Resource Planning for Massive Number of Process Instances

Jiajie Xu, Chengfei Liu, and Xiaohui Zhao

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

Abstract. Resource allocation has been recognised as an important topic for business process execution. In this paper, we focus on planning resources for a massive number of process instances to meet the process requirements and cater for rational utilisation of resources before execution. After a motivating example, we present a model for planning resources for process instances. Then we design a set of heuristic rules that take both optimised planning at build time and instance dependencies at run time into account. Based on these rules we propose two strategies, one is called holistic and the other is called batched, for resource planning. Both strategies target a lower cost, however, the holistic strategy can achieve an earlier deadline while the batched strategy aims at rational use of resources. We discuss how to find balance between them in the paper with a comprehensive experimental study on these two approaches.

1 Introduction

Nowadays, how to improve the business process performance and service quality becomes one of the key issues for organisations to survive and thrive in the market competitions. Works [4, 5, 7, 8] indicate that resource allocation is an important topic for business process management and service quality improvement. With limited resources, business processes are expected to execute with lower cost and in shorter period [3, 13]. In addition, resources should be properly allocated for business processes according to the natural characteristics and rationalities, such as periodical availability, fair use, and load balance.

The resource planning can be further complicated in the scenario where a massive number of process instances exist. The dependencies and conflicts between concurrent process instances and exclusively occupied resources have to be taken into account in the resource planning. For an organisation, resources must be allocated to guarantee the assigned volume of process instances can be finished before a negotiable deadline to customers at an acceptable lower cost to organisation. Due to those factors such as inventory capacity and resource availability patterns, the massive number of process instances have to be executed in batches for balancing the load.

Several works [1, 6, 13] have been done in investigating the process oriented resource planning. However, none of them considered all the above problems. Aiming

R. Meersman, T. Dillon, P. Herrero (Eds.): OTM 2009, Part I, LNCS 5870, pp. 219–236, 2009.

to address all these issues, we introduce a novel model to specify and analyse the relationships among roles, resources and process structures. For an optimised and rational resource planning for a massive number of process instances, we design a series of heuristic rules that consider a deadline posed by customers, and a lower cost, conflicts in concurrent use of resources, periodical resource availability and inventory capacity, etc. from the organisation point of view. Based on these rules, we propose two strategies, one is called holistic and the other is called batched, for resource planning. Both strategies target a lower cost, however, the holistic strategy can achieve a shorter deadline while the batched strategy aims at rational use of resources. We discuss how to find balance between them in the paper. A comprehensive experimental study is also conducted to demonstrate how our approach can support organisations to plan and adjust resource allocations for a massive number of process instances. The contribution of this paper is listed as the following aspects:

- Take into account the batching of process instance execution, to enhance the rational scheduling and utilisation of resources;
- Plan the resource allocation at build time to guarantee that all requirements on time, cost, batch pattern, etc., can be satisfied;
- Consider the run time dependencies between process instance in resource planning, and therefore support the multi-instance parallel execution planning;

The remainder of this paper is organised as follows: Section 2 discusses our problem for requirement oriented build-time resource optimisation with a motivating example. Section 3 introduces a role based business process model, which defines the related notions for resource planning and their relationship. Based on this model the problem to be addressed in this paper is formalised. Two strategies for process oriented resource planning are presented and discussed in Section 4. In Section 5 we conduct an experimental study to evaluate and compare two strategies proposed in this paper. Section 6 reviews the related work and discusses the advantages of our approach. Lastly, concluding remarks are given in Section 7.

2 Motivating Example

In this section, we analyse the resource planning for massive number of process instances with motivating example.

Figure 1 shows a simplified business process, as well as a set of resources which belong to different roles. Here, each resource has a maximal work load of 24 hours, and different roles have different capabilities for performing certain tasks with given expense rates. For the organisation, it has to consider how to meet time and cost requirements and how to maintain the rationalities of resource utilisation when allocating resources to all process instances.

Given a massive number of process instances, their execution can be enforced in two strategies, i.e., the holistic strategy - handling all instances as a whole lot and the batched strategy - handling the instances in batches. For illustration purpose, the number of instances is set to 40. Different strategies result in different resource allocation plans. Figure 2 (a) shows the resource plan for a holistic solution, where instances may overlap with each other in execution, and therefore resources should be allocated in an interleaved way to avoid resource conflicts. According to this holistic

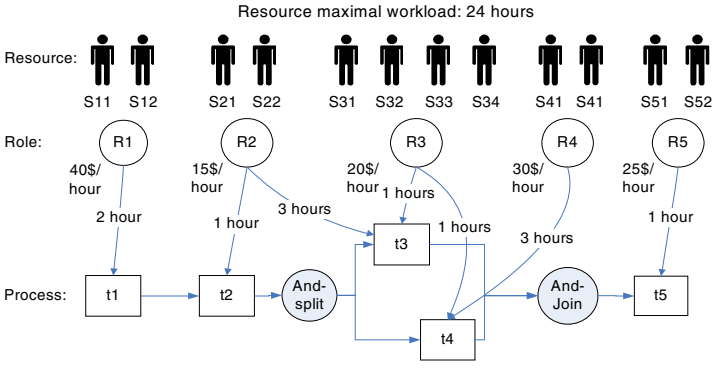

Fig. 1. Business process and environment

solution and the resource plan, 40 process instances can be finished in 120 hours at the cost of $1000. Using the batched strategy, a solution is shown in Figure 2 (b), where each batch contains 8 instances and can be finished in 30 hours, and 5 batches are executed in the sequential order. In this case, the total execution time becomes 150 hours and the cost is $980.

The second solution is intuitively preferable though it takes a bit more time for execution. This is because that it allows the resources to be used periodically, which adapts to the natural characteristics and rationalities of resources, such as periodical availability, fair use, and load balance. Figure 2 (c) shows the job schedule of s_{11} in the two discussed strategies. In the first solution, this resource has a rather skewed workload distribution, which may cause extra cost for the organisation to schedule the resource, e.g., double pay for overtime hours for human resources. While in the second solution the resource is evenly used.

Particular attention should be paid when evaluating a holistic solution and a batched solution, because their performance is sensitive to the given deadline, budget limit, number of instances, etc. In the example, if the deadline is set to 120 hours, then only the holistic solution is feasible. Yet, if the organisation is able to negotiate with customers to relax the deadline to 150 hours, the batched solution becomes feasible and more attractive, as it meets the deadline and follows a periodical resource utilisation. Alternatively, the organisation may negotiate with customers to change the number of instances to 36 if the deadline is strict. As shown in Figure 2 (d), we can have another batched solution that runs 4 batches at $900, with each batch running 9 instances within 30 hours, and thereby can meet the strict deadline of 120 hours.

From the above discussion, we can see that the alternative solutions for the batched strategy according to different input arguments are very useful for organisations to negotiate and determine resource allocations. Here, we summarise the main factors for tunning batched solutions and determining resource allocations:

- Cost difference relative to deadline relaxation;
- Time difference relative to cost increment;
- Time and cost difference relative to the number of assigned instances;
- Time and cost relative to the number of batches;
- Intention of using resources in a periodical manner.

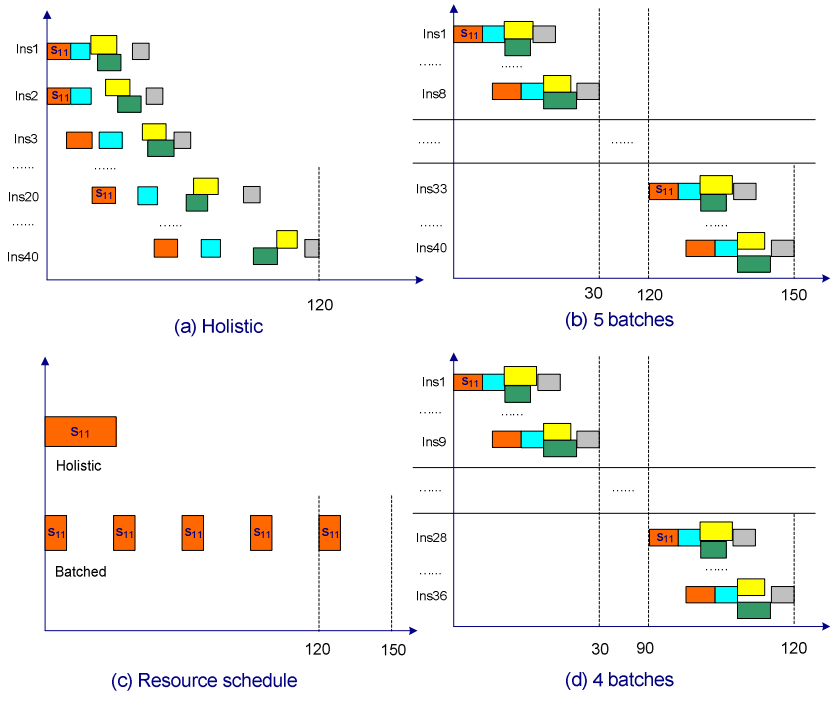

Fig. 2. Process execution

Aiming to assess the inter-influence between these factors, we propose a comprehensive resource planning approach to calculate these factors and determine the influences between them. Our approach is expected to support organisations to negotiate with customers and seek an optimal resource allocation plan.

3 Role Based Business Process Model with Problem Statement

In this section, a model comprising resources, roles, tasks, and business processes is presented to define those notions used in resource planning and describe their relationship. In this model, the scope of all definitions is on a single business process. A batch is a class of instances to be processed in parallel. Resource abundance is a concept to measure the capability and capacity relationship between the tasks in business process and the roles of available resources. Obligation indicates how much a role is required by the tasks, and support measures how much available workload can be used for a given task.

Definition 1 (Resource). A resource *s* denotes an available unit that is necessary for executing a task. In real cases, a resource can be a human, a machine or a computer. In this model, each resource has two attributes:

- *Role* indicates which group this resource belongs to according to its position. In this model a resource can have only one role to perform, yet a role may include multiple resources. For a resource *s*, function *r(s)* returns the role of this resource.
- *Maximal workload* denotes how much time this resource can provide service for the business process. Function *mload*(*s*) returns maximal workload of resource *s* for the business process.

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

- *Cost* denotes the monetary cost a resource of role *r* needs to spend to perform a task. In this model it is valued by the hourly pay of the role. Given a role *r*, function *costRatio(r)* returns the cost of this role.

The resources belonging to role *r* may be capable to perform several tasks, where function *capable*(*r*, *t*) depicts such capability mapping. Function *capable*(*r*) returns the task set $\{t|capable(r, t)\}$ role *r* is able to perform.

Definition 3 (Task). A task *t* is a logical unit of work that is carried out by a resource to process one instance. 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 has at least one role, therefore has a none-empty role set $R:\{r\}$.

When a resource *s* is allocated to a task *t*, the execution time is determined by the role of *s*. It can be returned by function *time*(*t*, *r*), and *r*=*r*(*s*). The cost for *s* to execute *t* is returned by function $cost(t, r) = time(t, r) \times costRatio(r)$. Accordingly, workload of *time*(*t*, *r*) will be consumed from *s*.

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 a directed acyclic graph in the form of $P(T, E, G, v_s, v_t, P)$, where

(1) $T = \{t_1,\ldots, t_n\}$, $t_i \in T$ $(1 \le i \le n)$ represents a task in the business process;
(2) $G = \{g_1,\ldots, g_m\}$, $g_i \in G$ $(1 \le i \le m)$ represents a gateway in the business process;
(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) v_s is the starting node of the business process, which satisfies that $v_s \in T \cup G$;
(5) v_t is the terminating node of the business process, which satisfies that $v_t \in T \cup G$;
(6) $Pth = \{pth_1,\ldots, pth_k\}$, $pth_i \in P$ $(1 \le i \le k)$ represents a path consisting a set of linked tasks from v_s to v_t. Total execution time of a path *pth* is *pth.tm*, which is calculated as the sum of processing time of each task on this path.

We analyse the environment to improve resource planning efficiency. Which resource is excessively required? Which task may have resource crisis? We use resource abundance to represent the capability and capacity between tasks and roles. Further, support of tasks and obligation of roles are defined based on resource abundance.

Definition 5 (Resource abundance). If a role r is capable of executing task t, resource abundance of r for t represents how much workload r can contribute to perform t. It is calculated as:

$$w(t, r) = \sum_{\forall s:r(s)=r} mload(s) / (n \times time(t, r))$$

where $\sum_{\forall s:r(s)=r} mload(s)$ is the total available workload of this role. n is the number of instances, and $n \times time(t, r)$ is the workload required to be assigned on t. $w(t, r)$ measures how much ratio of the total workload required by t can be contributed by r. For instance, task t_1 requires workload 18 hours on role r_1 to perform for all the instances, and resource of r_1 has workload 24 hours in sum, then $w(t_1, r_2) = 24/18=4/3$.

Definition 6 (Support). Support denotes the richness of resources available for a task t. Maximal support $supt_{max}$ is based on all resource used for t. Mean support $supt_{mean}$ is based on resource equally used for all capable tasks. They are calculated as:

$$supt_{max}[t] = \sum_{\forall r:capable(t,r)} w(t, r), \quad supt_{mean}[t] = \sum_{\forall r:capable(t,r)} w(t, r) / |capable(r)|$$

where $supt_{max}$ of t is the sum of value on resource abundance for all roles capable for t. Notice that if $supt_{max}[t]<1$, resource is insufficient to execute t and the process is not executable. In our example, task t_1 has two capable roles r_1 and r_2, $w(t_1, r_1) = (2\times12)/(12\times1.5)=4/3$ and $w(t_1, r_2) = 1$, then $supt_{max}[t_1] =7/3$. $supt_{mean}$ considers the situation resource workload is evenly used for capable tasks. In this example, we have $supt_{mean}[t_1] =4/3+1/3=5/3$.

Definition 7 (Obligation). Obligation denotes how much a role is obligated to the tasks. It reflects the tightness of this role in resource planning. Given the resource abundance relationship, obligation of a role is computed as:

$$o(r) = \sum_{\forall t:capable(t,r)} 1 / supt_{mean}[t]$$

In the pattern roles are equally used, for each task t executable by role r, the degree of t requires r is $1/supt_{mean}[t]$. Obligation of r is the sum for all tasks. For example, if both t_3 and t_4 can be performed by r_3, and $supt_{mean}[t_3]=4$, $supt_{mean}[t_4] \approx 4.42$, then $o(r_3)=(1/4)+(1/4.42) \approx 0.48$.

Definition 8 (Batch). Assume there are totally n instances to be processed, and a batch contains m instances ($m \leq n$). Those m instances in a batch are processed in parallel. We assume that there is no inference between instances belonging to different batches during execution.

Problem Statement

Suppose that a business process p together with its task set T, resource set S, role set R, the mappings from S to R, i.e., $S \rightarrow R$, and from R to T, i.e., $R \rightarrow T$, are given. The requirement is to process n instances of p in b batches before a deadline t_D.

Table 1 lists the involved variables in this problem statement, which will be used through out this paper.

Table 1. Variable definitions

n	The number of assigned instances
t_D	The given deadline
b	The number of batches
m	The size of batch
t_b	The execution time for one batch

Our goal is to find a batching scheme, i.e., the proper number of batches b, and a resource planning scheme ($S \rightarrow T$) such that:

(1) Time constraint can be satisfied ($t_b \times b \leq t_D$);
(2) Less total cost;
(3) The resource allocation is conflict free, i.e., no resource serves for multiple tasks or instances simultaneously.

Note, the holistic solution is a special case of batched processing, i.e., b=1.

4 Resource Planning

In this section, we discuss the strategies for finding resource planning solutions according to the problem statement discussed in the previous section. A set of heuristic rules for resource planning is first presented in Section 4.1. Based on the heuristic rules, Section 4.2 and Section 4.3 introduce the holistic and batched strategies, respectively. In Section 4.4 we discuss the balance of customer requirements and rationality of resource use.

4.1 Heuristic Rules

Because of the instance dependencies, resource allocation on one task may affect the execution of other tasks. Then, given a task in the business process, which resource to execute the task is optimal in the overall perspective? When multiple instances are executed in parallel, each of them needs to be allocated with a resource to execute this task. We use some heuristics in the allocation. The rules used in this paper are listed as follows:

Rule 1. Skewed tasks are allocated earlier than others in order to avoid crucial resources misused by other tasks.
Rule 2. If the obligation of role r is over a threshold ∂, which means resources of this role are over-required, such resources should be preserved if possible in the resource allocation for later use.
Rule 3. Resource allocation should avoid, if possible, increasing the overall time of the longest path.
Rule 4. Resource allocation intends to use economical resource as task performer in order for minimal overall cost.

Rule 1 is about the allocation sequence, to improve the effectiveness of resource planning. *Rule 2* is intended to avoid workload problem and therefore improve efficiency. In the resource allocation, there is a balance between saving cost and saving time. *Rule 3* and *Rule 4* deal with time and cost saving respectively. The balance is, if resource allocation on a task is likely to increase overall time, *Rule 3* is used to avoid violating the deadline; otherwise, *Rule 4* is applied to reduce cost. Through the heuristic rules, many solutions can be applied to plan the resource and schedule instances. In the holistic planning strategy of this paper, we propose a two step solution. In the first step, a basic resource allocation is applied to plan the resources for all instances for build time optimisation. However, due to the process structural characteristic and instance dependency, run time conflicts may occur in the first step. The second step handles those conflicts. In the batch based resource planning strategy, instances are executed in batches. We will address the problem of how to partition instances into batches, and then optimise the use of resource according to the fixed batch pattern.

4.2 Holistic Based Resource Planning

In this strategy, resource planning is applied for all instances. Based on the heuristic rules discussed in Section 4.1, we develop a two step strategy in 4.2.1 and 4.2.2 to optimise the use of resources by considering the total time constraint, instance dependencies for run time execution and cost optimisation.

4.2.1 Basic Resource Allocation

We first plan resource for the business process without considering conflicts. Such planning is as n instances processed in parallel. According to *Rule 3*, we keep track of the longest path among all instances to avoid increasing of the time required for processing the batch t_b. Given n instances ($ins_1,\ldots,ins_n$) to allocate, every instance ins_i has k paths $Pth_i=\{pth_{i1},\ldots, pth_{ik}\}$, then among those $k\times n$ paths in the batch, the longest path is such a pth_{ij} ($1\le i\le n$, $1\le j\le k$) with the maximal value of processing time $pth_{ij}.tm=\sum_{t\in pth_{ij}} time(t)$, where we set the processing time of task $t\in pth_{ij}$ as minimal executing time of all capable resources $time(t)[ins_i]=min(\{time(t, s)|\ capable(t, s)\})$ by default, and after t is allocated with resource s processing time is updated to $time(t)[ins_i]=time(t, s)$. In this way we track the whole structure to target the longest path in overall perspective, and then manage resources reasonably to reduce the longest path and enable instances within a batch to be executed in balance.

According to the resource abundance, some tasks may have multiple allocable resources, while some tasks may have few. We call those tasks whose mean support is lower than a threshold σ as 'skewed task'. We have skewed task set $SkT=\{\ t\mid t\in T\wedge supt_{mean}(t)<\sigma\ \}$. According to *Rule 1*, tasks in *SkT* should be allocated before other tasks to avoid resource shortage on the skewed tasks in resource planning.

Therefore, skewed tasks in *SkT* are firstly allocated before others in the ascending order of their support value. As followed, the first unallocated task $t\in T$ on the longest path of the batch structure is selected for planning iteratively, until the whole batch of instances are allocated. When task t is selected, we apply Algorithm 1 to allocate suitable resources to this task for each instance in the batch. In Algorithm 1, the input *allocT* is a resource plan before task t of instances in the batch are allocated, while the output *allocT'* is the new plan after t is allocated.

Input:	n	- number of instances
	S	- set of resources that are available
	t	- the task to be allocated for m instances
	$allocT$	- resource planning before allocation on t
Output:	$allocT'$	- resource planning after allocation on t

```
1   S1={ s | s∈S, r=r(s), capable(t, r) ∧ mload(s)≥ time(t, r) };
2   S2={ s | s∈S ∧ o(r)<∂ , r = r(s) };
3   TS={ t[ins_i] | 1 ≤ i ≤ m };
4     if ∃t[ins_j] ∈TS:t[ins_j] ∈ lpth(Pth_11,…,Pth_mk) then
5   if |S2|>m then   s=getMinTime(S2);
6     else          s=getMinTime(S1);
7     end if
8     allocT'=alloc(t[ins_j], s);  TS = TS - {t};
9   end if
10  for each t[ins_k] ∈TS in descending order of rlpth(t).tm
11    S3={s | s∈S1 ∧ rlpth(t[ins_k]).tm+time(t, s)≤ lpth(Pth1,…,Pthm).tm };
12      if S2∩S3 ≠ φ then  s=getMinCost(S2∩S3);
13     else if S2≠ φ then  s=getMinTime(S2);
14     else                s=getMinOblig(S1);
15    end if
16    allocT'=alloc(t[ins_k], s);  TS = TS - {t};
17     end for
18     return allocT';
```

Algorithm 1. Basic resource allocation

As task t in the business process is selected, there are m such tasks belonged to different instances in the batch need to be allocated, and they are included in the task set TS. Resource set S_1 includes all resources that can be allocated to t (Line 1). Among the resources in S_1, those under-required according to *Rule 2* are classified as set S_2 (Line 2). In Line 3, we check if there exists a task in set TS belonged to the current longest path of the batch, which is returned by function $lpth$ ($Pth_{11},\ldots,Pth_{mk}$), and Pth_i ($1\le i\le m$) is the path set of the ins_i in batch. If it is true, the task on the longest path is allocated in such a way (Lines 4-9): if S_2 contains at least m resources with no less workload than t requires, S_2 is the candidate (Line 5). Otherwise, S_1 is the candidate (Line 6). This task is on the longest path, thus we select a resource from the candidate with minimal execution time to satisfy *Rule 3* by function *getMinTime*() (Lines 5-6). After that we apply function $alloc(t, s)$ to update $allocT$ (to $allocT'$) and some necessary information (maximal workload, resource abundance, support and obligation) in Line 8. For the remaining tasks, they are processed in descending order of the overall time of relative longest path. In Line 10, relative longest path to task t is the longest path t is involved in, which is returned by function $rlpth(t)$. In Line 11, S_3 is a resource set selecting resource from S_1 that after using this resource does not increase time of the longest path. In Line 12, if $S_2\cap S_3\neq\phi$, then according to *Rule 4* a resource of minimal cost is selected by function $getMinCost(S_2\cap S_3)$. Otherwise if $S_2\neq\phi$, then the current longest path must be increased, and we select from S_2 with the one with

minimal time to satisfy *Rule 3* by function *GetMinTime*(S_2) (Line 13). Otherwise, all resources are over-required, and then we select one resource of minimal obligation by function *GetMinOblig*(S_1) according to *Rule 2* (Line 14). Then selected resource is allocated to this task in Line 16.

This algorithm allocates suitable resource to execute m instances in a batch respectively. However, this algorithm does not consider run time features, thus there may exist allocation conflicts due to instance dependency caused by competing resources. Therefore, we handle the allocation conflict immediately after Algorithm 1.

4.2.2 Handling Allocation Conflict

The allocation conflict handling strategy ensures the run time correctness of resource planning. Algorithm 2 will not be applied if the outcome of Algorithm 1 is over the deadline. In the first place, this algorithm checks the allocation conflicts in the order that tasks in a batch are executed. Once conflicts are observed, the conflicts are handled directly. Each checked task may have several conflicts with other tasks, and we will firstly determine the sequence of the conflicts to be handled. For each allocation conflict to be handled, we can reallocate one task to another suitable resource, or make the conflicting tasks executed in a sequential order. In Algorithm 2, we analyse the solutions and their impact on overall performance to select a proper one based on the heuristic rules.

Task set *TS* includes all the tasks for allocation (Line 1).The overall structure bp is n instances executed in parallel, and each ins_i ($1 \le i \le n$) has a process structure p_i (Line 2). Firstly, we check all tasks in *TS* for the run time conflicts in the sequence they start. When a task t_0 is checked and there are conflicts in which t_0 is involved, those detected conflicts are handled in Lines 5-32. Task set *Tc* (Line 5) includes all tasks conflict with t_0. If a task in *Tc* can be reallocated in such a way that both time and cost are reduced, reallocation will be applied (Lines 9-12). Among the resources that can reduce both time and cost, the one of minimal time is selected for reallocation when the task is on longest path according to *Rule 3*, while the most economical one is chosen if not on longest path according to *Rule 4*. In Lines 15-34, the remaining conflicts are handled in the decreasing order of the relative longest path that tasks in *Tc* are involved in. Both reallocation and task/instance delay can be applied. The solution that is optimal in the overall perspective will be selected.

Lines 16-25 deal with conflicts in such case that one of the two conflicting tasks is on the longest path. Reallocation may reduce time but increase cost. However if we choose task delay time cannot be reduced. bp_1 and bp_2 are two operational choices of task delay in arbitrary order (Line 17), t_1 and t_2 are their corresponding overall time of all instances accordingly, and t_{min} denotes the minimal overal time if we choose task delay (Line 18). Our first option is reallocation to reduce the longest path of bp according to *Rule 3*. If tsk_l, which is the longer of the two conflicting tasks, is on the longest path, then our first option is to reallocate on tsk_l and hence reduce the critical path. In Line 20, function *TimeReduceRes*(tsk, t_{min}) returns the resource set that through such reallocation on tsk overall time is shorter than task delay. When the resource set S_1 returned by *TimeReduceRes*(tsk_l, t_{min}) is not empty, we select the resource in set that can mostly reduce the time (Line 21). When S_2 is empty, we reallocate the other task tsk_s to reduce time. If resource set S_3 returned by *TimeReduceRes*(tsk_s, t_{min}) is not empty, we choose the one of minimal cost and do not incur

Input:	n	-	number of instances
	S	-	set of resources that are available
	$allocT$	-	resource planning before allocation on t
Output:	$allocT'$	-	resource planning after allocation on t
	t_b	-	execution time of the batch

```
1  TS={T1[ins1]+…+Tm[insm]};
2  bp=parallel(p1, p2,…, pm); allocT'=allocT;
3  while(TS ≠ ϕ)
4    select task t0: first task to start in TS in batch execution;
5    Tc={t | t∈ TS has time overlap with t0 and they use the same resource};
6    for each t∈ Tc: (t, s)∈ allocT
7      S1={s' | time(t, r(s')<time(t, r(s) ∧ cost(t, r(s'))<cost(t, r(s))};
8      if S ≠ ϕ
9        if t∈ lpth(Pth11,…,Pthmk) then  s'=getMinTime(S1);
10         else  s'=getMinCost(S1);
11           end if
12       allocT'=alloc(t, s'); Tc=Tc-{t};
13     end if
14   end for
15   for each t∈ Tc in descending order of rlp(t)
16     if rlpth(t0)>rlpth(t) then tskl= t0, tsks= t; else tskl= t; tsks= t0; end if
17     bp1=delay(t→t0, bp); bp2=delay(t0→t, bp);
18     t1 =lpth(bp1).tm; t2 =lpth(bp2).tm; tmin=min(t1, t2);
19     if tskl ∈ lpth(bp, allocT')  then
20       S2=TimeReduceRes(tskl, tmin), S3= TimeReduceRes(tsks, tmin);
21       if S2 ≠ ϕ then s=getMinTime(S2); allocT'=alloc(tl, s);
22         else if S3 ≠ ϕ then s=getMinCost(S3); allocT' =alloc(ts, s);
23         else if t1< t2 then bp= bp1 ;  break;
24         else  bp= bp2;
25       end if
26     else
27       allocT1=bestReallocte(p', allocT'); t3= lpth(bp, allocT1).tm;
28       if t3=min(t1, t2, t3)  then  allocT'= allocT1;
29         else if t1=min(t1, t2, t3)  then  bp= bp1;  break;
30         else  bp= bp2;
31       end if
32     end if
33     Tc=Tc-{t};
34   end for
35    if(Tc=ϕ)  then TS=TS-{t0};  end if
36 end while
37 output allocT', tb =lpth(p',allocT').tm;
```

Algorithm 2. Conflict handling algorithm

new conflict with previous tasks (Line 22). However when both of tasks cannot be properly reallocated, we use task delay to handle the conflict (Lines 23-24). Two tasks can be executed in arbitrary order. We compare the two sequences of execution, and the pattern leading to less value of the longest path is chosen. Lines 28-31 deal with conflict in cases none of the tasks are on the longest path. Function *bestReallocate*() returns updated allocation table after proper reallocation. Such reallocation firstly ensures minimal overall time and secondly minimal cost if total time is same. According to *Rule 4*, when Reallocation results in shorter longest path than task delay, it is performed in most economical way (Line 28). Otherwise, we compare two task delay pattern, and the one of less value of longest path is applied (Lines 29-30). When all the conflicts related with t_0 is handled, we check and handle conflicts for the next one task of *TS*. This procedure continues until end node is reached. The output of Algorithm 2 is a new planning scheme for the n instances without run time execution error and the total execution time t_b (Line 37). If $t_b < t_D$, such allocation solution is valid because time constraint is satisfied.

In summary, in the first step we plan the resources for build-time optimisation based on a set of heuristic rules. The conflict handling guarantees it is run time conflict free in the second step.

4.3 Batch Based Resource Planning

In this section, we discuss the strategy to allocate resources for batches of instances, where the proper size of a batch is to be calculated first.

4.3.1 Pre-analysis

In this part, we conduct an analysis on available resources, business requirements and process structures to sketch out a preliminary batch pattern for the resource planning.

When each batch contains m instances, then the total batch number $b=n/m$, and each batch requires time t_b to execute. Given that overall time cannot exceed the deadline, we can conclude formula (1) for pre-allocation planning as listed below. As resources tend to be periodically available in practice, such as work shifts for staff, we choose to process instances in batches. To use resources rationally, we attempt to set more batches while each batch contains fewer instances. Therefore, resources can be scheduled easily with less conflict. If each task in the business process uses the most efficient resource and accordingly the longest path of the process structure is lp, then the processing time of a batch has a lower bound $t_{lowerb} = \sum_{t \in lp} min(\{time(t, s) \mid capable(t, s)\})$, because every single instance requires a duration no less than t_{lowerb} for execution. With formula (1) and the range of $t_{lowerb} \leq t_b$, we can compute the minimal number of instance m_0, and conclude formula (2).

$$t_b \times n / m \leq t_D \tag{1}$$

$$\lceil t_{lowerb} \times n / t_D \rceil = m_0 \leq m \tag{2}$$

We use $m_0 = \lceil t_{lowerb} \times n / t_D \rceil$ and $b_0 = n/m_0$ as the initial batch pattern for resource planning in following sections. Previously, lower bound t_{lowerb} is obtained with the assumption that each batch contains only one instance. If each batch changes to

contain m_0 instances, t_{lowerb} will be updated accordingly. Given task $t \in T$, different instances in a batch may perform t with different resources. Assume the resource set capable of executing t is $S=\{s_1, s_2, \ldots\}$, and for each resource s_i, it may serve for k instances on task t in the batch according to its workload limit. Then s_i can be expressed as $s_{i1}, s_{i2}, \ldots, s_{ik}$. Each of them is for one of k instances in the batch. Therefore, there is a corresponding resource set:

$$A(t) = \{ s_{ij} \mid s_i \in S \wedge 1 \le j \le \lfloor m\_load(s_i) / (b_0 \times time(t, s_i)) \rfloor \}$$

where s_{ij} is a resource with the exactly workload to execute task t of one instance in the batch, and $j \le \lfloor m\_load(s_i) / (b_0 \times time(t, s_i)) \rfloor$ because that is the maximal number of instances that s_i can serve for t in a batch. Then, we start to update lower bound of batch time t_b and update the batch pattern if formula (1) cannot be satisfied, until a suitable batch pattern is selected for resource planning. Details of this procedure are listed as below:

(1) Resource checking. For each $t \in T$, if $|A(t)| < m_0$, i.e., there is no sufficient resource to serve task t of all instances based on this batch pattern, the pattern has to be updated and m_0 is to be increased in (3).

(2) Lower bound update. Otherwise, we firstly refine the t_{lowerb} based on this batch pattern (m_0 instances). For each $t \in T$, there are m_0 instances to allocate, and different instances may use different resources to perform task t. From $A(t)$, we choose a subset $B(t)=\{ s_{ij} \mid s_{ij} \in A(t)$ is one of the top m_0 efficient resources $\}$, hence using resources in $B(t)$ can provide the best result in aspect of time. Based on $B(t)$, we will refine the lower bound of batch time t_{lowerb} based on the following computations:

(a) Let p (from v_s to v_t) be a path in the process and p include tasks $T = \{ t_1, t_2, ...\}$, for $\forall t \in T$ we use a resource from $B(t)$ to execute t. Assume the processing time of each instance i ($1 \le i \le m_0$) the processing time is $pt(i)$, then the total time to execute m_0 instances is $\sum_{i=1}^{m_0} pt(i) = \sum_{t \in p} \sum_{s_{ij} \in B(t)} time(t, s_i)$. As we know, the time required for executing p for the whole batch is determined by the maximum path among m_0 instances, which is no less than the average of total time for executing p for all m_0 instances. Therefore we choose the lower bound for processing p as $TM(p) = \sum_{t \in p} \sum_{s_{ij} \in B(t)} time(t, s_i) / m_0$;

(b) According to the process structure, there may be multiple paths from v_s to v_t. Each path has its own lower bound according to (a). The lower bound of executing time for the whole process is the maximal value of $TM(p)$ for all paths. Therefore, given the paths $P = \{p_1, p_2, \ldots\}$ of the business process, the lower bound in this batch pattern is refined to:

$$t_{lowerb} = max(TM(p_1), TM(p_2), \ldots)$$

After the lower bound of batch time t_{lowerb} is updated, we check $t_{lowerb} \times \lceil n / m_0 \rceil < t_D$ again. If it is satisfied, this pattern is allocable and we move on to section 4.3 to do resource planning based on this pattern. Otherwise, we update the batch pattern in (3).

(3) Batch pattern update. In this pattern (m_0 instances), the total batch number is $b_0 = \lceil n / m_0 \rceil$. When we increase m_0, if b does not reduce accordingly, then the total time

is definitely increased. Therefore in order to reduce total processing time, m_0 must be updated to such a value that b is reduced as a result. Hence, the new batch pattern is $m_0 = \lceil n / (b_0+1) \rceil$, and we go back to (1) with this new pattern.

The procedure from (1) to (3) continues until a batch pattern can be found. Then in the next step, we investigate how to plan the available resources for the given batch pattern to achieve the business goals.

4.3.2 Resource Allocation for Batches of Instances

Through the previous computation we selected a batch pattern. For each batch, we use the holistic algorithm discussed in Section 4.2 to plan resources for m instances within a batch. If the total time of returned resource plan exceeds the deadline, we adjust the batch pattern and replan as below:

In order to find a new batch pattern for reducing the overall time, we have to decrease the number of batches b by 1 to b', accordingly increase the number of instances within a batch from m to m' ($m' > m$), to make more instances processed in parallel. Consequently, the batch time will be increased from t_b to t_b'. With more instances to be processed in a batch, we know that $t_b' \geq t_b$. We know that the basic requirement is $t_b \times n/m \leq t_D$, and we can deduce that $m' > n \times t_b / t_D$. Hence based on the previous allocation result, we predict the new number of instances $m = \lceil n \times t_b / t_D \rceil$, and then based on this pattern we reuse the holistic strategy to plan for the new batch. This procedure continues until either we get the result that satisfies time constraint or b gets decreased to 1 which means that we cannot find an allocation plan for even a single batch.

4.4 Discussion

Both holistic and batched strategies can achieve a common goal, i.e., a lower cost by applying the set of heuristic rules. However, the holistic approach may suffer in a skewed resource allocation while the batched approach may not obtain a shorter deadline compared with the holistic approach. Due to the availability pattern of resources, an organisation always has a minimal batch number B for better arranging workload for resources. This implies that a batched solution with $b < B$ may not be an attractive solution to the organisation. However, a deadline may be required by customers. Therefore, it is necessary to balance the deadline and the number of batches.

If the batched approach fails to find a batched solution with $b \geq B$, the organisation may negotiate with customers to avoid losing the deal. Our strategy can provide suggestive information to help negotiation: (1) Extend the deadline. If the deadline is not strict, we calculate the total time t_a of the batched solution on $b = B$, and relax t_D to t_a if customers agree. (2) Reduce number of instances. Assume n_a ($n_a < n$) is the number of instances finished within the deadline when we apply the batched strategy on $b = B$. If the number of instance is negotiable, it can be relaxed to n_a. (3) Suggested batch number. We go on process batched strategy starting from $b_a = B-1$, and find a batch solution with batch number b_a. The organisation can judge if it is worthy to accept the order by using resources in this resource plan.

5 Experiment

In this section, we present an experimental study of the resource planning strategies proposed in this paper. Through the implemental data, we investigate and compare the performance of the two strategies with different volumes of instances, batch patterns and business process scales. Also, we also discuss how to apply the strategies to help organisations rationally use their resources.

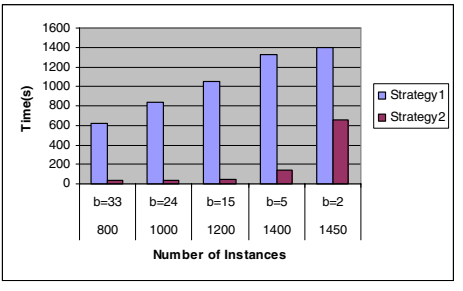

Fig. 3. Performance of two strategies

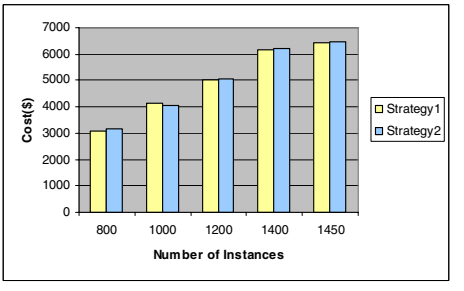

Fig. 4. Cost of two strategies

Firstly, we test the performance of the holistic planning strategy (as Strategy 1) and the batched planning strategy (as Strategy 2) with different amounts of instances. Extended from the motivating example discussed in Section 2, the test settings include a 30-task business process, a set of resources, and a requested completion time of 140 hours. In the same environment, we apply both strategies to plan resources for 800, 1000, 1200, 1400 and 1450 instances. For each test case, the schedule time of two strategies is compared in Figure 3, and the cost compared in Figure 4. Costs of two strategies are similar according to Figure 4, for example in producing 1000 instances the cost of strategy 1 is $4150, and the cost of strategy 2 is $4050. But strategy 2 has much shorter schedule time than that of holistic strategy according to Figure 3, for example, the schedule time on 1000 instances is 836s and 51s for two strategies respectively. We notice that the more batches we use, the more time can be saved by batched strategy. Suppose the organisation requires that the assigned 1450 instances to be executed in at least 5 batches, but the batched strategy can merely support 2 batches as shown in Figure 3. To cover the gap, the organisation may ask for either a relaxation on number of instances or an extension to the deadline. Figure 3 shows 5 batches can be supported with 1400 instances, and therefore we can recommend the organisation to negotiate with customer for relaxing number of instance to 1400.

Also, we compare the performance of planning with different batch patterns in a fixed environment. With 1450 assigned instances, Figure 5 lists the minimal time and overall cost to produce required instances based on 1, 2, 5, 10 and 20 batches. It shows that the total costs for different patterns are similar. However when the number of batches increases, the total time is likely to rise as compensation. When the deadline is 140 hours, only 2 batches can be supported according to Figure 5, although the organisation requires 5. Figure 5 shows it will take 145 hours to produce all instances if they are partitioned into 5 batches. Thus, we can recommend the organisation to negotiate with customers for extending the deadline to 145 hours.

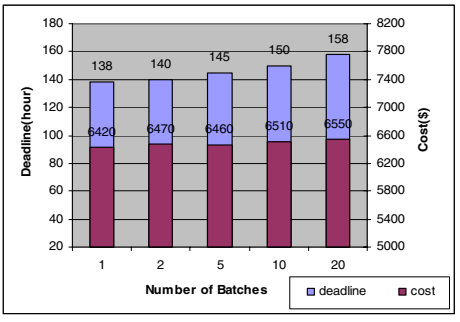

Fig. 5. Batch based resource planning

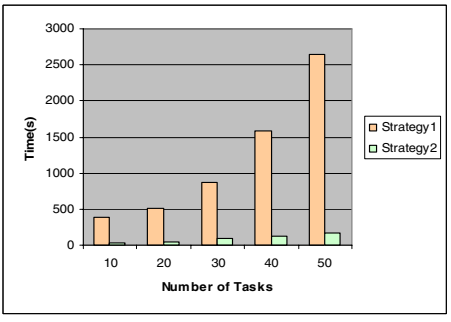

Fig. 6. Performance on different task scale

System performance is a crucial issue in some applications, especially when re-planning is frequently required because of the dynamic environment or requirements. Figure 6 shows the schedule time for 1000 instances under different task scales. As the figure shows, the schedule time increases dramatically when the number of tasks increases, and it is time consuming to plan resources for business processes containing a large number of tasks. While in comparison, the batched strategy is more efficient, and therefore is preferable and practical when the task scale rises.

According to these experiments, we see that the batch based strategy is of better performance and more rational in allocating resources while it sometimes may require more time as a compensation. By estimating the potential influences, our approach provides the organisation with the information on adjusting batch patterns for optimising resource utilisation.

6 Related Work and Discussion

Resource planning is a research topic related to task/workflow scheduling and resource allocation, which seeks an optimised execution sequence for a set of tasks to achieve certain goals. This procedure is dependent on which resources are allocated to execute the tasks. The previous related work can be generally classified into two categories: resource oriented scheduling and process oriented one. In the resource oriented scheduling such as [2, 12, 15], tasks are scheduled at the run time to achieve full utilisation of resources, and there is no dependency between these tasks. In contrast, the process oriented scheduling deals with tasks in the workflow where tasks have dependency with each other. In [14], Yu has proposed a genetic algorithm for scheduling scientific workflows for utility Grid applications by minimising the budget constraint while meeting user's deadline constraint. In his strategy, workflow is partitioned and deadline is assigned to the partitions for resource planning, but resource workload is not considered so resources are unlimitedly used. In [10], Topcuoglu, Hariri and Wu have proposed an Earliest-Finished-Time (HEFT) algorithm to reduce processing time, and HEFT has been demonstrated to be effective in the ASKALON project in [11]. While most work in this area of workflow scheduling focused on the temporal and causality constraints, [9] proposed a workflow scheduling framework under both temporal constraints and resource constraints, but did not discuss how to manage and allocate resource in workflow. Work [1] presented an adaptive scheduling algorithm that finds

a suitable execution sequence for workflow tasks by additionally considering resource allocation constraint and dynamic topology changes. This work was applied in the embedded and wireless network systems. All the above approaches were proposed from the run time perspective, however sometimes we hope to manage resources at the build time to guarantee some requirements can be satisfied. In [6], Eden, Panagos and Rabinovich developed the technique to check time constraints at process build time and to enforce these constraints at run-time. However, this paper did not investigate the inter-instance dependency. In [13], we proposed a method to improve performance of business processes by integrating build-time resource allocation with business process structural improvement. However, the focus of this paper is to investigate the impact between resource allocation and process structural improvement, only intra-instance dependencies have been considered.

In contrast to the previous works, this paper focused on the planning of resources for massive number of instances to meet process requirements and rational utilisation of resources before execution. Particularly, our approach highlighted the influences caused by the process structure and the concurrency of process instances for resource planning, and our strategy can enable resources with balanced workload distribution to cater for their rationalities.

7 Conclusion

In this paper, we investigated how to plan resources for a business process optimally, in aspect of meeting process requirements and rational utilisation of resources. Also, our approach can provide information to organisation for decision making and negotiation with customer in order to better use resources. When a massive number of instances are given, both inter and intra instance dependencies are considered in the planning of each batch of instances to be executed in parallel. As the problem is computationally hard [14], a set of heuristic rules were designed, and two strategies based on these heuristic rules were proposed and compared.

In the future, we plan to improve our planning strategy for supporting complex resource patterns.

Acknowledgement

The research work reported in this paper is supported by Australian Research Council under Linkage Grant LP0669660.

References

1. Avanes, A., Freytag, J.C.: Adaptive Workflow Scheduling Under Resource Allocation Constraints and Network Dynamics. Journal of PVLDB 1, 1631–1637 (2008)
2. Braun, T.D., Siegel, H.J., Beck, N., Bölöni, L., Maheswaran, M., Reuther, A.I., Robertson, J.P., Theys, M.D., Yao, B., Hensgen, D.A., Freund, R.F.: A Comparison of Eleven Static Heuristics for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems. Journal of Parallel Distributed Computing 61, 810–837 (2001)

3. Cardoso, J., Sheth, A.P., Miller, J.A., Arnold, J., Kochut, K.: Quality of Service for Workflows and web Service Processes. Web Semantics 1, 281–308 (2004)
4. Etoundi, R.A., Ndjodo, M.F.: Feature-oriented Workflow Modelling Based On Enterprise Human Resource Planning. Business Process Management Journal 12, 608–621 (2006)
5. Huang, Y.-N., Shan, M.-C.: Policies In A Resource Manager of Workflow Systems: Modelling, Enforcement and Management. In: Proceedings of the 15th International Conference on Data Engineering, p. 104. IEEE Computer Society, Los Alamitos (1999)
6. Eden, J., Panagos, E., Rabinovich, M.: Time Constraints in Workflow Systems. In: Jarke, M., Oberweis, A. (eds.) CAiSE 1999. LNCS, vol. 1626, pp. 286–300. Springer, Heidelberg (1999)
7. Russell, N., van der Aalst, W.M.P., Hofstede, A.H.M.t., Edmond, D.: Workflow Resource Patterns: Identification, Representation and Tool Support. In: Pastor, Ó., Falcão e Cunha, J. (eds.) CAiSE 2005. LNCS, vol. 3520, pp. 216–232. Springer, Heidelberg (2005)
8. Russell, N., van der Aalst, W.M.P.: Work Distribution and Resource Management in BPEL4People: Capabilities and Opportunities. In: Bellahsène, Z., Léonard, M. (eds.) CAiSE 2008. LNCS, vol. 5074, pp. 94–108. Springer, Heidelberg (2008)
9. Senkul, P., Toroslu, I.H.: An Architecture for Workflow Scheduling Under Resource Allocation Constraints. Information System 30, 399–422 (2004)
10. 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, 260–274 (2002)
11. Wieczorek, M., Prodan, R., Fahringer, T.: Scheduling of Scientific Workflows in The ASKALON Grid Environment. Journal of SIGMOD Record 34, 56–62 (2005)
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, 824–834 (2004)
13. Xu, J., Liu, C., Zhao, X.: Resource Allocation vs. Business Process Improvement: How They Impact on Each Other. In: Dumas, M., Reichert, M., Shan, M.-C. (eds.) BPM 2008. LNCS, vol. 5240, pp. 228–243. Springer, Heidelberg (2008)
14. Yu, J., Buyya, R., Tham, C.-K.: Cost-Based Scheduling of Scientific Workflow Application on Utility Grids. In: International Conference on e-Science and Grid Technologies, Melbourne, Australia, pp. 140–147 (2005)
15. Zomaya, A.Y., Teh, Y.-H.: Observations on Using Genetic Algorithms For Dynamic Load-balancing. IEEE Transactions on Parallel and Distributed Systems 12, 899–911 (2001)