Towards QoS Prediction Based on Composition Structure Analysis and Probabilistic Models*

Dragan Ivanović[1], Manuel Carro[1,2], and Peerachai Kaowichakorn[2]

[1] IMDEA Software Institute, Spain
[2] School of Computer Science, U. Politécnica de Madrid (UPM), Spain
dragan.ivanovic@imdea.org, mcarro@fi.upm.es

Abstract. The quality of service (QoS) of complex software systems, built by composing many components, is essential to determine their usability. Since the QoS of each component usually has some degree of uncertainty, the QoS of the composite system also exhibits stochastic behavior. We propose to compute probability distributions of the QoS of a service composition using its structure and the probability distributions of the QoS of the components. We experimentally evaluate our approach on services deployed in a real setting using a tool to predict probability distributions for the composition QoS and comparing them with those obtained from actual executions.

1 Introduction

Analyzing and predicting QoS of service compositions during the design phase makes it possible to explore design decisions under different environment conditions and can greatly reduce the amount and cost of maintenance, help the adaptation of software architectures, and increase overall software quality.

The QoS of a service composition depends both on the QoS of the individual components and on the structure of the composition. The effects of the execution environment also impact the observed QoS, which exhibits a stochastic variability due (among others) to changes in network traffic, machine load, cache behavior, database accesses at a given moment, etc. QoS prediction is notoriously challenging when, as in the case of service-oriented systems, boundaries and behavior are not fully specified.

```
Input: transport
if transport == "train"
    call SearchTrain
else
    call SearchFlight
end
```

Fig. 1. Simple orchestration

Fig. 1 shows a fragment of a service composition. Let us assume that we are interested on execution time and that we know (e.g., from observations) the probability distribution functions for the response times of the two services invoked in it (Fig. 2 (a) and (b)), whose averages are 5 and 3 seconds, respectively. The average response time for Fig. 1 may actually be seldom observed, as executions cluster around 3 and 5

* The research leading to these results has received funding from the EU FP 7 2007-2013 program under agreement 610686 POLCA, from the Madrid Regional Government under CM project S2013/ICE-2731 (N-Greens), and from the Spanish Ministry of Economy and Competitiveness under projects TIN-2008-05624 DOVES and TIN2011-39391-C04-03 StrongSoft.

X. Franch et al. (Eds.): ICSOC 2014, LNCS 8831, pp. 394–402, 2014.

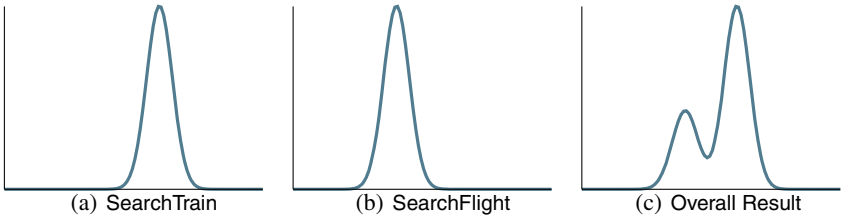


Fig. 2. Statistical profiles for Fig. 1

seconds. Moreover, this average is not useful to answer questions such as *what is the probability that the answer time is less than 4 seconds*, which is interesting to, for example, negotiate penalties to compensate for SLA deviations.

If we know the probability that train / plane trips are requested (e.g., 0.3 and 0.7, respectively), we can construct the probability distribution of the QoS of the composition (Fig. 2 (c)). This result gives much more information and insight on the expected QoS of the composition, and makes it possible to answer the question presented above.

2 Related Work

The basis for the classical approach to the analysis of QoS for service compositions [2,1,4] is *QoS aggregation*. Most approaches focus on the control structure without considering data operations, and return expected values as result. This falls short to describe the composition behavior. More recent approaches infer upper and lower bounds of QoS based on input data and environmental factors [5]. However, bounds often need to be too large, since services often exhibit a "long-tail" behavior, and bounds do not capture the shape of the distribution of values, which limits the usefulness of the description.

Recent proposals [7] work directly with statistical distributions of QoS, but use a very abstract composition model that is far from existing implementation languages and does not take into account internal compositions of data and operations, which is bound to give less accurate results. The work in [6] uses probability distributions to drive SLA negotiation and optimization. While its focus is complementary to ours, it does not take into account the relationships between the internal state and data operations of the composition (including the initial inputs) and its QoS.

3 Probabilistic Interpretation of Compositions

Our method interprets control structures and data operations (Fig. 3) in a probabilistic domain by computing with discrete probability distributions rather than just with representative data points. For each (tuple of) variable(s) there is a mapping ρ which assigns a probability to each value / tuple. These represent the uncertainty in the QoS of services, the values of data items, and the resulting QoS of the composition. The mapping initially assigns a separate probability distribution for the domain of each variable (including QoS metrics). When variables are used together in an expression or a branch condition, their values may become related and joint probabilities need to be used. The

$$C ::= \langle \text{variable} \rangle := E \mid \textbf{call}\ \langle \text{service} \rangle \mid \textbf{if}\ B\ \textbf{then}\ C\ \textbf{else}\ C \mid \textbf{while}\ B\ \textbf{do}\ C$$
$$\mid \textbf{begin}\ C[;\ C]^*\ \textbf{end} \mid \textbf{or}\ C[;\ C]^*\ \textbf{end} \mid \textbf{and}\ C[;\ C]^*\ \textbf{end} \mid \textbf{skip}$$

$$E ::= \langle \text{numeral} \rangle \mid \langle \text{variable} \rangle \mid E \circ E \qquad (\circ \in \{+, -, *, \textbf{div}, \textbf{mod}\})$$

$$B ::= E\ \delta\ E \mid B \wedge B \mid B \vee B \mid \neg B \qquad (\delta \in \{>, \geq, =, \neq, <, \leq\})$$

Fig. 3. Abstract syntax for composition constructs

interpretation of every construct starts with a distribution ρ before the construct is executed and produces a ρ' after it is executed. ρ' describes all possible executions (and only those) that are consistent with the distribution ρ before the execution.

Let us assume variables $x, y \in \{1,2\}$ and their probability distributions in Fig. 4 and the code in Fig. 5. At the beginning, all the combinations of these two values are equally probable. The question is what are the probabilities of the possible values of x and y at the end of the **if**–**then**–**else**; let us call these values x' and y'. Since whether x or y are updated depends on their concrete values, not all combinations of $x+10$ and $y+10$ are possible. If $x' = 1$, it must be $y' = 11$, but $y' = 12$ is not possible: y cannot be incremented if $x = 1, y = 2$. The only valid combinations are $(x', y') \in \{(1,11), (11,2), (2,11), (2,12)\}$ with probability 0.25 each. Then, the values of x and y have become entangled and need to be described as a joint probability distribution.

3.1 Elements of the Model

The elements that make up our model are: composition structure (to describe control constructs / data operations), the composition data (including input data and internal variables), the QoS attributes of interest, and statistical data on the services used in the composition. The QoS attributes may include execution time, amount of data sent / received, number of general / specific operations executed, availability, etc. as long as it can be numerically quantified. Our proposal is parametric on the QoS attribute: we only require probability distributions for each service and an aggregation operator.

Integer random variables are used to represent both the state of internal variables (discretized after an abstraction process, if necessary) and of the QoS attribute of interest. Random variables are categorized in three types:

- $\mathbf{X} = X_1 X_2 X_3 \ldots X_n$ represents variables in the composition.
- $\mathbf{S} = S_1 S_2 S_3 \ldots S_m$ represents the QoS attributes for each service in the composition.
- Q models the behavior of the selected QoS attribute for the composition.

We represent the values of the random variables in the set $Q\mathbf{XS}$ of $N = 1 + n + m$ variables with a *discrete joint probability distribution* $\rho : \mathbb{Z}^N \to [0,1]$ such that

Var.	*Val.* $\mapsto$ *Prob.*	*Val.* $\mapsto$ *Prob.*
x	$1 \mapsto 0.5$	$2 \mapsto 0.5$
y	$1 \mapsto 0.5$	$2 \mapsto 0.5$

Fig. 4. Probability distributions

```
if x < y then x = x + 10 else y = y + 10 end
```

Fig. 5. Code to be interpreted in a probabilistic domain

$\sum_{\mathbf{v}\in\mathbb{Z}^N}\rho(\mathbf{v})=1$. If $Y_1,Y_2,\ldots,Y_k$ are distinct variables from $Q\mathbf{XS}$ which we want to highlight and $\mathbf{V}$ is the ordered set of the $N-k$ remaining variables from $Q\mathbf{XS}$, we write

$$\rho(Y_1=y_1,Y_2=y_2,...,Y_k=y_k,\mathbf{V}=\mathbf{v}) \tag{1}$$

to denote the probability that $Y_1,Y_2,\ldots,Y_k$ and $\mathbf{V}$ have exactly the values $y_1,y_2,\ldots y_k\in\mathbb{Z}$ and $\mathbf{v}\in\mathbb{Z}^{N-k}$, respectively. When it is clear from the context we write (1) simply as $\rho(y_1,y_2,\ldots,y_k,\mathbf{v})$. When it is precisely known that $Q=q$, $\mathbf{X}=\mathbf{x}$ and $\mathbf{S}=\mathbf{s}$ for some $(q,\mathbf{x},\mathbf{s})\in\mathbb{Z}^N$, then $\rho(q,\mathbf{x},\mathbf{s})=1$, and for all other arguments ρ gives zero.

3.2 Initial Conditions and Independence

The interpretation starts with an initial distribution ρ. We do not enforce independence (non-entanglement) in this ρ, but we assume it here to simplify the presentation. For each state variable X_i and service S_j, $\rho_{X_i}:\mathbb{Z}\to[0,1]$ and $\rho_{S_j}:\mathbb{Z}\to[0,1]$ describe resp. their initial distributions of values and of QoS. $\rho_Q:\mathbb{Z}\to[0,1]$ describes the initial value for the composition QoS, Q, normally initialized to zero. The aggregate distribution $\rho_{\mathbf{X}}:\mathbb{Z}^n\to[0,1]$ (and similarly for $\rho_{\mathbf{S}}:\mathbb{Z}^m\to[0,1]$) is computed as

$$\rho_{\mathbf{X}}(x_1,x_2,\ldots,x_n)=\rho_{X_1}(x_1)\times\rho_{X_2}(x_2)\times\cdots\times\rho_{X_n}(x_n) \tag{2}$$

3.3 Assignments and Arithmetic

In an assignment $X:=E$, E may involve any number of variables from $\mathbf{X}$, including X. The *after* distribution ρ' needs to satisfy the condition $\rho'(x,\mathbf{v})=\sum\{\rho(u,\mathbf{v})\mid x=E[u,\mathbf{v}]\}$ where $E[u,\mathbf{v}]$ represents the result of E for $X=u$ and $\mathbf{V}=\mathbf{v}$. The probability for $(x,\mathbf{v})$ in ρ' aggregates the probabilities of all tuples $(u,\mathbf{v})$ from ρ where the expression E evaluates to x. For all other tuples, ρ' gives zero.

Assignments make the variable to the left of ":=" depend on the variables in E. E.g., for $X_1:=X_2+X_3$, the independent $\rho_{X_1}(x_1)$ from (2) is replaced with $\rho'_{X_1|X_2,X_3}(x_1\mid x_2,x_3)$ which gives the probability of $X_1=x_1$ given $X_2=x_2$ and $X_3=x_3$. Fig. 6 shows *before* and *after* distributions for the same example of assignment. The *after* state distribution ρ'_{X_1,X_2,X_3} is then computed as $\rho'_{X_1|X_2,X_3}\times\rho_{X_2}\times\rho_{X_3}$.

3.4 Service Invocation

A service invocation **call** s_i updates the expected QoS Q by composing its initial distribution with the random variable S_i representing the QoS of service s_i. For the case of execution time it amounts to adding random variables: $Q:=Q+S_i$.

x_1	ρ_{X_1}
0	1.0

x_2	ρ_{X_2}
1	0.3
2	0.5
4	0.2

x_3	ρ_{X_3}
0	0.4
1	0.6

x_1	x_2	x_3	$\rho'_{X_1\mid X_2,X_3}$
1	1	0	1.0
2	1	1	1.0
2	2	0	1.0
3	2	1	1.0
4	4	0	1.0
5	4	1	1.0

x_1	x_2	x_3	ρ'_{X_1,X_2,X_3}
1	1	0	0.12
2	1	1	0.18
2	2	0	0.20
3	2	1	0.30
4	4	0	0.08
5	4	1	0.12

Fig. 6. Sample probabilities for $X_1:=X_2+X_3$

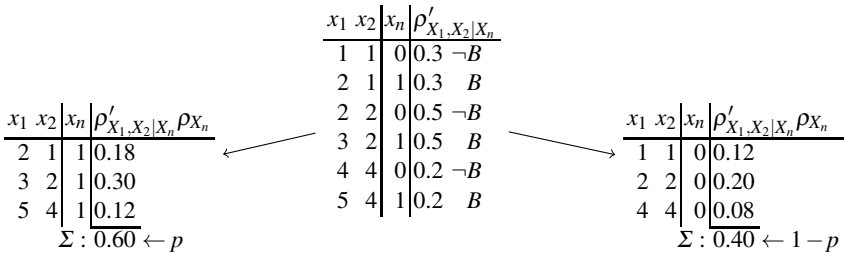


Fig. 7. Grouping and splitting under $X_2 > X_1$

Other QoS attributes will need specific aggregation operators. For example, availability will need to be aggregated with $\times$ instead of $+$. If the invoked service gives value to some variable X, its result value will have to be replaced in the after $\rho'_{\mathbf{X}}$.

We assume that the QoS and results of s_i do not depend on its input data. In our framework, taking this into account would require to include probability of outputs given inputs. Although doable, in practice this is a challenge for which we still do not have a satisfactory solution other than assuming a uniform distribution.

3.5 Sequential Composition

In a sequential composition **begin** C_1; C_2; …; C_k **end** the interpretation of each C_i computes ρ'_i from $\rho_i = \rho'_{i-1}$, and the sequence computes then $\rho' = \rho'_k$ from $\rho = \rho_0$.

3.6 Conditionals

In the construct **if** B **then** C_1 **else** C_2, we need to determine the probability of each branch. If $\mathbf{v}$ represents the value of all random variables from $Q\mathbf{XS}$, the probability of executing the **then** part is:

$$p = \sum \{\rho(\mathbf{v}) \mid B[\mathbf{v}]\} \tag{3}$$

where $B[\mathbf{v}]$ represents the truth value of B. If $p = 1$ or $p = 0$, we continue interpreting C_1 or C_2, resp. Otherwise, we interpret independently C_1 and C_2 with initial distributions ρ_1 and ρ_2, respectively, which are adjusted so that $\sum_{\mathbf{v}\in\mathbb{Z}^N} \rho_1(\mathbf{v}) = 1$ (resp. for ρ_2) according to the probability that B holds:

$$\rho_1(x,\mathbf{v}) = \rho(x,\mathbf{v})/p \tag{4}$$

$$\rho_2(x,\mathbf{v}) = \rho(x,\mathbf{v})/(1-p) \tag{5}$$

If ρ'_1 and ρ'_2 are the probabilistic interpretations of C_1 and C_2, the result for the whole construct will be $\rho' = p \times \rho'_1 + (1-p) \times \rho'_2$. ρ_1 and ρ_2 are generated by splitting the values of the variables in the condition B. Rather than filtering ρ in a straightforward implementation of (3), (4) and (5), we can group and split only the values of the random variables from $\mathbf{X}$ that appear in B. Fig. 7 shows the process using the same variables as in Fig. 6 for $B \equiv X_1 > X_2$. In the central table the probabilities for the **then** and **else** branches are normalized according to the probabilities of B and $\neg B$, and later split into two tables according to these two cases.

3.7 Loops

We restrict ourselves to terminating loops. Loop constructs (Eq. (6) are unfolded into a conditional, treated according to Section 3.6, and a loop (Eq. (7)):

while B **do** C_1 (6) **if** B **then begin** C_1; **while** B **do** C_1 **end else skip** (7)

Termination ensures that the unfolding is finitary. Existing techniques [3] can decide termination for many cases.

3.8 Or-Split and And-Split

For conciseness, we will not detail here the and- and or-split rules. We model them similarly to the sequential composition with two differences:

- The distributions for internal data are not carried over from C_i to C_{i+1}. The forked activities are assumed to work in independent environments.
- QoS aggregation differs. In the case of execution time, for the or-split, the total execution time is the minimum of the C_i; resp. maximum for the and-split.

3.9 Interpreting the Results

Let us recall that we want to answer questions such as what is the value $\Pr[Q \leq a]$. This can be computed from the final ρ' as $\Pr[Q \leq a] = \sum_{q \leq a} \sum_{\mathbf{v}} \rho'(q, \mathbf{v})$, where $\mathbf{v}$ is a tuple of values for all random variables from **XS**. Questions such as "*what is the probability that the process finishes in (exactly) 3 seconds*" are not useful, as it can be argued that the answer tends to zero. Questions such as "*what is the probability that the process finishes in 2.95 to 3.05 seconds?*" are more interesting; the answer can be computed as $\Pr[a \leq Q \leq b] = \Pr[Q \leq b] - \Pr[Q \leq a]$ for some a, b.

4 Experimental Validation

The experimental validation focused on execution time and was conducted on fully-deployed services. We compared actual execution times, obtained from a large number of repeated executions, with the distribution predicted by a tool.

4.1 Tool Implementation Notes

A fully-functional prototype of the tool has been implemented in Prolog, which gives excellent capabilities for symbolic representation and manipulation (for the abstract syntax and the probability distributions) and automatic memory management. The prototype receives the composition code, the values of the observed QoS for the services, and the expected values of the input variables; it interprets the program in a domain of probability distributions, and gives as result the expected QoS (time, in our examples) and, if requested, the distribution of the values of internal variables.

The services were implemented in Java and deployed on *Google App Engine*. The orchestration is a client-side Java application that connects to the services. The composition and the individual services were executed several hundred times to obtain a distribution ρ_E of the composition and the distributions ρ_{s_i} of the services. The distribution

ρ_P of the predicted execution time is produced from a single run of the interpreter. In order to find out the network impact on our results, we measured time both on the client and on the service to derive:

1. Total (round-trip) execution time (t_a), as measured on the client side.
2. Service execution time (t_e), measured by the service implementation and passed to the client. This excludes network transmission time.
3. Network transmission time $t_n = t_a - t_e$.

4.2 Experiment One: Matrix Multiplication

This service performs matrix multiplication. It receives two matrices from the orchestration (Fig. 8) and returns their product. Large square matrices (dimensions 500×500) are used to ensure meaningful execution times.

The multiplication service is called 500 times, recording t_a, t_e, and t_n for each invocation. The composition is executed 500 times.

```
begin
  x := 0;
  while x < 5 do begin
    call MatrixMultiplicationService;
    x := x + 1
  end
end
```

Fig. 8. Matrix multiplication

4.3 Experiment Two: Sorting

Implementations of *BubbleSort* and the *QuickSort* algorithms were deployed. Both services receive an array of integers and return a sorted array. The client-side composition creates ten 1000-element arrays of integers and invokes the services. To generate service time distributions for the analyzer, each service is invoked 500 times.

Fig. 9 sketches the composition. BubbleSort is invoked 20% of the times, QuickSort 30% of the time, and a mix of both 50% of the time.

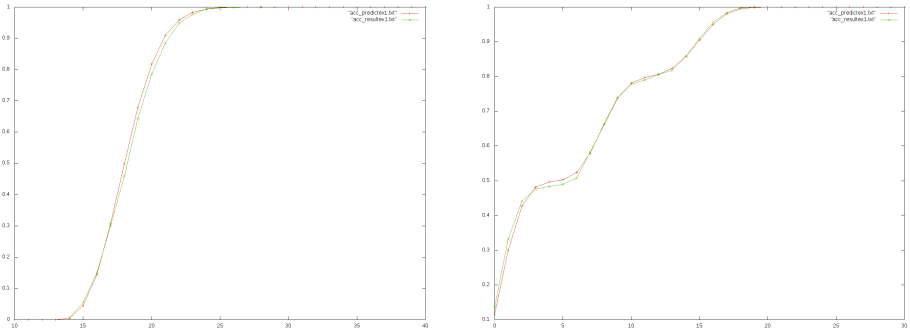
```
Input: n from 0 to 9
// Sort depending on the mode
if n < 2 then // bubble sort
    repeat 10 times:
      call BubbleSortService;
else if n < 5 then // quick sort
    repeat 10 times:
      call QuickSortService;
else // mix sort
    repeat 5 times:
      call BubbleSortService;
    repeat 5 times:
      call QuickSortService;
end
```

Fig. 9. Experiment two: composition structure

4.4 Experimental Results

Fig. 10 displays, for both experiments, the accumulated predicted and actual probability, i.e. $\Pr[T_P < t]$ and $\Pr[T_E < t]$ for the service execution times. Both lines are so close that it is difficult to distinguish them. While this suggests that the prediction is very accurate, it does not allow drawing clear conclusions about the prediction accuracy. Therefore we resorted to a numerical comparison using the *Mean Square Error* (MSE):

$$MSE = \frac{1}{t_{\max} - t_{\min}} \sum_{t=t_{\min}}^{t_{\max}} (\Pr[T_P < t] - \Pr[T_E < t])^2 \tag{8}$$



(a) Matrix multiplication (b) Sorting

Fig. 10. Comparisons for $\Pr[T_{Pn} < t]$ against $\Pr[T_{En} < t]$

Table 1. Mean Square Error

Measurement	*Observed Probability*	*Uniform Probability*	*Constant Probability*
Experiment 1			
$\Pr[T_{Pa} < t]$	0.070	0.383	0.577
$\Pr[T_{Pn} < t]$	0.012	0.310	0.434
$\Pr[T_{Pe} < t]$	0.0003	0.138	0.537
Experiment 2			
$\Pr[T_{Pa} < t]$	0.006	0.388	0.494
$\Pr[T_{Pn} < t]$	0.010	0.306	0.383
$\Pr[T_{Pe} < t]$	0.0001	0.126	0.488

The smaller the MSE, the more accurate the prediction is. However, the MSE is just a number whose magnitude we need to put in context to decide how good is the fitness we obtain — for example, comparing this number with the fitness obtained with other prediction techniques. This is not easy due to to the difficulty of installing and running tools implementing existing proposals.

Therefore we repeated the predictions using as input probability distributions to characterize external services either a single point (for the average approach) or a uniform distribution ranging from the observed lower to upper bound (for the bounds approach). We termed these scenarios *Constant Probability* and *Uniform Probability*, resp. Table 1 shows the evaluation results. From them it is clear that using the observed probability distribution produces much more accurate (for orders of magnitude) predictions. Most of the prediction errors come from the network characteristics, which are difficult to control. If the network issues are excluded ($\Pr[T_{Pe} < t]$), the predictions show very promising results with very small MSE.

References

1. Cardoso, J.: Complexity analysis of BPEL web processes. Software Process: Improvement and Practice 12(1), 35–49 (2007)
2. Cardoso, J., Sheth, A., Miller, J., Arnold, J., Kochut, K.: Quality of service for workflows and web service processes. Web Semantics: Science, Services and Agents on the World Wide Web 1(3), 281–308 (2004), http://www.sciencedirect.com/science/article/pii/S157082680400006X
3. Cook, B., Podelski, A., Rybalchenko, A.: Proving program termination. Commun. ACM 54(5), 88–98 (2011)
4. Dumas, M., García-Bañuelos, L., Polyvyanyy, A., Yang, Y., Zhang, L.: Aggregate quality of service computation for composite services. In: Maglio, P.P., Weske, M., Yang, J., Fantinato, M. (eds.) ICSOC 2010. LNCS, vol. 6470, pp. 213–227. Springer, Heidelberg (2010)
5. Ivanović, D., Carro, M., Hermenegildo, M.: Towards Data-Aware QoS-Driven Adaptation for Service Orchestrations. In: Proceedings of the 2010 IEEE International Conference on Web Services (ICWS 2010), Miami, FL, USA, July 5-10, pp. 107–114. IEEE (2010)
6. Kattepur, A., Benveniste, A., Jard, C.: Negotiation strategies for probabilistic contracts in web services orchestrations. In: ICWS, pp. 106–113 (2012)
7. Zheng, H., Yang, J., Zhao, W., Bouguettaya, A.: QoS Analysis for Web Service Compositions Based on Probabilistic QoS. In: Kappel, G., Maamar, Z., Motahari-Nezhad, H.R. (eds.) Service Oriented Computing. LNCS, vol. 7084, pp. 47–61. Springer, Heidelberg (2011)