Optimizing Distributed Execution of WS-BPEL Processes in Heterogeneous Computing Environments

Qishi Wu[1], Yi Gu[1], Liang Bao[2], Wei Jia[2], Huichen Dai[2], and Ping Chen[2]

[1] Dept of Computer Science, University of Memphis, Memphis, TN 38016, USA
{qishiwu,yigu}@memphis.edu

[2] College of Software, XiDian University, Xi'an, Shanxi, 710071, China
{baoliang,weijia,huichendai,pingchen}@mail.xidian.edu.cn

Abstract. Workflow-structured Web service composition is an emerging computing paradigm for constructing next-generation large-scale distributed applications within and across organizational boundaries. Mapping such application workflows in heterogeneous environments and optimizing their performance in terms of quick response and high scalability are vital to the success of these distributed applications. Workflows with complex execution semantics and dependencies are typically modeled as directed acyclic graphs. We construct cost models to estimate data processing and transfer overheads and formulate the restricted workflow mapping for minimum total execution time as an NP-complete optimization problem. We propose a heuristic approach to this problem that recursively computes and maps the critical path to network nodes using a dynamic programming-based procedure. The performance superiority of the proposed approach is illustrated by an extensive set of simulations and further verified by experimental results from a real network in comparison with existing methods.

Keywords: WS-BPEL, workflow mapping, optimization, heuristic algorithm.

1 Introduction

As the number of Web services of wide variety grows rapidly in the Internet, Web service composition has become an important computing paradigm for constructing next-generation large-scale distributed applications within and across organizational boundaries. Successful business operations require an efficient and flexible scheme for pooling globally available Web service-based resources together to quickly adapt to various customer needs and dynamic market conditions. WS-BPEL (Web Service Business Process Execution Language) is now a de facto specification for Web service composition.

A WS-BPEL application based on composite Web services features complex execution semantics and is typically coordinated by a single node referred to as a centralized orchestrator. The WS-BPEL process is usually designed by application developers according to certain business logics and manually deployed on a WS-BPEL engine. Fig. 1 diagrammatizes a typical execution setup of WS-BPEL process: a request is sent to the centralized WS-BPEL engine, which orchestrates the invocation of Web services located in different Web containers. As pointed out in [7], instead of transferring data directly from the point of generation to the point of consumption, this execution model

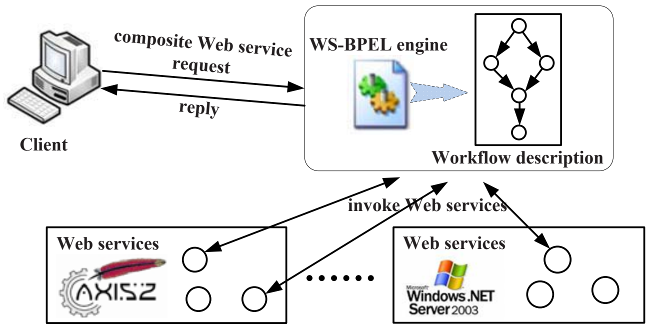

Fig. 1. A typical execution setup of WS-BPEL process

uses the engine as a central intermediator to exchange data between Web services, resulting in unnecessary network traffic. In addition, a Web service could generate a large volume of data that are irrelevant to the composite service but still need to be transferred to the engine where they are eventually discarded, hence causing unnecessary workload in the network. This client-server communication model poses an inherent performance limitation on scalability: the system performance degrades significantly as the traffic and workload increase in heterogeneous and cross-organization environments.

We propose a distributed approach to execute a WS-BPEL process that overcomes the above performance limitation by constructing and mapping a WS-BPEL workflow with direct inter-web service data transfer to a set of network nodes. The workflow is constructed by a static analyzer that takes three steps: (i) load a WS-BPEL process and transform it into a customized memory structure of WS-BPEL process (Java classes); (ii) load the generated memory representation of BPEL process and transform it into a BCFG (BPEL Control Flow Graph) representation; and (iii) apply a revised algorithm in [3] to remove control flow edges in BCFG and insert control and data dependence edges, which generates the final Program Dependence Graph (PDG) representation of the WS-BPEL process. We apply static analysis techniques of multi-threaded programs to BPEL process and some composition structure patterns, such as AND split (fork), XOR split (conditional), loop and AND join (merge), and XOR join (trigger), which can be modeled and represented in our tCFG (threaded Control Flow Graph)-structured BCFG [3, 10]. Note that loop operations can be managed by unfolding the cycles as proposed in [31]. We would like to point out that the activities in BPEL considered here are synchronous and stateless Web services invocations. Invocations of asynchronous and stateful Web services are more complicated and are out of the scope of this paper.

The workflow mapping may be subject to some restrictions and involves two types of graphs: (i) a Directed Acyclic Graph (DAG) that models the workflow of a WS-BPEL process, where each vertex represents an activity and each directed edge represents the data transfer or execution dependency between two activities (also referred to as PDG [12]); (ii) a directed weighted graph that represents an underlying physical computer network, where Web services are deployed on heterogeneous computing nodes that are connected by network links with different bandwidths. The topology of the computer network may not be complete in a dedicated network environment or even in the Internet due to different administrative policies and firewall settings. Furthermore, the Web services in the Internet come and go dynamically while those deployed in high-speed reliable enterprise intranet are more stable and predicable. Mapping such workflows into heterogeneous computing environments and optimizing their end-to-end performance are crucial to ensuring the success of business processes requiring quick response and the maximum utilization of system resources.

The workflow-structured WS-BPEL process requires distributed execution of complex Web service components with inter-component communications using massively dispersed computing and networking resources to support business collaborations in various domains. The workflow mapping objective is to strategically select an appropriate set of network nodes that host different Web services in the physical computer network and assign each activity in the WS-BPEL process to one of those selected nodes to achieve the Minimum Execution Time (MET) of the process for fast response. Certain activities in a WS-BPEL process might be restricted to some specific computing nodes providing the corresponding Web services. We refer to such activities as restricted activities as opposed to free activities, which can be mapped onto any computing nodes. We allow multiple activities to be mapped onto the same node and the computing resources of that node are shared in a fair manner by those activities running concurrently on that node. Note that activities assigned to the same node do not share computing resources if their executions do not overlap due to the dependency or unavailability of input data. Similarly, the bandwidth of a network link is fairly shared by multiple data transfers that take place concurrently on the same link. We formulate the workflow mapping with arbitrary node reuse and certain mapping restrictions as an NP-complete optimization problem, and propose a heuristic approach, *restricted Recursive Critical Path* (rRCP), which is modified from the Recursive Critical Path (RCP) algorithm in [29] by taking the mapping restrictions into consideration.

The rest of the paper is organized as follows. We conduct an extensive survey of WS-BPEL processes and workflow mapping in Section 2. We construct mathematical models and formulate the problem in Section 3. In Section 4, we design the rRCP algorithm for workflow mapping to achieve MET. The implementation details and performance evaluations are presented in Section 5. We conclude our work in Section 6.

2 Related Work

Web services have found pervasive applications in different domains over wide-area networks [17, 22, 27]. Guo *et al.* proposed the ANGEL model for service composition and adopted a redundant mechanism in ANGEL to improve system availability [17]. In [27], Shin *et al.* proposed a simple heuristic solution to Web service composition where the highest search priority is given to services providing the largest number of new responses. Li *et al.* proposed a general purpose Web Service Management System in [22] that enables execution optimization of composite services through multiple engines. In the Symphony project [24], Mangla [25] partitioned a composite Web service written as a single WS-BPEL program into an equivalent set of decentralized processes using a new code partitioning algorithm based on PGD to minimize communication costs and maximize the throughput of multiple concurrent instances of the input program. However, Mangla's work does not consider the situation where multiple services may be executed on a single server. Yildiz *et al.* proposed an efficient process transformation technique that converts a process conceived for centralized execution to a set of nested processes to be deployed on dynamically bound services [30]. Other research efforts along this line include the static optimization of WS-BPEL process [3] and batch invocation of Web services [10], where the former applies static analysis to the WS-BPEL process to identify "concurrent branches" and the latter reduces the number of connections by forming batch invocation request to implement "one request, many invocations of Web services". Security and performance issues of BPEL processes were studied in [5] and [6], respectively.

The workflow mapping problem in distributed network environments under different constraints has been extensively studied by researchers in various disciplines [4, 8, 9, 15, 28] and continues to be the focus of distributed computing due to its theoretical significance and practical importance. Zhu *et al.* proposed a model of overlay network with linear capacity constraints (LCC) [32], which incorporates correlated link capacities by formulating shared bottlenecks as linear constraints of link capacities. Guerin *et al.* tackled an all hops optimal path problem to minimize end-to-end delay or maximize bandwidth with a limit on the maximum number of possible hops [16]. Among the traditional workflow mapping problems in theoretical aspects of computing, subgraph isomorphism is known to be NP-complete [14] while the complexity of graph isomorphism still remains open. Many special cases of graph isomorphism under different topology constraints on the mapped (workflow) or mapping (network) graphs can be solved in polynomial time, including isomorphism between planar graphs [18] and bounded valence graphs [23]. The mapping computational complexity could also be reduced by introducing an adequate representation of the search space and process, and pruning unprofitable search paths in the search space [13].

Many research efforts have been focused on static scheduling algorithms for multiprocessors that are considered as identical resources. Kwok *et al.* proposed Dynamic Critical-Path (DCP) scheduling algorithm [20] to map task graphs with arbitrary computation and communication costs to a multiprocessor system with an unlimited number of identical processors in a fully-connected network. A task graph scheduling scheme for streaming data, *Streamline*, which places a coarse-grain dataflow graph on available grid resources, is proposed in [2] to improve the performance of graph mapping for streaming applications with various demands in distributed network environments. Most graph mapping or task scheduling problems in grid environments assume complete networks with heterogeneous resources. Similar mapping problems are also studied in the context of sensor networks. Sekhar *et al.* proposed an optimal algorithm for mapping subtasks onto a large number of sensor nodes based on an A^* algorithm, and also proposed a greedy A^* algorithm to reduce the complexity of the original optimal solution accounting for the limited energy of each sensor node [26].

3 Cost Model and Problem Formulation

We model the workflow of a WS-BPEL process as a task graph $G_t = (V_t, E_t)$, $|V_t| = m$, where vertices represent different computing activities: $w_0, w_1, \ldots, w_{m-1}$. The data or control dependency between a pair of adjacent vertices w_i and w_j is represented by a directed edge $e_{i,j}$ with data size $z_{i,j}$ between them and the entire workflow is modeled as a DAG starting from the source activity w_0 and ending at the destination activity w_{m-1}. An intermediate activity w_i cannot start any processing until it receives all required input data from its preceding activities. The computational complexity of an activity is modeled as a function $f_{w_i}(\cdot)$ on the total aggregated input data z_{w_i}, and the activity sends results to its succeeding activities once it completes the required processing. We estimate the computing time of an activity w_i running on network node v_j as $T_{comp}(w_i, v_j) = \frac{f_{w_i}(z_{w_i})}{p_j}$. The actual runtime of an activity does not only depend on the total aggregated incoming data size and computational complexity, but also the capacity of system resources deployed on the selected nodes as well as their availability during the runtime. Note that for an application with multiple source or destination activities, we could convert it to this model by inserting a virtual starting or ending activity of complexity zero connected to all source or destination activities with zero-sized output or input data transfers.

Table 1. Workflow and network parameters

Parameters	Definitions
$G_t = (V_t, E_t)$	task graph
m	number of activities in the workflow
w_i	the i-th computing activity
$e_{i,j}$	dependency edge from activity w_i to w_j
$z_{i,j}$	data size of dependency edge $e_{i,j}$
z_{w_i}	aggregated input data size of activity w_i
$f_{w_i}(\cdot)$	computational complexity of activity w_i
$G_c = (V_c, E_c)$	computer network graph
n	number of nodes in the network graph
v_i	the i-th network or computer node
v_s	source node
v_d	destination node
p_i	normalized computing power of v_i
$l_{i,j}$	network link between nodes v_i and v_j
$b_{i,j}$	bandwidth of link $l_{i,j}$
$d_{i,j}$	minimum link delay of link $l_{i,j}$
$T_{comp(w_i,v_j)}$	computing time of activity w_i running on node v_j
$T_{trans(z_{h,k},l_{i,j})}$	transfer time of data $z_{h,k}$ over link $l_{i,j}$
T_{total}	total execution time required for a WS-BPEL process

The underlying computer network is modeled as an arbitrary weighted graph $G_c = (V_c, E_c)$ consisting of $|V_c| = n$ computer nodes interconnected by directed communication links represented by a matrix $L[n \times n]$. The processing power of a computer node is a complex notion that combines a variety of host factors such as processor frequency, bus speed, memory size, I/O performance, and presence of co-processors. For simplicity, we use a normalized variable p_i to represent the overall processing power of a network node v_i without specifying its detailed system resources. There are two parameters, bandwidth (BW) $b_{i,j}$ and minimum link delay (MLD) $d_{i,j}$, associated with a network link $l_{i,j} \in L$ $(i,j \in n)$ between nodes v_i and v_j. The estimated time of transferring the data $z_{h,k}$ between modules w_h and w_k over the network link $l_{i,j}$ can be calculated as $T_{trans}(z_{h,k}, l_{i,j}) = \frac{z_{h,k}}{b_{i,j}} + d_{i,j}$.

For convenience, we tabulate in Table 1 the notations we define in the above workflow and network models to facilitate the problem formulation. A mapping example is illustrated in Fig. 2, where activities w_h and w_g are mapped to network node v_i, w_l is mapped to v_j, and w_r, w_t and w_u are mapped to v_k. The dashed arrows represent the data or control dependencies in a WS-BPEL process and the solid arrows represent the communication links between network nodes. Activity w_r cannot start its execution until it receives all required data from its preceding activities w_h and w_l. Note that w_r does not receive data directly from w_h and w_l in the centralized execution model, where a central engine is responsible for all data communication. Activity w_r aggregates incoming data and performs a predefined computing routine whose complexity is modeled as function $f_{w_r}(\cdot)$ on the total aggregated input data z_{w_r} and sends out the results to its succeeding activities upon finishing its processing.

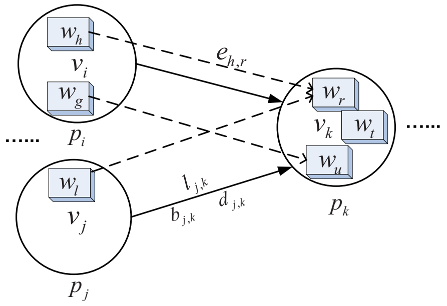

Fig. 2. A mapping example

The mapping objective is to map all the activities of a WS-BPEL process onto an appropriate set of computer nodes to minimize total execution time T_{total}, which is determined by its critical path (CP), i.e. the longest path of the workflow. Once a mapping scheme is determined, one may calculate T_{total} by adding up all the computing time and transfer time incurred on the CP, which can be estimated as:

$$\begin{aligned} T_{Total} &= \sum_{w_i \in CP} T_{comp}(w_i, v_h) + \sum_{e_{j,k} \in CP} T_{trans}(z_{j,k}, l_{f,g}) \\ &= \sum_{w_i \in CP} \frac{f_{w_i}(z_{w_i})}{p_h} + \sum_{e_{j,k} \in CP} \left(\frac{z_{j,k}}{b_{f,g}} + d_{f,g} \right) \end{aligned} \quad (1)$$

We assume that the inter-activity transfer time on the same node is negligible considering that the in-memory transfer rate is much faster than across networks.

The proposed workflow mapping problem considers node reuse and resource share. In the underlying network, multiple services might be mapped onto the same node but some services are only available on certain nodes. To simplify the time estimation of an activity, we combine the time cost for service invocation and activity processing.

4 Restricted Workflow Mapping Algorithm

The workflow mapping or scheduling problem is known to be NP-complete [2, 21] even on two processors without any topology or connectivity restrictions [1]. The mapping problem in this paper considers mapping restrictions: some activities in the WS-BPEL process can only be mapped onto certain nodes with specific resources to support the execution of those restricted activities. We modify and adapt the Recursive Critical Path (RCP) algorithm in [29] to this new problem and propose a restricted version of RCP algorithm, referred to as *restricted Recursive Critical Path* (rRCP).

4.1 rRCP Algorithm

rRCP features a recursive optimization strategy. In each round, it chooses the CP based on the previous round of calculation as shown in Fig. 3 and maps it to the network nodes using a dynamic programming-based procedure until the mapping results converge to an optimal or suboptimal point or a certain termination condition is met. The mapping restrictions are taken into consideration when each activity is being mapped.

The pseudocode of the rRCP mapping scheme is provided in Alg. 1. The initial mapping assumes resource homogeneity and connectivity completeness in computer network, that is, the computer network is considered as complete with identical computer nodes and communication links. Thus, we only need to consider the workflow when calculating the initial computing and transfer time cost components on each activity and over each dependency edge, respectively. With the initial time cost components in workflow G_t^1, we find its CP P_1 using a procedure defined in $FindCriticalPath()$, which essentially finds the longest path in a DAG. From this point on, we remove the assumption on resource homogeneity and connectivity completeness, and map the current CP, i.e P_1 to the real computer network using a dynamic programming-based pipeline mapping algorithm $MapCriticalPath()$ with arbitrary node reuse as well as mapping restrictions for MET. The activities that are not located on the CP, referred to as branch or non-critical activities, are mapped to the network nodes using a procedure defined in $MapNonCriticalActivity()$. Based on the current mapping, we compute a new CP using updated time cost components in G_t^i and calculate a new MET. The above steps are repeated until a certain condition is met, for example, the difference between two METs of two consecutive iterations is less than a preset threshold.

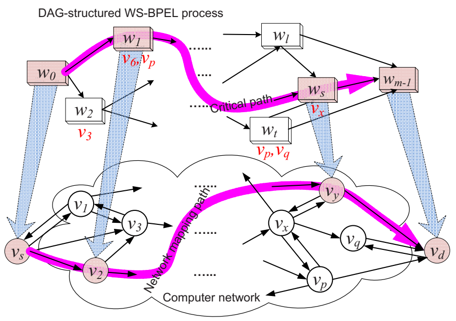

Fig. 3. An example of WS-BPEL process mapping using rRCP algorithm

The complexity of the rRCP algorithm is $O(k(m + |E_t|) \cdot |E_t|)$, where m represents the number of activities in the WS-BPEL process, $|E_t|$ and $|E_c|$ denote the number of dependency edges in the workflow and communication links in the computer network, respectively, and k is the number of iterations where CPs are calculated and mapped.

The algorithm for CP calculation is well studied and documented in the literature. The algorithms for CP mapping $MapCriticalPath()$ and non-critical activities mapping $MapNonCriticalActivity()$ are similar to those proposed in [29] using a dynamic programming-based and a greedy-based procedure, respectively. Note that when

Algorithm 1. rRCP(G_t, G_c, v_s, v_d)

```
1: $MET_0 = MET_{max} = MaxValue$;
2: Create $G_c{}^*$ by assuming resource homogeneity and connectivity completeness in $G_c$;
3: Calculate initial cost components for $G_t^1$ based on $G_c{}^*$;
4: $P_1 = FindCriticalPath(G_t^1, w_0, w_{m-1})$;
5: $MET_1(G_t^1) = \sum(T_{comp}(P_1) + T_{trans}(P_1))$;
6: $i = 1$;
7: while $|MET_i - MET_{i-1}| \geq Threshold$ do
8:     Call $MapCriticalPath(P_i, G_c, v_s, v_d)$ to map the activities on CP $P_i$ to network $G_c$ with
       mapping restrictions;
9:     Call $MapNonCriticalActivity(P_i, G_t^i, G_c, v_s, v_d)$ to map the activities not on CP to network
       $G_n$ with mapping restrictions;
10:    $i = i + 1$;
11:    Calculate new time cost for $G_t^i$ based on the current mapping;
12:    $P_i = FindCriticalPath(G_t^i, w_0, w_{m-1})$;
13:    $MET_i(G_t^i) = \sum(T_{comp}(P_i) + T_{trans}(P_i))$;
14: return $MET_i(G_t^i)$.
```

multiple activities are assigned to the same computer node, resources on this node are shared among these activities only if they can run concurrently. Two activities are considered "independent" if there does not exist any path between them, and only independent activities may run concurrently on the same node. It is worth pointing out that the time calculation based on this resource share strategy is still an approximation since the execution start time of an activity depends on the arrival time of its latest input data. Therefore, even independent activities deployed on the same node may not run concurrently if their execution start and end times do not overlap. Note that some activities in the WS-BPEL process can only be executed on a subset of computers in the network, which imposes additional constraints for selecting nodes. In Fig. 3, the IDs listed under an activity are the IDs of those computer nodes that have been ruled out for deploying that activity. For example, activity w_1 cannot be mapped to nodes v_6 and v_p.

5 Performance Evaluation

Despite the widespread application of WS-BPEL processes in a wide spectrum of fields, there still lacks a standardized benchmark for evaluating their performances. We present below the results from both simulations and real network experiments to illustrate the performance superiority of the proposed mapping solution over existing algorithms.

5.1 Simulation Results

The proposed rRCP algorithm is implemented in C++ and runs on a Windows XP desktop PC equipped with a 3.0 GHz CPU and 2 Gbytes memory. For performance comparison purposes, we also implement the other three algorithms, namely, Greedy A^*, Streamline, and Naive Greedy. A^* algorithm is a static allocation scheme proposed by

Sekhar *et al.* [26], which maps the subtasks of a DAG-like workflow onto a large number of sensor nodes. A greedy A^* algorithm, which is specifically designed to reduce the complexity of the A^* algorithm, explores only the least-cost path of the search tree in

Table 2. Simulation-based performance comparison of MET among four algorithms

Prb	Problem Size	MET (s)			
Idx	m, $\lvert E_t \rvert$, n, $\lvert E_c \rvert$	rRCP	Greedy A^*	Streamline	Naive Greedy
1	4, 6, 6, 35	1.05	1.08	1.15	1.08
2	6, 10, 10, 96	1.15	1.85	1.33	1.23
3	10, 18, 15, 222	1.59	1.89	1.95	1.92
4	13, 24, 20, 396	1.49	2.16	2.09	2.19
5	15, 30, 25, 622	2.29	2.57	2.67	2.32
6	19, 36, 28, 781	1.41	1.75	1.71	1.57
7	22, 44, 31, 958	1.17	1.43	1.61	1.74
8	26, 50, 35, 1215	3.14	3.76	3.83	3.57
9	30, 62, 40, 1598	4.40	5.38	5.41	4.92
10	35, 70, 45, 2008	4.24	5.19	5.99	4.48
11	38, 73, 47, 2200	3.21	3.64	5.16	4.40
12	40, 78, 50, 2478	2.69	3.73	4.31	3.17
13	45, 96, 60, 3580	1.41	1.52	2.07	1.81
14	50, 102, 65, 4220	1.99	5.01	3.87	4.59
15	55, 124, 70, 4890	7.64	12.35	9.49	10.84
16	60, 240, 75, 5615	9.98	11.45	15.07	13.55
17	75, 369, 90, 8080	11.57	19.37	14.68	15.13
18	80, 420, 100, 9996	24.83	31.73	30.69	28.50
19	90, 500, 150, 22496	17.33	24.74	20.77	21.37
20	100, 660, 200, 39990	35.79	41.37	38.66	39.29

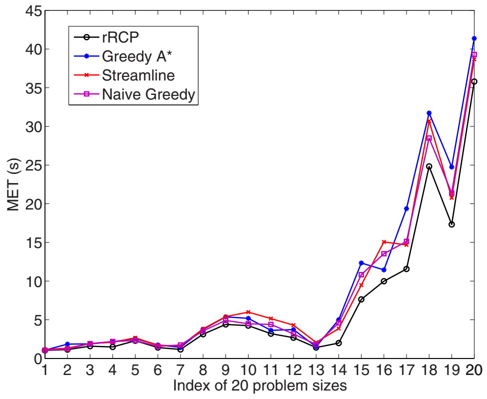

Fig. 4. Simulation-based MET performance comparison among the four algorithms

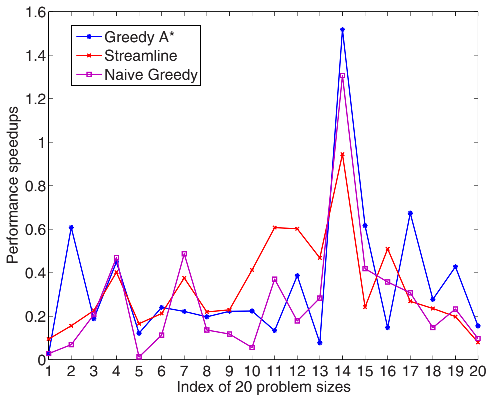

Fig. 5. Performance speedups of rCRP over the other three algorithms

Table 3. Simulation-based MET performance comparison of mean and standard deviation

Prb	Problem Size	MET (s)							
		rRCP		Greedy A^*		Streamline		Naive Greedy	
Idx	m, $\|E_t\|$, n, $\|E_c\|$	Mean	Std Div	Mean	Std Div	Mean	Std Div	Mean	Std Div
1	4, 6, 6, 35	0.5280	0.3824	0.5600	0.4121	0.5600	0.4105	0.5600	0.4121
2	10, 18, 15, 222	1.2750	0.4832	1.6120	0.4893	1.4680	0.7200	1.5530	0.6351
3	15, 30, 25, 622	2.0600	0.4323	2.0090	0.4967	2.4430	0.5029	2.3080	0.7288
4	22, 44, 31, 958	2.1160	0.5808	2.7720	0.8537	2.9720	0.8707	2.3200	0.7722
5	30, 62, 40, 1598	2.9780	1.4326	3.7160	1.8126	4.2450	1.4535	3.1270	1.4951
6	40, 78, 50, 2478	3.0360	1.2833	3.8980	1.4230	4.7650	1.5064	3.4040	1.8909
7	50, 102, 65, 4220	3.6840	1.1972	4.6110	1.6615	5.2930	1.3297	3.8940	1.2713
8	60, 240, 75, 5615	8.8360	2.0971	11.9580	3.0178	12.3640	2.2823	10.0900	2.3685
9	80, 420, 100, 9996	16.1200	2.5483	21.3010	2.9096	21.4230	3.6038	20.1380	5.1338
10	100, 660, 200, 39990	25.1450	4.4816	30.0550	6.6525	28.8300	5.1543	26.5470	6.5483

the solution space, instead of searching all feasible paths, assuming that the optimal solution is most likely to be found on this path. Streamline works as a global greedy algorithm that expects to maximize the throughput of a distributed application by assigning the best resources to the most needy stages in terms of computation and communication requirements at each step [2]. The greedy algorithm makes an activity mapping decision at each step only based on the current information without considering the effect of this local decision on the mapping performance at later steps.

We conduct an extensive set of mapping experiments for MET using a large number of simulated workflows for WS-BPEL processes and computer networks. These simulation datasets are generated by randomly varying the parameters of the workflows and computer networks within a suitably selected range of values: (i) the number of activities and the complexity of each activity, (ii) the number of inter-activity communications and the data or control flow between two activities, (iii) the number of nodes and the processing power of each node, and (iv) the number of links and the BW and MLD of each link. The topology and size of 20 simulated computing workflows and computer networks as well as the MET calculated by four mapping algorithms in comparison are tabulated in Table 2, where the problem size is represented by a four-tuple: m activities and $|E_t|$ edges in the workflow, and $|n|$ nodes and $|E_c|$ links in the computer network. For a visual performance comparison, we plot in Fig. 4 the MET performance measurements from these four algorithms for 20 different problem sizes ranging from small to large scales. We observe that rRCP exhibits comparable or superior MET performances over the other three algorithms. Note that the MET measurement points plotted along the x axis (index of problem size) are independent of each other due to the random generation of these 20 problem instances. However, since MET represents the total execution latency from source to destination, a larger problem size with more network nodes and computing activities generally, not absolutely though, incurs a longer mapping path resulting in a longer execution time, as the overall increasing trend indicates.

We also plot the MET performance speedup of rRCP over the other three algorithms in Fig. 5, which is defined as: $Speedup = \left|\frac{MET_{rRCP} - MET_{other}}{MET_{rRCP}}\right|$, where MET_{rRCP} represents the MET for rRCP and MET_{other} denotes MET for each of the other three algorithms in comparison. We observe that rRCP achieves an average performance improvement around 20%-80% in most of the cases and even more than 150% speedups in some cases, which demonstrates the MET performance superiority of the rRCP algorithm.

To further investigate the robustness of these mapping algorithms, for each of 10 problem sizes chosen from the previous 20 cases, we randomly generate 20 problem instances and run four mapping algorithms on them. We then calculate and plot the mean value and standard deviation over 20 instances for each problem size in Table 3 and Fig. 6. We observe that rRCP achieves the best MET performance in an expected sense with the smallest standard deviation, which demonstrates the performance robustness and optimization stability of rRCP in achieving MET in various workflows and computer networks of disparate topologies and different scales.

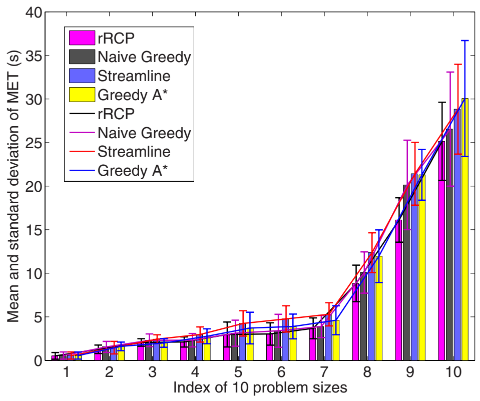

Fig. 6. Mean and standard deviation of MET performance of four algorithms

5.2 Experimental Results

We also conduct experiments on workflow deployment and WS-BPEL process execution in real networks. The experimental settings involve 10 Intel-based Windows machines labeled from 0 to 9, each of which runs ActiveBPEL, an open source WS-BPEL engine [11]. The hardware and software configurations of each computer are provided in Table 4. These computers are connected via a reliable and fast local-area network.

We execute two groups of processes in this experimental network environment: (i) The first group consists of four example services defined in OASIS WS-BPEL 2.0 Standard [19] with slight modification, i.e. Shipping Service, Ordering Service, Loan Approval Service, and Auction Service. These processes involve a relatively small number of activities. (ii) The second group consists of six typical WS-BPEL processes, each of which falls in one of these categories with unique characteristics: computation-intensive, service-invocation-intensive, and the combination of them. For example, the Office Automation and Draining System processes, the Tool Integration and Travel Reserve, and the Online Book Purchase and Train Tickets belong to the first, second and third category, respectively.

Table 4. Specifications of 10 computers used in the experiments

No.	CPU (GHz)	RAM (GB)	OS
0-5	2.5 x 2	1.99	Windows XP
6	1.8 x 2	0.99	Windows XP
7	2.8	1	Windows XP
8	2.8	1	Windows XP
9	2.8	1.5	Windows XP

We deploy and execute the activities of each process to a computer according to the mapping scheme computed by one of four mapping algorithms, and measure the corresponding MET as shown in Table 5. We observe that rRCP algorithm outperforms the other three algorithms in terms of real MET measurements, which is consistent with the simulation results. Qualitatively similar results are obtained from larger-scale processes in the second group. The experimental results based on these two groups of processes illustrate the performance superiority of rRCP algorithm in real network environments. Due to the limit on available physical resources, the problems of large scales as in the simulations are not tested.

We also investigate the performance comparison between distributed BPEL processes using the rRCP mapping scheme and traditional centralized execution (CntrExe) processes. The MET measurements in real networks and their corresponding simulation results are provided in Table 6. We observe that BPEL processes using the rRCP

Table 5. Experiment-based MET performance comparison of BPEL processes

Prb	Problem Size	MET (s)			
Idx	m, $\|E_t\|$, n, $\|E_c\|$	rRCP	Greedy A^*	Streamline	Naive Greedy
1	3, 2, 10, 98	70.03	70.03	83.52	70.03
2	5, 4, 10, 98	72.26	76.58	107.19	78.11
3	5, 5, 10, 98	105.76	113.16	134.52	113.14
4	14, 16, 10, 98	199.06	294.23	361.22	263.82

Table 6. MET comparison between distributed BPEL processes and centralized execution using both experiments and simulations

Prb Idx	Problem Size m, $\|E_t\|$, n, $\|E_c\|$	MET (s) using rRCP		
		BPEL Process (experiments)	CntrExe Process (experiments)	BPEL Process (simulations)
1	3, 2, 10, 98	70.03	106	31.95
2	5, 4, 10, 98	72.26	215	32.71
3	5, 5, 10, 98	105.76	292	71.41
4	14, 16, 10, 98	199.06	355	102.04

mapping scheme achieve 2-3 times MET performance improvements over centralized execution processes. The real MET measurements are generally larger than the simulation results since the latter does not consider network overheads, system dynamics, and the CP is an approximation of MET.

6 Conclusion

We tackled the problem of mapping the workflow of a BPEL process to the computer network to achieve MET and formulated it as a restricted workflow mapping optimization problem. We constructed mathematical models for BPEL processes and computer networks, and proposed rRCP algorithm with mapping restrictions of certain activities. The performance superiority of the rRCP algorithm was justified by both extensive simulation and experimental results. The activities considered in this paper are only synchronous and stateless Web services invocations. We will investigate the invocations of asynchronous and stateful Web services for performance improvement, and more sophisticated performance prediction models to characterize real-time computing node behaviors for more accurate activity execution time estimation. It will be also of our interest to deploy a large network testbed to test large problem sizes in real environments.

Acknowledgment

This work is partially supported by U.S. National Science Foundation under Grant No. CNS-0721980 with University of Memphis and the Defence Pre-Research Project of the “Eleventh Five-Year-Plan” of China under contract No.513060601 with XiDian University.

References

1. Afrati, F.N., Papadimitriou, C.H., Papageorgiou, G.: Scheduling DAGs to minimize time and communication. In: Reif, J.H. (ed.) AWOC 1988. LNCS, vol. 319, pp. 134–138. Springer, Heidelberg (1988)
2. Agarwalla, B., Ahmed, N., Hilley, D., Ramachandran, U.: Streamline: a scheduling heuristic for streaming application on the grid. In: The 13th Multimedia Computing and Networking Conf., San Jose, CA (2006)

3. Bao, L., Chen, P., Zhang, X.: Batch invocation of web services in BPEL process. In: Bouguettaya, A., Krueger, I., Margaria, T. (eds.) ICSOC 2008. LNCS, vol. 5364, pp. 511–516. Springer, Heidelberg (2008)
4. Bashir, A.F., Susarla, V., Vairavan, K.: A statistical study of the performance of a task scheduling algorithm. IEEE Trans. on Computer 32(12), 774–777 (1975)
5. Biskup, J., Carminati, B., Ferrari, E., Muller, F., Wortmann, S.: Towards secure execution orders for composite Web services. In: Proc. of the IEEE International Conference on Web Services, pp. 489–496 (2007)
6. Chafle, G., Chandra, S., Karnik, N., Mann, V., Nanda, M.G.: Improving performance of composite Web services over a wide area network. In: Proc. of the IEEE Congress on Services, pp. 292–299 (2007)
7. Chafle, G., Chandra, S., Mann, V., Nanda, M.G.: Decentralized orchestration of composite Web services. In: Proc. of ACM Int. Conference on World Wide Web (WWW 2004), May 17-22, pp. 134–143. ACM, New York (2004)
8. Chaudhary, V., Aggarwal, J.K.: A generalized scheme for mapping parallel algorithms. IEEE Trans. on Parallele and Distributed Systems 4(3), 328–346 (1993)
9. Chen, L., Agrawal, G.: Resource allocation in a middleware for streaming data. In: Proc. of the 2nd Workshop on Middleware for Grid Computing, Toronto, Canada (October 2004)
10. Chen, S., Bao, L., Chen, P.: OptBPEL: A tool for performance optimization of BPEL process. In: Pautasso, C., Tanter, É. (eds.) SC 2008. LNCS, vol. 4954, pp. 141–148. Springer, Heidelberg (2008)
11. A. Endpoints. Activebpel engine architecture (version 4.1) (2008), `http://www.activebpel.org/docs/architecture.html`
12. Ferrante, J., Ottenstein, K.J., Warren, J.D.: The program dependence graph and its use in optimization. ACM Transactions on Programming Languages and System 9(3), 319–349 (1992)
13. Foggia, P., Sansone, C., Vento, M.: A performance comparison of five algorithms for graph isomorphism. In: Proc. of 3rd IAPR-TC-15 Int. Workshop Graph-based Representations in Pattern Recognition, pp. 188–199 (2001)
14. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-completeness. W.H. Freeman and Company, New York (1979)
15. Gerasoulis, A., Yang, T.: A comparison of clustering heuristics for scheduling DAG's on multiprocessors. J. of Parallel and Distributed Computing 16(4), 276–291 (1992)
16. Guerin, R., Orda, A.: Computing shortest paths for any number of hops. IEEE/ACM Trans. Networking 10(5), 613–620 (2002)
17. Guo, H., Huai, J., Li, H., Deng, T., Li, Y., Du., Z.: ANGEL: optimal configuration for high available service composition. In: Proc. of IEEE Int. Conference on Web Services, July 2007, pp. 280–287 (2007)
18. Hopcroft, J., Wong, J.: Linear time algorithm for isomorphism of planar graphs. In: Proc. of the 6th Annual ACM Symp., Theory of Computing, pp. 172–184 (1974)
19. Jordan, D.: Web services business process execution language version 2.0. OASIS Specification (2007)
20. Kwok, Y.K., Ahmad, I.: Dynamic critical-path scheduling: an effective technique for allocating task graph to multiprocessors. IEEE Trans. on Parallel and Distributed Systems 7(5) (May 1996)
21. Kwok, Y.K., Ahmad, I.: Static scheduling algorithms for allocating directed task graphs to multiprocessors. ACM Computing Surveys 31(4), 406–471 (1999)
22. Li, W., Zhao, Z., Fang, J., Chen, K.: Execution optimization for composite services through multiple engines. In: Krämer, B.J., Lin, K.-J., Narasimhan, P. (eds.) ICSOC 2007. LNCS, vol. 4749, pp. 594–605. Springer, Heidelberg (2007)

23. Luks, E.M.: Isomorphism of graphs of bounded valence can be tested in polynomial time. J. of Computer System Science, 42–65 (1982)
24. Mann, V.: Symphony: decentralized orchestration of composite Web services (2007), `http://domino.research.ibm.com`
25. Nanda, M.G., Karnik, N.: Synchronization analysis for decentralizing composite Web services. In: Proc. of ACM Symposium on Applied Computing (SAC 2003), Melbourne, Florida, USA. ACM, New York (2003)
26. Sekhar, A., Manoj, B.S., Murthy, C.S.R.: A state-space search approach for optimizing reliability and cost of execution in distributed sensor networks. In: Proc. of Int. Workshop on Distributed Computing, pp. 63–74 (2005)
27. Shin, K., Han, S.: Efficient Web services composition and optimization techniques. In: Proc. of IEEE Int. Conference on Web Services, July 2007, pp. 1160–1161 (2007)
28. Shirazi, B., Wang, M., Pathak, G.: Analysis and evaluation of heuristic methods for static scheduling. J. of Parallel and Distributed Computing (10), 222–232 (1990)
29. Wu, Q., Gu, Y.: Supporting distributed application workflows in heterogeneous computing environments. In: Proc. of the 14th IEEE Int. Conf. on Parallel and Distributed Systems, Melbourne, Australia, December 2008, pp. 3–10 (2008)
30. Yildiz, U., Godart, C.: Towards decentralized service orchestrations. In: Proc. of the 2007 ACM Symposium on Applied Computing, pp. 1662–1666 (2007)
31. Zeng, L., Benatallah, B., Ngu, A.H.H., Dumas, M., Kalagnanam, J., Chang, H.: Qos-aware middleware for Web services composition. IEEE Tran. on Software Engineering 30, 311–327 (2004)
32. Zhu, Y., Li, B.: Overlay network with linear capacity constraints. IEEE Trans. on Parallel and Distributed Systems 19, 159–173 (2008)