Dieter Schuller, Melanie Siebenhaar, Ronny Hans, Olga Wenge, Ralf Steinmetz:
Towards Heuristic Optimization of Complex Service-based Workflows for Stochastic QoS Attributes.
In: International Conference on Web Services (ICWS), July 2014.

Towards Heuristic Optimization of Complex Service-based Workflows for Stochastic QoS Attributes

Dieter Schuller, Melanie Siebenhaar, Ronny Hans, Olga Wenge, Ralf Steinmetz
Multimedia Communications Lab (KOM)
Technische Universität Darmstadt
Darmstadt, Germany
Email: {firstname.lastname}@KOM.tu-darmstadt.de

Stefan Schulte
Distributed Systems Group
Vienna University of Technology
Vienna, Austria
Email: s.schulte@infosys.tuwien.ac.at

***Abstract*—The problem of selecting services from a set of functionally appropriate ones under Quality of Service constraints – the *Service Selection Problem* – is well-recognized in the literature based on deterministic parameters. However, Quality of Service may rather follow a stochastic distribution and, thus, may change at runtime. In order to cope with differing Quality of Service, we present a heuristic approach for efficiently addressing the Service Selection Problem in conjunction with stochastic Quality of Service attributes. Accounting for penalty cost which accrue due to Quality of Service violations, our approach reduces the impact of stochastic Quality of Service behavior on total cost significantly.**

***Keywords*-Service Selection, Stochastic Quality of Service, Optimization, Simulation**

I. Introduction

The problem of selecting services from a set of functionally appropriate ones and thereby best meeting cost and Quality of Service (QoS) requirements is widely recognized in the literature, e. g., [1]–[4]. It is commonly referred to as *Service Selection Problem (SSP)*. Solutions to the SSP describe execution plans indicating, which services to select for accomplishing the tasks of a workflow in order to satisfy aforementioned cost and QoS requirements. Corresponding optimization approaches for optimizing the SSP are most often based on deterministic QoS attributes.

But QoS, e. g., the *response time* of a service or its *availability*, is not always deterministic in reality. Various research activities provide evidence that fluctuations in the behavior of QoS attributes exist [5]–[7]. Service response times, for instance, may change dynamically due to network latency or server load. Thus, when actually executing an execution plan, the perceived QoS might differ from the expected QoS which has been used previously for computing the corresponding execution plan. Thus, although having computed an optimal solution to the SSP at design time that satisfies the QoS constraints, it might still be possible that these constraints will be violated at runtime.

The work at hand addresses this issue. Based on a service broker scenario, which is presented in Section II, we describe how differing QoS values due to stochastic QoS behavior negatively impact total cost. In order to account for this impact of stochastic QoS, we proposed an adaptation heuristic in our former work in [8] that successfully reduces the negative impact of stochastic QoS behavior on total cost but requires strong computational efforts. Extending our former work and addressing this shortcoming, we propose a *Genetic Adaptation Algorithm* that also achieves cost reductions but with significantly reduced computation times.

The remainder of this work is structured as follows. In Section II, we present a motivating scenario that will be used throughout the paper. In Section III, we formalize the SSP and briefly describe our solution approach for optimizing the SSP in the context of *deterministic* QoS attributes. Since QoS is not always deterministic, we present a heuristic solution approach that accounts for stochastic QoS behavior in Section V and provide corresponding evaluation results in Section VI. Finally, after having distinguished our approach from related work in Section VII, we draw conclusions and discuss future work in Section VIII.

II. Scenario

In this section, we present an example scenario used for illustrating the impact of stochastic QoS attributes. We assume a service broker who receives requests from his/her customers. Paying the broker a fixed amount of money, the customers require certain tasks and workflows, respectively, to be executed. For this, they provide the broker with a document that specifies the required tasks from a functional perspective and indicates the ordering of the tasks, i. e., the structure of the workflow. One of the broker's customers asks, for instance, for the workflow in Figure 1. The process steps PS_i thereby indicate the tasks that have to be accomplished. Each task i can be executed by a single service j.

In addition to these functional requirements, the customers also specify their QoS needs regarding the execution of the respective workflows. For this, they provide restrictions in the form of upper or lower bounds for specific QoS attributes, the so-called Service Level Objectives (SLOs). With this information, the broker tries to select those services among functionally appropriate ones that satisfy

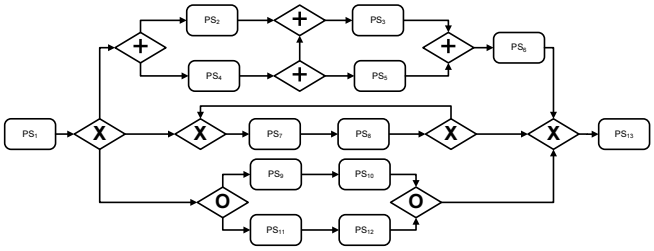


Figure 1: Example Workflow (in BPMN)

the customers' QoS requirements, as a violation of these requirements will be penalized by his/her customers. Having selected respective services, the broker pays and invokes these services in order to execute the customers' tasks and workflows, respectively. If the customers' QoS requirements are violated, penalty fees, which also have to be paid by the broker, will become due. In order to reach an optimized decision, the broker models the selection of services as an optimization problem aiming at minimizing invocation cost and satisfying QoS constraints. For this, the broker formulates an SSP and computes an optimal solution to it, which is described in the following Section III.

III. Service Selection Problem

In order to formulate an SSP and therewith to compute an execution plan, it is necessary to aggregate the QoS and cost values of eligible candidate services according to the considered workflow and its structure, respectively. Regarding our example scenario, this actually is a prerequisite for comparing workflow QoS with respective bounds issued by the broker's customers.

The workflow depicted in Figure 1 consists of an interlaced XOR-block that contains an OR-block, a Repeat loop, and an unstructured *Directed Acyclic Graph (DAG)*. Corresponding aggregation specifications accounting for these structural orchestration blocks would have to be applied according to their actual ordering within the workflow in Figure 1. For further details on aggregation specifications, the interested reader is referred to our former work [8]–[10].

Applying corresponding aggregation specifications according to the workflow depicted in Figure 1 enables the broker to model the SSP as an (potentially non-linear) optimization problem, which can be transformed into a *linear* optimization problem by applying appropriate linearization techniques [11]. Having obtained a linear optimization problem, the broker computes an optimal solution – minimizing service invocation cost and (presumably) satisfying the customers' constraints – by applying Integer Linear Programming from the field of Operations Research [12].

But applying the computed execution plan and invoking corresponding services may lead to violations of the customers' constraints, as the invoked services may show a different behavior at runtime than expected at design time. Thus, accounting for the scenario described in Section II, additional penalties will arise. In order to assess the impact of uncertainty resulting from stochastic QoS behavior *before* actually executing the obtained execution plan and invoking the corresponding services, we propose to *simulate* the execution plan firstly. The applied simulation approach is briefly described Section IV. Making use of this simulation approach, we present our proposed Genetic Adaptation Algorithm in Section V.

IV. Simulation

As previously stated, QoS is not constant but differs in reality, i. e., corresponding services do not always provide their functionality with the same quality resulting in differing QoS behavior. In order to capture this differing QoS behavior, frequencies for observed QoS values (or value ranges) can be derived indicating how often a certain value has been observed in comparison to the total number of measurements. The distribution of the observed frequencies thereby corresponds to a *probability distribution* indicating the relative frequency and therewith the probability at which a certain value has been and will be observed. Thus, in order to address stochastic QoS attributes, the application of probability distributions capturing the probabilities and relative frequencies, respectively, for observing values of those attributes appears appropriate.

As we do not focus on estimating probability distributions but rather on the capability to appropriately account for varying QoS behavior, we use standard probability distributions and corresponding distribution functions in this paper. An approach for estimating QoS probability distributions in the context of Web services and service compositions is, for instance, provided by Zheng et al. in [13].

Accounting for the measurements conducted and results achieved in [5]–[7], the QoS attributes explicitly accounted for in the work at hand are assumed to follow standard distributions as indicated subsequently. Response time r is assumed to follow a Normal Distribution $N(\mu_r, \sigma_r)$, while for availability a, a Triangular Distribution $T(min_a, med_a, max_a)$ is assumed. For throughput d, we consider a Uniform Distribution $U(min_d, max_d)$. This way, we account for different distribution functions so that the evaluation results cannot be attributed solely to dedicated properties of the selected, single probability distribution. For further details on different probability distributions, the interested reader is referred to [14], [15].

For actually simulating the execution of the previously computed execution plan, we use aforementioned distribution functions and conduct a series of simulation runs. For this, we draw in each iteration realizations for each QoS attribute of each selected service using random variables $R_{ij} \sim N(\mu_r, \sigma_r)$, $A_{ij} \sim T(min_a, med_a, max_a)$, and $D_{ij} \sim U(min_d, max_d)$. In addition, depending on the branching probabilities of (X)OR-splits and Repeat loops,

we draw realizations of uniformly distributed random variables $U \sim U(0,1)$ in order to determine the paths to be virtually executed in the current iteration. The realized values of the individual services are aggregated afterwards according to the structure of the considered workflow – applying aforementioned aggregation specifications. This way, we compute the overall QoS for the whole workflow. For further details on the simulation approach, the interested reader is referred to our former work in [8].

V. Genetic Adaptation Algorithm

Making use of the described simulation approach, it is possible to predict potentially occurring QoS violations due to varying QoS values, which lead – referring to the broker scenario presented in Section II – to additional penalties. In order to account for this (negative) impact of stochastic QoS and therewith to reduce total cost comprising invocation cost for invoking the selected services as well as penalty cost accruing due to QoS violation, we propose a heuristic solution method. Since we make use of concepts from Genetic Algorithms [16], [17] implementing the genetic operations *selection*, *mutation*, and *crossover* for continuously reducing total cost, our heuristic solution method will be referred to as *Genetic Adaptation Algorithm* (GAA).

Our GAA is indicated in Algorithm 1 using pseudocode. It is split into five steps. In the first step, the GAA is initialized and an initial solution is computed. In the second step, we apply aggregation specifications to compute the total cost for the current solution comprising invocation and penalty cost. Using this cost, we compare in the third step the current solution with the best solution obtained so far and *decide* how to proceed, i. e., whether to store the current solution as best or temporary solution or to restore the previously known best solution. In the fourth step, we determine the QoS attribute for which the penalty cost has been highest. This QoS attribute becomes the *current QoS attribute*, which is used in the fifth step to compute a new solution.

More details on these steps are provided in the following. Referring to Algorithm 1, the candidate services are firstly sorted in the first step according to their QoS values per process step and per QoS attribute (cf. line 2). Subsequently, a corresponding ranking is computed (cf. line 3). In line 4, we compute an initial solution to the SSP. How to compute an initial solution depends on the configuration of the GAA. For instance, regarding the evaluation presented in Section VI, we applied the optimization approach described in Section III.

In the second step, the computed initial solution is simulated in line 6 and corresponding QoS violations are computed in line 7. In line 8, we compute the total cost by adding up the invocation cost of the current solution and the penalty cost accruing due to determined QoS violations.

Depending on whether the cost of the current solution is lower than the cost of the currently known best solution, we either store the current solution in the third step as new best solution solution (cf. lines 10-12) or, potentially, as temporary solution (cf. lines 17-19) in case we do not *restore* the currently known best solution (cf. lines 14-15). Note that we draw a realization of a uniformly distributed random variable $U \sim U(0,1)$ in line 17 for deciding, whether to store the current solution although it is worse compared to the currently known best solution. Thus, we only store worse solutions in approximately 30% of the cases, as indicated in line 17. Whether we restore a solution depends on the number of temporarily allowed worse solutions. Temporarily accounting for worse solutions enables the algorithm to exit local maxima in order to find better solutions. If the number of allowed worse solutions is reached, the currently known best solution will be restored. Referring to Section VI, we evaluate the impact of the number of temporarily allowed solutions on total cost.

In the fourth step, we compute the penalty cost for each QoS attribute q (cf. lines 23-24) and store the QoS attribute that causes highest penalty cost as *current QoS attribute* (cf. lines 25-28). This way, we determine for which QoS attribute the penalty cost is highest.

Using the *current QoS attribute*, a new solution is computed during the fifth step (cf. line 31). The procedure for computing the new, i. e., the *next* solution, is indicated in Algorithm 2 using pseudocode. Note that the *current QoS attribute* refers to the variable q in Algorithm 2. In line 1 of Algorithm 2, we determine a set of tasks for which we try to improve the values for the *current QoS attribute*. For instance, if highest penalty cost is caused by violating the constraint on *response time* – which then becomes the *current QoS attribute* – we try to improve the workflow response time by exchanging the services currently selected to accomplish the set of tasks determined in line 1 for other services that have lower and therewith better values on response time. In line 2, we determine the genetic operator to be applied, i. e., either *selection*, *mutation*, or *crossover*. For this, we draw a realization of $U \sim U(0,1)$. The values $s \in [0,1)$ and $m \in (s,1]$ thereby serve as thresholds for determining the genetic operator. If, for instance, the value v obtained by drawing a realization of U is lower than s, a *selection* will be performed. If v is between s and m, i. e., $s < v \leq m$, a *mutation* will be carried out. Finally, if $v > m$, we will perform a *crossover*. For the evaluation in Section VI, we set $s = 0.6$ and $m = 0.9$. Using the switch statement in line 3, we route the execution of the algorithm to the right case.

In case, a *selection* needs to be carried out, we determine in line 6 the service number of the service that is currently selected for accomplishing a task $t \in taskList$. We retrieve its rank, i. e., its position in the sorted list of services according to the considered task t and current QoS attribute q, in line 7. Using the service ranking computed in line 2 of Algorithm 1, we determine the number of a candidate service

with a *better* rank in line 8. For deciding, how much better the rank of the new service should be, we use a parameter γ. The value of γ depends on the configuration of the GAA. For the evaluation presented in Section VI, we used a value of $\gamma = 2$. For instance, if the currently selected service for task t has the rank 7 with respect to response time, which indicates that it constitutes the 7^{th} best candidate service for task t regarding response time, the service determined in line 8 will have the rank $7 - 2 = 5$. Thus, we will select the corresponding 5^{th} best candidate service. This way, we (try to) improve the overall workflow response time. Finally, in line 9, the newly identified service will then be set as selected for accomplishing task t. This procedure, i. e., steps 6-9, will be repeated for all tasks in the determined task list.

In case, a *mutation* should be performed, we randomly select services for the tasks $t \in taskList$ by drawing realizations of $U \sim U(0, 1)$ for each task t and multiplying the result with the number of candidate services for this task in line 13. In line 14, we include the newly selected service into the solution for accomplishing task t.

For carrying out a *crossover*, we use the tasks $t \in taskList$ and their corresponding task numbers, respectively, as crossover points. The currently known best solution as well as the solution stored in line 18 of Algorithm 1 will be used as parent solutions. In lines 17 and 20 of Algorithm 2, we initialize a range of task numbers, for which the first swapping should be carried out, by setting the first task number to 0 (cf. line 17) and the last task number to the number of the first task $t \in taskList$ (cf. line 20). Since we need only one *next* solution with which we want to proceed, we take only one child solution from crossing the parent solutions. For deciding whether to use the selected services from the best solution or from the temporarily stored solution, we apply a boolean operator $swap$, which is initialized in line 18. For all task numbers t_i inside the current range, i. e., $startTaskNo \leq t_i < endTaskNo$ (cf. line 21), we either take the selected services from the currently best or from the temporarily stored solution, depending on the current value of $swap$. Afterwards, we set $swap$ to its opposite in line 28 and store the number of the current task t as new start task number in line 29. In the next iteration of the loop starting in line 19, the number of the next task $t \in taskList$ will be set as new end task number (cf. line 20), so that the range of task numbers is updated for the next iteration. Finally, the loop in lines 31-37 makes sure that also the tasks with task numbers t_i between the number of the last task $t \in taskList$ and the number of the last task of the workflow are considered for crossover.

By carrying out either a *selection*, a *mutation*, or a *crossover*, a new solution is determined which will be considered as *next* solution provided by the GAA. Repeating the second step (in Algorithm 1), this new solution is simulated again and its total cost is computed. If the total cost for the new solution is lower than the cost of the best solution, it will be stored as new best solution (third step). Otherwise, it will potentially be stored as temporary solution, or it will be replaced by the currently known best solution. Afterwards, the *current QoS attribute* causing highest penalty cost will be determined (fourth step) and a new solution will be computed (fifth step), which will then be simulated and so on and so forth. Thus, depending on whether further iterations of the described GAA have to be performed, the corresponding steps will be repeated.

Algorithm 1 Genetic Adaptation Algorithm

```
1:  //First step – initialize algorithm
2:  sortedServices = sortServices();
3:  serviceRanking = computeServiceRanking();
4:  solution = computeInitialSolution();
5:  //Second step – determine current cost
6:  sim = simulate(cs);
7:  v = computeQoSViolation(sim);
8:  cost = computInvCost(solution) + computePanalty(v);
9:  //Third step – decide procedure
10: if cost < bestCost then
11:     bestSolution = solution;
12:     bestCost = cost;
13: else
14:     if restoreBestSolution() then
15:         solution = bestSolution;
16:     else
17:         if random() ≤ 0.3 then
18:             storedSolution = solution;
19:         end if
20:     end if
21: end if
22: //Fourth step – determine the current QoS attribute
23: for all q ∈ QoSattributes do
24:     penaltyCost = computePenaltyForAttribute(q);
25:     if penaltyCost ≥ highestPenaltyCost then
26:         highestPenaltyCost = penaltyCost;
27:         currentQoSattribute = q;
28:     end if
29: end for
30: //Fifth step – compute next solution
31: computeNextSolution(currentQoSattribute);
```

Having provided details on the proposed GAA, we evaluate our solution approach in Section VI.

VI. Evaluation

As a proof of concept, we prototypically implemented our GAA using the Java programming language. For evaluating its performance, we conducted a set of experiments. The experiments have been performed on an Intel Core 2 Quad processor at 2.66 GHz, 4 GB RAM, running Microsoft Windows 7. Accounting for our broker scenario in Section II,

Algorithm 2 ComputeNextSolution(q)

```
1: taskList = comuteTaskWeights();
2: goMethod = determineGoMethod(s, m);
3: switch goMethod do
4:     case selection
5:         for all t ∈ taskList do
6:             serviceNo = solution[t];
7:             rank = sortedServices[t,q,serviceNo];
8:             newNo = serviceRanking[t,q,rank − γ];
9:             solution[t] = newNo;
10:        end for
11:    case mutation
12:        for all t ∈ taskList do
13:            newNo = random() · numOfServices(t);
14:            solution[t] = newNo;
15:        end for
16:    case crossover
17:        startTaskNo = 0;
18:        swap = true;
19:        for all t ∈ taskList do
20:            endTaskNo =numberOfTask(t);
21:            for startTaskNo ≤ t_i < endTaskNo do
22:                if swap then
23:                    solution[t_i] = bestSolution[t_i];
24:                else
25:                    solution[t_i] = storedSolution[t_i];
26:                end if
27:            end for
28:            swap = not(swap);
29:            startTaskNo =numberOfTask(t);
30:        end for
31:        for startTaskNo ≤ t_i ≤ numberOfTasks do
32:            if swap then
33:                solution[t_i] = bestSolution[t_i];
34:            else
35:                solution[t_i] = storedSolution[t_i];
36:            end if
37:        end for
```

we considered the workflow depicted in Figure 1 and applied the solution method indicated in Section III for computing optimal solutions to the SSP based on deterministic values, i. e., without accounting for stochastic QoS. For solving the obtained linear optimization problem, we used the linear programming solver CPLEX[1]. This solution approach will be referred to as *Solver*. Computed results have been compared with results obtained by applying the proposed GAA as well as with results obtained by applying the adaptation heuristic *A2ESTD* proposed in our former work in [8]. In brief, A2ESTD removes services with high uncertainties from the list of candidate services.

[1] http://www.ibm.com/software/integration/optimization/cplex-optimizer/

Table I: Assumed values for stochastic QoS attributes

Attribute	PS_{10}, PS_{11}	PS_1-PS_9, PS_{12}, PS_{13}
R_{ij}	$\mu_e \sim U(160, 240)$ $\sigma_e \sim U(0, 40)$	$\mu_e \sim U(80, 120)$ $\sigma_e \sim U(0, 20)$
A_{ij}	$min_a \sim U(0.9, 0.94)$ $med_a \sim U(0.94, 0.98)$ $max_a \sim U(0.98, 1.02)$	$min_a \sim U(0.95, 0.97)$ $med_a \sim U(0.97, 0.99)$ $max_a \sim U(0.99, 1.01)$
D_{ij}	$min_d \sim U(70, 105)$ $max_d \sim U(110, 135)$	$min_d \sim U(70, 105)$ $max_d \sim U(110, 135)$
c_{ij}	$U(0.7, 1.3) \cdot (80 + (0.3 \cdot (\mu_d - \mu_e)) \cdot \mu_a^2)$	$U(0.7, 1.3) \cdot (40 + (0.3 \cdot (\mu_d - \mu_e)) \cdot \mu_a^2)$

In the following, we describe our experimentation setup. As previously stated, we draw realizations of the random variables $R_{ij} \sim N(\mu_r, \sigma_r)$, $A_{ij} \sim T(min_a, med_a, max_a)$, and $D_{ij} \sim U(min_d, max_d)$ for determining QoS values for the candidate services. The parameterization of the random variables R_{ij}, A_{ij}, and D_{ij} is indicated in Table I. We assume the invocation cost of a service to be partly dependent on its QoS, i. e., *good* QoS values in terms of low response time r, high availability a, and high throughput d result in higher invocation cost. Thus, we compute the invocation cost as indicated in Table I, using a uniformly distributed random variable $U(0.7, 1.3)$.

For comparing the results of the aforementioned solution methods, we use *computation time*, i. e., the time it took the applied solution method to compute its results in milliseconds (msec), as well as *total cost*, i. e., the sum of invocation and penalty cost in monetary units, e. g., cents, as *dependent* variables. As *independent* variables, we use the number of iterations that the different adaptation heuristics, i. e., GAA and A2ESTD, have to perform as well as the number of thereby temporarily accepted worse solutions. The number of iterations – referred to as *greed* – will be varied from 0 to 20, step 2. The number of temporarily accepted worse solutions – referred to as *annealing* – will be varied from 0 to 10, step 1. In addition, we account for different levels of penalty fees as third independent variable. In this respect, we assume linear penalty fees per *unit* of QoS violation, e. g., cents per second the actual workflow response time has been higher than restricted by the corresponding bound, or cents per percent point the availability was lower than restricted. We could have alternatively used variable penalty fees that increase quadratically or exponentially with the size of the violation. Since this would have only influenced the calculation of the actual penalty cost depending on the chosen penalty cost model but does not change our approach, we sticked to linear penalty fees for the sake of simplicity. In this respect, we varied the *penalty fee percentage* from 0.0 to 0.2, step 0.2, i. e., from 0% to 20%. The resulting penalty fees are computed by multiplying the absolute QoS violation per QoS attribute with the product of penalty fee percentage and invocation cost. Thus, the absolute penalty cost is dependent on the current invocation

cost. As previously stated, we could have considered other cost models but finally sticked to the described cost model for simplicity reasons. For the evaluation, we chose a partial factorial design and fixed the values for the independent variables *greed*, *annealing*, and *penalty fee percentage* to 10, 2, and 0.1, if not currently varied. The evaluation results are depicted in Figure 2 and Figure 3.

Regarding the impact of the number of iterations on total cost, the evaluation results depicted in Figure 2a show that both, GAA and A2ESTD, achieve remarkable cost reductions. While the solution computed by applying the Solver without accounting for stochastic QoS causes total cost of 408.8 cent, solutions obtained after having performed 20 iterations of GAA and A2ESTD only cause 371.8 and 339.0 cent, respectively, which corresponds to a reduction by 9.1% and 17.1%. Thus, the broker could save up to 9.1% and 17.1% of the total cost when applying the corresponding adaptation heuristic. But, as Figure 3a reveals, this reduction in total cost acually "costs" additional computation time. While the Solver requires 143.1 msec for computing its solution, GAA and A2ESTD need 914.6 and even 6075.4 msec, i. e., up to 6.4 and even 42.3 times as much, for performing aforementioned 20 iterations.

Regarding the impact of the number of temporarily allowed worse solutions depicted in Figure 2b, again both, GAA and A2ESTD, achieve cost reductions compared to the Solver – up to 8.6% and 20.2%, respectively. But these cost reductions do not increase monotonously with higher values of *annealing*, as the number of iterations to be performed does not depend on the value of *annealing* but on the value of *greed*, which has been fixed to 10 in this case. Similar to the results for varying *greed*, higher values of annealing lead to higher computation times. While GAA requires up to 1456.9 msec, A2ESTD needs up to 7885.6 msec as indicated in Figure 3b, which corresponds to an increase by a factor of 10.2 and 55.1. Note that the total cost the Solver achieved as well as the therefore needed computation time is not impacted, neither by *greed* nor by *annealing*. Thus, neither total cost nor computation time changed.

Finally, increasing the penalty fee percentage results in higher penalty fees and, thus, in higher total cost. While the total cost achieved by the Solver amounts up to 570.7 cent, GAA and A2ESTD achieve 408.1 and 395.5 cent for a penalty fee percentage of 0.2, which corresponds to a reduction by 28.4% and 30.7%, respectively. Since the accruing penalty costs depend on performed simulation results for which we drew realizations of random variables (cf. Section V), they do not increase linearly with higher penalty fee percentages. Thus, we do not observe a linear increase in total cost. Since neither the GAA nor A2ESTD can find better solutions than the Solver for a penalty fee percentage of 0.0, i. e., for a scenario without penalties, they cannot reduce total cost. In fact, only for penalty fee percentages > 0, they start adapting. Thus, for a penalty fee percentage of 0.0, their computation times are nearly the same as for the Solver. Further, it can be observed that the computation time for GAA and A2ESTD does not increase remarkably with growing penalty fee percentages, because the number of iterations to be performed does not depend on the penalty fee percentage. While the GAA requires 516.9 msec in average, A2ESTD needs 3173.3 msec in average, which corresponds to an increase in computation time by a factor of 3.6 and 22.2, respectively.

Summing up, it turns out that GAA also achieves cost reductions, but they are not as high as the reductions A2ESTD achieves. On the other hand, the proposed GAA requires less (additional) computation time compared to A2ESTD, which actually was the goal for its development.

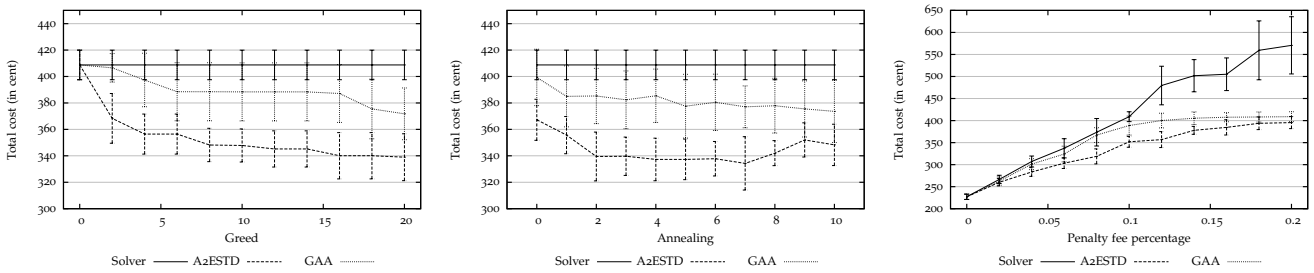
Having described our approach, we discuss related approaches in the following Section VII.

VII. Related Work

As previously stated, the SSP is well-recognized in the literature based on deterministic QoS attributes. A survey of current approaches can be found in [4]. In principle, current approaches can be divided into two categories: heuristic approaches which try to find rather good solutions within a reduced amount of computation time, e. g., [2], [18], [19], and approaches aiming at finding an optimal solution to the SSP, e. g., [1], [3], [20]. All those approaches presume that the use QoS attributes are deterministic. Related work in the area of stochastic QoS attributes, however, is rather sparse.

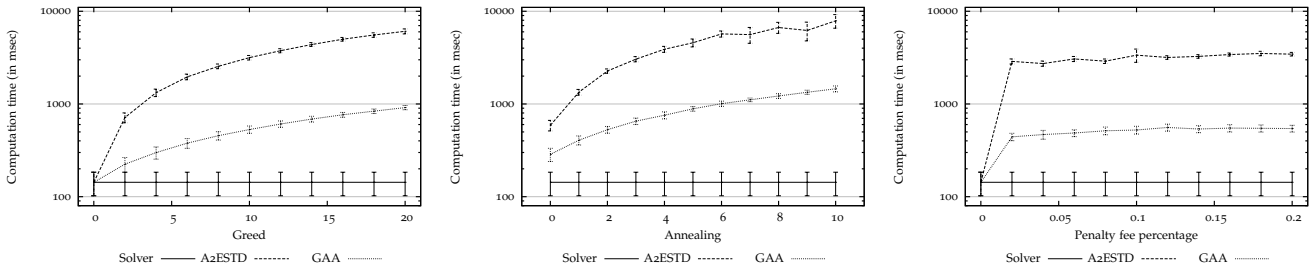
Rosario et al. consider probabilistic QoS, but not for the purpose of services selection [5]. They rather focus on SLA and contract composition, respectively, using *soft probabilistic contracts*. Hwang et al. use Probability Mass Functions (PMFs) instead of deterministic QoS values [21]. Providing approaches for aggregating the PMFs of single services, the authors aim at computing and estimating QoS for service-based workflows using a preselected set of services with discrete PMFs. Thus, they also do not address service selection. In their work in [22], Li et al. use historic data for predicting QoS values, which they then use for service selection instead of predefined values guaranteed by service providers. Thus, in effect, the authors use (predicted) deterministic values.

Cardellini et al. use α-percentiles (with $\alpha = 95\%$) instead of deterministic values for the QoS attribute *response time* [23]. Accordingly, the corresponding restriction on response time demands the probability of violating the bound for *response time* to be lower or equal to $1-\alpha$, i. e., $1-0.95 = 5\%$. Projected to our broker scenario, this means that the broker can assume satisfying the respective bound with a probability of 95%. But Cardellini et al. do not account for penalty costs accruing due to QoS violations in the remaining 5% of the cases. Depending on the ratio between invocation and penalty cost, it could be beneficial for the broker to select rather cheap services, which statistically cause QoS



(a) Impact of greed (b) Impact of annealing (c) Impact of penalty

Figure 2: Evaluation results regarding total cost



(a) Impact of greed (b) Impact of annealing (c) Impact of penalty

Figure 3: Evaluation results regarding computation time

violations more often, and paying the corresponding penalty cost rather than selecting expensive services for which the probability of violating QoS constraints is low. In contrast to Cardellini et al., out approach accounts for such situations.

In [24], Leitner et al. assume fixed service compositions and fixed sets of potential adaptations such as "use *Express Shipping*" instead of " use *Standard Shipping*". The authors thereby aim at selecting and applying adaptations in order to minimize total cost comprising invocation cost, penalty cost for QoS violation, and cost for applied adaptations. Abstracting from the term *adaptation* and interpreting available adaptations as alternative services, it can be stated that Leitner et al. are solving an SSP with the aim to minimize total cost considering penalties for QoS violation. Similar to Li et al., Leitner et al. predict prospective QoS values – and therewith expected QoS violation. In contrast to the work at hand, Leitner et al. estimate the impact of stochastic QoS behavior for each service separately and perform an optimization with these estimated, deterministic QoS values. Our approach does not consider stochastic QoS behavior of services independently from each other, but account for the whole workflow using the described simulation approach. Thus, potential reverse QoS deviations of different services from expected behavior can be considered, which is not possible in [24] due to their isolated consideration of expected QoS per service.

In summary, our approach extends related work as it considers on the one hand the impact of QoS violation in terms of accruing penalty costs. On the other hand, we do not only regard isolated stochastic QoS behavior for individual services, but account for probably compensating reverse QoS deviations of different services. Thus, we consider the impact of stochastic QoS behavior for the whole workflow. Conclusions are drawn in the following Section VIII.

VIII. Conclusion

The SSP is well-recognized in the literature based on deterministic QoS parameters. In the work at hand, we addressed the SSP in conjunction with stochastic QoS attributes which has been considered as yet only insufficiently in the literature. In our former work in [8], we proposed an adaptation heuristic – referred to as A2ESTD – that successfully reduces the negative impact of stochastic QoS behavior on total cost but requires strong computational efforts. Addressing this shortcoming, we proposed in this paper a *Genetic Adaptation Algorithm (GAA)* that also achieves cost reductions but with significantly reduced computation times. For instance, referring to Figure 2c, the GAA achieves cost reductions up to 28.4% whereas A2ESTD achieves reductions up to 30.7%. On the other hand, referring to Figure 2c, computation times increased only by a factor of 3.6 when applying GAA, whereas A2ESTD leads to an increase in computation time by a factor of 22.2. Thus, the

goal of achieving cost reductions with a significantly reduced computation time compared to our former work has been achieved. In this respect, it has to be noted that the provided absolute results strongly depend on the considered scenario. However, the proposed approach appears appropriate to reduce potentially negative impacts of stochastic QoS.

In our future work, we aim at merging the proposed GAA with the optimal solution approach in order to consider stochastic QoS during the optimization.

ACKNOWLEDGMENT

This work is supported in part by the Commission of the European Union within the ADVENTURE FP7-ICT project (Grant agreement no. 285220) and by E-Finance Lab e. V., Frankfurt am Main, Germany (http://www.efinancelab.com).

REFERENCES

[1] D. Ardagna and B. Pernici, "Adaptive Service Composition in Flexible Processes," *IEEE Transactions on Software Engineering*, vol. 33, no. 6, pp. 369–384, 2007.

[2] D. A. Menascé, E. Casalicchio, and V. K. Dubey, "A Heuristic Approach to Optimal Service Selection in Service Oriented Architectures," in *International Workshop on Software and Performance (WOSP 2008)*, 2008, pp. 13–24.

[3] A. F. M. Huang, C.-W. Lan, and S. J. H. Yang, "An Optimal QoS-based Web Service Selection Scheme," *Information Sciences*, vol. 179, no. 19, pp. 3309–3322, 2009.

[4] A. Strunk, "QoS-Aware Service Composition: A Survey," in *European Conference on Web Services (ECOWS 2010)*, 2010, pp. 67–74.

[5] S. Rosario, A. Benveniste, S. Haar, and C. Jard, "Probabilistic QoS and Soft Contracts for Transaction-Based Web Services Orchestrations," *IEEE Transactions on Services Computing*, vol. 1, no. 4, pp. 187–200, 2008.

[6] Z. Zheng, Y. Zhang, and M. R. Lyu, "Distributed QoS Evaluation for Real-World Web Services," in *International Conference on Web Services (ICWS 2010)*, 2010, pp. 83–90.

[7] U. Lampe, A. Miede, T. Lusa, S. Schulte, R. Steinmetz, and S. Dustdar, "An Analysis of Anonymity Side Effects in the Internet of Services," in *International Conference on Networked Systems (NetSys 2013)*, 2013, pp. 51–58.

[8] D. Schuller, U. Lampe, J. Eckert, R. Steinmetz, and S. Schulte, "Cost-driven Optimization of Complex Service-based Workflows for Stochastic QoS Parameters," in *International Conference on Web Services (ICWS 2012)*, 2012, pp. 66–74.

[9] D. Schuller, A. Polyvyanyy, L. García-Bañuelos, and S. Schulte, "Optimization of Complex QoS-Aware Service Compositions," in *International Conference on Service Oriented Computing (ICSOC 2011)*, 2011, pp. 452–466.

[10] D. Schuller, "QoS-Aware Service Selection – Optimization Mechanisms and Decision Support for Complex Service-based Workflows," Ph.D. dissertation, Technische Universität Darmstadt, 2013.

[11] D. Schuller, A. Miede, J. Eckert, U. Lampe, A. Papageorgiou, and R. Steinmetz, "QoS-based Optimization of Service Compositions for Complex Workflows," in *International Conference on Service Oriented Computing (ICSOC 2010)*, 2010, pp. 641–648.

[12] H. Taha, *Operations Research – An Introduction*, 8th ed. Upper Saddle River, NJ, USA: Prentice Hall, 2007.

[13] H. Zheng, J. Yang, and W. Zhao, "QoS Probability Distribution Estimation for Web Services and Service Compositions," in *International Conference on Service-Oriented Computing and Applications (SOCA 2010)*, 2010, pp. 1–8.

[14] R. Jain, *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*, ser. Wiley Professional Computing. Hoboken, NJ, USA: John Wiley & Sons, 1991.

[15] M. P. McLaughlin, "A Compendium of Common Probability Distributions," last access on 2014-01-31. [Online]. Available: http://www.causascientia.org/math_stat/Dists/Compendium.html

[16] D. Whitley, "A Genetic Algorithm Tutorial," *Statistics and Computing*, vol. 4, no. 2, pp. 65–85, 1994.

[17] M. Mitchell, *An Introduction to Genetic Algorithms*. Cambridge: MIT Press, 1996.

[18] F. Lécué, "Optimizing QoS-Aware Semantic Web Service Composition," in *International Semantic Web Conference (ISWC 2009)*, 2009, pp. 375–391.

[19] N. B. Mabrouk, N. Georgantas, and V. Issarny, "A Semantic End-to-End QoS Model for Dynamic Service Oriented Environments," in *International Workshop on Principles of Engineering Service-Oriented Systems (PESOS 2009)*, 2009, pp. 34–41.

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

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

[22] M. Li, J. Huai, and H. Guo, "An Adaptive Web Services Selection Method Based on the QoS Prediction Mechanism," in *International Conference on Web Intelligence (WI 2009)*, 2009, pp. 395–402.

[23] V. Cardellini, E. Casalicchio, V. Grassi, and F. L. Presti, "Adaptive Management of Composite Services under Percentile-Based Service Level Agreements," in *International Conference on Service Oriented Computing (ICSOC 2010)*, 2010, pp. 381–395.

[24] P. Leitner, W. Hummer, and S. Dustdar, "Cost-Based Optimization of Service Compositions," *IEEE Transactions on Services Computing*, no. PrePrints, pp. 1–14, 2011.