Resource Allocation vs. Business Process Improvement: How They Impact on Each Other

Jiajie Xu, Chengfei Liu, and Xiaohui Zhao

Centre for Information Technology Research
Faculty of Information and Communication Technologies
Swinburne University of Technology
Melbourne, Australia
{jxu, cliu, xzhao}@groupwise.swin.edu.au

Abstract. Resource management has been recognised as an important topic for the execution of business processes since long time ago. Yet, most exiting works on resource allocation have not paid enough attentions to process characteristics, such as structural and task dependencies. Furthermore, no effort has been made on optimising resource allocation by improving business processes. To address this issue, we propose an approach that optimises the use of resources in an enterprise by exploring the structural features of a business process and adapting the structures of the business process to better fit the resources available in the enterprise. After a motivating example, we describe a role-based business process model for resource allocation. Then we present strategies for resource allocation optimisation and discuss the relationship between resource allocation and business process improvement. A set of heuristic rules are discussed and algorithms based on these rules are designed for optimising resource allocation with a particular optimisation goal.

1 Introduction

Business Process Management (BPM) is aimed to investigate how to help enterprises improve their business processes, and thereby enable enterprises to achieve their business goals with lower cost, shorter time and better quality. Nowadays business process management systems [10] have been widely used in many business scenarios. Because the execution of business processes depends on the available resources, the performance of a business process is subject to the degree of match between the given resources and the structure of the business process. When the structure of a business process is fixed, the business process performance, in terms of cost and time, may vary greatly with different resource allocation plans. To this end, several works have addressed the impact of business process structures on resource management [2, 4, 7].

We reckon that resource allocation and business process impact on each other. Structures of business process set a constraint on how resources are allocated to tasks due to the dependency. However, it is possible that a business process is not well-defined, and as a result the resources may not be utilised optimally to reach certain

business goal. It is desired that the structure of a business process is improved so that resources can be utilised in a more optimal way. However, as far as we know, no work has discussed this kind of improvement. In this paper, we collectively discuss the problems of resource allocation optimisation for business processes, and resource oriented business process improvement.

To incorporate the resource allocation into business process improvement, this paper proposes a role-based business process model to specify the relationship between business processes, tasks, roles and resources. Based on this model, a comprehensive framework is established to pre-analyse and optimise the resource allocation and business process improvement, and thereby adapt the two to the best match. The contribution of this paper to current business process improvement and resource allocation lies in the following aspects:

- Enable the pre-analysis on resource allocation and utilisation before the execution of business processes, and therefore be able to check if some resource allocation requirements can be satisfied;
- Enable the business process structure change to better optimising resource allocation;
- Develop algorithms for allocating resources to a business process with a particular optimisation criterion for achieving minimal cost with a certain time constraint.

The remainder of this paper is organised as follows: Section 2 discusses our problem for collectively optimising resource allocation and improving business processes with a motivating example; Section 3 introduces a role based business process model, which defines the related notions for resource allocation, and the relationship among these notions; Two algorithms for resource allocation optimisation and resource oriented business process improvement are presented in Section 4; Section 5 reviews the work related to our approach, and discusses the advantages of our approach; Lastly, Concluding remarks are given in Section 6.

2 Motivating Example

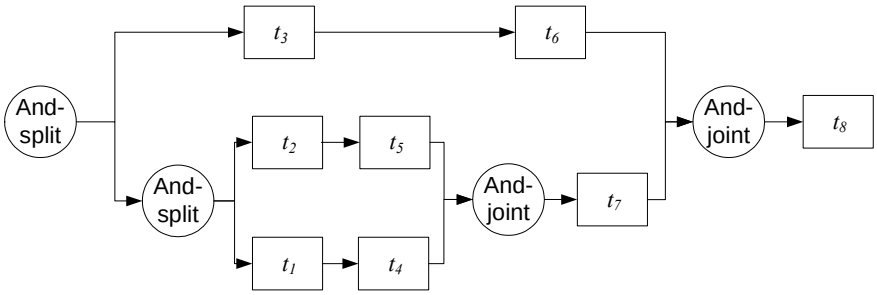

Fig. 1. Business process structure

We use an example to illustrate the problem that we are tackling in this paper. Figure 1 shows a business process with eight tasks and four gateways. Assume that the set of

resources used for this business process are given in Table 1 and are classified according to roles. The cost for each role is also shown in the table. A role describes the capability of its resources for performing certain tasks. For each task, the roles that are capable of performing it are shown is Table 2. The time for a role to perform a task is also indicated in the table. For example, Resources s31 and s32 can perform role r3 at the cost of $40 per hour. Task t_4 can be performed by resources of role *r1* and role *r4* in 2 hours and 2.5 hours, respectively.

Table 1. Resource classification

Role	Cost	Resource
r_1	$50/hr	s_{11}, s_{12}
r_2	$25/hr	s_2
r_3	$40/hr	s_{31}, s_{32}
r_4	$20/hr	s_4
r_5	$25/hr	s_5

Table 2. Capabilities of roles

Task / Role	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8
r_1	2hr			2hr			1hr	
r_2	3hr	1.5hr						
r_3		1hr			2hr			2hr
r_4			2hr	2.5hr				
r_5					3hr	2hr		3hr

Time and cost are two criteria to evaluate the performance of business process. Assume resources are allocated as Figure 2(a). The time required is 7 hours, and meanwhile the expense is $537.5. In the situation of allocation as Figure 2(b), the cost is reduced to $465, while the execution time is increased to 9.5 hours. In reality, an enterprise always has a time constraint on a production business process such that the processing time is no more than a deadline. Therefore, the resource allocation is considered to be optimised when the expense is low but the time constraint can be satisfied. In this example, we assume the deadline is 7.5 hours. An optimised resource allocation for this scenario is shown in Figure 2(c) where the expense is 487.5$ and time is 7.5 hours which just satisfies the time constraint. Compared with Figure 2(c), the allocation in Figure 2(a) is worse because it is more expensive, even though both of them can satisfy time constraint; the allocation in Figure 2(b) is less expensive, however, it violates the time constraint and hence not usable. Therefore, in order to improve the performance of this business process, resources are expected to be allocated as Figure 2(c) under the time constraint.

However, sometimes the structure of business process may prevent resources from being allocated in the optimised way. For instance, if the time constraint is 11.5 hours, Figure 2(b) is the optimised allocation pattern under the business process structure. However, because the limit of time is rather long, t_1 and t_2 can be done in sequential order rather than parallel order. In other words, the business process can be changed to a new business process as shown in Figure 3. If we choose to allocate resources as shown in Figure 2(d), we can achieve an expense of $457.5, which is less than that in

Figure 2(b), and the time is 11.5 hours thereby satisfy the time constraint. Therefore, based on the time requirement and available resources, business process redesign may contribute to improve the performance of business process through enabling resource to be allocated in a more optimised way.

Task	Resource
t_1	s_{12}
t_2	s_2
t_3	s_4
t_4	s_{12}
t_5	s_{31}
t_6	s_5
t_7	s_{11}
t_8	s_{31}

(a)

Task	Resource
t_1	s_2
t_2	s_{31}
t_3	s_4
t_4	s_4
t_5	s_{32}
t_6	s_5
t_7	s_{11}
t_8	s_5

(b)

Task	Resource
t_1	s_{12}
t_2	s_2
t_3	s_4
t_4	s_4
t_5	s_{32}
t_6	s_5
t_7	s_{11}
t_8	s_{31}

(c)

Task	Resource
t_1	s_2
t_2	s_2
t_3	s_4
t_4	s_4
t_5	s_{32}
t_6	s_5
t_7	s_{11}
t_8	s_5

(d)

Fig. 2. Resource allocation

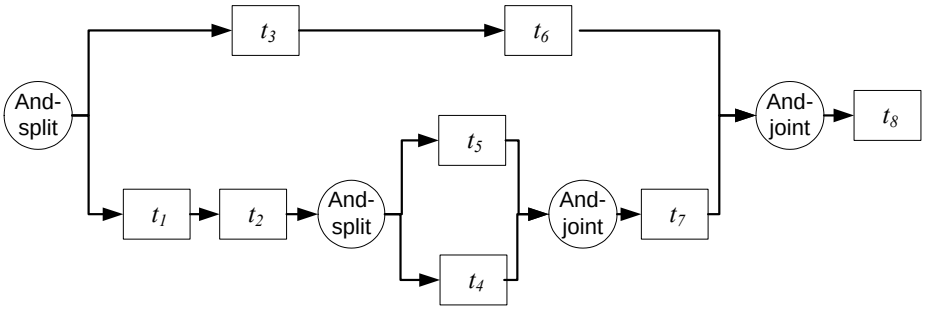

Fig. 3. Changed business process structure

From this example, we know that given a set of available resources, optimised resource allocation is based on the structure of business process and the requirements on the business process. Furthermore, a business process can be improved for the purpose of optimising resource allocation. In summary, we expect that the following requirements will be met in our resource allocation scheme:

- It should take into account the structural characteristics of a business process. The structural constraints and dependencies defined in a business process must be followed in resource allocation.
- It should guarantee the resource allocated with minimal expense within a given period.
- When necessary, a business process may be improved for better optimising resource allocation.

3 Role-based Business Process Model

In this section, a model comprising the definitions for resources, roles, tasks and business processes is introduced to describe the relationships among these notions that will be used in resource allocation and business process improvement.

Definition 1 (Resource). A resource *s* denotes an available unit for executing a task. In real cases, a resource can be a human, a machine or a computer. In this model, a resource has an attribute of role:

- *Role* indicates which group this resource is belonged to according to its position. In this model a resource can have only one role to perform, yet a role may include multiple resources.

Definition 2 (Role). A role *r* denotes a class of resources that own the same capability. A role has an attribute of cost.

- *Cost* denotes the monetary cost a resource of role *r* is chosen to perform a task. Cost in this model is valued by the hourly pay of the role.

A role is 'an abstraction to define the relationship between a set of resources and the capabilities of resources' [2]. The resources belonging to role *r* may be capable to perform several tasks, where function *capable*(*r*, *t*) depicts such mapping relationships. When resource *s* is capable of performing *t*, *capable*(*s*, *t*) is true. Therefore we have

$$capable(r, t) \;\&\&\; r = role(s) \rightarrow capable\ (s, t)$$

Definition 3 (Task). A task *t* is a logical unit of work that is carried out by a resource. A task has an attribute of *role*:

- *role* defines what kind of resources can perform this task. In other words, a task can be performed by resources that can match the role attribute of task. In this model, one task can have many roles, so each task has a none-empty role set R:$\{r\}$.
- *time* is associated with a role *r* and a task *t*, specifying the duration required for *r* to execute *t*. We denote this by a function time(*r*, *t*). When resource *s* of role *r* is used to execute task *t*, time can be returned by function time(*s*, *t*).

Definition 4 (Business process). A business process represents a series of linked tasks, which collectively describe the procedure how a business goal is achieved. The structure of a business process *p* can be modelled as an directed acyclic graph in the form of $P(T, E, G, type, v_s, v_t)$, where

(1) $T = \{t_1, t_2, \ldots, t_n\}$, $t_i \in$ T $(1 \le i \le n)$ represents a task in the business process fragment;
(2) $G = \{g_1, g_2, \ldots, g_m\}$, $g_i \in$G $(1 \le i \le m)$ represents a gateway in the business process fragment;
(3) E is a set of directed edges. Each edge $e = (v_1, v_2) \in E$ corresponds to the control dependency between *vertex* v_1 and v_2, where $v_1, v_2 \in T \cup G$;
(4) For each $v \in T \cup G$, *ind*(v) and *outd*(v) define the number of edges which take v as terminating and starting nodes respectively.

(5) *type*: $G \rightarrow Type$ is a mapping function, where *Type* = { *And-Joint*, *And-Split*, *Or-Joint*, *Or-Split* }. Therefore,
If *type*(*g*) = "*And-Split*" or "*Or-Split*" then *ind*(*g*) = 1, *outd*(*g*) > 1;
If *type*(*g*) = "*And-Joint*" or "*Or-Joint*" then *ind*(*g*) > 1, *outd*(*g*) = 1.

(6) v_s is the starting node of the business process fragment, which satisfies that $v_s \in T \cup G$ and $ind(v_s) = 0$;

(7) v_t is the terminating node of the business process *p*, which satisfies that $v_t \in T \cup G$ and $outd(v_t) = 0$;

(8) $\forall v \in N \setminus \{ v_s, v_t \}$, $ind(v) = outd(v) = 1$.

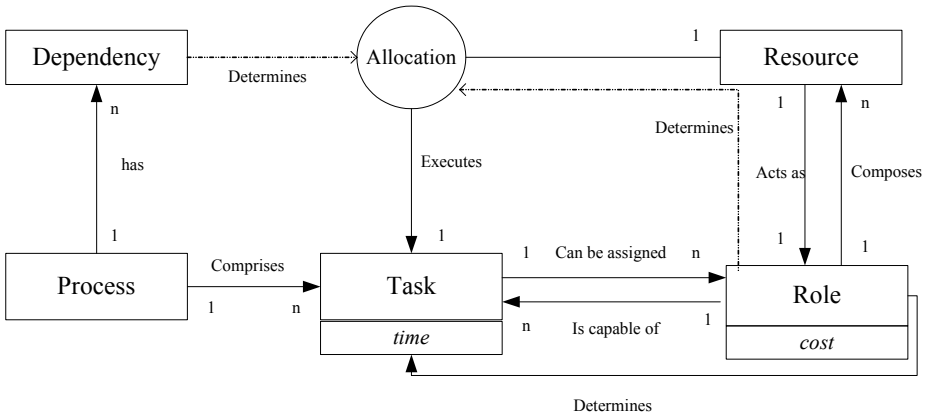

Fig. 4. Business process model

Figure 4 illustrates the relationships among these definitions for resource allocation purposes. A business process consists of a set of tasks and gateways. Each resource performs as one role, and one role may represent multiple resources. Any task can be executed by a set of roles, and a role may be capable of executing many tasks. When allocating resources to tasks, the allocation is subject to the dependency due to the structure of the process and the roles of resources. Cost is an attribute of a role, and it denotes the money that the enterprise has to pay for resources of that role when they are allocated to execute tasks. Time for a task to be executed is determined by which role is assigned to perform this task.

4 Resource Allocation and Business Process Improvement

As explained in the motivating example, we set the cost and time as the resource allocation optimisation criteria for the discussion in this paper. In this section, we first discuss the set of basic rules that follow the optimisation criteria. Then we describe the main data structures used for resource allocation. The main steps of our optimisation algorithms are highlighted afterward. Finally, we present two strategies and corresponding algorithms for resource allocation and business process improvement.

4.1 Basic Rules

As the resource allocation problem is to search for a resource allocation scheme that meets the requirements on cost and time. This searching process has to comply with the following rules:

***Rule* 1**. One resource can only serve for one task at one time. When this rule is violated, we call it resource allocation *conflict* at the task.

***Rule* 2**. The overall execution time of a business process is not allowed to exceed the time limit. If this rule is violated, resources will be required to be reallocated for shortening the process time.

***Rule* 3**. Whenever possible, the expense for executing a business process should be minimal.

In fact, *Rule* 3 is our optimisation objective while *Rule* 1 and *Rule* 2 are the constraints for achieving the objective. In other words, *Rule* 3 is applicable under the condition that *Rule* 1 and *Rule* 2 cannot be violated.

4.2 Data Structure

In Section 2, we introduced two tables for a business process p, the *role table* shown in Table 1 and the *capability table* shown in Table 2. For describing our resource allocation algorithms, we also require other two data structures: an *allocation table* and a path table.

An *allocation table* is used to record the allocation information for each task in a business process in the format of the following table.

Task(t)	*Role(r)*	*Resource(s)*	*Start Time(st)*	*End Time(et)*

When a task is allocated with a resource, the allocation table will be appended with a new record, where (1) "Task" t denotes the name of task; (2) "Role" r denotes the role that is selected for performing; (3) "Resource" s of role r is the specific resource that is assigned to execute t; (4) "Start Time" st is the starting time of t to be executed; (5) "End Time" et is the time t finishes. It is computed as $et = st + time(t, s)$, where $time(t, s)$ denotes the time that resource s needs to execute task t.

A *path table* records the information of all paths from the start node v_s to end node v_t on business process p.

Path(i)	*TaskSet(ts)*	*Time(tm)*

In the path table, "*Path*" i denotes the path number of this path. "*TaskSet*" ts records the set of tasks belonging to path i. "*Time*" t denotes the total time required to execute all the tasks in ts. Note, a task in business process p may appear in more than one path.

4.3 Resource allocation steps

As the cost and time for executing a task are unknown until it is allocated with actual

resource, the analysis on the business process performance is inevitably involved with resource allocation. Resource allocation to tasks will be done in such an optimised way that cost is minimal while satisfying time constraint.

According to the rules introduced in Section 4.1, optimised resource allocation for business process is carried out by the following three steps:

(1) A basic allocation strategy will be applied to searching for a resource allocation satisfying *Rule* 3 and *Rule* 1, which aims the minimal expense for executing a business process with balanced allocation for all paths. Surely, no resource is allocated to more than one task at any time.

(2) In case that the allocation scheme in Step (1) violates *Rule* 2, an adjustment strategy will be applied to shorten the execution time by re-allocating resources until time constraint is satisfied;

(3) In case that the time is less than the limit from Step (1) or Step (2), according to *Rule3*, the adjustment strategy will be applied to do the resource oriented business process improvement in order to achieve a lower expense while maintaining the time constraint to be satisfied.

Step (1) will be discussed in Section 4.4, and Section 4.5 introduces how Step (2) and Step (3) are carried out.

4.4 Basic allocation strategy

In the first step, we introduce the basic resource allocation strategy. The goal of this basic strategy is first to minimise the overall expense without considering the time limit. This strategy is achieved by two steps: Firstly, each task is allocated with a role which makes the expense to be minimal, and allocation table is updated according to the resource allocations. However, due to the characteristic of business process structure, it is possible that a role is over-allocated in such a way that at a time, a resource is allocated to perform more than one task, and hence allocation conflict is made and *Rule 1* is not satisfied. Therefore the second step is used to handle allocation conflicts through reallocation. In this procedure, overall expense is aimed to be minimal. Also, in order to improve efficiency, for the routings in parallel or selective blocks, balanced time is preferred for the allocation of different paths.

The basic resource allocation strategy is shown in Algorithm 1. Lines 1-3 initialise several variables *ntbp* for nodes to be processed, *pd* for processed nodes and *pathT* for storing all paths of the business process *p*. The function *genPathTable*(*p*) generates the path table for *p* with time for each path set to 0. Lines 4-17 are the loop for processing one node of the graph for *p*, starting from v_s to v_t. Function *getNextNode*(*ntbp*) (Line 5) finds the next node *v* in *ntbp* such that *v* cannot be processed before its predecessor nodes. For a task node *v* (Line 6), function *bestRole* (*v*) (Line 7) returns the role of minimal expense to execute *v*. A heuristic rule is used here: if two roles are capable to execute task *v* at same expense and one can only be assigned to *v*, then this role is selected. Function *allocRes*(*r*) (Line 8) assigns a resource *s* for *r*. When a resource of role *r* is allocated *v*, the one that is available to perform *v* is selected. Lines 9-10 calculate the maximum ending time for all paths involving *v* as a node. When all the required information is ready, function *alloc*(*t*, *r*, *s, st, et*) adds a new record into the allocation table *allocT*. Line 11 resets the ending

time for all paths for v. After that, for either a task node or a gateway node, the successor nodes of v will be added to *ntbp*, and v is removed from *ntbp* and added to *pd* (Lines 14-16).

To this point, each task has been allocated with a resource that is least expensive for executing the task. However, we need to check and see if *Rule* 1 is violated. If so, we have to change resources for the task at which the allocation conflict occurs. This is achieved by calling the function *conflictProc*(*allocT, 0, p*) to check the allocation starting from the beginning of the business process (Line 18).

Input:	*p*	–	a business process
	roleT	–	the associated role table for *p*;
	capaT	–	the capability table for *p*.
Output:	*allocT*	–	the result allocation table.

```
1   ntbp = { v_s }; // initial value of nodes to be processed
2   pd = ∅; // set for processed nodes
3   pathT = genPathTable(p);
4   while (ntbp ≠ ∅)
5      v = getNextNode(ntbp); // get v such that pred(v)∈pd or pred(v)=∅
6      if (v∈P.T) then
7         r = bestRole(v);
8         s = allocRes(r);
9         pts = paths(v); // find all paths that involve v
10        tm = max{pts[i].time}; // maximum time for all paths in pts
11        alloc(v, r, s, tm, tm+time(r, v))→allocT;
12        for each pt in pts do pt.time += time(r, v) end for;
13     end if
14     ntbp = ntbp ∪ succ(v);
15     ntbp = ntbp \ {v};
16     pd = pd ∪ {v};
17  end while
18  call conflictProc(allocT, 0, p);
19  return allocT;
```

Algorithm 1. Basic allocation

Algorithm 2 is a function for resolving resource allocation conflicts. *ConflictProc*(*allocT, t, p*) is to check conflict for tasks started after time t in the order they appear in p. When a task v_0 is checked, if conflicts exist, all conflicts involving this task are handled. When there is only one task v_1 conflict with v_0 and they are in the same nearest *And* block, three approaches can be made: reallocation on v_0 or on v_1, or change the structure. The longest path processing time in three cases are computed respectively and compared. Process is changed when its processing time is no less than other cases. Otherwise, resource is reallocated at the task which leads to minimal overall processing time. The resource that increase minimal expense but does not increase time is preferred. If there are multiple conflicts, each of them will be handled until there is no task conflict with v_0. Each task v conflicting with v_0 is selected, and

reallocation is done on the task in the longest path among those paths including v_0 or v. Function *replaceRow*(*allocT*, (v, r, s, $v.st$, $v.st.et$)) returns an allocation table that replaces the row for v in *allocT* with the specified new row. Function *adjustTime*(*allocT*, *tm, p*) update the start time and end time for those tasks that start after the time *tm* in *allocT* for process p, and returns the end time of v_t.

```
function conflictProc(allocationTable allocT, Time startTime, Process p)
timeline=startTime;
V={v | v.st ≥ startTime}
while(timeline < vt.st)
   select v0 ∈V : v0.st=min({v'.st| v'∈V})
   Vc={v'|v'.st< v0.et & v'∈V}
   timeline= v0.st;
   if (|Vc |=0) then
      V=V \{v0};
   else if((|Vc |=1& andStruc(v0, Vc [0])) then
      v1= Vc[0];
      r1=nextBestRole(v0); s1=allocRes(r1);        //reallocate v0
      allocT1=replaceRow(allocT, (v0, r1, s1, v0.st, v0.st+ time (r1, v0)));
      t1 = adjustTime(allocT1, v0.st, p);
      r2=nextBestRole(v1); s1=allocRes(r2);        //reallocate v1
      allocT2=replaceRow(allocT, (v1, r2, s2, v1.st, v1.st+ time(r2, v1)));
      t2 = adjustTime(allocT2, v1.st, p);
      p'=changeP(p, v0, v1);                       //change structure
      r3= allocT[v0].role; s3= allocT [v0].resource;
      allocT3=replaceRow(allocT, (v0, r3, s3, v2.et, v2.et+ time(r3, v0)));
      t3 = adjustTime(allocT3, v0.st, p');
      if t3≤min(t1, t2) then   p=p'; allocT=allocT3;
      else if(t2<t1) then   V=V \{v0}; allocT= allocT2;
      Else   V=V \{v0}; allocT= allocT1;
      end if
   else
      allocT'=allocT;
      for each v∈ Vc
         pt = longestPath(paths(v0) ∪ paths(v);
         v'= pt.Tasks∩{ v0, v };
         r'=nextBestRole(v'); s'=allocRes(r');
         allocT=replaceRow(allocT, (v', r', s', v'.st, v'.st+ time (r', v')));
         adjustTime(allocT, v'.st, p);
         if (v' = v0) then V=V \{v0};   break;   end if;
      end for
   end if
end while
return allocT;
```

Algorithm 2. Resource Allocation Conflict Resolution

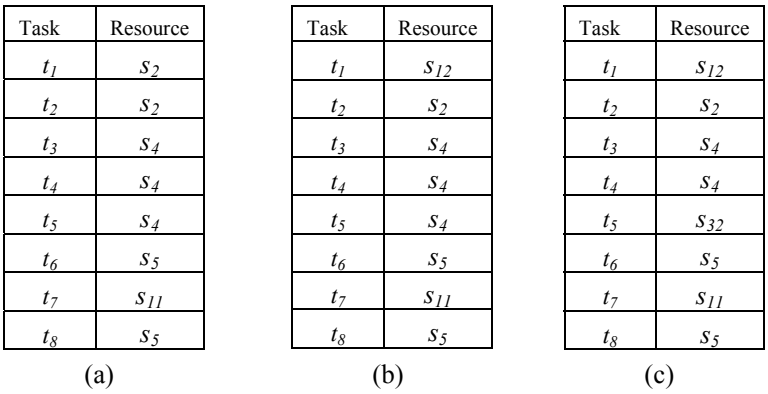

Fig. 5. Allocation conflict and handling

Consider the example introduced in Section 2. The basic allocation algorithm first comes up with an initial allocation as shown in Figure 5(a). After that, allocation conflicts are checked. Starting from t_1, each task will be checked. When t_1 is examined, we can easily detect that task t_1 conflicts with t_2 because they use same resource s_2 in an intervening duration. At this stage, reallocation will be applied on t_1 because it is on the longest path, and hence the overall time can be reduced and times of different paths become more balanced. This results in the change shown in Figure 5(b). Similarly, the conflict between t_4 and t_5 can be resolved by reallocating s_5 with s_{32} as shown in Figure 5(c).

4.5 Adjustment strategy

A time constraint is not considered in the basic strategy. As Rule 2 stated, the overall execution time of a business process is not allowed to exceed the time limit. In case this rule is violated, i.e., the ending time of v_t in the allocation table is greater than the time limit t_{max}, we have to follow Rule 2 and shorten the time until it is within t_{max}. However, if the overall time is less than t_{max}, we may relax the time and to reduce the expense based on possible business process improvement.

First we discuss the adjustment strategies for the case that time constraint is violated. We have several heuristic rules. As the overall time is dependent on the longest path in the business process, it is more effective to adjust those tasks belonging to the longest path. When a task t currently assigned with resource s is reallocated with s', both the time and cost may change accordingly. Assume $\Delta time$ denotes the time reduced and $\Delta expense$ is expense increased. The value of $\Delta time/\Delta expense$, called as *compensation ratio*, can be used to measure the effectiveness of an adjustment. Obviously a higher *compensation ratio* is preferable because more time can be reduced with less expense. If this adjustment is on a task that belongs to the longest path, the overall time will be reduced accordingly.

Algorithm 3 is designed for handling time constraint violation. The reallocation process is done until time constraint is satisfied. Firstly, the longest path *pt* is selected.

In Lines 3-17, we select a task on *pt* to be reallocated. *mcr*, with 0 as initial value, is used to record the maximal compensation ratio for each reallocation, and *mallocT* is the allocation table after such a reallocation has been made. For each task *v* in *pt*, the role r_v of maximal compensation ratio (calculated by *maxRatioRole*(*v*)) for *v* selected, and resource s_v of r_v is reallocated, and *allocT'* is the allocation table to record this reallocation in Lines 5-7. If *v* is in And block, this reallocation may cause allocation conflict, hence function *conflictProc*(*allocT', v.st, p*) is called to handle potential resource allocation conflicts from the starting time of *v*. Lines 9-11 computes the *compensation ratio cr* of this reallocation from overall perspective. If *cr* is lager than *mcr*, *mcr* is updated to *cr* and *mallocT* is changed to *allocT'* in Lines 13-14. After compensation ratio on all the tasks in *pt* has been computed, *allocT* is updated to *mallocT* and returned if time constraint is satisfied.

Input:	*p*	-	allocation table(based on business process *p*)
	allocT	-	the old allocation table
	pathT	-	a path table for *p*
	t_{max}	-	time limit
Output:	*allocT*	-	new allocation table

```
1   while(v_t.et> t_max)
2     pt = longestPath(pathT);
3     mcr=0;
4     for each v ∈ pt.Tasks
5       r_v = maxRatioRole(v);
6       s_v = allocRes(r);
7       allocT'=replaceRow(allocT, v, r_v, s_v, v.st, v.st+time(r_v, v));
8       if(v is in And block) then call conflictProc(allocT') end if;
9       et= allocT[v_t].et; exp=expense(allocT[v].role, v);          //previous
10      et'= adjustTime(allocT', v.st, p); exp'=expense(r_v, v);    //new
11      cr=(et-et') / (exp'-exp);
12      if(cr>mcr) then
13        mcr=cr;
14        mallocT=allocT';
15      end if
16    end for
17    allocT=mallocT;
18  end while
19  return allocT;
```

Algorithm 3. Time constraint violation handling approach

Come back to the example introduced in Section 2, the outcome from Algorithm 1 is shown in Figure 5(c), where the time constraint is violated. At this stage, adjustment strategy must be applied in order to guarantee time constraint be satisfied. The longest path is computed as path 1 ($t_1 \rightarrow t_4 \rightarrow t_7 \rightarrow t_8$). Therefore, reallocation will be done on tasks on path 1. It is easy to calculate and compare the compensation ratio for replacing each task in this path, and find that reallocation for t_8, which achieve the maximal compensation ratio. Therefore reallocation on t_8 is applied and the new

allocation is shown as Figure 2(c).

Now we discuss the adjustment strategies for the case that the overall time is less than the time limit t_{max}. We also have some heuristic rules to reduce expense while relaxing time. In Algorithm 1, the resource allocation conflict caused by two parallel executing tasks that required same role with minimal expense but there was no sufficient resource for allocating them was resolved by reallocating one of them with a higher expense resource. In this scenario, actually, we could change the structure of the process to support both of them to be assigned with the original cheapest resources. The reduced expense through process change is usually compensated with increased time. Therefore, it is wise to make change to those tasks that do not belong to a long path, because a task in a long path has less room for increasing time.

Algorithm 4 is designed for relaxing time for maximum reduction of expense. *allocT'* and *p'* records the allocation table and the process structure after change, and they are initialised with *allocT* and *p* respectively (Lines 1-2). While the end time of *allocT'* is less than t_{max} (Line 3), the process change is accepted (Line 22) and further process improvement can be made based on *allocT'* and *p'*. Process change is realized in the following way. In the process *p* the shortest path *pt* is first selected (Line 4), and the change is focused on task in *pt*. For each task *v* in *pt* (Line 5), we select a role *r* by function *minRatioRole*(*v*) that looks for the maximum expense deduction with minimum time increase (Line 6). If such a role exists and it is not used by *v* (Line 7), function *getTasks*(*r*, *v.st*, *v.et*) returns the set of tasks that are within the same nearest *And* block, are overlapped with *v*, and are assigned resources with the same role (Line 8), then Lines 9-16 finds the task *mv* with the minimal time in its involved longest path. Lines 17-21 change the structure and replace the resource.

Input:	*p*	-	allocation table(based on business process *p*)
	allocT	-	the old allocation table
	pathT	-	a bath table for *p*
	t_{max}	-	time limit
Output:	*allocT*	-	new allocation table
	p	-	new process structure

```
1   allocT'=allocT;    //new allocation table after reallocation, initially allocT
2   p'=p;              //new process structure after reallocation, initially p
3   while(allocT'[v_t.et]< t_max)
4     pt=shortestpath(PathT);
5     for each v∈pt
6       r = minRatioRole(v);
7       if(r≠null and allocT[v].role≠r)
8         ts=getTasks(r, v.st, v.et);
9         mtm= v_t.et;
10        for each v'∈ts
11          ts'=longestPath(paths(v'));
12          if (ts'.time<mtm)
13            mv=v'; mtm=ts'.time;
14          end if
15        end for
```

```
16        end for
17        p'=changeP(p, v, mv);
18        s=allocRes(r);
19        allocT'= replaceRow(allocT, (v, r, s, v.st, v.st+time(r, v)));
20        call conflictProc(allocT', v.st, p');
21        adjustTime(allocT', v.st, p');
22        if(allocT'[v_t.et]< t_max) then allocT=allocT'; p=p'; end if;
23      end while
24      return allocT;
```

Algorithm 4. Relaxation approach

For the example in Section 2, when the time constraint is 11.5 hours rather than 8, allocation after *Algorithm* 3 is shown as Figure 5(c). However, t_2 is not using resource of best role due to conflict with t_1, therefore, we examine if process change contributes to expense reduction. In the new process shown as Figure 3, resource allocation can be made as shown in Figure 2(d) and the expense is reduced. Therefore, in this case, we adopt the changed business process structure in Figure 3 and resource allocation in Figure 2(d).

5 Related Work and Discussion

Resource allocation is a topic related to task/workflow scheduling, which seeks the proper execution sequence for a set of tasks to achieve specific goals. This procedure is dependent on the resource allocated to execute the tasks. Scheduling problem has been discussed at task level and workflow level in previous work. Task level scheduling is based on independent tasks. Many algorithms have been proposed to schedule tasks within the homogenous systems in previous literatures such as [12, 15]. Our paper investigates the resource allocation in a heterogeneous environment, and for heterogeneous scheduling many work has been done. In order to reduce the processing time, Topcuoglu, Hariri and Wu have proposed an Earliest-Finished-Time (HEFT) algorithm in [9]. The HEFT algorithm selects the task with the highest upward rank value at each step and assigns the selected task to the processor. This algorithm can minimise the earliest finished time, but the cost is not considered. The work in [6] deals with problem of scheduling tasks to minimise transition cost within a rigid deadline for completion. A mathematical formulation and a two-phased algorithm are introduced to solve this problem in [6]. All the works on task level scheduling have their limit in effective resource management because they do not follow the structure of a business process.

Compared with task scheduling, workflow scheduling is based on tasks of compulsory execution order constraint. In [5], Johann, Euthimios and Michael have proposed some modelling primitives to express the lower- and upper-bound of time constraints for workflow scheduling. In addition, they have also developed the technique for checking if time constraints are satisfied at process build and instantiation time, and enforcing these constraints at run-time. Their work has solved the problem of deadline constraints and provided the solution about how to avoid the

deadline violation, but this work does not include any resource allocation strategy. Work [8, 3] has modelled the workflow with resource constraints. In the framework of [8], workflow scheduling is under both temporal constraints and resource constraints (composed of control constraints and cost constraints). However, this work does not touch how to manage and allocate resource in workflow. A number of Grid workflow management systems such as [1, 11] have proposed scheduling algorithms to facilitate the workflow execution and minimise the execution time. Yu and Buyya [13, 14] have considered not only execution time, but also execution cost. In [14], Yu has proposed a genetic algorithm for scheduling scientific workflows for utility Grid applications by minimising the execution time while meeting user's budget constraint. In [13], the problem of minimising the overall cost while meeting user's deadline constraint for the scientific workflow scheduling is also investigated.

In contrast to the previous work, the work discussed this paper is intended to improve performance of business processes by integrating resource allocation with business process structural improvement. Compared to existing approaches, our approach has the following features:

- Business process improvement for providing better resource allocation. In our approach, the structure of process can be modified to better adapt to the resources allocation when necessary. Based on the analysis on relationship between the types of available resource and the business process structure, such modifications can make the process structure more effective for available resources. In this way, resource allocation on the new process has better performance than allocation on the old process structure.
- Resource optimisation based on business process characteristics. In our approach, business process characteristics, which include the constraints and dependencies pre-defined by the process structure, are preserved in the resource allocation.
- Requirement oriented resource allocation. Our approach is able to guarantee the requirements set for resource allocation been satisfied. In this paper, time constraints and cost requirements for a business process have been considered.

6 Conclusion

This paper discussed the problem of resource allocation for business processes. An approach was proposed to allocate resources to tasks in such a way that the total expense is minimal while the requirement of executing time on a business process is satisfied. In this approach, a basic strategy is applied first to minimise total expense. Then, an adjustment strategy is applied to modify allocation such that the time constraint is met in a smart way. The advantage of this approach over previous approaches lies in the relationship between the effective resource allocation and the business process improvement. To cater for the resource allocation requirements and available resources of an enterprise, the structure of a business process can be changed. After the structure of the business process is changed, the performance of the business process in terms of better utilising resources is improved.

In the future, we plan to take more task dependencies into account in the resource allocation and explore more structure changes of a business process.

References

1. Blythe, J., Jain, S., Deelman, E., Gil, Y., Vahi, K., Mandal, A., Kennedy, K.: Task Scheduling Strategies for Workflow-based Applications in Grids. Proceedings of the 5th International Symposium on Cluster Computing and the Grid, Cardiff, UK (2005) 759-767
2. Du, W., Eddy, G., Shan, M.-C.: Distributed Resource Management in Workflow Environments. Proceedings of the 5th Database Systems for Advanced Applications, Melbourne, Australia (1997) 521-530
3. Etoundi, R.A., Ndjodo, M.F.: Feature-oriented Workflow Modelling based on Enterprise Human Resource Planning. Business Process Management Journal 12 (2006) 608-621
4. Huang, Y.-N., Shan, M.-C.: Policies in a Resource Manager of Workflow Systems: Modeling, Enforcement and Management. Proceedings of the 15th International Conference on Data Engineering. IEEE Computer Society, Sydney, Austrialia (1999) 104
5. Johann, E., Euthimios, P., Michael, R.: Time Constraints in Workflow Systems. Proceedings of the 11th International Conference on Advanced Information Systems Engineering. Springer-Verlag (1999)
6. Lee, Y.-J., Lee, D.-W., Chang, D.-J.: Optimal Task Scheduling Algorithm for Non-pre-emptive Processing System. Proceedings of the 8th Asia-Pacific Web Conference, Vol. 3841. Springer-Verlag, Harbin, China (2006) 905-910
7. R-Moreno, M.D., Borrajo, D., Cesta, A., Oddi, A.: Integrating Planning and Scheduling in Workflow Domains. Expert Systems with Applications 33 (2006) 389-406
8. Senkul, P., Toroslu, I.H.: An Architecture for Workflow Scheduling under Resource Allocation Constraints. Information System 30 (2004) 399-422
9. Topcuoglu, H., Hariri, S., Wu, M.-Y.: Performance-Effective and Low-Complexity Task Scheduling for Heterogeneous Computing. IEEE Transactions on Parallel and Distributed Systems 13 (2002) 260-274
10. van der Aalst, W., van Hee, K.: Workflow Management: Models, Methods, and Systems. MIT Press, Cambridge (2004)
11. Wieczorek, M., Prodan, R., Fahringer, T.: Scheduling of Scientific Workflows in the ASKALON Grid Environment. SIGMOD Record 34 (2005) 56-62
12. Wu, A.S., Yu, H., Jin, S., Lin, K.-C., Schiavone, G.A.: An Incremental Genetic Algorithm Approach to Multiprocessor Scheduling. IEEE Transactions on Parallel and Distributed Systems 15 (2004) 824-834
13. Yu, J., Buyya, R., Tham, C.-K.: Cost-Based Scheduling of Scientific Workflow Application on Utility Grids. International Conference on e-Science and Grid Technologies, Melbourne, Australia (2005) 140-147
14. Yu, J., Buyya, R.: Scheduling Scientific Workflow Applications with Deadline and Budget Constraints using Genetic Algorithms. Scientific Programming 14 (2006) 217-230
15. Zomaya, A.Y., Teh, Y.-H.: Observations on Using Genetic Algorithms for Dynamic Load-Balancing. IEEE Transactions on Parallel and Distributed Systems 12 (2001) 899-911