Improved Adaptation of Web Service Compositions Using Value of Changed Information

Girish Chafle[1], Prashant Doshi[2], John Harney[2], Sumit Mittal[1] and Biplav Srivastava[1]
[1]IBM India Research Laboratory
New Delhi, India 110016
{*cgirish,sumittal,sbiplav*}*@in.ibm.com*

[2]LSDIS Lab, Dept. of Computer Science
University of Georgia, Athens, GA 30602
{*pdoshi,jfh*}*@cs.uga.edu*

Abstract

Workflows often operate in volatile environments in which the component services' QoS changes frequently. Optimally adapting to these changes becomes an important problem that must be addressed by the Web service composition and execution (WSCE) system being utilized. We adopt the $\mathcal{A}$-WSCE framework that utilizes a three-stage approach for composing and executing Web workflows. The $\mathcal{A}$-WSCE framework offers a way to adapt by defining multiple workflows and switching among them in case of component failure or changes in the QoS parameters. However, the $\mathcal{A}$-WSCE framework suffers from the limitations imposed by a simple strategy of periodically checking the QoS offerings of randomly picked providers in order to decide whether the current workflow is optimal. To address these limitations, we associate the value of changed information (VOC) with each workflow and utilize the VOC to update which workflow to execute. We empirically demonstrate the improved performance of the workflows selected using the new approach in comparison to the original framework.

1 Introduction

Business environments in which Web services must function are often volatile. Both, the input data such as the quality-of-service (QoS) parameters as well as the execution data may change during the life time of a business process [10]. We label this as *data volatility*. Additionally, the workflow components may themselves fail or exhibit unexpected behavior, and we call this as *component volatility*. Both data and component volatility may adversely affect the performance of a Web service composition over time, resulting in sub-optimal workflows.

As a concrete example, consider a supply chain in which two suppliers compete for orders from a large manufacturer. The sequence in which an optimized manufacturer uses the services of the two suppliers depends on the probability with which the suppliers usually satisfy the orders and the cost of using them. If the preferred supplier's rate of order satisfaction (ie. availability) drops suddenly (due to unforeseen circumstances), a cost-conscious manufacturer should replace it with the other to remain optimal.

Often, our primary aim is to compose correct and optimal workflows and deploy them; adaptation of workflows is viewed as an afterthought. Following this line of research, previous methods of adaptation revolved around monitoring or querying the business environment for changes [2], gauging the impact of these changes on the existing workflows using sophisticated techniques [7] and adjusting the executing workflows to reflect these changes [5]. While these methods offer ways of adapting previously composed workflows, they do not consider potential overheads of adaptation while formulating the workflows.

In this paper, we propose a shift in paradigm by considering the possibility of adaptation while composing the workflow to execute. In this regard, we use the adaptive Web service composition and execution ($\mathcal{A}$-WSCE) framework [4]. The $\mathcal{A}$-WSCE framework exploits redundancy as a way to adapt workflows by identifying multiple diverse workflows that achieve the same goal. Workflows that provide optimal QoS performance are then selected for execution. The data and component volatility is accommodated by switching to alternate workflows using a staged procedure. However, in order to ascertain the changes in the QoS parameters of the Web services, the $\mathcal{A}$-WSCE framework utilizes a simple strategy of frequently querying all the service providers for their new QoS parameters. This imposes

two limitations on the performance of $\mathcal{A}$-WSCE: (a) Some of the queries may not yield new information that is significant enough to effect a change of the optimal process. This drawback becomes more potent when the queries are themselves expensive to perform. (b) The new information may result in possibly costly switches of the process without significant gains in the QoS values of the process.

We address these limitations by utilizing statistical models of volatility of the parameters and the *value of changed information* (VOC) [7] mechanism to analyze the expected impact of the volatility on candidate business workflows. We utilize the VOC associated with the executing workflow to test whether we should query the service providers participating in the workflow for their revised QoS parameter values. In particular, large VOC values likely signify that the current workflow is suboptimal and we may expect significant changes in the QoS of the workflow if we query for revised information. We empirically evaluate the $\mathcal{A}$-WSCE framework augmented with VOC in a simulated volatile environment and compare the QoS of the resulting workflows, with the previous approach and with the approach of randomly selecting some service providers to query.

2 Background: $\mathcal{A}$-WSCE System

Web service composition and execution is the process of realizing the requirements of a new Web service using existing component services, and running the composite service on an execution infrastructure. The requirements are specified by an end–user, and can be decomposed into two parts: The first part deals with the desired functionality of the composite service, called the *functional requirements*. The second part involves specification of the *non-functional requirements* that relate to issues like performance and availability. The adaptive Web service composition and execution ($\mathcal{A}$-WSCE) problem can be defined as follows: Given the specifications of a new service, create and execute a workflow that satisfies the functional and non-functional requirements of the service, while being able to *continually* adapt to dynamic changes in the environment.

Figure 1 gives an overview of the $\mathcal{A}$-WSCE system presented in [4] built over the Synthy service composition approach [1]. In Synthy, we differentiate between Web service *types*, which are groupings of similar (in terms of functionality) Web services, and the actual Web service *instances* (which are the providers) that can be invoked. As shown in Fig. 1, the **Logical Manager** and the **Physical Manager** are configured with details of available Web service types and instances, respectively. The **Runtime Manager** maintains a registry that receives information about QoS changes and/or failures of instances through a monitoring component.

During the composition process, the Logical Manager invokes the `Template Generator` to create K workflow templates that meet the functional requirements of the desired service. These templates are stored by the Physical Manager and instantiated into executable workflows by selecting an appropriate service instance for each service type in a template. The selection process is based on optimization of the non–functional requirements of the composite service, and is performed by the `Workflow Generator` in the Physical Manager. L executable workflows are passed from the Physical Manager to the Runtime Manager, which stores them and invokes `Runtime Selector` to select the one with the best overall QoS.

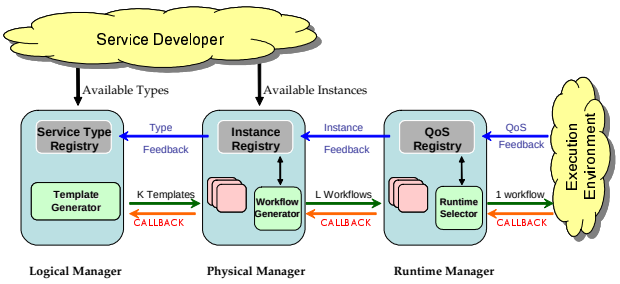

Figure 1. The staged architecture of the $\mathcal{A}$-WSCE system.

To make the system sensitive to changes in the environment, there is a *feedback channel* associated with each stage. For example, the execution environment provides periodic feedback about monitored QoS values of instances to the Runtime. The QoS changes are aggregated and sent periodically by the Runtime Manager to the Physical Manager. Also, if the instances of a particular service type are unavailable for a certain period, the information is passed by Physical Manager to Logical Manager. This ensures that this type is not used by the Template Generator in subsequent template creations. Whenever the failed instance(s) recovers, the information is fed back to the previous stages, thereby qualifying them again for inclusion in the composition process. Finally, information about availability of new service types and instances is fed into the *Type* and *Instance* registries, respectively.

The system also provides a *callback mode* for situations where all available workflows at a given stage have failed. This mode is used by the Runtime Manager to invoke the Physical Manager if no available workflows can be executed (due to failure of instances and/or QoS violations). It is also used by the Physical Manager to request the Logical Manager for generating new templates, if none of the available workflows can be concretized to executable ones.

The adaptation algorithm presented in [4] incorporates different changes in the environment at different stages. The monitoring infrastructure sends information about QoS changes and instance failures to the run-time stage every

T_M interval. The Runtime Manager runs every T_R interval to incorporate the QoS changes and select the best workflow to execute. Physical Manager, on the other hand, is invoked every T_P period to generate a fresh set of executable workflows which are then passed to the Runtime Manager. Finally, the Logical Manager creates new templates every T_L period. We believe that in a realistic setting, $T_M < T_R < T_P < T_L$.

The $\mathcal{A}$-WSCE system exploits redundancy as a way to adapt workflows by identifying multiple diverse workflows that achieve the same goal. The key idea is to increase the number of choices by generating multiple (feasible) workflows at different stages of the system that realize the same composition in terms of functional or non–functional requirements. This is accomplished by the following:

• Generation of *multiple templates* at the Logical Manager. Templates (abstract workflows) can be created either manually, or by using contingent planning techniques that work on semantically annotated Web services [1].

• Generation of *multiple executable workflows for a template*. $\mathcal{A}$-WSCE has an optimization framework [1] built on the QoS model and mathematical formulation proposed by Zeng et al. [13]. This framework helps the Physical Manager generate an executable workflow for a template by selecting an instance for each type in the template. The basic idea is to optimize QoS while meeting the constraints specified for each of the QoS dimensions (cost, response time, and availability).

• Selection of *workflows from different templates*. Having generated multiple workflows for each template, Physical Manager now selects L workflows from this set. The aim is to maximize the total sum of the QoS values of the selected workflows, while ensuring that each template gets a minimum representation in the selected set. Providing minimum representation for each template can help in providing more flexibility for selection of workflow to execute at the runtime stage.

3 Value of Changed Information (VOC)

Several characteristics of the service providers participating in a workflow may change. For example, in a supply chain scenario, the cost of using a preferred supplier's services may increase, and/or the availability of the services may decrease. Not all updates of the parameters lead to changes in the overall composition of the workflow. In addition, the change effected by the revised information may not be worth the cost of adapting the workflow. Typically, the cost of adaptation includes *the costs of obtaining the information, switching services providers and recomputing the workflow plan or switching between plans*, among others. For example, a manufacturer may have to subscribe to the information providing services of suppliers to obtain their updated QoS parameters, at a cost. In light of these arguments, the VOC mechanism introduced in [7] provides a method for measuring the *expected* impact of the changed information on the workflow.

While the approach is applicable to any model based workflow composition technique, in this paper we use it in the context of the composition technique adopted by the $\mathcal{A}$-WSCE framework introduced previously. As we mentioned before, the $\mathcal{A}$-WSCE uses an abstract workflow composed of service types as its point of departure. It utilizes a linear program to selectively bind those instances to the service types that maximize the following expression:

$$\mathcal{V} \stackrel{def}{=} w_c \times V_c + w_r \times V_r + w_a \times V_a \qquad (1)$$

where: w_c, w_r and w_a are the importance weights – ranging from 0 to 1 – of the QoS parameters, cost, service response times and availability, respectively; V_c, V_r, and V_a are the total costs, response times and availability rates, normalized between 0 and 1.

Let $\mathcal{V}_{cur}$ denote the total QoS value (Eq. 1) of the currently executing workflow, and $\mathcal{V}_{new}$ be the total QoS value of the workflow that is obtained by maximizing Eq. 1 given revised information about a QoS parameter, say cost, of a service instance (provider) participating in the original workflow. Since the actual revised cost is not known unless we query the service provider, computing the VOC involves averaging over all possible values of the QoS parameter, using the belief distributions over the values. These distributions may be provided by the service providers through predefined service-level agreements or they could be learned from previous interactions with the service providers. The expected value of change in the current workflow due to revised information about the QoS parameter, say cost, of instance i is formulated as:

$$VOC_i^C \stackrel{def}{=} \int_c Pr(C_i = c)[\mathcal{V}_{new} - \mathcal{V}_{cur}]\, dc \qquad (2)$$

The superscript to VOC, C, denotes the revised information inducing the change, C_i denotes the cost parameter of service instance i. Intuitively, Eq. 2 represents how badly, on average, the current workflow performs in the changed environment as statistically modeled by the probability distribution, $Pr(C_i)$. The VOC formulation for other QoS parameters such as response times and availability rates is analogous.

Our VOC formulation adopts a myopic approach to information revision, in which revised information about a single provider is considered at a time. Notice that in order to calculate the VOC, we must compute the revised values, $\mathcal{V}_{new}$ and $\mathcal{V}_{cur}$, for all possible $\mathbf{c}$ and average over their difference based on our distribution over $\mathbf{c}$. Computing $\mathcal{V}_{cur}$, which represents the total QoS value of the current workflow is straightforward; it involves evaluating

Eq. 1 for the instances in the current workflow. However, the revised value $\mathcal{V}_{new}$ is computed by finding the optimal workflow given the revised QoS parameter; this is done using linear programming. Note that VOC is guaranteed to be non-negative for a workflow. For the proof see [7].

We use VOC in the context of $\mathcal{A}$-WSCE for gauging if possibly revised information is sufficiently valuable to obtain. For example, if the VOC due to revised cost of using the preferred supplier is greater than the cost of switching workflows and querying the preferred supplier for the information, obtaining the revised cost is expected to be worthwile. We show how to augment $\mathcal{A}$-WSCE with VOC next.

4 $\mathcal{A}$-WSCE with VOC

To incorporate VOC in the context of the $\mathcal{A}$-WSCE system, we first need to compute the aggregate normalized VOC value of an executable workflow, and then utilize this value in the adaptation.

4.1. Calculation of Aggregate, Normalized VOC

The QoS model we use in the $\mathcal{A}$-WSCE system has three dimensions – cost, availability and response time.[1] We begin by aggregating the VOC over the entire workflow for each of the QoS dimensions. This results in a single value for each workflow that signifies how much the entire workflow is expected to change, given changes in the QoS dimension. To compute the aggregate VOC value for a workflow, we first aggregate and normalize the VOC values for each QoS dimension, followed by a weighted sum of these values in accordance with the relative weight (importance) assigned to these dimensions.

4.1.1 Aggregation

Let Q^C_{VOC} denote the aggregate VOC value of cost for a composite workflow. We define:

$$Q^C_{VOC} = \sum_{i \in W} VOC^C_i \tag{3}$$

where VOC^C_i is the VOC value for each instance i participating in the workflow W, calculated according to Eq. 2.

We define similar equations for Q^A_{VOC} (availability) and Q^R_{VOC} (response time):

$$Q^A_{VOC} = \sum_{i \in W} VOC^A_i \tag{4}$$

[1]The model can be easily extended to incorporate additional dimensions.

and,

$$Q^R_{VOC} = \sum_{i \in W} VOC^R_i \tag{5}$$

4.1.2 Normalization

On calculating the aggregated VOC value for each dimension, we normalize these values over the workflow. We do this to allow a comparison based on VOC across the different workflows. Let V^C_{VOC} represent the normalized VOC value of cost for a workflow. We define,

$$V^C_{VOC} = \begin{cases} (Q^C_{VOC} - Q^{C,min}_{VOC})/(Q^{C,max}_{VOC} - Q^{C,min}_{VOC}) \\ \qquad \text{if} \quad Q^{C,max}_{VOC} - Q^{C,min}_{VOC} \neq 0 \\ 1 \\ \qquad \text{if} \quad Q^{C,max}_{VOC} - Q^{C,min}_{VOC} = 0 \end{cases} \tag{6}$$

where $Q^{C,min}_{VOC}$ and $Q^{C,max}_{VOC}$ are the minimum and maximum values of Q^C_{VOC}, the aggregated VOC value of cost for the workflow. To compute the minimum value, we find out the minimum VOC value of each type in the workflow (among all instances of that type) and aggregating these values. The maximum value is calculated similarly – by finding out the maximum VOC value of each type in the workflow.

We define similar terms for availability and response time:

$$V^A_{VOC} = \begin{cases} (Q^A_{VOC} - Q^{A,min}_{VOC})/(Q^{A,max}_{VOC} - Q^{A,min}_{VOC}) \\ \qquad \text{if} \quad Q^{A,max}_{VOC} - Q^{A,min}_{VOC} \neq 0 \\ 1 \\ \qquad \text{if} \quad Q^{A,max}_{VOC} - Q^{A,min}_{VOC} = 0 \end{cases} \tag{7}$$

$$V^R_{VOC} = \begin{cases} (Q^R_{VOC} - Q^{R,min}_{VOC})/(Q^{R,max}_{VOC} - Q^{R,min}_{VOC}) \\ \qquad \text{if} \quad Q^{R,max}_{VOC} - Q^{R,min}_{VOC} \neq 0 \\ 1 \\ \qquad \text{if} \quad Q^{R,max}_{VOC} - Q^{R,min}_{VOC} = 0 \end{cases} \tag{8}$$

4.1.3 Weighing

Once the normalized, aggregated VOC values for each QoS dimension have been calculated, the aggregate VOC of the workflow across all dimensions is obtained by weighting the VOC values for each dimension in accordance with the associated importance weight. Let w_c, w_a and w_r represent

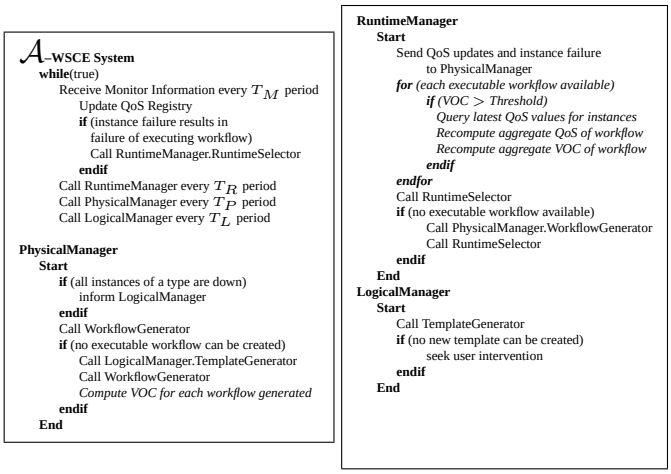

Figure 2. $\mathcal{A}$-WSCE with VOC: Algorithm for Adaptation

the weights for cost, availability and response time respectively. The aggregate VOC for the workflow V_{VOC} is:

$$V_{VOC} = w_c \times V^C_{VOC} + w_r \times V^R_{VOC} + w_a \times V^A_{VOC} \quad (9)$$

4.2. Modified $\mathcal{A}$-WSCE Algorithm

Figure 2 gives the adaptation algorithm for the $\mathcal{A}$-WSCE system enhanced with VOC. The changes made to the original algorithm to account for VOC is shown in *italics*. As in the original system, the Logical Manager generates abstract workflows that are passed to the Physical Manager. The Physical Manager, in turn, outputs executable workflows that are run by the Runtime Manager. Apart from optimizing the end-to-end QoS, the Physical Manager also computes the aggregate VOC for each workflow, in accordance with Eqs. 3 through 9. The Runtime Manager uses this VOC to determine whether to obtain the latest QoS of workflow instances. If it decides to query, it fetches the latest QoS values of instances, and recomputes the aggregate QoS of the workflow along with the VOC value. This value is used in the subsequent iterations to compare against the threshold. The feedback mechanisms from Runtime to Physical and from Physical to Logical, and the callback mode used when all available workflows at a stage have failed, remain the same as the original $\mathcal{A}$-WSCE system.

5 Evaluation

We evaluate the effectiveness of VOC for guiding the adaptation in $\mathcal{A}$-WSCE in comparison with the previous approach utilized in $\mathcal{A}$-WSCE. Our main goal is to study how VOC provides stability to the $\mathcal{A}$-WSCE system in terms of

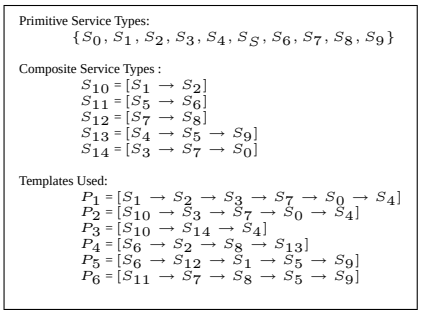

Figure 3. Service Types and Templates

workflow switches while intelligently querying for changes in the environment. To this effect, we conducted experiments in a simulated set-up with the empirical parameters chosen keeping in mind the dynamics of a real execution environment. We studied the performance – stability and the overall aggregate QoS [2] – of the following systems:

- *$\mathcal{A}$-WSCE with always query*: This is the original $\mathcal{A}$-WSCE system which queries all the instances often for the latest QoS values during a workflow execution. The algorithm for adapting to changes in the environment remains the same as in the original system [4].

- *$\mathcal{A}$-WSCE with VOC*: This system incorporates VOC for guiding adaptation in WSCE. As outlined in Section 4, the adaptation algorithm uses the aggregate VOC of a workflow at runtime to decide whether to query the latest QoS values of the instances in the workflow, by comparing against a threshold. Essentially, the idea is to query intelligently by estimating the relationship between the cost of adaptation and the gains expected.

- *$\mathcal{A}$-WSCE with random query*: In the runtime stage, the system decides whether to query the latest QoS values for instances in a workflow with a probability of 0.5. If the decision is to query, it fetches the latest QoS values of instances, computes the new aggregate QoS of the workflow and use it to select the best workflow to execute.

As part of the empirical analysis, we answer the following questions: (1) How does querying using VOC help in improving the system stability (manifested in the number of workflow switches by the Runtime Manager) and improving the aggregate QoS? (2) What is the effect of adaptation cost on the performance of the above mentioned $\mathcal{A}$-WSCE systems?

[2] A workflow executed in the run-time stage has two cost components - the cost of querying instances for their latest QoS values, and the cost of invoking the instances themselves. Consequently, both the query cost and the instance invocation cost are made a part of the aggregate QoS.

5.1 Experimental Setup

For our experiments, we consider the problem of composing and executing a Web service S, whose functional and non–functional requirements have been specified.

Types and Instances: For ease of simulation, we assume that there are 15 distinct service types (functionalities) in the domain – some are primitive types, while the others are composite types (i.e. obtained from functional composition of primitive types). Each service type has 8 different instances that are currently deployed. Each instance of a particular service type S_i has a response time that follows a normal distribution with mean μ^i_{rt} and variance σ^i_{rt}. Similarly, the cost is assigned using a normal distribution with mean μ^i_{cost} and variance σ^i_{cost}. Finally, the availability of an instance is assigned using an exponential distribution with μ^i_{av}. The μ_i's for a type are chosen randomly from a predetermined range, i.e. [$1000-$1400], [100ms–200ms], and [0.6–0.9] for cost, response time, and availability, respectively[3]. The values for σ^i_{cost} and σ^i_{rt} are set to $\$500$ and 50ms, respectively. We incorporated query cost in the system as a fraction of the cost of a service instance. The values of query cost used in our experiments range from 0 to 30% of the actual cost.

Templates and Executable Workflows: From the functional specification, we first generate six templates (abstract workflows) for the composite service S as shown in Figure 3. In our simulation setup, templates are limited to be sequential. For workflows that have tasks executing in parallel or have conditional branches, the computation of aggregate VOC has to be done similar to the one for cost in [13]. The templates are used to stitch together executable (composite) workflows that satisfy the non–functional requirements, and passed to the Runtime Manager. The Runtime Manager executes one of them at a given point in time. Periodically, the system performs *health* checks at each stage of composition, to ensure that the QoS commitments are met. In case of a failure or QoS violations, the system takes corrective measures (according to the corresponding adaptation algorithm) and replaces the currently executing workflow with a potentially better one.

Runtime events: To evaluate the performance of the systems, we periodically inject "QoS change" events at runtime. We randomly pick a fraction of the total instances ($f\%$) and change their QoS values, according to the distributions specified earlier. Finally, the systems schedule periodic events that instruct different Managers to perform health checks. Note that, the periodicity is chosen to be $50T$ (T_R), $250T$ (T_P), and $1440T$ (T_L), for the Runtime, Physical, and Logical Managers, respectively (where T corresponds to the duration of a clock tick in the simulation).

[3]The mean value of a composite type is obtained by aggregating the mean values of its primitive components [3].

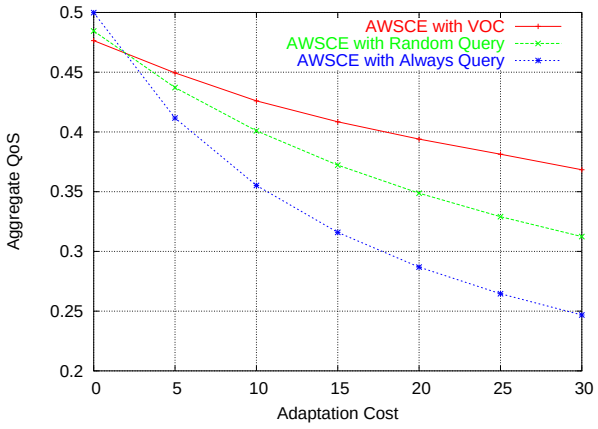

Figure 4. Performance of the various $\mathcal{A}$-WSCE systems

5.2 Results

We ran each of our experiments for 5000 units of simulated time with 10% instances changing in the environment. The adaptation cost values were taken as 0, 5, 10, 15, 20, 25 and 30%. The importance weights for cost, availability, and response time were set to 0.5, 0.25, and 0.25 respectively. The aggregate VOC threshold was set to 0.8. This was based on empirical observations during a few trial runs. For a given configuration of the system, we repeated the experiment five times. We measured aggregate QoS of the executed workflows and the number of switches (among the workflows executed) for each run. These values were then averaged over all runs and are reported in the plots. A workflow being executed in the run-time stage has two cost components: The cost of querying the instances for their latest QoS values, and the cost of invoking the instances themselves. Consequently, both the query cost and the instance invocation cost are made a part of the aggregate QoS.

Figure 4 shows the variation in aggregate QoS with adaptation cost for each of the systems described earlier. As expected, there is a degradation of QoS with increase in adaptation cost. Also, as can be seen from the figure, $\mathcal{A}$-WSCE with always querying gives the best results when the adaptation cost is negligible. However, as the adaptation cost increases, $\mathcal{A}$-WSCE with VOC gives the best aggregate QoS. This is because $\mathcal{A}$-WSCE with VOC selectively picks up only the significant instances to query. We observed that for very high adaptation costs, $\mathcal{A}$-WSCE with VOC collapses into an approach that does not query at all. Random querying is somewhere between these two ends of the spectrum. The improved performance of the $\mathcal{A}$-WSCE with VOC system in comparison to other approaches is not only due to the fact that the approach queries less and therefore incurs

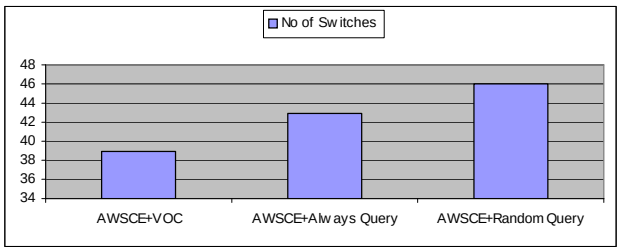

Figure 5. Stability of various $\mathcal{A}$-WSCE systems

less adaptation costs, but also because the approach queries instances that are likely to significantly improve the QoS of the workflow. Another important observation from the figure is that the magnitude of difference between the various strategies gets amplified as we increase the adaptation cost. This signifies the importance of VOC in environments where obtaining the latest QoS values of the deployed instances incurs substantial costs.

Figure 5 sheds light on the stability of various systems. Ideally, we would like to minimize the number of switches made while not compromising on the best workflow (in terms of QoS) to execute. As can be seen, $\mathcal{A}$-WSCE with VOC does the least number of switching, by intelligently estimating the relationship between the cost of querying and the gains expected from a workflow switch. Interestingly, random querying involves the maximum number of switches. This is because random querying picks up changes in the environment incrementally. Essentially, in one iteration it may pick up approximately half the changes in the environment (using a probability of 0.5 to query the latest QoS values of instances in a workflow) resulting in workflow switches. It may pick up some of the others in the next iteration resulting in more switches. This contrasts with the system that queries for all the changes at the same time, albeit, at the cost of incurring more queries.

Summary: Our experimental evaluation indicates that VOC does *selective* and *intelligent* querying. In addition to improving the performance of the $\mathcal{A}$-WSCE system, it also makes it more stable by switching lesser number of times among the executable workflows. Overall, inclusion of VOC leads to a more stable and cost-efficient WSCE in volatile environments.

6 Related Work

Recently, researchers are increasingly turning their attention to managing workflows in volatile environments. Au et al. [2] obtain current parameters of the workflow by querying Web service providers when the parameters expire. Plan recomputation is assumed to take place irrespective of whether the revised parameter values are expected to bring about a change in the composition. This may lead to frequent unnecessary computations. Improving on this work, Harney and Doshi [7] present a value of changed information mechanism that offers a way to ascertain the expected impact of the volatility of the environment on the workflow, one provider at a time. This facilitates the querying of those service providers whose revised information is worth the cost of obtaining it. In a somewhat different vein, Verma et al. [12] explore adaptation in workflows in the presence of coordination constraints between different WSs. We do not consider such constraints here.

In [9], graph based techniques were used to evaluate the feasibility and correctness of changes at the workflow instance level. Ellis et al. [6] used Petri-nets for formalizing the instance level changes. In a somewhat similar vein, Aalst and Basten [11] proposed a Petri-net based theory for workflow inheritance which categorized the types of changes that do not affect other interacting workflows. This line of work presents ways to verify the correctness of given changes, which complements our work on how and when to change the workflow. Muller et al. [8] propose a workflow adaptation strategy based on pre-defined event-condition-action rules that are triggered when a change in the environment occurs. While the rules provide a good basis for performing contingency actions, they are limited in the fact that they cannot account for all possible actions and scenarios that may arise in complex workflows. Additionally, the above work does not address optimality of workflow adaptation. Doshi et al. [5] offers such a solution using a technique that manages the dynamism of Web workflow environments through Bayesian learning. The workflow model parameters are updated based on previous interactions with the individual Web services and the composition plan is regenerated using these updates. This method suffers from being slow in updating the parameters, and the approach may result in plan recomputations that do not bring about any change in the workflow.

7 Discussion

Value of changed information (VOC) helps in selectively querying the relevant changes in the environment. In the $\mathcal{A}$-WSCE system, we used VOC in the Runtime Manager to decide which workflow instances should be queried for latest QoS values. It is also possible to use VOC at other stages of the $\mathcal{A}$-WSCE system. For example, this value can be used by the Physical Manager to pass only those workflows to the Runtime Manager having a low aggregate VOC. This should help in executing workflows having greater resiliency to changes in the environment, thereby resulting in less switches during the execution phase. However, this value has to be weighted appropriately in relation to the QoS

parameters of cost, availability and others.

The current VOC model (and the underlying QoS model) deals with only three QoS parameters - cost, response time and availability. It is straightforward to extend the model to other QoS dimensions by including the appropriate terms in the aggregate normalized VOC calculations. Moreover, instead of working at the level of VOC of a composite workflow, it is possible to incorporate at the granularity of individual instances. In this particular case, instead of comparing the VOC of a workflow against a threshold, we would use the VOC of each instance to decide whether to query for the latest QoS values or not. We wish to explore the tradeoffs that might arise in using the VOC at different granularities in the future.

The mechanism we have used to compute VOC is somewhat simplistic – VOC is computed only on the basis of distribution of QoS values for instances belonging to a particular service type. We are investigating ways to enhance this mechanism. For example, can we incorporate a measure of which instances are more likely to undergo changes at runtime as compared to others? Calculation of VoC is a computationally intensive task – extension of the model will further worsen the situation. However, in practice these computations can be done in parallel or in the background so as not to significantly affect the $\mathcal{A}$-WSCE system.

Finally, the choice of the VOC threshold in our adaptive system is important in deciding how frequently changes are incorporated by the system. A low threshold value would result in more querying, while a high threshold value results in less queries, albeit at the cost of missing out on some of the relevant changes in the environment. In our simulations, we arrived at the threshold value using initial empirical observations. In real systems, this value would be acquired through experience and refined as the system is used. Studying the effect of various thresholds on different environments should be an interesting direction for future research.

8 Conclusion

Adaptive Web service composition and execution systems often fail to consider the additional overhead of adapting to changes in the process environment. This could involve costs of querying for new information or modifying the currently executing workflow. We presented a method that improves these systems and in particular the $\mathcal{A}$-WSCE framework by reasoning whether updating the system with new information will likely yield an improved process that may overcome these costs. Specifically, our approach uses the value of changed information, which measures how a process is expected to perform in the changed environment. VOC guides the system to selectively query providers for useful information and reduce the switching between candidate workflows, altering them only when warranted. Our experiments demonstrate two important advantages that VOC adds to $\mathcal{A}$-WSCE) systems. First, workflows that use VOC maintain better aggregate QoS despite high information querying costs than other simpler strategies of acquiring revised information. Second, utilizing VOC helps yield a more stable system in terms of workflow switches.

References

[1] V. Agarwal, G. Chafle, K. Dasgupta, N. Karnik, A. Kumar, S. Mittal, and B. Srivastava. Synthy: A system for end to end composition of web services. *J. Web Semantics,Vol.3:4*, 2005.

[2] T.-C. Au, U. Kuter, and D. S. Nau. Web service composition with volatile information. In *International Semantic Web Conference*, pages 52–66, 2005.

[3] J. Cardoso, A. Sheth, J. Miller, J. Arnold, and K. Kochut. Quality of Service for Workflows and Web Service Processes. *Journal of Web Semantics*, 1:281–308, 2004.

[4] G. Chafle, K. Dasgupta, A. Kumar, S. Mittal, and B. Srivastava. Adaptation in web service composition and execution. In *International Conference on Web Services (ICWS)*, pages 549–557, 2006.

[5] P. Doshi, R. Goodwin, R. Akkiraju, and K. Verma. Dynamic workflow composition using markov decision processes. *Journal of Web Services Research (JWSR)*, 2(1):1–17, 2005.

[6] C. Ellis, K. Keddara, and G. Rozonberg. Dynamic change within workflow systems. In *COOCS*, pages 10–21, 1995.

[7] J. Harney and P. Doshi. Adaptive web processes using value of changed information. In *International Conference on Service-Oriented Computing (ICSOC)*, pages 179–190, 2006.

[8] R. Muller, U. Greiner, and E. Rahm. Agentwork: a workflow system supporting rule-based workflow adaptation. *Journal of Data and Knowledge Engineering.*, 51(2):223–256, 2004.

[9] M. Reichert and P. Dadam. Adeptflex-supporting dynamic changes of workflows without losing control. *Journal of Intelligent Information Systems*, 10(2):93–17, 1998.

[10] P. P. Tallon. Inside the adaptive enterprise: An information technology capabilities perspective on business process agility. Technical report, Center for Research on Information Technology and Organizations, University of California, Irvine, 2007.

[11] W. van der Aalst and T. Basten. Inheritance of workflows: an approach to tackling problems related to change. *Theoretical Computer Science*, 270(1-2):125–203, 2002.

[12] K. Verma, P. Doshi, K. Gomadam, J. Miller, and A. Sheth. Optimal adaptation in web processes with coordination constraints. In *International Conference on Web Services (ICWS)*, pages 257–264, 2006.

[13] L. Zeng, B. Benatallah, M. Dumas, J. Kalagnanam, and Q. Z. Sheng. Quality Driven Web Services Composition. In *Proc. 12th WWW*, May 2003.