PAPER

DISWOP: A Novel Scheduling Algorithm for Data-Intensive Workflow Optimizations

Yuyu YUAN[†,††], ***Nonmember***, **Chuanyi LIU**[†,††a)], ***Member***, **Jie CHENG**[†††], ***and*** **Xiaoliang WANG**[†], ***Nonmembers***

SUMMARY Execution performance is critical for large-scale and data-intensive workflows. This paper proposes DISWOP, a novel scheduling algorithm for data-intensive workflow optimizations; it consists of three main steps: workflow process generation, task & resource mapping, and task clustering. To evaluate the effectiveness and efficiency of DISWOP, a comparison evaluation of different workflows is conducted a prototype workflow platform. The results show that DISWOP can speed up execution performance by about 1.6-2.3 times depending on the task scale.

key words: *workflow optimization, task clustering, process expression, Differential Evolution algorithm*

1. Introduction

In a data-intensive workflow, a process is large-scale, I/O intensive and fine-grained, which leads to several execution challenges. For example, the complexity of the workflow execution is directly dependent on the task scale, so the large-scale workflows have to face the constraints of execution performance. Moreover, a data-intensive workflow is usually described with XML and stored as a Directed Acyclic Graph (DAG), thus the storage alone will be very costly, as well as the communication cost consumed for intermediate data I/O from one computing site to another. Furthermore, since the computation resources are shared and distributed and often managed using queue-based management systems, tasks are usually processed with queuing in a resource site, which leads to a great deal of extra time consumed by queue waiting [2]. Sometimes the queuing waiting time is much more than the execution time. To address these issues, we propose DISWOP: a novel scheduling algorithm for data-intensive workflow optimization.

Up to now, workflow structure transformation has been researched to satisfy three kinds of motivations. Firstly, for the flexibility and parallelism in distributed execution environment [3]–[6]; secondly, for the execution feasibility in resource-constrained execution environments [7], [8]; thirdly, for the workflow structure simplification that is convenient for the process understanding and analysis [9], [10]. Yet these methods are not suitable for data-intensive workflows where large number of tasks have a strong impact on the execution performance. To this end, structure optimization in data-intensive workflow is a new topic. In this field, Pegasus [2], [11] first presented the concept of performance optimization, where tasks are grouped into clusters by level-based or label-based approaches so as to be executed as a single task. But in their approaches, tasks to be clustered need be statically specified in advance, which is not reasonable in most workflow applications, thus they can not automatically determine the clustering granularity.

This paper focuses on workflow optimization for data-intensive structured workflow. The objective is to minimize the whole completion time that comprises not only the execution time but also the queue waiting time as well. The contribution of this paper is that: (1) we propose the concept of workflow process expression in term of the characteristics of structured workflows, which can greatly reduce the storage cost and computing complexity. We also present an algorithm to transform a DAG into a process expression. (2) Based on the workflow process expression, we put forward an approach to process structure optimization. The proposed approach includes two steps: first, map the tasks into available resources based on Differential Evolution algorithm; and simplify the workflow structure by task clustering.

The rest of this paper is organized as follows. In Sect. 2, the problem to be solved is formulated. The concept of process expression and the problem model are introduced here. In Sect. 3, the workflow optimization approach is described in details. Section 4 contains the experiments and the analysis of experimental results. Section 5 concludes the paper and gives some research perspectives.

2. Problem Formulations

2.1 Related Concepts

A ***task*** t in a workflow is a logical computing unit with atomic execution semantic, which is formally defined as $t_{id} = (ca, input, output, type)$, where,

- *id* is the globally unique identifier for the task;
- *ca* denotes the computing scale, which represents the computation cost or complex degree;
- *input* and *output* denote the input and output parameters respectively;

Manuscript received September 16, 2011.
Manuscript revised March 13, 2012.

[†]The authors are with Software School, Beijing University of Posts and Telecommunications, Beijing, China.
[††]The authors are with the Key Laboratory of Trustworthy Distributed Computing and Service (BUPT), Ministry of Education, Beijing, China.
[†††]The author is with the School of Computer Science and Technology, ShanDong University, Jinan, China.
a) E-mail: cy-liu04@mails.tsinghua.edu.cn (Corresponding author)

DOI: 10.1587/transinf.E95.D.1839

- *type* is the task type, which can be a split, a join or a simple task. A split or a join task describes the kind of tasks which have more than one successor or more than one prior tasks respectively; whereas a simple task only has one prior and one successor.

For the simplicity of description, we use notation $a.b$ to express the attribute b of concept a throughout the rest of the paper. For example, *t.input* denotes the input parameters of task t.

A ***workflow process*** is defined as $P = (T, E)$, where,

- $T = \{t_i | i = 1, 2, \ldots, n\}$ is a set of tasks, where n is the number of tasks. It is assumed that a workflow always has one unique starting task t_1, and one unique end task t_n;
- $E = \{e_{i,j} | t_i, t_j \in T\}$, where $e_{i,j} = t_i.output \cap t_j.input$, denotes the relations between task t_i and t_j, which is calculated by the data amount that task t_i transfers to t_j. t_i is called the prior task of t_j, and t_j is the successor task of t_i. Normally, the data amount between two tasks is given when a workflow process is defined.

In this paper, a workflow process is defined according to the following rules:

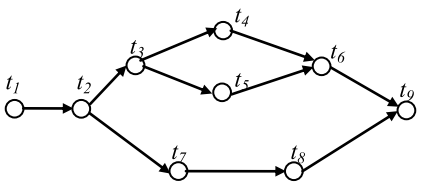
1: A single task is a process.

2: If P_1 and P_2 are processes, then $P_1 \rightarrow P_2$ is a process, where connector $\rightarrow$ denotes sequential relationship.

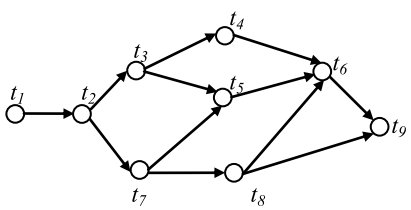
3: If P_1 and P_2 are processes, then $P_1 * P_2$ is a process, where connector $*$ expresses parallel relationship.

According to this recursive definition, a process can be refined by replacing a task with sequential or parallel structure level by level. We call a workflow process generated by this way a structured workflow.

A workflow process is normally represented as a Directed Acyclic Graph (DAG), where vertexes and directed arcs express the tasks and the dependence relations respectively. Figure 1 shows an instance of structured workflows and non-structured workflows.



(a) Structured workflow



(b) Non-structured workflow

Fig. 1 Workflow instances.

Process Expression (PE) of a workflow process P, denoted as $PE(P)$, is the abstract representation of the workflow structure in term of the process definition. For example, the process expression of process P shown as Fig. 1 (a) can be expressed as: $PE(P) = t_1 \rightarrow t_2 \rightarrow (t_3 \rightarrow (t_4 * t_5) \rightarrow t_6 * t_7 \rightarrow t_8) \rightarrow t_9$. It is assumed that connector $\rightarrow$ has higher precedence than $*$. For simplicity, a process expression can be simplified by omitting the sequence connector $\rightarrow$, then $PE(P)$ can be simplified as: $PE(P) = t_1 t_2 (t_3 (t_4 * t_5) t_6 * t_7 t_8) t_9$.

As we can see, by means of process expression, a workflow can be described with a one-dimensional link list or array instead of adjacency matrix or adjacent table, which will greatly reduce the time complexity and space complexity. This is especially meaningful in data-intensive workflow applications.

A ***resource*** r is a computation resource defined as $r_{id} = (ab, S)$, where,

- id is the globally unique identifier for the resource;
- ab is the execution capability of r;
- S is the set of tasks can be executed by r.

Let $R = \{r_i | i = 1, 2, \ldots, m\}$ be the set of available resources, where m is the number of resources, thus the precondition that a process $P = (T, E)$ can be executed by resource R is that $\bigcup_{i=1}^{m} r_i.S = T$.

Let $D = (d_{ij})_{m \times m}$ express the communication distance matrix among the execution resources, in which d_{ij} denotes the communication distance between r_i and r_j. It is assumed that $d_{ij} = d_{ji}$ and $d_{ij} = \infty$ if r_i and r_j are unreachable.

A ***task block*** v is a sub-graph of the original workflow DAG, denoted as $v_{id} = \{t_1, t_2, \ldots, t_k\}$, where k is the number of tasks contained in v. Let V denote the set of blocks. It is assumed that the task blocks satisfy the following polities:

- $\forall t \in T$, $\exists v \in V$, such that $t \in v$
- $\bigcup_{v \in V} v = T$
- $\forall v_i, v_j \in V$ and $i \neq j$, then $v_i \cap v_j = \phi$

Additionally, a task block must be of the structure defined as a structured workflow, and all tasks contained in a block must be executed in a same resource.

Given a workflow process $P(T, E)$, the dependence relationship L between two task blocks v_i and v_j is defined as: $L = \{\langle v_i, v_j \rangle | (\exists t_p \in v_i) \wedge (\exists t_q \in v_j) \wedge (\langle t_p, t_q \rangle \in P.E)\}$. The process P can be simplified as $P = (V, L)$.

To sum up, the relationship among process, task blocks, tasks and resource can be shown as Fig. 2, from which we can see, there is a many-to-many mapping between T and R, as well as V and R.

2.2 Problem Model Description

Based on the concepts mentioned above, we describe the problem model of workflow optimization as follows.

Input: The given process $P(T, E)$, resource set R and communication distance matrix D.

Output: The process $P(V, L)$ and the mapping relation between V and R: $\Psi(V) \rightarrow R$.

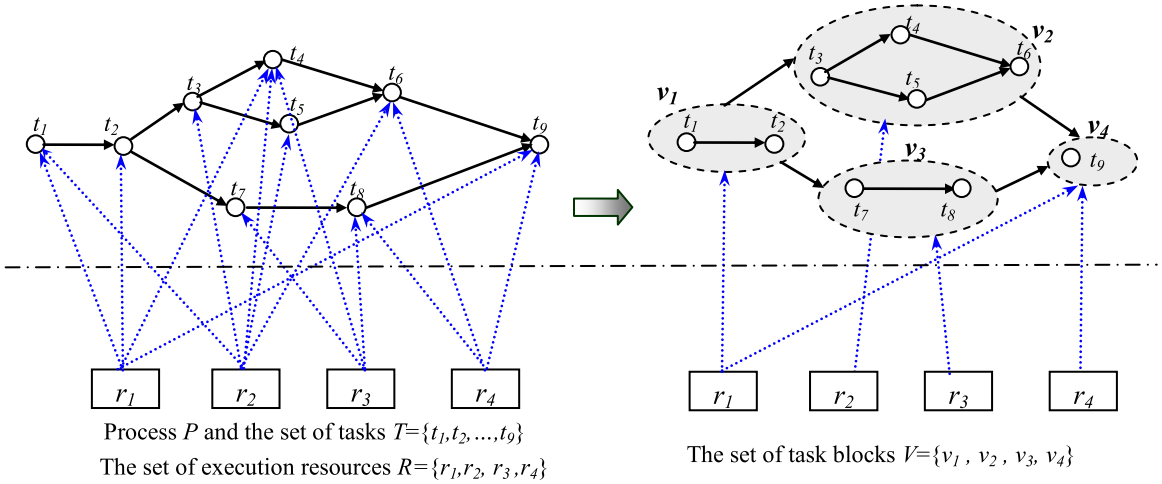


Fig. 2 Relationships among process, blocks, tasks and execution resource. (The directed broken line from resource r_i to task t_j expresses that $t_j \in r_i.S$)

Objective: The minimal completion time of the whole workflow, which comprises queue waiting time, execution time and communication time. In general, as a user of service resources, the overall queue waiting time is mainly related to the number of blocks and the average queue waiting time of a computing unit. Whereas, the execution time and communication time depend on the task planning strategy.

3. DISWOP Algorithm

In this section, we will detail DISWOP algorithm for structured workflow process optimizations, which includes the following steps:

Step1: Generate the process expression for the original workflow, as $PE(P)$;

Step2: Map the tasks into available resources;

Step3: Schedule the process expression and regenerate an optimized workflow process.

3.1 PE Generation

Generation of the process expression is a key step of the whole problem. It is closely related to the follow-up steps. Here, we set a stack and a two-way queue to save the vertexes (tasks) of the DAG of the given process P and the $PE(P)$ respectively. We transforming a DAG into its process expression by such a way that, from the start task to the end one, push a vertex into the stack; if the stack is not empty, pop a vertex t_i from the stack and enter t_i into the queue according to $t_i.type$, and then, push all of the successors of t_i into the stack. In this procedure, if t_i is a join-node ($indegree(t_i) > 1$), its visit frequency $visits(t_i)$ will be recalculated. If $visits(t_i) = indegree(t_i)$, t_i will be pushed into the stack; otherwise, the connector $*$ will be entered into the queue. This procedure will continue until the stack is empty, then $PE(P)$ can be output from the queue. The description of the algorithm mentioned is as Fig. 3.

Algorithm 1: Process expression generation
Input: the DAG of process P
Output: the process expression PE(P)

```
Begin
{Initial the Stack and the two-way Queue;
 Stack.push(t1);
 While (!IsStackEmpty())
       Do
         { Stack.pop(ti);
           If (indegree(ti)≤1 & outdegree(ti)>1)
                 {Queue.Enter(ti);
                  Queue.Enter('(');
                 }EndIf
           Else If (indegree(ti)>1 & outdegree(ti)≤1)
                 {Queue.Enter(')');
                  Queue.Enter(ti);
                 }EndIf
                 Else If (indegree(ti)>1 & outdegree(ti)>1)
                       {Queue.Enter(')');
                        Queue.Enter(ti);
                        Queue.Enter('(');
                       }EndIf
                      Else Queue.Enter(ti);
           For each tj=successor (ti)
               {Initial tj's frequency visits(tj);
                If (indegree(tj)>1)
                   {visits(tj)++;
                    If   (visits(tj)<indegree(tj))
                          Queue.Enter ('*');
                    Else
                          Stack.push(tj);
                         }
                Else Stack.push(tj) ;
               }EndFor
         }EndDo;
 Output PE(P) from Queue in sequence;
}End
```

Fig. 3 Process expression generating algorithm.

3.2 Tasks and Resources Mapping

Task planning here is to search an optimal mapping between tasks and the execution resources, with the objective of minimizing the execution and communication time. In this paper, we employ discrete Differential Evolution (DE) algorithm to address the problem.

(1) Solution Representation

For the resource allocation, we denote the target population, which is a potential solution of the problem, with an n-dimensional vector $X_i = (x_{i,1}, x_{i,2}, \ldots, x_{i,n})$, where, $x_{i,j}$, the j^{th} dimension of the vector, represents a service resource in which task t_j can be executed. Thus the vector represents a sequence of resources. The order of $x_{i,j}$ in X_i should be set as the order of t_j in the process expression $PE(P)$. For each task t_i, its corresponding resource will be chosen in a resource set in which each execution resource r satisfies $t_i \in r.S$.

(2) Fitness Calculation

According to the problem model, the fitness function should take into account the execution and communication cost, which is defined as formula (1):

$$Fitness(X_i) = \alpha \cdot \sum_{j=1}^{n} \frac{t_j.ca}{x_{i,j}.ab} + (1-\alpha) \cdot \sum_{e_{i,j} \in E} \frac{e_{i,j}}{d_{x_{i,i}, x_{i,j}}} \quad (1)$$

where, $\alpha (0 \le \alpha \le 1)$ is the weight of execution cost. The first term of the formula means the execution time which tasks are executed on resources. The second term means the communication time between two tasks. So the fitness function expresses the total time which includes the execution time and the communication time of the workflow.

(3) Mutation and Crossover Operations

Assumed that $X_i^k = (x_{i,1}^k, x_{i,2}^k, \ldots, x_{i.n}^k)$ represents an individual of the kth generation, in which the best solution is denoted as $G^k = (g_1^k, g_2^k, \ldots, g_n^k)$. The main idea of DE algorithm is that each individual X_i^k will compete with a trial one $U_i^k = (u_{i,1}^k, u_{i,2}^k, \ldots, u_{i,n}^k)$ by comparing their fitness function value to determine who can survive for the next generation. According to the DE/rand/1/bin schemes of Storn [12], the trial vector U_i^k is generated as follows:

$$u_{i,j}^{k-1} = \begin{cases} x_{a,j}^{k-1} + F \cdot (x_{b,j}^{k-1} - x_{c,j}^{k-1}), & \text{if } r < CR \text{ or } j = D(j) \\ x_{i,j}^{k-1}, & \text{otherwise} \end{cases} \quad (2)$$

where X_a^{k-1}, X_b^{k-1} and X_c^{k-1} are three different individuals which are randomly chosen from the $(k-1)$th generation population; F is the mutation factor which will affect the differential variation between the two individuals; r is an uniform random number between 0 to 1, CR is a user-specified crossover constant in the range [0, 1), and $D(j)$ is a randomly chosen integer in range $\{1, 2, \ldots, n\}$ to ensure that the trial vector U_i^{k-1} differs from X_i^{k-1} by at least one parameter [13].

Algorithm2: Task planning
Input: the given process P, resource set R, Matrix D and the set of initial population $\{X_1^1, X_2^1, ..., X_M^1\}$
Output: the optimal allocation solution.
Begin
{Initialize parameters for DE Algorithm;
Initialize iteration number k=1, the maximum iteration number is K;
Calculate the Fitness of each individual of the initial population and find the global best G^1;
While (k<K) Do
{For each individual X_i^k
{Find two different members X_a^k, X_b^k ($a \neq b$);
$U_i^{k+1} = F_1(c_1, X_i^k, F_2(c_2, G^k, F_3(t, X_a^k, X_b^k)))$;
If (Fitness(U_i^{k+1})>Fitness(X_i^k))
{$X_i^{k+1} = U_i^{k+1}$;
Update G^{k+1};
}
Else
$X_i^{k+1} = X_i^k$;
} Endfor
k=k+1;
} Enddo
Output the global best G^k as the optimal task planning;
}End

Fig. 4 Resource allocation algorithm.

Finally, X_j^k will be determined as formula (3).

$$X_j^k = \begin{cases} U_j^{k-1}, & \text{If Fitness}(U_j^{k-1}) < \text{Fitness}(X_j^{k-1}) \\ X_j^{k-1}, & \text{otherwise} \end{cases} \quad (3)$$

Obviously, the scheme above can not be directly applied in the discrete optimization problems. Instead, referencing [14], we propose a discrete DE algorithm for the resource allocation. The mutation and crossover operation of the algorithm are defined as follows.

$$U_i^k = F_1(c_1, X_i^{k-1}, F_2(c_2, G^{k-1}, F_3(t, X_a^{k-1}, X_b^{k-1}))) \quad (4)$$

Equation (4) consists of three components. The first one is $P^k = F_3(t, X_a^{k-1}, X_b^{k-1})$, in which, X_a^{k-1}, X_b^{k-1} are two different individuals ($a \neq b \neq i$) randomly chosen from the $(k-1)$th generation population; t is the mutation strength; and F_3 is the crossover operator which is conducted in such way that, generate a uniform random start number e between 1 and n, and then determine the $P^k = (p_1^k, p_2^k, \ldots, p_n^k)$ as follows.

$$p_i^k = \begin{cases} x_{a,i}^k, & \text{if } e \le i < e + t \\ x_{b,i}^k, & \text{otherwise} \end{cases} \quad (5)$$

The second component is $R^k = F_2(c_2, G^{k-1}, P^k)$, where G^{k-1} is the best individual of the $(k-1)$th generation population; c_2 is the mutation probability, and F_2 is the mutation operator which is employed to accept information from

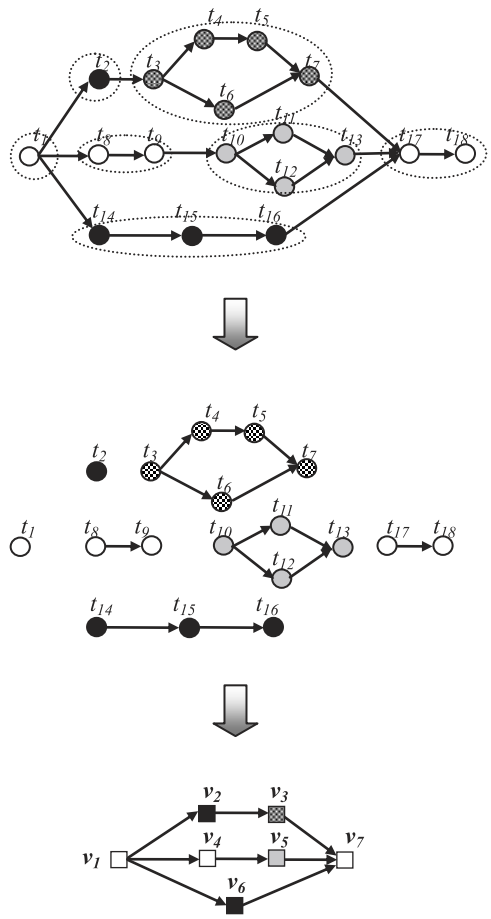


Fig. 5 Process clustering. (Circles denote tasks, and squares with different colors denote the blocks of different resources)

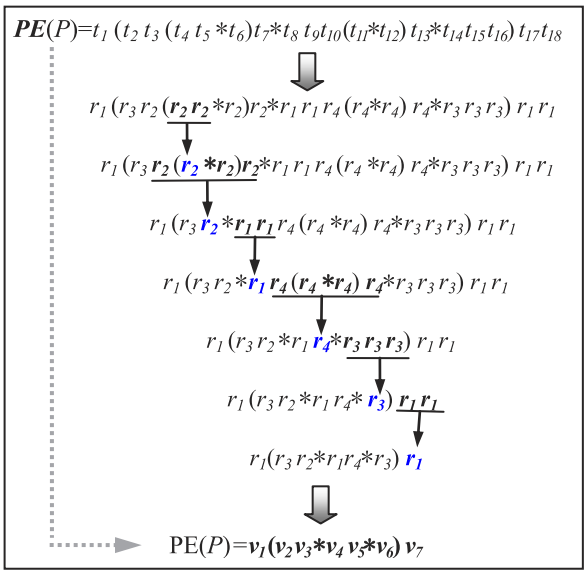


Fig. 6 Task clustering.

Table 1 Task blocks and their corresponding resources.

$PE(v_i)$, $v_i \in V$	$\Psi(v_i)$
$PE(v_1)=t_1$	$\Psi(v_1)=r_1$
$PE(v2)=t_2$	$\Psi(v_2)=r_3$
$PE(v3)=t_3(t_4\,t_5*t_6)t_7$	$\Psi(v_3)=r_4$
$PE(v4)=t_8\,t_9$	$\Psi(v_4)=r_1$
$PE(v5)=t_{10}\,(t_{11}*t_{12})\,t_{13}$	$\Psi(v_5)=r_4$
$PE(v_6)=t_{14}\,t_{15}\,t_{16}$	$\Psi(v_6)=r_2$
$PE(v_7)=t_{17}t_{18}$	$\Psi(v_7)=r_1$

the global best to the temporary member P^k. That is, if a uniform random number r generated between $(0, 1)$ is less than c_2, a start number $f(1 \le f \le n)$ and a length number $e(1 \le e \le \text{differ}(G^{k-1}, P^k)$, $\text{differ}(G^{k-1}, P^k)$ represents the difference between G^{k-1} and P^k) will be randomly selected, and then $p_e^k, p_{e+1}^k, \ldots, p_{e+f-1}^k$ will be replaced with $g_e^{k-1}, g_{e+1}^{k-1}, \ldots, g_{e+f-1}^{k-1}$.

The third component is $U_i^k = F_1(c_1, X_i^{k-1}, R^k)$, where c_1 is the choice probability, and F_1 is the selection operator which is applied to determine the generation of the trial individual. If a uniform random number r generated between $(0, 1)$ is less than c_1, there will be $U_i^k = R^k$, else $U_i^k = X_i^{k-1}$.

Finally, the selection is based on the comparison of the fitness between U_i^k and X_i^{k-1} such that:

$$X_i^k = \begin{cases} U_i^k, & \text{if Fitness}(U_i^k) < \text{Fitness}(X_i^{k-1}) \\ X_i^{k-1}, & \text{otherwise} \end{cases} \quad (6)$$

(4) Algorithm Description

To sum up, the pseudo code of the task planning algorithm is described as Fig. 4.

3.3 Task Clustering

Given a process P, according to the algorithm mentioned above, the optimal solution $G = (g_1, g_2, \ldots, g_n)$ gives the mapping relationship between tasks and resources, that is, $\Psi(t_i) = g_i$. Then the set of blocks can be determined by following way:

Step1: Substitute t_i with $\Psi(t_i)$ in $PE(P)$

Step2: Scan $PE(P)$ from left to right, if there exists such kinds of characteristic strings: "$r_i r_i$", "$(r_i * r_i)$", then cluster it into r_i and record the position of the clustering. This procedure continues until no characteristic string appears.

Step3: Substitute the r_i with a cluster and generate the set of blocks.

For example, in Fig. 5, the process can be partitioned into 7 task blocks denoted as v_1, v_2, v_3, v_4, v_5, v_6 and v_7. The clustering procedure is shown as Fig. 6. The structure of the task blocks and their corresponding resources are illustrated as Table 1. After the task clustering, the structure of the process is simplified.

4. Experiments

To evaluate the performance of the approach proposed above, we developed a tool to generate random structured DAG and the experimental environments. After generating workflow cases, we brought them into our workflow platform [15]. As to the values of DE parameters, the population size M was 30, the mutation strength t is set to 3. c_1 and c_2 are taken as 0.75. The maximum generation number K is set to 50 and the DE algorithm is terminated when the best solution is not improved in 10 consecutive generations.

4.1 Experimental Environment

(1) Structured Workflow Simulation

According to the process definition mentioned in Sect. 2, we generate the structured workflows with different scale by such a way that, start from a single-task structure process, randomly select the nesting order and nesting depth of sequence and parallel structures. Two parameters are related to the subsequent experiments.

- Process scale n, that is, the number of tasks in a process.
- The calculation amount difference factor ε. For each task t, $t.ca$ will be selected from $[a, \varepsilon a]$, where $\varepsilon \geq 1$, and a is the minimum value of $t.ca$.

(2) Computation Resources Simulation

The execution environment is mainly based on the following parameters:

- The number of available resources m.
- The executing capability difference factor ρ. For each resource r, $r.ab$ will be randomly selected from $[b, \rho b]$, where $\rho \geq 1$, and b is the minimum value of $r.ab$. Besides, $r.S$ is randomly selected and must be assured that $(r.S \neq \phi) \wedge \left(\bigcup_{i=1}^{m} r_i.S = P.T\right)$.
- The communication capability difference factor τ, which is applied to reflect the communication capability among different execution resources. All the communication distance will be selected from $[c, \tau c]$, among which $\tau \geq 1$, and c is the minimum value.

In the experiments, we simulated different execution environments by changing the value of the parameters above.

(3) For the convenience of analysis, we also introduce other two parameters:

- Number of task blocks w, which can be used to reflect the average granularity of task clustering.
- Resource distribution standard deviation s, which is applied to reflect the distribution of the tasks among the resources.

$$s = \left(\sum_{i}^{w} (\pi(v_i) - \bar{\pi})\right)^{1/2} \tag{7}$$

where $\pi(v_i)$ denotes the number tasks in block v_i and $\bar{\pi}$ is the average number of tasks contained in the resources.

Table 2 Experimental results with different communication capability among the resources.

τ	w	s
1	75.63	14.55
2	74.49	15.03
4	73.23	15.94
8	68.40	16.52
16	66.18	17.29

Table 3 Experimental results with different executing capability of resources.

ρ	w	s
1	79.89	7.58
2	72.63	13.53
4	72.18	13.46
8	66.03	15.91
16	62.49	16.97

4.2 Experimental Results and Analysis

In the first experiment, we test the impact of the execution environment on the structure optimization, which includes three phases as follows.

(1) Testing the impact of different communication capability between the resources

The workflow case was generated with the task scale $n = 300$. Other environmental parameters were set as: $m = 10$, $\varepsilon = 3$, $\rho = 4$ and $\tau = 1, 2, 4, 8, 16$. With the same workflow case, the experiments have been done 8 times for each value of τ, and the average experimental results are shown as Table 2.

(2) Testing the influence of resource computation capabilities

The task scale n is set to 300, and the other parameters are set as: $m = 10$; $\varepsilon = 3$; $\tau = 5$; and $\rho = 1, 2, 4, 8, 16$. The experiments have been done with the same process case, and the average experimental results are shown as Table 3.

(3) Testing the influence of different task calculation amount

The process scale n is set to 300, the other parameters are set as: $m = 10$; $\tau = 3$; $\rho = 5$; and $\varepsilon = 2, 4, 6, 8, 10$. The experiments have been done with the same process case, and the average experimental results are shown as Table 4.

The experimental results show that:

A. The communication capability difference has little influence on the process optimization. In general, the resource site, which has better communication capability between its prior, will has more possibility to be selected. For example, if the current resource is able to support the next task, then the most suitable successor should be itself. But, since an execution resource may not support all the tasks,

Table 4 The influence of different task calculation amount on the experimental results.

ε	w	s
2	68.43	15.47
4	76.29	14.13
6	71.34	13.74
8	73.29	11.12
10	72.75	13.61

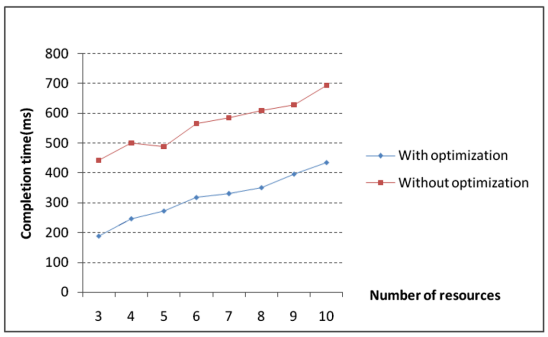


Fig. 7 Comparison of the execution performance with different number of resources.

the resource which has the best communication capability may not be selected because it can not support the next task, and that is the reason that difference of communication capability shows unobvious effect on the task clustering.

B. With the increasing of execution capability difference, the resource with more capability will have more chance to be selected. So when the execution overhead is reduced, the number of block increases, thus the clustering granularity decreases. We can also find that the recourse standard deviation is increasing, which reveals that the distribution of executers is more unbalanced.

C. The computing scale difference has little influence on the process simplifying. This is because, for a given workflow process, the task planning mainly depends on the attributes of resources such as *r.S* or *r.ab*. It is little dependent of the attribute of the workflow itself.

In the second experiment, we test the performance of the proposed approach by comparing the workflow execution of two situations: with optimization (using DISWOP) and without optimization. In the method without optimization, the mapping between tasks and resources was randomly selected as long as it satisfied the condition that the task can be executed at the corresponding resource.

First, we evaluate the performance of DISWOP by comparing the workflow execution with different number of resources. The workflow case was generated with the process scale $n = 40$. The number of resources m was setting from 3 to 10. The queue waiting time of each computing unit (a task or a block) was simulated as 10 ms per interval. Other parameters are set randomly. Each experiment was run 8 times. The average result obtained is shown as Fig. 7,

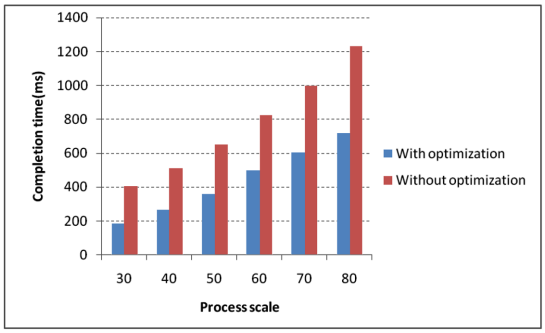


Fig. 8 Comparison of the execution performance with different process scale.

from which it is evident that there is an obvious advantage with structure optimization compared to without optimization.

Second, we test the execution performance with different process scale. The workflow cases were generated with the process scale $n = 30$ to 80. The number of resource was set to $m = 5$. Same to the first experiment, we simulate the queue waiting time of each computing unit as 10 ms per interval. The test result is shown as Fig. 8, from which we also find that the performance with structure optimization is much superior to that without optimization.

5. Conclusions and Future Works

Due to the increasing scale of data-intensive workflows, structure optimization is deemed as an effective step before execution. In this paper, we present a structure optimization approach, with the objective of minimizing the overall completion time which comprises not only the execution time and communication time but also the queue wait time. This paper also provides a simple method to store workflow definitions, instead of using DAG, we present the concept of process expression which will describe a workflow with a linear complexity. This is quiet meaningful in data-intensive workflows.

The proposed approach in this paper is based on structured workflows, which have been required in the applications that a process is defined from a simple structure to a complicated one with the function refinement dynamically. Addition to the structure of the process expression, convexity of task clustering is another advantage of structured workflow, which ensures that there will be no dependence cycle among the task blocks.

As we can see, a non-structured workflow can not be described as a process expression. So in the near future, we will introduce a non-structured workflow optimization approach and study the convexity of the structure optimization. Furthermore, we will apply the workflows optimization into scientific workflow applications to verify the effectiveness of our approaches.

Acknowledgements

This work is supported by the China NSFC Program under Grant No.91118002 and No.60273006. This work is also supported by the China 863 High Technology Program under Grant No.2011AA01A204.

References

[1] Y. Feng, W. Cai, and J. Cao, "Dynamic partner identification in mobile agent-based distributed job workflow execution," J. Parallel and Distribute Computing, vol.67, no.11, pp.1137–1154, Nov. 2007.
[2] E. Deelman, "Grids and clouds: Making workflow applications work in heterogeneous distributed environments," International Journal of High Performance Computing Applications, vol.24, no.3, pp.284–298, Aug. 2010.
[3] W. Tan and Y.S. Fan, "Dynamic workflow model fragmentation for distributed execution," Computers in Industry, vol.58, no.5, pp.381–391, June 2007.
[4] M.G. Nanda, S. Chandra, and V. Sarkar, "Decentralizing execution of composite Web services," Proc. 19th annual ACM SIGPLAN Conf. on Object Oriented Programming, Systems, Languages, and Applications, vol.39, no.10, pp.170–187, Oct. 2004.
[5] B.X. Liu, Y.F. Wang, Y. Jia, and Q.Y. Wu, "A role-based approach for decentralized dynamic service composition," J. Software, vol.16, no.11, pp.1859–1867, Nov. 2005.
[6] S.H. Bokhari, "Partitioning problems in parallel, pipelined, and distributed computing," IEEE Trans. Comput., vol.37, no.1, pp.48–57, Jan. 1988.
[7] M. Andrea and M. Stefano, "Partitioning rules for orchestrating mobile information systems," Personal and Ubiquitous Computing, UK: Springer-Verlag London, vol.9, no.5, pp.291–300, Aug. 2005.
[8] A. Neyem, D. Franco, S.F. Ochoa, and J.A. Pino, "Supporting mobile workflow with active entities," Proc. 11th International Conf. on Computer Supported Cooperative Work in Design, pp.795–800, April 2007.
[9] Y. Choi and J.L. Zhao, "Decomposition-based verification of cyclic workflow," Lect. Notes Comput. Sci., vol.3707, pp.84–98, 2005.
[10] J.Q. Li and Y.S. Fan, "Timing boundedness verification and analysis of workflow model," Computer Integrated Manufacturing Systems, vol.8, no.10, pp.770–775, Oct. 2002.
[11] G. Singh, C. Kesselman, and E. Deelman, "Optimizing grid-based workflow execution," J. Grid Computing, vol.3, no.3-4, pp.201–219, March 2006.
[12] R. Storn and K. Price, "Differential evolution – a simple and efficient adaptive scheme for global optimization over continuous spaces," Technical Report TR-95-012, ICSI, March 1995.
[13] O. Beaumont, V. Boudet, and Y. Robert, "The iso-level scheduling heuristic for heterogeneous processors," Proc. 10th Euromicro Workshop on Parallel, Distributed and Network-based Processing, pp.335–342, Aug. 2002.
[14] M.F. Tasgetiren, Q.K. Pan, Y.C. Liang, and P.N. Suganthan, "A discrete differential evolution algorithm for the total earliness and tardiness penalties with a common due date on a single-machine," Proc. IEEE Symposium on Computational Intelligence in Scheduling, pp.271–278, April 2007.
[15] X.G. Wu and G.Z. Zeng, "Goals description and application in migrating workflow system," Expert Systems with Applications, vol.37, no.12, pp.8027–8035, Dec. 2010.

Yuyu Yuan is a professor of computer science & engineering at Beijing University of Posts and Telecommunications. She has research interests in computer science, software testing & evaluation, cloud computing. Prof. Yuan is now the deputy director of Key Laboratory of Trusted Distributed Computing and Service, Ministry of Education.

Chuanyi Liu received his Ph.D. (2009) in computer science and technology from Tsinghua University, China. He is now an assistant professor of computer science & engineering at Beijing University of Posts and Telecommunications. He has broad research interests in computer systems, including architecture, file and storage systems, information security and data protection, he is now focus on cloud computing and cloud security. Dr. Liu spent one year as a visiting scholar at the Digital Technology Center of the University of Minnesota.

Jie Cheng is an associate professor of computer science & technology at Shandong University. Her research field includes scientific computing, data-intensive computing and data workflows.

Xiaoliang Wang is a master candidate in computer science and technology school of Beijing University of Posts and Telecommunications. He is now working in the field of cloud security.