Load-Balancing Dynamic Service Binding in Composition Execution Engines

Mathias Björkqvist, Lydia Y. Chen
IBM Research Zurich Laboratory
8803 Rüschlikon, Switzerland
Email: {mbj,yic}@zurich.ibm.com

Walter Binder
University of Lugano
6900 Lugano, Switzerland
Email: walter.binder@usi.ch

***Abstract*—Performance and scalability of service-oriented applications, such as Web service compositions or business processes, depend on the dynamically bound services. In order to handle an increasing number of clients, load-balancing techniques are important. In this paper we assume the presence of multiple functionally equivalent services and explore different load-balancing algorithms to dynamically select service bindings with the goal to reduce average service response time. Using mathematical queueing models of service performance and simulation, we compare different service selection algorithms, including Static Lottery, Round-Robin, and Shortest-Queue. Furthermore, we propose linear and quadratic Dynamic Lottery service selection algorithms, which assign and periodically update service selection probabilities according to monitored average service response time. Our simulation environment models both stateless and stateful services and offers a wide range of service performance models with different degrees in the variation of service response time. While the Shortest-Queue algorithm performs best in simulation settings with only stateless services or low variance of service response time, the Round-Robin and Dynamic Lottery algorithms work best in settings with stateful services and high variance of service performance.**

***Keywords*-Service compositions, dynamic binding, load-balancing algorithms, performance models, simulation**

I. Introduction

Service-oriented architecture (SOA) promotes the creation of applications by composing distributed services [1], [2]. SOA has been particularly successful on the Web, where the composition of Web services produces new value-added services that are accessible to clients through Web service interfaces. Web service compositions are often represented as business processes or as workflows that are executed by a dedicated engine. In this paper, the term "composition" denotes any composition of Web services, and the term "composition execution engine (CEE)" refers to middleware for executing compositions, such as BPEL engines [3]. In the remainder of this paper, "service" stands for "Web service".

As compositions are themselves services, which may be concurrently invoked by many clients, performance and scalability of composition execution becomes critical. For example, consider compositions offered to mobile users by a mobile network operator; a large number of clients may concurrently invoke these compositions and expect timely responses. The response time of a composition is largely influenced by the response times of the composed services. Individual services invoked by a composition may become performance bottlenecks and deteriorate overall composition response time.

If there are multiple functionally equivalent services, the CEE can optimize composition execution by dispatching service invocations to the fastest services, that is, to the services with the shortest response time. In general, the CEE may not know in advance which service in a set of equivalent services has the shortest response time. Service response time depends on many different factors, such as hardware properties of the server where the service has been deployed, service implementation, network latency, and the current server load. Typically, most of these factors are not known to the CEE. While the latter factor—the current server load—is influenced by the CEE since it invokes the service, it also depends on service invocations by third parties.

The CEE can monitor its own interactions with different services. It can measure overall service response time (including network latencies, message queueing time at the server hosting the service, and service execution time) and it can keep track of the number of pending invocations (i.e., invocations where the request message has been sent, but the response message has not yet been received) for each service. Using these locally available sources of information, the CEE can implement self-optimizing service selection algorithms that aim at minimizing service response time by balancing the load between equivalent services.

In this paper, we explore the performance of different service selection algorithms. Some of them are taken from the literature on scheduling (e.g., Round-Robin [4] or Shortest-Queue [5] service selection), while others are variations of probabilistic Lottery Scheduling [6]. In the latter approach, each service is assigned a selection probability, which can be periodically updated according to monitoring information. To investigate the performance of service selection algorithms for the CEE, we use performance models based on queueing theory and build a simulator for a wide range of service environments. Specifically, we consider "stateless system", which refers to an environment with only stateless services, and "stateful system", which denotes an environment where at least one service is stateful. Our simulator models both stateless and stateful systems, and validates different service performance models with different degrees in the variation of service time.

The scientific contributions of this paper are twofold. First, we develop mathematical performance models for service selection in a CEE. Specifically, we model variations in service response time due to service invocations by third parties. Models are explored to provide closed-form analysis of service response time for certain selection algorithms and scenarios considered. Second, through detailed models and simulation, we provide first principle guidelines for choosing service selection algorithms with respect to different variations of service performance and stateless respectively stateful systems. While there are well-known results for stateless systems, we also explore stateful systems. For stateful systems, existing results in queueing theory may not hold, such that simulation is essential.

The rest of this paper is structured as follows. Section II gives an overview of the system architecture of a CEE, introduces the terminology used in this paper, and explains the underlying assumptions. Section III presents our mathematical service performance model, and Section IV discusses several service selection algorithms, which are evaluated in Section V using simulation. Section VI discusses related work, and Section VII concludes.

II. System Architecture

In this section we give an overview of the general system architecture of a CEE and explain the assumptions on which this paper is based. Before showing the system architecture, we briefly introduce our terminology. We are not striving for a complete specification of a CEE with all its components, but only introduce those components that play an important role in the paper.

- An *interface* defines a "contract" between clients and services, which specifies operations and their semantics. We assume that each interface has a unique identifier. We do not assume a specific interface description language, nor do we require a formal specification of service semantics. In this paper we do not consider service-level agreements [7], which may specify quality-of-service parameters, such as the average service execution time or service invocation costs.
- A *service* implements an interface and is deployed by a provider. It corresponds to an endpoint reference [8]. In this paper we do not consider the way services are published and discovered. We assume the CEE is aware of functionally equivalent services that implement the same interface.
- A *composition* is a program that uses services. Compositions are also known as service orchestrations. We assume that compositions are programmed against interfaces. In order to invoke a service that implements a given interface, a composition must first select a service that implements the interface. This process is called *dynamic binding*, and the selected service is said to be *bound*.
- *Clients* invoke compositions.
- The *Composition Execution Engine (CEE)* executes compositions upon client invocations.
- The *Dispatcher* is the component in the CEE that handles *dynamic binding* and *service invocation* for executing compositions. Dynamic binding is provided by the *Service Selector* sub-component, while service invocations are performed by the *Service Invoker* sub-component.

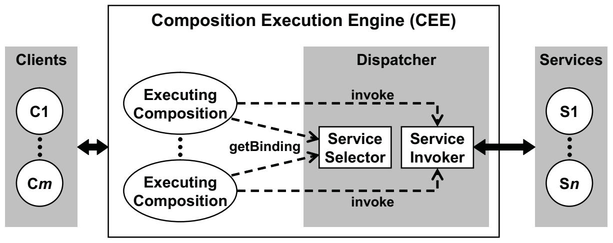


Fig. 1. System architecture

Fig. 1 illustrates the architecture of a CEE we assume in this paper. On the left hand side are clients who invoke compositions, and on the right hand side are the services invoked by the execution of the compositions. Executing compositions use the service selector for dynamic binding and afterwards invoke the bound services through the service invoker. As dynamic binding and service invocation are decoupled, service compositions can ensure that a series of service invocations target the same service, which is necessary for interacting with stateful services. Specifically, we introduce a configurable system parameter, referred to as the *state limit*, to control the series of invocation sent to the same service. The details can be found in Section V. For stateless services (i.e., state limit is one), a different service may be bound before each invocation.

This paper focuses on the dispatcher, and more concretely on the service selection algorithm. We assume that for each interface, there is a number of (functionally equivalent) services. Dynamic binding for a given interface (*getBinding* in Fig. 1) may return any of these services. The goal of the service selector is to minimize average service response time using a load balancing algorithm. That is, given an interface, it should bind a service that is expected to have short response time for subsequent service invocations. Note that in general, the service selector does not know how many times the executing composition will subsequently invoke the bound service. As a result, service selection algorithms optimized for local service invocations, such as Shortest-Queue, may result in very bad service response time.

Depending on the load-balancing algorithm, the service selector and the service invoker may share some information about the different services. For example, the service invoker may store the response time for each service invocation and may keep track of the number of pending service invocations for each service (i.e., the number of service invocations issued by the service invoker for which the response has not yet been received). The service invoker can make that information available to the service selector in order to make better dynamic binding decisions.

The response time of a service invocation depends on many factors, such as network latency, hardware and software configuration of the server where the service has been deployed, service implementation, input parameters, and the number of pending requests. The latter factor is influenced by the CEE invoking the service, as well as by third parties that may use the service as well. In the performance models considered in this paper, we take all these factors into account, but we do not explicitly distinguish between them. That is, we model that service response time is influenced by the pending requests from the CEE, and in addition may be influenced by some external factors the CEE is not aware of. In this way, our performance models remain valid when the CEE is replicated, where each replica is unaware of the other replicas. The possibility of replicating the CEE is important, to avoid that the CEE itself becomes a performance bottleneck and single point of failure.

In this paper we make a few additional simplifying assumptions. We assume that the response time of a service is not influenced by invocations of other services (with the same or with different interfaces). This implies that we do not take any bandwidth limitations of the CEE's network connection into account and do not consider that multiple services (with possibly different interfaces) may be hosted on the same server, sharing the resources of the server.

III. Performance Model

In this section we build performance models for a CEE with respect to service selection algorithms, using established methodologies and results in queueing theory. We first introduce the queueing terminology applied in this paper. Then, we discuss the corresponding queueing models and their performance under different service selection algorithms. Here, we focus on the modeling of a single interface. As all services are assumed to be independent (see end of Section II), the different service interfaces can all be modeled independently and in the same way. A CEE can select services in a static, dynamic load-oblivious, or dynamic load-aware fashion, assuming the service invoker and selector share information. Our objective is to use performance models to select and design an effective service selection algorithms, which can minimize the average response time for a wide range of services with different characteristics.

A. Preliminary of Queueing Terminology

We start by introducing Kendall's notation [9], by which elementary queueing systems are described: $A/B/k$. A denotes the distribution of the inter-arrival time of requests, B denotes the distribution of service execution time, and k denotes the number of servers. One can find analytical performance results, e.g., average response time, of a system by looking up its corresponding Kendall notation in queueing theory [9]. Note that the average service response time referred to here is composed of service execution time and waiting time, as shown in Fig. 2.

Consequently, with proper Kendall notation of a CEE, we can obtain service-wise response time, which is measured as the time from when the service invoker sends a request to a service until it receives the response, including the network latency, for a given service selection algorithm. To obtain

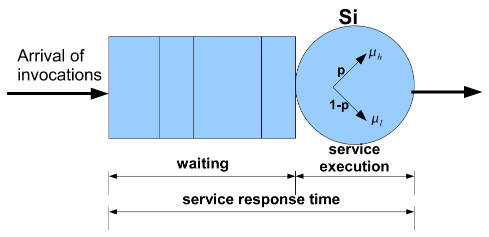


Fig. 2. Queueing system of a service from the perspective of the dispatcher in a CEE.

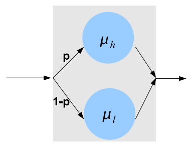
Kendall's notation of our system, we first need to know the distribution of the inter-arrival times of invocations sent to services and the distribution of the service execution time. The symbols used to represent the distributions of inter-arrival time and service execution time in this study are:

- M Exponential distribution or Poisson arrivals.
- H_2 Two-phase hyperexponential distribution, as shown in Fig. 3 (a). It has probability p to be exponentially distributed with mean rate μ_h and probability $1-p$ to be exponentially distributed with mean rate μ_l.
- E_n Erlang distribution with k phases, as shown in Fig. 3 (b). It is composed of a series of n exponential distributions with mean rate λ.

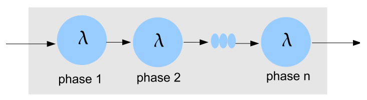
For example, $M/H_2/1$, depicted in Fig. 2, refers to a queueing system with Poisson arrivals, hyperexponentially distributed service time and one server. Actually, Fig. 2 also shows the queueing system of a service i from the perspective of the dispatcher in a CEE. With such a model, one can analyze how the service response time is affected by (1) the service execution time, which is influenced by third-party load, and (2) the waiting time, which is influenced by the invocation requests.

B. Queueing Models for a CEE

To analyze the performance of a CEE, we first characterize the distribution of the inter-arrival times of invocations received at the services. The CEE invokes services following Poisson arrivals with rate of λ per unit time. The distribution of inter-arrival times of invocations to each service i depends on the selection algorithm. For example, when selection is based on a probabilistic distribution (e.g., Static and Dynamic Lottery), the inter-arrival times of invocations sent to services are still exponentially distributed. In contrast, load-aware selection results in a non-exponential distribution that is much harder to treat using existing methodologies.



(a) Hyperexponential (H_2)



(b) Erlang (E_n)

Fig. 3. Illustrations of the H_2 and E_n distributions.

Prior to discussing the distribution of service execution time, we first clarify that service execution time is equivalent to the inverse of the service rate mentioned below. As each service could be used by other CEEs, the service rate received by a single CEE could vary greatly. When the third-party load at service i is high, the required service invocation time is longer and it implies that the service rate received is low. On the other hand, when there is low third-party load, the corresponding service rate (service execution time) is higher (shorter). Thus, we model the service execution time experienced by a CEE as a two-phase hyperexponential distribution [9] with parameters μ_l, μ_h, p and $1-p$. As shown in Fig. 3 (a), the service execution time from service i has the probability p to be drawn from a exponential distribution with mean rate μ_h, and the probability $1-p$ to be drawn from an exponential distribution with mean rate μ_l.

IV. Service Selection Algorithms

The service selection considered here is a very challenging problem in the field of queueing theory [10]. Most of the optimal selection algorithms are based on stateless services with exponentially distributed execution time, which does not model the external loads from other CEEs. However, little is known about the optimality of service selection in a system with stateful services and a non-exponential distribution, e.g., hyperexponential distribution. Nevertheless, existing queueing results can also provide some guidelines in designing service selection algorithms for stateful systems. The CEE simulation used in this paper handles both stateless and stateful services, which are controlled by the configurable parameter, state limit. To minimize the average service response time for the CEE, we investigate commonly known service selection algorithms (i.e., Round-Robin, and Shortest-Queue), and further explore lottery selection algorithms.

- Static Lottery (SL): Select each service (out of a group of n equivalent services) with the same constant probability $\frac{1}{n}$.
- Round-Robin (RR): Select services in a cyclic round-robin fashion.
- Shortest-Queue (SQ): Select the service which has the smallest number of pending requests from the CEE.
- Dynamic Lottery (DL): Select service i with probability q_i, which is updated periodically based on the observed average service response time.

The first two algorithms are load-oblivious and thus no monitoring is required; the last two algorithms are load-aware, especially Shortest-Queue. To obtain the load intensity on each service with lower monitoring overhead (than SQ), we propose to use DL selection—it periodically monitors the service response time, which is then used to estimate the load on each service, and update the lottery probabilities accordingly.

1) Static Lottery (SL): As the service is selected randomly, each service can be treated independently, and the invocations received by service i follow a Poisson distribution with rate $\lambda_i = \lambda q_i = \lambda \frac{1}{n}$. The distribution of the service execution time is two-phase hyperexponential. Thus, we can approximate the response time of service i by $M/H_2/1$ in Eq. 1,

$$W_i = \frac{\lambda_i \sigma_X^2}{2(1-\rho_i)} + \bar{X}, \quad \text{where} \tag{1}$$

$$\bar{X} = \frac{p}{\mu_h} + \frac{1-p}{\mu_l}, \quad \rho_i = \frac{\lambda_i}{1/\bar{X}}, \quad \sigma_X^2 = 2\{\frac{p}{\mu_h^2} + \frac{1-p}{\mu_l^2}\}$$

Specifically, ρ_i denotes the load intensity at service i, and $\bar{X}$ and σ_X^2 denote respecitvely the mean and variance of service execution time at service i. We use Eq. 1 to verify our simulation results for stateless systems and analyze performance of stateful systems. Parameters used in Eq. 1 can be obtained through statistical profiling on historical data, but estimation methodologies are out of the scope of this paper. Moreover, one can observe that the average service response time is dominated very much by the variance of service execution time under Static Lottery. The higher the variance σ_X^2 is, the higher the service response time may be. Furthermore, such observations can be generalized on all service selection algorithms considered.

2) Round-Robin (RR): The CEE selects services in a cyclic round-robin fashion, and thus it is load-oblivious. In a stateless system, all services receive a request every $n-1$ invocation requests, whose inter-arrival times also follow an exponential distribution. Consequently, the inter-arrival times of invocation requests for service i follow an n phase Erlang distribution under the Round-Robin algorithm. As a result, the queueing system of each service i seen by a CEE is a $E_n/H_2/1$ system. One can obtain the service response time analytically by inverting the P-K transform equation [9], which is not a trivial task. As the variance of an Erlang distribution is smaller than that of an exponential distribution, the service response time under Round-Robin is lower than under Static Lottery. Due to the lack of analytical performance analysis of RR for a stateful system, we seek evaluations using simulation.

3) Shortest-Queue (SQ): Shortest-Queue selection is shown to be an optimal algorithm in minimizing response time when (1) the system is stateless and (2) the distribution of inter-arrival times and service execution times are not highly varying [10], e.g., exponential distribution. Moreover, there are exact closed-form expressions for the response time in the case of two services [4], and approximations [5] for a large number of services, when the service execution time is exponentially distributed. However, there are neither performance guarantees nor analytical results for using SQ in a stateful system with highly varying service execution time.

4) Dynamic Lottery (DL): DL changes the service selection probabilities at discrete intervals, indexed by t. Each interval takes T unit time, which can be tuned to optimize the performance of DL. At interval t, DL selects service i with the probability $q_i(t)$, where $0 < q_i(t) \leq 1$, and $\sum_{i=1}^{n} q_i(t) = 1$. The selection probability, $q_i(t)$, is updated at the beginning of the interval t, based on the weighted average of observed service response times during a few past intervals, denoted by $W_i^{ob}(t)$. We use an auto-regressive model [11] to compute

$$W_i^{ob}(t) = aW_i(t-1) + bW_i(t-2), \tag{2}$$

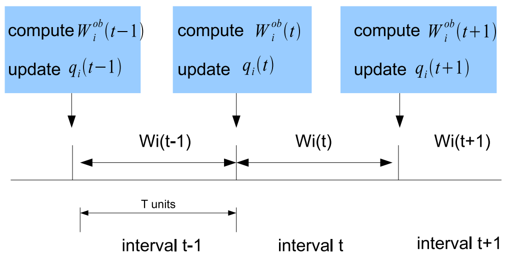


Fig. 4. Schematics of Dynamic Lottery service selection.

where a, b are coefficients and $W_i(t)$ is the average response time of a service invocation completed during interval t at service i. We also refer to $W_i(t)$ as the interval-wise average response time. The specific values of a and b can be estimated by historical data. The schematics of computing $W_i^{ob}(t)$ and $q_i(t)$ are illustrated in Fig. 4.

The weighted response time $W_i^{ob}(t)$ reflects the load at each service. A high value of $W_i^{ob}(t)$ indicates high load at service i, which might be explained by high third-party loads imposed by other CEEs and/or high loads from the local CEE. Consequently, one would like to reduce the invocation sent to service i. There is an inverse relationship between the service invocation rate and the response time. As a result, we propose to use the following two algorithms to adjust $q_i(t)$ as functions of $\frac{1}{W_i^{ob}(t)}$ and $\frac{1}{(W_i^{ob}(t))^2}$:

$$DL_1 : q_i(t) = \frac{1/W_i^{ob}(t)}{\sum_i 1/W_i^{ob}(t)} \tag{3}$$

$$DL_2 : q_i(t) = \frac{1/(W_i^{ob}(t))^2}{\sum_i 1/(W_i^{ob}(t))^2} \tag{4}$$

We refer to the Dynamic Lottery selection algorithms, where selection probabilities are updated according to the schemes in Eq. (3) and (4), as DL_1 and DL_2. DL_1 uses linear weights and DL_2 uses quadratic weights on $\frac{1}{W_i^{ob}(t)}$. As DL_1 is only linearly inverse with respect to the weighted response time, the variance of the control variable, $q_i(t)$, is smaller than DL_2, which uses the quadratic inverse of $W_i^{ob}(t)$. DL_2 can be more effective when the service execution time is highly varying; on the other hand, DL_2 can hamper the stability by amplifying the transient load changes. In Section V, we further discuss the advantages and disadvantages of DL_1 and DL_2 with respect to different interval lengths and system scenarios.

V. Evaluation

In the following text, we evaluate the different service selection algorithms in various scenarios. The performance metrics of interest are the average response time and the interval-wise response time ($W_i(t)$) of service i.

A. Simulation Setup

In our event-driven Java simulation environment, there are $m = 1000$ compositions executing in a single CEE, sending service invocation requests at a rate $\lambda = 1.3$ per unit time. The time unit considered is a millisecond. There are 10 services available, i.e., $n = 10$. We introduce a configurable system parameter, referred to as state limit, to control the series of service invocations targeting at the same bound service. For example, when the state limit equals one, services are stateless and each service invocation may have a new binding. When the state limit equals k, each selected service is subsequently invoked k times.

We consider fifteen different system scenarios for a CEE, based on (1) variation of service execution time, and (2) the configurable state limit. The variations in service execution time are defined as low, medium, and high, and the corresponding μ_h, μ_l, and p to generate the hyperexponential distributions are listed in Table I. A high variance in service execution time could imply services are used by other CEEs that have more dynamic interactions. In all system scenarios, we have the same average service execution time, i.e., $\bar{X}$ in Eq. 1. One can obtain the values of the variance by using σ_X^2 in Eq. 1. Note that all services are assumed to use the same set of parameters in all scenarios. The state limits applied are 1, 5, 10, 40, and 100. When the the state limit is 1, we model a stateless system, whereas the rest represent different stateful systems. We summarize the parameters and naming convention of the scenarios in Table I.

B. Comparisons of Selection Algorithms

We evaluate five selection strategies: Static Lottery, Round-Robin, Shortest Queue, Dynamic Lottery with linear weights, and Dynamic Lottery with quadratic weights. We first discuss how to choose a proper interval length for the two dynamic lottery algorithms. Secondly, we compare the pros and cons of all five selection algorithms in the different system scenarios.

C. Choice of interval length for DL_1 and DL_2

To implement DL_1 and DL_2 optimally, we consider five different interval lengths, $T = 10$, 25, 50, 75 and 100 ms. We define $W_i^{ob}(t)$ (explained in Eq. 2) used for updating the lottery probabilities of all services as

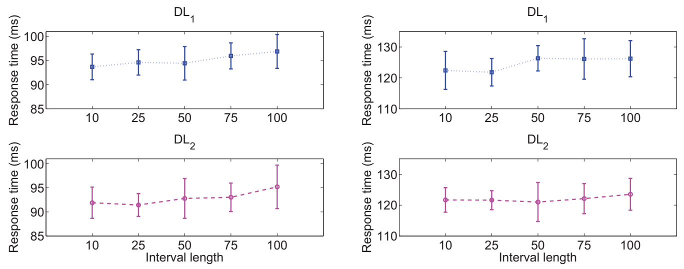
$$W_i^{ob}(t) = 0.9W_i(t-1) + 0.1W_i(t-2).$$

Note that the coefficients may be chosen differently, to optimize DL_1 and DL_2. The criterion of choosing a proper interval length are the average and standard deviation of response times.

We use Exp. 10 and Exp. 15 for the purpose of illustration. We replicate ten simulation runs of DL_1 and DL_2 for the intervals considered and plot the average response times and their corresponding standard deviations in Fig. 5. The interval length can result into roughly a 5% difference in the average and a 10% difference in the standard deviation. Due to the quadratic weight, DL_2 is more aggressive in adjusting the lottery probability, $q_i(t)$, compared to DL_1. As a result, one can adjust the lottery probabilities in DL_2 less frequently to reduce the risk of over-disturbing the system. On the other hand, as DL_1 adjusts the lottery probabilities of services in a linear fashion, it less sensitive to the load changes. Consequently,

TABLE I
SYSTEM SCENARIOS AND EXPERIMENT NUMBERING.

Exp. no.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
State limit	1	5	10	40	100	1	5	10	40	100	1	5	10	40	100
μ_h	1/10					1/8.71					1/12				
μ_l	1/5					1/3					1/2				
p	.4					.7					.5				
variance of service execution time	Low					Medium					High				



(a) Exp. 10 (b) Exp. 15

Fig. 5. Average service response time for different interval lengths for DL_1 and DL_2 in Exp. 10 and Exp 15.

DL_1 can perform better with a shorter interval length, $T = 10$, in cases where the state limit is higher, e.g., in Exp. 15. From our experience, we suggest that the optimal interval length for DL_1 is shorter than for DL_2. In practice, the optimal control length for DL_1 and DL_2 can be estimated and tuned by historical data.

1) Average Response Time : We summarize the average response time of Exp. 1–15 under all five service selection algorithms in Fig. 6 (a)–(c). For DL_1 and DL_2, we report the average response time for the best interval length. As we are interested in finding algorithms which can result in low service response time, we only show response time up to 120 ms, the upper limit in Fig. 6 (a)–(c). As a result, one can not find all values of the response times under Static Lottery, especially for experiments with state limits greater than one. There are two general trends observed:

- For the selection algorithms considered here, the average response time increases with the variance of service execution time for a given state limit. One can find similar analytical results in queueing theory in the case where the state limit equals to 1 (stateless). For example, the average service response time for SQ with state limit 10 (i.e., SQ in Exp. 3, Exp. 8, and Exp. 13) is increasing from Exp. 3 to Exp. 8, and then to Exp. 13, corresponding to the increase in the variance of service execution time.
- For most service section algorithms, the average response time increases with the state limit for a given variance of service execution time. The exceptions are applying DL_1 and DL_2 in a stateless system. Take Fig. 6 (b) as an example. The average response time of SL, RR, SQ, DL_1, and DL_2 increases from Exp. 6 (state limit = 5) to Exp. 10 (state limit = 100).

With respect to each particular service selection algorithm, we have following observations:

- Static Lottery: SL selection has the worst performance and it deteriorates much more with a higher value of state limit, i.e., 40 and 100, compared to other service selection algorithms. SL is too naive to deliver scalable performance. Specifically, when the state limit is one, we can apply Eq. 1 to obtain response time for a stateless system, i.e., Exp. 1, Exp. 6, and Exp. 11 in our evaluation. Correspondingly, we obtain the following values $\{86.44, 87.25, 113\}$, which match very well with the simulation results. This confirms the soundness of our simulator.
- Round-Robin: The response times under RR are quite stable across different state limits. When the state limit is high, e.g., 100, the response time is minimized under RR, especially in the case of higher variance of service execution time. Specifically, in Exp. 15, RR—a load oblivious algorithm—achieves the lowest response time. However, when the state limit is low, e.g., 5 and 10, RR has a much higher response time than the load-aware selection algorithms, i.e., SQ, DL_1, and DL_2.
- Shortest-Queue: In a stateless system, SQ has minimal service response time, as predicted by existing queueing results [9]. Its performance degrades most with the increasing state limit, compared to other service selection algorithms. When the state limit increases from 1 to 100, the response time under SQ increases by factor 10, whereas the response time only degrade by factor 2 for RR. SQ is known to be greedy and only optimizes "current" load. SQ is not an optimal service selection algorithm in a stateful system or in the case of highly varying service execution times.
- Dynamic Lottery: Both dynamic lottery algorithms, DL_1 and DL_2, perform poorly in stateless systems. In Exp. 1, Exp. 6, and Exp. 11, DL_2 has even higher response time than the Static Lottery algorithm, while DL_1 is only slightly better than SL. However, in a stateful system, DL_1 and DL_2 are the second best or the best selection algorithms in all system scenarios considered here. The adjustable length of the control interval in DL_1 and DL_2 can be well leveraged to handle different system scenarios encountered by a CEE, such as the state limit and variance of service execution time. This leads us to conclude that DL_1 and DL_2 can be used as self-optimizing service selection algorithms in stateful systems.

2) Interval-wise Response Time: We now present the dynamics of the interval-wise service response time. We focus on $W_i(t)$ of the system scenario specified in Exp. 10. Here, an interval takes $T = 10$ ms. Fig. 7 illustrates snapshots of service 2, 4, and 8 from the beginning of the simulation to interval 30000. To increase the readability of the figures, we smooth out the original time series by taking the average of

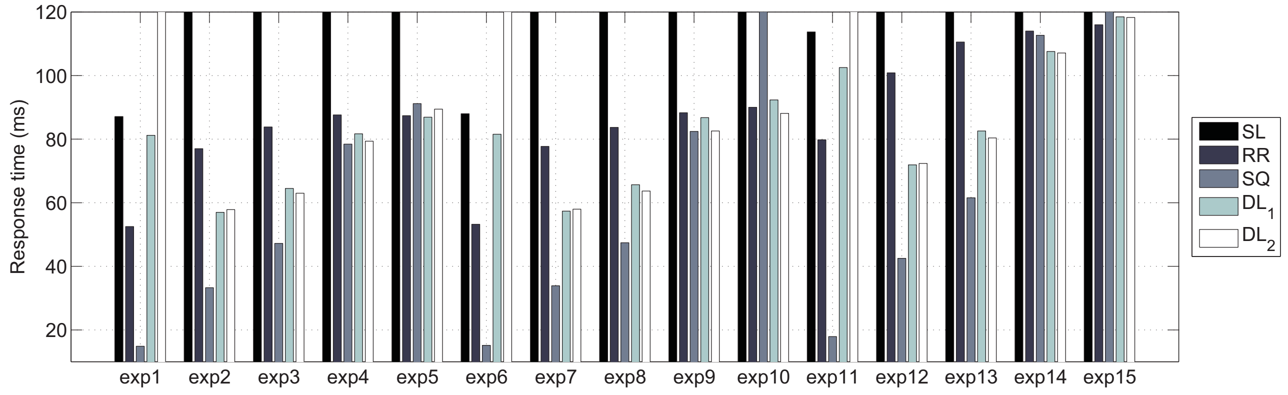
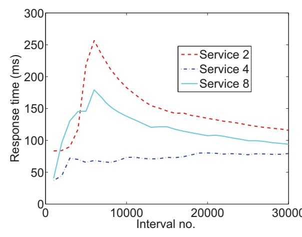
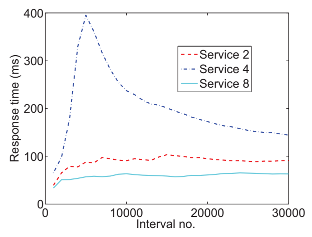


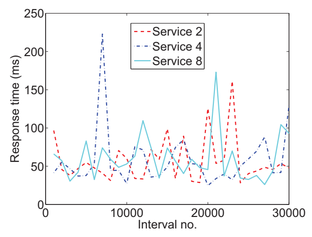
Fig. 6. Average service response time of different service selection algorithms.



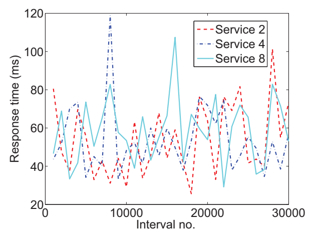
(a) Round-Robin



(b) Shortest-Queue



(c) Dynamic Lottery (DL_1)



(d) Dynamic Lottery (DL_2)

Fig. 7. Interval-wise response time of different selection algorithms under Exp. 10. Services 2, 4, and 8 have been arbitrarily selected among the 10 equivalent services, in order not to overload the figures with too many data series.

every 1000 interval-wise response time.

For Round-Robin, service 4 has very stable interval-wise response time, whereas service 2 and 8 have high response time for a quite long period. In contrast to RR, service 4 under SQ has very high response times for a long period of time, while the response time from service 2 and 8 are fairly stable. In principle, the response time of most services under RR and SQ are stable, but they could not responsively react/respond to inappropriate service binding selections, especially in a stateful system. Therefore, it takes a long time to even out the high loads on some services by redirecting requests to other services with low loads. On the other hand, the service response time under DL_1 and DL_2 are highly varying, due to the dynamically changing lottery probabilities. The response time under DL_2 oscillates more than under DL_1, but the maximum response time of DL_1 is higher than DL_2. Overall, one can observe that the loads among all services are more balanced under DL_1 and DL_2 than *RR* and *SQ*, especially when the state limit and the variation of service execution time is high.

VI. Related Work

In this section, we discuss related work on dynamic binding in CEEs and on scheduling algorithms.

Most of the related work in the area addresses dynamic binding for compositions expressed in BPEL, that is, for business processes, since BPEL is a de facto standard and there are many implementations of BPEL engines. While BPEL supports dynamic binding by partner link assignment, it is neither possible to add new services at runtime, nor to change the service selection algorithm at runtime. Furthermore, in BPEL, dynamic binding is coupled with process business logic.

VieDAME [12] is a service monitoring and selection system based on aspect-oriented programming that intercepts SOAP messages and is able to dynamically replace services used in a business process. It monitors process execution and stores the gathered data in a database. VieDAME requires that the services are registered in a repository and marked for monitoring and eventual replacement.

Dynamo [13] relies on an aspect-oriented engine extension of ActiveBPEL engine to support monitoring and failure recovery. Dynamo uses two domain specific languages, the Web Service Constraint Language (WSCoL) and the Web Service Recovery Language (WSReL) for specifying monitoring and recovery rules.

The find-and-bind mechanism [14] extends the BPEL language with enhanced support for dynamic binding, where the service to be bound is discovered at runtime.

RobustBPEL2 [15] provides dynamic binding with proxies. All services are invoked through proxies, which are also responsible for monitoring. Service invocation through proxies introduces some overhead in the execution of business processes.

An approach to optimize system performance, taking hardware resources into account, is presented in [16]. End-to-end service response time requirements are decomposed into atomic service response time requirements. Afterwards, atomic services with similar quality-of-service requirements are collocated on machines with similar utilization characteristics, which helps reduce the number of machines used.

In [17] BPEL processes are automatically transformed to interact with a separate system that handles dynamic binding. That system uses Dynamic Lottery service selection corresponding to the DL_1 algorithm in this paper. However, the service performance model is limited and a comparison with other service selection algorithms, such as Shortest-Queue selection, is missing.

In this paper, we did not integrate service selection algorithms into any particular CEE, but we evaluated different algorithms with extensive simulation, using a mathematical queueing model of service performance. Hence, the results presented in this paper are independent from and complementary to the aforementioned systems.

Dynamic Lottery service selection is closely related to lottery scheduling in operating systems [6]. In lottery scheduling, each operating system process receives a number of lottery tickets proportional to its importance. The scheduler randomly selects a ticket and schedules the process corresponding to the ticket. Hence, in a probabilistic manner, important processes receive more CPU than less important processes. Tickets are not numbered; it is sufficient to maintain the number of tickets per process. Tickets can be transferred between processes so as to reflect changes in process priorities. Our Dynamic Lottery service selection algorithms could be easily adapted to use tickets instead of probabilities.

VII. Conclusions

Service compositions are widely used for providing value-added services on the Web. When the service compositions are themselves published as Web services and made accessible to many clients that may concurrently invoke them, performance and scalability become important. In order to prevent that individual services in a composition become performance bottlenecks, the CEE can dynamically bind different alternative services, presuming that they are functionally equivalent. Hence, dynamic binding becomes a load-balancing problem.

In this paper we explore different dynamic binding algorithms using a mathematical model of service performance based on results in queueing theory. In this model, the service response time depends both on prior service selection and binding decisions by the CEE, as well as on service invocations by third parties. Our model is suited for stateless services, where a different service can be bound upon each invocation, as well as for stateful services, where a service can be bound for a series of service invocations.

With the aid of thorough simulation, we compare different service selection algorithms, such as Round-Robin, Shortest-Queue, as well as Static and Dynamic Lottery selection in systems with different degrees of third-party disturbance and state limits. With Static Lottery selection, each service has a fixed selection probability and it has poor performance, whereas Dynamic Lottery selection is a self-optimizing algorithm that periodically recomputes the service selection probabilities based on the recently monitored average service response time.

If there are no service invocations by third parties, and if a different service can be bound upon each invocation (i.e., assuming all services are stateless), it is known that Shortest-Queue selection is optimal [5]. As new results, we show how the performance of Shortest-Queue selection degrades in complex simulation settings, which take service invocations by third parties and stateful services into account. Compared to Shortest-Queue and Round-Robin, the self-optimizing Dynamic Lottery selection algorithms, DL_1 and DL_2, show robust performance in stateful systems, where service execution times may be highly varying due to third party workload.

In our ongoing research, we are working on improving scalability by replication of the CEE. Furthermore, we are considering self-optimization in a cloud computing setting, where the number of replicas of the CEE and of services can be adjusted at runtime. In addition, we will take service provisioning costs into account.

References

[1] G. Alonso, F. Casati, H. A. Kuno, and V. Machiraju, *Web Services - Concepts, Architectures and Applications*, ser. Data-Centric Systems and Applications. Springer, 2004.

[2] M. P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann, "Service-Oriented Computing: A Research Roadmap," *Int. J. Cooperative Inf. Syst.*, vol. 17, no. 2, pp. 223–255, 2008.

[3] BPEL, "BPEL 2.0 standard specification," http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf.

[4] R. D. Nelson and T. K. Philips, "An approximation to the response time for shortest queue routing," *SIGMETRICS Perform. Eval. Rev.*, vol. 17, no. 1, pp. 181–189, 1989.

[5] V. Gupta, M. Harchol-Balter, K. Sigman, and W. Whitt, "Analysis of join-the-shortest-queue routing for web server farms," *Perform. Eval.*, vol. 64, no. 9-12, pp. 1062–1081, 2007.

[6] C. A. Waldspurger and W. E. Weihl, "Lottery Scheduling: Flexible Proportional-Share Resource Management," in *OSDI:Proceedings of the First Symposium on Operating System Design and Implementation*, 1994, pp. 1–11.

[7] Open Grid Forum, "WS-Agreement specification," http://www.ogf.org/documents/GFD.107.pdf.

[8] WS-Addressing, "WS-Addresing standard specification," http://www.w3.org/Submission/ws-addressing/.

[9] L. Kleinrock, *Queueing Systmes*. Wiley, 1976.

[10] W. Whitt, "Deciding which queue to join: Some counterexamples," *Operation Research*, vol. 34, no. 1, 1986.

[11] G. Box, G. Jenkins, and G. Reinsel, *Time Series Analysis: Forecasting & Control*. Wiley, 2008.

[12] O. Moser, F. Rosenberg, and S. Dustdar, "Non-intrusive Monitoring and Service Adaptation for WS-BPEL," in *WWW '08: Proceeding of the 17th international conference on World Wide Web*. New York, NY, USA: ACM, 2008, pp. 815–824.

[13] L. Baresi, C. Ghezzi, and S. Guinea, "Towards Self-healing Composition of Services," in *Contributions to Ubiquitous Computing*. Springer Berlin / Heidelberg, 2007, pp. 27–46.

[14] D. Karastoyanova, A. Houspanossian, M. Cilia, F. Leymann, and A. Buchmann, "Extending BPEL for Run Time Adaptability," in *EDOC '05: Proceedings of the Ninth IEEE International EDOC Enterprise Computing Conference*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 15–26.

[15] O. Ezenwoye and S. M. Sadjadi, "A Proxy-Based Approach to Enhancing the Autonomic Behavior in Composite Services," *JNW*, vol. 3, no. 5, pp. 42–53, 2008.

[16] C. Zhang, R. N. Chang, C.-S. Perng, E. So, C. Tang, and T. Tao, "QoS-Aware Optimization of Composite-Service Fulfillment Policy," in *SCC '07: IEEE International Conference on Services Computing*, 2007, pp. 11–19.

[17] A. Mosincat and W. Binder, "Enhancing BPEL Processes with Self-tuning Behavior," in *SOCA '09: IEEE International Conference on Service-Oriented Computing and Applications*, 2009, pp. 210–217.