A Dynamic Approach toward QoS-Aware Service Workflow Composition

David Chiu   Sagar Deshpande[‡]   Gagan Agrawal   Rongxing Li[‡]
Department of Computer Science and Engineering
[‡] Department of Civil and Environmental Engineering and Geodetic Science
The Ohio State University, Columbus, OH 43210, USA

Abstract

Web service-based workflow management systems have garnered considerable attention for automating and scheduling dependent operations. Such systems often support user preferences, e.g., time of completion, but with the rebirth of distributed computing via the grid/cloud, new challenges are abound: multiple disparate data sources, networks, nodes, and the potential for moving very large datasets. In this paper, we present a framework for integrating QoS support in a service workflow composition system. The relationship between workflow execution time and accuracy is exploited through an automatic workflow composition scheme. The algorithm, equipped with a framework for defining cost models on service completion times and error propagation, composes service workflows which can adapt to user's QoS preferences.

1 Introduction

The success of service frameworks has preceded the distribution of interoperable processes, data sets, and other resources in various scientific domains [15]. But with the high availability of distributed heterogeneous data sets and web services comes the nontrivial challenge for scientists and other end-users to manage such information. For instance, certain information involves execution of several service operations with disparate data sources in a particular sequence, in a process known as service composition[§]. Certainly, the ultimate hope for enabling these workflows is to automate their composition while simultaneously hiding low-level details such as service and data discovery, integration, and scheduling from the user.

Often in practice, there exist multiple ways of answering a given query, using different combinations of data sources and services. Some combinations are likely to result in higher cost, but better accuracy, whereas others might lead to quicker results and lower accuracy. This could be due to that some data collection methods involve higher resolution or because some datasets are available at servers with lower access latencies. Meanwhile, different classes of users can have different querying requirements. For instance, some users may want the fastest answers while others require the most accurate response. Users may also prefer the faster of the methods which can meet certain accuracy constraints. While most efforts in workflow management systems focus directly on minimizing execution times [26, 30, 1] through scheduling heuristics, it would be highly desirable to enable user preferences for both accuracy and time.

Our system seeks to alleviate users from the need of understanding the cost and accuracy tradeoffs associated with different datasets and services that could be used to answer a query. This paper presents such a framework for service workflow composition, which uses a novel approach for dynamically supporting user preferences on time and accuracy. To automate the time-accuracy tradeoff in web service composition, we allow developers to expose an accuracy parameter, e.g., sampling rate. Our system takes unrestricted models as input for predicting process completion time and error/accuracy propagation of the applications. Our workflow composition algorithm employs an efficient algorithm to automatically regulate the accuracy parameter based on the defined cost models to meet the user-specified QoS constraints. We conducted experiments to evaluate two aspects of our algorithm. First, we show that, although the cost models can be invoked quite frequently during workflow planning, they contribute little overhead to the overall planning time. Secondly, we present the effect that the accuracy parameter adjustment scheme has on planned workflows.

The remainder of this paper is organized as follows. An overview of our system is presented in the next section. In Section 3 we discuss technical details of our cost models and workflow composition algorithm. Performance evaluations of our workflow composition algorithm are presented in Section 4, and a comparison of our work with related research efforts follows in Section 5. Lastly, we conclude and discuss future opportunities in Section 6.

2 System Architecture Overview

An overview of the system is presented here while detailed descriptions can be found in previous works [6, 7, 8].

[§]Throughout this paper, we use terms *service composition* and *workflow composition* interchangeably although it has been pointed out that *service workflow synthesis* is also appropriate.

Our system, as a whole, functions as a *service workflow broker*: as users submit high-level queries to the system, the broker automatically plans and executes the workflows involved in deriving the desired result (virtual data) while hiding such complexities as service composition, domain knowledge, and QoS optimization from the user.
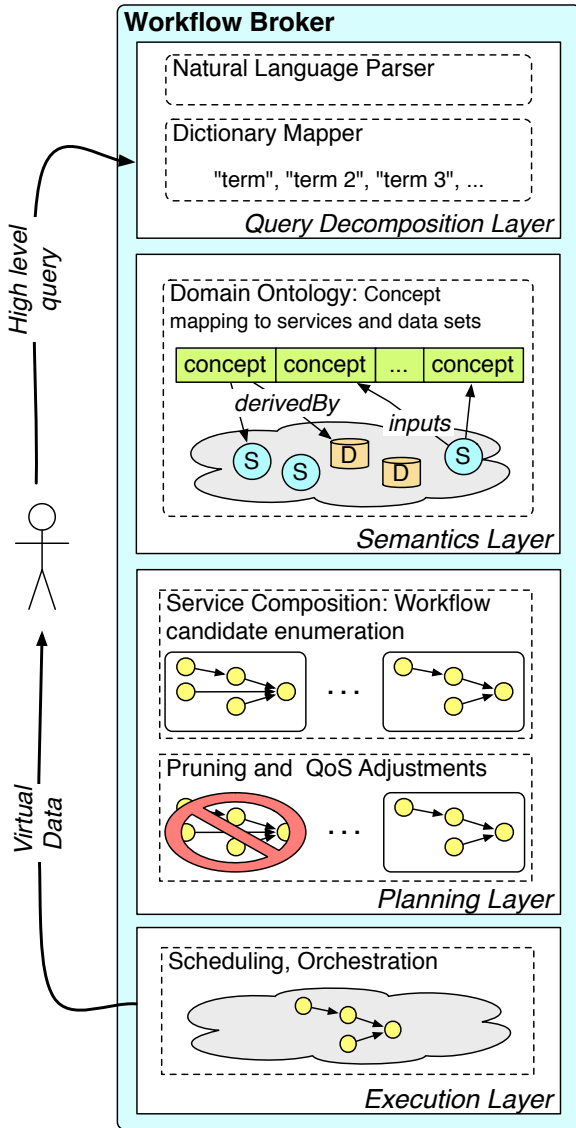

Figure 1. Workflow Broker Architecture

Depicted in Figure 1, the broker's architecture consists of four independent layers, each with a well-defined set of functionalities. The *Query Decomposition Layer* employs the StanfordNLP [20] parser and WordNet [13] libraries to decompose high-level user queries into a set of keywords and dependencies. The keywords are mapped to distinct concepts within the domain ontology resident in the subsequent layer. A more detailed description of this process is given in our other paper [6].

The task of the *Semantics Layer* is two-fold. First, this layer maintains an active list of available compute nodes, services, data sets, and their accompanying metadata. Services are described in WSDL [5] and data sets in their respective metadata standards. Since the work described in this paper deals with geospatial data sets, the Content Standard for Digital Geospatial Metadata (CSDGM) [14], is used for data annotation. The second task of this layer is providing a domain-specific ontology. The ontology defines
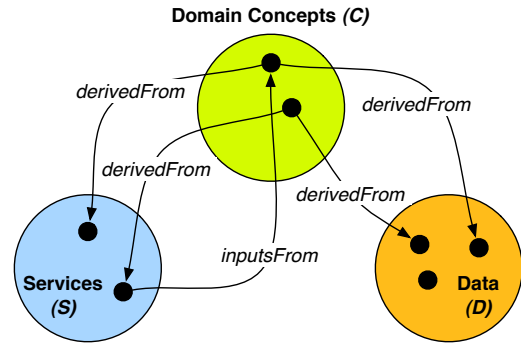
Figure 2. Ontology for Domain Description

relationships between the available data sets and services to concepts a domain. For example, there is a need for the system to understand how "water levels" in some scientific domain are derived using some existing data sets, services, or combinations of both. These relationships help facilitate planning algorithms for service composition. The ontology, specified in the Web Ontology Language (OWL) [9], is a directed graph with the following requirements:

- The ontology consists of three disjoint sets (classes) C, S, and D representing the set of domain concepts, the set of available services known to the system, and the known domain-specific data types, respectively.

- Two types of directed edges (relations) exist: concepts may be $derivedFrom$ data or service nodes and a service $inputsFrom$ concepts.

This ontological definition, shown in Figure 2, simplifies the effort to indicate which services and data types are responsible for deriving specific domain concepts. The ontology is predefined by experts within the scientific domain.

Next, the *Planning Layer* assumes that the ontology and metadata are in place and defined. The planning algorithm, discussed in detail in the following section, relies heavily on the Semantics Layer. The planner enumerates service-synthesized workflows to answer a particular query through traversals of the domain ontology. The existence of needed services and data sets is identified by the ontological index. This layer sends a set of workflows all capable of answering the user query to the *Execution Layer* for processing, and the resulting virtual data is finally returned back to the user.

This paper focuses on the mechanisms within the Planning Layer for handling QoS constraints while composing service workflows. For instance, one might issue the following query to retrieve a cropped image of Columbus, Ohio in under 60 seconds. To answer this query, the system might consider multiple workflow plans, possibly employing image compression services to speed up data movement in order to produce the desired image within 60 seconds. Appropriate services and data sets should be identified and selected for use and their composition is reified dynamically through communication with the domain ontology from the Semantics Layer. Through this process, the workflow composition algorithm, WFEnum, enumerates a list of valid workflow candidates such that when each is executed, returns a suitable response to the query.

From the list of workflow plans, the WFEnum algorithm must also examine the cost of each in order to determine a sub-list of candidates that meet user constraints. The algorithm can dynamically adjust workflow accuracy (by invoking data reduction services, for instance) in order to meet expected QoS requirements. Finally, the execution of workflows is carried out and the presence of faults within a certain execution, caused by such factors as network downtime or data/process unavailability, triggers the execution of the next queued workflow (if available) to provide the next best possible response.

3 QoS-Aware Service Composition

In this section we focus on problem formulation and implementation details of our approach. This paper focuses on modeling tradeoffs between time and errors within scientific domains, despite that other criteria can also be considered, e.g., utility costs. In practice, tasks can be composed of disparate services chained together to produce some desired values [2]. Also called composite services, workflows are often expressed as directed acyclic graphs where the vertices denote services and data elements and directed edges represent the flow of execution. Workflows, in our context, can also be recursively defined as follows. Given some set of data, D and a set of services S, a workflow

$$w = \begin{cases} \epsilon \\ d \\ (op, P_{op}) \end{cases}$$

such that terminals ϵ and $d \in D$ denote a null workflow and a data instance respectively. Nonterminal $(op, P_{op}) \in S$ is a tuple where op denotes a service operation with a corresponding parameter list $P_{op} = (p_1, \ldots, p_k)$ and each p_i is itself a workflow. To put simply, a workflow is a tuple which either contains a single data instance or a service operation whose parameters are, recursively, (sub)workflows.

3.1 Modeling Workflow Cost

Two cost functions are proposed for aggregating workflow execution time and error propagation respectively. A workflow's time cost can be estimated by:

$$T(w) = \begin{cases} 0, & \text{if } w = \epsilon \\ t_{net}(d), & \text{if } w \in D \\ t_x(op, P_{op}) + & \text{if } w \in S \\ \quad t_{net}(op, P_{op}) + \max_{p_i \in P_{op}} T(p_i), \end{cases}$$

If workflow w is a base data element, then $w = d$, and the cost is trivially the data transmission time, t_{net}. When w is a service, then $w = (op, P_{op})$, and its time can be summarized as the sum of the service's execution time t_x, network transmission time of its product, and, recursively, the maximum time taken by all of its parameters (assuming their execution can be carried out concurrently).

The error aggregation function, $E(w)$, which represents the error estimation of a given workflow, is also in a recursive sum form:

$$E(w) = \begin{cases} 0, & \text{if } w = \epsilon \\ \sigma(d), & \text{if } w \in D \\ \sigma(op, P_{op}) + \max_{p_i \in P_{op}} E(p_i), & \text{if } w \in S \end{cases}$$

Due to the heterogeneity of datasets and processes, it is expected that disparate workflows will yield results with fluctuating measures of accuracy. Again, at the base case lies the expected error of a particular data set, $\sigma(d)$. An error value can also be attributed to a service execution, $\sigma(op, P_{op})$. For instance, errors will be introduced if a sampling service is called to reduce data size or some interpolation/extrapolation service is used estimate some value.

3.2 Workflow Enumeration and Pruning

The goal of a workflow planning algorithm is to enumerate a sequence of workflows $W_q = (w_1, \ldots, w_n)$ capable of answering some query q by employing the available services and data sets. The execution of each $w_i \in W_q$ is carried out, if needed, by an order determined by cost or QoS parameters. Thus, upon workflow execution failure, the system can persistently attempt alternative, albeit potentially less optimal, workflows.

Our workflow planning algorithm, WFEnum (Algorithm 1), takes as input the query's targeted domain concept, $target$, the user's time constraint, QoS_{time}, and error constraint, QoS_{error}. WFEnum runs a modification of Depth-First Search on the domain ontology starting from $target$. It is defined by the ontology that every concept can be realized by various data types or services. WFEnum starts (Line 2) by retrieving a set, Λ_{data} of all data types that can be used to derive the input concept, $target$. Each element in Λ_{data} is a potential data workflow candidate, i.e., $target$ can be derived by the contents within some file. Correctly and quickly identifying the necessary files based on the user's query parameters (Line 4) is a challenge and out of the scope of this work.[†] On Line 7, each file is used

---

† Details on query parsing, the handling of immediate data values, and data identification can be found in [6]

to call an auxiliary procedure, QoSMerge, to verify that its inclusion as a workflow candidate will not violate QoS parameters.

Algorithm 1 WFEnum($target, QoS_{time}, QoS_{error}$)

1: $W \leftarrow ()$
2: $\Lambda_{data} \leftarrow Ontology.\text{derivedFrom}(D, target)$
3: **for all** $dataType \in \Lambda_{data}$ **do**
4:    $F \leftarrow dataType.\text{getFiles}()$
5:    **for all** $f \in F$ **do**
6:       $w \leftarrow (f)$
7:       $W \leftarrow (W, \text{QoSMerge}(w, \infty, \infty, QoS_{time}, QoS_{error}))$
8:    **end for**
9: **end for**
10: $\Lambda_{srvc} \leftarrow Ontology.\text{derivedFrom}(S, target)$
11: **for all** $op \in \Lambda_{srvc}$ **do**
12:    $P_{op} \leftarrow op.\text{getParams}()$
13:    $W_{op} \leftarrow ()$
14:    **for all** $p \in P_{op}$ **do**
15:       $W_p \leftarrow \text{WFEnum}(p.target, QoS)$
16:       $W_{op} \leftarrow W_{op} \times W_p$
17:    **end for**
18:    **for all** $pm \in W_{op}$ **do**
19:       $w \leftarrow (op, pm)$
20:       $W \leftarrow (W, \text{QoSMerge}(w, \infty, \infty, QoS_{time}, QoS_{error}))$
21:    **end for**
22: **end for**
23: **return** W

The latter half of the WFEnum algorithm handles service-based workflow planning. From the ontology, a set of relevant service operations, Λ_{srvc} is retrieved for deriving $target$. For each service operation, op, there may exist multiple ways to plan for its execution because each of its parameters, p, by definition, is a (sub)problem. Therefore, workflows pertaining to each parameter p must first be computed via a recursive call (Line 15) to solve each parameter's (sub)problem, whose results are stored in W_p. The combination of these parameter (sub)workflows in W_p is then established through a cartesian product of its derived parameters (Line 16). For instance, consider a service workflow with two parameters of concepts a and b: $(op, (a, b))$. Assume that target concepts a is derived using workflows $W_a = (w_1^a, w_2^a)$ and b can only be derived with a single workflow $W_b = (w_1^b)$. The distinct parameter list plans are thus obtained as $W_{op} = W_a \times W_b = ((w_1^a, w_1^b), (w_2^a, w_1^b))$. Each tuple from W_{op} is a unique parameter list, pm. Each service operation, when coupled with a distinct parameter list (Line 19) produces an equally distinct service-based workflow which again invokes QoSMerge for possible inclusion into the final workflow candidate list (Line 20). In our example, the final list of workflows is obtained as $W = ((op, (w_1^a, w_1^b)), (op, (w_2^a, w_1^b)))$.

When a workflow becomes a candidate for inclusion, QoSMerge (Algorithm 2) is called to make a final decision: prune, include as-is, or modify workflow accuracy then include. For simplicity, we consider a single error model, and hence, just one adjustment parameter in our algorithm.

Algorithm 2 QoSMerge($w, t', e', QoS_{time}, QoS_{error}$)

1: /* no time constraint */
2: **if** $QoS.Time = \infty$ **then**
3:    $C_T \leftarrow \infty$
4: **end if**
5: /* no accuracy constraint */
6: **if** $QoS.Err = \infty$ **then**
7:    $C_E \leftarrow \infty$
8: **end if**
9: /* constraints are met */
10: **if** $T(w) \leq QoS_{time} \wedge E(w) \leq QoS_{error}$ **then**
11:    **return** w /* return w in current state */
12: **end if**
13: /* convergence of model estimations */
14: **if** $|T(w) - t'| \leq C_T \wedge |E(w) - e'| \leq C_E$ **then**
15:    **return** $\emptyset$ /* prune w */
16: **else**
17:    $\alpha \leftarrow w.\text{getNextAdjustableParam}()$
18:    $\gamma \leftarrow \text{suggestParamValue}(\alpha, w, QoS_{error}, C_E)$
19:    $w_{adj} \leftarrow w.\text{setParam}(\alpha, \gamma)$
20:    **return** QoSMerge($w_{adj}, T(w), E(w), QoS_{time}, QoS_{error}$)
21: **end if**

QoSMerge inputs the following arguments: (i) w, the workflow under consideration, (ii) t' and (iii) e' are the predicted time and error values of the workflow from the previous iteration (for detecting convergence), and (iv) QoS_{time} and QoS_{error} are the QoS objects from the query.

Initially, QoSMerge assigns convergence thresholds C_E and C_T for error and time constraints respectively. These values are assigned to ∞ if a corresponding QoS is not given. Otherwise, these thresholds assume some insignificant value. If the current workflow's error and time estimations, $E(w)$ and $T(W)$, meet user preferences, the workflow is included into the result set. But if the algorithm detects that either of these constraints is not met, the system is asked to provide a suitable value for α, the adjustment parameter of w, given the QoS values.

Taken with the suggested parameter, the QoSMerge procedure is called recursively on the adjusted workflow, w_{adj}. After each iteration, the accuracy parameter for w is adjusted, and if both constraints are met, w is returned to WFEnum for inclusion in the candidate list, W. However, when the algorithm determines that the modifications to w provide insignificant contributions to its effects on $T(w)$ and $E(w)$, i.e., the adjustment parameter converges without being able to meet both QoS constraints, then w is left out of the returned list. As an aside, the values of t' and e' of the initial QoSMerge call on (Lines 7 and 20) of Algorithm 1 are set to ∞ for dispelling the possibility of premature convergence.

QoSMerge employs the suggestParamValue procedure (Algorithm 3) to tune the workflow's adjustment parameters.[‡] This algorithm has two cases: The trivial case is that

---

[‡]Although Algorithm 3 only shows error QoS awareness, time QoS is handled in much the same way.

Algorithm 3 suggestParamValue($\alpha, w, QoS_{error}, C_E$)

1: /* trivially invoke model if one exists for suggesting α */
2: **if** $\exists$ model(α, $w.op$) **then**
3:      $M \leftarrow$ getModel($w.op, \alpha$)
4:      return M.invoke($QoS.Err$)
5: **else**
6:      $min \leftarrow \alpha.min$
7:      $max \leftarrow \alpha.max$
8:      **repeat**
9:          $m' \leftarrow (min + max)/2$
10:         $w_{adj} \leftarrow w$.setParam(α, m')
11:         **if** $QoS.Err < E(w_{adj})$ **then**
12:            $min \leftarrow m'$
13:         **else**
14:            $max \leftarrow m'$
15:         **end if**
16:      **until** ($max < min \vee |E(w_{adj}) - QoS.Err| < C_E$)
17:      return m'
18: **end if**

a model is supplied for arriving at an appropriate value for α, the adjustment parameter. Sometimes this model is simply inverse of either the time or error models that exist for solving $T(w)$ and $E(w)$.

However, finding a model for suggesting α can be non-trivial. In these cases, α can be found by employing the existing models, $T(w)$ and $E(w)$ in a forward fashion. For example, consider that α is the rate at which to sample some specific file as a means for reducing workload. The goal, then, is to maximize the sampling rate while being sure that the workflow it composes remains below QoS_{time} and QoS_{error}. Since sampling rates are restrained to a specific range, i.e., $\alpha \in [0, 1]$, binary search can be utilized to find the optimal value (Lines 5-16 in Algorithm 3). For this to work properly $T(w)$ and $E(w)$ are assumed to be monotonically increasing functions, which is the case in most cases.

If either QoS constraint is not given by the user, its respective models is actually never invoked. In this case, QoSMerge becomes the trivial procedure of immediately return workflow candidate w. In Algorithm 2 this is equivalent to assigning the QoS constraints to ∞.

3.3   Service Framework Implementation

Our goal of automatically composing QoS-aware services relies on a structure encompassing service and data availability, access costs, and their relations to the supported domain. In this direction we implemented a service registration framework, shown in Figure 3. To register a service into our system, a domain expert initially inputs the service's WSDL file. Our system validates the WSDL and peruses through each supported operation. For each operation, (op, P_{op}), the system asks the expert for multiple inputs: (i) K_p — keywords describing each of the operation's parameters $p \in P_{op}$, (ii) K_{out} — keywords describing the operation's output, and (iii) a set of models/equations defining the operation's error propagation and execution time, i.e., those

models discussed previously in Section 3.1.
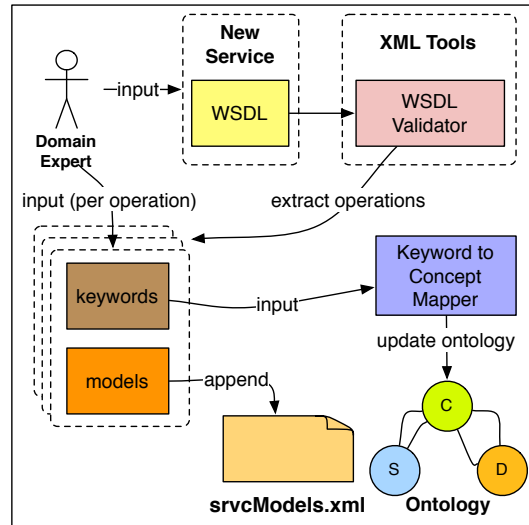
Figure 3. Service Registration

Upon the given inputs, our system *registers* each prescribed service operation in the following way. First, the system's ontology must be updated to reflect the new service operation. A new service instance for op is added into the service class, S. Next, relational edges for the new service are computed. To do this, WordNet [13] libraries are used to match like-terms in each of the provided K_p sets. The reduced set of terms is then matched to concepts within the system's domain ontology (keyword-to-concept mapping is assumed to be already provided). For each parameter p, an *inputsFrom* edge to the computed domain concept is added into the ontology. The same process is taken for prescribing the *derivedFrom* edge for K_{out}. With ontological updates in place, the new operation is now available to the WFEnum algorithm for workflow planning.

Next, the registration process handles the input cost models per operation. Equipped with an equation parser, our framework allows general infix equations to be specified representing each model. Alternatively, algorithms can also be defined (in Java) for realizing more complex models. These models are appended onto the system's service configuration file, srvcModels.xml. In Figure 4, models are defined for an operation, DEMDiff, which inputs two files, DEM_1 and DEM_2. A Digital Elevation Model (DEM) is a file consisting of a matrix whose points represent surface elevation. For this particular operation, notice that the error model, σ, is defined as a method call. Upon the need to invoke σ, our system dynamically loads the specified class, *geodomain.GeoError*, and calls the *DEMComputeError* method on some given sampling rate.

```xml
<operation name="DEMDiff">
 <!-- sized -->
 <model type="outputModel"
   equation="max(DEM1.SIZE, DEM2.SIZE)" />
 <!-- tx -->
 <model type="execTimeModel"
   equation="8.11E-7 * max(DEM1.SIZE, DEM2.SIZE) + .." />
 <!-- σ -->
 <model type="errorModel"
   class="geodomain.GeoError"
   method="DEMComputeError(RATE)" />
 <!-- ... -->
</operation>
```

Figure 4. SrvcModels.xml Snippet

3.4 An Experimental Example

As a concrete example for this work, we consider an often utilized workflow in the geospatial domain. The query, which we refer to as DEM Query, measures land elevation change in a span of time. We focused on the particular adjustable accuracy parameter of *sampling rate* on data sets.

The DEM Query involves a simple algorithm which extracts the elevation between two Digital Elevation Model (DEM) files: DEM_1 and DEM_2. A DEM is essentially an $m \times n$ grid with elevation values at each point. But suppose that DEM_2 is sampled to reduce execution time cost. Sampling effectively widens the gap between each pair of points. This variation between resolutions presents difficulties to the otherwise trivial computation of elevation difference, $DEM_1 - DEM_2$. In order to compensate for the unknown values of DEM_2, interpolation becomes necessary, which likely produces errors.

Together with the Department of Civil Engineering and Geodetic Science at Ohio State University, we analyzed the physical errors introduced by interpolation as a function of sampling rate. Details on this effort are discussed in another paper [7]. This process, σ, for modeling DEM errors is implemented as a method call in the QoSMerge algorithm for solving error cost. The time models presented in Figure 4 were computed using multiple regression. In our example, we did not include a trivial model for suggesting parameters, which forces the invocation the nontrivial case in Algorithm 3.

4 Experimental Results

The performance evaluation focuses on two goals: (i) To evaluate the overhead of workflow enumeration algorithm and the impact of pruning. (ii) To evaluate the efficiency and effectiveness of our adaptive QoS parameter scheme. The initial goal is to present the efficiency of Algorithm 1. This core algorithm, called upon every given query, emcompasses both auxiliary algorithms: QoSMerge — the decision to include a candidate and SuggestParam-Value — the invocation of error and/or time models to ob-

tain an adjustment value appropriate for meeting user preferences. Thus, an evaluation of this algorithm offers a holistic view of our system's efficiency. A synthetic ontology, capable of allowing the system to enumerate thousands of workflows, consisting of five activities each, for a user query, was generated for purposes of facilitating this scalability experiment. The results, depicted in Figure 5, was repeated for an increasing number of workflow candidates (i.e., $|W| = 1000, 2000, \ldots$) enumerated by WFEnum on four configurations (solid lines). These four settings correspond to user queries with (i) no QoS constraints, (ii) only error constraints, (iii) only time constraints, and (iv) both constraints.
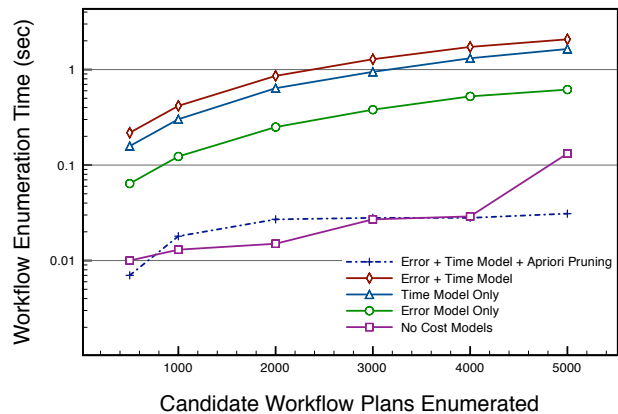

Figure 5. Cost Model Overhead and Pruning

Expectedly, the enumeration algorithm runs in proportional time to the numbers of models supported. To evaluate our algorithm's efficiency, we altered our previous experimental setting to contain exactly one workflow within each candidate set that meets both time and error constraints. That is, for each setting of $|W| + 1$, the algorithm now prunes $|W|$ workflows (dashed line). The results show that cost-based pruning algorithm is as efficient as *no-cost* model since the amount of workflows considered is effectively minimized due to their cost being unable to fulfill QoS requirements.

Next, we demonstrate the system's efforts for supporting user preferences. We begin by presenting an evaluation of the adaptive workflow parameter suggestion procedure. For this experiment, the sampling rate is the exposed workflow accuracy adjustment parameter. Table 1 shows the ideal and actual, i.e., system provided, error targets. On the left half of the table, the ideal accuracy % is the user provided accuracy constraint and the ideal error is the error value (from the model) expected given this corresponding accuracy preference. The right half of the table shows the actual accuracy % and errors that the system provided through the manipulation on sampling rate. As seen in the table, although the error model appears to be extremely sensitive to diminutive amounts of correction, our system's sugges-

Table 1. Suggested Value of Parameters

Ideal		System Suggested	
Acc %	Error (meters)	Acc %	Error (meters)
10	8.052	11.81	8.052001
20	7.946	21.15	7.945999
30	7.911	28.61	7.911001
40	7.893	34.96	7.892999
50	7.868	50.52	7.867996
60	7.859	60.16	7.858989
70	7.852	70.65	7.851992
80	7.847	80.71	7.847001
90	7.8437	89.07	7.843682
100	7.8402	99.90	7.840197

tion of sampling rates does not allow a deviation of more than 1.246% on average. It is also observable that the % of deviation causes minute, if not negligible, differences (in meters) as compared to the ideal accuracies.

Finally, the DEM query was executed with user given accuracy preferences of 10%, 20%, ..., 100% on DEM files of sizes 125mb and 250mb. As seen in Figure 6, the sampling rates along with the workflow's corresponding execution times at each accuracy preference, increase as the user's accuracy preference increases. The figure clearly shows the benefits from using sampling, as the execution time is reduced polynomially despite some loss in accuracy.
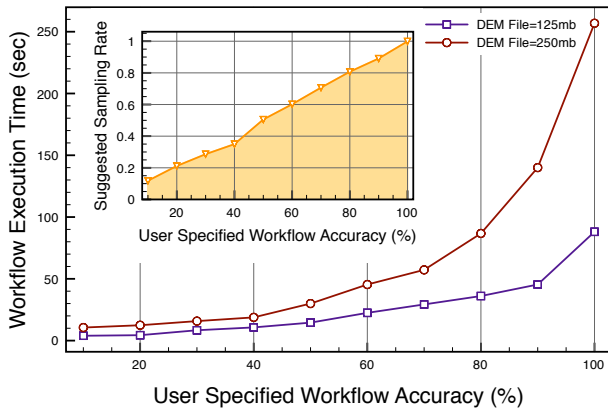
Figure 6. Workflow Accuracy and Corresponding Execution Times for DEM Query

We believe that our experimental results suggest that the system maintains robustness against user defined cost, and although not shown due to space limitations, parameter adjustment for meeting time-based QoS constraints exhibited similar results.

5 Related Works

The class of service/workflow composition systems for supporting composite business and scientific processes has been studied extensively in a number of works [22, 28, 16, 17, 3, 25, 24]. These systems, which borrow techniques from AI Planning, typically enable end-users to compose workflows from a high level perspective and automate workflow scheduling and execution. Workflow systems with QoS support have also been developed. Most works in this area concentrate on process/service scheduling in order to minimize total execution times. Eder et al. suggests heuristics for computing process deadlines and meeting global time constraints [12]. Other works, including Zeng et al.'s workflow middleware [30], Pegasus [10, 18], Amadeus [4], Askalon [26], and more recently, stochastic modeling approaches [1, 27] exploit grid service technologies, where datasets are inherently assumed heterogeneous and intelligent workflow scheduling on resource availability becomes a greater issue in meeting time constraints.

The notion and merits of utilizing service workflows, or so-called service chains, within the specific scope of geoinformatics were originally highlighted in [2]. However, to truly realize the autonomous construction of workflows relies heavily on well-defined and standardized domain specific information. Consequently, studies on the use of geospatial ontologies for automated workflow composition have been carried out. The work of Lemmens et al. [21] describes a framework for semi-automatic workflow composition. Yue et al. successfully demonstrated that automatic construction of geospatial workflows can be realized using their ontological structure [29, 11]. Hobona et al. [19] combines a well-established geospatial ontology, SWEET [23], with an adopted notion of semantic similarity of the constructed workflows and the user's query.

Our system differs from the above in the way that we assume multiple workflow candidates can be composed for any given user query. This set of candidates is pruned on the apriori principle from the given user preferences, making the workflow enumeration efficient. Furthermore, we focus on an accuracy-oriented task by allowing the user to specify application/domain specific time and error propagation models. Our system offers the online ability to adjust workflow accuracies in such a way that the modified workflow optimizes the QoS constraints.

6 Conclusion and Future Work

This paper reports an approach toward enabling time and accuracy constraints in service workflow composition through methods for modeling application-specific error and execution times. We evaluated our system in many dimensions, and overall, our results show that the inclusion of such cost models contributes insignificantly to the overall execution time of our workflow composition algorithm, and in fact, can reduce its overall time through pruning unlikely candidates at an early stage. We also showed that our

adaptive accuracy parameter adjustment is effective for suggesting relevant values for dynamically reducing the size of data.

As we seek to further our development of this system, we are aware of features that have not yet been investigated or implemented. One area is service scheduling on distributed heterogeneous resources. The problem, which is inherently NP-Hard, has received much recent attention. While many heuristics have been developed to address this issue, integrating novel features such as partial workflow caching and migration adds complexity to the problem. We plan to investigate the support for these aspects and develop new heuristics for enabling an efficient and robust scheduler.

Acknowledgments

References

[1] A. Afzal, J. Darlington, and A. S. McGough. Qos-constrained stochastic workflow scheduling in enterprise and scientific grids. In *GRID*, pages 1–8, 2006.

[2] N. Alameh. Chaining geographic information web services. *IEEE Internet Computing*, 07(5):22–29, 2003.

[3] J. Blythe, E. Deelman, Y. Gil, C. Kesselman, A. Agarwal, G. Mehta, and K. Vahi. The role of planning in grid computing. In *The 13th International Conference on Automated Planning and Scheduling (ICAPS)*, Trento, Italy, 2003. AAAI.

[4] I. Brandic, S. Benkner, G. Engelbrecht, and R. Schmidt. Qos support for time-critical grid workflow applications. *E-Science*, 0:108–115, 2005.

[5] R. Chinnici, J.-J. Moreau, A. Ryman, and S. Weerawarana. Web services description language (wsdl) 2.0. w3c recommendation 26 june 2007. http://www.w3.org/tr/wsdl20/.

[6] D. Chiu and G. Agrawal. Enabling ad hoc queries over low-level scientific datasets. In *Proceedings of the 21th International Conference on Scientific and Statistical Database Management (SSDBM'09)*, 2009.

[7] D. Chiu, S. Deshpande, G. Agrawal, and R. Li. Composing geoinformatics workflows with user preferences. In *GIS '08: Proceedings of the 16th ACM SIGSPATIAL international conference on Advances in geographic information systems*, New York, NY, USA, 2008. ACM.

[8] D. Chiu, S. Deshpande, G. Agrawal, and R. Li. Cost and accuracy sensitive dynamic workflow composition over grid environments. In *Proceedings of the 9th IEEE/ACM International Conference on Grid Computing (Grid'08)*, 2008.

[9] M. Dean and G. Schreiber. Owl web ontology language reference. w3c recommendation, 2004.

[10] E. Deelman, G. Singh, M.-H. Su, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, G. B. Berriman, J. Good, A. C. Laity, J. C. Jacob, and D. S. Katz. Pegasus: A framework for mapping complex scientific workflows onto distributed systems. *Scientific Programming*, 13(3):219–237, 2005.

[11] L. Di, P. Yue, W. Yang, G. Yu, P. Zhao, and Y. Wei. Ontology-supported automatic service chaining for geospatial knowledge discovery. In *Proceedings of American Society of Photogrammetry and Remote Sensing*, 2007.

[12] J. Eder, E. Panagos, and M. Rabinovich. Time constraints in workflow systems. *Lecture Notes in Computer Science*, 1626:286, 1999.

[13] C. Fellbaum, editor. *WordNet: An Electronic Lexical Database*. The MIT Press, 1998.

[14] Metadata ad hoc working group. content standard for digital geospatial metadata, 1998.

[15] I. Foster. Service-oriented science. *Science*, 308(5723):814–817, May 2005.

[16] K. Fujii and T. Suda. Semantics-based dynamic service composition. *IEEE Journal on Selected Areas in Communications (JSAC)*, 23(12), 2005.

[17] Y. Gil, E. Deelman, J. Blythe, C. Kesselman, and H. Tangmunarunkit. Artificial intelligence and grids: Workflow planning and beyond. *IEEE Intelligent Systems*, 19(1):26–33, 2004.

[18] Y. Gil, V. Ratnakar, E. Deelman, G. Mehta, and J. Kim. Wings for pegasus: Creating large-scale scientific applications using semantic representations of computational workflows. In Proceedings of the 19th Annual Conference on Innovative Applications of Artificial Intelligence (IAAI), Vancouver, British Columbia, Canada, July 22-26,, 2007.

[19] G. Hobona, D. Fairbairn, and P. James. Semantically-assisted geospatial workflow design. In *GIS '07: Proceedings of the 15th annual ACM international symposium on Advances in geographic information systems*, pages 1–8, New York, NY, USA, 2007. ACM.

[20] D. Klein and C. D. Manning. Accurate unlexicalized parsing. In *Proceedings of the 41st Meeting of the Association for Computational Linguistics*, pages 423–430, 2003.

[21] R. Lemmens, A. Wytzisk, R. de By, C. Granell, M. Gould, and P. van Oosterom. Integrating semantic and syntactic descriptions to chain geographic services. *IEEE Internet Computing*, 10(5):42–52, 2006.

[22] S. R. Ponnekanti and A. Fox. Sword: A developer toolkit for web service composition. In *WWW '02: Proceedings of the 11th international conference on World Wide Web*, 2002.

[23] R. Raskin and M. Pan. Knowledge representation in the semantic web for earth and environmental terminology (sweet). *Computer and Geosciences*, 31(9):1119–1125, 2005.

[24] B. Svrivastava and J. Koehler. Planning with workflows - an emerging paradigm for web service composition. In *Workshop on Planning and Scheduling for Web and Grid Services*. ICAPS, 2004.

[25] P. Traverso and M. Pistore. Automated composition of semantic web services into executable processes. In *3rd International Semantic Web Conference*, 2004.

[26] M. Wieczorek, R. Prodan, and T. Fahringer. Scheduling of scientific workflows in the askalon grid environment. *SIGMOD Rec.*, 34(3):56–62, 2005.

[27] W. Wiesemann, R. Hochreiter, and D. Kuhn. A stochastic programming approach for qos-aware service composition. *The 8th IEEE International Symposium on Cluster Computing and the Grid (CC-GRID'08)*, pages 226–233, May 2008.

[28] D. Wu, E. Sirin, J. Hendler, D. Nau, and B. Parsia. Automatic web services composition using shop2. In *ICAPS'03: International Conference on Automated Planning and Scheduling*, 2003.

[29] P. Yue, L. Di, W. Yang, G. Yu, and P. Zhao. Semantics-based automatic composition of geospatial web service chains. *Comput. Geosci.*, 33(5):649–665, 2007.

[30] L. Zeng, B. Benatallah, A. H. Ngu, M. Dumas, J. Kalagnanam, and H. Chang. Qos-aware middleware for web services composition. *IEEE Transactions on Software Engineering*, 30(5):311–327, 2004.