Published at 2015 IEEE 19th International Enterprise Distributed Object Computing Workshop, pp. 14-23

Analyzing the Effects of Reordering Work List Items for Selected Control Flow Patterns

Johannes Pflug
Research Group Workflow Systems and Technology
University of Vienna
Vienna, Austria
Email: johannes.pflug@univie.ac.at

Stefanie Rinderle-Ma
Research Group Workflow Systems and Technology
University of Vienna
Vienna, Austria
Email: stefanie.rinderle-ma@univie.ac.at

***Abstract*—Efficient resource management is an important requirement for many process-oriented applications. Typically, work items are assigned to resources through their work lists. There are many reasons for reordering work items in a resource's work list. For process scheduling, for example, swapping process instances constitutes a mean to keep due times. At the same time, reducing the throughput time of the global process is typically not the primary goal. For process optimization, in turn, the implications of reordering work items on the overall temporal performance of the process might be crucial. In this paper, we investigate how reordering work items affects performance parameters that are typically associated with a *first-in-first-out* processing mechanism at resources. The analysis is conducted for single process tasks and for typical control flow patterns such as sequence as well as parallel and alternative branchings. It is shown that the implications on the global throughput time are less than expected, while the effects on instance-based parameters strongly depend on the control-flow pattern in which the *reordering* mechanism is implemented. The results are supported by means of a simulation.**

I. Introduction

Efficiently managing resources is a crucial requirement for almost any enterprise and is, in principle, a particular strength of business processes that are implemented through a process-aware information system (PAIS) [1], [2]. At the core of the PAIS, the process engine executes process instances during runtime. The work items to be performed for each process instance are assigned to the resources through their work lists. Note that work lists also realize a temporal alignment of instances. How the assignment of items to resources is realized depends on the chosen processing strategy[1]. It defines the logic according to which the process instances are processed, including the mapping of activities on resources and batching attributes. Of particular interest is the order in which the instances are processed.

Most of today's workflow engines implement a default processing strategy that is strongly oriented towards the temporal order of the work items [4]. Following a temporal order mostly means implementing a *first-in-first-out* logic. According to this approach, the entry with the earliest arrival time is the head of the work list (queue) and processed first. From a global point of view, this means that the order of the instances stays structurally equal throughout the whole workflow execution. However, there are scenarios in which a fixed *first-in-first-out* logic turns out to be not the optimum approach. This holds especially true for environments with defined due times or for advanced mechanisms like exception handling. Simulation results show that for almost all workloads rules such as earliest due date first are statistically significantly better than the commonly used FIFO rule regarding the number of late jobs [5]. It might also be the case for scenarios in which the order of the workflow instances influences the performance of the processing step. All these scenarios require a *reordering* of the work items in the resource's work lists.

We argue that reordering work items potentially has effects on the overall execution. However, the implications on typical temporal performance parameters have not been analyzed in detail. In this paper, we address the following questions:

- How does the reordering of work items in the resource's work list affect the temporal performance of the process execution?
- Which performance variables need to be distinguished? How do they interrelate?
- Which interdependence occurs with the processes' basic incorporated control-flow patterns (*single-task*, *exclusive split*, *sequence*, *parallelization* and *loop*)?

Some of the mentioned questions have been analyzed implicitly by assessing the performance of specific approaches which apply a reordering of work items, e.g. as part of scheduling techniques [5], [6], [7]. However, these findings focus on the specific target the particular approach follows.

So why do we revisit a basic scheduling problem? At first, the performance parameters that we analyze in our analysis are often neglected or only considered in a limited way in scheduling scenarios (where the number of kept due times is typically considered more significant) but we believe it is important to the workflow architect to be aware of the implications on overall time aspects as well. Second, the majority of today's workflow engines are not designed to handle defined due times at all. Especially in regards of process optimization and flexibility, reordering work items more probably represents a mean for exception handling or the optimization of the resource behavior at critical activities. To the best of our knowledge, no work that considers the implications of a reordered work queue systematically on temporal performance parameters exists at the moment.

This is especially true as research has neglected issues associated with the resource perspective in favor of the process

[1]Different strategies are described by means of resource patterns [3].

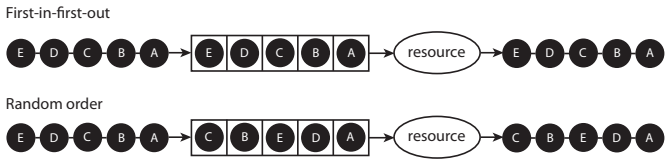

Figure 1. Comparison of non-reordering and reordering approach

perspective [3]. However we consider this an essential question as the global temporal performance is of vital importance in nearly any scenario. In this paper, we will elaborate on the implications of *reordering* work items on the temporal performance variables waiting time, processing time and throughput time, both on an instance and a global basis.

Our analysis is structured as follows: In Section II we will explain the basic concept and of reordering work items and provide background information about existing techniques that implement such a scheduling logic. The single-task scenario is analyzed in Section III, while the more enhanced control-flow patterns, *sequence*, *exclusive-choice*, *parallelization* and *loop* are investigated within Section IV. Section IV-E explains the hierarchical composition of processes and their temporal performance parameters based on the principle of process structure trees, while Section V provides an exemplary application scenario as well as an evaluation based on simulation data.

In the discussion (cf. Section VI), we draw a theory-based conclusion about the applicability of reordering in different scenarios. Moreover, our assumptions will be discussed: (a) Work items are not withheld from processing in order to execute a permutation, (b) tasks do not share resources and (c) the processing time is not dependent from the order in which the work items are processed. Related work is demarcated in Section VII. Finally, Section VIII concludes and shows future research directions.

II. Background

A. Implications of a reordered work list

Figure 1 shows the immediate effects of reordering work items within the resource's work list. A workflow instance is represented by a circle. As workflow instances are specifiable, in this scenario, a unique name (A, B, C, D, E) is assigned. The alphabetical order also indicates the arrival time of the instances, meaning that instance A arrived first and instance E last. As we consider the resource to start processing just when the last instance arrives, the instances are transferred to the resource's work list. A circle inside the rectangle represents a work item. Processing a work item means dequeuing it, performing the actions defined by the process model and assigning it to the next workflow entity according to the workflow engine's rule provider.

On top, a *first-in-first-out* approach is illustrated. As one can see, instances are placed in the resource's work list according to their entry date, putting the earliest work item representing instance A first. The lower part represents a reordering approach. The work items are reordered in an arrangement (A, D, E, B, C). This alignment could be referred to a random order. The resulting alignment also determines the arrival order of instances for the upcoming entity according to the process model.

There are several approaches which apply a reordering of work items in the resource's work lists. In the following, we illustrate commonly used approaches and techniques which are promising in terms of the process optimization. For all of them, the findings of this paper in terms of temporal performance parameters are valid.

Last-in-first-out [8], [9] is the complement to the first-in-first-out strategy. According to this approach, the item that has been added to the list last is handled first. It refers to the *stack* data structure and is accessed in opposite order to a queue. The last-in-first-out strategy is typically implemented in technical scenarios with a low level of abstraction.

Random-based means that the resource's work lists do not implement a certain order criteria. A random-based processing might be applied if there is a massive oversupply of resources compared to work items associated with different processing times of the resources for the same activities.

Due time or scheduling [10] settings are scenarios in which the processing end of workflow instances is fixed. To meet the temporal restrictions, it might be reasonable to process work items based on the priority of their due times. This scenario occurs for instance in the industry domain where goods need to be produced in time.

Priority determined processing [11] represents a strategy according to which instances are processed based on a specified priority. The priority is typically an instance attribute or can be evaluated out of other attributes. This approach is sometimes applied in the service domain, e.g. at telephone hotlines where the customer value represents the priority.

Dynamic instance queuing acknowledges the fact that the order of work items often influences the performance of the processing step [1]. Similar instances are classified based on artificial intelligence techniques and thus processed in a row. Scenarios are the industrial domain, in which one needs to embrace the implications of changeover times; or the domain of information technology in which techniques such as caching or multi-threading influence the performance.

B. Temporal performance variables

The most common temporal performance variables in a workflow system are the waiting time, the processing time and the throughput time of the instances within the workflow. In a single-resource activity pattern, instances arrive according to a certain distribution λ at the node associated to the activity [12]. Work items that represent the instances are then placed into the respective resource's work list. According to the scheduling logic, an order within the work list arises. The resource then processes the work items corresponding to the order of the work list and transfers the associated instances to the next entity of the process model.

The throughput time represents the time from the arrival of an instance at the node associated to the activity until the completion of its processing. This duration splits into the waiting time and the processing time. While the waiting

time specifies the part until the beginning of processing, the processing time represents the duration when the resource is busy processing the work item.

We refer to the throughput time for instance i at activity a as $t_{i,a}$, the waiting time as $w_{i,a}$ and the processing time as $p_{i,a}$. The cumulated throughput, waiting and processing times over all instances for a certain activity are represented by $\sum_{i\in I} t_{i,a}$, $\sum_{i\in I} w_{i,a}$ and $\sum_{i\in I} p_{i,a}$. The global throughput time beginning from the first instance until the end of processing of the last instance for a certain activity is referred to as T_a.

The arrival time is represented by the delay to a certain constant reference time and is symbolized by $d_{i,a}$ for instance i and activity a. We define $d_{i,a} > 0$, as we expect any instance to arrive at activity a with a positive delay. If the resource is busy at the time of the arrival of the first instance, an initial waiting time will occur. We refer to the duration until the resource is ready to process the first work item as k.

In order to be able to swap work items without withholding instances from processing, the work list must contain two or more items at the same time. This is the case when the $i-th$ instance arrives before the previous instance $i-1$ is started processing, i.e. $d_{i,a} < d_{i-1,a} + t_{i-1,a}\ \forall i \in I,\ \forall a \in A$. Otherwise, the work item representing the instance that arrived first is already completed processing when the second instance arrives. Note that if $d_{i,a}$ is equal for all $i \in I$, all instances arrive at the same time at the specified activity. As the throughput time is the sum of the waiting time and processing time, $t_{i,a} = w_{i,a}+p_{i,a}$ as well as $\sum_{i\in I} t_{i,a} = \sum_{i\in I} w_{i,a}+\sum_{i\in I} p_{i,a}$ are valid. However, as waiting times might overlap, $\sum_{i\in I} t_{i,a}$ is not necessarily the same as T_a.

III. Single task scenario

In this section, the effects of reordering of work items at a single resource are discussed.

A. Performance implications in the single task scenario

We will now evaluate the temporal performance variables introduced in Section II-B in a single-task scenario. This setting represents one activity with a single resource attached which can process exactly one work item at a time. The resource processes its work items according to its work list. We will show that in this scenario, reordering work items does not influence the waiting time, processing time or throughput time both on an instance and a global basis, if the following assumptions are made:

1) Work items are not withheld from processing in order to permute items in the work list.
2) The processing time is equal for any instance at the same activity-resource combination.

Assumption (2) represents a basic principle in almost any process simulation. It means that all instances share an equal processing structure for the combination of a certain activity and resource. The processing time therefore represents a *resource attribute* that is potentially different for any activity the resource is associated to. In process simulations, the processing parameters are typically estimated based on of previous values or expert knowledge. Remember that this assumption neither restricts different resources associated to the same activity from having divergent performance parameters nor expects equal effort for processing different activities.

The resource provides a certain set of processing ability which distributes over all work items. This means any instance requires a processing *slot* throughout the control-flow pattern. This abstract entity represents the way of the $n-th$ instance from the arrival at the resource's work list until the processing end. Temporal parameters of the slot represent the respective variables of the instance that is associated with the particular slot. The slot id s is the number of the slot in ascending order starting with 1. We consider a processing slot *used* if the processing ability offered by this slot was spent for an instance to pass the activity.

The set of instances within an interval has a certain order when the work items arrive at the node represented by the activity to be considered. We refer to this order as the *initial* order. The processing *slots* imply a certain order as well. This means the processing order results in a mapping of the initial order on the available slots. We argue there is a *bijective mapping* between the set of instances and the set of used slots:

Let X be the finite set of work items that are arranged at the same time in a resource's work list, and Y the finite set of used processing slots available at the resource. We define f as the mapping of the instances from set X on the slots from set Y. For any $x \in X$, exactly one $y \in Y$ exists, as a workflow is finished just when any instance has passed the whole process model. This is the case when all work items are finished processing by the associated resources. For this, exactly $|x|$ used processing slots are needed. On the other hand, for any $y \in Y$, exactly one $x \in X$ exists by definition: As a processing slot is defined as *used*, if its processing ability has been spent to pass a work item through an activity, any used processing slot is associated to a work item. As for any $x \in X$, exactly one $y \in Y$ exists and vice versa, the set of work items represents *bijective mapping* with the used processing slots. In consequence, the performance of a processing slot directly represents the performance of the associated work item, independent from the order in the resource's work list.

In the following, we will calculate the waiting, processing, and throughput time for the $s-th$ slot. According to assumption (1), no work items will be withheld from processing if the resource turns idle. This means the workflow execution at one resource can be split into intervals of the resource being nonstop busy and those in which the resource is idle. The occurrence of an idle time represents the end of a *busy* interval; the next instance to be processed marks the beginning of a new *busy* interval. For each *busy* interval, $d_i < d_{i-1} + t_{i-1}\ \forall i \in I$ for is true. Such an interval is shown for activity a in the following evaluation of temporal performance parameters. Note that we will omit index a for clarity.

First slot: The instance associated with slot 1 arrives at time d_1. As described above, slot 1 needs to wait until the resource has the capacity to process it, which takes k time units. This means the processing start is at $d_1 + k$. The throughput time for slot 1 is the sum of the waiting time (k) and processing time p_1. In consequence, the processing end is the throughput time ($t_1 = w_1 + p_1 = k + p_1$) added to the arrival time d_1.

Table I. PERFORMANCE PARAMETERS FOR THE PROCESSING SLOTS

j	arrival	w_j	p-start	p_j	t_j	p-end
1	d_1	k	$d_1 + k$	p_1	$w_1 + p_1$	$d_1 + t_1$
2	d_2	$d_1 + t_1 - d_2$	$d_1 + t_1$	p_2	$w_2 + p_2$	$d_2 + t_2$
3	d_3	$d_2 + t_2 - d_3$	$d_2 + t_2$	p_3	$w_3 + p_3$	$d_3 + t_3$
(...)						
s	d_s	$d_{s-1} + t_{s-1} - d_s$	$d_{s-1} + t_{s-1}$	p_s	$w_s + p_s$	$d_s + t_s$

Second slot: The processing starts when the instance from slot 1 is finished, which is at d_1+t_1. The waiting time evaluates as the difference between processing start and arrival at the node, which is $d_1 + t_1 - d_2$ (remember our assumption $d_i < d_{i-1} + t_{i-1}\ \forall i \in I$ so that at least two work items are in the work list at the same time). The throughput time for slot 2 is the sum of the waiting time ($d_1+t_1-d_2$) and processing time p_2. In consequence, the processing end is the throughput time ($t_2 = w_2 + p_2$) added to the arrival time d_2.

s-th slot: The processing starts when the processing of the previous slot $s-1$ is finished. The processing of the previous slot is evaluated as the sum of the previous slot's arrival time and its throughput time, which results in $d_{s-1} + t_{s-1}$. The waiting time of the *s-th* slot evaluates as the difference between processing start ($d_{s-1}+t_{s-1}$) and arrival at the node (d_s, which is $d_{s-1} + t_{s-1} - d_2$). The throughput time then evaluates as the sum of waiting time and processing time $t_s = w_s + ps = d_{s-1} + t_{s-1} - d_2 + p_2$.

An overview over the the arrival time (*arrival*), processing start time (*p-start*) and processing end time (*p-end*) as well as the waiting time w_j, processing time p_j and throughput time t_j for the slots is shown in Table I.

The global throughput time between the arrival at the first instance and the processing end of the *s-th* work item T_s is $\sum_{j=1}^{s} p_j$ (Equation 1). This means the global throughput time until the *s-th* slot is the sum of the processing times of the work items associated to all slots that have been processed so far. With assumption (2) comprised, the global throughput time is $s \cdot p$ for intervals where the resource is nonstop busy.

$$\begin{aligned} T_s &= d_s + t_s - d_1 = d_s + w_s + p_s - d_1 \\ &= d_s + d_{s-1} + t_{s-1} - d_s - d_1 \\ &= \sum_{j=1}^{s} d_j - \sum_{j=2}^{s} d_j - d_1 + \sum_{j=1}^{s} p_j = \sum_{j=1}^{s} p_j \overset{(2)}{=} s \cdot p \end{aligned} \tag{1}$$

The information about the waiting time, processing time and throughput time of the *s-th* slot allows us to evaluate the cumulated totals as well. Equation 2 shows that the cumulated waiting time over all slots of an interval is $s \cdot k + (s-1) \cdot d_1 - \sum_{j=2}^{s} d_s + \sum_{j=1}^{s}((s-j) \cdot p_j)$. While the cumulated processing time is $\sum_{j=1}^{s} p_j$, the cumulated throughput time over all slots evaluates as $s \cdot k + (s-1) \cdot d_1 - \sum_{j=2}^{s} d_s + \sum_{j=1}^{s}((s-j+1) \cdot p_j)$ (as shown in equation 3).

Any slot is associated to an instance being processed by the resource. The characteristics of this instance influences the temporal performance. However, equations 2 and 3 show that the cumulated waiting time, processing time and throughput time are only dependent from the associated work item in terms of its processing time. If assumption (2) is valid, i.e. the processing time is constant for any instance at the same activity (resp. resource), the cumulated totals do not depend at all from the instance that is associated with the slot. In this case, the cumulated waiting time would be $s \cdot k + (s-1) \cdot d_1 - \sum_{j=2}^{s} d_s + \frac{(s-1) \cdot s}{2} \cdot p$, the cumulated processing time would be $s \cdot p$ and the cumulated throughput time evaluates as $s \cdot k + (s-1) \cdot d_1 - \sum_{j=2}^{s} d_s + \frac{s \cdot (s+1)}{2} \cdot p$.

$$\begin{aligned} \sum_{j=1}^{s} w_j &= w_1 + \sum_{j=2}^{s}(d_{j-1} + t_{j-1} - d_j) = k + d_1 - d_s + \sum_{j=2}^{s} t_{j-1} \\ &= k + d_1 - d_s + \sum_{j=2}^{s}(w_{j-1} + p_{j-1}) \\ &= s \cdot k + (s-1) \cdot d_1 - \sum_{j=2}^{s} d_s + \sum_{j=1}^{s}((s-j) \cdot p_j) \\ &\overset{(2)}{=} s \cdot k + (s-1) \cdot d_1 - \sum_{j=2}^{s} d_s + \frac{(s-1) \cdot s}{2} \cdot p \end{aligned} \tag{2}$$

$$\begin{aligned} \sum_{j=1}^{s} t_j &= \sum_{j=1}^{s} w_j + \sum_{j=1}^{s} p_j \\ &= k + (s-1) \cdot d_1 - \sum_{j=2}^{s} d_s + \sum_{j=1}^{s}((s-j) \cdot p_j) + \sum_{j=1}^{s} p_j \\ &= s \cdot k + (s-1) \cdot d_1 - \sum_{j=2}^{s} d_s + \sum_{j=1}^{s}((s-j+1) \cdot p_j) \\ &\overset{(2)}{=} s \cdot k + (s-1) \cdot d_1 - \sum_{j=2}^{s} d_s + \frac{s \cdot (s+1)}{2} \cdot p \end{aligned} \tag{3}$$

As for the bijective mapping of the work items on the used processing slots and for the independence of the processing slots from a certain work item, all the performance parameters are valid for any order of the work items.

This means if there is more than one item in a resource's work list at the time another item is added, the set of work items can be reordered without having any effect on the cumulated waiting time, processing time and throughput time over all instances as well as on the global throughput time compared to a first-in-first-out logic.

B. Exemplary scenario

The findings from the previous section are illustrated based on a scenario that consists of one activity with an associated resource. The analysis consists of 5 instances, labeled as instances $A-E$. The arrival times as well as d_i as the delay to the reference time $0:00$ are shown in Table II. All instances share the same processing time p for this activity. We define p as 60 seconds. The time until the resource is able to process the first instance (k) is supposed to be 30 seconds.

In the non-reordering approach, instance A is mapped onto the first slot, instance B onto the second, etc. For the reordering approach, we assume a mapping as depicted in Table III. This results in a processing order of instance A, C, D, E, B.

The processing starts and ends, waiting times, processing times and throughput times are shown in Tables II (non-reordering approach) and III (reordering approach). It can be seen that although the mapping of instances on the slots differs, the cumulated totals are equal. The cumulated waiting time $\sum_{j=1}^{s} w_j$ is $s \cdot k + (s-1) \cdot d_1 - \sum_{j=2}^{s} d_s + \frac{(s-1) \cdot s}{2} \cdot p = 5 \cdot 30 + 4 \cdot 2 - (11 + 58 + 95 + 175) + 10 \cdot 60 = 419$. The cumulated throughput time over all slots $\sum_{j=1}^{s} t_j$ is $s \cdot k + (s-1) \cdot d_1 - \sum_{j=2}^{s} d_s + \frac{s \cdot (s+1)}{2} \cdot p = 5 \cdot 30 + 4 \cdot 2 - (11 + 58 + 95 + 175) + 15 \cdot 60 = 719$.

Table II. TEMPORAL PERFORMANCE PARAMETERS FOR THE NON-REORDERING APPROACH

slot	inst.	arrival	w_s	p-start	p_s	p-end	t_s
1	A	0:02 (2)	30	0:32 (32)	60	1:32 (92)	90
2	B	0:11 (11)	81	1:32 (92)	60	2:32 (152)	141
3	C	0:58 (58)	94	2:32 (152)	60	3:32 (212)	154
4	D	1:35 (95)	117	3:32 (212)	60	4:32 (272)	177
5	E	2:55 (175)	97	4:32 (272)	60	5:32 (332)	157
$\sum$	-	419	-	300	-	719	

Table III. TEMPORAL PERFORMANCE PARAMETERS FOR THE REORDERING APPROACH

slot	inst.	arrival	w_s	p-start	p_s	p-end	t_s
1	A	0:02 (2)	30	0:32 (32)	60	1:32 (92)	90
2	C	0:58 (58)	34	1:32 (92)	60	2:32 (152)	94
3	D	1:35 (95)	57	2:32 (152)	60	3:32 (212)	117
4	E	2:55 (175)	37	3:32 (212)	60	4:32 (272)	97
5	B	0:11 (11)	261	4:32 (272)	60	5:32 (332)	321
$\sum$	-	419	-	300	-	719	

IV. ANALYSIS OF CONTROL-FLOW PATTERNS

In this section, we expand the analysis to the commonly used control-flow patterns *sequence*, *exclusive-choice*, *parallelization* and *loop* (cf. [3]). We show that reordering of work items potentially has negative impact on the performance parameters when executed in a parallelization pattern, while the application in the other patterns is neutral.

All the addressed control-flow patterns are deeply interrelated in terms of a hierarchical structure. *Process structure trees* are a possibility to describe the hierarchical composition of a process [13], [14]. Both a sequence, parallelization, loop and exclusive-choice are composed out of other control-flow patterns (if not composed out of single-tasks). We refer to these elements, out of which a pattern is orchestrated as *subpatterns*. In the lowest level, any of the patterns analyzed in this work can be described by a single-task. In this context, determining the temporal performance variables for a certain pattern means evaluating the impact of this pattern on the *incorporated* subpattern.

A. Sequence

The sequence pattern is orchestrated of n consecutively executed subpatterns ($n \geq 2$). Each of these subpatterns represents either another control-flow pattern or a single-task. Applying a reordering approach in a sequence pattern means reordering the work items in the incorporated subpatterns. In the following, we will first argue that the cumulated waiting time over all instances for a sequence of two subpatterns corresponds to the sum of the cumulated waiting times of first and the second subpattern. Cumulated processing and throughput time are treated analogously. Based on this, it can be shown that the sequence's performance corresponds to the concatenation of the performance of the two incorporated subpatterns, but the sequence *itself* does not influence the performance. On this basis, we show that this statement is valid for the concatenation of n subpatterns ($n > 2$) as well.

We now consider one instance i to be executed for a sequence pattern consisting of two subpatterns subpattern 1 and subpattern 2. i has throughput time $t_{i,sub1}$ for subpattern 1 and throughput time $t_{i,sub2}$ for subpattern 2. Subpatterns 1 and 2 are consecutive. This means that the processing end time of instance i at subpattern 1 equals its arrival time at subpattern 2. As the beginning of the throughput time is defined as the time of the arrival at a certain node, there is no gap in the throughput time between subpattern 1 and 2. Therefore, the cumulated throughput time of the instance for the whole sequence is the sum the instance's throughput time for subpattern 1 and 2. In formal terms, for $P = \{sub1, sub2\}$, $\sum_{sub \in P} t_{i,sub} = t_{i,sub1} + t_{i,sub2}$. The cumulated throughput time over all instances $i \in I$ that pass the sequence hence evaluates as $\sum_{i \in I} \sum_{sub \in P} t_{i,sub} = \sum_{i \in I} (t_{i,sub1} + t_{i,sub2})$.

Due to the definition of a sequence, any instance passes subpattern 1 and subpattern 2 exactly once. Therefore, the cumulated throughput time over all instances can be described as $\sum_{i \in I} t_{i,sub1} + \sum_{i \in I} t_{i,sub2}$ as well. This means, the sequence's cumulated throughput time over all instances represents the sum of the first subpattern's throughput time over all instances and the second subpattern's throughput time over all instances.

These considerations can be transferred to a sequence orchestrated out of more than two subpatterns. For n subpatterns ($n \geq 2$), two directly linked subpatterns can be interpret as a *subsequence* of two subpatterns. For this subsequence, the statements previously explained are valid. Combining iteratively all subpatterns to 2-tuples, one can explain a sequence of n subpatterns gradually by a sequence of two subpatterns. Therefore, the cumulated throughput time of a sequence with n subpatterns ($n \geq 2$) is the sum of all incorporated subpatterns' throughput times over all instances. Formally this means for the throughput time over all instances I and subpatterns P $\sum_{sub \in P} \sum_{i \in I} t_{i,sub}$. This is both true for a reordering and non-reordering approach.

The correlation for the cumulated waiting time and processing time over all instances is equivalent. The global throughput time is defined as the time span between the first instance's arrival at the first subpattern's node and the processing end of the last instance at the last subpattern's activity. Therefore, the evaluation of the global throughput time is equal both for a reordering and non-reordering approach.

Remember that in this analysis, we observed the structural impact of a sequence control-flow pattern on typical temporal performance variables. We argue that both for a reordering and non-reordering approach, a sequence's cumulated waiting time, processing time and throughput time represents the sum of the respective incorporated subpattern's cumulated values. This means, one can concatenate several elements using a reordering approach without expecting negative impact *through the concatenation itself*. However, this does not mean that the absolute values of the sequence's temporal performance variables are identical between a reordering and non-reordering approach. The concrete values are the result of the evaluation of the incorporated subpatterns.

B. Exclusive-choice

The *exclusive-choice* pattern represents the divergence of the process execution into several branches, of which each represents a *subpattern*. Based on a certain decision function, any instance passes exactly one branch. In consequence, only the subpattern associated to the chosen branch is relevant for the performance of the exclusive-choice pattern. This means that the exclusive-choice pattern *itself* does not have any

influence on the temporal performance variables both on an instance and global basis.

C. Parallelization

For a parallelization pattern, a single thread of execution is split into two or more branches which can execute tasks concurrently [15] and are synchronized in the sequel. Each of the branches represents a *subpattern*, which can turn out to be either a single task, exclusive-choice, sequence, loop or another parallelization.

Contrary to the sequence pattern, the parallelization pattern incorporates two or more threads that process independently from each other. However, these threads culminate in a combined performance for the whole pattern. As the instance execution cannot proceed before the synchronization the thread or branch with the longest throughput time dominates the overall performance of the execution of the parallelization pattern.

On an instance basis, results from the (independent) processing of the branches cannot be necessarily delegated on the parallelization. The reason is that the several branches are finally merged to a single thread of execution. During the execution of the parallelization, an instance is associated to several work items, each for one branch. If the work items are reordered within the subpatterns, the alignment of the work items differs between the subpatterns. As the parallelization is finished just when the execution of any work item associated to a certain instance is finished, reordering can have negative impact on the cumulated performance parameters. Compared to a non-reordering approach, the influence is neutral, if the reordering is executed synchronous over the work lists of the resources in the subpatterns. The more asynchronous the alignment of work items is, the higher is the negative impact on the cumulated performance parameters. The worst-case scenario represents an inverse processing order.

The throughput time $t_{i,P}$ for one instance at parallelization P is the maximum throughput time of the incorporated branches $\in P$ for the associated work item. This means, $t_{i,P} = max(t_{i,branch}) \; \forall \; branch \in P$ and for the cumulated throughput time, $\sum_{i \in I} t_{i,P} = \sum_{i \in I} max(t_{i,branch}) \; \forall \; branch \in P$ is valid. The throughput time over all instances for any branch is similar in a reordering and non-reordering approach, if the respective branches consist out of single-tasks, sequences, loops and/or exclusive choices (cf. Sect. III, IV-A, IV-B, IV-D). This means $max(t_{i,branch})$ is minimum for any instance if the difference between the throughput times of the subbranches is minimum. This corresponds to a synchronous processing as described above.

Figure 2 illustrates this fact in a simple scenario. It shows a parallelization pattern with two branches, each subpattern represented by a single task pattern. We assume there are five instances that arrive at the parallelization pattern in alphabetical order (A, B, C, D, E) at the same time. The processing time of both resource 1 and resource 2 is one time unit per instance. In a non-reordering approach (upper figure), the order obviously remains the same over the branches. The parallelization is completed for any instance just at the time the corresponding subbranches are completed, meaning instance A one time unit after start, instance B two time units after start

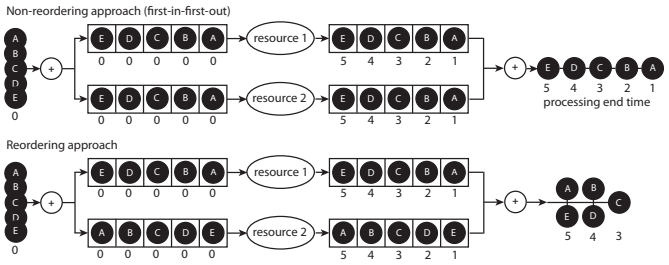

Figure 2. Comparison of a reordering and non-reordering approach in a parallelization pattern

until instance E, which is finished five time units after the processing start. In the lower part of Figure 2, a reordering scenario using the same parameters is shown. We assume in the upper branch that the work items remain the order of the associated instances (A, B, C, D, E), while in the lower branch, the order is inverted (E, D, C, B, A). In consequence, the work item associated with instance A is finished processing by resource 1 only one time unit after start; while processing it by resource 2 is finished after five time units. Since the parallelization for instance A is finished at the time when both resources are finished processing the associated work items, instance A is completed after five time units. In fact, the first instance to be finished is instance C after three time units, instance B and D take four time units each.

Obviously, the cumulated throughput time over all instances is different concerning the *reordering* and *non-reordering* approach, although this is not the case for the single task scenarios themselves that are incorporated by the parallelization. If the processing time is constant, the optimum cumulated throughput time is achieved when the instances are processed simultaneously. This particularly is the case for the *first-in-first-out* logic, but also for any other synchronous reordering approach over all branches. Any asynchronous reordering of instances increases the cumulated throughput time. This is most probably the case if the reordering is executed independently in the branches of the parallelization.

The global throughput time is the duration between the arrival of the first instance and the processing end of the last instance at the parallelization pattern. If the branches are orchestrated out of single-tasks, sequences, loops and/or exclusive-choices, the throughput time over all instances for each branch is similar both in a reordering and non-reordering approach. In consequence, the maximum cumulated throughput time of these branches is similar as well. The time when the branch with the maximum cumulated processing time becomes unbusy also represents the processing end of the parallelization in any interval. Therefore, the global throughput time as the duration between the arrival of the first instance and the processing end of the last instance is similar in a reordering and non-reordering approach for a parallelization.

In summary, reordering work items in the subpatterns of a parallelization pattern potentially has negative impact on the cumulated waiting time and throughput time over all instances compared to a non-reordering approach. If this parameter is important to the workflow director, the application of an algorithm that implements a reordering logic is not reasonable in terms of temporal parameters. If one focuses on the global

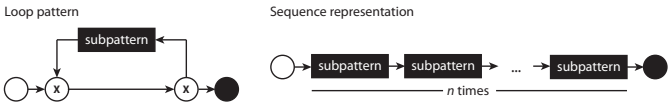

Figure 3. Sequence representation of a loop

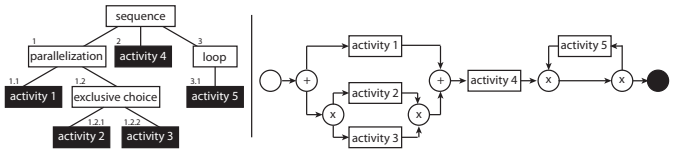

Figure 4. Sample process structure tree

throughput time only, meaning it is important when the last instance is processed, but not when the instances within are finished, one can also apply a reordering approach without expecting negative impact on the overall time.

D. Loop

A loop pattern expresses the repeated execution of a task or another subpattern. The loop has either an associated pre-test or post-test condition that is evaluated at the beginning resp. end of the loop to determine whether it continues [15]. If the evaluation of the condition results in the decision not to go on with the repeated execution, the loop pattern is finished and the instances are transferred to the next entity as described in the process model.

Figure 3 shows the basic representation of a loop scenario with subpattern (black color). As discussed in literature, loop patterns can be linearized [16] and hence treated as a sequence, i.e., a repeated processing of n subpatterns, with n being the number of iterations (cf. Figure 3). The subpatterns, in turn, might incorporate further subpatterns. This means that all the results from analyzing the sequence pattern are valid for the loop as well: The cumulated throughput time over all instances in set I and subpatterns P is $\sum_{sub \in P} \sum_{i \in I} t_{i,sub}$. As all subpatterns are equal, the number of iterations directly influences the cumulated performance parameters. For n iterations, the cumulated throughput time over all instances is $n \cdot \sum_{i \in I} t_{i,sub}$. As for the sequence pattern, the global throughput time does not depend on the order of the work items. Please note that by describing a loop by a sequence, we interpret each iteration of the subpattern as a unique entity with a fixed single resource attached.

In summary, the analyzed temporal performance parameters for the loop pattern are not dependent of the order of the work items in the subpattern that is repeated. However, the actual performance strongly depends on the characteristics of the applied subpattern as described in the previous sections. Represented as a process structure tree (cf. Sect. IV-E), the involved patterns within the subpattern in the different layers are of particular interest. If only sequences, loops and single tasks are incorporated, reordering work items does not influence the temporal performance parameters at all. If a parallelization is involved, the impact described in Sect. IV-C occurs. The effect then multiplies with the number of iterations.

E. Composition of control-flow patterns

An important question is how to estimate the effects of reordering work items at composed process patterns. *Process structure trees* represent a mean to describe the hierarchical composition of a certain process [13]. We use a notation that is based on process structure trees to visualize the relation between the control-flow patterns and single-tasks in different application scenarios. Single-tasks are represented by dark rectangles, while white rectangles describe the specific control-flow patterns. Transitions represent the structural aspect of a process model and capture the relation between the layers of abstraction.

Figure 4 shows an example of a process structure tree. It represents a sequence that is orchestrated out of three elements on the first level: Node 1 represents a parallelization; Activity 4 is a single-task and represents the second element of the sequence, while the third one is a repeated execution of activity 5 (loop). The parallelization (node 1) consists of two branches. The first branch of the parallelization is considered node 1.1 and represents activity 1, while the other branch (node 1.2) represents an exclusive choice out of activity 2 (node 1.3) and activity 3 (node 1.4). One can see that the single-tasks representing the activities are the base building block for any pattern. In the process structure tree, they can be considered *leaf* elements.

The overall temporal performance can be evaluated by consecutively evaluating the performance of the incorporated patterns, their subpatterns and incorporated single-tasks in a bottom-up approach regarding the execution order within the process. This means the waiting, processing and throughput times are evaluated for each element beginning from the leafs up to the root pattern.

V. Demo Scenario and Simulation

We will now the evaluate the temporal performance parameters for the example from Figure 4. We assume a scenario of five instances (A, B, C, D, E) that arrive at the same time at the first node of the process. Furthermore, we expect each instance to utilize a processing time of one time unit for any of the single-tasks. For the exclusive-choice, we assume instances A, B and E to tread the path of activity 2, while instances C and D pass the node of activity 3. We expect any instance to run the loop three times. Please note that we assume a different resource attached to activity 5 for each iteration of the loop. In any resource's work list associated to a single-task, the work items are being reordered on a random-basis. As work items are not withheld from processing and the processing times are constant, both assumptions from Sect. III are kept. This means that all the findings from the previous sections are valid for this scenario. However, the structural approach to evaluate the overall performance parameters would be the same if the order of work items, the arrival times or the performance values were differently.

Table IV shows the runtime behavior of the instances. For any element within the process model, the order of the work list before processing (pre-order) and the order after processing (post-order) is indicated. The relative time of the arrival resp. processing end is put in parentheses.

Table IV. RUNTIME BEHAVIOR OF THE INSTANCES IN A RANDOM-REORDERING APPROACH

#	name	pre-order (time)	post-order (time)
1	paralleliz.	ABCDE(0)	A(2)-BE(3)-D(4)-C(5)
1.1	activity 1	ABCDE(0)	B(1)-A(2)-E(3)-D(4)-C(5)
1.2	excl. choice	ABCDE(0)	AD(1)-CE(2)-B(3)
1.2.1	activity 2	ABE(0)	A(1)-E(2)-B(3)
1.2.2	activity 3	CD(0)	D(1)-C(2)
2	activity 4	A(2)-BE(3)-D(4)-C(5)	A(3)-E(4)-D(5)-B(6)-C(7)
3	loop	A(3)-E(4)-D(5)-B(6)-C(7)	A(6)-E(7)-D(8)-B(9)-C(10)
3.1	act. 5 (run1)	A(3)-E(4)-D(5)-B(6)-C(7)	A(4)-E(5)-D(6)-B(7)-C(8)
3.1	act. 5 (run2)	A(4)-E(5)-D(6)-B(7)-C(8)	A(5)-E(6)-D(7)-B(8)-C(9)
3.1	act. 5 (run3)	A(5)-E(6)-D(7)-B(8)-C(9)	A(6)-E(7)-D(8)-B(9)-C(10)

Table V. TEMPORAL PERFORMANCE ASSESSMENT

#	name	$\sum_{i \in I} w_i$		$\sum_{i \in I} t_i$		T	
		reorder	fifo	reorder	fifo	reorder	fifo
1	parallelization	14	10	19	15	5	5
1.1	activity 1	10	10	15	15	5	5
1.2	exclusive-choice	4	4	9	9	3	3
1.2.1	activity 2	3	3	6	6	3	3
1.2.2	activity 3	1	1	3	3	2	2
2	activity 4	3	0	8	5	5	5
3	loop	0	0	5	5	7	7
3.1	activity 5 (run1)	0	0	5	5	5	5
3.1	activity 5 (run2)	0	0	5	5	5	5
3.1	activity 5 (run3)	0	0	5	5	5	5

To determine the cumulated waiting time and throughput time both on an instance and global basis, we will proceed as described in Sect. IV-E: For each of the three incorporated branches, the temporal performance parameters are evaluated in a bottom-up approach. For the parallelization, first the performance of activities 2 and 3 is evaluated, which is then merged to the performance for the exclusive-choice, which is, again, merged with the performance of activity 1 to the parallelization. The performance of the second element represents the performance of its leaf element activity 4, while the third branch is the repeated execution of activity 5.

Table V shows the cumulated waiting time and throughput time over all instances as well as the global throughput time for each element in the process model both in the reordering and non-reordering approach. According to a bottom-up approach, the performance of activities 1, 2 and 3 as the leafs of part one of the sequence are evaluated first. It becomes obvious that all the performance parameters are equal for the reordering and non-reordering approach (as proven in Sect. III). Activities 2 and 3 are merged to the exclusive-choice, which does not influence the results as well (cf. Sect. IV-B). However, when merging the two branches of the parallelization, the cumulated waiting time and processing time differs. As described in Sect. IV-C, the different alignment of work items within the two branches in the reordering context results in extended waiting times, which finally increase the throughput times as well.

The second part of the overall sequence, activity 4, represents a single-task scenario. According to Sect. III, the temporal performance does not differ between a reordering and non-reordering approach for single-tasks. However, in this case, the structure of the incoming work items is different due to the inhomogeneous processing during the preceding parallelization. If activity 4 was analyzed solely, there would not be any difference as a result of a different alignment of the resource's work list. The repeated execution of activity 5 (loop) represents the last part of the sequence. As described in Sect. IV-D, reordering of work items does not affect the temporal performance parameters. This is also the result of the application scenario at hand.

The global throughput time of the exemplary scenario is 10 time units both in the reordering and non-reordering approach. However, the cumulated waiting and processing time differ as a result of the parallelization implemented in the application scenario. The results of this example correspond with the findings from the previous sections: Work items can be reordered without having any effect on the global throughput time. For the temporal performance parameters at an instance basis, one needs to distinguish between the different control-flow patterns. For the parallelization, a different alignment from the first-in-first-out logic results in a worse performance. The results from the other elements, meaning the single-tasks, exclusive-choice and loop, are not affected by reordering work items.

The process from Fig. 4 was also executed in a tool-based simulation with a more comprehensive set of parameters[2]: Different from the previous example, we don't assume all the instances to arrive at the first node at the same time. In total, 128 instances traverse the process. The number of iterations in the loop differs between the instances as well. We expect a processing time of 6 time units (tu) at activity 1, 4 tu at activity 2, 8 tu at activity 3, 12 tu at activity 4 and 15 tu at activity 5.

We execute the simulation twice: In the first run, instances are not reordered in the resource's work lists at all. For the second run, instances are being reordered. However to ensure full comparability, the arrival time as well as the traversed branch for the exclusive-choice pattern and the number of iterations in the loop are equal for the same instance in both scenarios. As work items are not withheld from processing in order to reorder, we defined the distribution of arrival times in a way that the resources don't have the capacity to process all instances in time and therefore, queues of work items arise.

In Sect. III and IV, we proved that the global throughput time is not affected by a reordering of work items. The cumulated totals for the waiting time and throughput time over all instances depend on the specific control-flow patterns that are incorporated. As the current scenario includes a parallelization, we expect a better cumulated throughput time for the non-reordering approach than for the reordering approach. The simulation results confirm these expectations. The global throughput time T, meaning the time frame between the arrival of the first instance at activity 1 and the processing end of the last instance at activity 5 is 2750 tu both for the reordering and non-reordering simulation run. The cumulated throughput time over all instances and activities is 133831 tu for the non-reordering approach and 136567 tu for the reordering approach, which represents a difference of 2.04%. The difference is a result of the waiting times for the join of the two branches of the parallelization.

The simulation shows that our theorems are valid for a scenario with a comprehensive set of parameters as well. Again we see that reordering of work items has no impact on the

[2]The simulation data sets with the corresponding process models can be found at http://cs.univie.ac.at/wst/research/projects/project/infproj/1060/.

Table VI. OVERVIEW OF THE INFLUENCE OF REORDERING WORK ITEMS WITHIN DIFFERENT CONTROL-FLOW PATTERNS ON THE TOP LEVEL

Control-flow pattern	$\sum_{i \in I} w_i$	$\sum_{i \in I} p_i$	$\sum_{i \in I} t_i$	T
Single-task	neutral	neutral	neutral	neutral
Sequence	neutral	neutral	neutral	neutral
Exclusive-choice	Effects depend from the subpattern of the executed branch			
Parallelization	negative	neutral	negative	neutral
Loop	neutral	neutral	neutral	neutral

global throughput time and little impact on the cumulated totals. Without applying a reordering within the parallelization, all relevant temporal performance indicators were equal.

VI. DISCUSSION

In our analysis, we argued that the influence of reordering work items on temporal performance parameters that are typically associated with the *first-in-first-out* logic depends on the specific context they are applied in as well as from the goals that are pursued. Reordering work items in the resource's work list does not have any influence on the global throughput time, i.e. the time between the appearance of the first instance and the processing end of the last instance, independent from the workflow pattern it is applied in.

When focusing on the instance basis, one needs to distinguish between the different control-flow patterns. For the single-task scenario as well as for the sequence and loop, all temporal performance parameters are equal both for the *reordering* and *non-reordering* approach. When work items are reordered within a parallelization, the cumulated throughput time and waiting time over all instances will most probably be higher than in a *first-in-first-out* processing approach. All results are shown in Table VI.

Remember that Table VI shows the effects of the control-flow patterns on the top level of a process structure tree, i.e. the patterns are interpret as *self-contained* workflow elements. However, control-flow patterns are often nested, i.e. one control-flow pattern incorporates others. In this case, all the involved subpatterns need to be evaluated. To put it in a nutshell, if no parallelization pattern is involved, one can apply any combination of subpatterns without expecting an impact on performance parameters through the reordering of work items. If a parallelization is involved, the reordering will most likely have negative influence on the cumulated waiting time and processing time.

For our analysis, the following assumptions have been made: (a) Work items are not withheld from processing in order to execute a permutation, (b) tasks do not share resources (c) the processing time is not dependent from the order in which the work items are processed. These assumptions leave room for further investigations.

1) *Extension of swapping work items:* Reordering instances is only possible if there are two or more associated work items in the resource's work list at the same time. This is the case when the resource does not have the capacity to handle all work items in time and hence, a queue arises. However, one could withhold work items from processing even if the resource is not busy in order to generate a work list of two items. This could be a promising approach for scenarios with strict due times or in terms of exception handling. However, pursuing this logic will definitely have negative impact on the temporal performance parameters compared to a *non-reordering* approach. Further research on the extent of the consequences on the temporal variables seems valuable.

2) *Extension of resource allocation:* In the paper at hand, we analyzed scenarios in which the resource behavior is deterministic for the single task it is associated to. This is the case for scenarios with one or more resource per task as well as for batch-processing scenarios. However, one could also think of alignments in which resources are associated to several tasks, i.e. the tasks share their processing capacity. For an evaluation of temporal performance parameters, the work lists of all the resources need to be respected, as it might be valuable to apply the processing capacity for one task in order to have extended permutation possibilities for another work list.

3) *Implementation of dynamic instance queuing:* For our evaluation, we assumed that the processing times are not dependent from the order in which the work items are processed. However, this assumption is not true for some scenarios, as the processing of *similar* instances often enables a better processing performance. Parameters like changeover times in industrial scenarios or the gaining of routine by humans influence the efficiency of work. The *dynamic instance queuing* approach takes advantage of this effect by evaluating the similarity of instances applying artificial intelligence techniques [1]. This algorithm also makes use of reordering techniques for work items. In future work, we will investigate the performance parameters on dynamic instance queuing.

VII. RELATED WORK

Reordering work items in resource's work lists relates to several topics that have been addressed by contemporary literature. We consider our work transcendent to the following research areas:

Time aspects in workflow systems: During the last decade, the management of temporal aspects for workflows has attracted considerable research efforts across different dimensions [17]. Different approaches address this issue [18], [19], [20], [21], [10], [22]. Approaches can be divided into those which capture time aspects at design time (in order to predict certain parameters like the throughput times) and those which are applied at runtime. At design time, for example, it has been investigated how to cover uncertainty of processing and critical paths. At runtime, adherence of the process instances to imposed time restrictions such as deadlines is monitored. All these questions become more challenging when considered for process choreographies [17]. An analysis of existing approaches based on time patterns can be found in [23]. All these approaches share the characteristic that - according to the specific scenario - a certain target is defined, implemented and ultimately assessed. The work presented in this paper, however, investigates the implications on global targets such the throughput time. That means, we approve the need to focus on the main targets, but we argue that one should also involve *secondary targets* or be at least aware of the consequences on the temporal parameters evaluated in this paper when applying scheduling mechanisms that apply a *reordering* of work items.

Scheduling approaches: Scheduling mechanisms aim to handle scenarios in which the due times of instances are fixed, i.e. certain end times are defined at which the processing needs to be completed. Several approaches exist to address the problem of keeping the due times. The work of Son et al. [6] aims at maximizing the number of workflow instances satisfying the given deadline by determining the necessary number of resources. It first evaluates the critical activity, which is considered the task whose delay directly affects the overall processing time and based on this conclusion, the need for resources is evaluated. However, most of the work addresses the problem of finding a correct execution sequence for the workflow tasks; [24], [25], [26] created an algebra to describe constraints in workflows which culminate in modern scheduling techniques applied in today's production planning systems. Flockhart et al. [7], e.g., offer an approach to optimize results expressed through completion times by repeatedly reordering work items for which the analysis from this paper can be applied. All these approaches have in common that work items are reordered. The promising character of this approach is described by [5], who argue that for almost all workloads rules such as earliest due date first, and guess and solve are statistically significantly better than the commonly used FIFO rule regarding the number of late jobs.

Workflow resource patterns: These patterns aim to capture the various ways in which resources are represented and utilized in workflows [3]. Of particular interest for the topic of this paper is the *System-Determined Work Queue Content*. This pattern represents the ability of the workflow engine to order the content and sequence in which work items are presented to a resource for execution. Modeling these techniques has been a topic of interest as well. Especially what concerns the support for BPEL, serious research and implementation efforts have been undertaken [27].

VIII. Summary

This paper systematically analyzed the effects of reordering instance executions on the temporal performance of the process. The analysis started from considering the reordering for single tasks to commonly used process patterns, i.e., sequence, exclusive choice, parallelization, and loops. For the process patterns, the "top level" was analyzed, i.e., the effects on, for example, a sequence of subpatterns. The subpatterns can be single tasks or again process patterns. It was shown that reordering does not affect the temporal performance for the single task scenario as well as for the sequence and loop pattern. For exclusive-choices, the effects depend on the subpatterns of the chosen branch. For parallelization, reordering will most likely lead to negative effects on the temporal performance. All results were theoretically deduced and illustrated by a example scenario. Moreover, the results were evaluated based on a simulation. Though the results of this analysis might seem partly intuitive, their systematic consideration and results provide valuable input for further consideration for, e.g., process optimization based on queuing.

References

[1] J. Pflug and S. Rinderle-Ma, "Dynamic instance queuing in process-aware information systems," in *SAC*, 2013, pp. 1426–1433.

[2] L. Pufahl, A. Meyer, and M. Weske, "Batch regions: Process instance synchronization based on data," in *EDOC*, 2014, pp. 150–159.

[3] N. Russell, W. van der Aalst, A. ter Hofstede, and D. Edmond, "Workflow resource patterns: Identification, representation and tool support," in *CaISE*, 2005.

[4] Y. Yu, T. Xie, and X. Wang, "A handling algorithm for workflow time exception based on history logs," *The Journal of Supercomputing*, vol. 63, no. 1, pp. 89–106, 2013.

[5] G. Baggio, J. Wainer, and C. Ellis, "Applying scheduling techniques to minimize the number of late jobs in workflow systems," in *SAC*, 2004, pp. 1396–1403.

[6] J. H. Son and M. H. Kim, "Improving the performance of time-constrained workflow processing," *J. Syst. Softw.*, vol. 58, no. 3, pp. 211–219, 2001.

[7] A. Flockhart, D. Maxwell, K. McFarlane, P. Richman, and L. Sanders, "Workflow-scheduling optimization driven by target completion time," Oct. 8 2002, uS Patent 6,463,346.

[8] P. Delias, A. Doulamis, N. Doulamis, and N. Matsatsinis, "Optimizing resource conflicts in workflow management systems," *TKDE*, vol. 23, no. 3, pp. 417–432, 2011.

[9] X. Meining and X. Qinglin, "Workflow model analysis based on colored petri nets," *International Proceedings of Computer Science & Information Tech*, vol. 53, 2012.

[10] S. Sadiq, O. Marjanovic, and M. Orlowska, "Managing change and time in dynamic workflow processes." *IJCIS*, vol. 9, no. 1&2, pp. 93–116, 2000.

[11] N. Russell, A. H. M. Ter Hofstede, and D. Edmond, "Workflow resource patterns," Eindhoven University of Technology, Tech. Rep., 2004.

[12] J. Little, "A proof for the queuing formula: L= λ w," *Operations research*, vol. 9, no. 3, pp. 383–387, 1961.

[13] J. Vanhatalo, H. Völzer, and J. Koehler, "The refined process structure tree," in *BPM*, 2008, pp. 100–115.

[14] A. Senderovich, M. Weidlich, A. Gal, and A. Mandelbaum, "Queue mining – predicting delays in service processes," in *CaISE*, 2014, pp. 42–57.

[15] N. Russell, A. ter Hofstede, W. van der Aalst, and N. Mulyar, "Workflow control-flow patterns: A revised view," BPM Center, Tech. Rep. BPM-06-22, 2006.

[16] S. Rinderle, M. Reichert, and P. Dadam, "Flexible support of team processes by adaptive workflow systems," *Distributed and Parallel Databases*, vol. 16, no. 1, pp. 91–116, 2004.

[17] J. Eder, E. Panagos, and M. Rabinovich, "Workflow time management revisited," in *Seminal Contributions to Information Systems Engineering*, 2013, pp. 207–213.

[18] J. Eder and A. Tahamtan, "Temporal conformance of federated choreographies," in *DEXA*, 2008, pp. 668–675.

[19] H. Li and Y. Yang, "Dynamic checking of temporal constraints for concurrent workflows," *Electronic Commerce Research and Applications*, vol. 4, no. 2, pp. 124–142, 2005.

[20] J. Li, Y. Fan, and M. Zhou, "Timing constraint workflow nets for workflow analysis," *IEEE Trans. on Systems, Man, and Cybernetics*, vol. 33, no. 2, pp. 179–193, 2003.

[21] Y. Pan, Y. Tang, H. Ma, and N. Tang, "Workflow analysis based on fuzzy temporal workflow nets," in *Computer Supported Cooperative Work in Design II*, 2006, vol. 3865, pp. 545–553.

[22] H. Pichler, M. Wenger, and J. Eder, "Composing Time-Aware web service orchestrations," in *CaISE*, 2009, pp. 349–363.

[23] A. Lanz, B. Weber, and M. Reichert, "Workflow time patterns for Process-Aware information systems," in *Enterprise, Business-Process and Information Systems Modeling*, 2010, vol. 50, pp. 94–107.

[24] P. Attie, M. P. Singh, A. P. Sheth, and M. Rusinkiewicz, "Specifying and enforcing intertask dependencies," in *VLDB*, 1993, pp. 134–145.

[25] M. P. Singh, "Semantical considerations on workflows: An algebra for intertask dependencies," in *Int'l Workshop on Database Programming Languages*, 1995.

[26] M. Singh, "Synthesizing distributed constrained events from transactional workflow specifications," in *ICDE*, 1996, pp. 616–623.

[27] W. van der Aalst, "Don't go with the flow: Web services composition standards exposed," 2003.