Intelligent Overload Control for Composite Web Services

Pieter J. Meulenhoff[1], Dennis R. Ostendorf[2], Miroslav Živković[1], Hendrik B. Meeuwissen[1], and Bart M.M. Gijsen[1]

[1] TNO ICT, P.O. Box 5050, 2600 GB Delft, The Netherlands
{pieter.meulenhoff,miroslav.zivkovic}@tno.nl,
{erik.meeuwissen,bart.gijsen}@tno.nl
[2] Quintiq, 's-Hertogenbosch, The Netherlands

Abstract. In this paper, we analyze overload control for composite web services in service oriented architectures by an orchestrating broker, and propose two practical access control rules which effectively mitigate the effects of severe overloads at some web services in the composite service. These two rules aim to keep overall web service performance (in terms of end-to-end response time) and availability at agreed quality of service levels. We present the theoretical background and design of these access control rules as well as performance evaluation results obtained by both simulation and experiments. We show that our access control rules significantly improve performance and availability of composite web services.

Keywords: Availability, Performance, Quality of Service, Service Oriented Architecture, Web Admission Control, Web Service Composition, Web Service Orchestration.

1 Introduction

Service oriented architectures (SOAs), based on Web Service technology, are becoming increasingly popular for the development of new applications due to the promises of easier development and shorter time-to-market. These so-called SOA-based composite services are offered by service providers, and typically consist of multiple web services, developed by third parties, which are executed in multiple administrative domains.

Currently, service developers and providers focus on the functional aspects of composite web services. However, too little attention is paid to the non-functional aspects of composite web services such as availability, performance, and reliability.

Since several composite web services can make use of the same web services, these popular web services used by multiple composite web services may experience high demand, resulting in more requests than they can handle, leading to degradation of all services that rely on these web services. These overload situations lead to reduced availability as well as higher response times, resulting in degraded quality as perceived by end users.

This paper concentrates on improving performance and availability of *composite* web services. In particular, a solution is proposed to improve the quality as perceived

L. Baresi, C.-H. Chi, and J. Suzuki (Eds.): ICSOC-ServiceWave 2009, LNCS 5900, pp. 34–49, 2009.

by end users by increasing the average number of successfully served requests per second. This solution is based on intelligently preventing overload on any one of the services in the composition, by denying service to specific requests based on dynamic admission control rules.

To illustrate our problem setting, Fig. 1 shows a simplified SOA architecture with an orchestrating web service, also referred to as an orchestrating broker. Let us suppose that the composite web service consists of three web services identified by W1 thru W3.

The broker consists of a scheduler and a controller. The scheduler determines the order of the jobs submitted to web services W1 thru W3, since it may be different per client. Each web service, W1 thru W3, has implemented the Web Admission Control (WAC) mechanism. The broker's controller keeps track of the total request execution time, and decides if the latency is within the required limit.

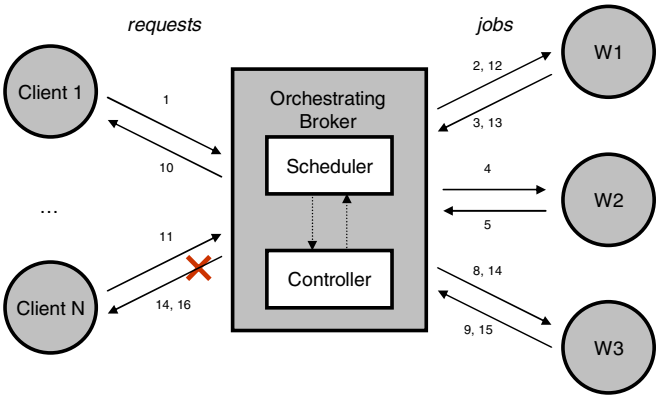

Fig. 1. Jobs for client requests are routed through a network of web services (W1, W2 and W3) by an orchestrating broker

To illustrate normal operation, let us suppose that a request from Client 1 (#1) arrives at the broker. The scheduler analyses the request, and determines that the web service W1, W2, and W3 should be invoked in that order. If the total execution time of the request is less than the required limit, the job is delegated to component W1 (#2). Before actually executing the job in W1, the WAC mechanism decides that W1 is not in overload and executes the job. On the response (#3) from W1, the scheduler checks with the controller that the total latency is less than the required limit and invokes the next job at W2 (#4). This is repeated until all web services are invoked, and the response (#10) to Client 1 is given within a maximum amount of time.

To illustrate an overload situation, let us suppose that a request from Client N (#11) arrives at the broker. The scheduler analyses the request, and determines that the web services W1 and W2 should be invoked in that order. When the job is delegated to web service W1 (#12), the WAC mechanism in W1 decides that W1 is not in overload and the job is executed. On the response (#13) from W1, the scheduler

delegates the next job to W2 (#14). The WAC mechanism in W2 denies the job as W2 is in overload, and an unavailable message is returned to the broker (#15). As a result, the broker is able to respond to Client N with a service unavailable message (#16) within the maximum amount of time as well as to prevent escalation of the overload situation of W2. Obviously, in this described overload situation resources of web service W1 have been wasted.

Providers of web services W1 thru W3 may apply different state-of-the-art techniques, such as overdimensioning of computing resources, load balancing, and caching, to prevent overload in their own domain. *In this paper we focus on the use of admission control in the web services in combination with a simple response time limit check in the orchestrating broker to prevent the composite web service from becoming generally unavailable in an overload situation*. Admission control is already widely used in telecommunications. Research has also been performed on the use of admission control for Web *Servers*; see for instance [1]-[3], [7]. The use of WAC to prevent overload for stand alone Web *Services* has been discussed in [4]-[5]. In the field of composite web services several contributions have been made more recently, focusing on web service scheduling; for instance in [8]-[9]. However, to the best of the knowledge of the authors admission control schemes that include awareness of the state of the workflow in a composition of web services, have not been published yet.

Specifically, we investigate how each individual web service can intelligently deny service to some of the jobs in the system in order to *maximize* the number of client requests for which the entire composite web service is available with a given maximal response time. Each composite web service is responsible for preventing it from collapsing in overload situations, and with it the entire composite web service. We thereby assume that the broker is not a single point of failure, i.e. that it can instantly serve and process all requests and jobs. In our solution to control quality of composite web services, mathematically derived using queuing theory, denial of service will typically occur when the number of active jobs at specific web services reaches the allowed maximum. As a result, we serve as many client requests as possible with the requested end-user perceived quality, including a guarantee on the maximum response time.

The rest of the paper is organized as follows. In Section 2, we define the mathematical foundation of the admission control problem. In Sections 3 and 4, two algorithms for admission control by the web services are derived from the results in Section 2. In Section 5, the simulation setup to investigate our solutions is described as well as two simulation cases. In Section 6, the results of an experimental validation are described. In Section 7, we end with conclusions and suggestions for the future work.

2 Mathematical Foundation for Admission Control

In this section, we will derive a queueing model of an composition of web services, including an orchestrating web service (broker), see Fig. 1. This queueing model forms the mathematical foundation for our access control rules.

Let us suppose that the composite web service consists of web services from the set $\boldsymbol{W} = \{W_1, W_2, \ldots, W_N\}$. In general, the $W_j \in \boldsymbol{W}$ may be composite web services

themselves. The incoming client requests at the broker are composed of jobs to be sequentially executed by a chain of web services from the set $\boldsymbol{W}$. Thus, each job within the request is served by a single web service. Since the broker controls different composite web services, the order in which jobs are executed may differ per client request. The broker tracks job execution on a per request basis.

In practice, web services serve jobs using threading, which could be modeled using a round-robin (RR) service discipline in which jobs are served for a small period of time ($\delta \rightarrow 0$) and are then preempted and returned to the back of the queue. Since $\delta \rightarrow 0$, assuming there are n jobs with the same service rate μ_w, the per job service rate is μ_w/n. To simplify analysis, this process is modeled as an (egalitarian) processor sharing (PS) service discipline.

The service time distribution of web service W_j is assumed to be exponential with parameter μ_j. Jobs arrive at web service W_j with arrival rate λ_j and the web service load is defined as $\rho_j = \lambda_j/\mu_j$.

We define the latency L_i of an incoming client request i as the total time it takes for a request to be served. The sojourn time (i.e. time spent in the system) of job j at web service W_j from request i is denoted by S_{ij}. We ignore possible delay due to network traffic and broker activity, so it holds that

$$L_i = \sum_j S_{ij} \tag{1}$$

The clients are willing to wait only a limited amount of time for the request(s) to the composite web service to be completed. Within the SOA architecture, Service Level Agreements (SLAs) can be defined between the clients and the provider of the composite web service in order to quantify whether a request has been successful or not. For example, the SLA may contain the description that a client request i is considered successful when its latency L_i is smaller than maximum latency L_{max}. The maximum latency tolerated by clients may depend on the application itself. Some studies [6] show that users are on average willing to wait up to eight to ten seconds for the response from a website. However, atomic commercial transactions may require latencies that are much shorter [1]. The same SLA negotiation can be done between the broker and each composite service. An existing standard that serves as inspiration is WS Reliability [10]. Using the WS Reliability standard it is possible to give jobs so called 'expiry times', which define the maximum time it may take to receive a response.

We denote by c_j a maximum number of jobs allowed to be served simultaneously by web service W_j. When c_j requests are served and the next job arrives it is denied service by the admission control rules at the web service. This admission control rule for web service W_j can be modeled by the blocking probability p_{cj}. Since our objective is to serve as many requests as possible (within L_{max}) in an overload situation, our goal is to find the optimal values of the c_j.

To further simplify analysis, we assume that the web services W_j have the same values of c_j, λ_j, $p_{cj,}$ and μ_j, denoted as c, λ, p_c and μ, respectively. We address this optimization problem by modeling the web services $W_j \in \boldsymbol{W}$ as an *M/M/1/c* Processor

Sharing Queue (PSQ). It is generally known that the blocking probability of the *M/M/1/c PSQ* equals

$$p_c = \frac{\rho^c}{\sum_{k=0}^{c} \rho^k} \tag{2}$$

And that the expected sojourn time at each of the web services equals

$$E(S) = \frac{1/\mu}{1-\rho(1-p_c)} \tag{3}$$

In the subsequent sections, two dynamic admission control algorithms S and D are derived from the model discussed in this section.

3 Dynamic Admission Control Algorithm *S*

The basic underlying principle of algorithm S is that the expected sojourn time $E(S)$ of a job in a web service should be less than or equal to the average available time for the jobs within the request. Thus, the problem of serving the client request within $L_{\max}$ is split up in consecutive steps. In each step, a limit on the expected sojourn time is calculated in the following way.

The broker, which is the only component that `knows' the structure of the request, divides the total allowed latency $L_{\max}$ over all jobs. The moment t^* when a request enters the broker the due date for the next job j^* is calculated. First, the total remaining time for this request, i.e. $\mathrm{L}_{\max} - \sum_{j=1}^{j^*-1} S_{ij}$, is determined. Then, it is divided over all remaining jobs in proportion to their service requirements. Let D_{ij^*} be the due date of job j^* from request i, let J_i be the total number of jobs from request i, let t^* be the time at which the due date for job j^* is calculated, and let v_{ij} denote the expected service time of job j from request i. Now the following relation holds:

$$D_{ij^*} = t^* + \left(\mathrm{L}_{\max} - \sum_{j=1}^{j^*-1} S_{ij}\right) \frac{v_{ij^*}}{\sum_{j=j^*}^{J_i} v_{ij}}$$

As a result, the remaining time for job j from request i at time t is given by $R_{ij}(t) = D_{ij} - t$. When the total remaining time of a request is less than zero, the request is discarded by the broker and the client is notified. Let $\overline{R}$ denote the average remaining available service time of all jobs in the Web Service $R_{ij}(t)$. Now dynamic admission control algorithm S is derived based on the following constraint: *the expected sojourn time E(S) of a job in a web service should be less than or equal to the average available time*. Now our optimization problem is defined as follows:

$$\max_{c} \left\{ c : E(S) \le \overline{R} \right\} \tag{4}$$

In (4), both c and $\overline{R}$ are time-dependent, but we omit this to simplify our notation. Computation of $\overline{R}$ is straightforward since due times of all jobs within the composite service are known.

Substituting (3) in (4) yields:

$$\max_{c} \left\{ c : \frac{1/\mu}{1-\rho(1-p_c)} \le \overline{R} \right\} \tag{5}$$

Substituting (2) in (5) yields:

$$\max_{c} \left\{ c : c \le \log_{\rho}(1+\mu\overline{R}(\rho-1)) \right\} \quad for \quad \rho > 1 \tag{6}$$

Therefore, the admission control algorithm S is now defined as:

Allow arriving jobs service if ρ<1 or $n \le \log(1+\mu\overline{R}(\rho-1))$ *still holds after the new job is allowed service.*

In the remainder of this section, we discuss two issues of algorithm *S*. In order to compute c the value of ρ is needed and thus the values of λ and μ as well. It is assumed that the service requirement rate μ is known, but the value of λ is not. The arrival process (of a web service) will in reality not be known and thus must be estimated. Therefore, the question arises what is the time period to estimate λ and how to estimate this value.

Another issue is that the arrival rate is explicitly used to estimate the value of c. Intuitively the number of jobs, which can be simultaneously served, does not depend on the number of jobs which arrive at the system. The web service is capable of simultaneously serving c jobs. The blocking probability corrects for this fact, but further investigation of this issue is required.

In the next section, an alternative dynamic admission control rule is derived, in which the arrival rate λ (and hence ρ) is not used to determine the maximum value of the number of jobs allowed.

4 Dynamic Admission Control Algorithm *D*

The goal of algorithm *D* is to implement admission control without knowledge of the arrival rate λ. This algorithm is based on the relaxed constraint that only the average job has to be completed on time. Theoretically, the average job completes on time when the number of jobs in the system remains the same for the entire service time of each job. Although jobs may enter the jobs may enter the system or depart from the system, we investigate whether effective admission control is possible under the assumption that the number of jobs remains the same.

When the number of jobs n in the queue is assumed to be constant, the expected sojourn time for a job equals n/μ. When all jobs must be served before their due dates the problem is defined as follows:

$$\max_c \left\{ c : E(S) \le R_{ij}, \text{for all jobs in service} \right\} \tag{7}$$

In our case $E(S)$ equals c/μ, and R_{ij} is replaced by $\overline{R}$, where $\overline{R}$ determines the average remaining available service time for all jobs in service. These relaxations lead to the following optimization problem:

$$\max_c \left\{ c : \frac{c}{\mu} \le \overline{R} \right\}, \tag{8}$$

The solution of this trivial problem yields $c = \mu\overline{R}$. Hence we define the more practical admission control algorithm D as follows:

Allow arriving jobs service if $n \le \mu\overline{R}$ *still holds after the arriving job is allowed service.*

Note that for the calculation of the admission control parameter c, the arrival rate (and thus ρ) is not needed. This is a major advantage from a practical point of view compared to algorithm S.

5 Simulation Setup

A discrete-event simulation model is constructed to evaluate the proposed admission control algorithms. The model is implemented using the software package eM-Plant see [11]. The simulation model basically consists of four components, see Fig. 2. Component 'Client' generates new requests according to a Poisson process with rate λ. Requests are dispatched through the network by component 'Broker'. After a request has been generated a request type is randomly assigned, to indicate which web services need to be visited. Each web service is an instance of component 'WS'. The completed or denied requests arrive at component 'Output', where relevant data is collected.

When a job is sent to one of the web services in the composition, the web service checks whether it is allowed or denied service. In case admission control is not used, all incoming jobs are allowed. When admission control is used, the web service uses an access control rule to decide whether the incoming job may be served or not. Fig. 3 illustrates the flowchart of the broker component in case of admission control. When a new request comes in, the broker determines whether the latency of this request has already reached its limit, i.e. the remaining time for the request is less than zero. If the limit is reached, the request is denied service and sent to the output component. It may happen that the request has been allowed by the broker, but still the web service itself can not serve the request. Even when the remaining time is greater than zero, the broker determines whether the request has previously been denied service by the web

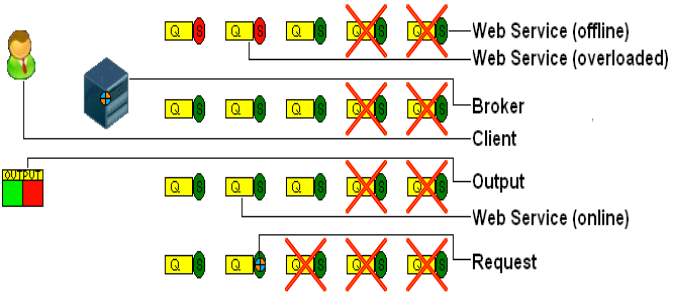

Fig. 2. Overview of the simulation model

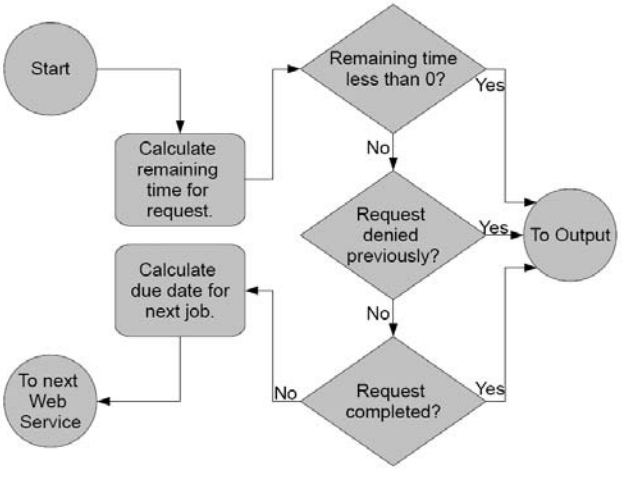

Fig. 3. Flowchart of the broker component in case of admission control

service component. If so, the request is also sent to the output component. If neither the latency limit has been reached nor the request has been denied service previously, the next web service needed to complete the request is determined. The web service calculates the due time for the next job, and then sends the job to the determined web service. For this calculation the total remaining time for the request is divided over all remaining jobs in proportion to their service requirements. When all jobs in the request are served, the request is sent to the output component as well.

Two simulation cases were designed to be used to compare the proposed admission control algorithms:

Case 1: The web services are placed in a specific order i.e. if web service X is before web service Y in one request type, it will be in every request type (in which both web services are present).

Case 2: There is no specific order of web services, but almost all request types make use of two specific web services.

Both cases are identified by

- the (order of) web services which need to be used by each request type.
- the distribution of requests over the different request types.
- the (required) service rates of all web services.

Note that the arrival rate λ is not part of the case characteristics, nor is the maximum allowed latency, L_{max}. These are considered to be parameters within a given case.

There are two performance indicators for the given admission control algorithms that we observed in greater detail

- Number of successfully served requests
- Goodput, which is defined as the average number of successfully served requests per second.

All simulations are executed on a desktop computer with a dual Pentium IV 3.2GHz processor and 1GB RAM memory. Unfortunately, the simulation package eM-Plant7 is not capable of using both processors. A bootstrap period (used to estimate λ) of 15 minutes is chosen as well as a simulation time of 15 minutes. A total of 15 simulations per case have been run.

Simulation Case 1

In the first case a total of 11 web services W_1, W_2, …, W_{11} and 10 different request types r_1, r_2, …, r_{10} were used. Most requests start in W_1 or W_5 and finish in W_{10} or W_{11}. The characteristics of this case are as follows:

$$Y = \begin{pmatrix} r_1 \\ r_2 \\ r_3 \\ r_4 \\ r_5 \\ r_6 \\ r_7 \\ r_8 \\ r_9 \\ r_{10} \end{pmatrix} = \begin{pmatrix} W_1 & W_2 & W_3 & W_7 & W_8 & W_9 & W_{11} \\ W_1 & W_2 & W_3 & W_8 & W_9 & W_{10} & \\ W_1 & W_2 & W_7 & W_8 & W_9 & W_{10} & \\ W_1 & W_2 & W_8 & W_9 & W_{10} & & \\ W_1 & W_4 & W_8 & W_9 & W_{10} & & \\ W_1 & W_4 & W_8 & W_9 & W_{11} & & \\ W_5 & W_{10} & & & & & \\ W_1 & W_6 & W_8 & W_9 & W_{11} & & \\ W_2 & & & & & & \\ W_4 & & & & & & \end{pmatrix}, p = \begin{pmatrix} 0.05 \\ 0.20 \\ 0.05 \\ 0.05 \\ 0.05 \\ 0.10 \\ 0.40 \\ 0.05 \\ 0.03 \\ 0.02 \end{pmatrix}, \mu = \begin{pmatrix} 5 \\ 10/3 \\ 5 \\ 5 \\ 10/3 \\ 5 \\ 5 \\ 10 \\ 10 \\ 10/3 \\ 10 \end{pmatrix}$$

In this notation Y is a matrix which shows the (order of) web services which need to be used by each request. The vector p denotes the distribution of requests over the different types and vector μ denotes the (required) service rates of all web services. Using test runs, the system (with L_{max}=8s) is found to get in overload around $\lambda=3s^{-1}$. Therefore arrival rates around $\lambda=3s^{-1}$ were investigated as well as other extreme values. Without WAC, the simulation runtime rapidly increases as λ increases. For $\lambda=1s^{-1}$ the runtime (without WAC) is about half a minute. For $\lambda=10s^{-1}$ the runtime has increased to about 45 minutes. To keep simulation run times acceptable, the extreme

arrival rates are not investigated for the situation without admission control. It is expected that the fraction of successfully served request and the goodput both have value 0 in these situations. Total simulation time of this case was approximately 8 hours. Simulation results are summarized in Fig. 4, including 99.7% individual confidence intervals. Notice that the scale of the horizontal axis changes after $\lambda=10s^{-1}$.

It can be seen that both admission control rules have a positive effect on goodput. Both admission control schemes seem to perform equally well. Only at extreme arrival rates the difference with the theoretical maximum increases. Goodput drops when admission control is not used. However, when admission control is not used, there is a slight increase in goodput between $\lambda=5s^{-1}$ and $\lambda= 9s^{-1}$. Especially at $\lambda=9s^{-1}$ the percentage of successful requests is much larger than expected. Given the (very small) confidence intervals it seems unlikely that this is due to the stochastic nature of the experiment results. This phenomenon will be called the *arrival paradox* and is explained by the following example:

Consider three web services, W_1, W_2 and W_3 (see Fig. 5) each with service rate 5. Requests go from W_1 or W_2 to W_3. If both W_1 and W_2 are not overloaded, the goodput

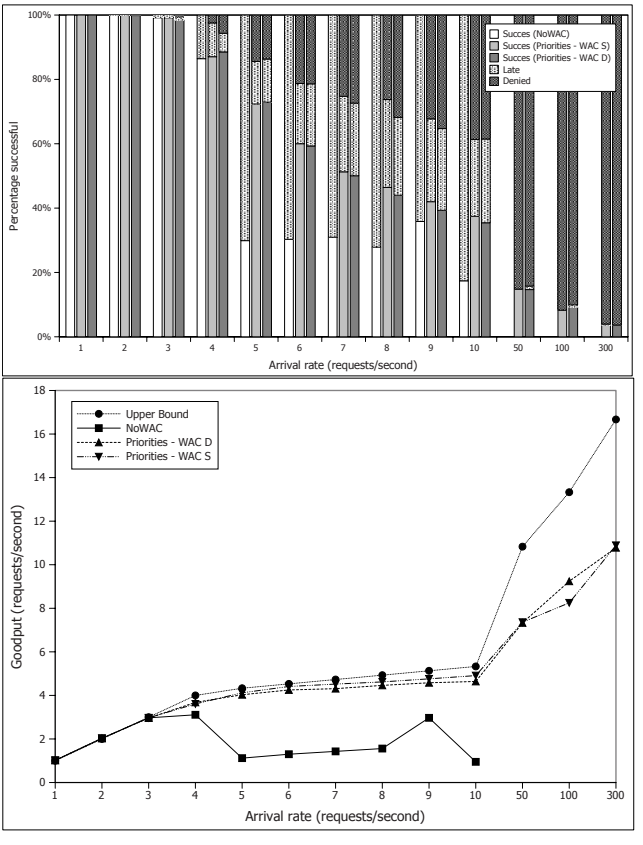

Fig. 4. Simulation results for Case 1

from these web services equals the arrival rate of these web services. Therefore the arrival rate at W_3 equals the sum of the arrival rates at W_1 and W_2 and hence W_3 is in overload and its goodput drops to zero. When the arrival rates are doubled, one of the web services W_1 and W_2 may get overloaded. Because admission control is not used, sojourn times will explode and requests will exceed their maximum allowed latencies. Recall that late requests are preempted at the broker. Therefore the arrival rate at web service W_3 decreases due to the higher overall arrival rate and W_3 no longer is in overload, hence its goodput increases.

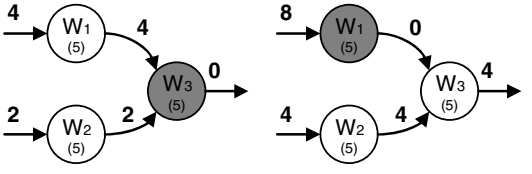

Fig. 5. Example of the arrival paradox, where web services in grey indicate overload

Simulation Case 2

In this case there are 10 request types and 9 web services. Most requests will visit W_5 and/or W_6, but these web services are not on a specific location in the chain, nor is there any other general sequence in which web services are called. The characteristics of Case 2 are as follows (using the same notation as in Case 1).

$$Y = \begin{pmatrix} r_1 \\ r_2 \\ r_3 \\ r_4 \\ r_5 \\ r_6 \\ r_7 \\ r_8 \\ r_9 \\ r_{10} \end{pmatrix} = \begin{pmatrix} W_8 & W_6 & W_2 & & & & & \\ W_5 & W_2 & W_3 & W_7 & W_6 & W_1 & & \\ W_4 & W_3 & W_7 & W_2 & W_6 & & & \\ W_8 & W_7 & W_5 & W_5 & W_9 & W_1 & & \\ W_7 & W_8 & W_2 & W_5 & W_9 & W_1 & W_6 & \\ W_7 & W_4 & W_6 & W_3 & W_5 & & & \\ W_8 & W_9 & W_1 & W_5 & & & & \\ W_5 & W_8 & W_3 & W_9 & & & & \\ W_6 & W_5 & W_4 & & & & & \\ W_1 & W_9 & W_8 & W_{21} & & & & \end{pmatrix}, p = \begin{pmatrix} 0.15 \\ 0.1 \\ 0.05 \\ 0.1 \\ 0.2 \\ 0.05 \\ 0.05 \\ 0.1 \\ 0.15 \\ 0.05 \end{pmatrix}, \mu = \begin{pmatrix} 5 \\ 5 \\ 4 \\ 10 \\ 4 \\ 4 \\ 10 \\ 5 \\ 5 \end{pmatrix}$$

In Case 1 it could be argued that some web services would never get in overload. For Case 2 this cannot be argued. Requests start in web services W_1, W_4, W_5, W_6, W_7 or W_8, thus these web services will get in overload if the arrival rate is high enough. For the other web services the line of reasoning used in Case 1 cannot be followed. This is because Case 2 lacks the structure like Case 1 has. Therefore it seems that each web service may get in overload. Total simulation time of this case was approximately 11 hours. Simulation results are summarized in Fig. 6. Just as in the previous case, the differences between the admission control algorithms seem almost negligible. The

only (relevant) difference occurs in terms of goodput for high arrival rates. For low arrival rates ($\lambda<5s^{-1}$) the *D* rule results in a slightly worse situation than if admission control is not used. In all other cases the admission control rules both behave better than when admission control is not used.

The difference between the theoretical maximum for the goodput and the observed goodput is larger compared to case 1, even for small values of λ. In case 1 the goodput kept increasing, even at high arrival rates. In this case however, the goodput decreases after $\lambda=12s^{-1}$.

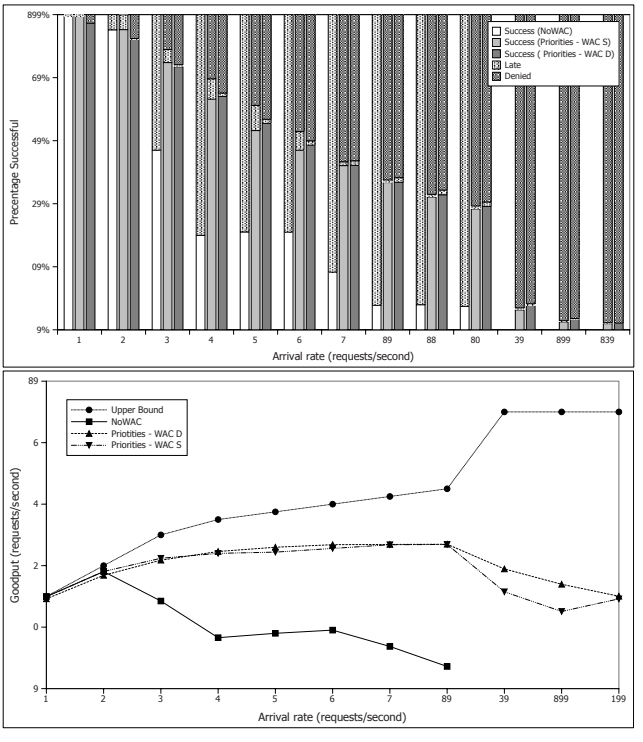

Fig. 6. Simulation results for Case 2

6 Experimental Validation

Besides theoretical analysis and simulation of admission control, an empirical experiment is set up to validate the simulations. Concrete web services were built and the results are compared to the simulation results. For this purpose of comparison it does not matter what function the web services perform. In addition, for setting up the tests it is convenient if the CPU demand of executing a web service can be controlled. Therefore, we implemented web services that calculate a specific Fibonacci number (each service has its own number to calculate) according to a CPU consuming

algorithm. By choosing the Fibonacci number the CPU consumption of this web service can be influenced. During the experiments two scenarios were evaluated: One where admission control rule *D* is enabled (WAC *D*); the other where admission control is disabled (NOWAC). To obtain the results from the web service the software package JMeter [12] was used. A global overview of the experimental setup is given in Fig. 7.

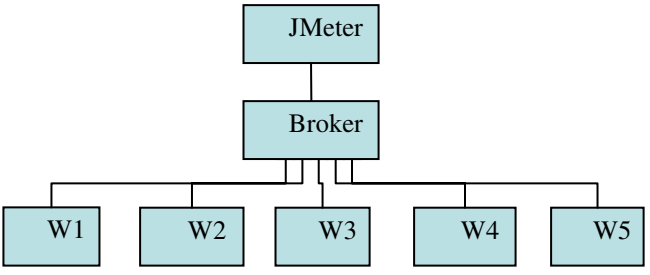

Fig. 7. System setup for empirical validation of admission control

The orchestrating broker (see Fig. 3) and the individual web services (W_1 thru W_5) are implemented following the design and implementation of the corresponding components in the simulations. All software was written in Java and executed on Tomcat [13] extended with Axis2 [14] for web service functionality. The case used in these experiments resembles the first case, where the web services are placed in a specific order. The characteristics of the web services are as follows (using the same notation as in Case 1):

$$Y = \begin{pmatrix} r_1 \\ r_2 \\ r_3 \\ r_4 \\ r_5 \\ r_6 \\ r_7 \\ r_8 \\ r_9 \\ r_{10} \end{pmatrix} = \begin{pmatrix} W_1 & & & & \\ W_4 & & & & \\ W_1 & W_2 & W_3 & W_4 & W_5 \\ W_1 & W_2 & W_4 & W_5 & \\ W_1 & W_2 & & & \\ W_1 & W_4 & W_5 & & \\ W_1 & W_3 & W_5 & & \\ W_1 & W_3 & W_4 & & \\ W_2 & W_5 & & & \\ W_3 & W_4 & & & \end{pmatrix}, p = \begin{pmatrix} 0.1 \\ 0.1 \\ 0.1 \\ 0.1 \\ 0.1 \\ 0.1 \\ 0.1 \\ 0.1 \\ 0.1 \\ 0.1 \end{pmatrix}$$

Note that no values for the service rate of each web service are given. All web services were configured to calculate the same Fibonacci number. Both the JMeter and the Broker run on the system equipped with 2GB RAM and single Pentium IV processors clocked at 3.2GHz. The web services W_1,…, W_5 run on systems equipped with 0.5GB, 1GB, 1GB, 0.5GB, 0,5GB and with Pentium IV processors at 1GHz, 2.4GHz,

2.4Ghz, 1GHz, 1GHz respectively. JMeter was configured to generate the requests r_1, r_2, …, r_{10} based on the probabilities p_1, p_2, …, p_{10}. In each run of JMeter a fixed number of threads (between 1 and 200) were active. Each run used a warm-up time of 15 minutes, and a test time of 15 minutes; the latter has been used to gather the results shown here.

In any composite web service the orchestrating broker is a suspect to become a performance bottleneck and should therefore be kept light. In our case the admission control rules are executed by the web services, and the broker is only responsible for service orchestration and tracking total latency of a composite request. In our experimental validation the orchestration is implemented in such a way, that performing admission control does not add a bottleneck to the composite web service. If, the broker would become the bottleneck in the system due to its orchestration function, then it would be possible to distribute the work by using more brokers. This is possible since the admission control rules are implemented in the web services. An overview of the experimental results is given in Fig. 8.

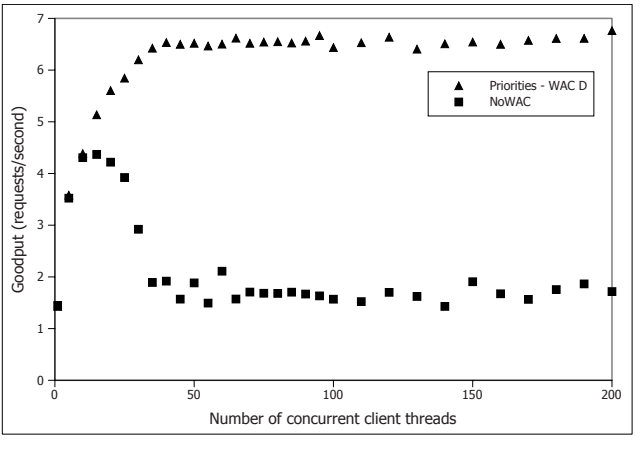

Fig. 8. Results of the empirical tests

The empirical and simulation results correlate well. Using WAC the overall goodput was noticeably higher than the NOWAC scenario. The NOWAC scenario reaches a maximum goodput when there are a little bit more than 4 requests per second at 15 concurrent threads. The WAC scenario seems to level between 6 to 7 requests per second at 50 concurrent threads.

7 Concluding Remarks

In this paper two different overload control algorithms for composite web services in service oriented architectures, were derived. These algorithms, *S* and *D*, were derived based on a *M/M/1/c* Processor Sharing Queue. In addition, a simulation model was

constructed and used to conduct simulations with these two rules and a benchmark (in which no admission control rule is used). Moreover, an experimental setup was constructed to conduct an empirical evaluation of rule *D* and the benchmark.

Based on simulation results, we conclude that in most situations both admission control rules *S* and *D* resulted in a higher objective value (measured in goodput) than the benchmark. While the difference is small, rule *S* does perform better than rule *D*. However, it can be observed that the results are dependent on the case, the structure and interaction patterns of the used web service components. The experimental evaluation of rule *D* gives similar results to the simulations performed for this rule.

To achieve further improvements, the empirical experiments should be scaled up to evaluate a broader range of different and larger service oriented infrastructures. Such experiments would be primarily focused on obtaining the most optimum goodput as well as incorporating business objectives in the admission control rules.

Another area of research is to extend the proposed admission control mechanisms in more complex environments, e.g. when the sequence of composite services is not known in advance, or when there is more variation in the resource requirements of each composite service.

Ackowledgement

Part of this work has been carried out in the context of the IOP GenCom project Service Optimization and Quality (SeQual), which is supported by the Dutch Ministry of Economic Affairs via its agency SenterNovem.

References

1. Gijsen, B.M.M., Meulenhoff, P.J., Blom, M.A., van der Mei, R.D., van der Waaij, B.D.: Web admission control: Improving performance of web-based services. In: Proceedings of Computer Measurements Group, International Conference, Las Vegas, USA (2004)
2. Xu, Z., Bochmann, G.V.: A Probabilistic Approach for Admission Control to Web Servers. In: Proceedings of Intern. Symp. on Performance Evaluation of Computer and Telecommunication Systems, SPECTS 2004, San Jose, California, USA, July 2004, pp. 787–794 (2004) ISBN 1-56555-284-9
3. Elnikety, S., Nahum, E., Tracey, J., Zwaenepoel, W.: A Method for Transparent Admission Control and Request Scheduling in E-Commerce Web Sites. In: Proceedings of the 13th international conference on World Wide Web, New York, USA, pp. 276–286 (2004) ISBN:1-58113-844-X
4. Urgaonkar, B., Shenoy, P.: Cataclysm: Scalable Overload Policing for Internet Applications. Journal of Network and Computer Applications (JNCA) 31, 891–920 (2008)
5. Xi, B.: Quality of service (QoS) for web-based applications. Technical report, TNO-ICT and Eindhoven University of Technology (2007)
6. Bouch, A., Kuchinsky, A., Bhatti, N.: Quality is in the eye of the beholder: Meeting user's requirements for internet quality of service. In: Proceedings of CHI 2000 Conference on Human Factors in Computing Systems (2000)

7. Abdelzaher, T., Bhatti, N.: Web server QoS management by adaptive content delivery. In: Proceedings of the International Workshop on Quality of Service, London, UK (June 1999)
8. Dyachuk, D., Deters, R.: Scheduling of Composite Web Services. In: Meersman, R., Tari, Z., Herrero, P. (eds.) OTM 2006 Workshops. LNCS, vol. 4277, pp. 19–20. Springer, Heidelberg (2006)
9. Dyachuk, D., Deters, R.: Improving Performance of Composite Web Services. In: Proceedings of IEEE International Conference on Service-Oriented Computing and Applications, June 2007, pp. 147–154 (2007) ISBN 0-7695-2861-9
10. Iwasa, K., Durand, J., Rutt, T., Peel, M., Kunisetty, S., Bunting, D.: Web Services Reliable Messaging TC, WS-Reliability 1.1. (2004), http://docs.oasis-open.org/wsrm/ws-reliability/v1.1/
11. Tecnomatix, eM-Plant 7.0 Manual. Tecnomatix GmbH (2004)
12. Apache JMeter, http://jakarta.apache.org/jmeter
13. Apache Tomcat, http://tomcat.apache.org
14. Apache Axis2, http://ws.apache.org/axis2/