Performance Evaluation of Dynamic Web Service Selection Strategies in Service Oriented Architecture

Miroslav Živković [*], Hans van den Berg, Hendrik B. Meeuwissen, Bart M. M. Gijsen
TNO
Delft, The Netherlands
miroslav.zivkovic@tno.nl
[*] *also with*
Design and Analysis of Communication Systems
University of Twente
Enschede, The Netherlands
m.zivkovic@utwente.nl

***Abstract*—In this paper, we explore the performance potential of dynamic (runtime) web service selection within the scope of Service Oriented Architecture (SOA). The web service selection is executed by a service orchestrator (request dispatcher) which is responsible to deliver composite web services at desired quality levels for the orchestrator's clients. We investigate service response times for the case where SOA state–of–the–art static web service composition is used and for the case where dynamic web service selection is applied. Modelling request scheduling at individual web services as Processor Sharing queueing systems, simulation results are presented for different runtime selection strategies in scenarios ranging from the "ideal" situation (up–to–date state information, no background traffic) to more realistic scenarios in which state information is stale and/or background traffic is present. In particular, we show the effectiveness of a selection strategy based upon the "synthesis" of Join the Shortest Queue and Round Robin strategies. For some specific scenarios we derive and validate insightful approximate formulae for the resulting response times. Our investigations quantify the performance gains that can be achieved by dynamic service selection compared to static (a–priori) service selection currently used.**

***Keywords*-Service Oriented Architecture, Join the Shortest Queue, Processor Sharing, Response Time, Background Traffic, Stale Information.**

I. Introduction

The composition of web services within a SOA environment could be static or dynamic. With static composition the concrete services are determined and integrated into the specification at design time. With dynamic composition on the other hand, at design–time there is only a specification of the required abstract services given. The concrete services are then integrated at runtime.

For both static and dynamic composition, the choice of concrete services for a particular abstract service may be based on non–functional parameters. Examples of such parameters are availability, throughput, response time, security and cost. References [4], [21], [23] and [24] discuss the problem of static QoS–aware service composition in detail. As an extension, [4] considers sub–optimal algorithms to enable fast replacement of underperforming services.

Existing SOA–based solutions for web services do not support dynamic web service selection ([12], [13]). Dynamic selection provides flexibility and therefore has advantages with respect to availability and reliability compared to a static approach. Another possible advantage of dynamic service selection is performance improvement, i.e., achieving a decrease in the requests' response times by exploiting statistical variations in the loads at the various concrete services. This paper aims at investigating the potential performance gain of dynamic, runtime web service selection for service composition within the scope of SOA, evaluating different selection strategies.

As a starting point, we assume that a *set* of concrete services is a-priori selected per abstract service, which is in contrast with a-priori selecting a *single* concrete service per abstract service in the case of static composition. Therefore, the discovery mechanisms ([19]) and their performance are outside the scope of this paper. As an example, in Figure 1, the choice of a particular concrete service (from the set of selected concrete services) is made by the dispatcher at runtime on a per–client request basis for a composite web service consisting of N services that are invoked consecutively. In each consecutive step $i,\ i = 1, 2, \dots, N$, exactly 1 out of K_i concrete services is invoked by the dispatcher, where K_i represents the number of choices in step i. In this example, the total response time $\mathrm{RT}_{\mathrm{total}}$ is the sum of the individual response times RT_i. Within SOA the dispatcher is part of the orchestrator, which typically runs in the domain of the composite service provider.

The main research question addressed in this paper is what is the performance potential of dynamic web service selection versus static selection? As outlined above, dynamic web service selection is made from K_i pre-selected concrete services for each step i (note that this pre-selection is beyond the scope of this paper). We focus on the achievable performance gain of dynamic selection in case of a single abstract service (i.e., $N = 1$ in Figure 1). This analysis will give us also an indication of the potential gain for a composite service that consists of more abstract services ($N > 1$). Besides, analysis for the case $N = 1$ is more feasible than for the general case $N > 1$. Specific research questions addressed in the paper are: what is the influence of the number of pre-selected concrete services, which stateless and statefull dispatching algorithms perform well, which gains are achievable, and what is the impact of practical conditions such as background traffic and delayed state information on these gains?

Notice that existing dispatching strategies, e.g., Round-Robin (RR), Bernoulli, Join the Shorted Queue (JSQ), are included in our analysis. In the literature, these dispatching strategies are mostly investigated in the context of systems with First Come First Served (FCFS) queues, but hardly for systems with Processor Sharing (PS) queues, as considered in this paper. Note that in the current context the PS services model is more realistic than FCFS, see e.g., [10]. In addition, we consider background traffic and delayed state information.

We summarize the main contributions of this paper as follows:

1) Quantification of the achievable gain versus the number K of pre-selected concrete services by a fair comparison with respect to the base case of static web service selection.
2) Quantification of the achievable gain in terms of response times when background traffic is present at the pre-selected concrete services for different dispatching strategies.
3) Quantification of the achievable gain in terms of response times in case of delayed state information.

In order to investigate the above–mentioned potential performance gains several assumptions have been made, which allow us to quantify the "ideal" system performance. In that sense our analysis should be understood as a "baseline" analysis. Besides, the assumptions are made with the goal to represent our findings in an unequivocal way. Our analysis makes a significant step towards analysis of models that take into account more practical conditions regarding the observed system.

The remainder of this paper is organized as follows. First, in Section II, we describe the performance model and explain the underlying assumptions that capture the essential system characteristics needed for our study. Next, in Section III, we discuss literature related to our work. In Section IV, our simulation results and results obtained by analytical modelling are presented, and we discuss and explain the observations. Finally, in Section V, we draw conclusions and give suggestions for further research.

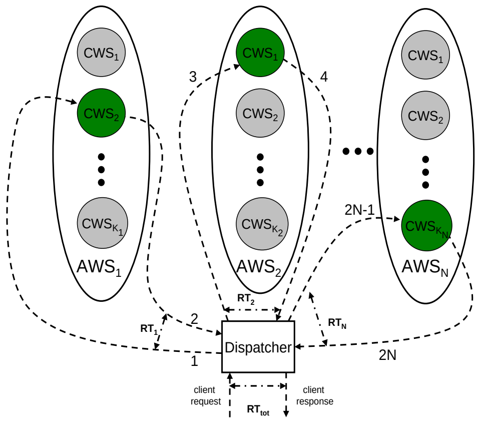

Figure 1. Illustration of dynamic SOA–based web service composition. A request by the client is served by one of the K_1 implementations of abstract service AWS$_1$, then by one of the K_2 implementations of abstract service AWS$_2$, and so on, until client request is completed and response is sent back. Note that every client request may be served by a different chain of concrete web services (CWS).

II. Model description

We consider one abstract service with K concrete service implementations as given in Figure 2. There are two classes of incoming service requests:

- Foreground service requests are received by the dispatcher, which decides at runtime to which of the K service instances a particular request is assigned for getting the required service. The foreground requests arrive to the dispatcher according to a Poisson process [15] with rate Λ and have exponentially distributed service requirements with mean $\frac{1}{\mu}$. The rate at which foreground traffic requests are offered (by the dispatcher) to service i is denoted by $\lambda_{\mathrm{FTi}},\ i = 1, 2, \ldots, K$.
- Background service requests arrive at service instance i according to a Poisson process with rate $\lambda_i,\ i = 1, 2, \ldots, K$, respectively. The background service requests are exponentialy distributed with mean $\frac{1}{\mu_i},\ i = 1, \ldots, K$. The background traffic arrival processes are independent from each other and are also independent from the foreground arrival process.

Request scheduling at each service instance is modelled by a processor sharing infinite–capacity single server queue. Served requests leave the system.

Obviously, the achievable performance gain of dynamic service selection compared to a pure static approach depends heavily on the nature of the workload fluctuations at the different service instances. It is clear that in the case of slowly and independently varying loads (e.g., due to fluctuations in the service demand over the day) high performance gain can relatively easy be achieved. However, in such cases the runtime character of dynamic service selection may be

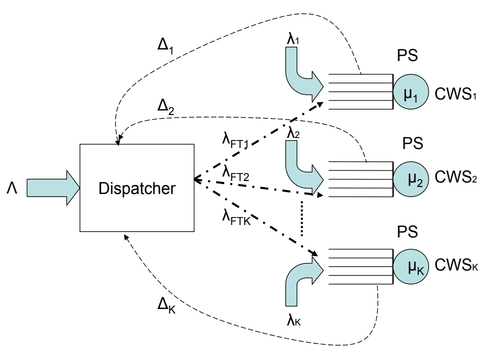

Figure 2. Performance model for the case of a single abstract service with K different implementations (concrete services).

an "overkill" and the performance gain could largely also be obtained by less flexible service selection approaches. Therefore, the present paper focuses on exploiting workload fluctuations at the services instances that occur at relatively small time scales mainly caused by the random behaviour of individuals in a large population of potential users. In that perspective, and to keep the parameter space manageable, we will assume that the model is symmetric, i.e., $\lambda_i = \lambda_{\rm BT},\ \mu_i = \mu,\ i = 1, 2, \dots, K$. The utilization per service $i, i = 1, 2, \dots, K$ is then defined as $\rho_{\rm tot} = \frac{\lambda_{\rm tot}}{\mu}$, where $\lambda_{\rm tot}$ is the rate of the aggregated (foreground and background) traffic, i.e., $\lambda_{\rm tot} = \lambda_{\rm FTi} + \lambda_i = \lambda + \lambda_{\rm BT}$. The stability condition per service requires that the expected number of requests per service remains finite, i.e., $\rho_{\rm tot} < 1$.

Ignoring possible delays due to the queueing and processing at the dispatcher, as well as network delay, arriving service requests are instantaneously forwarded to one of the K service instances according to the dispatching strategy. Various strategies can be used for selection of one of the K service instances upon arrival of a new request. The dispatching strategies could be roughly divided into two categories, namely *stateless* and *statefull*. Decision making is independent of the system state information for the stateless strategies. Conversely, decision making takes into account (stale) system state information for statefull strategies. The delay in obtaining the information per service instance is represented by parameter $\Delta_i,\ i = 1, 2, \dots, K$. In this paper we have adopted the case when system state information (queue length, response time, etc.) is collected periodically with the same period $\Delta > 0$. This information gathering may require sending separate requests ("probes") by the dispatcher to all of the K web services, and collecting information in such a manner introduces an overhead to the system, which influences system performance. This issue as well as using other ways to collect system state information are beyond the scope of this paper. The update period Δ has been related to the intensity of the aggregated traffic as $\Delta = D \cdot \frac{1}{\lambda_{tot}}$. where D is integer. All dispatching decisions between time instances $t_i = i \cdot \Delta$ and $t_{i+1} = (i+1) \cdot \Delta,\ i = 0, 1, 2, \dots$ are made based on the system state information obtained at t_i. The dispatching strategies considered for this paper are Bernoulli (BL), Round-Robin (RR), Join the Shortest Queue (JSQ) and a combination of the latter two, JSQ–RR. Bernoulli and RR are typical examples of stateless strategies while JSQ and JSQ–RR are examples of statefull dispatching strategies. In case of the Bernoulli strategy, the requests are randomly distributed over the queues, i.e., a newly arriving request is assigned to queue $i,\ i = 1, \dots, K$, with probability $\frac{1}{K}$. This case is used as representation of the performance for the static SOA service selection. For RR, the k–th request is assigned to queue $(k \bmod K) + 1$. In JSQ, the request is assigned to the queue with the smallest number of requests waiting to be served. Ties are resolved by randomly assigning the request to one of the shortest queues. In case when system state information delay is present in the system, an additional statefull dispatching strategy could be defined, namely JSQ–RR. For JSQ–RR, once the actual system information is obtained, the queues are sorted in non–descending order by the queue lengths. Any request coming to the dispatcher between two state updates is then assigned following the RR scheme, i.e., the first request is assigned to the queue with smallest queue length, the second request is assigned to the queue with smallest queue length from the remaining queues, etc.

III. Related Work

In this section, we give a short overview of papers related to different aspects (e.g., web service selection, composition, performance) of the runtime web service selection in SOA. However, each of these papers treats only a (different) subset of issues relating to runtime web service selection. The analysis of potential performance improvements of different dispatching strategies, based on PS modelling of the request scheduling at web service(s) within SOA, which includes impact of stale system state information and/or background traffic is, to our best knowledge, non–existent.

A. Web service selection and composition

In [13], an overview of common misconceptions about SOA is given. Among others, the issue of dynamic selection of web services is identified, and it is indicated that current SOA solutions lack advanced automatic discovery and composition of web services at runtime.

A lot of attention within SOA community has been dedicated to static QoS–aware composition problem, see e.g., [4], [21], [23] and references therein. The problem of static QoS–aware composition is known to be NP–hard, see [24], where two service selection approaches for constructing composite services have been proposed: local optimization and global planning.

In paper [20], several architectures and their respective models that assist in dynamic invocation of web services are discussed. These models allow the *client* to dynamically select the current best web service, based on certain non–functional criteria (availability, reliability, and estimated response time). These clients gather runtime web service information, evaluate the performance of the previously used web services, and may share this information with other clients. The selection decision is let to the clients, which contain intelligent agents and therefore the complexity of the clients increases. The inherent problem is that different clients may decide to use the same web service, which would eventually result in worsened performance e.g., due to the overload of the targeted web service.

The framework proposed in [14] enables quality–driven web service selection, based upon evaluation of the QoS of a vast number of web services. The fair computation and enforcing of QoS of web services takes place when making the web service selection. In order to provide fair computation the feedback from clients is gathered.

B. Performance of dispatching strategies

Performance of dispatching strategies in multi–server systems has been a topic that received a lot of attention within the queueing theory research community. Specifically, a lot of work has been done for systems with First Come First Served (FCFS) scheduling at the queues, e.g., in [3] and [6]. In the most of the papers written for JSQ/FCFS, explicit results for response times are given only for the case $K = 2$ servers, an exponential job size distribution and the mean response time metric, [9]. The performance of the JSQ/FCFS strategy for $K > 2$ servers has been analysed in [17] where the approximation of the mean response time for K homogeneous servers is given. In [18], an extension to this approximation has been given, however, the approximation is less accurate as the requests' size variability increases.

Opposite to the JSQ/FCFS systems, JSQ/PS systems have not received so much attention. The notable exceptions are [2], and, more recently, [10] where approximate analysis of JSQ in the PS server farm model for general job size distributions is presented. The queue length of each queue in the system is approximated by a one–dimensional Markov chain, and based on this approximation the distribution of the queue length at each queue is determined. In [1], the authors investigate optimal dispatching strategies for a multi–class multi–server PS systems with a Poisson input stream, heterogeneous service rates, and a server-dependent holding cost per unit time.

C. Performance of dispatching strategies with stale system information or background traffic

In [16], the problem of dispatching with stale system status information (server load) is analysed in case of FCFS. Servers' status information is periodically updated and three strategies are compared: random selection, selection of the server with the least load (based on the stale system information), and random selection of a small subset of servers and then selecting the least loaded of the chosen servers (based on up-to-date information about their loads). It is shown that the latter strategy mostly outperforms the other ones, even for a small randomly chosen subset of e.g., two servers, while the overhead (due to processing and information retrieval) remains limited. In [5], the authors present a strategy that routes the jobs to the server with expected shortest FCFS queue. The decisions are made based on stale information and elapsed time since the last state update. This strategy works well, but does not always minimise the average response time.

In [11], a dispatching policy based on splitting foreground traffic according to a predefined rule described by a certain parameter vector is analysed while background traffic is modelled as independent Poisson processes with different rates. Due to the assumptions made each of the N servers in isolation can be represented as a two–class M/G/1 PS queue. The approximation of the response times is deduced for the case of light foreground traffic and an optimal parameter vector is found.

IV. PERFORMANCE ANALYSIS

In this section, we present and discuss simulation results for the runtime service selection strategies described in Section II in order to investigate their performance potential. For some special scenarios we also present numerical results obtained from analytical modelling.

The simulations were performed using the simulation tool implemented in Java programming language, and using the Java library for stochastic simulation (SSJ) [8]. In order to make the simulations less sensitive to the startup transient, the number of foreground traffic arrivals per simulation has been set to at least $0.5 \cdot 10^6$. Besides, in order to improve the accuracy, we have trimmed simulation results for certain number of foreground traffic arrivals at the end of the arrival process.

We have considered four main categories of simulation scenarios:

- Baseline scenarios – these simulations were performed for the system without background traffic and with up-to-date system state information. The simulation results are given in subsection IV-A.
- Scenarios with stale system state information – these simulations were performed for the system without background traffic in which system state information is only periodically updated i.e., the dispatching process does not (always) use up-to-date information. The results are presented in subsection IV-B.
- Scenarios with background traffic – these simulations were performed for the system with up-to-date system state information and different intensities of background traffic. In addition to the simulations we derived

an analytical approach to study the performance for these scenarios. The results are presented in subsection IV-C.

- Scenarios with background traffic and stale system state information. The simulation results are presented in subsection IV-D.

A. Baseline scenarios

The goal of these simulation scenarios was to establish the performance results in case when there is no background traffic and system state information is up-to-date at the dispatcher.

In Figure 3, we show mean response times for different dispatching strategies (JSQ, RR, BL) as a function of the number of concrete services, K and for different values of utilization per service, ρ_{tot}. The utilization per service is kept constant in order to have a fair assessment of the impact of K; otherwise, an increase of K would simply be interpreted as capacity add-on to the system. Since JSQ–RR is identical to JSQ when up-to-date system state information is available at the dispatcher, the results for JSQ–RR are not shown. For $\rho_{\text{tot}} = 0.8$, the mean response time for JSQ strategy with $K = 4$ services is around 66% of mean response time for the same strategy when $K = 2$. Similarly, in case of JSQ with $K = 8$ and $K = 16$ services, response times are 49% and 40% of the response time for $K = 2$, respectively. In case when one of the K services becomes unavailable, the performance of the system (response time) does not deteriorate dramatically, as long as the utilization per queue remains (approximately) the same. The utilization per queue can be kept the same when, e.g., $K + 1$ services are pre–selected, of which given (fixed choice) K services are used for dispatching. The remaining ($K + 1$th) service is placed ”on hold” and when one of the chosen K services becomes unavailable, it is immediately replaced.

Figure 4 shows relative comparisons between JSQ and BL (with BL as the baseline) and JSQ and RR strategies (with RR as the baseline), respectively. Statefull strategy (JSQ) is superior to either of the stateless strategies (BL, RR), which confirms that more (and accurate) state information made available to the dispatcher leads to better decision making.

The potential performance improvements in the first case range from 28% to 46% for $K = 2$, depending upon the utilization per queue, ρ_{tot} and are in the range from 49% to 86% for $K = 16$. What is also of interest is when do the gradient of the performance improvement is highest, taking into account the increase of the number of services. From Figure 4 we see that this is the case when the number of services increases from 2 to 4. The gradient of the gains is (significantly) smaller when the number of services increases from 4 to 8 or 8 to 16, respectively. Based on these simulation results, we can draw the following conclusions:

- Large performance improvements compared to the static service selection are possible with relatively small values of K.
- The largest relative improvements of response time for number of services, $K > 1$, are obtained when we increase the number of services that are used from 2 to 4.

B. Scenarios with stale system state information

The goal of these simulations was to analyse the impact of the stale system state information to the response time of the system for different dispatching strategies. No background traffic has been assumed. The simulations were performed only for the statefull strategies, i.e., JSQ (see Figure 5) and JSQ–RR, see Figure 6. The response times for stateless strategies, RR and BL, are not affected by (stale) system state information, and are shown for comparison as well.

From Figure 5 we see that, for relatively small values of parameter D that determines the update interval, JSQ still performs better than RR or BL dispatching strategy. However, as expected, when parameter D increases performance of JSQ deteriorates e.g., for $D = 20$ response time for JSQ is worse than either RR or BL for almost complete range of parameter ρ_{tot}. In case when $D \to \infty$, the system state information is obtained just once, and then all arrivals are “blindly” assigned to the queue which had the smallest queue length when system state information was obtained. In that case, the service composition in this case is static, and the system model reduces to a M/M/1/PS queue with arrival rate Λ and mean service time μ.

We have also investigated the behaviour of the JSQ–RR strategy for systems with stale state information. Figure 6 shows that, as expected, JSQ–RR strategy is less sensitive to stale information than “blind” JSQ strategy. For example, when $D = 10$ and $\rho_{\text{tot}} = 0.7$, response times for Bernoulli, RR, JSQ and JSQ–RR are 2250 ms, 1515 ms, 3200 ms (!) and 1350 ms, respectively. For comparison, the response time for JSQ without stale information and the same ρ_{tot} is approximately 1020 ms.

Based on the simulations which results are presented at Figure 5 and Figure 6 we can draw the following conclusions:

- When $D \to 0$ JSQ–RR is identical to JSQ and when $D \to \infty$, JSQ–RR is identical to the common RR strategy.
- With respect to the response time, the JSQ–RR strategy is never worse than RR, regardless of the delay within the system. This makes JSQ–RR appealing strategy for systems with delay without background traffic.

C. Scenarios with background traffic

In the previous simulation scenarios we have assumed that the concrete services were used by the foreground traffic clients only. In what follows we look into the situation when background traffic is present as well, and the dispatcher has up-to-date system state information.

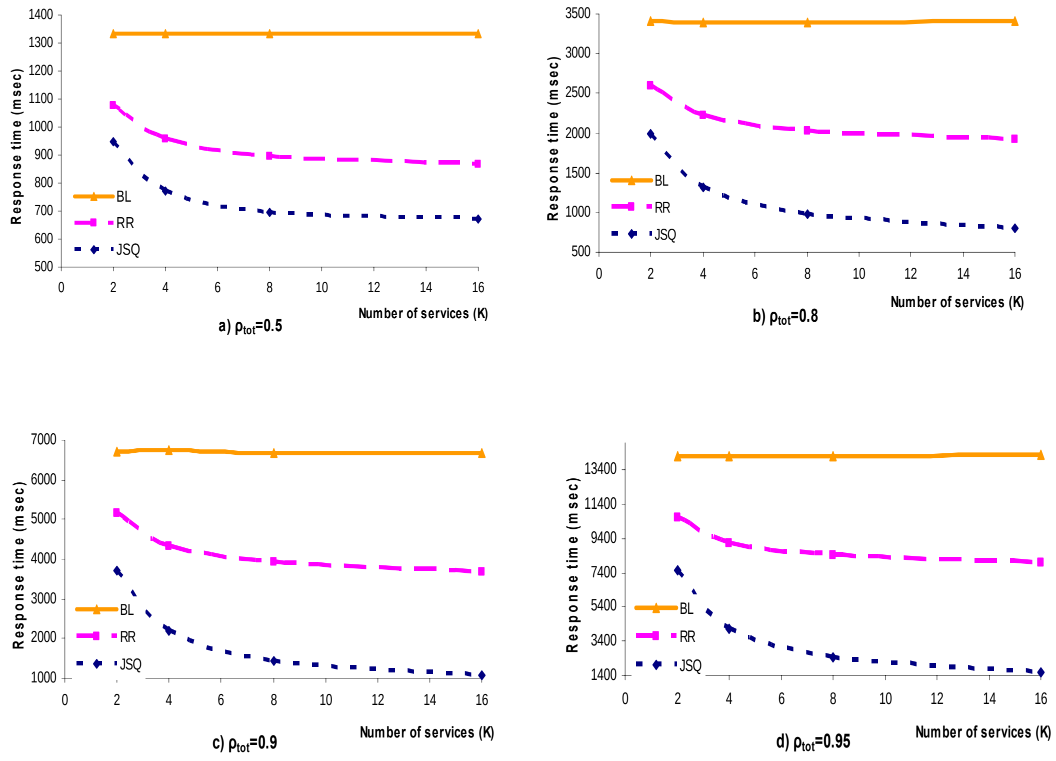

Figure 3. Comparison of mean response times for JSQ, RR and BL strategies for different number of services K and different values of ρ_{tot}. There is no system state information delay and only foreground traffic is present in the system.

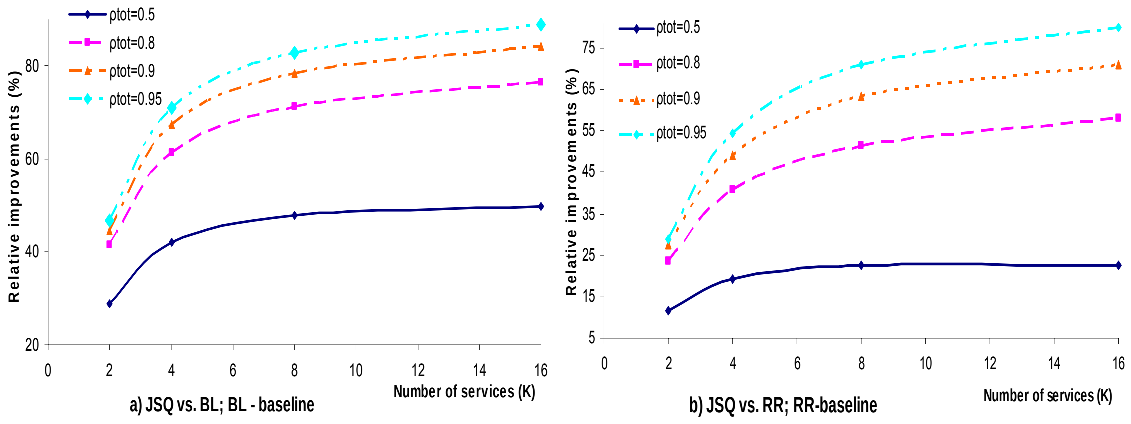

Figure 4. Relative comparison of mean response times between JSQ and BL (left) and JSQ and RR (right) strategies for different number of services K and different values of ρ_{tot}. The system state information is up-to-date and only foreground traffic is present in the system.

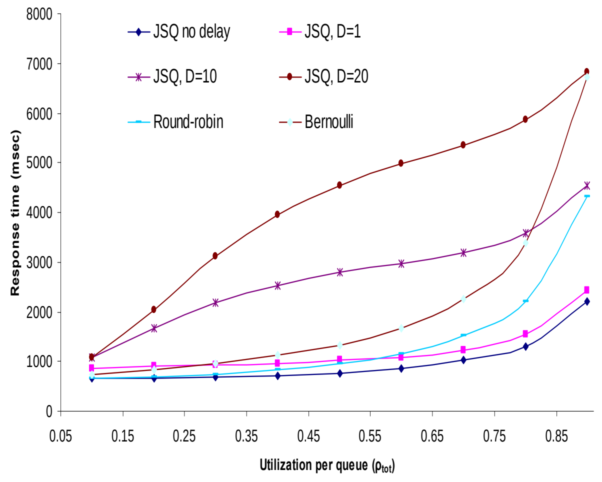

Figure 5. Comparison of mean response times for the following dispathcing strategies: BL, RR, JSQ with up-to-date system state information, and JSQ when system state information (queue lengths) is obtained with period $\Delta = D \cdot \frac{1}{\lambda_{\rm tot}}$, where $D \in \{1, 10, 20\}$. Background traffic is not present and the number of services $K = 4$.

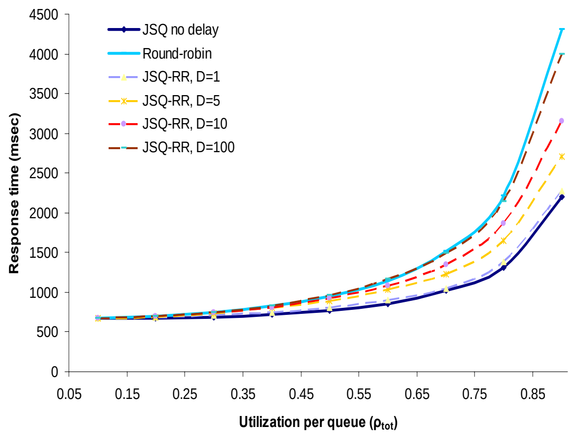

Figure 6. Comparison of mean response times for the following dispatching strategies: RR, JSQ with up-to-date system state information and JSQ–RR when system state information (queue lengths) is obtained with period $\Delta = D \cdot \frac{1}{\lambda_{\rm tot}}$, where $D \in \{1, 5, 10, 100\}$. Background traffic is not present and the number of services, $K = 4$.

Our simulations and analysis are directed to answering the question of the impact of the background traffic to the response times. The simulations results are shown in Figure 7 for $K = 4$ services and BL, RR, and JSQ strategies. Since the system state information is assumed to be instantaneously available, JSQ–RR is identical to JSQ, and therefore not shown. We have recorded the response times of the foreground requests only. For given utilization per queue $\rho_{\rm tot}$, and dispatching strategy, foreground traffic percentage of $\rho_{\rm tot}$ has been varied from as little as 10%

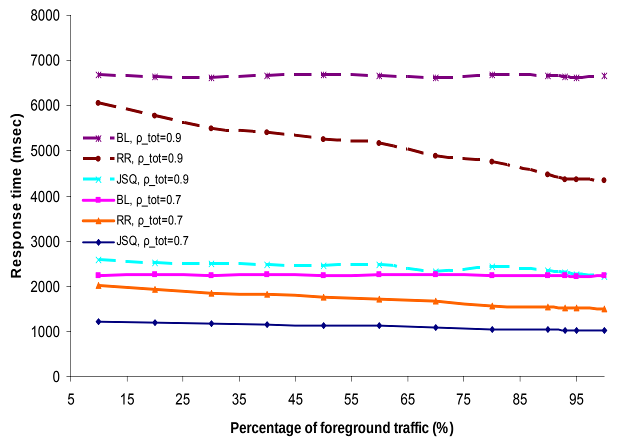

Figure 7. Response times for BL, RR and JSQ strategies for scenario with background traffic and no information delay. Utilization per queue $\rho_{\rm tot}$ is 0.7 or 0.9, and number of services $K = 4$.

(i.e., 90% background traffic) to 99% (i.e., 1% background traffic). Apart from the case of the Bernoulli dispatching strategy when response times are constant, as expected, from Figure 7 it follows:

- In case of the Round-Robin strategy response times decrease as the percentage of foreground traffic increases. It seems that response times dependency from the given percentage is linear.
- In case of the JSQ strategy response times show linear non-increasing dependency from the given percentage of foreground traffic. The decrease of the response time is limited by 15% for the considered cases. It seems that the JSQ strategy is not much sensitive to the background traffic.

The intuitive explanation for the decreasing nature of response times in case of RR and JSQ strategies may be given as the following – the response time in the case of these two strategies is biggest for the smallest percentage of foreground traffic, due to the fact that only foreground traffic is "intelligently" assigned to one of the queues.

Response time for JSQ with low foreground traffic load: Let us now consider the situation where the foreground traffic constitutes only a small percentage of the total traffic. We will analyse the mean response time of a tagged foreground traffic arrival. According to the JSQ policy this arrival will be dispatched to the queue (out of K queues) with the smallest length. Since the foreground traffic is negligible and the background traffic arrival processes are i.i.d., the random processes representing the queue lengths are also independent from each other and behave as the queue length of an M/M/1 PS queueing model with load ρ_{tot}. The queue length distribution for this model is geometric with parameter ρ_{tot}, see [22]. Hence, the probability that the queue selected for the tagged foreground job contains n

(background) jobs is given by:

$$\Pr\{n \text{ jobs in selected queue}\} = \left(1-\rho_{\text{tot}}^K\right)\left(\rho_{\text{tot}}^K\right)^n.$$

Once the tagged arrival is placed to a particular queue, that queue further behaves as an "ordinary" M/M/1 PS queue with utilization $\rho_{\text{tot}} = \frac{\lambda_{\text{BT}}}{\mu}$, as the foreground traffic is negligibly small.

Now, let us denote by $X_n(\tau)$ the random variable whose distribution is that of the "delay" experienced by the tagged arrival if it would have service requirement τ and arrives when there are n background jobs in the queue. The total time spent in the system for the tagged arrival (i.e., response time) is then $X_n(\tau)+\tau$.

From the detailed analysis of the M/M/1 PS queue in [7], it follows that (cf. Eq. (33) in [7]):

$$E\{X_n(\tau)\} = \frac{\rho_{\text{tot}}\tau}{1-\rho_{\text{tot}}} + [n(1-\rho_{\text{tot}})-\rho_{\text{tot}}]\cdot\frac{1-e^{-(1-\rho_{\text{tot}})\mu\tau}}{\mu(1-\rho_{\text{tot}})^2}.$$

Since the r.v. $X_n(\tau)$ is conditioned by n, the mean $E\{X(\tau)\}$ is given by the following equation

$$E\{X(\tau)\} = \sum_{n=0}^{\infty}\left(1-\rho_{\text{tot}}^K\right)\left(\rho_{\text{tot}}^K\right)^n\cdot E\{X_n(\tau)\},$$

which leads to

$$E\{X(\tau)\} = \frac{\rho_{\text{tot}}\tau}{1-\rho_{\text{tot}}} + \frac{\rho_{\text{tot}}^K-\rho_{\text{tot}}}{1-\rho_{\text{tot}}^K}\cdot\frac{1-e^{-(1-\rho_{\text{tot}})\mu\tau}}{\mu(1-\rho_{\text{tot}})^2}.$$

The overall mean response time for the tagged arrival is given by

$$\text{RT} = \frac{1}{\mu} + \int_0^\infty E\{X(\tau)\}d(1-e^{-\mu\tau}),$$

which finally gives

$$\text{RT} = \frac{1}{\mu} + \frac{\rho_{\text{tot}}}{\mu(1-\rho_{\text{tot}})} + \frac{\rho_{\text{tot}}^K-\rho_{\text{tot}}}{1-\rho_{\text{tot}}^K}\cdot\frac{1}{\mu(1-\rho_{\text{tot}})}\cdot\frac{1}{2-\rho_{\text{tot}}}.$$

This equation gives a surprisingly simple relationship between the response time for the foreground traffic, the number of services K, the utilization per queue ρ_{tot} and the mean of the foreground job sizes $\frac{1}{\mu}$.

These formulae have been deduced under the assumption that foreground traffic intensity is negligible compared to background traffic. Inspired by the numerical results in Figure 7 we investigated whether this response time formula could be used as an approximation for larger values of the percentage of the foreground traffic. A first comparison between our approximate formula and simulations, taking the simulations as the baseline, is given in Table I. The comparison indicates that:

- As expected, for a fixed number K of concrete services, the difference between our analytical results and simulation increases when the percentage of foreground traffic becomes larger. This is because our formula has been deduced under the assumption that there is only one foreground traffic arrival.
- Roughly speaking, the error of our approximate formula increases as the number of services K increases (and all other parameters remain the same).
- The relative difference between our formula and simulation increases when ρ_{tot} increases and all other parameters remain the same

TABLE I
RELATIVE COMPARISON BETWEEN THE RESPONSE TIMES OBTAINED BY SIMULATIONS AND RESPONSE TIMES CALCULATED USING THE FORMULA.

	$K=2$		$K=4$		$K=8$	
FG traffic (%) →	5	10	5	10	5	10
$\rho_{\text{tot}}=0.5$	0.1%	0.19%	0.5%	1.6%	1.8%	1.5%
$\rho_{\text{tot}}=0.7$	1.1%	1.7%	1%	1.3%	2.4%	8.2%
$\rho_{\text{tot}}=0.9$	1.7%	4.6%	5.2%	8.6%	5.0%	13.5%

D. *Scenarios with background traffic and stale system state information*

For these scenarios we have conducted simulations in order to investigate which factor has more impact to the response time: delayed system state information or background traffic.

The simulation results presented in Figure 8 for the JSQ–RR strategy, apply to the case when ρ_{tot} is fixed at 0.7 and the number of services $K=4$. Results are shown for four different values of the parameter D representing the system state information delay: $D\in\{1,2,5,10\}$. As for the case with up-to-date system state information ($D=0$) considered in the previous subsection, we see that the response time as function of the percentage of foreground traffic has a decreasing trend. Obviously, when background traffic diminishes, the response time approaches the values obtained for the scenarios without background traffic considered in subsection IV-B. However, all together, it is hard to determine from this figure which of the two factors has predominant influence on the response time.

In order to investigate whether delayed system state information or intensity of the background traffic has more impact to the system performance, we compare results from Figure 8 ($RT_{\text{BG}+\Delta}$) to results when only stale information is present (RT_Δ). The comparison is presented at Figure 9 and represents the ratio $r=\frac{RT_{\text{BG}+\Delta}}{RT_\Delta}$ for different values of the system state information delay parameter D. The ratio r is lower bounded by 1, and when $r\to 1$ delay has more influence on $RT_{\text{BG}+\Delta}$ than background traffic. The following conclusions can be made from Figure 9:

- The larger D, the more influence has the background traffic on $RT_{\text{BG}+\Delta}$. Suppose that the percentage of the foreground (background) traffic in the system is fixed. As D increases, the interval when state information is collected becomes larger. The larger the interval, the

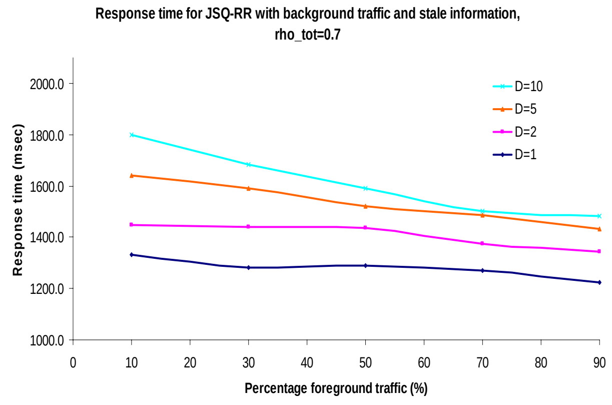

Figure 8. Response times for JSQ–RR strategy in case of the scenario with background traffic and stale information, with parameter $D \in \{1, 2, 5, 10\}$. Utilization per queue, ρ_{tot} is 0.7, the number of services is $K = 4$.

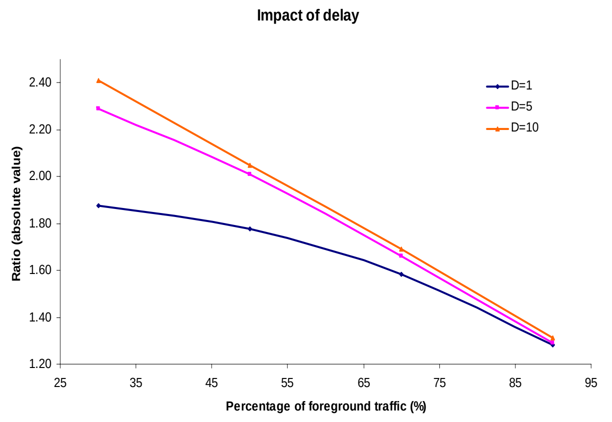

Figure 9. Ratio of response times between the system with background traffic and stale information and the system with stale information only. The dispatching strategy is JSQ–RR. Utilization per queue is $\rho_{\text{tot}} = 0.7$ and the number of services is $K = 4$.

more background traffic arrivals to a queue between two state information updates. The response time of the tagged foreground arrival will therefore be influenced by more background arrivals.

- For smaller values of parameter D, the relative change of ratio r is smaller. For example, when $D = 1$ the ratio changes from 1.88 (30% foreground traffic) to 1.28 (90% foreground traffic), compared to change from 2.41 to 1.31 when $D = 10$, respectively. This means that absolute influence of background traffic is smaller for smaller values of D. The smaller the period of the system state information update, less background traffic arrivals are probable within one such period.

V. Conclusion and Future Work

In this paper, we have investigated the performance potential of dynamic, run-time web service selection within SOA, under various assumptions regarding the available system state information and/or presence of background traffic. Using simulation and analytical modelling it has been shown that, compared to static (a–priori) service selection, considerable performance improvements are possible, even when the state information is stale and/or background traffic is present. These improvements result from exploiting workload fluctuations that occur at relatively small time scales mainly caused by the random behaviour of potential clients.

The main results of the paper could be summarized as the following:

1) Quantification of the achievable gain versus the number K of pre-selected concrete services by a fair comparison with respect to the base case of static web service selection. For relatively small numbers of K, e.g., $K = 4$ or $K = 8$, significant response time reductions are obtainable.
2) Quantification of the achievable gain in terms of response times when background traffic is present at the pre-selected concrete services for different dispatching strategies. We show that the response time performance of JSQ is quite robust with respect to the presence of background traffic. An insightful approximate formula for the response time under the JSQ dispatching strategy is derived for cases where the background traffic is dominant.
3) Quantification of the achievable gain in terms of response times in case of delayed state information. A stateless dispatching algorithm such as RR always improves upon the base case. Statefull dispatching algorithms such as JSQ should be carefully applied as can potentially perform worse than the base case when delay is present. However, a combination of RR and JSQ, referred to as JSQ-RR, always improves on RR and hence the base case, even if the delay of the system status updates tends to infinity. In fact, the response time performance of JSQ-RR is upper bounded by the performance of RR, and lower bounded by JSQ.

Nevertheless, the promising results raise several research questions still to be answered, e.g.:

- What is the performance under more general assumptions regarding the requests' arrival processes and their service requirements?
- What is, eventually, the impact of the resulting overhead (due to making the required system state information available) on the performance?
- What is the performance of alternative runtime dispatching strategies that don't introduce additional overhead, e.g. dispatching strategies based on response times from previously assigned jobs instead of explicit

("stale") system information?

- What is the performance of the observed dispatching strategies when the service composition comprises multiple abstract services?

ACKNOWLEDGMENT

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 Agentschap NL.

REFERENCES

[1] E. Altman, U. Ayesta, and B.J. Prabhu. Optimal load balancing in processor sharing systems. In Value Tools '08, pages 1–10, ICST, 2008.

[2] F. Bonomi. On job assignment for a parallel system of processor sharing queues. IEEE Trans. Comput., 39(7):858–869, 1990.

[3] S. Borst. Optimal probabilistic allocation of customer types to servers. Proc. ACM SIGMETRICS, pages 116–125, ACM, 1995.

[4] G. Canfora, M. Di Penta, R. Esposito, and M.L. Villani. An approach for QoS-aware service composition based on genetic algorithms. Gecco Proceedings, 2005.

[5] J. Cao and C. Nyberg. An approximate analysis of load balancing using stale state information for servers in parallel. Proc. Second IASTED International Conference on Communications, Internet, and Information Technology, pages 17–19, 2003.

[6] Y.-C. Chow and W. H. Kohler. Models for dynamic load balancing in a heterogeneous multiple processor system. IEEE Trans. Comp., 28(5):354–361, 1979.

[7] D. G. Coffman, N. R. Muntz, and H. Trotter. Waiting time distributions for processor-sharing systems. Journal of the ACM, 17(1):123–130, 1970.

[8] Université de Montréal, Département d'Informatique et de Recherche Opérationnelle (DIRO). Stochastic simulation in java, SSJ. http://www.iro.umontreal.ca/~simardr/ssj/indexe.html, last accessed January 2011.

[9] L. Flatto and H. P. McKean. Two queues in parallel. Comm. Pure and Applied Mathematics, 30:255–263, 1977.

[10] V. Gupta, M. Harchol-Balter, K. Sigman, and W. Whitt. Analysis of join-the-shortest-queue routing for web server farms. Performance Evaluation, 64(9–12):1062–1081, 2007.

[11] G.J. Hoekstra, Rob van der Mei, Y. Nazarathy, and A.P. Zwart. On the sojourn time tails of a file–splitting processor sharing network. Proc. NET-COOP, LNCS Vol. 5894, 2009.

[12] S. Hwang, E. Lim, C. Lee, and C. Chen. Dynamic web service selection for reliable web service composition. IEEE Trans. Services Comp., 1(2):104–116, 2008.

[13] G. A. Lewis, E. Morris, D. B. Smith, S. Simanta, and L. Wrage. Common misconceptions about service-oriented architecture. Proc. Sixth International IEEE Conference COTS-Based Software Systems, pages 123–130, IEEE, 2007.

[14] Y. Liu, A.H.H. Ngu, and L. Zeng. QoS computation and policing in dynamic web service selection. Proc. 13th International World Wide Web Conference, pages 66–73, 2004.

[15] Z. Liu, N. Niclausse, C. J. Vilanueva, and S. Berbier. Traffic model and performance evaluation of web servers. Performance Evaluation, 46(2–3):77–100, 2001.

[16] M. Mitzenmacher. How useful is old information? IEEE Trans. Parallel and Dist. Syst., 11(1):6–20, 2000.

[17] R. D. Nelson and T. K. Philips. An approximation to the response time for shortest queue routing. ACM Performance Eval. Review, 17(1):181–189, 1989.

[18] R. D. Nelson and T. K. Philips. An approximation to the response time for shortest queue routing with general interarrival and service times. IBM T.J. Watson Research Lab Technical Report RC15429, 1990.

[19] L. D. Ngan, M. Kirchberg, and R. Kanagasabai. Review of Semantic Web Service Discovery Methods, Proc. IEEE 6th World Congress on Services, pages 176–177, IEEE, 2010.

[20] A. Padovitz, S. Krishnaswamy, and S. Loke. Towards efficient selection of web services. Second Int. Joint Conf. on Autonomous Agents and Multi-Agent Systems, July 2003.

[21] S.R. Ponnekanti and A. Fox. Sword: A developer toolkit for web service composition. Proc. 11th International World Wide Web Conference, pages – 2002.

[22] Henk Tijms. *A First Course in Stochastic Models*. John Willey and Sons, Chichester, England, 2003.

[23] T. Yu, Y. Zhang, and K.-J. Lin. Efficient algorithms for web services selection with end–to–end QoS constraints. ACM Trans. Web, 1(1):p.6-es, 2007.

[24] L. Zeng, B. Benatallah, A.H.H. Ngu, M. Dumas, J. Kalagnanam, and H. Chang. QoS-aware middleware for web services composition. IEEE Trans. Softw. Eng., 30(5):311–327, 2004.