Negotiation Strategies for Probabilistic Contracts in Web Services Orchestrations

Ajay Kattepur, Albert Benveniste, Claude Jard

Negotiation Strategies for Probabilistic Contracts in Web Services Orchestrations

Ajay Kattepur, Albert Benveniste
Equipe DistribCom, IRISA/INRIA,
Campus Universitaire de Beaulieu,
Rennes, France.
Email: Firstname.Lastname@inria.fr

Claude Jard
ENS Cachan, Equipe DistribCom IRISA,
Université Européenne de Bretagne,
Bruz, France.
Email: Claude.Jard@bretagne.ens-cachan.fr

***Abstract*—Service Level Agreements (SLAs) have been proposed in the context of web services to maintain acceptable quality of service (QoS) performance. This is specially crucial for composite service orchestrations that invoke many atomic services to render functionality. A consequence of SLA management entails efficient negotiation protocols among orchestrations and invoked services. In composite services where data and QoS (modeled in a probabilistic setting) interact, it is difficult to select an atomic service for negotiation, in order to improve end-to-end QoS performance. A superior improvement in one negotiated domain (eg. latency) might mean deterioration in another domain (eg. cost); improvement in one of the invoked services may be annulled by another due to the control flow specified in the orchestration. In this paper, we propose a integer programming formulation based on first order stochastic dominance as a strategy for re-negotiation over multiple services. A consequence of this is better end-to-end performance of the orchestration compared to random selection of services for re-negotiation. We also demonstrate this optimal strategy can be applied to negotiation protocols specified in languages such as *Orc*. Such strategies are necessary for composite services where QoS contributions from individual atomic services vary significantly.**

***Keywords*-Orchestrations; Negotiation; SLAs; Stochastic Dominance; Integer Programming.**

I. Introduction

Web services continue to attract applications in many areas [1], with focus now shifting to improving Quality of Service (QoS). Maintaining efficient QoS levels of invoked services is a major prerogative of composite web service orchestrations, in order to maintain end-to-end QoS requirements. Contractual guarantees and service level agreements (SLAs) [2] are critical to ensure adequate QoS performance of such composite services.

An important aspect of such service level agreements is negotiation among service providers [3] [4]. The orchestration considers the end-to-end QoS against individual SLAs agreed with service providers. Negotiation ensures an acceptable level of QoS is maintained in composite service orchestrations, where the deterioration of individual services results in deteriorating overall performance. End-to-end performance is generally estimated through Monte-Carlo runs for composite services, having complex data and QoS interactions. QoS metrics being random variables, the treatment of such contractual obligations tends toward probabilistic criterion [5]. SLAs (used synonymously with contracts) may be specified as varying percentile values of such distributions rather than "hard" values. In [5], composition and monitoring such contracts with stochastic dominance have been examined.

In this paper, we examine negotiation of such probabilistic contracts having assumptions and guarantees. If the assumptions on certain metrics (such as *throughput*) are maintained by an orchestration, the sub-contractors guarantee a certain level or performance, for example *latency*. Considering this setting, the negotiation involves improvement in guarantees of the sub contractors such that overall improvement in end-to-end QoS is observed.

The problem here is to select the necessary service to re-negotiate with. In case of large orchestrations having both returned *data* and *QoS* values interacting, improving one service might not necessarily improve the end-to-end QoS. By the term *improvement*, we refer to *first order dominance* [6], that has been used to compare probability distributions (in the sense, drawing from one distribution is more likely to produce lower values). In composite service orchestration where individual sites may be invoked using a number of constructs (parallel, in sequence, fork-join, using timeouts/halting), identifying a particular service that may improve end-to-end QoS is difficult.

In order to overcome this difficulty, we formulate the problem as an optimization over minimizing *cost* of re-negotiation with respect to improvements in *latency* distribution (in the first order dominance sense): this is referred to as a *optimization strategy*. Using the notion of monotonicity [7], an improvement in the QoS performance of an individual service contributes positively to the overall improvement in the QoS (though with varying data-dependent contributions). As we are dealing with distributions and uncertainties, the use of stochastic dominance constraints is necessary. Stochastic dominance [6] has been used extensively in econometrics and related areas to perform decisions based on uncertainties. We make use of linear relaxations of these stochastic constraints [8] to formulate it as in integer programming problem. This provides a straightforward optimization problem that can be solved to obtain the most efficient re-negotiation strategy considering end-to-end QoS.

We evaluate our approach on a generic `GarageOnline` example that has both *data* and *QoS* values interacting. From our evaluation, we demonstrate that optimizing over con-

straints relating to stochastic dominance will produce better end-to-end contracts over multiple rounds of negotiation. This is compared against random selection of services (referred to as *random strategy*) for re-negotiation of composite services. As it is difficult to estimate the contribution of individual services to overall improvement, we believe such an optimization strategy is the best possible approach for transaction based orchestrations. Making use of optimization specifications in [9], the integer programming formulation as a negotiation strategy can be specified in languages such as *Orc*. An advantage of this is that runtime deterioration can be monitored to enter re-negotiation directly with participating services.

The rest of the paper is organized as follows: Section II provides foundation material for our paper including QoS in web services in Section II-A, probabilistic contracts in Section II-B, contract negotiations in Section II-C and an overview of Orc in Section II-D. The optimization formulation based on first order stochastic dominance constraints is presented in Section III. In Section IV we introduce the `GarageOnline` example. The problem of re-negotiating with individual services from a composite orchestration context is presented Section V. The methodology used to overcome these problems are discussed in Section V-C. Negotiation specifications as an extension of Orc is presented in VI. Discussion of results from the negotiation strategy is presented in Section VII. Related literature and conclusions are finally presented in Sections VIII and IX.

II. Foundations

This section provides a broad overview of topics relevant to our work.

A. Web services' QoS

Web services have protocols: they may be invoked using SOAP/REST with description provided by WSDL that are available on UDDI registries [1]. Industry standards in QoS [10] provide a family of QoS metrics that are needed to specify SLAs. These can be subsumed into the following four general QoS observations [1]:

1) $\delta \in \mathbb{R}_+$ is the service latency. When represented as a distribution, this can subsume other metrics such as availability and reliability of the service.
2) $\$ \in \mathbb{R}_+$ is the per invocation service cost.
3) $\zeta \in \mathbb{D}_\zeta$ is the output data quality. This can represent other metrics such as data security level and non-repudiation of private data over a scale of values.
4) $\lambda \in \mathbb{R}_+$ is the inter-query interval, equivalent to considering the query rate for a service. Performance of the service will depend on negotiations with the amount of queries that can be made in a given time interval.

Along with QoS, the web service performs its task and returns some functional data $\rho \in \mathbb{D}_\rho$ as the output.

[1]Aspects such as scalability, interoperability and robustness are not dealt with as they are specific to the supplier side operation (not necessarily part of SLAs).

B. Probabilistic Contracts

For a domain $\mathbb{D}_Q$ of a QoS parameter Q, behavior can be represented by its distribution F_Q:

$$F_Q(x) = \mathbb{P}(Q \leq x) \tag{1}$$

Making use of stochastic ordering [6], this is refined for probability distributions F and G over a totally ordered domain $\mathbb{D}$:

$$G_Q \preceq F_Q \quad \iff \quad \forall x \in \mathbb{D}_Q, \quad G_Q(x) \geq F_Q(x) \tag{2}$$

That is, there are more chances of being less than x (partial order $\leq$) if the random variable is drawn according to G than according to F.

Following the established approach of WSLA [11], a contract must specify the obligations of the two parties.

- The obligations that the orchestration has regarding the service are seen as *assumptions* by the service - the orchestration is supposed to meet them.
- The obligations that the service has regarding the orchestration are seen as *guarantees* by the service - the service commits to meeting them as long as assumptions are met.

Definition 1. *A probabilistic contract is a pair (Assumptions, Guarantees), which both are lists of tuples $(Q, \mathbb{D}_Q, F_Q)$, where Q is a QoS parameter with QoS domain $\mathbb{D}_Q$ and corresponding distribution F_Q.*

Once contracts have been agreed, they must be monitored by the orchestration for possible violation as described in [5]. Monitoring applies to each contracted distribution F individually, where F is the distribution associated to some QoS parameter Q having partially ordered domain $\mathbb{D}_Q$. By monitoring the considered service, the orchestration can get an estimate of the actual distribution of Q. The problem is, for the orchestration, to decide whether or not G complies with F, where compliance is defined according to:

$$\sup_{x \in \mathbb{D}_Q} \{F_Q(x) - G_Q(x)\} \leq \epsilon \tag{3}$$

where ϵ is the level of deviation allowed from the contractual distribution.

Monotonicity - It is important to make note of monotonicity in orchestrations as specified in [7]. This implies that a superior performance of a particular service invoked in the orchestration contributes positively to the overall performance of the orchestration. Such an assumption is crucial in negotiation based framework where contracts are composed.

C. Contract Negotiation

Contract negotiations relates to the procedure of parties (clients and providers) agreeing on the terms of an SLA. The typical negotiation steps are the following:

1) The provider publishes a template describing the service and associated metrics, including the QoS and possible compensations in case of violation.

2) The client fetches the template, and fills it in with values which describe the planned resource usage.
3) This non-binding document, is then modified by the provider (based on the current resource availability) to provide the client with a quote.
4) The client, if satisfied with the quote, applies his/her signature to the document, and sends it back to the provider as a SLA proposal.
5) The provider, receiving the proposal, is free to reject or accept it. In the latter case, the proposal becomes an SLA officially signed by both parties, and starts to be a valid legal document.

The quotes exchange (steps 2 and 3) can be repeated any number of times. The parties may freely modify the different terms: lower fees, lower QoS, longer time slots, fewer resource needs, lower compensations and so on. Once a contract has been signed and agreed, the necessity of changing it could be envisaged (re-negotiation).

D. Orc

Orc [12] serves as a simple yet powerful concurrent programming language to describe web services orchestrations. The fundamental declaration used in the Orc language is a *site*. The type of a *site* is itself treated like a service - it is passed the types of its arguments, and responds with a return type for those arguments. An Orc *expression* represents an execution and may call external services to publish some number of values (possibly zero).

Orc has the following combinators that are used on various examples as seen in [12]. The *Parallel* combinator $F \mid G$, where F and G are Orc expressions, runs by executing F and G concurrently; returns from F and G are interleaved. The execution of the *Sequential* combinator $F >x> G$ ($F \gg G$) starts by executing F. Values published by copies of G are published by the whole expression, but the values published by F are not published by the whole expression; they are consumed by the variable The *Pruning* combinator, written $F <x< G$, allows us to block a computation waiting for a result, or terminate a computation. The execution of $F <x< G$ starts by executing F and G in parallel. Whenever F publishes a value, that value is published by the entire execution. When G publishes its first value, that value is bound to x in F, with the execution of G immediately terminated. The *Otherwise* combinator, written $F \mathbf{;} G$ starts by executing F. If F completes, and has not published any values, then G executes. If F did publish one or more values, then G is ignored.

Further details on site declarations, data structures, channels, semaphores and timers may be found in the Orc documentation [2].

III. Optimization Formulation

In this section, we formulate the re-negotiation strategy as an optimization problem. We start from a general stochastic optimization setting and move on to the web services' re-negotiation strategy. As stochastic optimization involve comparison of random variables, this can be suitably applied to distributions of QoS values. To reduce the search space and complexity of comparison, approximations to convert the problem to integer programming proposed by [8] is used.

A. Stochastic Optimization

For formulating the problem with first order stochastic dominance, we define a triple $(\Omega, \mathbf{F}, \mathbb{P})$ as a probability space, where Ω is the entire space, $\mathbf{F}$ is a subset of this space where probability measure $\mathbb{P}$ is defined. For the space of all random variables $\mathbf{X}$ defined on the $(\Omega, \mathbf{F})$, the right continuous cumulative distribution function (CDF) is defined as $F_X(\eta) = \mathbb{P}(X \leq \eta), \quad \eta \in \mathbb{R}, \ X \in \mathbf{X}$. For variable $X, Y \in \mathbf{X}$, X dominates Y in the first order sense $X \succeq Y$, if $F_X(\eta) \leq F_Y(\eta), \forall \eta \in \mathbb{R}$. The stochastic optimization problem with first order dominance constraints may be written as follows with $f(X)$ as the cost function and $X \succeq Y$ specifying first order dominance constraints:

$$\begin{array}{cc} \min & f(X) \\ \text{subject to:} & X \succeq Y, \quad X, Y \in \mathbf{X} \end{array} \tag{4}$$

Let $(\Omega, 2^{\Omega}, \mathbb{P})$ be a probability space with finite events $\Omega = (\omega_1, .., \omega_W)$ and corresponding probabilities $p_1, ..., p_W$. Consider a discrete random variable $Y \in \mathbf{X}$ with finite support and realizations y_i with probabilities q_i $(i = 1, 2, .., m)$. Assuming a right continuous step function F_Y with $y_1 < y_2 ... < y_m$, the first order stochastic dominance constrains in eq. (4) may be written as:

$$\mathbb{P}(X \leq y_i) \leq \mathbb{P}(Y \leq y_i), \quad i = 1, .., m \tag{5}$$

We have $\mathbb{P}(Y \leq y_i) = \sum_{k=1}^{i-1} q_k$ and the $\mathbb{P}(X \leq y_i) = \sum_{w=1}^{W} p_w z_{iw}; \ \ i = 1, ..., m; \ \ w = 1, ..., W$ such that binary decision variables:

$$z_{iw} = \begin{cases} 1 & \text{if } y_i - X(\omega_w) > 0 \\ 0 & \text{otherwise} \end{cases}$$

If we define a big number $M \in \mathbb{R}$ satisfying $M \geq \max_w \{y_m - X(\omega_w)\}$, the first order stochastic constraint problem may be formulated as a binary integer programming problem:

$$\begin{array}{cc} \min & f(X) \\ \text{Subject to:} & y_i - X(\omega_w) \leq M z_{iw} \\ & \displaystyle\sum_{w=1}^{W} p_w z_{iw} \leq \sum_{k=1}^{i-1} q_k \\ & z_{iw} \in \{0, 1\} \\ & X \in \mathbf{X} \\ & i = 1, ..., m; \ \ w = 1, ..., W \end{array} \tag{6}$$

Such a formulation has been proposed in [8] as a relaxation to first order dominance constraints.

[2] http://orc.csres.utexas.edu/documentation.shtml

B. Web Services' Negotiation

Assume there are $\mathcal{S}_1, ...\mathcal{S}_N$ services characterized by the triple $\mathcal{S}_j := (c_j, F'_j(\delta), F_j(\delta))$ representing cost of the service c_j along with reformulated F'_j and original latency F_j distribution. As we are dealing with re-negotiating a single service from this set, we also use the inverse proportionality of cost c_j and the mean/median of the reformulated distribution $\widehat{F}'_j$ (a bigger reformulated mean/median would mean higher cost). While we focus on the tradeoff between latency and cost, other metrics can be formulated similarly using weighted or lexicographic ordering.

$$\begin{aligned} \min \quad & \sum_{j=1}^{N} s_j c_j \\ \text{Subject to:} \quad & F'_j(\delta) s_j \preceq F_j(\delta) s_j \\ & c_j = \frac{K_j}{\widehat{F}'_j} \\ & \sum_{j=1}^{N} s_j = 1, \quad s_j \in \{0,1\} \end{aligned} \tag{7}$$

Continuing with the binary integer formulation, we assign cumulative distributions F_j with realizations f_{wj} and corresponding probabilities $p_{wj}, w = 1, ..., W$ with index j referring to a particular service. Similarly, assign cumulative distributions F'_j with realizations f'_{ij} and corresponding probabilities $q_{ij}, i = 1, ..., m$. With $F'_j \approx \sum_{k=1}^{i-1} q_{kj}$ and the $F_j \approx \sum_{w=1}^{W} p_{wj} z_{iwj}; \;\; i = 1, ..., m; \;\; w = 1, ..., W$ and binary decision variables:

$$z_{iwj} = \begin{cases} 1 & \text{if } f'_{ij} - f_{wj} > 0 \\ 0 & \text{otherwise} \end{cases}$$

Also define big numbers $M_j \in \mathbb{R}$ satisfying $M_j \geq \max_w \{f'_{ij} - f_{wj}\}$ (with f_{wj} characterized by p_{wj}), the first order stochastic constraint problem may be formulated as a binary integer programming problem.

$$\begin{aligned} \min \quad & \sum_{j=1}^{N} s_j c_j \\ \text{Subject to:} \quad & (f'_{ij} - f_{wj}) s_j \leq M_j z_{iwj} s_j \\ & s_j \sum_{w=1}^{W} p_{wj} z_{iwj} \leq s_j \sum_{k=1}^{i-1} q_{kj} \\ & c_j = K_j / \tfrac{1}{m} \sum_{i=1}^{m} f'_{ij} \\ & z_{iwj} \in \{0,1\}; \; f'_{ij}, f_{wj} \in \mathbf{X} \\ & \sum_{j=1}^{N} s_j = 1, \quad s_j \in \{0,1\} \\ & i = 1, ..., m; \; w = 1, ..., W; \; j = 1, ..., N \end{aligned} \tag{8}$$

The optimization formulation in eq. (8) essentially selects the best service $\mathcal{S}_j$ to re-negotiate with in terms of cost and latency. It makes use of the selection procedure outlined in eq. (7) with the relaxation of first order dominance on latency as in eq. (6). In this formulation, we select a single service $\mathcal{S}_j$ (indexed by s_j) with the lowest corresponding negotiation cost c_j that provides at least first order dominance with respect to previous latency distributions. Note that higher order stochastic dominance tests exist [6]; however, we limit our formulation to first order dominance for contract compliance. This is done so that pointwise comparison of QoS values can be made as equivalent to comparing distributions (see Theorem 5 in [5]).

IV. GarageOnline Example

We consider the `GarageOnline` demonstrative example presented informally in Fig. 2. It describes a web services orchestration to hire/order cars from garages with associated credit and insurance companies. When an order with a preference of *Gold/Standard* insurance is placed, the fastest responding `Garage` is chosen. The services for `Credit` and `Insurance` are then chosen depending on the lowest price returned. Notice that if the preference is set to "Gold", the orchestration chooses the `InsureGold` service. The

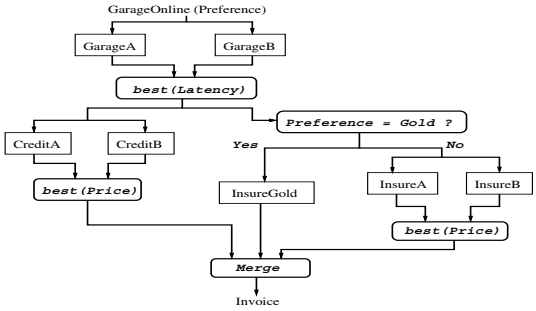

Figure 2. The `GarageOnline` orchestration.

Orc specification of the `GarageOnline` orchestration is presented in Fig. 1. The `Dictionary()` site is used as a mutable map from field names to values which are obtained using the . access. Values held by references are obtained using `x?` and set using `x:=y`. Operations on lists proposed in Orc are used to efficiently deal with multiple sites offering similar functionalities. In this specification we make use of the `bestQ` site to select among multiple domains (pruning with respect to latency or other QoS domains). In this orchestration multiple functionalities (eg. returned *best* price) and QoS values (eg. *best* latency) interact. Due to such subtle interactions contributing to the end-to-end contractual guarantees, re-negotiation with a particular service may not necessarily improve overall performance considerably (as discussed in Section V). Hence, an optimization formulation is required in case of such orchestrations to choose a feasible negotiation plan.

V. Composite Contract Re-Negotiation

This section starts from the runtime negotiation of sites making use of the competition operator proposed in [13]. The difficulty in choosing an optimal strategy for re-negotiation for end-to-end contractual obligations is then analyzed. Finally, a methodology for re-negotiation is provided keeping in mind monotonic conditions.

```
def GarageOnline(Order,Preference) =
  val GarageList = ["Garage A", "Garage B"]
  val CreditList = ["Credit A", "Credit B"]
  val InsureList = ["Insure A", "Insure B"]
  val InsureGoldList = ["InsureGold"]
  def bestQ(comparer,publisher) =  head(sortBy(comparer,collect(publisher)))
  def comparePrice(x, y) = x.price? <= y.price?
  def compareTime(x, y) = x.time? <= y.time?
  def inquireTime(List) = each(List) >sup> Dictionary() >ProductDetails> ProductDetails.Company := sup >>
    ProductDetails.time := (Rclock().time())  >> ProductDetails
  def inquirePrice(List) = each(List) >sup> Dictionary() >ProductDetails> ProductDetails.Company := sup >>
    ProductDetails.price := c >> ProductDetails

  def GenerateInvoce(Order,Preference) = Dictionary() >Invoice> Invoice.SubmitOrder := Invoice.ordernumber?
    >> Invoice.acceptedTime := Rclock().time() >> (Invoice,Preference)
  def Garage(Invoice) = bestQ(compareTime, defer(inquireTime,GarageList)) >q> Invoice.GarageQuote := q
  def Credit(Invoice) = bestQ(comparePrice, defer(inquirePrice,CreditList)) >q> Invoice.CreditQuote := q
  def Insure(Invoice,Preference) = if Preference = "Gold" then defer(inquirePrice,InsureGoldList) else
    bestQ(comparePrice, defer(inquirePrice,InsureList)) >q> Invoice.InsureQuote := q
GenerateInvoce(Order,Preference) >(Invoice,Preference)> Garage(Invoice) >> (Credit(Invoice),Insure(Invoice,Preference))
  >x> Invoice
```

Figure 1. The `GarageOnline` Orc specification.

A. Runtime Negotiation

Making use of the QoS calculus proposed in [13], the QoS domain is a tuple $\mathbb{Q} = (\mathbb{D}, \leq, \oplus, \triangleleft)$ defined as:

1) $(\mathbb{D}, \leq)$ defines a QoS domain along with associated partial order. For domains such as latency and cost, the partial ordering $\leq$ is preferred while for domains such as data quality, partial ordering$\geq$ is preferred.
2) $\oplus : \mathbb{D} \times \mathbb{D} \rightarrow \mathbb{D}$ defines the increments in QoS values (which can be zero). Note that the operator $\oplus$ is monotonic with $\delta_1' \leq \delta_1$ and $\delta_2' \leq \delta_2$ implying $\delta_1' \oplus \delta_2' \leq \delta_1 \oplus \delta_2$.
3) $\triangleleft : \mathbb{D} \times \mathbb{D}* \rightarrow \mathbb{D}$ is the competition operator that may be applied as choosing the "best" choice in many ways: pareto optimal, lexicographic or weighted choice. In case of synchronization across domains, for example $(c_1, \delta_1) \triangleleft (c_2, \delta_2)$ when ordered lexicographically would mean if $c_1 \leq c_2$ then $(c_1, \max(\delta_1, \delta_2))$ else $(c_2, \max(\delta_1, \delta_2))$. Note that the competition operator is monotonic with $\delta_1' \leq \delta_1, \quad \delta_2' \leq \delta_2$ implying $\delta_1' \triangleleft \delta_2' \leq \delta_1 \triangleleft \delta_2$.

The runtime specification of the "best" operator specified in Section III makes use of domains *(cost,latency)* for competing services for negotiation. The $\triangleleft$ operator is monotonic as demonstrated below. Consider two services $\mathcal{S}_1, \mathcal{S}_2$ with QoS increments to the orchestration δ_1, δ_2. They are both candidates for negotiation with the tuple of $(c_1, \delta_1'), (c_2, \delta_2')$. The operator $\triangleleft$ works by:

$$\begin{aligned} \mathcal{S}_1 \triangleleft \mathcal{S}_2 = \quad & \min(s_1 c_1 + s_2 c_2) \\ & \text{subject to: } \delta_1' \leq \delta_1, \delta_2' \leq \delta_2, \\ & c_i = K_i/\delta_i', \textstyle\sum_i s_i = 1, s_i \in \{0,1\} \end{aligned} \tag{9}$$

As we have chosen the minimum from cost domain c with a partial order condition $\leq$ on latency, the operation is indeed monotonic. The chosen (re-negotiated) service will not deteriorate the latency $\delta' \leq \delta$, which in turn will not deteriorate the end-to-end QoS for a monotonic orchestration. Similar competition policies are shown to be monotonic using branching cells and unfolding of Petri nets in [13].

B. End-to-end QoS

We examine the difficulty of re-negotiating with individual service when data and QoS interact in a composite service orchestration. A set of services called with Orc combinators sequential ($\gg$), fastest response (pruning $\ll_\delta$ where δ refers to latency) or "best" response (fork-join $\ll_q$ where q refers to other metrics such as price, returned data, security level etc.) can have significant differences in overall contribution to the contract. We summarize the effect of a change ϵ in contractual obligations for latency (can be seen as a shift in the median relative to the ordering). Assume here monotonic orchestrations with an improved contract contributing positively to overall behavior. The Orc combinators determine the effect on overall behavior:

- *Sequential* - The original contract would be $A \overset{\delta_A}{\gg} B$. If a reformulated contract decreases the value, it will be seen by the whole orchestration as: $A \overset{\delta_A - \widehat{\epsilon}_A}{\gg} B$, where $\widehat{\epsilon}$ is the contribution to the end-to-end QoS.
- *Fastest Response* - The expression $\ll_\delta (A|B)$ (also written as $\texttt{let}(A|B)$) refers to choosing the "best" service according to δ. The original contract would choose the fastest responding service $\min(\delta_A, \delta_B)$. Now we consider improving the contract such that we have to choose from $\min(\delta_A - \epsilon_A, \delta_B - \epsilon_B)$. However, the end-to-end QoS will decrease only by $\widehat{\epsilon}$ that is dependent on the "best" latency response.
- *Best value* - The expression $\ll_q (A, B)$ refers to choosing the "best" service according to q (as in fork-join $(A, B) >(x, y)> (x, y)$). The original contract would entail the worst responding service $\max(\delta_A, \delta_B)$. Now we consider improving the contract such that we have to choose from $\max(\delta_A - \epsilon_A, \delta_B - \epsilon_B)$. However, the end-to-end QoS will decrease only by $\widehat{\epsilon}$ that is dependent on the "best" value response.
- *Orchestration* - Emphasis must be placed here on the difficulty in selecting a particular service in case of a composite orchestration. For example, consider the Orc expression $((\texttt{let}(A \gg B \gg C) \mid X), Y)$ with the orchestration aware of the causal history of individual sites. Latency improvements ϵ in say site B (called sequentially) can improve the overall latency of $\texttt{let}(A \gg B \gg C)$ and still be nullified by the performance of site Y ($\widehat{\epsilon} \approx 0$). Thus, choosing a sequential or "best" cost/QoS service that can provide optimal improvement in the overall performance is difficult.

As discussed, the *level* of improvement may not lead to first order dominance over previously observed contracts. However, the end-to-end QoS improvement $\widehat{\epsilon}$ is indeed positive due to our assumption of monotonicity. Our methodology provides local improvements to contracts, which in turn improves end-to-end QoS (or does not, at least, deteriorate it).

C. Re-negotiation methodology

In order to produce an optimal strategy for re-negotiation in orchestrations, we present the following methodology:

1) Using subcontracts F_j proposed by individual services $\mathcal{S}_j$, generate an end-to-end contract for the orchestration. Note that some services have no sub-contracts and must be accepted as-is for performance. Denote this overall contract of the orchestration as F_{orch}.
2) If the end-to-end contract is acceptable, stop and accept all sub-contracts. Else, proceed to step 3.
3) Re-negotiate with one of the sub-contractors offering lowest costing improvements for an improved contract (in the first order sense). This choice is performed using Eq. 8. Once a new contract F'_a for service $\mathcal{S}_a, a \in j$ is selected, repeat step 1 and 2. Increment the number of *rounds* of re-negotiation.

In the rest of the paper, this methodology is also referred to as an *optimal strategy* for re-negotiation. Unlike [13], we use only the guarantees for re-negotiation and ignore the assumptions of the orchestration (throughput of service calls). However, formulating the assumptions as a tradeoff can be done using a similar methodology.

As we are considering improvements in only the sub-contractors performance, we intend to study the effect on end-to-end QoS. As discussed in Section V-B, combinators used in languages such as Orc/BPEL cannot be incorporated into the formulation without introducing some bias (sequentially invoked services given larger weights) to selecting a particular service - hence, it is ignored and the notion of monotonicity is used. For a monotonic orchestration, the re-negotiation formulation will always improve the overall contract F_{orch}. This follows from the objective function (c_j, F'_j, F_j) used. As the new service performs with a new contract: $F'_j \preceq F_j$ (partial ordering defines the dominance relation), the monotonic orchestration cannot deteriorate due to the re-negotiated contract. This might incidentally be due to the inverse relation between cost and latency, which seems a plausible model for service behavior (higher costs produce better service). Note that we make use of the property of first order stochastic dominance that allows ordinary comparison of QoS values for monotonic orchestrations.

VI. Negotiation Specification

While XML based languages like WSLA [11] has been proposed for specifying SLA contracts, standard languages for negotiation are limited. We have introduced a percentile-based approach to improve WSLAs in [14]. However, the WSLA language, though suited for intricate contractual specification, is not sufficient to deal with multiple rounds of negotiation proposals. Further, WSLA does not support percentile based handling or optimization among metrics, which limits its applicability to our methodology.

We believe languages such as Orc, that can provide access to external sites and inherently support recursion, to be a better alternative to specify multiple negotiation rounds. This can be extended to specify optimization in Section III making use of optimization sites proposed in [9]. Thus, the procedure can be directly applied to specifications of orchestrations, used to re-configure runtime aspects of the specification. Essentially, a `Negotiation` service specified in Orc can use historical runtime metrics to re-negotiate with services.

To fulfill the negotiation process described in Section V-C, the `Negotiation` service has the following operations:

- The `getQoS` service, when passed the inputs `sitex` and `QoS` returns the QoS values. This may be a distribution (latency) or a constant value (security level) and are stored in a `Channel()` site. When a list of values from a distribution is obtained, it is converted into quantile values and associated probabilities (`l` $\rightarrow$ `(f, p)`).
- The `getOffer` site, when passed the inputs `sitex` and `QoS` returns a tuple of `(QoSnew, cost)`. This represents the increments provided by the individual sites (`sitex`) for the QoS domain queried. It must be noted that a `null` value may be returned by sites that do not wish to re-negotiate or that have an *as-is* acceptance policy.
- The **`Optima`** site [9] can be used to specify the integer programming formulation when presented with a set of distributions. It can return the optimal site for a new contract according to eq. (8). We use online optimization services such as *lp-solve* provided by [15] for performing the optimization procedure. Note that the `Cost` weight is kept as 1 in the optimization specification.

We provide this specification for a simple version of `GarageOnline` having only `GarageA` and `GarageB` as part of the negotiation protocol. This is specified in Orc in Fig. 3.

Once this procedure is completed, a Monte-Carlo run of the overall orchestration `GarageOnline` may be performed to estimate the improvement in QoS. In case the improvement is not satisfactory, a renewed round of negotiation may be necessary. An advantage of this technique is that runtime evaluation of the orchestrations can trigger re-negotiation protocols to be entered. This is advantageous for systems where QoS deterioration may mean substantial loss of revenue.

VII. Negotiation Results

For the `GarageOnline` example, it is difficult to estimate which service to re-negotiate with in case of probabilistic SLAs. We use the optimization strategy developed in Section V-C to re-negotiate with individual atomic services. The re-negotiation is done using Eq. 8 with 10,000 values generated

```
def Negotiation(GarageOnline) =
  type Latency = Number
  type Cost = Number
  def getQoS(site, QoSDom) =
    val chsite = Channel()
    chsite.put(QoS) >> stop; chsite.getAll()
  def getOffer(site, QoSDom) =
    val chsite = Channel()
    chsite.put(QoS) >> stop; (chsite.getAll(), Cost)
  def Optima(Objective, Weight, Constraint, Solver) = Output
getQoS(GarageA(),latency) >(l_GarageA)> (f_GarageA,p_GarageA) >> getQoS(GarageB(),latency) >(l_GarageB)> (f_GarageB,p_GarageB)
  >> getOffer(GarageA(),latency) >(l_GarageA,c_GarageA)> ((f'_GarageA,q_GarageA),c_GarageA)
  >> getOffer(GarageB(),latency) >(l_GarageB,c_GarageB)> ((f'_GarageB,q_GarageB),c_GarageA) >>
  Optima([c_GarageA,c_GarageB], 1, (f_GarageA - f'_GarageA,(>),K1),  (sum(q_GarageA) - sum(p_GarageA),(>),K2),
  (f_GarageB - f'_GarageB,(>),K1), (sum(q_GarageB) - sum(p_GarageB),(>),K2), ''binary integer'')
```

Figure 3. The `GarageOnline` Negotiation specification in Orc using FIFO channels (`Channel()`) and pattern matching.

for both distributions. The values of f_j and f'_j are set as the $0.1, 0.2, ...0.9$ quantile in each case with associate probability p values. Here, f_j refers to the quantiles from the original latency distribution and f'_j refers to the new distribution associated with a cost c_j for a site $\mathcal{S}_j$.

Fig. 4 compares the improvements with respect to latency (t-location distribution with varying medians) and Fig. 4(c) for cost (generated from a normal distribution). These values may be drawn from real measurements (web services deployed on a network) or through re-sampling/bootstrapping values. As noticed, random choices made at runtime for 10 rounds of negotiation produces neither superior latency nor lower costs. By a round, we refer to one negotiation policy accepted with a particular service. Hence, an efficient technique for selecting the service to be negotiated with (Eq. 8) is needed. As demonstrated, the improvements in the end-end-end contract for both latency and cost produce significant improvements using an optimal strategy.

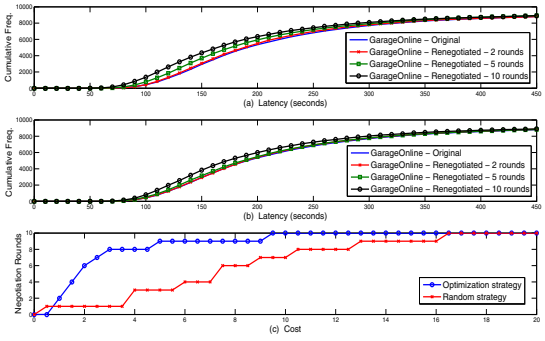

Figure 4. Latency improvements with 10 rounds of negotiation for `GarageOnline` (a) Using optimization strategy (b) Random strategy. The Fig. (c) shows Cost incurred with 10 rounds of negotiation.

Further, we demonstrate the improvements seen from individual services in Fig. 5 after 10 rounds of negotiation. Some services (eg. `GarageA`) might improve with every round while others (eg. `CreditB`) are not selected for negotiation owing to higher costs in the optimal strategy. This differs from those services that are selected randomly for re-negotiation, hence producing higher costs in Fig. 4(c). The experiments were conducted in MATLAB using the `bintprog` output. Other commercial solvers such as CPLEX or LPSOLVE should work similarly when invoked as a web service.

As we see in Fig. 5, the services do not contribute equally

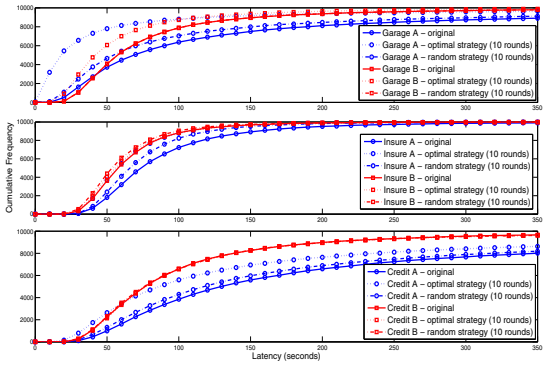

Figure 5. Improvements in performance of individual services after 10 rounds of optimization strategy.

to overall performance of the orchestration. An optimization strategy is thus imperative when a number of rounds of negotiations are taking place. The advantage of using such a strategy is monotonic improvement in end-to-end QoS, with re-negotiated sites guaranteeing latency improvements from a first order dominance point of view.

VIII. Related Work

The use of probabilistic QoS and soft contracts was introduced by Rosario et al. [13] and Bistarelli et al. [16]. Instead of using fixed hard bound values for QoS metrics, the authors proposed a soft contract monitoring approach to model the QoS measurement. The composite service QoS was modeled using probabilistic processes by Hwang et al. [17] where the authors combine orchestration constructs to derive global probability distributions.

Service level agreements (SLAs) have been specified in a number of papers using WSLA [11]. In [18], the framework needed for handling SLAs are described in detail. In [14], we have presented methodology for introducing percentile based constraints to WSLA framework, that is essential for probabilistic models. This is in line with high-level and rich specification languages such as QML [19] that provide a variety of constructs for QoS management. Related studies of optimal QoS compositions make use of genetic programming in Canfora et al. [20] and linear programming in Zeng et al. [21]. These make use of optimization strategies for composition but do not deal with negotiations.

Probabilistic models for on the fly negotiation of service agreements are presented in [3]. Multiple QoS dimensions are considered with tradeoffs included for negotiation be-

tween a single service provider and customer/orchestration. The automatic negotiation process in [4] aims at identifying the maximum quality level admissible with respect to the user budget. In [16], negotiation is said to proceed using relaxed constraints with algebraic operators modeled in a semi-ring.

While these papers examine algorithms for negotiation with individual services, the problem of optimizing negotiation and end-to-end QoS has not been studied. This is an important problem to consider, specially in the case of data and QoS dependent orchestrations. We make use of relaxations in stochastic constraints proposed in [8] to develop a generic strategy for re-negotiation. Such a strategy is critical when there are a number of services with different contributions to the end-to-end QoS.

The use of dynamic/stochastic programming [22] in service selection has been proposed in [23]. However, incorporating such stochastic constraints entails heavy computational costs that may not be necessary. Alternatives to using first order dominance constraints include lighter higher order constraints [24]. In this paper, we make use of linear approximations of stochastic constraints [8] to improve computational efficiency. Using the optimization specifications provided in [9], a number of online solvers [15] may be invoked within Orc. This specification allows for runtime re-negotiation of contractual obligations with the optimal service (dependent on cost, latency) selected.

IX. Conclusions

In this paper, we study the negotiation of SLAs in web services orchestrations. While available literature deals with one-to-one negotiation between customers and clients, little work has been done in optimizing negotiation to improve end-to-end performance. This procedure is difficult to perform when data and QoS interact, when varied control flow combinators are applied in orchestrations: thus leading to uncertainty in an individual service's contribution to overall performance. We demonstrate that using an integer programming formulation, constraints may be specified to select the service that provides the best re-negotiation strategy in monotonic cases. This, in turn, would perform much better than random re-negotiation strategies. These procedures may be specified in languages such as Orc to aid in runtime re-negotiation.

References

[1] G. Alonso, F. Casati, H. Kuno, and V. Machiraju, *Web Services - Concepts, Architectures and Applications*. Springer-Verlag, 2004.

[2] P. Bhoj, S. Singhal, and S. Chutani, "Sla management in federated environments," *Computer Networks*, vol. 35(1), pp. 5–24, 2001.

[3] C. Cappiello, M. Comuzzi, and P. Plebani, "On automated generation of web service level agreements," *CAiSE, LNCS,*, vol. 4495, pp. 264–278, 2007.

[4] J. Yan, R. Kowalczyk, J. Lin, M. B. Chhetri, S. K. Goh, and J. Zhang, "Autonomous service level agreement negotiation for service composition provision," *Future Gener. Comput. Syst.*, vol. 23, pp. 748–759, July 2007.

[5] S. Rosario, A. Benveniste, and C. Jard, "A theory of qos for web service orchestrations," INRIA Research Report 6951, Tech. Rep., 2009.

[6] G. F. Barrett and S. G. Donald, "Consistent tests for stochastic dominance," *Econometrica*, vol. 71, pp. 71–104, 2003.

[7] A. Bouillard, S. Rosario, A. Benveniste, and S. Haar, "Monotony in service orchestrations," INRIA Research Report 6528, Tech. Rep., 2008.

[8] G. R. Nilay Noyan and A. Ruszczynski, "Relaxations of linear programming problems with first order stochastic dominance constraints," *Operations Research Letters*, vol. 34, pp. 653–659, 2006.

[9] A. Kattepur, A. Benveniste, and C. Jard, "Optimizing decisions in web services orchestrations," in *International Conference on Service Oriented Computing*, 2011.

[10] I. Toma and D. Foxvog, "Non-functional properties in web services," Web Services Modeling Ontology (WSMO) Final Draft, Tech. Rep., 2006.

[11] H. Ludwig, A. Keller, A. Dan, R. P. King, and R. Franck, "Web service level agreement (wsla) language specification," IBM Corporation, Tech. Rep., 2003.

[12] D. Kitchin, A. Quark, W. R. Cook, and J. Misra, "The orc programming language," in *Proceedings of FMOODS/FORTE 2009*, ser. Lecture Notes in Computer Science, vol. 5522. Springer, 2009, pp. 1–25.

[13] S. Rosario, A. Benveniste, and C. Jard, "Flexible probabilistic qos management of transaction based web services orchestrations," in *IEEE International Conference on Web Services*, 2009, pp. 107 –114.

[14] A. Kattepur, "Importance sampling of probabilistic contracts in web services," in *International Conference on Service Orinted Computing*, 2011.

[15] R. Fourer and J.-P. Goux, "Optimization as an internet resource," *Interfaces: OR/MS and E-Business*, vol. 31, pp. 130–150, 2001.

[16] S. Bistarelli and F. Santini, "Soft constraints for quality aspects in service oriented architectures," in *Young Researchers Workshop on Service-Oriented Computing*, 2009.

[17] S.-Y. Hwang, H. Wang, J. Srivastava, and R. A. Paul, "A probabilistic qos model and computation framework for web services-based workflows," in *ER 2004, LNCS 3288, Springer-Verlag Berlin Heidelberg*, 2004.

[18] A. Dan, D. Davis, R. Kearney, A. Keller, R. King, D. Kuebler, H. Ludwig, M. Polan, M. Spreitzer, and A. Youssef, "Web services on demand: Wsla-driven automated management," *IBM Systems Journal*, vol. 43, 2004.

[19] S. Frolund and J. Koistinen, "Qml: A language for quality of service specification," Software Technology Laboratory, Hewlett Packard, Tech. Rep., 1998.

[20] G. Canfora, M. D. Penta, R. Esposito, and M. L. Villani, "An approach for qos-aware service composition based on genetic algorithms," in *GECCO*, 2005.

[21] L. Zeng, B. Benatallah, A. H. Ngu, M. Dumas, J. Kalagnanam, and H. Chang, "Qos-aware middleware for web services composition," *IEEE Transactions on Software Engineering*, vol. 30, pp. 311–326, 2004.

[22] A. Shapiro, D. Dentcheva, and A. Ruszczyski, *Lectures on Stochastic Programming: Modeling and Theory*. Society for Industrial Mathematics, 2009.

[23] Y. Gao, J. Na, B. Zhang, L. Yang, and Q. Gong, "Optimal web services selection using dynamic programming," in *11th IEEE Symposium on Computers and Communications*, 2006.

[24] M. Levy and H. Levy, "Testing for risk aversion: a stochastic dominance approach," *Economics Letters*, vol. 71, pp. 233–240, 2001.