Probabilistic QoS and Soft Contracts for Transaction-Based Web Services Orchestrations

Sidney Rosario, Albert Benveniste, *Fellow*, *IEEE*, Stefan Haar, and Claude Jard

Abstract—Service level agreements (SLAs), or *contracts*, have an important role in Web services. These contracts define the obligations and rights between the provider of a Web service and its client, with respect to the function and the Quality of Service (QoS). For composite services like orchestrations, such contracts are deduced by a process called *QoS contract composition*, based on contracts established between the orchestration and the called Web services. These contracts are typically stated in the form of hard guarantees (e.g., response time always less than 5 msec). Using hard bounds is not realistic, however, and more statistical approaches are needed. In this paper, we propose using *soft probabilistic contracts* instead, which consist of a probability distribution for the considered QoS parameter—in this paper, we focus on timing. We show how to compose such contracts to yield a global probabilistic contract for the orchestration. Our approach is implemented by the *TOrQuE* tool. Experiments on *TOrQuE* show that overly pessimistic contracts can be avoided and significant room for safe overbooking exists. An essential component of SLA management is then the continuous monitoring of the performance of called Web services to check for violations of the agreed SLA. We propose a statistical technique for runtime monitoring of soft contracts.

Index Terms—Web services, composite Web services, Quality of Service (QoS).

1 Introduction

Web services and their orchestrations are now considered an infrastructure of choice for managing business processes and workflow activities over the Web infrastructure [33]. BPEL [3] has become the industrial standard for specifying orchestrations. Numerous studies have been devoted to relating BPEL to mathematical formalisms for workflows, such as WorkFlow nets (WFnets) [31], a special subclass of Petri nets, or the pi-calculus [27]. This has allowed developing analysis techniques and tools for BPEL [25], [4], including functional aspects of contracts [34], as well as techniques for workflow mining from logs [32]. Besides BPEL, the ORC formalism has been proposed to specify orchestrations, by Misra and Cook at Austin [23]. ORC is a simple and clean academic language for orchestrations with a rigorous mathematical semantics. For this reason, our study in this paper relies on ORC. Its conclusions and approaches, however, are also applicable to BPEL.

1.1 Contract-Based QoS Management

When dealing with the management of QoS, *contracts*—in the form of *Service Level Agreements* (SLA) [8]—specify the commitments of each subcontractor with regard to the orchestration. Standards like Web service Level Agreement (WSLA) [18] proposed by IBM allow for specifying (and monitoring) QoS parameters of Web services through contracts. Though there is no such standardization for QoS parameters of Web services, most SLAs commonly tend to have QoS parameters which are mild variations of the following: response time (latency), availability, maximum allowed query rate (throughput), and security. In this paper, we focus on response time.

From QoS contracts with subcontractors, the overall QoS contract between orchestration and its customers can be established. This process is called *contract composition*; it will be our first topic in this paper. Then, since contracts cannot only rely on trusting the subcontractors, *monitoring* techniques must be developed for the orchestrator to be able to detect possible violation of a contract by a subcontractor. This will be our second topic.

1.2 Hard versus Soft Contracts

To the best of our knowledge, with the noticeable exception of [21], [16], [17], all composition studies consider performance-related QoS parameters of contracts in the form of *hard bounds.* For instance, response times and query throughput are required to be less than a certain fixed value and the validity of answers to queries must be guaranteed at all times. When composing contracts, hard composition rules are used such as addition or maximum (for response times) or conjunction (for validity of answers to queries).

Whereas this results in elegant and simple composition rules, we argue that this general approach using hard bounds does not fit reality well. Fig. 1 displays a histogram

- *S. Rosario and A. Benveniste are with IRISA/INRIA, Campus de Bealieu, Avenue du Général Leclerc, 35042 RENNES Cedex, France. E-mail: {srosario, benveniste}@irisa.fr.*
- *S. Haar is with INRIA Saclay, LSV, CNRS & ENS de Cachan, 61, avenue du Président Wilson, 94235 CACHAN Cedex, France. E-mail: Stefan.Haar@inria.fr.*
- *C. Jard is with ENS Cachan, IRISA, ENS-Cachan/Bretagne, Campus de Ker-Lann, F-35170 BRUZ Cedex, France. E-mail: Claude.Jard@bretagne.ens-cachan.fr.*

Manuscript received 30 Oct. 2008; revised 10 Dec. 2008; accepted 12 Dec. 2008; published online 18 Dec. 2008.
For information on obtaining reprints of this article, please send e-mail to: tsc@computer.org, and reference IEEECS Log Number TSC-2008-10-0092.
Digital Object Identifier no. 10.1109/TSC.2008.17.

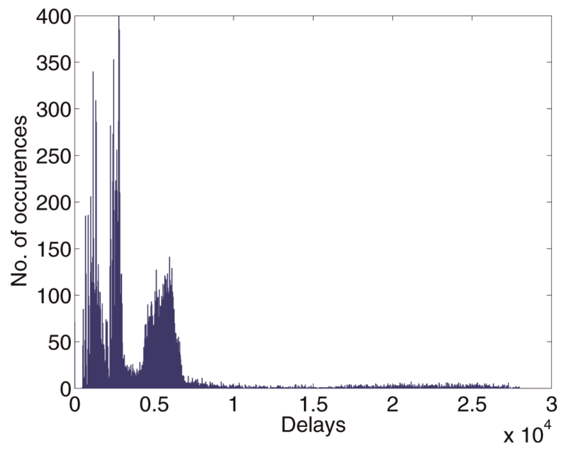

Fig. 1. Measurement records for response times for Web service StockQuote.

of measured response times for a "StockQuote" Web service which returns stock prices of a queried entity [35]. These measurements show evidence that the tail of the above distribution cannot be neglected. For example, in this histogram, percentiles of 90 percent, 95 percent, and 98 percent correspond to response times of 6,494 ms, 13,794 ms, and 23,506 ms, respectively. Setting hard bounds in terms of response time would amount to selecting, e.g., the 98 percent percentile of 23,506 ms, leading to an overly pessimistic promise for this service.

In fact, users would find it very natural to "soften" contracts: a contract should promise, e.g., a response time in less than T milliseconds for 95 percent of the cases, validity in 99 percent of the cases, accept a throughput not larger than N queries per second for 98 percent of a time period of M hours, etc. This sounds reasonable but is not used in practice, partly because soft contracts based on a single percentile (e.g., 95 percent or 99 percent of the cases) as above *lack composition rules*. To cope with this difficulty, we propose soft contracts based on probability distributions. As we shall see, such contracts compose well.

1.3 Soft Probabilistic Contract Composition

Having agreed on SLA or contracts with the different subcontractors, the orchestrator can then attach a probability distribution to the considered QoS parameters. If a combined executable functional-and-QoS model of the orchestration is available, it is then possible to compute the probability distribution of the same QoS parameter for the orchestration.

Such a combined functional-and-QoS model of the orchestration requires enhancing orchestration specifications with QoS attributes seen as random variables. This, however, is by itself not enough in general. More precise information regarding causal links relating events is needed. For example, latencies are added among events that are causally related, not among concurrent events. Thus, we need to explicit causality, concurrency, and sequencing in the orchestration in a precise way, which amounts to representing orchestrations as *partial orders* of events. Some mathematical models of orchestrations provide this, e.g., the partial order semantics of WorkFlow nets [31]. Our group has developed a tool, *TOrQuE* (**T**ool for **Or**chestration **Qu**ality of Service **E**valuation), that directly produces executions as partial orders from an ORC program. The results reported here were obtained by this tool.

1.4 Soft Probabilistic Contract Monitoring

An essential component of SLA management is the runtime monitoring of contracts. SLA monitoring must be continuous to timely detect possible SLA violations. In case of a violation, the called service may have to incur some agreed penalty. Alternatively, if the service is called by an orchestrator, the orchestrator might consider reconfiguring the orchestration to call an alternative service. The monitoring of probabilistic contracts requires using methods from statistics. We propose using statistical testing to check if the observed performance deviates from the performance promised in the contract.

1.5 Organization of the Paper

In Section 2.1, we present an example of an orchestration, which is then used to illustrate the primary challenges involved in QoS studies of Web services and their compositions. The example is also used in our experiments. In Section 3, we present our general approach for contract composition and describe the *TOrQuE* tool supporting it. The simulations on contract composition, which show a potential for overbooking, are given in Section 4. In Section 5, we introduce our technique for monitoring soft contracts. The experiments done on monitoring are reported in Section 6. Section 7 gives a survey of the existing literature on QoS-enabled WS composition. Finally, Section 8 presents conclusions and outlooks.

2 QoS Issues in Web Services and Their Compositions

In this section, we will explain the main challenges faced in QoS studies of Web services and their compositions. From this, we will draw conclusions regarding how QoS studies should be performed for Web services orchestrations. This is done with the help of a sample orchestration, CarOnLine, which we will present first. The CarOnLine example, which was developed in the *SWAN* project [26], is also used in our experimentations with the TOrQuE tool.

2.1 Example of an Orchestration

CarOnLine is a composite service for buying cars online, together with credit and insurance. A simplified graphical view of it is shown in Fig. 2.

On receiving a car model as an input query, the CarOnLine service first sends parallel requests to two car dealers (GarageA, GarageB), getting quotations for the car. The calls to each garage are guarded by a timer, which stops waiting for a response once the timeout occurs. If a timeout occurs, the response of the call is a Fault value. The best offer is chosen by the (local) function Mux, which returns the minimum nonfaulty value. If both timeouts occurs, Mux returns a Fault. Credit and insurances are found in parallel for the best offer. Two banks (AllCredit, AllCreditPlus) are

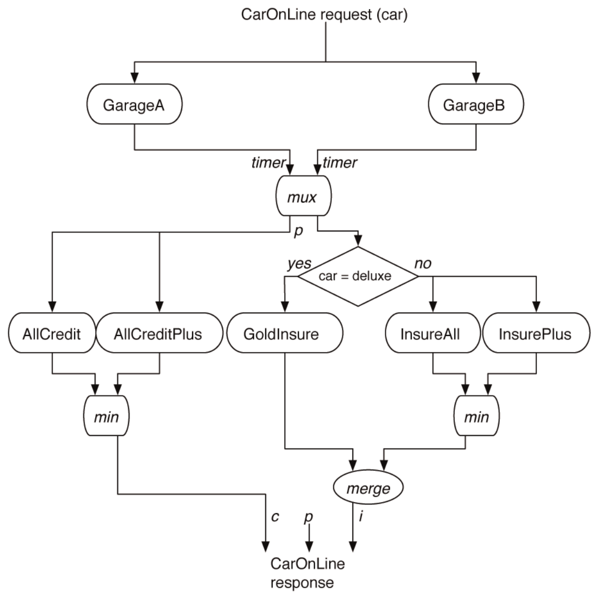

Fig. 2. A simplified view of the CarOnLine orchestration. The calls to GarageA and GarageB are guarded by a timer that returns a "Fault" message whenever the timeout occurs—this is not shown in the figure. In the discussion in Section 2.2 regarding "monotonicity," the test car = deluxe is changed to p $\geq$ limit.

TABLE 1
CarOnLine in ORC

```
$CarOnLine(car) \triangleq\ CarPrice(car)\ >p>\ \mathbf{let}(p, c, i)$
     where  $c :\in\ GetCredit(p)$
            $i :\in\ GetInsur(p, car)$

$CarPrice(car) \triangleq\ \{Mux(p1, p2)$
    where
       $p1 :\in (NetGA \gg GarageA(car)) \mid Timer(T)$
       $p2 :\in (NetGB \gg GarageB(car)) \mid Timer(T)$
  $\}\ >p>\ \{\ \mathbf{if}(p \neq Fault)) \gg let(p)\ \}$

$GetCredit(p) \triangleq\ Min(c1, c2)$
    where
       $c1 :\in NetC \gg AllCredit(p)$
       $c2 :\in NetCP \gg AllCreditPlus(p)$

$GetInsur(p, car) \triangleq\ \{\ \mathbf{if}(car = deluxe) \gg GoldInsure(p)\} \mid$
  $\{\ \mathbf{ifnot}(car = deluxe) \gg\ \{min(ip, ia)$
       where  $ip :\in\ InsurePlus(p)$
              $ia :\in\ InsureAll(p)$
    $\}\}$
```

queried for credit rates and the one offering a lower rate is chosen. For insurance, if the car belongs to the deluxe category, any insurance offer by service GoldInsure is accepted. If not, two services (InsurePlus, InsureAll) are called in parallel and the one offering the lower insurance rate is chosen. In the end, the (car-price (p), credit-rate (c), insurance-rate (i)) tuple is returned to the customer.

The ORC program for CarOnLine is given in Table 1. We chose to use ORC because it is an elegant language equipped with formal semantics [19], [28]. ORC defines three basic operators.

For ORC expressions f, g, "$f \mid g$" executes f and g in parallel. "$f > x > g$" evaluates f first and for *every* value returned by f, a *new* instance of g is launched with variable x assigned to this return value; in particular, "$f \gg g$" (which is a special case of the former where returned values are not assigned to any variable) causes *every* value returned by f to create a *new* instance of g. "f **where** $x :\in g$" executes f and g in parallel. When g returns its *first* value, x is assigned to this value and the computation of g is terminated. All site calls in f having x as a parameter are blocked until x is defined (i.e., until g returns its first value).

CarPrice calls GarageA and GarageB in parallel for quotations. Calls to these garages are guarded by a timer site Timer which returns a fault value T time units after the calls are made. The **let** site simply returns the values of its arguments—sites can only execute when all their parameters are defined and, thus, can be used to synchronize parallel threads. The value returned by CarPrice (here, the variable p) is passed as an argument to GetCredit and GetInsur which find credit and insurance rates for the price in parallel. The service NetGA in NetGA $\gg$ GarageA(car) is a dummy service that captures the contribution of the network to the response time of GarageA as perceived by the orchestration. No such call occurs in GetInsur. This is because the orchestration does not enter into contracts with the insurance sites, which are assumed to be freely available. The absence of a contract requires estimating the insurance sites' and the associated network's performance. This is discussed in the next section.

2.2 QoS Issues for Web Service Orchestrations

With the help of CarOnLine, we now discuss how the QoS issues for service orchestrations differ from traditional QoS studies.

2.2.1 Flow May Be Data Dependent

In the GetInsure component of CarOnLine, there are two exclusive ways for getting insurance quotes for a car: either by calling GoldInsure or by calling InsureAll and InsurePlus in parallel. The choice of which branch is taken depends on the value of the parameter "car." In most orchestrations, the execution flow usually depends on the values of its different data parameters, which are unknown a priori. Thus, by changing its execution flow, data values in an orchestration can directly affect its QoS.

2.2.2 Flow May Be Time Dependent

In CarPrice component of CarOnLine, the calls to GarageA and GarageB are guarded by a timer. Depending on whether or not the garages respond before the timeout occurs, the orchestration may decide to take different execution paths, directly affecting its performance. Thus, the presence of timers in orchestrations can also alter its control flow.

2.2.3 Orchestrations May Not Be "Monotonic"

An implicit assumption in contract-based QoS management is: "the better the component services perform, the better the orchestration's performance will be." Surprisingly, this property that we called "monotonicity" [9] can easily be violated, meaning that the performance of the orchestration may improve when the performance of a component service degrades. This is highly undesirable since it can make the process of contract composition inconsistent. A contract-based approach needs monotonicity.

Consider the CarOnLine orchestration of Fig. 2, but slightly modified. The condition "car = deluxe" for deciding calls to insurance services is changed as follows: If the best price returned by the garages is p, then GoldInsure is called if $p \geq limit$, where limit is a certain constant value. If $p < limit$, InsurePlus and InsureAll are called in parallel. Assume that the credit services AllCredit and AllCreditPlus respond extremely fast (almost 0 time units) and so the response time of the orchestration only depends on the response time of the garage and insurance services. Let the response times of the garage and insurance services GarageA, GarageB, GoldInsure, InsureAll, and InsurePlus be δ_A, δ_B, δ_G, δ_{I_1}, and δ_{I_2}, respectively. Also assume that the price quotes p of GarageA are always greater than limit and that the price quote of GarageB is always less than limit. Now, the overall orchestration response time is $\delta_O = max(\delta_A, \delta_B) + max(\delta_{I_1}, \delta_{I_2})$, assuming that both δ_A and δ_B are less than the timeout value T.

Suppose that the performance of GarageB now deteriorates, and it does not respond before timeout time T. GarageA's price quote is now the best quote. Since we assumed that the quotes of GarageA are always greater than limit, GoldInsure is called and the orchestration's latency is $\delta_{O'} = T + \delta_G$. In the case when $\delta_G \ll max(\delta_{I_1}, \delta_{I_2})$, it is possible that $\delta_{O'} < \delta_O$. In other words, the deterioration of the performance of GarageB, could lead to an improvement in the performance of the orchestration.

Such a pathological situation does not occur in our original example since the response time of GetInsur depends only on the external parameter car. Once car is fixed, response times behave in a monotonic way. Thus, our example is *monotonic*.

Of course, it may not be considered fair to compare the different situations on the only basis of time performance, since they do not return the same data. A call always immediately returning "*nothing found*" will have the best timing performance, but is clearly not satisfactory from the user's viewpoint.

Further results regarding monotonicity can be found in [9]. To conclude on this aspect, we believe that monotonicity should be considered from a broader perspective, taking into account both timing and other QoS parameters as well as data.

2.2.4 Orchestrations Face the Open World Paradigm

The actors affecting the QoS of a Web service orchestration are:

- the orchestration server,
- the Web services called by the orchestration, and
- the transport network infrastructure.

All of these actors contribute to the overall QoS characteristics of the orchestration. Therefore, to be able to offer QoS guarantees, the orchestration needs QoS data from the other two types of actors.

In the context of networks, QoS studies assume knowledge of end-to-end resources and traffic, and use these to predict or estimate end-to-end QoS [15]. This can, for example, be used for evaluating the end-to-end performance of streaming services, supported by a dedicated cross-domain VPN. The reason for being able to do this is that, once defined and deployed, the considered VPN has knowledge of its own resources and traffic, which is enough to evaluate the QoS offered to the considered streaming service.

For our case of Web services orchestrations, however, the situation is different:

- The orchestration has knowledge about the resources of its own server architecture. It knows the traffic it can support, and it can monitor and measure its own ongoing traffic at a given time.
- The resources and extra traffic for each called Web service are *not* known to the orchestration—other users of these sites belong to the "open world" and the orchestration just ignores their existence.
- The resources and extra traffic for the transport network infrastructure are *not* known to the orchestration—other traffic belongs to the "open world" and the orchestration just ignores it.

Due to the issues discussed above, traditional QoS techniques are not very appropriate when applied to the study of QoS in Web services orchestrations. Contracts have emerged as the adequate paradigm for QoS of orchestrations and, more generally, of composite Web services in open world contexts.

2.3 Conclusions Drawn from this Discussion

From the above analysis, the following conclusions emerge regarding how QoS studies should be performed for Web services orchestrations:

- To ensure consistency of QoS studies, we must only consider *monotonic* orchestrations; that is, orchestrations such that, if QoS of some called service improves, so does the orchestration itself. Conditions ensuring monotonicity are found in [9]. Our CarOnLine example is monotonic.
- Since, for general orchestrations, control flow may be data- and time-dependent, analytical techniques for performance studies—such as those typically used for networks [15]—do not apply. One may consider restricting ourselves to finite data types and discrete domains for real-time, but then the computational cost of evaluating the QoS of the orchestration in all configurations may become prohibitive. This is why we chose to rely on simulation techniques. Of course, such simulations must take into account both data and QoS aspects.
- Because of the "open world" paradigm, QoS evaluation cannot rely on a joint model of resources and traffic for the Web services called by the orchestrator. The contribution of each of the Web services called to the QoS of the orchestration must then be abstracted in some way. In our open world, this relies on a notion of *trust* between the partners (the orchestration on one hand, and the called services on the other), formalized as an SLA. An

SLA here is a contract about QoS, relating the orchestration to the services it calls. In this approach, the orchestration has no means to be sure that such an SLA is faithful. Therefore, runtime *monitoring* of such contracts for possible violation is needed.

As advocated in the introduction, we decided to work with soft *probabilistic* contracts. Then, for the above-mentioned reasons, we chose to resort to *Monte-Carlo simulations* to compose contracts and tune our monitoring algorithms. As this is a first study of this subject, we left aside the issue of implementing *efficient* Monte-Carlo simulations, e.g., by using importance sampling [29].

In the following sections, we shall study *contract composition,* i.e., how the orchestration's contract relates to the contracts established with the different called services, seen as subcontractors. Then, we shall study *contract monitoring,* i.e., the monitoring of subcontractors for possible QoS contract violation.

3 Contract Composition and the *TOrQuE* Tool

3.1 How to Establish Probabilistic Contracts and How to Compose Them

In general, the orchestration will establish contracts or SLAs with the Web services it is calling. For S, a called Web service, we call S a *subcontractor* in the sequel and the contract for the considered QoS parameter has the form of a *cumulative distribution function*

$$F_S(x) = \mathbb{P}(\delta_S \leq x), \tag{1}$$

where δ_S is the random QoS parameter (here, the response time) and x ranges over the domain of this QoS parameter (here, $\mathbf{R}_+$).[1]

Regarding transport, different approaches might be considered. In a first "agnostic" approach, the orchestration will not contract regarding transport. The reason is that the orchestration does not want to know the network domains it may traverse. If QoS information regarding the transport layer is still wanted, this can be coarsely estimated by sending "pings" to the considered site. In another approach, the orchestrator may want to contract with the network service provider (e.g., as part of Virtual Private Network guarantees of service), very much in the way contracts are established with called Web services.

Finally, some Web services, such as Google, may address huge sets of users and would therefore not enter in a negotiation process with any orchestration. The distribution of such sites can be estimated on the basis of measurements.

To summarize, in designing contracts with its own customers, the orchestration: 1) uses the contracts it has agreed upon with its subcontracting Web services, 2) may estimate QoS parameters for other Web services it is using, and 3) may estimate QoS parameters for transport.

1. In practice, F_S will be abstracted by either a finite set of *quantiles* $(F_S(x_1), \ldots, F_S(x_K)$ for a fixed family $x_1, \ldots, x_K$ of values for the QoS parameters) or a finite set of *percentiles* (e.g., the set of values $y_1, \ldots, y_9$ such that $F_S(y_1) = 10\%, \ldots, F_S(y_9) = 90\%$). Such contracts are easily expressible in terms of the WSLA standard [18].

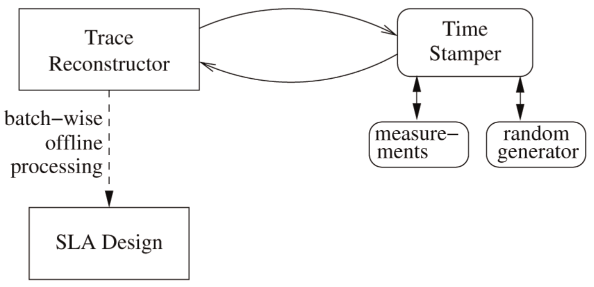

Fig. 3. Overall architecture of the *TOrQuE* tool.

Based on this approach, we have developed the following Monte-Carlo procedure for QoS contract composition. This procedure is applied at design time:

- Contracts with the called sites have the form of probability distributions for the considered QoS parameters. From these, we draw successive outcomes for the tuples:

  $$\{\text{response to queries, associated QoS parameters}\}.$$

  If no contract is available for a given site, we replace the missing probability distribution by empirical estimates of it, based on QoS measurements.
- Using a partial order execution model for the orchestration, we run Monte-Carlo simulations of the orchestration involving independent successive trials for the random latencies, thus deriving empirical estimates for the global QoS parameters of the orchestration.
- Having these empirical estimates, we can properly select quantiles defining soft contracts for the end user.

3.2 The *TOrQuE* Tool

The *TOrQuE* (***T**ool for **Or**chestration simulation and **Qu**ality of service **E**valuation*) tool implements the above methodology. Its overall architecture is shown in Fig. 3. The steps involved in the QoS evaluation and the *TOrQuE* modules that perform them are commented on next.

3.2.1 The Orchestration Model

To ease the development of this tool, we decided to replace the (complex) BPEL standard for specifying Web services orchestrations by a lightweight formalism called ORC [23]. The authors of this formalism have developed a tool [13] which can animate orchestrations specified in ORC.

3.2.2 Getting QoS-Enhanced Partial Order Models of Executions

This is performed by the "Trace Reconstructor" module. Jointly with the authors of ORC, we have developed an alternative mathematical semantics for ORC in terms of *event structures* [28]. Event structures [5] provide the adequate paradigm for deriving partial order models of ORC executions, in which causality and concurrency relationships between the different events of the orchestration is made explicit. Partially ordered executions can be

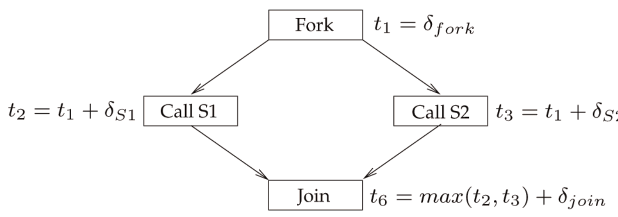
Fork
$t_1 = \delta_{fork}$
$t_2 = t_1 + \delta_{S1}$
Call S1
Call S2
$t_3 = t_1 + \delta_{S2}$
Join
$t_6 = max(t_2, t_3) + \delta_{join}$

Fig. 4. Deriving response time for a fork-join pattern. The "Fork" and "Join" are the branching and synchronization events and $S1$ and $S2$ are two Web services called in parallel. δ_a denotes the time taken for event a to execute.

tagged with QoS parameters which can then be composed. For example, Fig. 4 shows how the response time of a fork-join pattern is computed from that of its individual events. These max-plus rules are used to combine delays in the partial order. The QoS parameter tagging of the partial-ordered executions and their composition is implemented in *TOrQuE*'s *trace reconstructor* module (see Fig. 3). Arbitrary patterns encountered in ORC specifications can be handled by this module.

Fig. 5 shows a diagram of the event structure corresponding to the CarOnLine program written in ORC. The event structure is generated by our tool and it collects all the possible executions of CarOnLine, taking into account timers and other interactions between data and control. Each execution has the form of a partial order and can be analyzed to derive appropriate QoS parameter composition for each occurring pattern. Each site call to a service M is translated into three events, the *call* (M), the *call return* ($?M$) and the *publish action* (!), which adds to the length of the structure. For more information regarding these event structures, the reader is referred to [28].

3.2.3 Drawing at Random, Samples of QoS Parameters for the Called Sites

This is performed by the "Time Stamper" module. To perform Monte-Carlo simulations using the Trace Reconstructor, we need to feed it with actual values for the QoS parameters. For the called sites, these values should be representative of the contracts established between them and the orchestration. This is achieved by drawing such parameters at random from the probability distribution specified in each contract.

If no contract is available with a given site, the needed probability distribution may alternatively be estimated from measurements. For example, calling the considered site a certain number of times and recording the response times provides an empirical distribution that can be resampled by simple bootstrapping techniques [14]. The Time Stamper module supports both techniques: sampling from contract's probability distribution or bootstrapping measured values.

3.2.4 Exploiting Results from Monte-Carlo Simulations to Set Contracts for the Orchestration

This is performed by the "SLA Design Unit," which is mainly a GUI module that displays simulation logs and

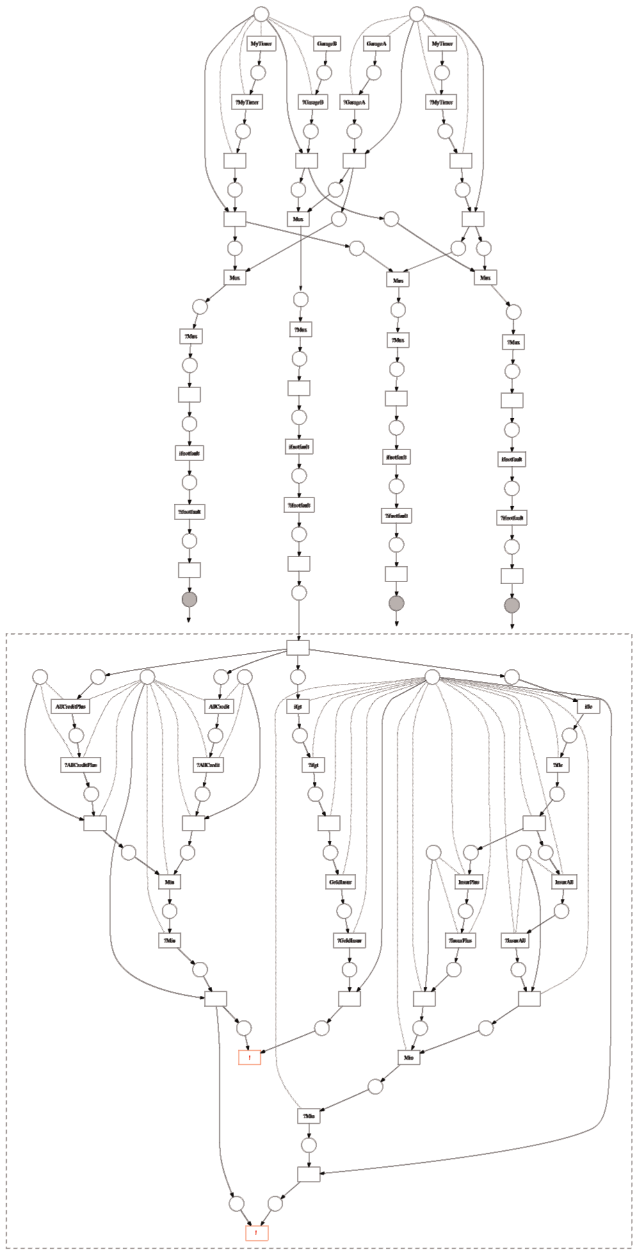

Fig. 5. A labeled event structure collecting all possible executions of CarOnLine, as generated by our tool. The three dangling arcs from the shaded places are followed by copies of the boxed net. The aim of the figure is to show the partial order structure. Zooming-in the electronic version reveals the detailed labels of the transitions as generated from the detailed ORC specification.

histograms or empirical distributions of the QoS parameters and allows selecting appropriate quantiles.

3.3 Discussion on Criticality

At a first sight, not all sites in an orchestration have an equal impact on the QoS of the orchestration. Some sites may be *critical*, in that a slight degradation/improvement in their performance will directly result in a degradation/improvement in the performance of the overall orchestration. Other sites may not be critical and a degradation in their performance would not affect the performance of the orchestration very much.

To address this in the context of classical timing performance studies, e.g., for scheduling purposes, the notion of *critical path* was proposed. However, this notion must be revisited under our probabilistic approach.

For instance, consider the example of Fig. 4, we have $t_6 = t_1 + \max(\delta_{S_1}, \delta_{S_2}) + \delta_{\text{join}}$, so it seems that only the "slowest" among the two sites, S_1 and S_2, matter. This intuition is wrong, however. Assume that the two sites S_1 and S_2 behave independently from the probabilistic point of view. Setting $\delta = \max(\delta_{S_1}, \delta_{S_2})$, $F_i(x) = \mathbb{P}(\delta_{S_i} \leq x)$, and $F(x) = \mathbb{P}(\delta \leq x)$, we have $F(x) = F_1(x) \times F_2(x)$. Next, suppose that the two sites S_1 and S_2 possess unbounded response times. Thus, for any $x > 0$, we have $0 < F_i(x) < 1$ for $i = 1, 2$. In this case, since $F(x) = F_1(x) \times F_2(x)$, any change in F_1 or F_2 will result in a change in F. Thus, both sites S_1 and S_2 are equally critical, even if, say, $F_1(x) > F_2(x)$ for every x, meaning that there is a good chance that S_1 will respond faster. Of course, if F_1 and F_2 possess disjoint supports, meaning that there exists some separating value x_o such that $F_2(x_o) = 0$ but $F_1(x_o) = 1$, then we know that $\delta_{S_1} < \delta_{S_2}$ will hold with probability 1, so that S_1 is never on the critical path.

This discussion justifies that all subcontractors are individually monitored for possible contract violation, as they all have an impact on the overall orchestration QoS in general—see Section 5 regarding monitoring.

4 Experimental Results for Contract Composition: Opportunities for Overbooking

In this section, we report the results obtained on the composition of contracts from the simulations of the *TOrQuE* tool. The results show the possibilities for overbooking and validate our approach of using probabilistic contracts.

In orchestrations, exceptions and their handling are frequently part of the orchestration specification itself. In addition, collecting measurement data from existing Web services regarding this type of parameter is difficult (actually, in our experiments, no exceptions were observed). For these two reasons, we did not include exceptions in our simulation study.

4.1 Approach

4.1.1 Probabilistic Contracts for the Sites

The sites in the CarOnLine example were not implemented as real services over the Internet. In order to assign realistic delay behavior to these sites during the simulations, we associated their behavior to that of actual Web services over the Internet. For this, we measured response times of calls to these actual Web services. The response time recorded were used in a bootstrap mode and also to fit distributions which would be sampled during simulations.

We considered six different Web services for this purpose [35]: `StockQuote`, which returns stock prices for a queried enterprise, `USWeather`, which gives the weather forecast of a queried city for a week from the day of the call, `CongressMember`, which returns the list of the members of the US Congress, `Bushism`, which returns a random quote of George W. Bush, `Caribbean`, which returns information related to tourism in the Caribbean, and `XMethods`, which queries a database of existing Web services over the Web. We made 20,000 calls to each of these six Web services and measured their response times. The calls were made in sequence, a new call being made as soon as the previous call responded. We could roughly categorize these services into three categories based on their response times:

- **Fast.** The service *Caribbean* with response times in the range 60-100 ms or the *CongressMember* service with response times between 300-500 ms.
- **Slow.** Service *StockQuote* which responded typically between 2 and 8 seconds.
- **Moderate.** The services like *USWeather*, *XMethods*, and *Bushism*, with response times in the 800-2000 ms range.

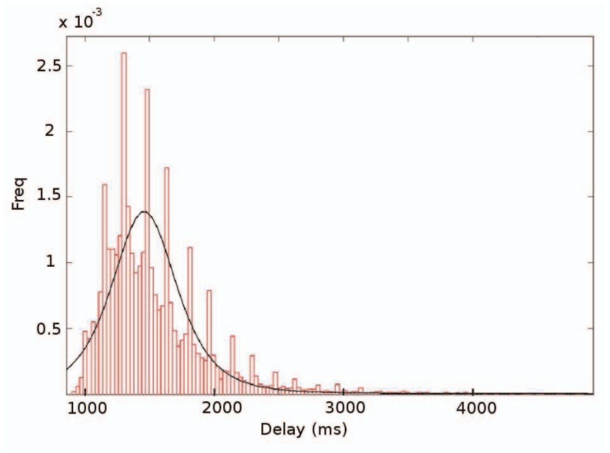

Fig. 6. Fitting of a T Location-scale distribution on the plot of 20,000 measured delays of the service *USWeather*.

4.1.2 Fitting Distributions on Measured Data

To validate the use of certain families of distributions, we performed their best fit on the measured data. When applied to the measured response times of the six different Web services, we observed that T location-scale distributions served as good approximations in most cases. Moreover, Gamma and the Log-Logistic distributions [20] were also reasonably good fits for the response times. Fig. 6 shows the results of the fit of a T Location-Scale distribution on the response times of the service `USWeather`.

While the quality of fit is reasonably good, this point is not central in our study. We only see the use of certain families of distributions as an alternative to bootstrap techniques when measurements are not available. In general, however, we prefer using bootstrapping techniques.

4.1.3 Orchestration Engine Overhead

The events of an orchestration could be seen as one of these two types: 1) the service call events which are calls to external sites or 2) the events internal to the orchestration, implementing the processing and coordination actions of the orchestration. Depending on the relative cost (in terms

TABLE 2
Response Time Associations for Sites in CarOnLine

Site	Service
GarageA	USWeather
GarageB	Bushism
AllCredit	XMethods
AllCreditPlus	StockQuote
GoldInsure	Caribbean
InsureAll	CongressMembers
InsurePlus	CongressMembers

of execution time) of these events, the following scenarios can be considered:

- **Zero delay.** The delay due to the internal events is zero (or negligible) when compared to that of the site calls. The overall delay of the orchestration would depend solely on the response times of the services it calls in this case.
- **Nonzero delay.** The delays of the internal events in this case are nonzero, comparable to the delays of site calls.

Since the performance of our prototype cannot be regarded as representative of that of a real orchestration engine, we considered only the first scenario.

4.2 Simulation Results

All the measurements and simulations were performed on a 2 GHz Pentium dual core processor with 2 Gb RAM. We consider two cases of simulations, depending on the timeout value T for the calls to the garages (see site $Timer(T)$ in Table 1): 1) No timeout (equally, T is infinite) and 2) T is a finite value, which is lesser than the maximum response time of a garage.

4.2.1 Case 1: No Timeouts

Based on the way delays of site calls are generated, we performed two types of simulations: those in which delay generation is done by 1) bootstrapping measured values and 2) sampling a T location-scale distribution, previously fit to measured data.

Bootstrap-Based Simulations. In these simulations, we associated each service in the CarOnLine example with delay behaviors of one of the six Web services mentioned previously. The associations are shown in Table 2 and the cumulative distribution functions of the observed response times for each of the called services are shown in Fig. 7. During any run of CarOnLine, the response time of a call is picked uniformly from the set of 20,000 delay values of its associated site. Since the response times of these services were measured from the client's side, they include the network's delay too. So, we do not consider the explicit delays modeled by the sites $NetGA$, $NetGB$, $NetC$, and $NetCP$, and give them zero delay each (if the contracts modeled only the performance from the server's perspective, without accounting for the network, we could give delays to each of these sites according pings done to the Web services).

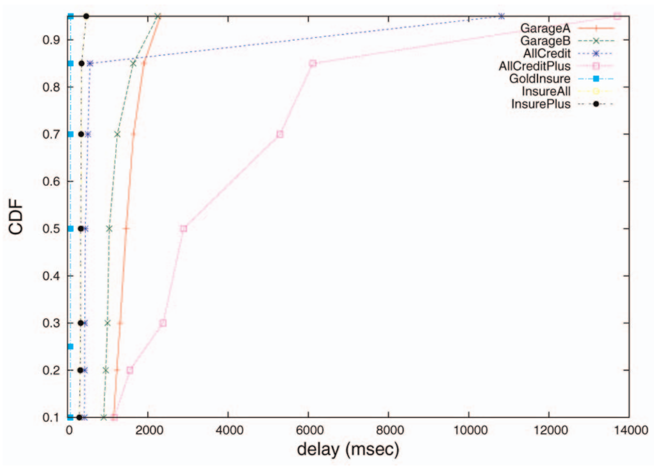

Fig. 7. The cumulative distribution function for the measured delays of the six Web services.

Results Using Hard Contracts. Consider the following "hard contract" policy, which is close to current state of practice. Contracts have the form of a certain quantile, e.g., "the response time shall not exceed x ms in y percent of the cases."

More precisely, let contracts of the orchestration with a site be of the form

$$\mathbb{P}(\delta_i \leq K_i) \geq p_i, \tag{2}$$

where $i = 1, \ldots, m$ ranges over the sites involved in the orchestration, δ_i is the response time of site i, K_i is the promised bound of site i, and p_i is the corresponding probability (so that $\delta_i \leq K_i$ holds in y percent of the cases, where $y = 100 \times p_i$). Assuming the called sites to be probabilistically independent, what the orchestration can guarantee to its clients is

$$\mathbb{P}(\delta \leq K) \geq \prod_{i=i}^{m} p_i, \tag{3}$$

where δ is the response time of the orchestration and K is the max-plus combination the K_is, according to the orchestration's partial ordering of call events.

By setting the delay contracts (maximum delay values) of each of the sites involved in CarOnLine to their 99.2 percent quantile values, we get the end-to-end orchestration delay bound to be $K = 44{,}243$ ms, which can be guaranteed for 94.53 percent of the cases.

Results Using Probabilistic Soft Contracts. We now compare the above results with our approach using probabilistic contracts. To this end, we performed 100,000 runs of the orchestration in the bootstrap mode. The empirical distribution of end-to-end delays of the orchestration is shown in Fig. 8. The minimum delay observed in this case is 1,511 ms and the maximum is 369,559 ms. The 94.53 percent delay quantile of this distribution is 23,189 ms, to be compared with the more pessimistic value of 44,243 ms that we obtained using the usual approach.

T Location-Scale Sampling-Based Simulations. In this mode of simulation, T location-scale distributions are sampled to generate delay values for site calls. The delay values of the six Web services were fitted with a

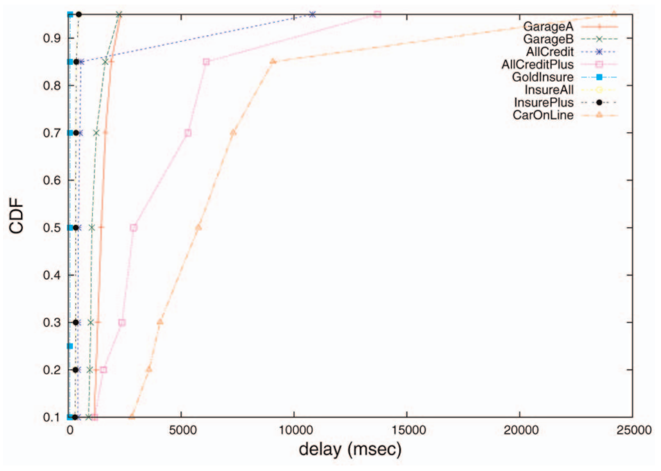

Fig. 8. Empirical distribution of end-to-end orchestration delays for 10,0000 simulations in the bootstrapping case.

T location-scale distribution, giving the estimated μ, σ, and ν parameters of the distribution. The pdf for this distribution is:

$$p(x) = \frac{\Gamma(\frac{\nu+1}{2})}{\sigma\sqrt{\nu\pi}\Gamma(\frac{\nu}{2})}\left[\frac{\nu + \left(\frac{x-\mu}{2}\right)^2}{\nu}\right]^{-\left(\frac{\nu+1}{2}\right)}.$$

The association of sites of CarOnLine and the Web services remain unchanged, as given in Table 2. The parameter ν for the fitted T location-scale distribution for each of the sites is given in Table 3.

Results Using Hard Contracts. On setting the delay contracts of each of the sites to their 99.2 percent quantile values, we get the end-to-end orchestration delay bound to be $K = 1469,539$ ms, which can be guaranteed for 94.53 percent of cases.

Results Using Probabilistic Soft Contracts. As before, we assume zero delay for all the internal orchestration actions and perform 100,000 runs of the orchestration in this configuration. End to end orchestration delays from the simulations were recorded. In this case, the 94.53 percent quantile is found to be 14,658 ms.

The results are summarized in Table 4.

The time taken for the 100,000 simulations in the bootstrap mode was 37.74 seconds and in the T location-sampling mode was 42.13 seconds.

4.2.2 *Case 2: Finite Timeouts*

Using hard contracts in orchestrations having timeouts raises difficulties. As an illustration, consider again Fig. 4.

TABLE 4
No Timeout Case: Comparison of Delay Quantiles

Mode	Soft contract 94.53% quantile	Hard contract 94.53% quantile
BootStrap	23,189	44,243
T Location Dist	14,658	1,469,539

Let K_1 and K_2 be the two hard bounds (in ms) for response times in the contracts of sites $S1$ and $S2$, respectively. Assume that timers are used to guard the two site calls, with timeout occurring at λ ms. Then, clearly, the contract that results for this orchestration entirely depends on the relative position of λ, K_1, and K_2. If $\lambda > K_i$ for $i = 1, 2$, then a timeout is supposed to never occur (unless one of the site contracts is violated). On the other hand, if $\lambda < K_i$ for $i = 1, 2$, then, even if the sites respect their contracts, this may at times be seen by the orchestration as a timeout. Clearly, using timers in combination with hard contracts makes little sense.

In contrast, probabilistic soft contracts allow using timers with no contradiction. The reason is that Monte-Carlo simulations have no problem simulating timers and their effect on the distribution of the orchestration response time. As a consequence, we only present the results from our simulations without a comparison to the hard-contract-based composition.

We again perform simulations in two modes: Bootstrap and T location-scale-based simulations.

Bootstrap-Based Simulations. As before, we associated each service in the CarOnLine example with delay behaviors of one of the six Web services measured. The associations are the same as before, given in Table 2. We now have timeouts for the calls to sites GarageA and GarageB. The 99.2 percent delay quantiles for these two sites are 3,304 msec and 4,183 msec, respectively. We perform simulations with different timeout values: 3,000, 4,000, and 5,000 msec. The results are given in Table 5.

T-Location Scale Sampling-Based Simulations. We maintain the associations of Table 2 and perform simulations by sampling the fitted T location-scale distributions. The results of these simulations are summarized in Table 5. The average time for 100,000 simulations in the bootstrap mode was 34.29 seconds and, in the T location-sampling mode, was 43.75 seconds.

TABLE 3
Parameter ν of the Fitted T Location Distributions

Site	Param ν	Site	Param ν
GarageA	2.678	GoldInsure	1.338
GarageB	0.835	InsureAll	0.835
AllCredit	0.297	InsurePlus	0.835
AllCreditPlus	2.258		

TABLE 5
Finite Timeout Case: Delay Quantiles

Mode	Soft contract 94.53% quantile	Timeout Value T
BootStrap	23,040	3,000
BootStrap	22,681	4,000
BootStrap	22,834	5,000
T Location Dist	13,258	3,000
T Location Dist	13,364	4,000
T Location Dist	13,582	5,000

5 Monitoring

In this section, we describe our technique for monitoring soft contracts. We show how monitoring is done for any contracted service when it is called by the orchestration.

We want to compare the observed performance of a service S to that promised in its soft contract F_S. Recall that the soft contract F_S is a distribution on the response times of S: $F_S(x) = \mathbb{P}(\delta_S \leq x)$. We denote by G_S the *actual* distribution function of S. We say that *contract F_S is met* if

$$\forall x, G_S(x) \geq F_S(x) \tag{4}$$

holds. Condition (4) expresses that the response time of S is *stochastically smaller* than the promise F_S [2]. Now, we want to perform *online* monitoring, meaning that we want to detect as soon as possible if S starts breaching its contract. To this end, denote by $G_{S,t}$ the actual distribution function of site S at time t. We want to detect as quickly as possible when condition (4) gets violated by S, that is, to set a red light at the first time t when the following condition occurs:

$$\sup_x (F_S(x) - G_{S,t}(x)) > 0, \tag{5}$$

which is the negation of (4).

Unfortunately, the orchestrator does not know $G_{S,t}$; it only can estimate it by observing S. To this end, let Δ_t be a finite set of sample response times of S, collected up to time t, we call it a *population*. For a while, we remove subscript $_t$ for notational convenience. For a set X, let $|X|$ denote its cardinality. Then,

$$\widehat{G}_{S,\Delta}(x) =_{\text{def}} \frac{|\{\,\delta \mid \delta \in \Delta \text{ and } \delta \leq x\}|}{|\Delta|} \tag{6}$$

is the *empirical distribution function*, defined as the proportion of sample response times less than x among population Δ. Then, as a first sight, the contract is violated when

$$\sup_{x \in \mathbf{R}_+} (F_S(x) - \widehat{G}_{S,\Delta}(x)) \tag{7}$$

occurs. The problem with (7) is that $\widehat{G}_{S,\Delta}(x)$ can randomly fluctuate around $F_S(x)$, especially when $|\Delta|$ is small. A solution to this problem is to have a *tolerance zone* for such deviations.

Our online monitoring procedure is then as follows: Decide that site S violated its contract at the first time t (if any) when

$$\sup_{x \in \mathbf{R}_+} (F_S(x) - \widehat{G}_{S,\Delta_t}(x)) \geq \lambda \tag{8}$$

occurs, where λ is a small positive parameter which defines the tolerance zone. Reducing λ improves the chances of detecting contract violation earlier (it reduces the *detection delay*), but it also increases the risk of a false alarm (it increases the *false alarm rate*), see [6]. Thus, tolerance parameter λ has to be tuned in a meaningful way. This is done in an offline "calibration phase," performed prior to the monitoring.

5.1 Calibration Phase

As explained in Section 3, during contract composition, sample response times are drawn from the contract distribution $F_S(x)$ for each service S involved in the orchestration. Suppose that the total number of samples drawn for a given service S is M, i.e., the set of sampled delay values for S during the simulation is $\boldsymbol{\Delta} = \{\delta_1, \dots \delta_M\}$. In the calibration phase, we apply the following *bootstrapping* method [14]:

1. Generate Δ^* by resampling $\boldsymbol{\Delta}$ at random. This means that Δ^* is a randomly selected subset of $\boldsymbol{\Delta}$ of fixed size $|\Delta^*| = N$. According to the bootstrapping discipline, N should be smaller than $log(M)$. Using Δ^*, we can produce a bootstrap estimate $\widehat{G}_{S,\Delta^*}(x)$ of $F_S(x)$ using (6). Denote by Ω the set of such randomly generated $\Delta^* \subseteq \boldsymbol{\Delta}$. In our experiments, we have chosen its cardinality $|\Omega|$ to be about 10,000.
2. A *false alarm* level L (e.g., 5 percent) during monitoring is agreed between the orchestrator and the service S. Taking $\widehat{G}_{S,\Delta^*}(x)$ as a population, where Δ^* ranges over Ω, the tolerance parameter λ is tuned to the smallest value such that

$$\sup_{x \in X} (F_S(x) - \widehat{G}_{S,\Delta^*}(x)) \leq \lambda$$

holds for $100 - L$ percent (e.g., 95 percent) of the $\Delta^* \in \Omega$.

In fact, it is a result due to Kolmogorov [20, Section 14.2], that, for N large enough, the so obtained value for the tolerance zone λ does not depend on the distribution F_S. Yet, to avoid dealing with size issues of N, we prefer calibrating tolerance parameters for each site individually. But, clearly, there is room for saving computations at this step.

5.2 Monitoring Phase

Once the tolerance parameter λ is set, monitoring can be done in the following way: suppose the first N responses of service S have latencies $\{\delta_1, \dots \delta_N\}$. Taking $\Delta = \{\delta_1 \dots \delta_N\}$, we compute $\widehat{G}_{S,\Delta}(x)$ and then check if (8) is violated. When the $(N+1)$st delay δ_{N+1} is recorded, we *shift* Δ by one observation, making it $\{\delta_2, \dots \delta_{N+1}\}$. We compute $\widehat{G}_{S,\Delta}(x)$ for this new Δ and check the violation of (8) again. This process is repeated for further observed response times, each time shifting Δ by one observation.[2] So, Δ is a sliding window of fixed size N. The window size N is the same as the size $|\Delta^*|$ in the calibration phase.

Window length N appears as an additional design parameter for the monitoring procedure. N can be entirely decided by the orchestrator and does not need to be a part of the contract. The rationale for tuning N is as follows: Observe that N is strongly correlated with the detection delay in case of a contract violation. On the one hand, the proportion of breaching data must be large enough in the window Δ in order for (8) to get violated. Thus, reducing N contributes to the reduction of detection delay. On the other hand, reducing N increases random fluctuations of $\widehat{G}_{S,\Delta^*}(x)$ when Δ^* ranges over Ω, thus resulting in the need for increasing tolerance parameter λ to maintain the agreed

2. Actually, we do not need to shift the window by 1; any fixed amount can be used instead, provided that successive windows overlap.

TABLE 6
Contract and Experimental Distributions of GarageA

Contract Delay Quantile (msec)	CDF	Experimental Delay Quantile (msec)
1149	0.1	1199
1229	0.2	1279
1310	0.3	1360
1462	0.5	1520
1645	0.7	1745
1905	0.85	2005
2312	0.95	2412

false alarm rate, which in turn increases the delay for detecting violations. This results in a trade-off leading to an optimal choice for N. Anyway, this need not be part of the agreed contract.

6 Experimental Results: Monitoring

We now describe the implementation of our monitoring technique and the results obtained. We first discuss the kind of soft contracts we use in the simulations. After this, we present results on the monitoring on contracts, as explained in Section 5.

6.1 Contract of the Orchestration

We take the contract of a service S, F_S to be a probability distribution of the response time. Expecting a service provider to able to give a precise probability for *every possible* value of latency is however impractical. So, we take the contract with provider S to be a set of quantiles of latencies $\{x_1 \dots x_k\}$ with the corresponding probabilities $\{F_S(x_1) \dots F_S(x_k)\}$. Hard contracts are just a special case of our soft contracts in which only one such quantile exists. We thus require the provider to pass from promising a performance probability of one quantile to multiple quantiles.

During simulation, two possibilities may be considered when using $F_S = \{F_S(x_1) \dots F_S(x_k)\}$ for sampling response times:

- Use F_S as it is, by sampling each time one of the quantiles $\{x_1 \dots x_k\}$, in proportion with F_S. This would lead to overly pessimistic distributions, however.
- Hypothesize a constant probability density within each quantile, except for the last one where exponential distribution is hypothesized. From our experiments regarding Web service response times, we preferred this second approach.

While monitoring, we check for violation of (8) only for the set of quantiles that have been promised by the service S in its contract F_S. The set of positive reals $\mathbf{R}$ in (8) is thus replaced by the set $X = \{x_1 \dots x_k\}$ of latency quantiles promised in the contract.

6.2 Results

We ran CarOnline orchestration and monitored the single service GarageA in isolation, according to Section 5. We only show the monitoring of one service since the process of monitoring is identical for any other service of the orchestration. There was no particular reason for choosing to monitor GarageA, since we could have done the same with any other service of CarOnline. The delay behaviors associated with each of CarOnLine remains the same as in Section 4.2, given by Table 2. The contract of GarageA, the finite set of quantiles and their corresponding probabilities, is given in the first and second column of Table 6, respectively. These values were derived from the measured response times of the USWeather service. The false alarm rate agreed with the orchestrator is 95 percent.

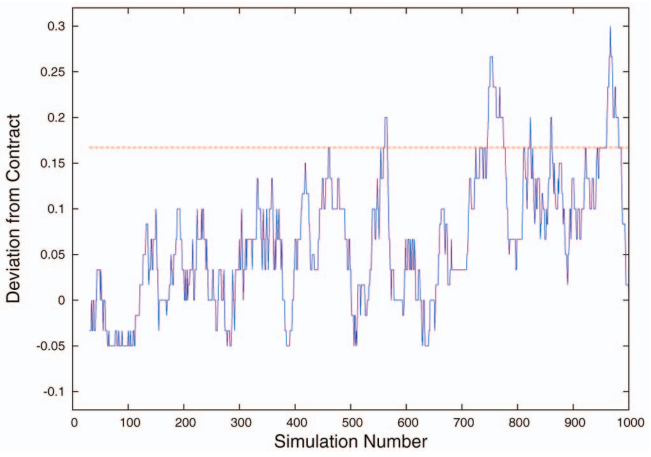

Fig. 9. Monitoring of GarageA. The plot shows the deviation from its contract for each run of the simulation. This deviation is $\sup_{x \in X}(F_S(x) - \widehat{G}_{S,\Delta^*}(x))$.

As mentioned in the end of Section 5, we need to find a good value for the window length N for the calibration and the monitoring phase (it directly affects the detection delay). For this, we ran the calibration and monitoring on GarageA for three different window lengths: 10, 30, and 50. The violations were detected after 10 to 25 calls (with lots of variations) when $N = 10$, 20 to 30 calls when $N = 30$, and between 40 to 80 calls when $N = 50$. $N = 30$ was preferred to $N = 10$ because fewer variations were observed in the detection delay, and is clearly preferred over $N = 50$, where the detection delay was too large.

With $N = 30$, the calibration phase (5.1) on this distribution of GarageA gave the tolerance parameter λ equal to 0.167. After the calibration phase, the CarOnline orchestration was run 1,000 times as follows: From run 1 to 700, GarageA's actual performance was exactly the same as the promised distribution. From run 700 to 1,000, we slightly deteriorated GarageA's performance to follow a "slower" distribution. The delay quantiles and their corresponding probabilities of this slower distribution are given in the third and second column of Table 6, respectively.

The result of the monitoring is shown in Fig. 9. The value of $\sup_{x \in X}(F_S(x) - \widehat{G}_{S,\Delta^*}(x))$ is plotted for each call made to GarageA. The horizontal line shows the value of λ, 0.167. The detection occurs around the 747th run, i.e., around 47 calls later.

The test statistics used in Fig. 9 behave in a quite noisy way. This suggests that the ratio of false alarm rate versus detection delay may not be optimal. Monitoring procedure (8) could be improved in many respects, however, using the huge background of sequential and nonparametric statistics [6]. First, empirical estimate (6) for the distribution function G_S of S could be improved by using (possibly adaptive) *kernel* estimators. Second, instead of relying on an estimate based on a sliding window, truly sequential estimates could be used. We have, however, decided to keep basic in the techniques we used from statistics for two reasons: They are easily understandable by nonspecialists and they are robust and easy to tune.

7 Related Work

Proposals for such QoS-based compositions are few and no well-accepted standard exists to date. Menascé [22] discusses QoS issues in Web services, introducing the response times, availability, security, and throughput as QoS parameters. He also talks about the need of having SLAs and monitoring them for violations. He does not however, advocate a specific model to capture the QoS behavior of a service, or a composition approach to compose SLAs.

Aggarwal et al. [1] view QoS-based composition as a constraint satisfaction/optimization problem in the METEOR-S project. Services have selection criteria, which are constraints, for which an optimal solution is found using integer linear programming. Cardoso et al. [11] aim to derive QoS parameters for a workflow, given the QoS parameters of its component tasks. Using a graph reduction technique, they repeatedly rewrite the workflow, merging different component tasks and also their QoS attributes according to different rules.

Zeng et al. [36] use Statecharts to model composite services. An orchestration is taken to be a finite execution path. For each task of the orchestration, a service is selected from a pool of candidate services using linear programming techniques such that it optimizes a specific global QoS criteria. In [24], the authors propose using fuzzy distributed constraint satisfaction programming (CSP) techniques for finding the optimal composite service.

Canfora et al. [10] use genetic algorithms for deriving optimal QoS compositions. They use techniques similar to [11] for modeling QoS of services. Compared to the linear programming method of Cardoso et al. [11], the genetic algorithm is typically slower on small to moderate size applications, but is more scalable, outperforming linear programming techniques when the number of candidate services increase.

A distinguishing feature of our proposal from the above composition techniques is that we do not consider the QoS parameters of a service to be fixed, hard bound values. We believe that, in reality, these parameters exhibit significant variations in their values and are better modeled by a probability distribution. This alternative approach has two advantages: First, it reduces pessimism in contract composition, as we shall see. And, second, it allows for "soft" monitoring of contract breaching (having a delayed response once upon a time should not be seen as a breaching).

In [12], the authors use WSFL (Web Service Flow Language)—a language proposed by IBM to model Web service compositions—and enhance it with the capability to specify QoS attributes. These are then translated into a simulation model in Java (JSim), which can then be simulated for performance analysis. The fundamental difference from our approach is that the approach assumes a "nonopen world" scenario, assuming that the services of the orchestration can be instrumented with measurement code to get information about its performance. This information then seems to be used in queuing-based models, to generate queuing and service execution time during simulations. The authors, however, do not give any detailed information about the models and the associated parameters used in the simulations. This approach also requires the orchestrator to be able to control all the load on the external service too, which is often an unrealistic approach.

Web service Performance Analysis Center (sPAC) [30] is another similar approach for the performance evaluation of services and their compositions. The authors use UML diagrams to model a service composition, which is then translated into a simulation model in Java (SimJava). sPAC also generates code to call the services in the composition under a light load to record the performance of the services. The performance statistics collected are then used in the simulation mode to model the performance of the services. However, as in [12], sPAC also assumes that the services whose performance it evaluates can be instrumented to collect the performance statistics for use in simulations.

The notion of probabilistic QoS has been introduced and developed in [16], [17] with the ambition to compute an exact formula for the composed QoS, which is only possible for restricted forms of orchestrations without any data dependency. We propose using simulation techniques to analyze the QoS of a composite service. This allows us to use nontrivial distributions as models for performance and also permits the analysis of orchestrations whose control flows have data and time-related dependencies.

Most of the work in QoS monitoring is dedicated to the design of service monitoring architectures [37]. Service monitoring needs to be integrated in the infrastructure at large in order to enable the detection and routing of the service operational events. We have proposed a framework for probabilistic contracts and shown how they can be composed. For runtime monitoring, this leads directly to the use of statistical testing techniques to detect violation of QoS contracts. Such techniques have already been used in [7] to adapt SLA checkers to the variation of the environment, but in a context of deterministic contracts.

8 Conclusion

We have studied soft *probabilistic* contracts, their composition, and their monitoring for Web services orchestrations. Probabilistic soft contracts have a number of advantages: they compose easily, as shown by our Monte-Carlo-based dimensioning tool *TOrQuE*; they provide the opportunity for well sound overbooking, thus avoiding pessimistic contracts; and they allow handling timers as part of the orchestration, a frequent and desirable practice. We stress that our *TOrQuE* tool can indeed be used for the dimensioning of realistic orchestrations, as the cost of running a Monte-Carlo simulation for design space exploration is acceptable. We have also proposed a statistical approach to design monitors for services promising soft contracts for monotonic orchestrations. Our method requires prior calibration of the detection threshold in order to achieve an agreed false alarm rate.

We plan to extend our work in two directions. The first direction is the real deployment of the method on the Web, based on the ORC runtime environment. More precisely, we are currently working on QoS-enabled extensions of this language.

The second direction is to generalize what we have done on response times to other QoS parameters, addressing the fact that different QoS parameters are often correlated. Indeed, we believe that a large part of the techniques we have developed generalize to other QoS parameters (e.g., availability, reliability, security, and possibly quality of data). In particular, our abstract representation of runs of orchestrations as partial orders of events allows us to combine performance quanta in a flexible way.

Acknowledgments

This work was partially funded by the ANR national research program DOTS (ANR-06-SETI-003), DocFlow (ANR-06-MDCA-005), and the project CREATE ActivDoc.

References

[1] R. Aggarwal, K. Verma, J.A. Miller, and W. Milnor, "Constraint Driven Web Service Composition in METEOR-S," *Proc. IEEE Int'l Conf. Services Computing (SCC '04),* pp. 23-30, 2004.

[2] G Anderson, "Nonparametric Tests of Stochastic Dominance in Income Distributions," *Econometrica,* vol. 64, no. 5, pp. 1183-1193, 1996.

[3] T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, I. Thatte, D. Trickovic, and S. Weerawarana, "Business Process Execution Language for Web Services," version 1.1, OASIS, May 2003.

[4] J. Arias-Fisteus, L. Sánchez Fernández, and C. Delgado Kloos, "Applying Model Checking to BPEL4WS Business Collaborations," *Proc. Symp. Applied Computing (SAC '05),* pp. 826-830, 2005.

[5] P. Baldan, A. Corradini, and U. Montanari, "Contextual Petri Nets, Asymmetric Event Structures, and Processes," *Information and Computation,* vol. 171, no. 1, pp. 1-49, 2001.

[6] M. Basseville and I. Nikiforov, *Detection of Abrupt Changes—Theory and Application.* Prentice-Hall, Apr. 1993.

[7] A. Bertolino, G. De Angelis, A. Sabetta, and S. Elbaum, "Scaling Up SLA Monitoring in Pervasive Environments," *Proc. Int'l Workshop Eng. of Software Services for Pervasive Environments (ESSPE '07),* pp. 65-68, 2007.

[8] P. Bhoj, S. Singhal, and S. Chutani, "SLA Management in Federated Environments," *Computer Networks,* vol. 35, no. 1, pp. 5-24, 2001.

[9] A. Bouillard, S. Rosario, A. Benveniste, and S. Haar, "Monotony in Service Orchestrations," Technical Report 6658, Inria, Apr. 2008.

[10] G. Canfora, M. Di Penta, R. Esposito, and M. Luisa Villani, "An Approach for QoS-Aware Service Composition Based on Genetic Algorithms," *Proc. Genetic and Evolutionary Computation Conf. (GECCO '05),* pp. 1069-1075, 2005.

[11] J. Cardoso, A.P. Sheth, J.A. Miller, J. Arnold, and K. Kochut, "Quality of Service for Workflows and Web Service Processes," *J. Web Semantics,* vol. 1, no. 3, pp. 281-308, 2004.

[12] S. Chandrasekaran, G.A. Silver, J.A. Miller, J. Cardoso, and A.P. Sheth, "XML-Based Modeling and Simulation: Web Service Technologies and Their Synergy with Simulation," *Proc. Winter Simulation Conf.,* pp. 606-615, 2002.

[13] W. Cook and J. Misra, "Implementation Outline of Orc," internal report, http://orc.csres.utexas.edu/papers/OrcImpDraft.pdf, 2005.

[14] A.C. Davidson and D.V. Hinkley, *Bootstrap Methods and Their Application.* Cambridge Univ. Press, 1997.

[15] V. Firoiu, J.-Y. Le Boudec, D. Towsley, Z.-L. Zhang, and J.-Y. Le Boudec, "Theories and Models for Internet Quality of Service," *Proc. IEEE,* vol. 90, no. 9, pp. 1565-1591, 2002.

[16] S.-Y. Hwang, H. Wang, J. Srivastava, and R.A. Paul, "A Probabilistic QoS Model and Computation Framework for Web Services-Based Workflows," *Proc. Int'l Conf. Conceptual Modeling (ER '04),* pp. 596-609, 2004.

[17] S.-Y. Hwang, H. Wang, J. Tang, and J. Srivastava, "A Probabilistic Approach to Modeling and Estimating the QoS of Web-Services-Based Workflows," *J. Information Sciences,* vol. 177, no. 23, pp. 5484-5503, 2007.

[18] A. Keller and H. Ludwig, "The WSLA Framework: Specifying and Monitoring Service Level Agreements for Web Services," *J. Network and System Management,* vol. 11, no. 1, 2003.

[19] D. Kitchin, W.R. Cook, and J. Misra, "A Language for Task Orchestration and Its Semantic Properties," *Proc. Int'l Conf. Concurrency Theory (CONCUR '06),* 2006.

[20] E.L. Lehmann and J.P. Romano, *Testing Statistical Hypotheses.* Springer Verlag, 2005.

[21] Z. Liu, M.S. Squillante, and J.L. Wolf, "On Maximizing Service-Level-Agreement Profits," *Proc. ACM Conf. Electronic Commerce,* pp. 213-223, 2001.

[22] D.A. Menascé, "QoS Issues in Web Services," *IEEE Internet Computing,* vol. 6, no. 6, pp. 72-75, 2002.

[23] J. Misra and W.R. Cook, "Computation Orchestration: A Basis for Wide-Area Computing," *J. Software and Systems Modeling,* May 2006, doi:10.1007/s10270-006-0012-1.

[24] X. Thang Nguyen, R. Kowalczyk, and M. Tan Phan, "Modelling and Solving QoS Composition Problem Using Fuzzy DisCSP," *Proc. IEEE Int'l Conf. Web Services (ICWS '06),* pp. 55-62, 2006.

[25] C. Ouyang, E. Verbeek, W.M.P. van der Aalst, S. Breutel, M. Dumas, and A.H.M. ter Hofstede, "Formal Semantics and Analysis of Control Flow in WS-BPEL," *Science of Computer Programming,* vol. 67, nos. 2-3, pp. 162-198, 2007.

[26] The SWAN Project, http://swan.elibel.tm.fr, 2008.

[27] F. Puhlmann and M. Weske, "Using the *pi*-Calculus for Formalizing Workflow Patterns," *Business Process Management,* pp. 153-168, 2005.

[28] S. Rosario, D. Kitchin, A. Benveniste, W. Cook, S. Haar, and C. Jard, "Event Structure Semantics of Orc," *WS-FM,* pp. 154-168, 2007.

[29] P.J. Smith, M. Shafi, and H. Gao, "Quick Simulation: A Review of Importance Sampling Techniques in Communications Systems," *IEEE J. Selected Areas in Comm.,* vol. 15, no. 4, pp. 597-613, 1997.

[30] H.G. Song and K. Lee, "sPAC (Web Services Performance Analysis Center): Performance Analysis and Estimation Tool of Web Services," *Business Process Management,* pp. 109-119, 2005.

[31] W.M. P. van der Aalst, "Verification of Workflow Nets," *Proc. Int'l Conf. application and Theory of Petri Nets (ICATPN '97),* pp. 407-426, 1997.

[32] W.M.P. van der Aalst, B.F. van Dongen, J. Herbst, L. Maruster, G. Schimm, and A.J.M.M. Weijters, "Workflow Mining: A Survey of Issues and Approaches," *Data and Knowledge Eng.,* vol. 47, no. 2, pp. 237-267, 2003.

[33] W.M. P. van der Aalst and K.M. van Hee, *Workflow Management: Models, Methods, and Systems.* MIT Press, 2002.

[34] A. van Dijk, "Contracting Workflows and Protocol Patterns," *Business Process Management,* pp. 152-167, 2003.

[35] XMethods, http://www.xmethods.net, 2008.

[36] L. Zeng, B. Benatallah, A.H.H. Ngu, M. Dumas, J. Kalagnanam, and H. Chang, "QoS-Aware Middleware for Web Services Composition," *IEEE Trans. Software Eng.,* vol. 30, no. 5, pp. 311-327, May 2004.

[37] L. Zeng, H. Lei, and H. Chang, "Monitoring the QoS for Web Services," *Proc. Int'l Conf. Service Oriented Computing (ICSOC '07),* pp. 132-144, 2007.

Sidney Rosario received the BS degree in computer science in 2004 and the MS degree in 2006. He is currently a PhD candidate in the DISTRIBCOM team at INRIA Rennes under the supervision of Albert Benveniste. His research interests include the formal modeling of distributed systems and their performance evaluations.

Albert Benveniste graduated in 1971 from Ecole des Mines de Paris. He performed his These d'Etat in mathematics, probability theory, in 1975, under the supervision of Paul-André Meyer. From 1976 to 1979, he was an associate professor in mathematics at the Université de Rennes I. From 1979 to the present, he has been the director of research at INRIA. His previous working areas include system identification and change detection in signal processing and control (until 1995), vibration mechanics, reactive, real-time, and embedded systems design in computer science, and network and service management in telecommunications with an emphasis on distributed systems. His current interests also include component and contract-based design of embedded systems, real-time architectures, Quality of Service management of Web services, and distributed active XML documents. In 1980, he was a cowinner of the *IEEE Transactions on Automatic Control*'s Best Transaction Paper Award for his paper on blind deconvolution in data communications. In 1990, he received the CNRS silver medal and, in 1991, was elected a fellow of the IEEE.

Stefan Haar has a background in mathematics, where he specialized in stochastic processes, and Petri net theory, to which he converted in his PhD thesis. After receiving the doctorate degree from Hamburg University, Germany, and post-doctoral fellowships in Berlin and Nancy, he became a researcher with INRIA Rennes in 2001 and joined INRIA Saclay in 2008 after returning from a leave at the University of Ottawa and Alcatel-Lucent Bell Labs, Ottawa, Canada. His research interests include Petri nets, event structures and graph grammars, conformance testing, and probability.

Claude Jard graduated as an engineer in telecommunications in 1981. He received the PhD degree in computer science from the University of Rennes in 1984. He was the head of the protocol validation group at the French National Research Centre in Telecommunications (CNET) from 1981 to 1986. He was a research supervisor, and then a research director, at the French National Centre for Scientific Research (CNRS) starting in 1987. Currently, he is a professor in computer science at the Ecole Normale Supérieure de Cachan. He is the director of the research of the Brittany branch of this school. Professor Jard's research works relate to the formal analysis of asynchronous parallel systems. They lie within the general scope of using formal methods for programming distributed systems and relate to the stages of specification, verification, test, and supervision of distributed software on networks of processors. The central topic of his work is the study of dynamic methods of verification, in which verification is carried out during the execution of the—abstracted, simulated, or real—program to be analyzed. His current interests are the nonfunctional aspects, like time and QoS, in large-scale systems. He is the author or coauthor of more than 140 publications, carried out primarily within three research communities: theoretical computer science, protocol engineering, and distributed systems.