Critical Path-Based Iterative Heuristic for Workflow Scheduling in Utility and Cloud Computing

Zhicheng Cai[1], Xiaoping Li[1], and Jatinder N.D. Gupta[2]

[1] Computer Science and Engineering, Southeast University, Nanjing, China
[2] College of Business Administration, University of Alabama in Huntsville, Huntsville, USA

Abstract. This paper considers the workflow scheduling problem in utility and cloud computing. It deals with the allocation of tasks to suitable resources so as to minimize total rental cost of all resources while maintaining the precedence constraints on one hand and meeting workflow deadlines on the other. A Mixed Integer programming (MILP) model is developed to solve small-size problem instances. In view of its NP-hard nature, a Critical Path-based Iterative (CPI) heuristic is developed to find approximate solutions to large-size problem instances where the multiple complete critical paths are iteratively constructed by Dynamic Programming according to the service assignments for scheduled activities and the longest (cheapest) services for the unscheduled ones. Each critical path optimization problem is relaxed to a Multi-stage Decision Process (MDP) problem and optimized by the proposed dynamic programming based Pareto method. The results of the scheduled critical path are utilized to construct the next new critical path. The iterative process stops as soon as the total duration of the newly found critical path is no more than the deadline of all tasks in the workflow. Extensive experimental results show that the proposed CPI heuristic outperforms the existing state-of-the-art algorithms on most problem instances. For example, compared with an existing PCP (partial critical path based) algorithm, the proposed CPI heuristic achieves a 20.7% decrease in the average normalized resource renting cost for instances with 1,000 activities.

Keywords: Cloud computing, workflow scheduling, utility computing, critical path, dynamic programming, multi-stage decision process.

1 Introduction

Cloud computing is a new economic-based computational resource provisioning paradigm, in which customers can outsource their computation and storage tasks to Cloud providers and pay only for resources used. At present, only simple pricing models (posted prices) are applied in cloud computing and resources for a task are usually from a single cloud provider. However, there is a trend towards the use of complex pricing models (such as spot-pricing) and a federated architecture (which spans both Cloud and Grid providers) [1]. That is to say, a Utility

S. Basu et al. (Eds.): ICSOC 2013, LNCS 8274, pp. 207–221, 2013.

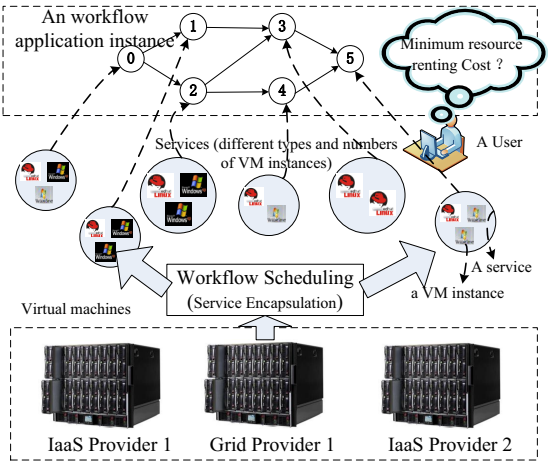

Fig. 1. Architecture of the Utility Computing Environments

Computing based global market (such as Federated Cloud) containing different types of computing and storage resources with different prices is established. For example only in the commercial Amazon Cloud, there are many types of Virtual Machines (VM), each of which provides different levels of services (number of virtual cores, CPU frequency, memory size and I/O bandwidth) with various prices per hour.

Many complex applications such as commercial data analysis, scientific earthquake prediction, weather forecast, are usually modeled as workflow instances and executed on Utility Computing based platforms. Workflows are always denoted by Directed Acyclic Graphs (DAG), in which nodes represent activities and arcs represent precedence relations between activities. Most workflow applications have deadlines. It is desirable to select appropriate services (appropriate type and right number of VM instances) for each activity to get a balance between the task execution time and resource costs. For DAG based task scheduling, there have been many related works such as in homogeneous [2] and heterogeneous distributed systems [3]. They usually try to maximize the utilization of fixed number of resources from the perspective of the service providers. In this paper, we consider the minimization of resource renting cost over unbounded dynamic resources (such as Clouds) for executing a workflow with a given due date from the perspective of customers. For customer-oriented resource renting cost minimization, Byun et al. [4] allocates fixed number of resources to the whole lifespan of the workflow. But in this paper, resources can be acquired at any time and released when they are idle, saving renting cost.

Exact methods, heuristics, and meta-heuristics are commonly used for the DAG-based scheduling problems. Since the workflow scheduling considered in this paper is known to be NP-hard [5], exponential amount of computation time is required for exact algorithms, such as dynamic programming, Branch and Bound, and Benders Decomposition. The three available heuristics for these

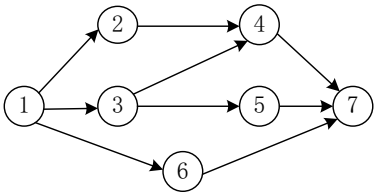

Fig. 2. An Illustrative Example for the Workflow

problems, MDP [6], DET [7], and PCP [8] are single or partial critical-path based. Meta-heuristics, such as [9,10,11] are time-consuming for complex applications (such as workflows with thousands of activities).

DET [7] and PCP [8] schedule activities of a workflow by partitioning them into different types of critical paths and assigning a priority to each critical path. The procedures used to assign services to activities and to schedule (partial) critical paths immensely impact the performance of a scheduling algorithm. Therefore, in this paper, CPI (Critical Path based Iterative) heuristic is developed. CPI heuristic generates complete critical paths as opposed to the partial ones produced by PCP. The proposed CPI heuristic iteratively generates multiple critical paths, which is distinct from the single critical path way adopted in DET. All unscheduled activities are assigned to the longest services. Assignment of all scheduled activities are kept unchanged to generate new complete critical paths. As soon as a new critical path is found, its cost is minimized. In order to simplify the optimization process, the critical path optimization problem is relaxed to the Multi-stage Decision Process (MDP) problem, which can be solved by a dynamic programming algorithm in pseudo-polynomial time.

The rest of the paper is organized as follows. Section 2 describes the workflow scheduling problem in detail and constructs its mathematical model. Section 3 presents the proposed CPI, the complexity analysis and an illustrative example. Experimental results are given in Section 4, followed by conclusions in Section 5.

2 Problem Description

Workflows in Utility and Cloud Computing environments can be depicted by a Directed Acyclic Graph (DAG),$G = \{V, E\}$ where $V = \{V_1, V_2, \ldots, V_N\}$ is the set of activities of the workflow, $E = \{(i, j)|i < j\}$ is the precedence constraints of activities, which indicates that V_j cannot start until V_i completes, P_i and $\pounds_i$ represents the immediate predecessor set and the immediate successor set of V_i, and $path(i, j) = 1$ means that there exists a path from V_i to V_j, otherwise, $path(i, j) = 0$. Figure 2 depicts a workflow example with five activities (V_1 and V_7 are dummy source and sink nodes).

As shown in Figure 1, different types of Virtual Machines (with distinct price per hour) are provided by different IaaS or even Grid providers. For every activity, there are several candidate services, each of which has different types and numbers of VMs with distinct execution times and costs. The candidate services

Table 1. Service Pool of the Illustrative Example

Services	Configurations	Execution time	Cost
S_2^1	1 Small VM	24 hours	\$1.44
S_2^2	1 Medium VM	15 hours	\$1.8
S_2^3	1 Large VM	8 hours	\$1.92
S_2^4	1 Extra Large VM	6 hours	\$2.88
S_3^1	1 Extra Large VM	18 hours	\$8.6
S_3^2	2 Extra Large VM, 1 Large VM	9 hours	\$10.8
S_3^3	4 Extra Large VM	6 hours	\$11.52
S_4^1	1 Large VM	30 hours	\$7.2
S_4^2	1 Extra Large VM	20 hours	\$9.6
S_4^3	1 Extra Large VM, 1 Medium VM	18 hours	\$10.8
S_5^1	1 Small VM	35 hours	\$2.1
S_5^2	2 Small VM	20 hours	\$2.4
S_5^3	4 Small VM	13 hours	\$3.12
S_6^1	1 Medium VM	25 hours	\$3
S_6^2	2 Medium VM	20 hours	\$4.8

for activity V_i form the service pool $S_i = \{S_i^1, S_i^2, \cdots, S_i^{m_i}\}$, $m_i = |S_i|$. And, the service is denoted as $S_i^k = (d_i^k, c_i^k)$, in which d_i^k means the execution time (contains VM set up time and data transfer time) and c_i^k represents the resource renting cost. Shorter activity execution time needs higher resource renting cost. The cost function for the activity execution time may be concave, convex and hybrid. For example, the candidate services for each activity of the workflow in Figure 2 are illustrated in Table 1.

Appropriate services must be selected for each activity to make a balance between execution time and cost. The binary variable $\chi_i^k \in \{0,1\}, 1 \leq i \leq N, 1 \leq k \leq m_i$ is used to demonstrate whether the service S_i^k is chosen for V_i. Each activity is allocated to the most appropriate service to minimize the total resource renting cost, under the constraint of a given deadline D, i.e., the objective of the problem is to find the activity-service mapping for all activities and services to minimize the total cost within deadline D. This problem can be modeled as a mixed integer linear programming (MILP) problem as follows.

$$\text{Min} \quad \sum_{i \in V} \sum_{k=1}^{m_i} c_i^k \chi_i^k \tag{1}$$

$$\text{S.t.} \quad \sum_{k=1}^{m_i} \chi_i^k = 1, 1 \leq i \leq N \tag{2}$$

$$f_i \leq f_j - \sum_{k=1}^{m_i} d_j^k \chi_j^k, \forall (i,j) \in E \tag{3}$$

$$f_1 \geq \sum_{k=1}^{m_1} d_1^k \chi_1^k \tag{4}$$

$$\chi_i^k \in \{0,1\}, 1 \leq k \leq m_i \tag{5}$$

$$d_i^k \in I^+, 1 \leq i \leq N, 1 \leq k \leq m_i \tag{6}$$

$$Max\{f_i\} \leq D, 1 \leq i \leq N \tag{7}$$

The objective function (1) is to minimize the total cost. One and only one service (mode) is assigned to each activity according to constraint (2). Constraint

(3) and (4) guarantee the precedence constraints. Constraint (5) denotes a binary decision variable. The execution times of activities are integers according to constraint (6). The deadline is met according to the constraint (7).

3 Proposed Heuristics

Though critical paths are commonly calculated in DAG-based scheduling methods, only single critical path or partial critical paths are utilized to distinguish critical activities from the activity set. For the grid workflow scheduling considered by Yuan et al. [7], the criticality of non-critical activities is measured by activity floats after the activities on the single critical path have been scheduled. For the workflow scheduling in IaaS Clouds and Utility Grids considered by Abrishami et al. [12,8], partial critical paths are obtained by setting all unscheduled activities with the shortest services. Path structure information is not fully used in these two methods. In this paper, a novel critical path construction method and a new critical path optimizing method are investigated.

3.1 Multiple Complete Critical Path Construction

Let $\mathcal{F}$ and U denote the set of scheduled and unscheduled activities respectively. $\mathcal{Q}_{[j]}$ denotes the index of the selected services for scheduled activity V_j in partial solution $\mathcal{Q}$ and $EFT^L(V_i)$ denotes the earliest finish time of the activity V_i, calculated by assigning all activities in U to the services with the longest execution times while keeping the scheduled activities in $\mathcal{F}$ unchanged. If the activity V_i has been scheduled, i.e., $V_i \in \mathcal{F}$, the execution time of the V_i remains $d_i^{\mathcal{Q}_{[i]}}$, the same as the assigned service in $\mathcal{Q}$. Otherwise, the execution time of V_i is set as $\max_{k=1}^{m_i}\{d_i^k\}$. $EFT^L(V_i)$ for each activity V_i is calculated from V_1 to V_N sequentially according to the above procedure. Initially, the sink activity V_N is chosen as the last node of the critical path (CP) and set as the current activity. The activity with the largest EFT^L among immediate predecessors of the current activity, denoted as V_b, is inserted at the head of CP. V_b is set as the current activity and the activity with the largest EFT^L among its immediate predecessors is denoted as V_b and inserted at the head of CP. The procedure is repeated until V_1 is inserted into CP and the final CP is constructed. The construction process is formally described in Algorithm 1.

Multiple complete critical paths are generated iteratively during the whole optimization process. To ensure that the whole workflow can finish before D, the latest finish time $LFT^S(V_i)$ of V_i is calculated by keeping the scheduled activities unchanged and assigning the unscheduled activities to their shortest execution times, i.e., $\min_{V_j \in \mathcal{L}_i}\{\{LFT^S(V_j) + d_j^{\mathcal{Q}_{[j]}}, V_j \in \mathcal{F}\}, \{LFT^S(V_j) + \min_{k=1}^{m_j}\{d_j^k\}, V_j \in U\}\}$. The solution of the workflow is feasible if and only if the finish time of every V_i is not greater than $LFT^S(V_i)$. Then, the length of CP is $\ell_{CP}^L = \sum_{V_i \in \mathcal{F} \cap CP}\{d_i^{\mathcal{Q}_{[i]}}\} + \sum_{V_i \in U \cap CP}\{\max_{1 \le k \le m_i}\{d_i^k\}\}$. If ℓ_{CP}^L is less than D, the current solution is cheapest because all unscheduled activities take their

Algorithm 1. CP Construction using Longest services

1: $EFT^L(V_1) \leftarrow 0, CP \leftarrow (V_N), V_b \leftarrow V_N$.
2: **for** $j = 2$ to N **do**
3: **if** $V_j \in \mathcal{F}$ **then**
4: $EFT^L(V_j) \leftarrow \max_{V_i \in P_j} \{EFT^L(V_i)\} + d_j^{\mathcal{Q}_{[j]}}$.
5: **else**
6: $EFT^L(V_j) \leftarrow \max_{V_i \in P_j} \{EFT^L(V_i)\} + \max_{k=1}^{m_j} \{d_j^{m_j}\}$.
7: **while** $(V_b != V_1)$ **do**
8: $V_b \leftarrow \arg \max_{V_k \in P_c} \{EFT^L(V_k)\}\}$.
9: Insert V_b at the head of CP.
10: **return** CP.

cheapest services. The whole algorithm terminates. Otherwise, some activities on CP should be assigned to shorter execution times at higher cost services, which may result in new critical paths. In other words, unscheduled activities on CP are reassigned to services in order to minimize the total cost C_{CP} of CP while all activities in V meet their LFT^S constraints, i.e, all unscheduled activities on CP are reassigned to the most appropriate services to minimize C_{CP}. The details of the critical path optimization process is described in Section 3.2 below. The critical path of the workflow would be changed and a new one could be obtained by Algorithm 1 again. The process is iterated until the length of the newly found critical path is not greater than D.

3.2 Critical Path Optimization

The model of the critical path optimization problem is as follows:

$$\text{Min} \quad C_{CP} = \sum_{i \in CP} \sum_{k=1}^{m_i} c_i^k \chi_i^k \tag{8}$$

$$\text{S.t.} \quad \sum_{k=1}^{m_i} \chi_i^k = 1, \forall V_i \in U \tag{9}$$

$$\chi_i^k \in \{0,1\}, \forall V_i \in U, 1 \le k \le m_i \tag{10}$$

$$\chi_i^k = y_i^k, i \in \mathcal{F}, 1 \le k \le m_i \tag{11}$$

Equations (3),(4),(6),(7)

where $y_i^k = 1$ if activity $V_i \in \mathcal{F}$ and 0 otherwise.

The objective (8) is to minimize the total cost C_{CP} of the critical path CP, Constraints (9), and (10) are similar to the constraints (2) and (5) of the original problem respectively. Constraint (11) means that the scheduled activities keep their services assignments unchanged.

In this paper, a Dynamic Programming based Pareto Method (DPPM) is proposed to simplify the critical path optimization problem. For each activity V_i, $EFT^S(V_i)$ is the earliest finish time in terms of the scheduled activities, i.e., execution time of the scheduled activities remain $d_i^{\mathcal{Q}_{[i]}}$. At first, the critical path optimizing problem is relaxed to a Multi-stage Decision Process (MDP) problem

Algorithm 2. Dynamic Programming for MDP_i

Initialize the $PS_1 \leftarrow \{< 0,0,0 >\}, i \leftarrow 1$;
for i=1 to $|CP|$ **do**
  for each $s \in PS_i$ **do**
    for k=1 to $|S_{i+1}|$ **do**
      Generate a solution $s^{'}$ for SSP_{i+1}, $C(s^{'}) \leftarrow C(s) + c_{i+1}^k$;
      $T(s^{'}) \leftarrow T(s) + d_{i+1}^k, s^{'} \leftarrow < T(s^{'}), C(s^{'}), I_{(1,i)}(s), k >$;
      $PS_{i+1} \leftarrow PS_{i+1} \bigcup \{s^{'}\}$;
  Update PS_{i+1}, remove solutions which are dominated by others or the finish time is already greater than D;
return $PS_{|CP|}$;

by temporarily deleting activities and precedence constraints not on the critical path *CP* Then, the MDP problem can be formulated as follows:

$$\text{Min} \quad [\sum_{i \in CP} \sum_{k=1}^{m_i} c_i^k \chi_i^k, \sum_{i \in CP} \sum_{k=1}^{m_i} d_i^k \chi_i^k]^T \tag{12}$$

$$\text{S.t.} \quad \sum_{k=1}^{m_i} \chi_i^k = 1, \forall V_i \in U \cap CP \tag{13}$$

$$f_i \leq f_j - \sum_{1 \leq k \leq m_i} d_j^k \chi_j^k, \forall (i,j) \in CP \tag{14}$$

$$\chi_i^k \in \{0,1\}, \forall V_i \in U \cap CP, 1 \leq k \leq m_i \tag{15}$$

$$\chi_i^k = y_i^k, i \in \mathcal{F} \cap CP, 1 \leq k \leq m_i \tag{16}$$

$$d_i^k \in I, V_i \in CP, 1 \leq k \leq m_i \tag{17}$$

$$Max\{f_i\} \leq D, V_i \in CP \tag{18}$$

Function (12) means that it is a multi-objective optimization problem to find Pareto optimal solutions. Only precedence relations on CP are considered, which are represented in Constraints (14) and (18).

Then, a Dynamic Programming (DP) algorithm is proposed to optimize the MDP problem in pseudo-polynomial time. Sub-problems should be defined before a DP algorithm can be used. In this paper, the i^{th} sub-problem *SSP*$_i$ of the MDP is defined to get a Pareto solution set for partial critical path PCP_i=$\{CP_1, CP_2, CP_3, \ldots, CP_i\}$. Solutions of current sub-problem SSP_i are constructed by combining the Pareto solutions of the immediate former sub-problem SSP_{i-1} and services of the i^{th} activity CP_i. Solutions, which are dominated by others or the solutions with the total execution time larger than the deadline, are removed from the Pareto set. Pareto solutions of the SSP_i can be represented with a set PS_i= $\{< T(s), C(s), I_1(s), I_2(s), I_3(s), \ldots, I_i(s) >, \ldots\}$. Each element of the PS_i is a $(i+2)$-tuple. The first and second elements of the tuple represents the finish time and the total cost of the PCP_i. The element $I_i(s)$ of the tuple represents the index of the selected service for the i^{th} activity of the *SSP*$_i$ in solution s. The $I_1(s), I_2(s), I_3(s), \ldots, I_i(s)$ can be denoted with $I_{(1,i)}(s)$. Since *SSP*$_{i+1}$can be solved based on the solutions of *SSP*$_i$ directly, *SSP*$_1$ to *SSP*$_L$ are solved one by one. At last, a Pareto optimal solution set $PS_{|CP|}$, in which there are at most D solutions, is found. The formal description of this DP procedure is given in Algorithm 2.

Algorithm 3. DPPM(SP_i)

Relax the critical path optimizing problem to MDP;
Call Algorithm 2 to get a Pareto solution set PS for MDP;
Sort solutions of PS by the total cost in non-decreasing order;
for each $s \in PS$ **do**
  for $j = 1$ to N **do**
    if $V_j \in CP_i$ **then**
      $EFT^S(V_j) \leftarrow \max_{i \in P_j}\{EFT^S(V_i)\} + d_j^{s[j]}$;
    else if $V_j \in \mathcal{F}$ **then**
      $EFT^S(V_j) \leftarrow \max_{i \in P_j}\{EFT^S(V_i)\} + d_j^{\mathcal{Q}[j]}$;
    else
      $EFT^S(V_j) \leftarrow \max_{i \in P_j}\{EFT^S(V_i)\} + \min_{k=1}^{m_j}\{d_j^{m_j}\}$
    if $EFT^S(V_j) > LFT^S(V_j)$ **then**
      Goto Step 4;
  $s_{best} \leftarrow s$, break;
return s_{best};

After MDP is solved by Algorithm 2, the cheapest feasible solution should be distinguished from $PS_{|CP|}$. Firstly, solutions of $PS_{|CP|}$ are sorted by the total costs $T(s)$ in non-decreasing order. Then, their feasibility is verified by checking whether $EFT^S(V_i)$ is less than $LFT^S(V_i)$ for all activity $V_i \in V$. Once a feasible solution is found, the feasibility verification process stops. Finally, $LFT^S(V_i)$ are recalculated once a critical path is scheduled. The formal description of the DPPM procedure is shown in Algorithm 3, in which $s[j]$ is the services index of activity V_j in solution s.

3.3 The Proposed CPI Heuristic

The proposed CPI heuristic can be described as follows: Initially, $U \leftarrow V$ and $\mathcal{F}$ are set as empty. $LFT^S(V_i)$ is calculated one by one with all activities being assigned the shortest services. A new critical path CP is generated by assigning all unscheduled activities to the longest services while keeping the assignment of scheduled activities in $\mathcal{F}$ unchanged. If the total duration of the CP is less than the deadline D (i.e., $\ell_{CP}^L \leq D$), the algorithm stops because the partial solution $\mathcal{Q}$ (unscheduled activities select the longest durations) is feasible. Otherwise, CP is optimized by DPPM. $U \leftarrow U/\{V_j \in CP\}$ and $\mathcal{F} \leftarrow \mathcal{F} \cup \{V_j \in CP\}$. s_{cp} is appended to $\mathcal{Q}$. LFT_i is recalculated in terms of $\mathcal{Q}$. The steps of the CPI heuristic are formally described in Algorithm 4.

3.4 An Illustrative Example for CPI

Take the workflow in Figure 2 for example. Set deadline $D = 35$.

(1) *Calculate* $LFT^S(V_7) = 35$, $LFT^S(V_6) = 35$, $LFT^S(V_5) = 35$, $LFT^S(V_4) = 35$, $LFT^S(V_3) = 17$ *and* $LFT^S(V_2) = 17$. *Since no activities has been scheduled in*

Algorithm 4. CPI

1: Set $U \leftarrow V, \mathcal{F} \leftarrow NULL$,$LFT^S(V_N) \leftarrow D$.
2: Assign V_i the service with $\min_{k=1}^{m_i}\{d_i^k\}$,$V_i \in V$.
3: **for** $i = N-1$ to 1 **do**
4: Calculate $LFT^S(V_i)$;
5: **while** $U \neq Null$ **do**
6: $CP \leftarrow$ Call Algorithm 1.
7: **if** $\ell_{CP}^B \leq D$ **then**
8: Set $V_i \in U$ with the longest services,Update $\mathcal{Q}$, Go to step 11.
9: $s_{cp} \leftarrow$DPPM(CP).
10: $U \leftarrow U/\{V_j \in CP\}$, $\mathcal{F} \leftarrow \mathcal{F} \cup \{V_j \in CP\}$. Append s_{cp} to $\mathcal{Q}$. Recalculate $LFT^S(V_i), 1 \leq i \leq N$ in terms of $\mathcal{Q}$.
11: **return** $\mathcal{Q}$.

the first iteration, all activities are allocated to their longest services. Calculate $EFT^L(V_2) = 24, EFT^L(V_3) = 18$, $EFT^L(V_4) = 54$, $EFT^L(V_5) = 53$ *and* $EFT^L(V_6) = 25$ *one by one . Initially,* V_7 *is added to the* CP *first, then, immediate predecessor* V_4 *with largest* $EFT^L(V_4) = 54$ *is added to the front of* CP *and set as the current activity. Later,* V_2 *is added. At last the first critical path* $CP_1 = (V_1, V_2, V_4, V_7)$ *is constructed. Since* $\ell_{CP_1}^L = 54 > D$, CP_1 *should be optimized. In Algorithm 3, a Pareto solution set* $PS_{CP_1} = \{(35, 11.4, S_2^2, S_4^2), (28, 11.52, S_2^3, S_4^2), (33, 12.6, S_2^2, S_4^3), (26, 12.48, S_2^4, S_4^2), (24, 13.68, S_2^4, S_4^3)\}$ *is generated by Algorithm 2. Later, the cheapest feasible solution* $s = (35, 11.4, S_2^2, S_4^2)$ *for the* CP_1 *is distinguished from* PS_{CP_1}. *Update* $\mathcal{Q} = \{\chi_2^2 = 1, \chi_4^2 = 1\}$, $LFT^S(V_7) = 35$, $LFT^S(V_6) = 35$, $LFT^S(V_5) = 35$, $LFT^S(V_4) = 35$, $LFT^S(V_3) = 15$ *and* $LFT^S(V_2) = 15$. $\mathcal{F} = \{V_1, V_2, V_4, V_7\}$, $U = \{V_3, V_5, V_6\}$.

(2) *After* $EFT^L(V_2) = 15$, $EFT^L(V_3) = 18$, $EFT^L(V_4) = 38$, $EFT^L(V_5) = 53$ *and* $EFT^L(V_6) = 25$ *are calculated in terms of the solution of* CP_1, *a new critical path* $CP_2 = (V_1, V_3, V_5, V_7)$ *is generated. Since* $\ell_{CP_2}^L = 53 \geq D$, CP_2 *still needed to be optimized. At first, a Pareto solution set* $PS_{CP_2} = \{(31, 11.72, S_3^1, S_5^3), (29, 13.2, S_3^2, S_5^2), (22, 13.92, S_3^2, S_5^3), (26, 13.92, S_3^3, S_5^2), (19, 14.64, S_3^3, S_5^3)\}$ *for the MDP is generated. Later, each solution of* PS_{CP_2} *is checked to find if* $EFT^S(V_i) \leq LFT^S(V_i), V_i \in V$. *For the first solution* $(31, 11.72, S_3^1, S_5^3)$, $EFT^S(V_2) = 18 > LFT^S(V_2)$. *So, it is infeasible. Considering the second solution* $(29, 13.2, S_3^2, S_5^2)$, $EFT^S(V_2) = 15$, $EFT^S(V_3) = 9$, $EFT^S(V_4) = 35$, $EFT^S(V_5) = 29$, $EFT^S(V_2) = 18$ *is calculated. Since* $EFT^S(V_i) \leq LFT^S(V_i)$, *for all* $V_i \in V$, *the second solution is the cheapest feasible solution for* CP_2.

(3) *Update* $\mathcal{Q} = \{\chi_2^2 = 1, \chi_4^2 = 1, \chi_3^2 = 1, \chi_5^2 = 1\}$, $\mathcal{F} = \{V_1, V_2, V_4, V_3, V_5, V_7\}$, $U = \{V_6\}$. *Then calculate* $EFT^L(V_2) = 15$, $EFT^L(V_3) = 9$, $EFT^L(V_4) = 35$, $EFT^L(V_5) = 29$ *and* $EFT^L(V_6) = 25$. *The* $CP_1 = (V_1, V_2, V_4, V_7)$ *is got again and* $\ell_{CP_1}^L = 35 \leq D$, *which demonstrate that the left unscheduled activities need not to be optimized again, i.e.,* $\chi_6^2 = 1$ *is added to* $\mathcal{Q}$. *The CPI algorithm terminates.*

3.5 Complexity Analysis

Let N is the number of activities and $M = \max_{i=1}^{N}\{m_i\}$. In step 2 of Algorithm 2, there are $|CP| \leq N$ iterations. And in step 3, the Pareto solution set PS_i has at most D Pareto non-dominated solutions. That is because the finish time

of solutions can only be an integer in the interval [1,D] and for each finish time there can be only one Pareto non-dominated solution. For step 4 of Algorithm 2, $|S_{i+1}| \leq M$ services are available. And the complexity of generating a solution is O(N). So the complexity of Algorithm 2 is $O(N^2DM)$. The complexities of the step 3 and 4 of the Algorithm 3 are $O(D^2)$ and $O(DN)$ respectively. So, the complexity of Algorithm 3 is $O(N^2D^2M)$. It now remains to find the number of times step 5 of algorithm 4 is executed. The following theorem helps to do that.

Theorem 1 (Unequal critical path property in CPI). *In the CPI, if* $\ell^L_{CP_i} > D$*, at least one activity in the new generated critical path* CP_i *is unscheduled, i.e.,* CP_i *has never been found in previous steps.*

Proof. In each iteration step $j, j < i$, the critical path $CP_j, j < i$ is scheduled, while the LFT^S is not violated, i.e., ℓ^S of all paths is less than D. Assuming that all activities of CP_i have been scheduled after iteration $k, k \leq i-1$, we can conclude that in each iteration step $j, j > k$, $\ell^S_{CP_i} = \ell^L_{CP_i}$ and $\ell^S_{CP_i} < D$. Therefore,$\ell^L_{CP_i} < D$ is obtained, which is conflict with $\ell^L_{CP_i} > D$. □

In view of the above theorem, there are at most N iterations of step 5 of Algorithm 4. And the complexity of Algorithm 1 is $O(N^2)$. Therefore, the overall complexity of the proposed CPI heuristic is $O(N^3D^2M)$.

4 Computational Results

We now describe the computational tests used to evaluate the effectiveness of the proposed CPI heuristic in finding good quality workflow schedules. To do this, the proposed CPI heuristic is compared with three existing state-of-the-art algorithms (DET heuristic [7], PCP_F heuristic where PCP heuristic with fair policy is used [8], and PCP_D heuristic in which PCP heuristic deploys the decrease cost policy [8]. For comparison purposes, we also included the ILOG CPLEX v12.4 with default settings to solve the MILP model of the workflow problem formulated in Section 2 earlier. All four algorithms (CPI, DET, PCP_F and PCP_D) were coded in Java. Computational experiments for all four algorithms and ILOG CPLEX v12.4 were performed on Core 2 computer with one 3.1GHZ processor, 1G RAM, and Windows XP operating system.

4.1 Test Problems

Since parameters exert influence on the performance of an algorithm, they should be tested on different values. Existing test problem instances used by Abrishami et al. [12] only have paths with at most 9 activities. However, in practice, paths of workflow have much more activities. In our computational experiments, therefore, parameters of the problem instances are as follows:

– the number of activities, N in a workflow takes a value from $\{200, 400, 600, 800, 1000\}$;

- the number of services for each activity i, m_i is generated from a discrete uniform distribution $DU[2, 10]$,$DU[11, 20]$, or $DU[21, 30]$;
- the complexity of the network structure, measured by OS according to [13], takes a value from $\{0.1, 0.2, 0.3\}$;
- the cost function (CF), denoting the functional type of cost to duration, is concave, convex, or hybrid;
- deadlines are generated by $D = D_{min} + (D_{max} - D_{min}) * \theta$ where D_{min} is the minimal total duration (using shortest duration of each activity), D_{max} is the maximal total duration of the workflow, and θ is the Deadline Factor, which takes a value from $\{0.15, 0.3, 0.45, 0.6\}$. This ensure the existence of at least one feasible solution.

The services alternatives for each activity are generated according to [14] and details are as follows: First the number of services, m_i, is generated from DU[2,10] (i.e., discrete uniform distribution with parameters 2 and 10), DU[11,20] or DU[21,30]. Then, the execution time of these services are randomly generated between 3 and 163 as follows: The range [3, 163] is divided into intervals of size 4 and a simple randomized rule is used to decide whether one of the services will have an execution time within that interval. If so, the execution time is generated within the interval using Discrete Uniform distribution. After all the m_i number of execution times are determined, the costs of the services are generated sequentially, starting with that of the minimum-cost services, c_{m_i}, which comes from U[5,105]. Given the execution time, cost pair (d_k, c_k), for services k and d_{k-1} for service $k-1$, c_{k-1} is calculated as $c_k + s_k(d_k - d_{k-1})$, where s_k is the randomly generated slope. For convex cost functions, s_{k-1} is generated from U(s_k,$s_k + S$), where S is the maximum change in slope per service and generated from U(1,2). s_{m_i-1} (the minimum slope) is set to be 0.5. For the concave functions,s_{m_i-1} is randomly generated as $1 + u(m_i - 1)S$, where u is generated from U[0.75,1.25] (so that the initial slope is large enough to allow for smaller slopes for the other services), and then s_{k-1} is generated from U[max(1,(s_k-S)),s_k]. For the hybrid functions, we randomly determine the number of times the slope will increase/decrease compared to the previous one.

The paths connecting various activities are randomly generated, during which the redundant arc avoiding method given by [15] are adopted. The details are shown in Algorithm 5. $path(i, j) = 0$ means that there is no path from V_i to V_j. Step 1 of Algorithm 5 generates the node number of activities with ascending integer numbers. Then, in step 2, random arcs are added to the network one by one where an arc is accepted only if it does not produce redundancy [15].

Using the above test problem generation schemes, for each combination of m_i, OS, and CF, 10 problem instances are generated. Thus, a total of 1,350 problem instances (= 5(N) * 3(m_i)*3(OS)*3(CF)*10) are generated and used in our computational experiments.

4.2 Comparison with Existing Algorithms

To compare the effectiveness of the proposed CPI algorithm with existing algorithms, several measures are used. Let C_b^* be the total cost if all activities of

Algorithm 5. Random Instance Generating Algorithm

Generate activities $V = \{1, 2, \ldots, N\}$;
while $OS_c \leq OS$ **do**
  Generate a non-existed arc $(i, j), i < j$ randomly;
  if $(\forall V_{t_1} \in P_j \forall V_{t_2} \in \pounds_i \forall V_{t_3} \in P_i \forall V_{t_4} \in \pounds_j)(path(i, j) = 0 \wedge path(t_1, i) = 0 \wedge path(j, t_2) = 0 \wedge path(t_3, t_4) = 0)$ **then**
    Accept the arc (i, j), recalculate Order Strength OS_c.
return

a problem instance b select the cheapest services. Let $best_b$ and $worst_b$ be the best and worst solutions among all compared algorithms on instance b. For the convenience of reporting, let W_p be the total number of all problem instances for parameter p which are grouped together (shown in Count column of Table 3). Further, let $C_b(A)$ be the total cost of instance b obtained by algorithm A. Then, the ANC (average normalized resource renting cost), ARDI (Average Relative Deviation Index), and VAR (variance of RDI), are defined as follows:

$$ANC = (\sum_{b=1}^{W_p} C_b(A)/C_b^*)/W_p \tag{19}$$

$$RDI_b = (C_b(A) - best_b)/(worst_b - best_b) \tag{20}$$

$$ARDI = (\sum_{b=1}^{W_p} RDI_b)/W_p \tag{21}$$

$$VAR = (\sum_{b=1}^{W_p} (RDI_b - \sum_{b=1}^{W_p} RDI_b/W_p)^2)/W_p \tag{22}$$

Due to the excessive computational time requirements, CPLEX cannot optimally solve most of the random instances with the above parameters. Therefore, to fairly compare the algorithms, the computation time of CPLEX is set to be identical to that of CPI on the same instance and the best solution obtained within this time is taken as the solution by CPLEX.

Table 2 illustrates that CPI outperforms PCP_F, PCP_D and DET on average normalized renting cost (ANC) and ARDI for all cases. The percentage number in the ANC column for the CPI heuristic in Table 2 shows the decreased percentage of average normalized renting cost, comparing the CPI and PCP_F. CPI gets better performance (lower renting cost) than CPLEX when $N > 400$ or $OS \geq 0.2$. As N increases, Both ANC and ARDI of CPI decease faster than the other heuristics, which implies that CPI is more suitable than the compared heuristics for complexity network structure instances, i.e., great N and big OS. As m_i and OS increase, ANC of all algorithms increases because the problems become more and more complex. ANC of the compared algorithms with concave CF is significantly bigger than those with convex and hybrid CF because the concave CF has fewer cheap service candidates than the other two.

From VAR column of Table 3, it can be observed that the CPI heuristic generates solutions with the lowest VAR for all cases except for $N = 200$. This illustrates that proposed CPI heuristic is more robust than the compared existing algorithms. VAR of CPI decreases as N increases, which means that the robustness of the CPI heuristic increases with the increase in the size of the problem instance. As m_i becomes larger, VAR of each algorithm increases. Among

Table 2. ANC and ARDI(%) of the Random Instances

ParasVals		CPLEX		DET		PCP_F		PCP_D		CPI	
		ANC	ARDI	ANC	ARDI	ANC	ARDI	ANC	ARDI	ANC(Perc en)	ARDI
N	200	5.27	0	12.45	95.6	7.74	33.2	8.15	51.2	7.43(4%↓)	26.2
	400	3.78	1.7	8.53	91.7	4.99	27.9	5.27	48	4.25(14.8%↓)	15.2
	600	5.67	10.6	8.76	87.9	5.02	26.7	5.27	47.1	**4(20.3%↓)**	8.4
	800	7.32	16.7	8.9	86.1	4.88	24.6	5.14	44.4	**3.84(21.5%↓)**	6.7
	1000	5.77	14.8	7.34	85.7	4.16	23.3	4.41	42.7	**3.3(20.7%↓)**	6.1
m_i	[2,10]	1.44	1.6	3.03	99.2	1.75	23.5	1.76	30.6	1.55(11.4%↓)	10.5
	[11,20]	4.4	8	8.04	89.1	4.71	29.6	4.98	54.1	**4.09(13.2%↓)**	15.5
	[21,30]	11.11	14.4	18.07	80.5	10.63	30.5	11.28	58.6	**9.11(14.3%↓)**	15.4
OS	0.1	3.63	2.1	7.29	87.1	4.68	30.8	5.03	55.3	3.95(15.6%↓)	13.2
	0.2	5.69	9	9.67	89.8	5.57	26.3	5.87	46.5	**4.78(14.2%↓)**	13
	0.3	7.65	15.3	10.93	90.8	5.93	23.9	6.17	39.4	**5.05(14.8%↓)**	11.6
CF	convex	2.77	3.6	5.81	97.8	3.65	34.1	3.84	38.1	2.87(21.4%↓)	12.6
	concave	12.4	11.7	20.92	93.9	11.67	27.5	11.95	28.2	**10.22(12.4%↓)**	16
	hybrid	1.31	8	1.73	78.2	1.36	21.4	1.8	75.3	**1.3(4.4%↓)**	12.5
θ	0.15	8.84	14.1	12.2	89.8	9.4	40.8	9.68	56.5	**8.47(9.9%↓)**	23.3
	0.3	6.59	10.5	10.03	89.3	6.37	33.5	6.7	51.5	**5.4(15.2%↓)**	16.4
	0.45	4.24	4.8	8.38	90.3	4.02	23.4	4.33	44.4	**3.28(18.4%↓)**	10.1
	0.6	2.34	1.8	7.25	90.4	2.42	13	2.68	36.2	**2.01(16.9%↓)**	4.9

Table 3. VAR(%) and Time (s) of the Random Instances

Paras	Vals	Count	CPLEX		DET		PCP_F		PCP_D		CPI	
			VAR	Time	VAR	Time	VAR	Time	VAR	Time	VAR	Time
N	200	270	0	12.9	1.9	0.83	2.7	0.31	7.8	0.27	3.62	14.05
	400	270	1.67	20.4	3.65	1.06	3.02	1.65	9.69	1.41	**1.48**	21.18
	600	270	9.26	27.0	4.93	2.28	3.52	4.89	10.72	4.20	**0.73**	27.74
	800	270	13.77	38.5	5.38	4.78	3.31	10.25	10.48	8.93	**0.61**	39.58
	1000	270	12.45	40.9	5.84	6.43	3.32	13.34	10.95	11.82	**0.54**	41.17
m_i	[2,10]	450	1.55	14.3	0.41	2.32	1.55	3.08	3.26	2.88	**1.09**	15.18
	[11,20]	450	7.28	29.3	4.18	3.28	3.46	6.82	10.71	5.81	**2.67**	30.28
	[21,30]	450	12.09	36.2	6.78	2.77	4.66	6.46	11.44	5.59	**2.83**	37.50
OS	0.1	450	2.0	23.6	5.7	2.96	3.7	4.18	10.5	3.72	**1.7**	24.58
	0.2	450	8.0	29.7	4.0	3.14	2.9	5.57	10.0	4.90	**2.0**	30.86
	0.3	450	12.9	27.9	3.8	2.71	2.8	7.85	8.5	6.74	**2.2**	28.89
CF	convex	450	3.48	24.0	1.15	2.84	4.61	4.79	6.55	5.16	**1.96**	25.08
	concave	450	10.26	21.4	3.13	2.22	2.54	4.71	3.19	4.67	**2.71**	22.15
	hybrid	450	7.13	33.3	6.39	3.26	1.82	6.63	7.43	4.25	**1.89**	34.41
θ	0.15	1350	11.8	37.1	4.0	2.48	3.4	4.72	6.9	3.90	**3.7**	37.98
	0.3	1350	9.2	29.5	4.4	2.61	2.8	5.26	7.8	4.52	**2.0**	30.17
	0.45	1350	4.5	22.9	4.2	2.82	1.7	5.64	9.8	5.00	**0.9**	23.61
	0.6	1350	1.8	15.5	4.5	3.19	0.8	5.88	12.4	5.35	**0.3**	17.10

the compared algorithms, as N and m_i increase, VAR of CPLEX increases the fastest and that of CPI increases the slowest. This demonstrates that the stability of CPLEX decreases rapidly as the complexity of the problem increases. VAR of CPLEX and CPI on concave instances is bigger than that on convex and hybrid instances. As the deadline factor θ increases (the deadline becomes looser), the performance of CPLEX, CPI and PCP_F becomes more stable.

Time columns of Table 3 show that CPI and CPLEX consume more computation time than the other algorithms. DET is the fastest algorithm. As N, m_i, and OS increase, more computation time is needed by all algorithms. CF exerts little influence on computation time. Instances with bigger θ consume less computation time for CPI and CPLEX whereas it is reverse situation for PCP and DET. However, computational time to solve a problem by any heuristic is less than one minute, which is reasonable and acceptable in practice.

5 Conclusions

In this paper, services with different time and cost attributes are allocated to workflows in Utility Computing environments by the proposed Critical-Path based Iterative (CPI) heuristic. All activities are grouped and scheduled by iteratively constructed critical paths. In every iteration, a new critical path is generated by keeping the activity-service mapping of the scheduled activities and temporarily assigning the unscheduled activities to the longest services. Dynamic programming based Pareto method is developed for the renting cost minimization of critical paths, in which the workflow is relaxed to a Multi-stage Decision Process (MDP) problem by removing the activities and relations not on the critical path. CPI heuristic is compared with the state-of-the-art algorithms (PCP, DET, and CPLEX) for the considered problem. Experimental results show that the proposed CPI heuristic outperforms the PCP and the DET algorithms for all cases. CPI heuristic is better than CPLEX for most instances and is more stable than CPLEX. Though CPLEX outperforms the CPI heuristic on small size and simple structure problems, the stability of CPLEX is much worse than that of CPI. While computational time required to solve the workflow scheduling problem using the CPI heuristic is more than that required for the PCP and DET algorithms, it is never more than one minute, which is reasonable and acceptable in practice.

In the future, it is worth developing more effective methods for critical path optimization, introducing new decomposition methods for workflows, and investigating the bounds of the problem during the search process.

Acknowledgment. This work was supported in part by the National Natural Science Foundation of China (61003158 and 61272377) in part by the Research Fund for the Doctoral Program of Higher Education of China (20120092110027) and in part by the Southeast University (CXLX12_0099).

References

1. Chard, K., Bubendorfer, K.: High performance resource allocation strategies for computational economies. IEEE Transactions on Parallel and Distributed Systems 24(1), 72–84 (2013)
2. Kwok, Y.K., Ahmad, I.: Static scheduling algorithms for allocating directed task graphs to multiprocessors. ACM Computing Surveys (CSUR) 31(4), 406–471 (1999)
3. Bajaj, R., Agrawal, D.P.: Improving scheduling of tasks in a heterogeneous environment. IEEE Transactions on Parallel and Distributed Systems 15(2), 107–118 (2004)
4. Byun, E.K., Kee, Y.S., Kim, J.S., Deelman, E., Maeng, S.: Bts: Resource capacity estimate for time-targeted science workflows. Journal of Parallel and Distributed Computing 71(6), 848–862 (2011)
5. De, P., Dunne, E., Ghosh, J., Wells, C.: Complexity of the discrete time-cost tradeoff problem for project networks. Operations Research 45(2), 302–306 (1997)
6. Yu, J., Buyya, R., Tham, C.: Cost-based scheduling of scientific workflow applications on utility grids. In: First International Conference on e-Science and Grid Computing, p. 8. IEEE (2005)
7. Yuan, Y., Li, X., Wang, Q., Zhu, X.: Deadline division-based heuristic for cost optimization in workflow scheduling. Information Sciences 179(15), 2562–2575 (2009)
8. Abrishami, S., Naghibzadeh, M., Epema, D.: Cost-driven scheduling of grid workflows using partial critical paths. IEEE Transactions on Parallel and Distributed Systems 23(8), 1400–1414 (2012)
9. Yu, J., Buyya, R.: Scheduling scientific workflow applications with deadline and budget constraints using genetic algorithms. Scientific Programming 14(3), 217–230 (2006)
10. Chen, W.N., Zhang, J.: An ant colony optimization approach to a grid workflow scheduling problem with various qos requirements. IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews 39(1), 29–43 (2009)
11. Wu, Z., Liu, X., Ni, Z., Yuan, D., Yang, Y.: A market-oriented hierarchical scheduling strategy in cloud workflow systems. The Journal of Supercomputing 63(1), 256–293 (2013)
12. Abrishami, S., Naghibzadeh, M., Epema, D.: Deadline-constrained workflow scheduling algorithms for iaas clouds. In: Future Generation Computer Systems (2012)
13. Demeulemeester, E., Vanhoucke, M., Herroelen, W.: Rangen: A random network generator for activity-on-the-node networks. Journal of Scheduling 6(1), 17–38 (2003)
14. Akkan, C., Drexl, A., Kimms, A.: Network decomposition-based benchmark results for the discrete time–cost tradeoff problem. European Journal of Operational Research 165(2), 339–358 (2005)
15. Kolisch, R., Sprecher, A., Drexl, A.: Characterization and generation of a general class of resource-constrained project scheduling problems. Management Science 41(10), 1693–1703 (1995)