QoS Driven Dynamic Binding in-the-many

Carlo Ghezzi, Alfredo Motta, Valerio Panzica La Manna,
and Giordano Tamburrelli

Politecnico di Milano
Dipartimento di Elettronica e Informazione, Deep-SE Group
Via Golgi 42 – 20133 Milano, Italy
{ghezzi,motta,panzica,tamburrelli}@elet.polimi.it

Abstract. Modern software systems are increasingly built out of services that are developed, deployed, and operated by independent organizations, which expose them for the use by potential clients. Services may be directly invoked by clients. They may also be composed by service integrators, who in turn expose the composite artifact as a new service. We envision a world in which multiple providers publish software artifacts which compete with each other by implementing the same "abstract" service (i.e. they export the same API and provide the same functionality), but offering different quality of service. Clients may therefore select the most appropriate services targeting their requirements, among all the competing alternatives, and they may do so dynamically. This situation may be called *dynamic binding in-the-many*. Service selection may be performed by clients by following different strategies, which may in turn affect the overall quality of service invocations.

In this paper we address the problem of analyzing and comparing different service selection strategies and we define a framework to model the different scenarios. Furthermore, we report on quantitative analyses through simulations of the modeled scenarios, highlighting advantages and limitations of each solution.

1 Introduction

Software is rapidly changing in the way it is developed, deployed, operated, and maintained. Applications are often built by integrating components that are produced by independent organizations, or by personalizing and adapting generalized frameworks, rather than developing new solutions in-house and from scratch. Increasingly, applications are built by integrating and composing services that are developed, deployed, operated, and maintained by separate organizations, which offer them as *services* on the market. Services may be invoked directly by clients through some standardized protocols. They may also be composed and integrated by brokers to provide added-value services that are–in turn–exposed for use by potential clients. The long-term vision of *service-oriented computing (SOC)* is that Software as a Service is publicized in an open marketplace through *registries*, which contain the service descriptions that may be of interest for potential clients, and then invoked directly as needed [9].

G.T. Heinemann, J. Kofron, and F. Plasil (Eds.): QoSA 2010, LNCS 6093, pp. 68–83, 2010.

Another key aspect of modern software is its continuous change. Traditionally, change is primarily due to changing requirements, which–in turn–occur because of changes in the business world. Increasingly, however, software is becoming pervasive, and ubiquitous application settings are becoming more and more common. In this context, dynamic environmental change is a dominant factor. It is therefore necessary to develop solutions that may adapt to continuous changes that may arise dynamically, as an application is running and providing service. In conclusion, software increasingly lives in a dynamic and open world, and this poses new challenges to the way it is developed, operated, and evolved [1].

This paper deals with the SOC setting. Although in the current stage most existing approaches are still rather static, continuous evolution is intrinsic in SOC and will play a stronger role in the future. In our work, we envision a world that is fully populated by services that are offered in an open *service marketplace*. Multiple providers publish software artifacts which compete with each other, possibly implementing the same "abstract" service (i.e. they export the same API, which is advertised in registries) but offering different quality of service. Quality may refer to several factors, such as performance, reliability, or cost-per-use. Clients may therefore select the most appropriate services targeting their requirements, among all the competing alternatives, and they may do so dynamically. To support this dynamism, we assume that application components that use external services do not bind to them statically, but rather refer to the required external services through a *required interface*. External services are therefore viewed as *abstract* resources, and the binding to a specific, concrete service may occur dynamically. Since many possible concrete services may be selected for each abstract service, this may be called *binding in-the-many*. The main advantage of a late binding regime is its flexibility. A client may at any time refer to the best possible server, where *best* is defined by some appropriate strategy that may take into account the current state of the world and may maximize the ability of the resulting solution to satisfy the evolving requirements.

This paper focuses on how to support clients in performing dynamic binding to services. In particular, it deals with the service selection problem where clients try to bind to the external services that can ensure the best results in terms of *performance*. In principle, performance attributes might be publicly available as part of the published description of *quality of service* (QoS) that is stored in registries [11]. Clients might use this information to drive their binding policy. In practice, publication of QoS data is unlikely to happen, and even if it does, the published figures might not correspond to the real values. Clients may thus base their binding strategies on the real performance data observed in practice. In this work we describe a first step towards the development of a framework that supports a quantitative analysis of the different strategies that may be followed by clients to perform dynamic binding.

The paper is organized as follows: Section 2 illustrates the service selection problem and the assumptions under which we analyze possible solutions. Section 3 describes different service selection strategies. Section 4 illustrates the simulations we performed to evaluate the different strategies. Section 5 discusses

related work. Section 6 concludes the paper summarizing the current limitations of our approach and how they can be tackled by future work.

2 Problem Statement: Assumptions and Formalization

In this section we formalize the service selection problem using an appropriate stochastic framework. For simplicity, we assume that services can be grouped in equivalence classes (called *concrete target sets CTSs*), where all services in a CTS provide the same interface and implement the same functionality. In reality, equivalent services may actually differ in details of their interface, but we can assume that suitable adaptation mechanisms allow clients to abstract away from these technical differences. Adaptation techniques are beyond the scope of this paper but many existing techniques, such as [3,8], tackle this research problem. By relying on these assumptions, any two elements in a CTS are substitutable to one another, since they implement the same abstract service. Any of them can be bound to an abstract service invocation performed by a client, thus reifying a specific dynamic binding. They may differ, however, in the *quality of service* they provide.

Given a CTS, which groups all substitutable implementations of a certain abstract service, we analyze the behavior of competing clients, which need to bind to a concrete service offered by a specific provider to submit their requests. We assume services to be stateless and interactions between clients and services to be of type *request-response*. For simplicity, we measure the execution time of a service invocation as the overall time observed by the client, without further decomposing it into transmission time and service processing time. This simplifying assumption will be removed in future research.

The problem may be formalized as a *multi-client multi-provider stochastic framework*, which is a 6-tuple:

$$< C, P, F1, F2, F3, SS >$$

where:

- $C =< c_1, ..., c_n >$ is a set of *clients*, all of which use the same abstract service,
- $P =< p_1, ..., p_m >$ is a *concrete target set* of substitutable services,
- $F1 : C \times \mathbb{N} \rightarrow [0, 1]$ is a probabilistic *request submission* function,
- $F2 : C \times \mathbb{N} \rightarrow \mathbb{R}$ is a probabilistic *request payload* function,
- $F3 : P \times \mathbb{N} \rightarrow \mathbb{R}$ is a *service processing rate* function,
- SS is a service provider selection strategy.

We assume *time* to be modeled discretely by Natural numbers in functions $F1$, $F2$, and $F3$. The intuitive interpretation of the above functions is provided below. Providers are characterized by their time-varying processing rate $F3$, a real number, which defines the speed at which providers process requests coming from clients. Each client, at each time instant, is either *idle* or *busy*. Clients that

are busy wait for the completion of their last request. Clients that are idle may submit a new request with probability given by $F1$. Whenever a new request is generated, the client selects one of the available service providers according to the strategy SS. Function $F2$ describes the *payload* of a service request, issued by a certain client at a certain time. The payload is a real number that expresses the complexity of the submitted request. It is used to normalize the service's response time to the complexity of the requests issued by the client. If a client c submits a request with payload $F2(c, t_{start})$ at time t_{start} and the response is received at time t_{stop}, we can compute the service's *timeliness* as:

$$T = (t_{stop} - t_{start})/F2(c, t_{start})$$

Any conceivable strategies used by clients, including the ones we will describe and analyze in this paper, use historical information collected by the client during its past interactions with external services. Such historical information reifies the client's experience and its view of the external world. Each client c traces its history into a vector, the *efficiency estimator* ee_c. The length of this vector corresponds to the cardinality of CTS, and the i^{th} entry in the vector $ee_c(i)$ is a positive real value that represents the client's estimate of the current efficiency of service provider i. In addition to ee_c , client c has a vector jd_c, which stores the number of completed requests it submitted to each of the service providers, since the beginning of time. When the request is completed in normalized time T, vector ee_c associated with client c is updated as follows:

$$ee_c(p) := W \cdot T + (1 - W)ee_c(p)$$

where W is a real value in the interval $[0, 1]$, whose actual value depends on $jd_c(p)$ using the following formula:

$$W = w + (1 - w)/jd_c(p)$$

In the above formula w is a real-valued constant in the interval $[0, 1]$. The term $(1-w)/jd_c(p)$ is a correcting factor, whose effect becomes negligible when $jd_c(p)$ is sufficiently high. Other functions may be used to update ee_c and could be explored in a future work.

As we assumed, any request generated by clients may be bound to any service provider of a CTS. We may further assume that service providers handle the clients' requests sequentially, for example following a FIFO policy (this assumption can be easily relaxed if necessary). The response time of a certain provider p at a given time t, as observed by clients, obviously deteriorates as the queue of clients' requests waiting to be served grows.

3 Binding Strategies

In this section we illustrate six possible selection strategies through the concepts introduced in the previous section. Each of these strategies defines the binding

between the services and clients which issue requests. In particular, two of them (the *Minimum* and the *Probabilistic* strategy) are based on autonomous decisions made by each client, which selects the target service only on the basis of its own locally available information. Conversely, the *Collaborative* strategy allows a simple form of coordination among clients. In the *Proxy-based* approaches the service binding is delegated to a proxy, which assigns providers to clients based on its global knowledge of the current situation. Two strategies are analyzed in this case: *Proxy Minimum* and *Proxy Probabilistic*. Finally, we also analyze an *Ideal* strategy, which assumes an ideal case where all the clients have a perfect instantaneous knowledge on the real services' performance. This strategy, even if not applicable in a real implementation, has been studied in our experiments to show how far can we go if each client tries to maximize its own performance goals in isolation. As introduced in Section 1 we focus on performance expressed in terms of response time. In the following, we describe the selection strategies and we anticipate some remarks on their practical effectiveness. An empirical assessment is reported later in section 4.

3.1 Minimum Strategy

The Minimum Strategy (MS) is the simplest selection strategy in which each client selects the service provider with the minimum locally estimated expected response time. This information is derived by the efficiency estimator ee_c:

$$selected\_provider = p \in P | (min_{p \in P}\{ee_c(p)\})$$

This strategy suffers from a number of drawbacks, since ee_c (1) is different for each client (it depends on its own history) and (2) contains information that may be outdated, because each value $ee_c(p)$ is re-computed only when provider p is used.

This strategy has limitations also from the provider's perspective. In fact, the provider offering best performance figure is likely to be selected by the majority of clients, and hence its performance may deteriorate because of the increased number of issued requests.

3.2 Ideal Strategy

The Ideal Strategy (IS) is a benchmarking strategy used only for the comparison with other strategies. It is based on the following assumptions:

Each time a client needs to select a service, it knows:

- The number of pending requests for each provider p, $pr(p)$.
- The processing rate of each service provider p, $R(p)$.

IS selects the concrete service p which exhibits the maximum value of the current processing capacity cr, defined as:

$$cr(p) = \frac{R(p)}{pr(p)}$$

This strategy is ideal because it assumes complete and accurate knowledge of the available concrete services. It can be seen as an extreme optimal case of MS.

3.3 Probabilistic Strategy

The Probabilistic Strategy (PS) performs service selection with a certain probability. PS uses a probabilistic function based on the efficiency estimator ee_c.

We first define the following function:

$$pd'_c(p) := \begin{cases} ee_c(p)^{-n} & \text{if } jd_c(p) > 0, \\ E[ee_c]^{-n} & \text{if } jd_c(p) = 0 \end{cases}$$

where n is a positive real-valued parameter and $E[ee_c]$ represents the average of the values of $ee_c(p)$ over all the providers that were used in the past. To turn this value into a probability function we define the pd_c as the normalized version of pd'_c:

$$pd_c(p) := \frac{pd'_c(p)}{\sigma}$$

where $\sigma = \sum_{p \in P} pd'_c(p)$

The function pd_c depends on n: the larger n is, the higher the probability to select providers with smaller estimated response time. In extreme cases, where the value of n is very high (e.g., 20), the client behaves like in the Minimum Strategy, always choosing the provider with the best record in ee_c. Using the terminology of reinforcement learning, parameter n defines how much a client is *explorative* in its search. The intuitive advantages of using PS over MS, confirmed by the experiments described later in section 4 are:

1. It guarantees a better load balancing of clients among available providers. In fact, by construction, the probability defines a distribution of client requests with respect to providers. This reduces the size of their associated queues and improves the average response time.
2. It improves the efficiency of the choice with respect to the MS. According to MS, a client performs the same binding to the provider p until the estimated response time of the selected provider (stored in $ee_c(p)$) exceeds the value of the estimate of another provider q. PS is less conservative, and may select a new provider q earlier, avoiding to experience performance deterioration of provider p.

3.4 Collaborative Strategy

According to both MS and PS, the pth entry in the efficiency estimator is updated only if p is used. Thus both strategies are based on information (kept in ee_c), which may not reflect the current situation of available concrete services. Because the ideal strategy cannot be implemented, we try to achieve a better view of the current performance offered by concrete services through the introduction of a simple form of communication among clients in PS. This yields a new algorithm, called Collaborative Strategy (CS). In CS, the set of clients C is partitioned in classes (NCs) according to a *near-to* relation. We assume that

each client can communicate only with the clients belonging to the same NC. The communication consists of sharing all the efficiency estimators of clients in a NC in order to derive a new vector called *near-to estimator* ne, computed as the average of all the ees. On the basis of ne, the client performs its choice in the same way as in MS or in PS. Each client c keeps its own ee_c and communicates with other clients whenever a new binding occurs, by recomputing ne. By grouping together a set of clients which perform their decision on the basis of a common vector we can mitigate the problem arising in PS due to outdated records in ee_c. Two possible collaborative strategies are thus possible: minimum value based and probabilistic. We later show an assessment of the latter, which outperforms the former. In fact, the Collaborative Strategy, as an extended version of PS, inherits all the advantages of PS over MS in terms of efficiency and fair load balancing.

3.5 Proxy-Based Approach

We can try to achieve a global view of the world by introducing a decoupling layer between the clients and the concrete services, i.e. a *proxy*. In proxy-based strategies a proxy is responsible for: (1) selecting the best concrete service within CTS and (2) linking the selected service with the client that issued the request. The proxy can be considered as a binding establishing service. In a configuration without a proxy, the client invokes the selected concrete service directly. To do so, the client has to take into consideration all the different interfaces offered by the various providers to invoke the service, and must implement a particular mapping from the abstract interface into the concrete interfaces offered by concrete services, if needed. The proxy-based solution simplifies the way clients interact with concrete services by letting the proxy handle the nitty-gritty details of the interaction and select the concrete service on their behalf. Furthermore, the proxy can be seen as a load balancer, which tries to distribute load to get the best response time on behalf of clients.

Beside keeping trace of t_{start} and t_{stop} for each requests submitted by clients, the proxy is also in charge of computing two vectors:

- The efficiency estimator ee which records the history of interaction with all concrete services. Unlike the previous strategies, ee records information about response time for *all* clients.
- The pending requests vector pr, which collects all the pending requests sent to each provider.

The main advantage of proxy based solutions is that a single efficiency estimator, which is shared by all clients, is updated more frequently than in the other approaches. For this reason a more accurate performance estimate is obtained for the concrete services. We investigate two possible service selection strategies, Proxy Minimum Strategy (PMS) and Proxy Probabilistic Strategy (PPS) which represent a modified version of the strategies described previously.

The reader might suspect that proxy-based strategies require a centralized component acting as a proxy, thus generating a bottleneck and a single point

of failure. This, however, is only a logical view of the architecture. The proxy layer may in fact be physically distributed over a number of nodes for example connected in a token ring structure. Each node of the ring contains:

- the centralized efficiency estimator ee computed as the average of all the ee_i locally stored in each proxy.
- the pr as the sum of all the pr_i maintained by each proxy.

However this is just one possible architecture that has not yet been validated by concrete experiments. The appropriate token speed and the impact of communication delays are beyond the scope of this paper and must be investigated. Future work will focus on these issues.

4 Validation

In this section we illustrate the results obtained by simulating in Matlab the selection strategies introduced in Section 3, to draw some conclusions about their effectiveness.

Given the probabilistic nature of our framework every simulation has been performed 100 times and then the average of results has been considered. We used the following parameters for our simulations:

- $\#C = 100$, number of clients.
- $\#P = 5$, number of service providers.
- time interval= 3 hours.

Each provider has a certain processing rate $F3$ that may assume one of the values in the set [40, 20, 10]. The value of the payloads of service invocation $F2$ has a uniform distribution between 50 and 150. Finally for CS we also define the number (5) and the size (20 clients) of the NC classes for all the simulations. In order to compare the different service selection strategies, we focus on service timeliness T. That is, we evaluate the different selection strategies by comparing the resulting values of T, averaged over all requests.

For space reasons, we cannot provide an extensive report on the experimental results we obtained through simulation. In particular, we will only summarize some basic findings regarding parameter tuning in Section 4.1 while section 4.2 will give some detail insights through charts on the different selection strategies.

4.1 Parameters Tuning

The selection strategies presented in Section 3 are characterized by two internal parameters which may affect the selection behavior and the resulting efficiency: (1) w, which defines how to update ee, and affects all the strategies; (2) n, which defines the probability distribution of selection, which affects probabilistic strategies. On the basis of preliminary experiments, concrete values of these parameters are derived and commented in this subsection.

A related study on the importance of such parameters in the context of multi-agent load balancing is described in [10].

Parameter w determines the importance of the last normalized response time with respect to the previous one. The greater w is, the more weight is given to the current record. Given these premises it is interesting to understand how the efficiency of the strategies is influenced by this parameter in different settings of the load of the system.

If the load of the system is fixed and equally distributed among all clients, the setting is completely static and at any invocation the current response time record is the same as the previous. In this case, both pure client-based (MS, PS, IS) and proxy-based strategies (PMS, PPS) are independent of w. Instead, if the load generated by clients changes dynamically, client-based strategies perform better with higher values of w ($w = 0.9$), because in such cases the recently experienced timeliness values weigh more than the previous in the update of *ee*. Proxy-based strategies are instead less dependent on w, since *ee* intrinsically keeps a more current view of services' performance.

It is also interesting to understand how n influences the choices of PS and PPS. This may be used to find a good balance between an *explorative* and a *conservative* behavior. In a situation of fixed load, PS achieves a better result with large values of n. This is due to the fact that, whenever there are no changes in the dynamics of the system, it is obviously better to always choose the best provider, hence to reduce PS to MS. On the other hand, if the load is variable when n is too high the results are worse. The same results were obtained for PPS. Good values of n are in the middle of the interval [1 10]. If n is too high, PPS tends not to change the already selected provider p, so if the processing capacity of p decreases, performance may deteriorate. On the other hand, if n is too small PPS does not correctly exploit the knowledge given by *ee* and *pr* performing a random selection.

In conclusion, $w = 0.9$ and $n = 6$ are the values used in the simulations described in 4.2.

4.2 Empirical Results

We performed experiments to evaluate the six binding strategies discussed in Section 3. Experiments focus on the following cases:

1. Fixed provider processing rates
2. Variable provider processing rates
3. Heterogeneous client populations

Fixed provider processing rates. In our first experiment (Figure 1) we compare the performance of the six strategies focusing on the case where all the clients have the same request submission function $F1$ and the providers have a constant processing rate $F3$. We fixed $F1$ for the set of clients to one of the values in the set [0.001, 0.003, 0.010], and this value is kept constant over time. Function $F3$ for each of the available concrete services, [p_1, p_2, p_3, p_4, p_5] assumes the values [40, 20, 20, 10, 10], respectively.

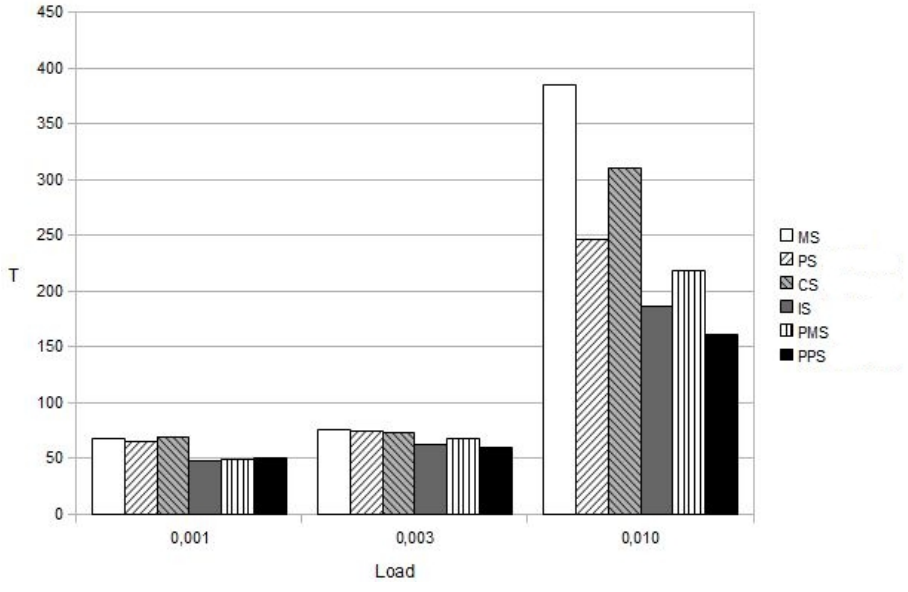

Fig. 1. Performance of selection strategies: fixed provider processing rate and fixed client load

We show the timeliness results provided by the simulation over all the requests made by the clients to the service providers using a certain strategy. Thus, the lowest the bar, the better is the performance of the strategy. From Figure 1 we can draw some conclusions:

- PS performs better than MS, especially as the load of client submissions increases. This is fairly intuitive because all the clients tend to select the same provider that is performing well in that specific instant of time, with the side effect that some clients will wait in a long queue. PS, thanks to its exploration characteristics does not have the same problems and performs reasonably well.
- PS outperforms also CS. NC classes tend to reproduce the same conservative behavior that characterize MS.
- With high load, PPS outperforms IS. Thus, despite its name, IS does not yield the ideal result. Indeed, although IS has a global view of the current providers' performance, it misses a crucial piece of global information, namely the current requests issued by the other clients. Proxy-based strategies, on the other hand, are aware of the current load exerted by clients.

Similar conclusions can be derived by examining Figure 2, which illustrates a situation where clients change their behavior. Specifically, the load is variable among the values [0.001, 0.003, 0.01]: during a 3 hour simulation, the request submission function $F1(c, t)$ of each client randomly changes every 20 minutes

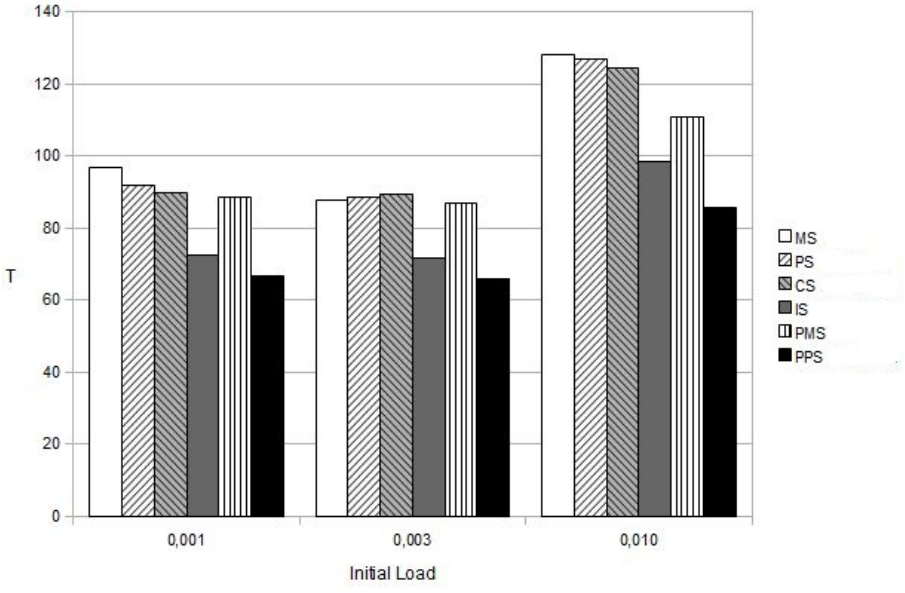

Fig. 2. Performance of selection strategies: fixed provider processing rate and variable client load

and assumes one of the previously specified values. The 3-hour simulation shows slightly different results (but a similar pattern) depending on the initial load chosen for the simulation. A larger simulation period would smooth the differences due to the initial load.

Variable provider processing rates. In Figures 1 and 2 we have seen how the different selection strategies behave depending on changes in the load generated by clients. We now show what happens when concrete services change their behaviors; e.g., a provider decreases or increases its performances. This is exactly the case of Figure 3 where every 20 minutes we change the processing rate of the service providers, which randomly takes a value in the set $[40; 20; 10]$, and this value is kept constant until the next random selection. The results show that the on-client selection strategies (MS, PS and CS) suffer most the changing environment because they do not have enough information for timely reaction. IS turns out to deliver the best results thanks to its complete knowledge of the instantaneous changes in the world. However, PPS still remains very close to IS and it gives evidence of its adaptive capability.

Figure 4 shows the results of a scenario with variable client load in addition to the changing processing rate of the providers. This change increases the unpredictability of the system, but the different strategies behave like in the previous case, although here PPS performs better than IS.

Heterogeneous client populations. In the next examples we are going to relax the hypothesis that all the clients use the same request submission function $F1$.

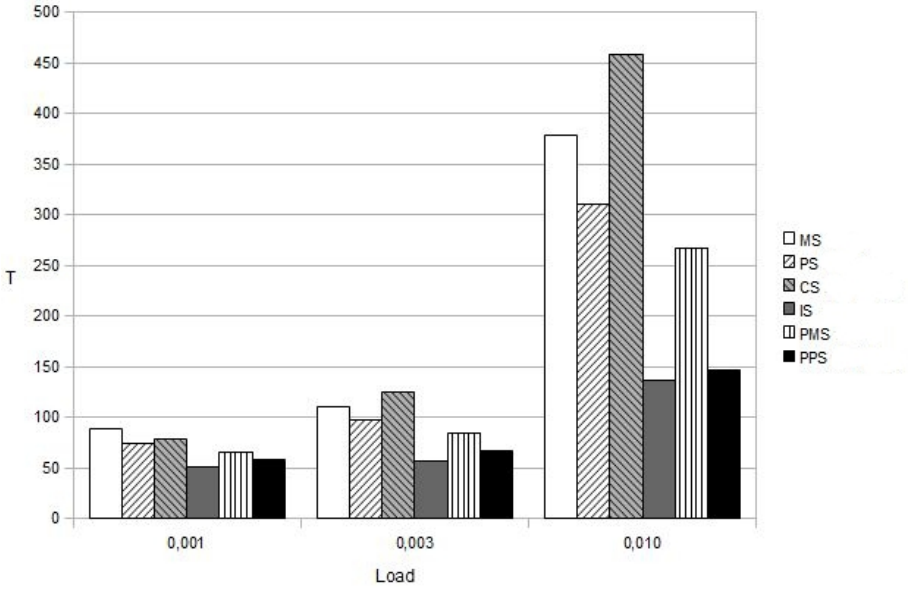

Fig. 3. Performance of selection strategies: variable provider processing rate and fixed client load

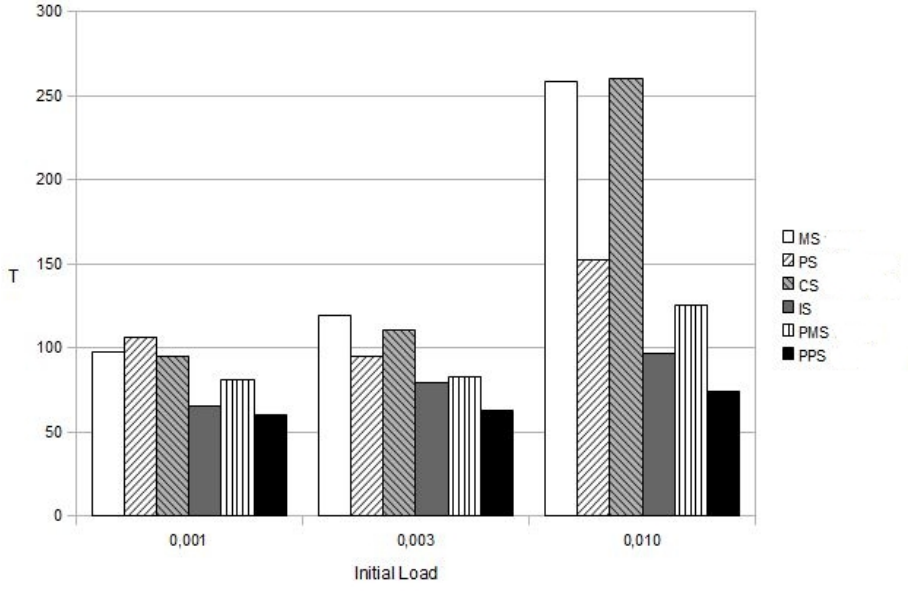

Fig. 4. Performance of selection strategies: variable provider processing rate and variable client load

Specifically, we consider the set of clients C to be partitioned in two populations: the first, called C1, which submits a low number of requests ($F1(c,t) = 0,001\ \forall c \in C1,\ t \in \mathbb{N}$) and the second, called C2, which submits a higher number of requests ($F1(c,t) = 0,01\ \forall c \in C2,\ t \in \mathbb{N}$).

Figure 5 shows the performance of $C1$ and $C2$ using the different strategies in a setting where the processing rate of the providers is fixed.

As shown in Figure 5, the two different classes perform equally in any of the service selection strategies. This is fairly intuitive because the processing capacities of the providers and the load of the system are fixed. As a consequence, the performance estimation, represented by the efficiency estimator ee, is always updated.

The last results we report (Fig. 6) illustrate the case where we modify the environment at run-time by changing the processing rate $F3$ of the service providers in order to see if the class $C1$ of clients that submit requests with less frequency still performs like the class of clients $C2$. Figure 6 shows that the clients of class $C2$ (high load) that are using MS, CS and PS perform better than the clients belonging to class $C1$ (low load). This shows that the more one issues requests to the service providers, the more local efficiency estimators are updated, hence the closer one gets to selecting the best service provider. However this is an undesirable behavior, because in principle a fair strategy should not distinguish clients with different submission rates. Proxy-based approaches (both PMS and PPS) solve this problem. In fact the ee vector is the same for all the clients and it is stored in the proxy, and obviously the two classes of clients will have the same performance. Notice also that IS, thanks to its complete knowledge, is always up-to-date, thus even if clients of class $C1$ submit less requests than clients of class $C2$, it always chooses the best service provider.

5 Related Work

The problem of supporting automatic service composition has become a hot research theme in the recent years. Many recent papers deal with service selection, focusing on particular aspects that differ from this study.

Most of them concentrate on the *Web Service* scenario where delivering QoS is a critical and significant challenge because of its dynamic and unpredictable nature. Service composition here is done using the performances declared by the external service provider assuming that those values do not change at run-time. This is also the main difference with our research work, which does not rely on declared performance data, but rather on values measured at run-time. In other words we cannot assume that the declared level of performance is guaranteed. For this reason, the strategies we propose automatically select the services at run-time based on response time measurements which reflect the current situation.

In this context both Liu et al. [5] and Yu et al. [12] reason on how to satisfy end-to-end QoS constraints by modeling service composition as a graph. The former solves the problem using a genetic algorithm which produces a set of paths defining the optimal composition among services that satisfy the global QoS

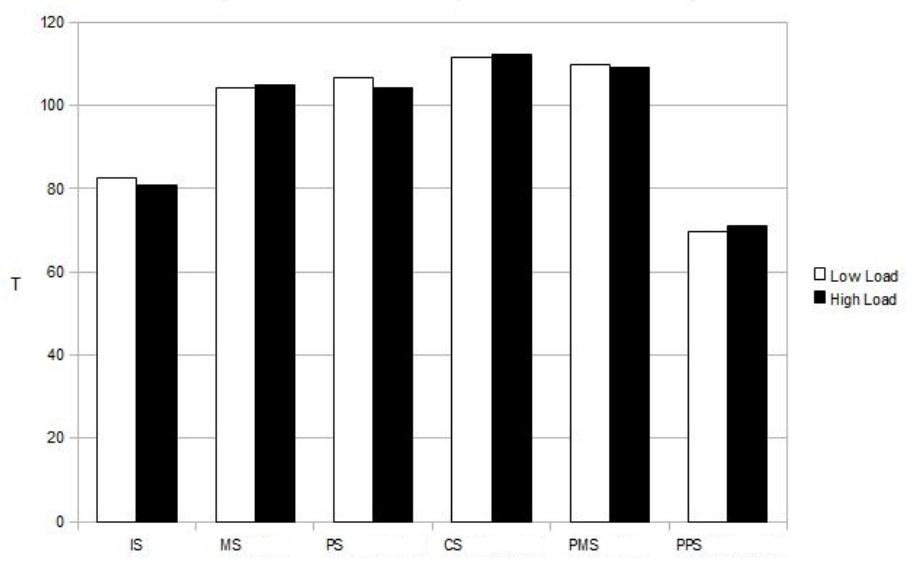

Fig. 5. Service Selection strategies performances of two populations of clients: C1 (low value of F1) and C2 (high value of F1). Providers have fixed processing rate.

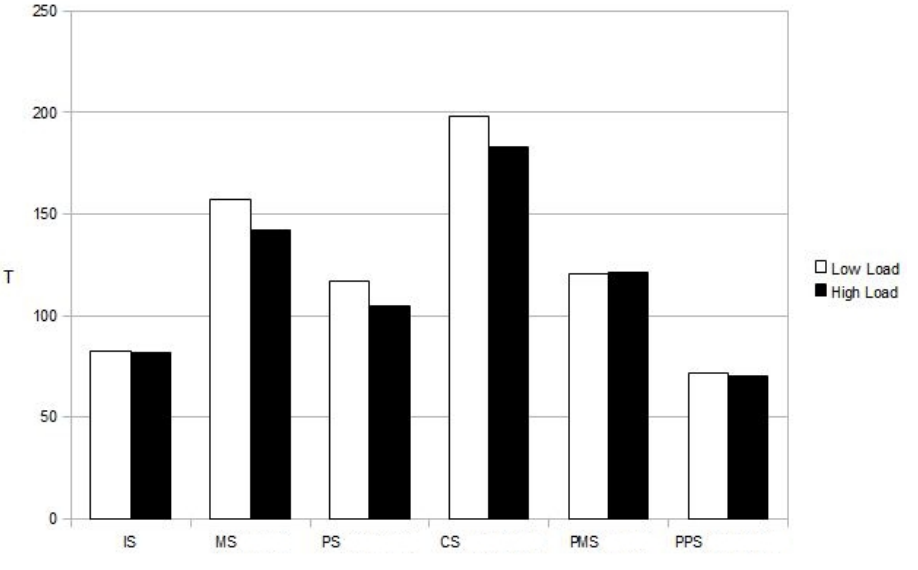

Fig. 6. Service Selection strategies performances with two heterogeneous population of clients: C1 (low value of F1) and C2 (high value of F1). Providers have variable processing capacities.

constraints on the basis of declared parameters. The latter proposes a broker-based architecture that can solve a multi-dimension multi-choice knapsack problem (MMKP) or a multi-constrained optimal path (MCOP) problem. We do not analyse end-to-end QoS because we concentrate on the substitution of a single service trying to optimize the expected performance of the new one.

Single service substitution is studied in [7], [4] and [6]. In [7] link analysis is applied to rank the QoS of the available services. In [4] a centralized service selector is proposed that collects user's feedback in order to rank the services. Finally, [6] relies on a centralized entity that ranks the available services using a recommendation system which exploits the item based collaborative filtering approach (see also [2]). None of these papers uses performance data to select external services, as instead we discussed here.

Finally, an important aspect of service substitution is to identify possible interaction mismatching [8] and to produce the correct mapping of the interfaces [3]. These, however, are out of the scope of the work we described here.

6 Conclusions

In this paper we addressed the issue of dynamically reconfiguring a composite service-based application in order to optimize its performance through dynamic binding. We focused on a number of binding strategies and we evaluated them via simulation. This is an initial but encouraging step towards understanding dynamically adaptable software applications.

We plan to extend this work in a number of interesting directions. First, we wish to refine our model and enrich its experimental assessment. For example, we would like to improve its accuracy by decomposing the response time observed by clients into network latency and service response time. Next, we wish to explore a distributed architecture for the proxy-based strategies, to eliminate a possible single point of failure that would arise in a centralized solution. Last, we wish to explore other reconfiguration strategies through which one can react to performance violations, other than dynamic binding, which may restructure the internals of the application or dynamically discover and add new service providers.

Acknowledgments

This research has been partially funded by the European Commission, Programme IDEAS-ERC, Project 227977-SMScom and by Project Q-ImPrESS (FP7-215013) funded under the European Union's Seventh Framework Programme (FP7).

References

1. Baresi, L., Di Nitto, E., Ghezzi, C.: Toward open-world software: Issue and challenges. Computer 39(10), 36–43 (2006)
2. Bianculli, D., Binder, W., Drago, L., Ghezzi, C.: Transparent reputation management for composite web services. In: ICWS 2008, pp. 621–628. IEEE Computer Society, Washington (2008)

3. Cavallaro, L., Di Nitto, E., Pradella, M.: An automatic approach to enable replacement of conversational services. In: ICSOC/ServiceWave, pp. 159–174 (2009)
4. Huebscher, M.C., McCann, J.A.: Using real-time dependability in adaptive service selection. In: Proceedings of the Joint International Conference ICAS-ICNS 2005. 76 p. IEEE Computer Society Press, Washington (2005)
5. Liu, S., Liu, Y., Jing, N., Tang, G., Tang, Y.: A dynamic web service selection strategy with QoS global optimization based on multi-objective genetic algorithm. In: Zhuge, H., Fox, G.C. (eds.) GCC 2005. LNCS, vol. 3795, pp. 84–89. Springer, Heidelberg (2005)
6. Manikrao, U.S., Prabhakar, T.V.: Dynamic selection of web services with recommendation system. In: NWESP 2005: Proceedings of the International Conference on Next Generation Web Services Practices, 117 p. IEEE Computer Society, Washington (2005)
7. Mei, L., Chan, W.K., Tse, T.H.: An adaptive service selection approach to service composition. In: ICWS 2008: Proceedings of the 2008 IEEE International Conference on Web Services, pp. 70–77. IEEE Computer Society, Washington (2008)
8. Motahari Nezhad, H.R., Benatallah, B., Martens, A., Curbera, F., Casati, F.: Semi-automated adaptation of service interactions. In: WWW, pp. 993–1002 (2007)
9. Di Nitto, E., Ghezzi, C., Metzger, A., Papazoglou, M., Pohl, K.: A journey to highly dynamic, self-adaptive service-based applications. Automated Software Eng. 15(3-4), 313–341 (2008)
10. Schaerf, A., Shoham, Y., Tennenholtz, M.: Adaptive load balancing: a study in multi-agent learning. J. Artif. Int. Res. 2(1), 475–500 (1994)
11. Skene, J., Davide Lamanna, D., Emmerich, W.: Precise service level agreements. In: ICSE 2004, pp. 179–188 (2004)
12. Yu, T., Zhang, Y., Lin, K.-J.: Efficient algorithms for web services selection with end-to-end qos constraints. ACM Trans. Web 1(1), 6 (2007)