Workflow-Based Service Selection Under Multi-Constraints

[1]Chao Xia, [2]Chi-Hung Chi, [3]Raymond Wong, [4]Andreas Wombacher, [4]Luis F. Pires, [4]Marten van Sinderen, [5]Chen Ding
[1]School of Software, Tsinghua University, Beijing, China
[2]Digital Productivity Flagship, CSIRO, Australia
[3]Department of Computer Science, University of New South Wales, Sydney, Australia
[4]University of Twente, Netherlands
[5]Department of Computer Science, Ryerson University, Toronto, Canada

Abstract — **Despite the availability of services with similar functionality but from different providers in the cloud, using them in a workflow might subject to constraints such as service QoS and service bundling. Service bundling refers to the situation where the subscription of two services have to be done together; such requirement might be imposed by service providers and/or by the alliance group that the providers join in. In this paper, we focus on the service selection problem under the QoS constraints from the user and the bundling constraints associated with the chosen services. We first formulate the service selection problem as a multi-constrained selection problem. Then we propose a recursive heuristic search algorithm that takes the required QoS and bundling constraints into consideration for service selection. This algorithm has two unique functions: (i) utility function to measure the quality of the selection strategy under consideration, and (ii) acceptance function to limit the selection strategy only to those potential service candidates that have higher chance to satisfy the bundling constraints. Experiments show that our proposed solution can find better solutions than the existing ones without too much extra performance overhead.**

Keywords – service computing, workflow, scheduling, multi-constraints

I. Introduction

With the popularity of services available in the cloud, putting these services together in some workflow is a classic, yet challenging problem. Services with similar functionality from different providers can easily be found in the cloud. One key aspect that differentiates them is their non-functional properties, which include not only QoS and cost, but also service bundling. Service bundling refers to the situation where the subscription of two services have to be done together; such requirement might be imposed by service providers and/or by the alliance group that the providers join in. Service bundling is a reasonable practice from both business and also security viewpoint (e.g. single sign-on).

In this paper, we focus on the service selection problem under multiple constraints, which include service QoS, cost, and service bundling. We first formulate the service selection problem as a multi-constrained selection problem. Then we propose a recursive heuristic search algorithm that takes the required QoS and bundling constraints into consideration for service selection. This algorithm has two unique functions: (i) utility function to measure the quality of the selection strategy under consideration, and (ii) acceptance function to limit the selection strategy only to those potential service candidates that have higher chance to satisfy the bundling constraints. Experiments show that our proposed solution can find better solutions than the existing ones without too much extra performance overhead. This work is important because it facilitates users to customize their workflow with more realistic constraints that they expect from cloud services.

The organization for the rest of this paper is as follows. Section 2 gives a formal description of the problem that this paper wants to address. Assumptions made in the paper are also listed. Section 3 describes the proposed algorithm. Both the ideas behind and the details of the algorithm implementation are given. Section 4 presents the experimental result of our algorithm and shows that it can actually out-perform the existing algorithms. Section 5 gives a survey on research efforts related to this paper. Finally, the paper concludes in Section 6.

II. Problem Definition and Assumptions

In this section, we would like to formulate the service selection problem under multi-constraints formally. To make the formulation clearer, we will first define the notations used in this paper and give the assumptions made.

A. Notations and Workflow Structure Description

Table 1 gives the notations used in the workflow description. Note that in this paper, the term "service" usually refers to "web service" and they might be used interchangeably unless it is explicitly stated.

A workflow W is represented as $<T, E>$, where T is the set of tasks involved in W and E is the set of dependencies among tasks. A task t is a function node in W; it is expected to be performed by some web service offering its expected function. To describe W, we use a recursive definition for tasks and define four basic types of tasks here: (i) atomic, (ii) sequential, (iii) parallel, and (iv) exclusive. The latter three types are shown in Figure 1. And they are defined below.

978-1-4673-7281-7/15 $31.00 © 2015 IEEE
DOI 10.1109/SCC.2015.53

Table 1: Notations Used in Workflow Description

Notation	Explanation
W	Workflow
$T = \{t_i, i = 1, \ldots, n\}$	Tasks of workflow
$E = \{e = (t_i, t_j), i, j = 1, \ldots, n\}$	Dependency of tasks
$S = \{s_{ij}, i = 1, \ldots, n; j = 1, \ldots, m\}$	Candidate services of task
$F = \{f_l(t_j), i = 1, \ldots, k; j = 1, \ldots, n\}$	QoS of tasks
$C = \{c_i, i = 1, \ldots, k\}$	QoS constraints

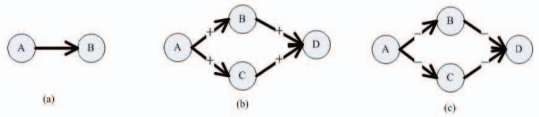

Figure 1: Three Different Types of Composite Tasks. (a) Sequential, (b) Parallel, (c) Exclusive

An atomic task is the basic unit in W to be performed by a web service. It can be described by the attribute vector AtomicTask:

AtomicTask =
<TaskType,TaskID,CandidateServiceList,Edge,SolutionList>

where CandidateServiceList is the set of candidate services for a given atomic task, SolutionList is the selection strategy for the task, and Edge is the dependency of atomic task given below:

Edge = <EdgeType,TaskID,ParentTaskList,ChildTask>

There are five types of dependencies: sequential, AND-Split, AND-Join, OR-Split, and OR-Join.

A sequential task is a composite task; it is composed of two tasks, as is shown in Figure 1(a). Both the start task A and the end task B can be of any arbitrary types, hence the definition is recursive. It is described as follows:

SequentialTask = <TaskType,Task1,Task2,SolutionList>

A parallel task is a composite task; it is composed of four tasks, as is shown in Figure 1(b). Task A and task D are split node and join node respectively; they need to be of atomic type. For task B and task C, they can be of any arbitrary types.

The final type is the exclusive task shown in Figure 1(c). Just like the parallel task, Task A and task D need to be atomic type, while task B and task C can be of any arbitrary types. It is shown in Figure 1(c).

Both the parallel task and the exclusive task are called branch task. The only difference between them is in their execution of branch. The branch task can be described as follows:

BranchTask=<TaskType,StartTask,Branch1Task,Branch2Task,EndTask,SolutionList>

where StartTask and EndTask are of atomic type, and Branch1Task and Branch2Task are of any arbitrary type.

B. QoS Constrants

We use a vector to describe the QoS constraints that a user specifies for a given workflow execution.

Constraint = <TimeConstraint,CostConstraint>

where TimeConstraint and CostConstraint are the maximum time and cost that the user will accept. Given this definition, the QoS of a workflow needs to satisfy the following inequalities:

$$\text{Time}(W) \leqslant \text{TimeConstraint}$$
$$\text{Cost}(W) \leqslant \text{CostConstraint}$$

C. Service Selection Solution

The service selection solution for a give workflow is given as follows:

Solution = <Mapper,Time,Cost,Utility>
Mapper = <TaskID,ServiceID>

where Mapper is the mapping of an atomic task to an available service, Time and Cost are the execution time and cost of the service respectively, and Utility is a value calculated using the time of cost of current task (Refer to Section 3A for its computation formula).

A (web) service can be described by an attribute vector given below:

Service = <ServiceID,TaskID,Time,Cost,Availability,
ExpectedTime,BundlingServiceList>

where the expected time can be computed as follows:

ExpectedTime = Time +Time*(1-Availability)

Here, we assume that the selected service might fail to complete, and we need to find another service to replace it and continue the execution, thus resulting in the ExpectedTime formula above. Note that the failure of the service will affect only the Time, but not the Cost involved. Finally, the service bundling constraints can be described in the form of a set given below:

Bundling = $\{\text{ServiceID}_1, \text{ServiceID}_2, \ldots \text{ServiceID}_N\}$

D. Assumptions

In this paper, there are two assumptions made. The first assumption is related to the structure of the workflow. We only consider three types of structures: sequential, parallel and exclusive. In order word, we do not consider complex structures such as loops. We also assume that branch

structures do not overlap each other. The second assumption is related to the QoS criteria. In this paper, we only consider time and cost as the service QoS.

E. Problem Definition

Given workflow W, N atomic tasks with M candidate services for each task, and the constraint vector C and service bundling requirements, the service selection problem can be formulated as the mapping of tasks to services as follows:

$$\text{R: } t_i \rightarrow s_{ij},\ i \in [1,N],\ j \in [1,M] \text{ such that } f_k(W) \le c_k,\ \forall i \in [1,K]$$
and the bundling constraints is satisfied

Note that the selection problem seeks for solution that satisfies the constraints rather than for an optimal solution.

Input:

We use the overall composite task to represent the workflow; it includes all atomic tasks inside as well as the correlated dependencies among them.

For the constraints, we use time and cost as the QoS criteria and also the service bundling requirements that need to be observed.

Related to the web services that the atomic tasks can be mapped to, each web service has both functional attributes and non-functional attributes. Functional attributes are necessary requirements to map task to service; and non-functional attributes are time, cost and the availability of service. Furthermore, the service bundling requirements are included in the WSDL document of the web service. Finally, the list of possible web services for a given atomic task is included in its task description.

Output:

The output of the problem will give multiple feasible solutions, each of which will describe one possible mapping of tasks to services that satisfies all the constraints. The number of feasible solutions can be configured, and it is highly correlated to the execution time of the algorithm.

III. ALGORITHM

In this section, we first give the key ideas behind our algorithm, followed by the algorithm implementation details.

A. Ideas Behind the Algorithm

There are five aspects of the algorithm that we would like to highlight below.

(i) Non-linear utility function

To measure the quality of the current service selection strategy, we would like to adopt some utility function to quantify it. About its choice of, we reference the ones found in TAMCRA and SAMCRA [12] [13] and give the function used in this paper as follow:

$$\text{utility}(s) = \max\left\{\frac{\text{cost}(s)}{\text{costConstraint}}, \frac{\text{time}(s)}{\text{timeConstraint}}\right\}$$

Obviously, when utility(s) > 1, the strategy fails.

(ii) Non-dominated strategy

Given strategy s_1 and s_2, if {{cost(s_1) ≤ cost(s_2) and time(s_1) < time(s_2)} or {cost(s_1) < cost(s_2) and time(s_1) ≤ time(s_2)}}, s_1 dominates s_2, (or s_2 is dominated by s_1). If s_i is not dominated by any strategies, s_i is a non-dominated strategy. We only keep non-dominated strategy in our list.

(iii) K feasible solutions and relaxation operation

To guarantee the success ratio of the algorithm, each task will keep k feasible selection strategies with the smallest utility values. Furthermore, we will use the relaxation operations (given in the next section) to update the k feasible selection strategies.

(iv) Acceptance function

To avoid too many services from the service bundling that cannot contribute to the final solution, we introduce the acceptance function to decide whether to accept the strategy. The formula for the acceptance function is given as follows:

$$p = \begin{cases} \dfrac{1}{1 + e^{\text{utility} * 2^{\text{diff}}}}, & \text{diff} > 0 \\ 1, & \text{diff} = 0 \end{cases}$$

where p is the probability of acceptance, and *diff* is the number of extra services introduced by the bundling. The function has the following features:

- p is negatively correlated to the utility value and the number of services in the bundling; and
- p is negatively correlated to the growth of the strategy size because the utility value is non-decreasing.

Our experiment shows that our algorithm with the acceptance function gives better success ratio than the one without it.

(v) Divide and conquer

We use divide and conquer to solve the algorithm. Based on the recursive definition of tasks given in Section 2A, our algorithm will recursively find solution for each sub-task, and then merge solutions of sub-tasks to find the final solution.

B. Algorithm Description

In this section, details of the service selection algorithm will first be given, followed by its complexity analysis. Algorithm 1 shows the pseudo-code of service selection algorithm and Algorithm 2 shows the pseudo-code of

relaxation function used. Our implementation will first find solutions for each sub-task recursively, and then merge the solutions of sub-tasks. Relaxation function is used to select *k* strategies.

Algorithm 1: Service Selection Algorithm

Function: Find mapping between atomic task and web services
Input: composition task, QoS constraints
Output: feasible selection solutions

```
1.  proc findFeasibleSolutions(task,constraint):
2.    if task.type=atomic:
3.      then for each service s in candidate service list:
4.        solution.time←s.time
5.        soltuion.cost←s.cost
6.        solution.utility←
          max {(solution.cost)/(constraint.cost),(solution.time)/
          (constraint.time))
7.        solution.mapper←pair(task.taskID,s.serviceID)
8.        for each bundling service a of s:
9.          solution.mapper←pair(a.TaskID,a.ServiceID)
10.       task.solutionList.add(solution)
11.     eliminate dominated solutions in solution list
12.   else if task.type=sequential:
13.     then task1Solutions←
        findFeasibleSolutions(task.Task1,constraint)
14.       task2Solutions←findFeasibleSolutions(task.Task2,constraint)
15.       for each solution s1 in task1Solutions:
16.         for each solution s2 in task2Solutions:
17.           solution.time←s1.time+s2.time
18.           solution.cost←s1.cost+s2.cost
19.           solution.utility←
              max {(solution.cost)/(constraint.cost),(solution.time)/
              (constraint.time))
20.           solution.mapper←merge s1.mapper and s2.mapper
21.           relaxation(task,solution)
22.   else if task.type=parallel or task.type=exclusive:
23.     then seqTask←merge startTask and endTask as a sequential
        task
24.       seqSolutions←findFeasibleSolutoins(seqTask,constraint)
25.       branch1Solutions←
          findFeasibleSolutions(task.Branch1Task,constraint)
26.       branch2Solutions←
          findFeasibleSolutions(task.Branch2Task,constraint)
27.       for each solution s in seqSolutions:
28.         for each solution s1 in branch1Solutions:
29.           for each solution s2 in branch2Solutions:
30.             solution.time←s.time+ max(s1.time,s2.time)
31.             if task.type=parallel:
32.               then solution.cost←s.cost+s1.cost+s2.cost
33.             else if task.type=exclusive:
34.               then solution.cost←s.cost+ max (s1.cost,s2.cost)
35.             solution.utility←
                max {(solution.cost)/(constraint.cost),(solution.time)/
                (constraint.time))
36.             solution.mapper←merge s.mapper,s1.mapper and
                s2.mapper
37.             relaxation(task,solution)
38. return task.solutionList
```

Algorithm 2: Relaxation Function

Function: Check whether current solution can be kept in the solution list
Input: current solution, solution list
Output: true/false

```
1.  proc relaxation(task,solution):
2.      if solution.mapper exists conflicts:
3.        return False
4.      if solution.utility>1:
5.        return False
6.      if solution is dominated by any solution in task.solutionList:
7.        return False
8.      diff←solution.size-task.size
9.      if diff>0:
10.       p←1/(1+e^(solution.utility*2^diff ) )
11.       randomP← random(0,1)
12.       if p<randomP:
13.         return False
14.     if task.solutionList.size<K:
15.       task.solutionList.add(solution)
16.       return True
17.     else if task.solutionList.size=K:
18.       find maximum utility in task.solutionList as maxUtility
19.       if maxUtility>solution.utility:
20.         remove solution with maximum utility in task.solutionList
21.         task.solutionList.add(solution)
22.         return True
23. return False
```

Algorithm 1 gives the algorithm for service selection. The algorithm is based on the "divide and conquer" method, and solves the problem based on different types of tasks. Line 2-11 deal with atomic tasks, and line 12-21 deal with sequential tasks, which recursively solve two tasks first. Line 22-37 solve branch tasks; they combine split task and join task into a temporary sequential task, and then solve the composite task recursively to get the solution for the branch task.

Algorithm 2 gives the relaxation function. It is used to check whether the strategy can be put into the solution list. It checks the following aspects:

- whether the strategy violates the service bundling constraints;
- whether the strategy violates the QoS constraints;
- whether the strategy is dominated by other solutions;
- whether the strategy is accepted by the acceptance function; and
- whether the strategy can be put into the solution list according to its utility value.

To study the complexity of our algorithm, let us denote the number of atomic tasks be N, the number of candidate services per task be M, and the number of solutions for each task be K.

For Algorithm 2, the time complexity of the relaxation function is $O(N+K)$.

For Algorithm 1, its time complexity is different based on different task types:

- For atomic task, it is $O(M)$.
- For sequential task,
  - Best time complexity is $O(K^2*(N+K))$
  - Worst time complexity is $O(N*K^2*(N+K))$
- For branch task,
  - Best time complexity is $O(K^3*(N+K)+M^2)$
  - Worst time complexity is $O(N*K^3*(N+K)+N*M^2)$

Assume that M, N, K are of the same scale, the worst time complexity is approximately $O(N^5)$, and the best time complexity is $O(N^3)$.

With regard to its space complexity, it is $O(N*M+N*K*N)$ (or $O(N*M+N^y*y)$)

IV. EXPERIMENT

In this section, results of the performance study of the service selection algorithm under different input parameters will be given. Table 2 gives the experiment input parameters and the default values used. For the output, we mainly focus on two measurements: approximation degree and execution time.

The first one, approximation degree, refers to the degree of approximation between the feasible strategy and the workflow constraint values. It is obtained as follows. First, the time, solutionTime, and cost, solutionCost, of a feasible strategy under the workflow QOS constraints timeConstraint and costConstraint are calculated. Then the approximation degrees, timeApproximation and costApproximation, are obtained as follows:

$$\text{timeApproximation} = \frac{\text{solutionTime}}{\text{timeConstraint}}$$

$$\text{costApproximation} = \frac{\text{solutionCost}}{\text{costConstraint}}$$

According to the utility function defined in Section 3A, a feasible strategy should have utility value equal to the maximum of timeApproximation and costApproximation. The smaller the utility value is, the lower will be the probability of violating the specified constraints, which implies that the strategy is better. In an extreme case, if the value of timeApproximation or that of costApproximation is greater than 1, the strategy will not be feasible because of violation of the QoS constraints.

The second measurement is the execution time. We measure the performance of the service selection algorithm under different parameter settings. For each parameter setting, the algorithm is executed 20 times and the average time will be reported.

Table 2: Input Parameters in the Experiment Study

Parameter	Explanation	Default
taskNum	# of tasks	50
serviceNumPerTask	# of services per task	50
solutionNum	# of solutions per task	50
sequentialRatio	Sequential structure ratio	0.5
bundlingSize	Size of bundling	5
bundlingRatio	Services in bundling ratio	0.5
timeConstraintDegree	Time constraint degree	1
costConstraintDegree	Cost constraint degree	1
minTaskTime	Shortest execution time(s)	100
maxTaskTime	Longest execution time(s)	1000
minTaskCost	Lowest cost	10
maxTaskCost	Highest cost	100
SDRatio	Standard deviation ratio	0.1
availMean	Availability mean	0.9
availSD	Availability SD	0.05

In our experiments, we choose the algorithms proposed by Yu [15], the deadline distribution (DD) algorithm and its variant, the budget distribution (BD) algorithms, for comparison. We picked these two algorithms because their functionalities are quite similar to our service selection problem for workflow. Detailed description of these two algorithms can be found in [15].

A. Results

In this section, results of all the three algorithms, our SA (service selection algorithm), DD, and BD algorithms on different number of tasks and web services will be presented below. And the measurement matrices are the approximation degrees of time and cost, and also the execution time. Note that sensitivity studies of SA with respect to the constraint degree, solution list size, service bundling ratio, and workflow structures were also performed. However, due to the space limitation, they are not included here (They can be available upon request).

B. Task Number

The main focus of this set of experiments is to study the performance of SA, DD, and BD under different task numbers. The number of atomic tasks in the workflow is changed from 10 to 100, and the other parameters used are the default values shown in Table 2.

The result of the time and cost approximation degrees (with formula given in the beginning of Section 4) of the three algorithms with different task numbers is shown in Figure 2. From this figure, we can see that our selection algorithm SA can give a superior viable strategy over the other two algorithms (DD and BD), with approximation degree of time and cost being lower than those of the other

two. Furthermore, the figure shows that the approximation degrees (both time and cost) of all the three algorithms are quite insensitive to the number of tasks in the workflow, except for the time approximation degree which decreases with the increase of the number of tasks initially and then levels off. This shows that SA is more effective in handling more complex workflow.

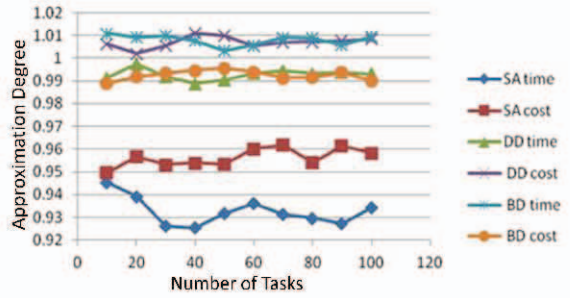

Figure 2: Comparison of Approximation Degrees of Algorithms under Different Task Numbers

The result of the execution time of all the three algorithms with different number of tasks is shown in Figure 3. As is expected, the execution time required by our SA algorithm is higher than that of DD and BD algorithms. This is due to its higher algorithmic complexity. However, it is still within the acceptable range for reasonable size workflow with about 100 tasks.

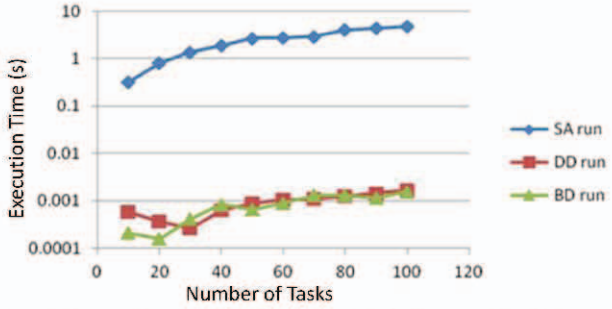

Figure 3: Comparison of Execution Time of Algorithms under Different Number of Tasks in the Workflow

C. Web Service Number

The main focus of this set of experiments is to study the performance of SA, DD, and BD when the number of possible web services available for each task changes. In the experiment, the number of web services for each task is changed from 10 to 100, and the other parameters used are the default values shown in Table 2.

The result of the time and cost approximation degrees of the three algorithms with different number of services per task is shown in Figure 4. Just like the previous situation of varying the total number of tasks in the workflow, the figure shows that our service selection SA algorithm performs superior than the other two algorithms, both in terms of the time and cost approximation degree. Furthermore, when the number of services available per task increases, our SA algorithm decreases first, and then levels off. Once again, this shows the effectiveness of SA in handling complex workflows with larger number of possible service choices.

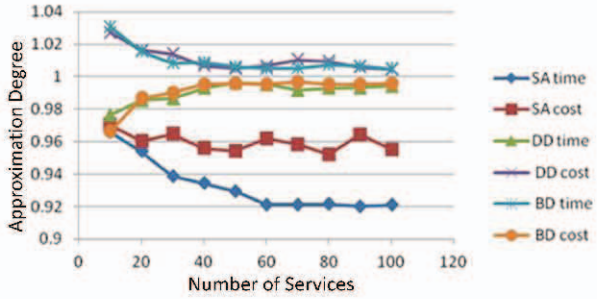

Figure 4: Comparison of Approximation Degrees of Algorithms under Different Number of Possible Services per Task

The result of the execution time of all the three algorithms with different number of service choices per task is shown in Figure 5. Similar to the situation in Figure 3, the execution time overhead of SA is higher than those of DD and BD algorithm due to its higher computational complexity. And this execution time overhead of SA is still within the acceptable range for reasonable size workflow with about 100 tasks.

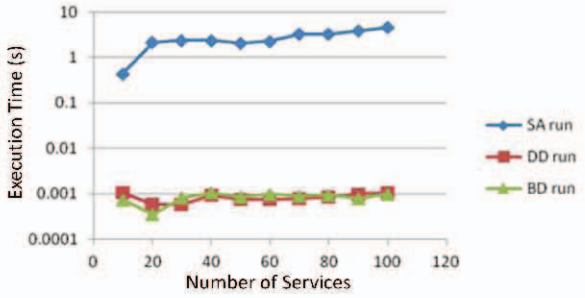

Figure 5: Comparison of Execution Time of Algorithms under Different Number of Services Available per Task

V. RELATED WORK

There are three types of research efforts related to this paper. They are: (i) multi-constrained path selection, (ii) workflow scheduling, and (iii) service selection for service composition.

Multi-constrained path selection problem is originated from computer network routing subject to constraints such as bandwidth, packet loss rate, propagation delay, transmission, and price. Its goal is to ensure that the propagation from the source point to the end point satisfies the given constraints. According to the survey by Garroppo [1], there are three basic types of multi-constrained path selection problems with different objectives: (i) multi-constrained optimal path (MCOP), (ii) multi-constraint path

(MCP), and (iii) restricted shortest path (RSP). Our research is closer to the MCP problem because the emphasis is on the satisfaction of the constraints, not the optimal solution. Ours is different from the MCP problem because while our solution seeks for solutions that satisfy the constraints of the overall problem, MCP focuses on the satisfaction of each individual path constraint.

Given that the algorithm is an NP problem, there are three typical approaches to solve it. The first one is the exact search using label setting, label correcting, ranking, and two-phase method. The basic idea behind is to extend the Bellman-Ford and Dijkstra's algorithm [2] [3] [4] [5] [6] by defining relations of domination to present the relationship of paths with multiple criteria. The second one is the approximate search, by converting the original NP problem into an approximate polynomial-time problem. The basic idea behind the approximation is to discrete criteria values of edges in the graph [7]. Others such rounding and scaling [8] [9], and interval partition are also used [10]. The third one is the heuristic search.

It is difficult to find the perfect balance between approximation and execution time of the algorithm. Therefore, many researchers try to use heuristic search algorithm to solve the problem. Here we describe three typical algorithms. Jaffe [11] proposes to use a utility function for the heuristic search algorithm, but it has strong requirements that all the criteria need to be positively co-related. TAMCRA [12] and SAMCRA [13] are two algorithms based on the ideas of non-linear utility function, K shortest paths, and non-dominated path. Our solution actually references these two algorithms and makes modification to them (as described in Section 3A) due to the difference in the emphasis (i.e. individual path constraints vs. overall workflow constraints).

For workflow scheduling, there are two basic types: deadline constraints and budget constraints. Good examples are back-tracking algorithm by Menasce and Casalicchio [24], and deadline distribution algorithm by Yu [21]. The latter one is actually quite close to what this paper wants to do (except that they focus on single criterion while ours focuses on multiple criteria), hence, we choose it and its variant, budget constraint algorithm for comparison purposes.

For service selection in service composition, Yu [16][17][18][19] proposed a series of method using combination and/or graph model to find solutions. Ours is different from these works in the following aspects: (i) Yu's graph model is on web service graph while ours focuses on task graph, (ii) Yu's work keeps multiple paths in each service while ours keep multiple solutions in each task, and finally (iii) Yu's work gives feasible paths while our work gives selection solutions for the entire workflow.

VI. Conclusion and Future Work

In this paper, we propose a new algorithm to address the service selection problem for workflow. Two key unique features of our algorithm are the service bundling and the acceptance function. Experiment results show that solutions obtained from our algorithm can out-perform those from the existing deadline distribution algorithm and the budget distribution algorithm, though with higher execution time. As for future work, we will focus on the reduction of the performance overhead and also on the handling of more complex workflow structures including loops.

References

[1] R. Garroppo, S. Giordano, and L. Tavanti. A Survey on Multi-Constrained Optimal Path Computation: Exact and Approximate Algorithms. Computer Networks, 2010, 54:3081–3107.

[2] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein. Introduction to Algorithms. 2nd edition. MIT Press, 2001.

[3] P. Hansen. Multiple Criteria Decision Making: Theory and Application. Economics and Mathematical Systems, 1980, 177:109-127.

[4] E. Martins. On a Multi-Criteria Shortest Path Problem. Journal of Operational Research, 1984, 16:236–245.

[5] X. Gandibleux, F. Beugnies, and S. Randriamasy. Martins' Algorithm Revisited for Multi-Objective Shortest Path Problems with a Maxmin Cost Function. Journal of Operations Research, 2006, 4:47–59.

[6] D.A. Van Veldhuizen, and G.B. Lamont. Multiobjective Evolutionary Algorithms: Analyzing the State-of-the-Art. Evolutionary Computation, 2000, 8:125–147.

[7] S. Sahni. General Techniques for Combinatorial Approximation. Operations Research, 1977, 25:920–936.

[8] S. Chen, and K. Nahrstedt. On Finding Multi-Constrained Paths. Proceedings of IEEE International Conference on Communications, 1998, 2:874–879.

[9] M. Song, and S. Sahni. Approximation Algorithms for Multiconstrained Quality-of-Service Routing. IEEE Transactions on Computers, 2006, 55: 603–617.

[10] A. Orda, A. Sprintson. Precomputation Schemes for QoS Routing. ACM Transactions on Networking, 2003, 11:578–591.

[11] J. M. Jaffe. Algorithms for Finding Paths with Multiple Constraints. Networks, 1984, 14:95-116.

[12] H. De Neve and P. Van Mieghem. TAMCRA: A Tunable Accuracy Multiple Constraints Routing Algorithm. Computer Communications, 2000, 23:667-679.

[13] P. Van Mieghem, H. De Neve and F.A. Kuipers. Hop-by-hop Quality of Service Routing. Computer Networks, 2001, 37:407-423.

[14] Daniel A. Menasce, and Emiliano Casalicchio. A Framework for Resource Allocation in Grid Computing. Proceedings of the IEEE 12th Annual International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems, 2004, 259-267.

[15] J. Yu, Rajkumar Buyya, and Chen Khong Tham. Cost-Based Scheduling of Scientific Workflow Applications on Utility Grids. Proceedings of 1st International Conference on e-Science and Grid Computing, 2005, 8-17.

[16] Tao Yu, Yue Zhang, and Kwei-Jay Lin. Efficient Algorithms for Web Services Selection with End-to-End QoS Constraints. ACM Transactions on the Web (TWEB), 2007, 1:1-26.

[17] Tao Yu, and Kwei-Jay Lin. Service Selection Algorithms for Composing Complex Services with Multiple QoS Constraints. Proceedings of International Conference on Service-Oriented Computing (ICSOC), 2005, 130-143.

[18] Tao Yu, and Kwei-Jay Lin. Service Selection Algorithms for Web Services with End-to-End QoS Constraints. Information Systems and E-Business Management, 2005, 3:103-126.

[19] Tao Yu, Tao, and Kwei-Jay. Lin. A Broker-Based Framework for QoS-Aware Web Service Composition. Proceedings of IEEE International Conference on E-Technology, e-Commerce and e-Service (EEE), 2005, 22-29.