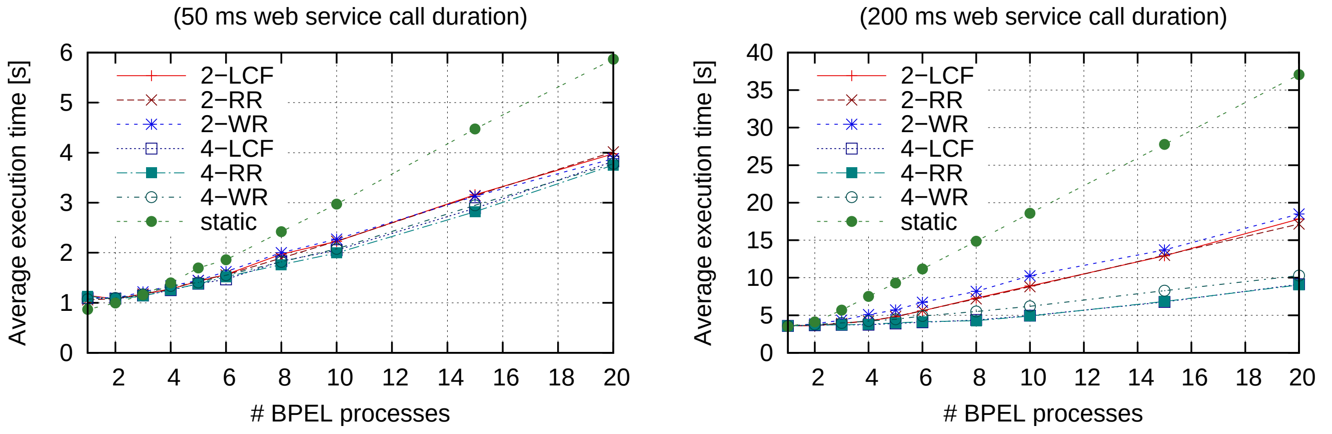

Figure 2: Average execution time of one BPEL process for an increasing number of concurrent BPEL processes on a homogeneous collection of Web service providers. Left: 50 ms per Web service call; right: 200 ms.

to-peer round-trip time of 0.12 ms in average in our setup. We investigated the performance of the dynamic BPEL processes (with scheduling support) for a given number of Web service providers in the homogeneous and the heterogeneous case.

In the homogeneous case we used two configurations. The first configuration contains two Web service providers, 1 and 2 from Table 1. The second configuration contains all Web service providers given in this table. Similarly, two different configurations were considered in the heterogeneous case. The first configuration consists of the first two Web service providers of Table 1, whereas the second consists of all listed Web service providers. For each experiments, the original BPEL process (without dynamic Web service endpoint selection) always uses the first Web service endpoint (provider 1). All experiments were conducted on dedicated machines. Different Web Service response times were realized by the different processing speeds of the machines in the heterogeneous case.

The synthetic function provided by the Web service endpoints was calibrated for each system so that the Web service call returns after approximately 50 ms or 200 ms. The so calibrated arguments were integrated in the BPEL processes. In the heterogeneous environment, the calibration was performed on the fastest Web service endpoint (provider 4). For a better comparison of the experimental results we used an initialization run of a BPEL process on the BPEL engine. With this first run we ensure that all necessary classes are already loaded into the Java VM. Additionally, in case of the WR scheduler this run also records the current call duration time for each Web service endpoint.

The original BPEL process consists of a sequence of 10 Web service calls to a Web service endpoint of the tested type (50 ms or 200 ms). The transformed BPEL process, which is enriched with the dynamic endpoint selection, also contains a sequence of 10 calls to Web service endpoints. For the experiments, a specific number of instances of the original or the transformed BPEL process are executed concurrently. We considered the concurrent execution of $x = 1, 2, 4, \ldots, 20$ instances of a BPEL process. Each test run consisting of the concurrent execution of x BPEL processes was conducted 10 times.

Figure 2 shows the experimental results for the different scheduling strategies in the homogeneous case. The diagrams show the average execution time of one BPEL process executed concurrently with x-1 other BPEL processes. Because one BPEL process is sequentially structured, the number of competing Web service calls is as large as the number of instantiated processes.

The resulting execution time of a BPEL process obtained with the investigated scheduling strategies differ more for a Web service call duration time of 200 ms than for 50 ms. For a concurrent execution of BPEL processes the BPEL engine is more stressed when the Web service call duration times are shorter. The BPEL engine can become a bottleneck for the concurrent execution. Thus, for a larger duration time (e. g., 200 ms) the resulting performance depends more on the scheduling strategy and the number of Web service providers.

In the homogeneous case all scheduling strategies show a good performance and for a call duration of 200 ms a speedup of almost 4 could be achieved for 4 Web service providers. In the homogeneous environment it is important to distribute one n-th of the calls to each Web service endpoint in order to gain high performance. The RR and the LCF scheduling algorithms provide this distribution and show therefore the best resulting

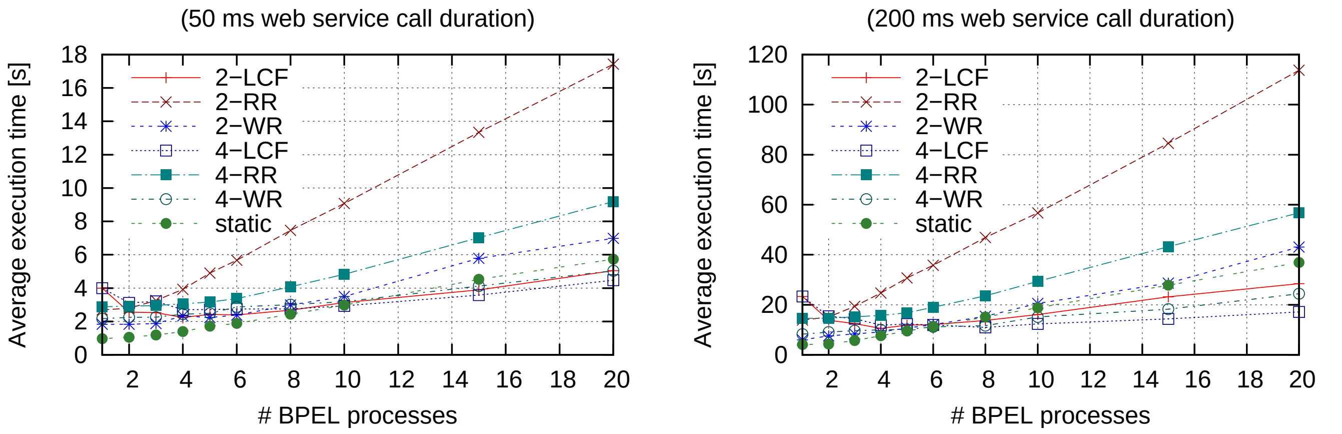

Figure 3: Average execution time of one BPEL process for an increasing number of concurrent BPEL processes on a heterogeneous collection of Web service providers. Left: 50 ms per Web service call; right: 200 ms.

performance. The WR method does not guarantee such an equal distribution of the Web service calls and is thus inferior to the others in this setup.

We also examined the overhead that is entailed by the injection of the scheduling mechanism. The overhead of the scheduler was determined by using the same Web service endpoint running the original and the transformed BPEL process. An overhead of about 15-16% (average) of the overall execution time was detected for a Web service call duration time of 50 ms. For 200 ms the overhead dropped down to 4-5% in average.

Figure 3 shows the results for a heterogeneous collection of Web service providers. It can be seen that the average execution time of a dynamic BPEL process is only reduced if many BPEL processes are executed concurrently. The problem of selecting a Web service endpoint in heterogeneous environments is to find a good distribution of the Web service calls among the available Web service endpoints which reflects their specific performance. An inefficient distribution can lead to a larger execution time of the dynamic BPEL process compared to the original one. For a small number of concurrently executed BPEL processes (1..6) this can be seen with the LCF strategy which always selects a slow Web service endpoint from the group of Web service endpoints without considering its speed.

However, the LCF scheduling algorithms provides good results for many concurrent BPEL processes. The WR scheduler provides a better average execution time for a few BPEL processes since it relies on the actual call duration time of the Web service endpoints. When many BPEL processes are executed simultaneously the WR scheduler is outperformed by the LCF scheduler. The random choice of the WR scheduler does not always guaranty an efficient load distribution. At last, the RR scheduler is not suitable for a heterogeneous configuration of Web service endpoints in any case. For scalability analysis we examined the behavior for 50 and 100 concurrently executed BPEL processes. These results were not included in the diagrams since the graph lines continue in the same proportion.

It can be observed that the experimental results of the scheduling algorithms match well with the theoretical evaluation of Section 4. As expected, the WR strategy provides the best overall solution, even though the RR and LCF strategies can be efficient in certain cases.

6. Related work

Dogdu and Mammidenna also approached the problem of scheduling Web service calls onto similar Web service providers [8]. In contrast to our work, the strategies were evaluated by discrete event simulation. Kadav et al. deal with an automatic assignment and scheduling of grid resources to workflow activities [9]. Deelman et al. give a good overview of the currently relevant workflow technologies and workflow description languages [4]. Emmerich et al. examined how scientific workflows that consist of grid services can be orchestrated with BPEL, and how they can be efficiently executed using the activeBPEL community engine [10]. In a similar work a proprietary grid workflow system is examined using a Bio-informatics application[11]. Karastoyanova et al. propose different solutions to “find and bind” Web service endpoints of a BPEL process at runtime [12]. A survey of online scheduling strategies is presented in [13]. The presented methods, mostly randomized scheduling approaches, are only applicable to homogeneous machines and assume that only one job is executed on a machine at a time. Several strategies for scheduling Web service calls are examined in [14] with the result that the utilization of an endpoint has to be considered when making a scheduling decision. A proxy approach for distributing Web service calls to multiple endpoints is proposed in [15]. Zeng et al. discuss an abstract way of service composition, using techniques of linear programming to meet specific quality of service constraints [16]. A workload model for benchmarking BPEL engines based on end-user requirements is introduced in [17]. In [18] a light-weight BPEL engine is presented to improve process execution performance.

7. Conclusions

In this article we have proposed a low-cost strategy to distribute the workload of Web service calls over several Web service providers. We have shown how a statically defined BPEL process can automatically be adapted to allow a dynamic selection of Web service endpoints at runtime. Several scheduling strategies have been examined in experiments that only require information that is collected by the BPEL engine itself. We have shown that a randomized algorithm which takes the time of recent Web service calls into account achieves good performance results. On a set of four heterogeneous Web service providers the average execution time of a dynamic BPEL process is reduced to about 27% of the time of a static BPEL process when many BPEL processes are executed concurrently. However, the performance gain is much higher when applying the scheduling mechanism to a homogeneous set of Web service providers. In future work we extend our analysis to multiple types of Web service.

References

[1] "WS-BPEL 2.0 Specification," 2007. [Online]. Available: http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html

[2] J. Pasley, "How BPEL and SOA are changing Web services development," *IEEE Internet Computing*, vol. 9, no. 3, pp. 60–67, 2005.

[3] "XML Process Definition Language Specification," 2005. [Online]. Available: http://www.wfmc.org/xpdl.html

[4] E. Deelman, D. Gannon, M. Shields, and I. Taylor, "Workflows and e-science: An overview of workflow system features and capabilities," *Future Generation Computer Systems*, vol. 25, no. 5, pp. 528–540, July 2008.

[5] "ActiveBPEL Community Edition Engine." [Online]. Available: http://www.activevos.com/community-open-source.php

[6] "Apache Tomcat Application Server Documentation." [Online]. Available: http://tomcat.apache.org/tomcat-5.5-doc/index.html

[7] "Web Services Description Language," 2001. [Online]. Available: http://www.w3.org/TR/wsdl/

[8] E. Dogdu and V. Mamidenna, "Efficient Scheduling Strategies for Web Services-Based E-Business Transactions," in *Proc. of the 6th VLDB Workshop on Technologies for E-Services (TES-05)*, ser. LNCS, vol. 3811. Springer, 2005, pp. 113–125.

[9] A. Kadav and S. K. Aggarwal, "A Workflow Editor and Scheduler for Composing Applications on Computational Grids," *Proc. of the Int. Conf. on Parallel and Distributed Systems*, vol. 2, pp. 127–132, 2006.

[10] W. Emmerich, B. Butchart, L. Chen, B. Wassermann, and S. L. Price, "Grid Service Orchestration using the Business Process Execution Language (BPEL)," *Journal of Grid Computing*, vol. 3, no. 3-4, pp. 283–304, 2005.

[11] S. W. Sorde, S. K. Aggarwal, J. Song, M. Koh, and S. See, "Modeling and Verifying Non-DAG Workflows for Computational Grids," *IEEE Congress on Services*, vol. 0, pp. 237–243, 2007.

[12] D. Karastoyanova, A. Houspanossian, M. Cilia, F. Leymann, and A. Buchmann, "Extending BPEL for Run Time Adaptability," in *Proc. of the 9th IEEE Int. EDOC Enterprise Computing Conf. (EDOC '05)*. IEEE Computer Society, 2005, pp. 15–26.

[13] S. Albers, "On Randomized Online Scheduling," in *Proc. of the 34th annual ACM Symposium on Theory of Computing (STOC '02)*. ACM, 2002, pp. 134–143.

[14] E. Alwagait and S. Ghandeharizadeh, "A Comparison of Alternative Web Service Allocation and Scheduling Policies," in *Proc. of the 2004 IEEE Int. Conf. on Services Computing (SCC '04)*. IEEE Computer Society, 2004, pp. 319–326.

[15] D. Dyachuk and R. Deters, "Transparent Scheduling of Web Services," in *Proc. of the 3rd Int. Conf. on Web Information Systems and Technologies (WEBIST 2007)*, 2007.

[16] L. Zeng, B. Benatallah, M. Dumas, J. Kalagnanam, and Q. Z. Sheng, "Quality Driven Web Services Composition," in *Proc. of the 12th Int. Conf. on World Wide Web (WWW '03)*. ACM, 2003, pp. 411–421.

[17] G. Din, K.-P. Eckert, and I. Schieferdecker, "A Workload Model for Benchmarking BPEL Engines," in *Proc. of the 2008 IEEE Int. Conf. on Software Testing Verification and Validation Workshop (ICSTW '08)*. IEEE Computer Society, 2008, pp. 356–360.

[18] T. Gunarathne, D. Premalal, T. Wijethilake, I. Kumara, and A. Kumar, "BPEL-Mora: Lightweight Embeddable Extensible BPEL Engine," in *Emerging Web Services Technology*, ser. Whitestein Series in Software Agent Technologies and Autonomic Computing, vol. 2. Birkhäuser Basel, 2007, pp. 3–20.