Business process scheduling with resource availability constraints

Business Process Scheduling with Resource Availability Constraints

Jiajie Xu[1], Chengfei Liu[1], Xiaohui Zhao[2], and Sira Yongchareon[1]

[1] Faculty of Information and Communication Technologies
Swinburne University of Technology, Australia
{jxu,cliu,syongchareon}@groupwise.swin.edu.au
[2] Department of Industrial Engineering and Innovation Sciences
Eindhoven University of Technology, The Netherlands
x.zhao@tue.nl

Abstract. Resources tend to follow certain availability patterns, due to the maintenance cycles, work shifts, etc. Such availability patterns heavily influence the efficiency and effectiveness of enterprise process scheduling. Most existing process scheduling and resource management approaches focus on process structure and resource utilisation, yet neglect the resource availability constraints. In this paper, we investigate how to plan the business process instances scheduling in accordance with resource availability patterns, so that enterprise resources can be rationally and sufficiently used. Three planning strategies are proposed to maximise the process instance throughput using different criteria.

1 Introduction

Resource planning is a classical issue for enterprise operation management, whilst the global financial crisis further urges enterprises to seek optimal resource utilisation for cost effectiveness. Planning resources for a large number of process instances before execution guarantees the business requirements to be satisfied and benefits enterprises for intelligent marketing-decision support, and such planning for enterprise operations is always subject to resource capacity and availability [3, 5]. However, most existing works [1, 3, 5, 6] focus on handling this problem at the run time. This paper incorporates the resource availability constraints and process structures into build time process scheduling, and devises a comprehensive framework for maximising process instance throughput with a set of strategies.

Apart from the deadline constraint in classical resource management for business processes, high process instance concurrency is highly sought after in most application scenarios. This is because that the concurrently running instances lead to high instance throughput and efficient resource utilisation for resource management and process scheduling. However, it is not realistic to increase the process instance concurrency by pushing all resources together to serve instances, because resources themselves are only available in certain time periods in practice, e.g., a worker is not supposed to work after hours. Thus, the process scheduling should take into account

R. Meersman et al. (Eds.): OTM 2010, Part I, LNCS 6426, pp. 419–427, 2010.

the resource availability, resource capability, process task dependency, instance deadline, as well as the inter-influence among these factors.

As a classical enterprise management topic, process scheduling has attracted a lot of research efforts. Some algorithms in this area using different heuristics (such as GA, SA, etc.) are compared in work [12]. However, most of these approaches assume that the availability information of resources can only be known at run time. In our previous work [4], we assume that the availability of resources can be tailored to tasks. To the best of our knowledge, so far no work has addressed the influence of resource availability on scheduling large scale of process instances at the build time, which is an important issue. To tackle this problem and further improve the process scheduling performance, in this paper we propose a comprehensive scheduling framework in this paper to deal with the scheduling of heterogeneous process instances under resource availability constraints at build time, so as to maximise the number of instances to be successfully scheduled in different criteria.

The rest of this paper is organised as follows: Section 2 introduces a model including the key notions for process instance scheduling, and formally defines the problem we intend to address. Three strategies for process scheduling are proposed in Section 3. Section 4 reviews the related work and discusses the advantages of our approach. Lastly, concluding remarks and future works are given in Section 5.

2 Model and Problem Definition

In this section, we first present a model comprising resources, tasks, process instances and resource allocation. Based on the definitions, the problem we intend to investigate in this paper is formally defined.

Definition 1. (Resource) Resource is used to perform tasks in business processes. A resource satisfies time availability constraints defined by a sequence of available time periods.

Definition 2. (Task) A task in a business process can be executed by a set of capable resources. For each resource capable of executing a task, the time required for executing this task may be different.

Definition 3. (Resource Slot) Resource slot measures a time duration (within the available time periods) of a particular resource. The resource of this slot may be *available* for use in a time duration (e.g., from start time *st* to end time *et*), or has been *assigned* to perform a task for a business process.

Definition 4. (Process Instance) A business process instance *ins* has a task set T and an edge set E, which defines the dependency between tasks. An edge $e(t_i, t_j) \in E$ indicates that t_j can only start after t_i finishes. The instance *ins* is required to be executed within a range $[D_s(ins), D_f(ins)]$, where $D_s(ins)$ stands for the earliest start time and $D_f(ins)$ the latest finishing time (deadline).

Definition 5. (Allocation) Allocation is the resource assignment to a task t of a process instance *ins*. An allocation <*ins*, *t*, *r*, *st*, *et*> indicates that an available slot *slt* of resource *r* is able to be allocated for executing task *t* of *ins* from time *st* to *et*. Such allocation may result in adjustment on those slots of *r* before or after *slt*.

Problem Statement
Given a set of resources *R*, and a number of instances *I*, the problem is to find a scheduling scheme *S* (which consists of a set of allocations) to schedule instances in *I* using resources in *R* such that maximal number of instances can be scheduled. During scheduling process, four constraints must be satisfied: **(C1)** Time constraint of instance - each instance *ins* must be executed between a required duration within the time range $[D_s(ins), D_f(ins)]$; **(C2)** Availability constraint of resource – each schedule from a task *t* to resource *r* must be in such a time duration that *r* is available; **(C3)** Process structural constraints – if there is an edge from task t_1 to task t_2 in the business process, then t_2 cannot start until t_1 finishes; **(C4)** Conflict free – at one time, one resource can only be used for executing one task.

3 Scheduling Strategies

Finding the optimal solution to the above defined problem is computationally hard. As such, near optimal strategies based on reasonable heuristic rules are sought after. To reduce the calculation for each allocation of a resource to a task, we first apply a so-called optimistic pre-allocation scheme to all instances by only satisfying constraints C1, C2 and C3 while ignoring constraint C4 between different instances (C4 on the single instance level is satisfied) at the beginning. For each allocation of the optimistic scheme, the most efficient resource is used. This pre-allocation sets a basis for the following process scheduling because after the pre-allocation, the time gap denoted as $g(ins)$ for each instance *ins* can be easily calculated as the difference between $D_f(ins)$ (the deadline of *ins*) and the finishing time of *ins* in the optimistic allocation. It is obvious that if $g(ins)$ is negative, then *ins* cannot be scheduled because the most efficient resources are used. We also know that this initial time gap is obtained by allowing conflict resource allocation. The scheduling process then is to re-allocate tasks such that constraint C4 can also be satisfied.

The time gap is an important indicator for the priority of instance allocation. An instance with smaller time gap is considered to be more "dangerous". Our first scheduling strategy is based on the rule to iteratively save the most "dangerous" instance *ins* which owns the minimum value in $g(ins)$. This strategy operates in a depth first manner and falls into the category of greedy algorithm as only local optimisation is applied to one instance (i.e., the most dangerous one) at a time. Sometimes, this strategy is practical because it guarantees that an instance is scheduled once it is processed. However, allowing one instance to go through may be at the cost of sacrificing other instances.

Given the limitation of the first strategy which is local optimisation based, we propose some holistic strategies that are global optimisation in nature. A holistic approach operates more or less in breadth first manner. Instead of scheduling one instance at a time, it allocates resource to a task of an instance at a time based on a particular rule. So this approach allows balanced scheduling and thus gives chances to all instances. It focuses more on dependencies among instances. Compared with the greedy strategy, instances scheduled in this approach tend to success or fail together. In most cases, the balanced scheduling approach enables more instances to be schedulable. However when resource is extremely limited, a holistic approach may cause

more instances non-schedulable compared with the greedy strategy. In the holistic approach, we would like to allocate resource to the current task of an instance at a time. As to which instance to select, we design two strategies. The first strategy is to select the instance based on several heuristic rules about that instance, including whether it owns the minimum time gap. We call the current task in such an instance the most urgent task. This strategy is different from the greedy strategy because after the allocation, the time gap for other instances will be adjusted to use the remaining available resources in an optimistic way, and the instance with minimum time gap may be changed to another instance after the adjustment. However, it does bear similarity with the greedy strategy so we call it a dynamic local optimization strategy. The second strategy is dynamic global optimization. The holistic rules designed for this strategy are based on the penalty calculated from all instances, including the summation of the gap increases of all instances. This strategy chooses the instance with the minimum penalty to schedule.

Table 1. Scheduling strategies

Strategy	**Priority**
DM – Depth first/Min gap	Most dangerous instance (instance with the minimum time gap)
BL – Breadth first/dynamic Local optimisation	Most urgent task (mainly determined by the task in the most dangerous instance)
BG – Breadth first/dynamic Global optimisation	Least penalised task (the allocation of the task that results in the minimum time gap increases of all instances and other factors)

Table 1 summarises the three strategies introduced. The details of these strategies will be introduced in 3.1, 3.2, and 3.3, respectively. Also, a discussion about the comparison among three proposed strategies will be conducted in 3.4.

3.1 DM Scheduling Strategy - Depth First/Min Gap

This strategy is based on a greedy algorithm for process scheduling. Resource allocation is applied for the most dangerous instance one by one. The allocating sequence is in descending order of instance time gap. Given a set of instances, the instance *ins* with the minimal time gap has the least room to delay. If the resources are allocated to other instances first, this instance is most likely affected, i.e., re-allocation even using the remaining best available resources may cause it to exceed its deadline ($D_f(ins)$). Therefore the capable resources, we set the highest priority to this instance for occupying most efficient ones. The algorithm proceeds as follows. We first pre-allocate all instances in optimistic way such that each instance has a time gap initially. Then we select the instance with minimum time gap and keep the optimistic allocations for the instance. Once an instance is scheduled, due to the allocated resources to the instance, we have to adjust those affected allocations of other instances by using the remaining resources. After that, we continue to select the instance with minimum time gap among the un-scheduled instances. The selection and scheduling of the instance with

the minimum time gap and allocation adjustment are repeated until none of the remaining instances are schedulable. The algorithm is conducted as the following steps.

1. Initial optimistic allocation. Firstly, we pre-allocate all instances and calculate their time gap to deadline before allocation, without considering the resource conflict (constraint C3). It provides the best scenarios for all the instances. Given instance set *I* and resource set *R*, we first find the current task *t* (or the task to be scheduled next) of *ins*, and then calculate the minimal end time of *t* with available resources by function. If the end time of *t* exceeds deadline, this instance is dropped out as it is definitely non-schedulable. Otherwise, *t* is allocated with the most efficient resource. This pre-allocation procedure continues until all instances have been processed.

2. Resource allocation. Based on the time gap of un-scheduled instances, resource allocation becomes easy. Basically, we use the criterion of time gap to deadline to evaluate the priority of instances. The less the time gap between the finishing time to deadline, the more dangerous this instance is and hence more priority for allocation. After the most dangerous instance i_s is selected, resource allocation is made on it. When an instance is successfully scheduled, it is removed from the scheduling list.

3. Optimistic allocation adjustment. After resource allocation of instance *ins*, the optimistic allocation of other instances must be adjusted due to the resource availability change. We only need to adjust the optimal allocations of some instances, which use any resource slot occupied by the instance *ins*. The adjustment is made by selecting the most efficient slot from the remaining available resources. For each remaining instance *ins'*, the task set *T* conflicting with previous allocation is generated for possible re-allocation. For each task *t* in *T*, the new finishing time of optimistic allocation is re-calculated. If the updated finishing time is within the deadline, optimistic allocation is re-applied for *t* of *ins'*. Otherwise, this instance is dropped out. Resource allocation and optimistic allocation adjustment are repeated until all instances are processed, i.e., either scheduled or dropped out.

Above three steps continue until no remaining instance is schedulable. Success rate is finally returned together with the process schedules.

3.2 BL Scheduling Strategy - Breadth First / Dynamic Local Optimisation

The BL strategy attempts to plan resources for instances in a holistic way yet using dynamic local optimisation. We know that the DM strategy allocates resources to one instance at a time. The BL strategy is different in that resource allocation is made to one task of one instance at a time. In this way, it balances all instances by giving them the same chance for occupying resources. Among those current tasks competing for a resource, allocation will be made to schedule the most urgent task, and task urgency is evaluated by certain criteria on the instance this task belongs to. Specifically, in each round we select a next available resource slot for allocation, and this resource is supposed to be used for the most urgent task according to a set of heuristics for local optimisation. Initially, the result of pre-allocation is used as input for BL strategy. Then instance scheduling is conducted in an iterative approach using the following three steps until no remaining instance is schedulable.

1. Selecting resource and generating candidate task set. First we select a resource slot. We choose the resource slot *slt* that is the one first in use among all available resource slots. This optimistically allocated resource slot may be conflicting with more than one current task of different instances. Obviously only one of them can be allocated with the slot (i.e., satisfying constraint C3 for conflict free allocation). In this step, we first find the conflicting task set on this resource slot. Assume task t is the earliest one using *slt*, the time period *tp* of the allocation on t is derived. The conflicting task set T includes all un-scheduled tasks using resource slot *slt* during a period overlapping with *tp*. Afterwards, we select the most urgent task from T based on a set of heuristics.

2. Resource allocation. Given resource slot *slt* and conflicting task set T from 3.3.1, this step is to choose the most urgent task. In the BL strategy, the urgency of a task is determined by a set of heuristic rules about the instance containing the task.

> ***Rule 1.*** The priority of a task t for allocation is influenced by the time gap of the instance it belongs to. The smaller value of time gap, the higher priority of t for allocation as the instance is more likely to exceed the deadline otherwise.
>
> ***Rule 2.*** The number of alternative resource slots to resource slot *slt* influences the priority of a task t. If t has many alternative resource slots (capable resources to which t can be re-allocated), t has abundant allocation choices hence may not have the priority of to be allocated using the slot *slt*.
>
> ***Rule 3.*** If a task t is not allocated and there is no alternative resource slot for t, time gap of the instance *ins* that t belongs to may reduce from $g(ins)$ to $g(ins)'$. An instance with a higher ratio = $g(ins)' / g(ins)$ is more likely to exceed deadline, hence has more priority to be scheduled immediately.

For each task t of instance *ins* in conflicting task set T, we generate the alternative resource set S_a according to Rule 2. Based on the heuristic rules, the priority of each task for requesting this resource is calculated by function $u(t)$. Assume $x = 1 + |S_a|$ and $r = g(ins)' / g(ins)$ for task t, the urgency of this task of being selected is computed as *formula 1*:

$$u(t) = \frac{r}{g(ins) \times \sqrt{x}} \quad (g(ins) \neq 0) \tag{1}$$

In formula 1, the urgency of task t is in reverse proportion to $g(ins)$, because resources tend to save tasks in dangerous instances according to Rule 1. Urgency is also in reverse proportion to $\sqrt{x}$, because the task with more alternative resources can be more likely scheduled by other resources without affecting optimistic allocation (Rule 2). To reduce the effect of x, square root of x is used. In contrast, the urgency of task is in proportion to the ratio r. If a missed allocation of a task to this resource will cause a dramatic decrease of the time gap from $g(ins)$ to $g(ins)'$, this task has a high value of r and is more urgent to be allocated at this time (Rule 3). Among task set T we select and schedule the task t_s with maximum value of $u(t_s)$.

3. Allocation adjustment. After resource allocation in the previous step, the allocations of some instances may be affected and required to change accordingly. Similar to the DM strategy, affected allocations are adjusted. But if an instance becomes unschedulable with remaining resources, its occupied resources are released. After

optimistic pre-allocation, steps from 3.2.1 to 3.2.3 are repeated until no remaining instance is schedulable. Lastly, we try to re-schedule the allocated tasks to save the other instances. Finally, the process scheduling result is returned.

3.3 BG Scheduling Strategy - Breadth First / Dynamic Global Optimisation

The BG strategy uses a different optimisation criterion for holistic process scheduling compared with the BL strategy. Process scheduling is also carried out for one task of one instance in each round, and the instances are scheduled in a balanced way. However, this strategy targets a global optimisation for every allocation. Resources are used to schedule the task with minimal penalty based on all instances rather than a single instance. We propose three heuristic rules, and based on these rules the penalty of each task for allocation can be calculated by a formula. In comparison, this strategy considers more impact among different instances than the previous two strategies. Given a resource slot *slt* and a conflicting task set *T* on *slt*, task priority is determined by the following rules:

Rule 1. When *slt* is allocated to $t \in T$ of an instance, the total time gap increase of all instances influences the task penalty. The more the total time gap increases, the more penalty it will get and hence the less priority this task will be scheduled from the overall perspective.

Rule 2. The task gets lower penalty if it belongs to an instance with less number of un-scheduled tasks. If the instance of a task has fewer un-scheduled tasks, the task has more priority to be scheduled because we are more likely to guarantee that the instance can be finished.

Rule 3. The task gets higher penalty if it results in more instances become un-schedulable. Each allocation is aimed to cause the least number of instances becoming un-schedulable.

Based on the result of pre-allocation, we select the next available resource slot that is first used (minimal start time) in all un-allocated instances, and then generate the task candidate set of the resource slot. The penalty of task candidates are evaluated according to the heuristic rules proposed above, and the resource slot is allocated to the task with minimal penalty from the global view. Given a task *t* of instance *ins*, the penalty $p(t)$ for scheduling this task using resource slot *slt* is calculated as:

$$p(t) = (1 - \frac{1}{2x}) \cdot y^2 \cdot \left(\sum_{i \in I} (g(i)' - g(i)) \right) \tag{2}$$

where x is the number of remaining tasks of *ins* including *t*, y is the number of un-schedulable instance resulted from the allocation of *t*, and $\sum_{i \in I} (g(i)' - g(i))$ is the total time gap increase. The penalty has direct relationship with the total time gap increase of all instances (Rule 1). Also, the penalty is in proportion to x because when an instance is about to finish, we tend to finish it (Rule 2). However, Rule 2 is less dominant so we design $(1 - 1/2x)$ as the coefficient in range [0.5, 1) to restrict its effect. In addition, task penalty is also in proportion to y because the allocation should avoid affecting other instances (Rule 3). We emphasise its effect with y^2. In each round, the task t_s with minimal penalty is selected, and resource slot *slt* is allocated to schedule

this task for global optimisation. Optimistic allocations are updated after task scheduling. Similar to the BL strategy, these three steps continue until no remaining instance is schedulable.

3.4 Discussion

A preliminary experiment is implemented to evaluate the performance of the above three strategies. Experimental results will not be introduced in this paper because of the limitation of space. Through the experiment, we observe that the BG strategy has higher success rate when resource is sufficient and performs worse when resource becomes tighter. This coincides with our early analysis that instances scheduled in this strategy tend to success or fail together. When resources are insufficient, the DM strategy becomes a practical scheduling strategy. This also coincides with our analysis due to the nature of the depth first scheduling. Compared with BG and DM, the BL strategy adopts the method in between and it performs best when the resource sufficiency is in middle of BG (sufficient mode) and DM strategy (skewed mode).

4 Related Work

Workflow scheduling is to investigate resource management while considering the complex task dependencies of workflows. Related work in this area can be classified into two categories. The first category is to allocate suitable resources for workflow instances at run time. In [11], Yu and Buyya developed a genetic approach to solve the deadline constrained scheduling problem. The fitness function combines time fitness and cost fitness. Based on the fitness value, their algorithm searches for a solution which has minimal execution cost with the deadline by two types of mutation operations: the swapping mutation and replacing mutation. YarKhan and Dongarra [10] proposed a solution using simulated annealing to select a suitable size of a set of resources for scheduling ScaLAPACK application in Gird environment. Work [6] presented architecture of workflow scheduling under the resource constraints. In work [7], a novel framework of resource patterns for workflow resource management is proposed by Senkul and Toroslu. In contrast, the second category is to plan resources for the workflow instances at the build time. In this category, more instance dependency information is assumed to be available and therefore can be explored for resource planning at build time. In work [9], two strategies are proposed to plan resources for a massive number of process instances before execution, in order to meet the deadline and minimise total cost. Also, resource planning for service oriented workflows is investigated in [2]. It firstly introduces both the required architecture for resource planning and workload prediction, and then presents the optimization approaches and heuristics for solving the resource planning with low computational overhead. Work [8] incorporates process structural improvement into resource allocation to optimize the process execution in meeting certain requirements.

As far as we know, none of them considers the time availability patterns of resources, which is a crucial issue for rational use of resource in practice. Compared with these works, this paper focuses on the scheduling of process instances with resource availability constraints before execution. In particular, our approach considers

the resource work shift constraints and supports both homogeneous and heterogeneous structured process instances.

5 Conclusion

In this paper, we tackled the problem of business process instance scheduling satisfying certain availability constraints. We investigated how to allocate resources for process instances before execution such that the success rate of scheduling is maximised. As the problem is computationally hard, we explored a set of heuristic rules and proposed one greedy algorithm and two holistic algorithms.

Acknowledgement. The research work reported in this paper is supported by Australian Research Council under Linkage Grant LP0990393.

References

1. Avanes, A., Freytag, J.C.: Adaptive workflow scheduling under resource allocation constraints and network dynamics. Proceedings of VLDB 1(2), 1631–1637 (2008)
2. Eckert, J., Ertogrul, D., Miede, A., Repp, N.: Ralf Steinmetz: Resource Planning Heuristics for Service-Oriented Workflows. In: Web Intelligence, pp. 591–597 (2008)
3. Iosup, A., Jan, M., Sonmez, O., Epema, D.H.J.: On the dynamic resource availability in grids. In: Proceedings of GRID, pp. 26–33 (2007)
4. Jensen, M.T.: Generating robust and flexible job shop schedules using genetic algorithms. IEEE Trans. Evolutionary Computation 7(3), 275–288 (2003)
5. Rood, B., Lewis, M.J.: Scheduling on the Grid via multi-state resource availability prediction. In: Proceedings of GRID, pp. 126–135 (2008)
6. Russell, N., van der Aalst, W.M.P., ter Hofstede, A.H.M.: 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)
7. Senkul, P., Toroslu, I.K.: An architecture for workflow scheduling under resource allocation constraints. Inf. Syst. 30(5), 399–422 (2005)
8. 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)
9. Xu, J., Liu, C., Zhao, X.: Resource planning for massive number of process instances. In: Proceedings of CoopIS, pp. 219–236 (2009)
10. YarKhan, A., Dongarra, J.J.: Experiments with scheduling using simulated annealing in Grid Environment. In: Proceedings of GRID, pp. 232–242 (2002)
11. Yu, J., Buyya, R.: Scheduling scientific workflow applications with deadline and budget constraints using genetic algorithms. Scientific Programming 14, 217–230 (2006)
12. Yu, J., Buyya, R.: A taxonomy of workflow management systems for grid computing. J. Grid Comput. 3(3-4), 171–200 (2005)