Dynamic Instance Queuing in Process-Aware Information Systems

Johannes Pflug
University of Vienna
Faculty of Computer Science
johannes.pflug@univie.ac.at

Stefanie Rinderle-Ma
University of Vienna
Faculty of Computer Science
stefanie.rinderle-ma@univie.ac.at

ABSTRACT

Reducing the processing time of instances at critical activities is essential for many application domains. We refer to an activity as being critical if due to restricted resources assigned to the activity, the arrival of a certain number of process instances might lead to a waiting queue. So far, queuing has been adopted for process optimization in a merely static manner, i.e., the strategy in which order the instances are processed from the queue is fixed. We argue that determining the processing strategy for instance queues at runtime (*dynamic queuing*) offers the potential to reduce the processing time at critical activities. The core idea is that instances arriving at critical activities are first clustered based on similar features and are then distributed to dynamic queues accordingly. The decision on the processing order for the resulting queues requires a state management for allocating the appropriate number of resources during runtime. For this, a configurable performance index is used. The proposed dynamic queuing approach is prototypically implemented and evaluated based on a realistic data set.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous; D.2.8 [**Software Engineering**]: Metrics—*complexity measures, performance measures*

Keywords

Process performance measurement, Process optimization, Queuing in PAIS

1. INTRODUCTION

Supporting the efficient execution of process instances is crucial for many application scenarios. Especially the arising of waiting queues as a result of restricted resources assigned to process activities (referred to as critical activities in the following) can have massive impact on the process success, ranging from delays over failures even to deadlocks. Queuing has been already adopted for process optimization by different approaches, particularly combining queuing with batching [16] which can be utilized as a strategy to deal with delays and possible deadline violations [27]. For all these approaches, the strategy how the instances are processed from the queue is static, i.e., fixed at runtime. Static strategies typically range from First-In-First-Out to rule-based batching and scheduling as proposed by [16]. However, due to the complexity of parameters possibly relevant for the queuing processing strategy, it is quite difficult to foresee dynamic behavior at runtime and therefore hampers the definition of strategies at design time. It might be even more difficult for users to take control of queuing strategies at runtime. Hence, it would be desirable to offer strategies that enable automatic queuing strategies defined at runtime. We refer to such strategies as *dynamic queuing*.

In this paper, we present a dynamic queuing approach based on artificial intelligence techniques to reduce the processing time at critical activities. The core idea is that similar instances in a row can be processed faster than randomly distributed instances making use of optimizations such as caching or the gaining of routine by humans. This idea has been mainly addressed as batching in combination with queuing [16, 27] so far. In this paper, we refine batching towards clustering into multiple queues. More precisely, the queuing is split into two phases: first of all, clustering techniques are applied to the instances arriving at a critical activity. For clustering different existing algorithms are applicable. We assume clustering based on numerical attributes of process instances such as print form, size, and quantity. With OPTICS [2], for example, a clustering algorithm is available that does not require any a-priori knowledge on the number of clusters which provides flexibility to react on the particular runtime setting the queuing takes place. The resulting clusters determine a corresponding set of queues from which the instances to be processed together are selected. This requires a state management for allocating the appropriate number of resources during runtime. In this paper, we base the state management on a configurable performance index which incorporates optimization factors time and cost. However, by extending the performance index, further factors such as satisfaction can be taken into consideration as well. The dynamic queuing approach is implemented and evaluated based on a case study on a daily observation of print jobs within a hospital. We will show that dynamic queuing results in a reduction of processing times. We will also discuss the different possibilities to ap-

SAC'13 March 18-22, 2013, Coimbra, Portugal.

ply and extend the approach presented in this paper by, for example, taking into consideration also human resources and semantic clustering strategies.

The paper is structured as follows: In Section 2, we present an introduction into the application of queuing in Process-Aware Information Systems (PAIS). Section 3 describes the core idea of the dynamic queuing approach.

2. QUEUING IN PROCESS-AWARE INFORMATION SYSTEMS

Queuing has been well researched in mathematics. The latest models can cover a variety of factors including uncertainty. Most important parameters of any model are the average arrival times and processing times, the number of resources and the capacity. Little's theorem [15] allows a relation between the long-term average number of instances, the arrival rate, and the average sojourn time. Optimization of queues is basically achieved by estimating probability distributions of incoming elements or processing times. However, in Process Aware Information Systems, the prediction of parameters like arrival times is often impossible due to the dynamic environment the instances are executed in. Therefore adaption of mathematical optimization to Process Aware Information Systems is limited in ways of flexibility. However mathematical theorems may serve a basis for dynamic instance queuing.

In Process Aware Information Systems, queues can be considered as complex artifact. Normally, queues are not scheduled to arise at design time. Modeling notations such as BPMN do not include shapes for queuing constructs as well. However, queues might occur during runtime, i.e. at the instance level. They emerge, if a resource does not have the capacity to handle all instances in time. The activity linked to this resource is also anticipated a *critical activity*. If queues arise unpredicted, they can have significant impact on the overall process success. This makes queue handling a very complex task for the process designer.

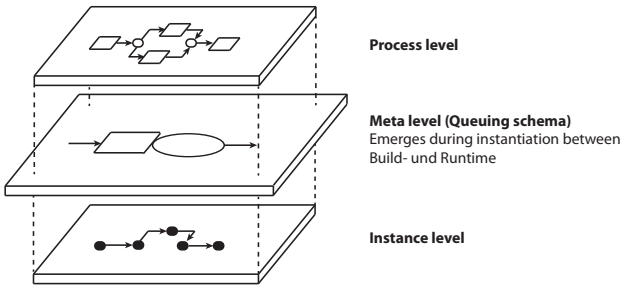

Figure 1: Layers of abstraction within a PAIS

Figure 1 shows a schema of the relation between process level and instance level. Queues are not designed at process level, but arise during the instantiation. Queuing models can therefore be considered as a meta level between process and instance schema. Queues emerge at runtime and disappear automatically, when the peak of incoming instances is released. The processing of instances can happen in different ways: Most common is the First-In-First-Out logic, which means that instances are processed in the same order as they arrived at the queue. However, different techniques, e.g. Last-In-First-Out, can be applied as well. The main point is that existing approaches that apply queuing in Process Aware Information Systems [26, 16] are merely *static*, i.e., the processing strategy for process instances within the queue is fixed. Contrary, in this paper, we introduce a *dynamic queuing* approach that leaves the processing strategy to the runtime based on the current facts, i.e., number and attribution of instances arriving at the critical activity.

3. DYNAMIC INSTANCE QUEUING

In this section, we will first present the general structure of the dynamic queuing approach, followed by a more detailed elaboration on the different phases of dynamic queuing. Highlights of implementing the dynamic queuing approach conclude this section.

3.1 General Approach

Figure 2 shows the basic concept of dynamic instance queuing. The core idea is that similar instances can be processed faster in a row by making use of optimizations such as caching or gaining of routine by human resources than randomly distributed instances. A similar idea is pursued by combining queuing with batching [26, 16]. However, first of all, batching requires that the resources are able to process batches, i.e., the parallel processing of a set of instances. Secondly, batching at one critical activity often results in batch-wise arrival of instances at preceding activities in the process that are not designed for batch processing. Hence, a reduction of processing time by batching at one critical activity might cause subsequent queues and subsequently lead to no reduction or even increase of the overall processing time. Therefore, in this paper, we adopt a *sequential* processing strategy rather than a batch-based one. More precisely, even though instances are grouped together based on similar features, they are not processed as batches but within separated queues.

Getting back to the overall concept as displayed in Fig. 2: incoming instances at a critical activity are at first collected and, when the first resource shifts to idle state due to a lack of instances to be processed, clustered into groups (step 1). The instance clusters are then transferred to the buffers of the queuing system where each buffer represents one instance cluster (step 2). The instances from one buffer are then processed at once by the resource (step 4).

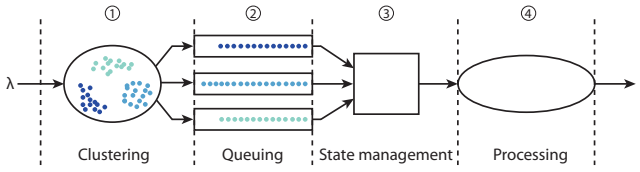

Figure 2: Visualization of dynamic instance queuing

The processing logic remains the same during the whole workflow life cycle. This includes the handling of the initial elements: When the first instance arrives, the resources are in idle state. Hence, clustering is executed instantly. It is not performed earlier since clustering works best with the highest possible number of elements. The emerging cluster consisting of this single instance will be transferred to one of the resources and processed momentarily. No waiting

times arise, the throughput time and processing time are the same. With respect to the layers of abstraction (Figure 1), no queue at the instance level arises. This is intuitively understandable, since queuing is necessary only if more instances are arriving than the resources are able to process.

3.2 Instance Clustering and Queuing

The clustering component classifies incoming elements into groups of similar instances based on certain attributes the instances possess. In the printing example, such instance attributes might be print form (letter, poster, flyer), print job size, or coloring. In the health care domain, more exactly in a laboratory process, instances reflecting the processing of samples on one laboratory apparatus could be clustered based on instance attributes analysis technique or sample size. The emerging instance classification is mapped onto the buffers and, finally, onto the resources, i.e., there will be a 1:1 mapping between clusters and buffers (resources). By classifying the instances in a logic way, the algorithm uses the fact that processing times are not independent by means of probability theory: Their processing times decrease when similar instances are handled sequentially. Saving potentials arise both from lower processing times by humans when working on similar instances and from computer technologies such as caching.

The classification is based on a clustering mechanism. Rule based systems are not applicable since they require specifications during design time. Decision learning trees or theorems like support vector machines require a training set. Only clustering algorithms meet the flexible character of dynamic instance queuing. However, there is a variety of clustering approaches that can be considered for the implementation. Basically, there are two kinds of clustering algorithms: In centroid clustering methods, groups are represented by a central vector. They base on a given order of objects and thus a fixed number of clusters. Using a replacement algorithm, the individual elements are exchanged until a certain target criteria is met. Density based clustering methods separate a set into areas of higher and lower density. Some objects are considered to be more related to nearby objects than to objects farther away.

We propose the use of a density based clustering method since a previous specification of the numbers of clusters is not needed. One of the modern clustering methods is the OPTICS algorithm (Ordering Points To Identify the Clustering Structure) [2]. It is based on the DBSCAN approach [13], but can also identify clusters of different densities. The number of clusters must not be known a-priori. This constitutes another tribute to the dynamic character of the proposed approach since fixing a number of clusters would have be done at design time or by the user during runtime. Both options are less favorable than the automatic detection by the OPTICS algorithm. Further, having a complexity of $O(n^2)$ where n corresponds to the number of objects / instances to be clustered, the OPTICS algorithms performs efficiently. However, since the overall approach is configurable, other clustering techniques can be chosen.

Algorithm 1 summarizes steps 1 and 2 of the overall approach as depicted in Figure 2, i.e., the clustering of arriving instances at a critical activity and the subsequent distribution onto buffers. Buffers serve as data structures to store the instances between clustering and processing. The number of buffers is determined by the number of clusters of the previous step, hence the number of buffers may vary over time. We impose neither a lower nor an upper bound to the number of clusters in order to achieve the best performance of the dynamic clustering approach. The order in which the instances are queued within the buffers is not relevant for dynamic clustering. Nonetheless, one could envisage a certain priority order on the buffers to work well with the later processing behavior of the resources.

Algorithm 1 Clustering and Queueing

```
1: while Process instances are active do
2:     while Buffers are not empty do
3:         gather arriving instances in dataset d;
4:     end while
5:     C := Cluster(d)          ▷ In our approach Cluster(d)
   invokes the OPTICS algorithm;
6:     for cluster c ∈ C do
7:         select free buffer b;
8:         move instances from cluster c to buffer b;
9:     end for
10: end while
```

3.3 State Management

The dynamic state management system is responsible for the allocation of the proper number of resources during runtime, i.e., we determine the optimal number of resources to be assigned from the overall number of available resources. The decision on the number of resources is a trade-off between time, costs, flexibility and quality of service. Especially time and costs appear to be at conflict. Therefore, a decision function is needed. We base the selection on a scalable performance index (formula 1) respecting the individuality of application domains.

$$pi(o) = (\alpha \cdot E(B) \cdot n_i) \cdot (\beta \cdot C_{o,t} \cdot o \cdot E(B)) \qquad (1)$$

In formula 1, the performance index pi in respect of the number of resources o is shown. The decision function represents the trade-off between time and costs, where $E(B)$ is the estimated average processing time and $C_{o,t}$ are the costs per resource and time unit. The equation is intuitively reasonable: The estimated processing time per instance $E(B)$ multiplies with the number of instances from one iteration. On the other hand, the costs $C_{o,t}$ per resource and time unit multiplies with the number of resources o and the actual processing time $E(B)$. α represents an individual factor for the importance of the time, β for the costs. Remember that $\alpha + \beta = 1$ and $\alpha, \beta \geq 0$ must be fulfilled. In this decision function, lower values represent a better performance.

The performance index can be considered a scalable decision function for the estimation of the proper number of resources during runtime. It is applicable on most of the application scenarios. However, different decision functions can be designed, if necessary.

In the overall concept, dynamic state management is situated as follows: After clustering and queuing instances within the buffers, the dynamic resource allocation algorithm decides on the number of resources assigned for processing the instances within the current iteration. It selects between three possible alternatives based on the output of the performance index $pi(o)$, i.e., keeping the same number

of resources as in the previous iteration, decreasing, or increasing the number. In the latter two cases, the number of resources is decreased (increased) as long as a performance gain is achieved (as shown in lines 5-14 of algorithm 2). Based on the monotony properties of the performance index, decreasing / increasing will always lead to an optimum, i.e., no local minimum / maximum will be reached. Further exactly one of the loops will be executed per iteration. Having successfully processed Algorithm 2, variable *o* represents the appropriate number of resources to be allocated. The variables for the performance index (formula 1) are derived from the iteration before which enables an immediate response on changing process environment parameters.

Algorithm 2 Dynamic resource allocation

```
Require: int o; double li0, li-, li+;
 1: li0 = pi(o);
 2: li- = pi(o-1);
 3: li+ = pi(o+1);
 4:
 5: while li- ≤ li0 do ▷ Decreasing the number of resources
 6:     li0 ⟵ li-;
 7:     o ⟵ o - 1;
 8:     li- = pi(o - 1);
 9: end while
10:
11: while li+ ≤ li0 do ▷ Increasing the number of
    resources
12:     li0 ⟵ li+;
13:     o ⟵ o + 1;
14:     li+ = pi(o + 1);
15: end while
```

3.4 Temporal Concurrency Approach

In Figure 2, one iteration of the dynamic instance queuing approach was shown. One iteration contains the collecting of arriving instances as long as all resources are busy, followed by the clustering step and finally the processing of the instances. However, during runtime, iterations overlap since new instances are arriving steadily.

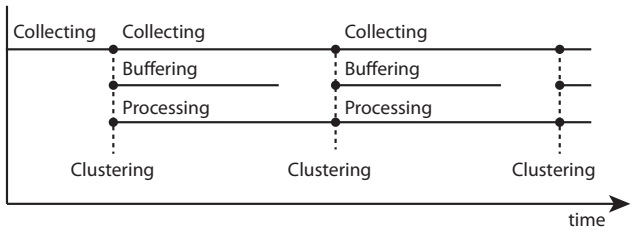

Figure 3: Visualization of dynamic instance queuing

Figure 3 shows the concurrency concept of our dynamic instance queuing approach. At first, arriving instances need to be collected for a certain time. The classification of instances represents a point in the timeline, visualized as the dark dot. From then on, the classified instances are buffered and, concurrently, the processing of the first instances from the buffers begins. The process of buffering is finished, when all its containing instances are moved to a resource and hence, all buffers are empty. For that reason, the processing step always takes longer that the process of buffering. One iteration is finished, when the processing of all instances is finished.

Note that iteration 2 begins right at the time point of clustering from the iteration before, since instances are arriving concurrently. As the point of clustering is dependent from the capacity utilization of the resources, at most two iterations are running concurrently at any point during runtime. Dynamic instance queuing ends when the overall process is finished.

4. SIMULATION AND EVALUATION

In this section, the application of dynamic queuing in a real-world scenario is described. For this, we focus on a specific task referring to a print job. This task is embedded in a complex Process Aware Information System (PAIS). Due to restrictions in scope, we focus on the subprocess which will serve as the scenario for the implementation of our dynamic instance queuing approach.

4.1 Application scenario

The application of dynamic instance queuing has been simulated in a real-world scenario from the health care domain. The simulation setup covers the printing management in a hospital. In this hospital, a department of 15 to 40 employees shares two common network printers. Each print job is sent to one of the two printers, depending on the circumstances of the load. Print jobs though differ in a bunch of characteristics: The number of pages, e.g., the color mode, the sheet size, or a special configuration for slide-printing.

A change in one of the modes as well as a change in the sheet size implies changeover times that need to pass until printing can begin. The so called preparation time is therefore based on different variables. Waiting times in the queue and the actual printing time affect the sojourn time as well. Since in some extent, more jobs are triggered than the printers can handle, queues emerge. In this scenario, print jobs represent instances and printers are resources.

Until now, the hospital uses a First-In-First-Out logic to process the print jobs. That means that the jobs are printed in the same order as they arrived in the queue. Dynamic instance queuing is simulated based on a dataset that contains any information (trigger times of print jobs, arrival times at the printer, preparation times, configuration of the print job) about the Is-process. That way, a thoroughly reproducibility is given. Based on this dataset, the same scenario is simulated using the dynamic instance clustering approach.

4.2 Scenario analysis

The given dataset includes any information needed on the application scenario. We consider one working day as a characteristic model for the overall printing management in the hospital. That day, the first print job is triggered at 6:13:27 a.m. and the latest one is given at 21:48:27 p.m. Throughout the day, 273 print jobs with a total of 4717 pages were printed.

Figure 4 shows the average time line of one instance passing the printing process. The average sojourn time is 149,9 seconds. 46,1 seconds are allotted to waiting time, i.e. the interval between triggering a print job and the arrival at the printer. Preparation times such as changeover procedures between color or slide modes take an estimated time of 5,0 seconds. The rest of the sojourn time is considered to

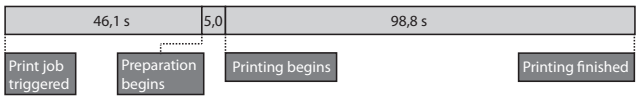

Figure 4: Average subtimes

be the "actual" printing time. Note that waiting times and changeover times might be decreased by optimization, while the printing time is static and can't be altered.

An estimated sojourn time of 149,9 seconds theoretically would be enough to handle all print jobs without any waiting times. This would require an equal distribution of incoming print jobs. In fact, the arrival time of print jobs is not equally distributed, as an analysis of the arrival times shows: Throughout the working day, there are two peaks of incoming instances in the morning and in the afternoon. Between 9.00 a.m. and 9.30 a.m., for example, 35 new print jobs arrive. This exceeds the capacity of the two existing resources and therefore, waiting times emerge. A queue arises, if more than two instances are in the system.

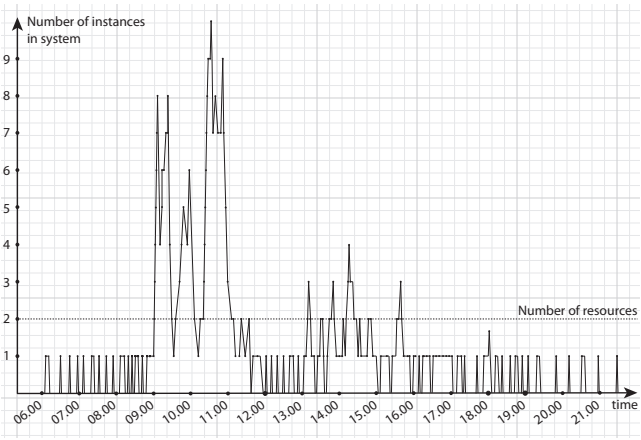

Figure 5: Number of instances in the system

The number of instances in the system at any time within the period under consideration is shown in figure 5. The print jobs that are currently processed are included in the number of instances. It appears that exceeding a critical number of arriving instances implies heavily increasing queues. Waiting times increase considerably. On the other hand, dynamic instance queuing offers potentials to reduce changeover times by grouping similar instances. A decrease in changeover times therefore implies multiple positive effects on the waiting time.

4.3 Implementation and simulation results

Dynamic instance queuing was simulated under the exact same parameters. That way, full comparability is ensured. The algorithm was implemented in Java using high level concurrency techniques (as shown in figure 3). The centerpiece of the dynamic instance queuing implementation is a QueuingSystem class that is responsible for the initialization and coordination of all components. The threads for existing resources are managed by a ResourceManager class which offers functions to assign the buffers to the resources and to adapt the number of active resources as calculated by the decision function from the state management system. Buffers are similarly managed by a BufferManager. Since it is a real time simulation, the runtime environment does not have impact on the performance of the simulation. As proposed before, the OPTICS algorithm was chosen for clustering. The state management component is based on a decision function similar to the one explained in section 3.3.

From a temporal point of view, the results are positive: The average changeover time declines by 14%. In total, this means a decrease from 1347 to 1186 seconds (figure 6). It is apparent that most of the time savings emerge during the peaks in the morning and the afternoon. As supposed, the decrease in changeover times has also massive impact on the waiting times: Between 9.00 a.m. and 11.30 a.m., dynamic instance queuing achieves a saving from around nine minutes, based on the time the printers are active.

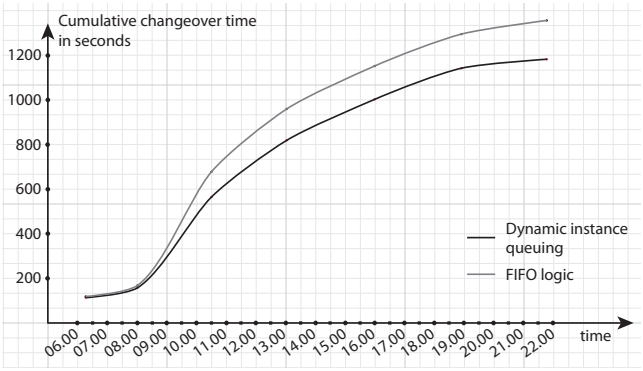

Figure 6: Cumulative changeover times

The results from the dynamic state management are positive as well. Since two printers are available, the decision function (formula 1) allocates the estimated number of resources and switches between three states (no printer active, one printer active, parallel activation of both printers). The decision function is stable against deviations: In fact, the state is changed only five times throughout the period under consideration. Peaks were determined quite exactly. Compared to the existing approach which is based on an automatic stand-by service with fixed idle times, dynamic state management offers savings from around 7%. This means lower costs (energy, supplies) and less attrition.

4.4 Interpretation

The theoretical and practical application of dynamic instance queuing allows a further performance assessment. [22] evaluate the overall performance of a process by means of four criteria: Time, costs, flexibility and quality of service. Our concept allows a reduction of throughput times in a considerable extent. The decrease is based on the classification of elements to groups of similar instances which offers potentials to reduce the processing times. Since the worst case of the clustering is a random distribution, dynamic instance queuing never achieves a lower performance than queues using a First-In-First-Out logic. The state management is a dynamic scheduling mechanism that offers potentials to reduce costs. Based on a decision function, the algorithm allocates the proper number of active resources during runtime. In most cases, dynamic state management achieves better results than static rules which have to be defined at design

time and cannot adapt on the events during runtime.

Flexibility is an asset of dynamic instance queuing. Specifications are not needed to be made a-priori. For that reason, our concept can be easily introduced in a wide field of application scenarios. Since the algorithm does not require adaptions in other parts of the process, universal applicability is given.

The actual performance difference between dynamic instance queuing and static queuing approaches is based on the instances of the specific application scenario. In general, a high number of instances is always an asset. The quality of classification is dependent from the underlying clustering algorithm, which supports instances with numerical attributes best. Dynamic instance queuing works best in scenarios with an improper ratio between number of instances and resource capacity. The more instances are waiting, the better results can be obtained. For that matter, dynamic instance queuing should be considered in application scenarios where the process environment is dynamically changing, important process specifications, i.e. arrival distributions and capacity development, can't be estimated in an adequate quality, and economic or temporal realities don't allow to develop, test and implement complex static rule systems.

In these scenarios, dynamic instance queuing with its capability to adapt dynamically to the process environment offers potentials to reduce time and costs. Furthermore, it achieves higher flexibility and a better quality of service.

5. DISCUSSION

Our approach for dynamic instance queuing meets several requirements. It is applicable to any Process-Aware Information System since it is a self-contained component that does not imply the need for adaptations in other parts of the process. The optimization takes place during runtime, so no previous specifications have to be made. In fact, every process is different and has unique characteristics. Dynamic instance queuing therefore represents a framework that can be adapted on the specific circumstances of the application scenario: Individual decision functions might be defined that support the trade-off between the particular variables of the workflow comprehensively. Moreover, other clustering approaches than OPTICS algorithm might be chosen in order to cope the instance attributes from the process best. That way, universal applicability is guaranteed without neglecting the individuality of each workflow scenario.

The following assumptions have been made for this paper (a) clustering based on numerical process instance attributes, (b) optimization of process times, (c) resource behavior is fixed.

The case study and its results described in Sect. 4 refer to an ideal scenario with respect to the above assumptions, i.e., the resource is a printer for which the behavior is fixed, the processing times of the instances are reduced, and the instances can be clustered based on numerical attributes such as form or job size. As we know from other case studies, this kind of scenario can often be found in practice, e.g., processing samples on a laboratory apparatus with probably high changeover times. However, the assumptions still leave room for further investigations. Promising directions are shortly discussed below.

1. *Extension of Process Application Settings:* Our concept is also applicable on multi queue and multi server scenarios. At the moment, we are working on a simulation in the industry sector. The scenario is about a number of variants n of a product that is being manufactured on several production plants. This is typically considered a mathematical problem. In fact, Dynamic Instance Queuing can be implemented supplementary to Operations Research algorithms: While Operations Research approaches strive to determine the ideal allocation at designtime, our dynamic instance queuing approach is able to optimize during runtime within the specifications of the Operations Research algorithm.

2. *Extension of resource behaviour:* Our concept doesn't address different resource behaviour in performing certain work items so far. Different resources may have different specializations which may render them more or less suitable to performing certain activities. At the moment, all resources are considered the same. Assuming that clusters represent collections of similar process instances, one need to find a proper mapping between resources and clusters. Keeping the dynamic character of our instance queuing approach alive, this mapping might be implemented using a decision tree learning algorithm. We will enhance our concept in this area.

3. *Semantic clustering:* It cannot always be assumed that process instances are distinguishable based on numerical attributes such that "good" input for clustering algorithms such as OPTICS is provided. This becomes particularly true if the process instances to be clustered and queued reflect subjects such as patients or customers rather than objects such as print jobs or samples. Whereas similarity between process instances can be determined for objects, e.g., the same print job, the same kind of sample, this can become much more difficult for subjects. Hence, in order to be able to apply similar techniques to the one proposed in this paper, we have to come from a clustering purely based on numerical attributes towards a semantic clustering with respect to the process instances. Current proposals on semantic clustering algorithms [14] are promising and we will investigate them together with similarity measures for processes [3] from the BPM community.

4. *Integration of further optimization factors:* Aside time (and costs), related work (cf. Sect. 6) mentions further optimization factors such as quality, flexibility, or customer satisfaction [22]. Clearly, it is natural that a queuing approach first tackles processing times by reducing waiting times. However, the optimization of other factors could be considered by adjusting the performance index (formula 1).

5. *Integration of human resources:* Finally, and with respect to the current trend towards human-oriented PAIS [7], the behavior of human resources might be very interesting in connection with optimization techniques such as queuing. In other words, in addition to considering humans being reflected by process instances, also human resources might open new perspectives, and both aspects are most likely to be treated in combination. Think for examples of full waiting room of a pediatrician clinics. First of all, it would be interesting to see whether dynamic queuing might lead to

reduction of the overall waiting time of all patients. Secondly, the satisfaction of single patients as well as of the pediatrician have to be analyzed as well. Here, research on the psychology of waiting lines [17] constitutes a valuable input for our further research.

6. RELATED WORK

Queuing in Process-Aware Information System: Queuing is part of basically any PAIS, mostly implicitly, sometimes explicitly. Existing approaches, as described in [26], cover arising queues as result of an imbalanced ratio between available resources and the number of process instances to be handled by the PAIS. Explicit queuing is addressed by Liu and Hu [16], who apply dynamic batch processing to Workflow Management Systems. [27], however, understands queues as a mean to handle escalations in PAIS.

Dynamic Queuing in Other Areas: Having its foundation in mathematics, queuing has been adopted to numerous areas in information technology. The most common application scenario is the computer processor's operation of processing batch tasks. The operating system provides a logic to handle a sequence of tasks to be processed. In this area, optimizations are achieved by improving the classification system to compute the ideal temporal order of pending tasks.

Queuing in messaging systems and middleware: Kumar et al. [8] describe a novel self-adaptation algorithm that has been designed to scale efficiently for thousands of streams and aims to maximize the overall business utility attained from running middleware-based applications. In a subsequent work, Kumar [9] enhances his approach by a dynamic element to react on the resources available ("resource awareness"). The approaches culminate in a distributed stream processing middleware that provides sharing-aware component composition [23]. In this approach, optimizations are achieved a-posteriori by implicitly evaluating each iteration. Our approach, however, strives to achieve optimizations during runtime. Regarding middleware systems, load balancing is a major topic. As load is represented by a queue of tasks, dynamic processing strategies are a possibility for optimization. An exemplary approach is presented by Drougas [4] as well as by [25]. Amini et al. [1] describe an algorithm that is designed to meet the challenges of extreme-scale stream processing systems, where over-provisioning is not an option, by making the best use of resources even when the proffered load is greater than available resources. This scenario is similar to the approach presented in this paper in the way that the need for dynamic instance queuing also arises when a resource does not have the capacity to handle all instances in time. However, dynamic instance queuing is a process oriented approach, while load balancing has kind of a static character by definition as it is a methodology that applies typically in multilayer architectures.

Process Instance Queuing in (Commercial) PAIS: The technical prerequisite for queuing process instances is the system-based provision of synchronization between multiple process instances. Different patterns for synchronizing multiple instances of one activity in PAIS (12 - 15, 34 - 36) have been described by the workflowpatterns.com initiative. Synchronizing multiple process instances at runtime can be conceptually put down on Pattern 36 (Dynamic Partial Join for Multiple Instances). According to the evaluation provided on workflowpatterns.com, there are no commercial or academic tools that support any implementation of Pattern 36. The only process engine that provides direct support instance synchronization is the CPEE (http://cpee.org/) [18]. Here we aim at an integration with the queuing approach presented in this paper.

Time aspects in PAIS: Since time aspects constitute a major challenge for modeling and execution of business processes different approaches address this issue [5, 6, 11, 12, 20, 24, 21]. It has been investigated, for example, how to capture time aspects at design time, in particular, to cover uncertainty of processing times and to determine critical paths. At runtime, adherence of the process instances to imposed time restrictions such as deadlines is monitored. Escalation strategies [27] provide means to deal with violations of time restrictions and deadlines. All these questions become more challenging when considered for process choreographies [6]. An analysis of existing approaches based on time patterns can be found in [10]. The work presented in this paper does not directly relate to the above mentioned approaches. Since the presented approach reduces the processing time for all process instances within a certain time frame, it might be even counterproductive for handling possible deadline violations of single process instances, but is more suited for optimizing the processing time of a set of process instances being executed within the same time frame.

Process analysis and optimization: Techniques for analyzing and optimizing business processes are of particular interest for business process re-engineering. The core challenge when optimizing business process is to find the redesign strategies. As described in [19], in practice, they are often engineered within an expert workshop. For different reasons it would be more beneficiary to provide strategies that can be applied in certain situations in order to support the user. In [22], an overview of existing approaches on best practices or heuristics for business process redesign is provided that address different optimization factors such as time, costs, quality, or flexibility. This paper narrows the optimization factors down to time (and as a side-effect probably costs). As discussed in Sect. 5, however, more factors can be included into the considerations. However, the main difference is that this approach enables an automatic optimization of throughput times at runtime that do not require any redesign measures.

7. SUMMARY & OUTLOOK

In this paper we proposed an approach for dynamic instance queuing in Process Aware Information Systems that leverages on reducing execution time at critical activities by processing arriving instances together in groups that possess similar attributes, e.g., print jobs. The novel contribution is to automatically determine and distribute the instance clusters to resources at runtime. The dynamic queuing algorithm utilizes clustering algorithms and a state management strategy for instance distribution to resources that is based on a configurable performance index. The algorithm was evaluated based on a realistic printer data set observed during one day in a hospital. We simulated the print job processing when applying dynamic queuing and compared the results with the actual data. Overall, a reduction of 14% of the cumulated processing time could be achieved. Several extensions of the presented approach have been discussed including the incorporation of human resources, semantic clustering algorithms, and further optimization factors such as satisfaction or flexibility.

In future work, we will analyze these different extensions in detail. Specifically, we are interested in the semantic clustering of process instances and the integration of human resources. The latter constitutes a promising bridge to projects on human-centered Process Aware Information Systems. Further on, we will integrate the dynamic clustering algorithm with our process execution engine CPEE (http://cpee.org/), in particular utilizing the synchronization component at critical activities.

Furthermore, runtime optimization based on our dynamic instance queuing approach is considered in the ADENTURE EU project (fp7-adventure.eu) aiming to run processes more efficiently in virtual factories.

ACKNOWLEDGEMENTS

This work was partially supported by the Commission of the European Union within the ADVENTURE FP7-ICT project (Grant agreement no. 285220).

8. REFERENCES

[1] L. Amini, N. Jain, A. Sehgal, J. Silber, and O. Verscheure. Adaptive control of extreme-scale stream processing systems. In *In ICDCS 2006*, pages 71–79, 2006.

[2] M. Ankerst, M. M. Breunig, H. Kriegel, and J. Sander. OPTICS: ordering points to identify the clustering structure. *SIGMOD Rec.*, 28(2):49–60, 1999.

[3] R. Dijkman, M. Dumas, B. v. Dongen, R. KÃd'Ãd'rik, and J. Mendling. Similarity of business process models: Metrics and evaluation. *Information Systems*, 36(2):498–516, 2011.

[4] Y. Drougas, T. Repantis, and V. Kalogeraki. Load balancing techniques for distributed stream processing applications in overlay environments. In *Int'l Symposium on Object and Component-Oriented Real-Time Distributed Computing*, pages 33–42, 2006.

[5] J. Eder, E. Panagos, and M. Rabinovich. Time constraints in workflow systems. In *Proc. Int'l Conf. on Advanced Information Systems Engineering*, pages 286–300, 1999.

[6] J. Eder and A. Tahamtan. Temporal conformance of federated choreographies. In *Database and Expert Systems Applications*, pages 668–675. 2008.

[7] S. Kabicher-Fuchs and S. Rinderle-Ma. Work experience in PAIS - concepts, measurements and potentials. In *In Proc. Int'l Conf on Advanced Information Systems Engineering*, pages 678–694, 2012.

[8] V. Kumar, B. Cooper, and K. Schwan. Distributed stream management using utility-driven self-adaptive middleware. In *Int'l Conf. on Autonomic Computing*, pages 3 –14, 2005.

[9] V. Kumar, B. F. Cooper, Z. Cai, G. Eisenhauer, and K. Schwan. Resource-aware distributed stream management using dynamic overlays. In *In Proc. Int'l Conf. on Distributed Computing Systems*, pages 783–792, 2005.

[10] A. Lanz, B. Weber, and M. Reichert. Workflow time patterns for Process-Aware information systems. In *Enterprise, Business-Process and Information Systems Modeling*, volume 50, pages 94–107. 2010.

[11] H. Li and Y. Yang. Dynamic checking of temporal constraints for concurrent workflows. *Electronic Commerce Research and Applications*, 4(2):124–142, 2005.

[12] J. Li, Y. Fan, and M. Zhou. Timing constraint workflow nets for workflow analysis. *IEEE Trans. on Systems, Man, and Cybernetics*, 33(2):179–193, Mar. 2003.

[13] J. Li, L. Han, S. Zhen, and L. Yao. Soil moisture content error detection based on DBSCAN algorithm. In *Advances in Computer Science, Intelligent System and Environment*, volume 106, pages 85–89. 2011.

[14] T. Lippincott and R. Passonneau. Semantic clustering for a functional text classification task. In *Computational Linguistics and Intelligent Text Processing*, pages 509–522. 2009.

[15] J. D. C. Little. A proof for the queuing formula: L= λw. *Operations Research*, 9(3):383–387, 1961.

[16] J. Liu and J. Hu. Dynamic batch processing in workflows: Model and implementation. *Future Generation Computer Systems*, 23(3):338 – 347, 2007.

[17] D. Maister. The psychology of waiting lines. *The service encounter*, 1:13–23, 1985.

[18] J. Mangler and S. Rinderle-Ma. Rule-based synchronization of process activities. In *2011 IEEE 13th Conference on Commerce and Enterprise Computing (CEC)*, pages 121–128. IEEE, 2011.

[19] M. Netjes, I. Vanderfeesten, and H. Reijers. "intelligent" tools for workflow process redesign: A research agenda. In *Business Process Management Workshops*, pages 444–453. 2006.

[20] 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*, pages 545–553. 2006.

[21] H. Pichler, M. Wenger, and J. Eder. Composing Time-Aware web service orchestrations. In *Advanced Information Systems Engineering*, pages 349–363, 2009.

[22] H. A. Reijers and S. Liman Mansar. Best practices in business process redesign: an overview and qualitative evaluation of successful redesign heuristics. *Omega*, 33(4):283–306, Aug. 2005.

[23] T. Repantis, X. Gu, and V. Kalogeraki. Synergy: Sharingaware component composition for distributed stream processing systems. In *In Middleware*, pages 322–341. Springer-Verlag, 2006.

[24] S. Sadiq, O. Marjanovic, and M. Orlowska. Managing change and time in dynamic workflow processes. *IJCIS*, 9(1&2):93–116, 2000.

[25] R. Strom, C. Dorai, G. Buttner, and Y. Li. Smile: distributed middleware for event stream processing. In *Int'l Conf. on Information processing in sensor networks*, pages 553–554. ACM, 2007.

[26] W. van der Aalst and K. van Hee. *Workflow Management*. MIT Press, 2002.

[27] W. M. van der Aalst, M. Rosemann, and M. Dumas. Deadline-based escalation in process-aware information systems. *Decision Support Systems*, 43(2):492–511, Mar. 2007.