Dynamic Resource Allocation for Executable BPMN Processes Leveraging Predictive Analytics

Yliès Falcone, Gwen Salaün, Ahang Zuo

Dynamic Resource Allocation for Executable BPMN Processes Leveraging Predictive Analytics

Yliès Falcone, Gwen Salaün, and Ahang Zuo
Univ. Grenoble Alpes, CNRS, Grenoble INP, Inria, LIG, F-38000 Grenoble, France

Abstract—Resource allocation is a critical problem in business processes due to the simultaneous execution of tasks and resource sharing among them. The number of allocated resources affects both the execution cost and time of the process. In the context of runtime processes, a well-defined resource allocation strategy is essential for optimising waiting times and costs by mitigating delays and enhancing resource utilisation. This paper introduces a novel approach to dynamically adjust resource allocation during the execution of BPMN (Business Process Model and Notation) processes. The BPMN process is monitored in real-time, and the execution traces produced during its multiple executions are analysed. These execution traces are used to compute various properties or metrics of interest, including resource usage and average execution time. The approach then relies on predictive analytics to compute the future values of the aforementioned metrics. Based on these predicted results, strategies for the dynamic allocation of resources are defined, which anticipate changes in resource usage and thus dynamically update the number of resources in advance. This approach is fully automated using a toolchain and has been validated with multiple examples.

Keywords–Business Processes; BPMN; Monitoring; Resource Allocation; Quantitative Analysis; Predictive Analytics.

1. Introduction

A business process is a sequence of tasks executed by some specific resources (e.g., an employee, a machine, or a computer) to accomplish a specific objective [1]. When allocating resources to process tasks, it is essential to consider the specific time, cost, and quality goals associated with each business process [2]. The Business Process Model and Notation (BPMN) is a standard modelling language for business processes, maintained by the Object Management Group (OMG) [3]. Each process modelled with BPMN is usually executed multiple times, resulting in multiple instances running simultaneously. It is essential to address conflicts arising from concurrent processing of several processes/instances, which can result in competing requests for the same resources and imbalances in the allocation of resources. Failure to address these conflicts and imbalances may impede efficient resource management [4]. Due to the concurrent execution of tasks and the sharing of resources, allocating resources is a crucial issue in business processes, which can also be seen as a decision-making problem for executable processes [5]. Therefore, it becomes critical to specify an adequate allocation strategy, as it influences the efficient execution of processes. In other words, an appropriate allocation of resources allows the reduction of the process's associated cost and execution time. Resource allocation aims to ensure that each business process task is carried out with the appropriate resources and timing [6].

Predictive analytics techniques rely on traditional machine learning models such as Hidden Markov Model (HMM) [7], Decision Trees, Random Forests, Bayesian model [8], etc. Since the development of deep learning, predictive analytics techniques have been applied to different fields, especially in areas like computer vision and natural language processing. Similarity, these techniques can also be applied to business process analysis. The deep learning models are instrumental in predicting outcomes such as the next activity [9], [10] or the next event [10]. Predictive analytics techniques are based on specific input data and are trained to produce a predictive model that can perform the prediction task. Furthermore, predictive analytics techniques can also be combined with resource allocation strategies. In that case, the prediction model permits a better resource allocation, thus avoiding low usage or lack of resources. This influences the time for executing specific tasks and, therefore, the time for completing the entire process.

This paper proposes a novel approach for dynamically allocating resources to executable BPMN processes at runtime. The approach requires an executable BPMN process as input, where each task explicitly specifies the required resources and its duration. The process is executed multiple times, resulting in the generation of multiple instances. These executions are monitored, and execution traces can be automatically retrieved. Each trace keeps track of all tasks executed by each process instance. These execution traces are then used to compute different key metrics/properties of the process (e.g., average execution time, resource usage, total cost). The results of these real-time computations are used as input to a resource allocation algorithm, which aims at dynamically adjusting the number of each resource. Several strategies are possible for this algorithm. In this paper, we introduce a strategy relying on a predictive model for computing the aforementioned metrics, which allows us to anticipate the change in resources and thus improve these metrics by reducing process execution time, costs associated with resources and have a more balanced usage of resources.

To summarise, this paper provides several contributions:

- Runtime quantitative analysis techniques for executable BPMN processes.
- A generic approach for dynamic resource allocation.
- The integration of predictive analytics to enhance both the

applicability and quality of our approach.
- A toolchain implementing all the steps of our approach and its validation on several examples.

The remainder of this paper is organised as follows. Section 2 provides an introduction to the BPMN notation employed in this study. Section 3 and Section 4 outline the approach to perform the dynamic allocation of resources and the predictive analytics component. Section 5 describes the tool we implemented to automate all the steps of the proposed solution. Section 6 illustrates the approach using a specific case study. Section 7 presents related work and Section 8 concludes.

2. Models

In this section, we describe preliminary concepts. Section 2.1 introduces the basics of BPMN. Section 2.2 explains how resources are described in processes. Section 2.3 presents the execution traces resulting from the multiple executions of a given process. Section 2.4 describes the properties based on time and resources that we evaluate at runtime.

2.1 BPMN

BPMN is the standard notation for describing business processes [11]. Figure 1 overviews the BPMN syntax used in this work, which consists of the following constructs:

- *Start event* indicates where the process starts and *End event* indicates where the process ends.
- A *task* contains a description, a duration and a set of resources necessary to execute this task.
- There are three main types of *gateways*: inclusive, exclusive, and parallel. A split inclusive gateway indicates that at least one branch is executed, and a split exclusive gateway indicates that only one branch is executed. A split parallel gateway indicates that several branches can be executed simultaneously. A merge gateway indicates that the executed branches need to converge at this gateway.
- *Flows* are used to connect nodes (start/end events, tasks, gateways).

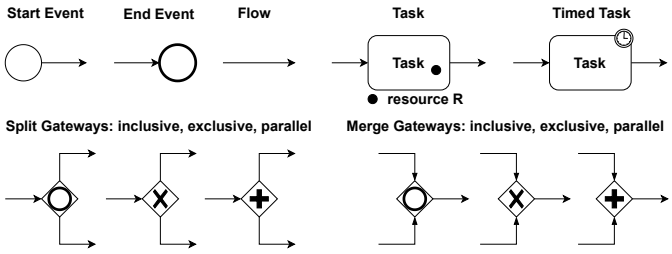


Figure 1: Overview of the BPMN syntax

2.2 Resources

The execution of some tasks in a BPMN process requires the availability of some specific resources to be executed. Therefore, these resources associated with tasks need to be made explicit when defining the BPMN process. Several tasks can use the same resource, and a task can use several resources. In the rest of this paper, we will present how a global pool of resources is adjusted depending on a workload (that is, the number of process instances executed simultaneously). Let us now describe how these global resources are defined.

Definition 1 (Resource): A resource is defined as a tuple $(ID, name, number, cost, BU, TC)$, where ID is the resource identifier, $name$ is the resource name, and $number$ is the number of replicas of that resource. $cost$ is a pair $(active, inactive)$, where $active$ is the cost per unit of time when the resource is active, and $inactive$ is the cost per unit of time when the resource is inactive. BU is an interval $[min, max]$ specifying the expected resource usage ratio, where min denotes the minimum resource usage and max denotes the maximum resource usage, with $min \leq max \leq 1$. TC is the time for resource adjustment. The number of replicas is adjustable once per TC. This is to avoid the oscillation of resource usage due to frequent resource adjustment.

For a resource $R = (ID, name, number, cost, BU, TC)$, we refer to its constituents as R_{ID}, R_{name}, R_{number}, R_{cost}, R_{BU} and R_{TC}, respectively.

2.3 Execution Traces

A process can be executed multiple times, resulting in multiple instances. Each process instance being executed can be in one of the following three states: waiting state, running/ongoing state, and completed state. In a waiting state, the process token is in the start event. In a running/ongoing state, the token has not yet reached the end event. In a completed state, the token has reached the end event.

a) Task

We use the following definition to describe the task of a BPMN process.

Definition 2 (Task): A task T is an atomic activity part of a process. It contains an instance ID, a task ID, an execution time, a set of resource parameters, a start time ($T.\mathrm{startTime}$), and an end time ($T.\mathrm{endTime}$).

In Definition 2, a set of resource parameters is optional and contains information about the required resources (such as resource type and quantity).

b) Instance

We take into account the multiple executions of a process.

Definition 3 (Instance): Each execution of a process is referred to as an *instance*, which contains a process ID, an identifier (instance ID), the sequence of tasks carried out by the process, a start time ($I.\mathrm{startTime}$), and an end time ($I.\mathrm{endTime}$).

c) Execution Traces of Tasks

The computation of resource properties requires the use of execution traces of tasks, where a task may require the use of different resources.

Definition 4 (Execution Trace): An execution trace refers to a sequence of tasks that are part of a specific instance of the process.

The execution traces of tasks, denoted as $\mathcal{T}_{tasks}$, consist of sequences of ongoing or completed tasks generated by the process during the execution of each instance. These traces provide a detailed account of task execution, including their order, duration, and resource utilisation.

d) Resource Trace

We need a resource trace to describe the number of resources R at a given time, which is useful to define the dynamic resource allocation algorithm.

Definition 5 (Resource Trace): A resource trace, denoted as $\mathcal{T}_{resource}$, refers to a sequence of data points that captures resource information. A data point is defined as a triple (ts, R, $replica$), where ts represents a timestamp, R represents a resource, and $replica$ represents the number of resource R at the given timestamp ts.

The purpose of a resource trace is to monitor and track the changes in the availability or quantity of a resource throughout the execution of a process or system. By analysing the resource trace, insights can be gained into resource utilisation patterns, fluctuations in resource availability, and resource allocation strategies.

Definition 6 (Resource Usage Trace): A resource usage trace, denoted as $\mathcal{T}_{U(R)}$, refers to a sequence of data points that capture the utilisation of resources. A point is defined as a triple (ts, R, $U(R)$), where ts represents a timestamp, R represents a resource, and $U(R)$ represents the usage of resource R at the given timestamp ts.

The purpose of a resource usage trace is to monitor and analyse the patterns of resource utilisation throughout the execution of a process. It provides insights into the intensity or frequency of resource usage, peak or idle periods of resource utilisation, and overall resource consumption trends.

2.4 Properties

We consider time- and resource-based properties in this work.

a) Time-based properties

Time-based properties generally refer to execution time. Some examples of such properties are the total execution time of a process or the minimum, maximum, and average execution time of an instance. In this paper, we focus on computing the average execution time for each instance.

b) Resource-based properties

Resource-based properties generally refer to the usage of the resources required to execute tasks involved in a process. Some examples of such properties are the global time usage of all instances of each resource or the average usage percentage for a specific resource over the global execution time. In this paper, we focus on the average usage of each replica of each resource.

By relying on time- and resource-based properties as well as costs associated to each resource, we can also compute the total cost of the multiple executions of a process.

3. Methodology

This section details the approach. After giving an overview (Section 3.1), Section 3.2 introduces the monitoring techniques. Section 3.3 presents how to perform the computation of the properties. Section 3.4 provides a description of the algorithm for dynamic resource allocation.

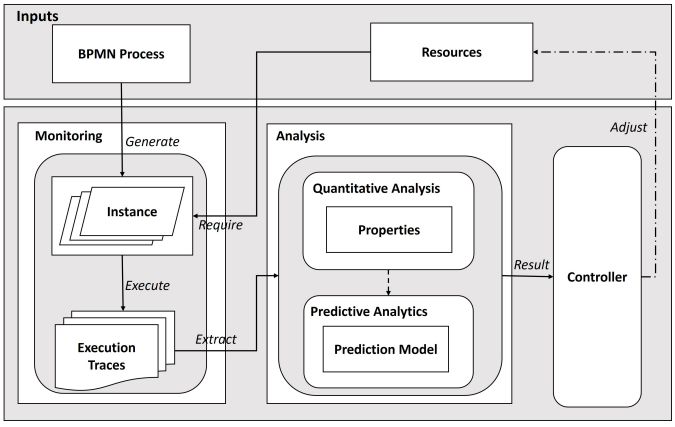


Figure 2: Approach overview

3.1 Overview (Figure 2)

The inputs are an executable BPMN process and the global resources available initially for executing it. The approach is realized by three main components, which are a monitoring component, an analysis component and a controller component. In the end, the output is used to adjust the number of resources required to execute multiple process instances.

The main idea is to monitor the execution of the BPMN process and to analyse the execution traces it generates. Based on these traces, the properties of the process at runtime are evaluated. Finally, a dynamic resource allocation algorithm is used so that the number of replicas of relevant resources are continuously adjusted based on the evaluation of the aforementioned properties.

3.2 Monitoring

Figure 3 overviews the monitoring of BPMN processes. Any new execution of a BPMN process results in a process instance. When the tasks of this process instance execute, they need first to acquire the required resources. All this execution-related information is stored in a database. For example, the stored information contains ongoing tasks for each process instance, completed tasks for each process instance, etc. We can then extract execution traces for each process instance by extracting relevant information from the database. These execution traces allow the computation of several properties, such as the resource usage (percentage), the average execution time and the total execution cost.

3.3 Computation of Properties

In this subsection, we present how to carry out the computation of three properties: resource usage, average execution time, and total cost.

There are two ways to compute these properties at runtime: cumulative and non-cumulative. Regarding the cumulative computation of properties, we consider all relevant execution traces. This type of computation considers all past completed and ongoing tasks executed for this BPMN process. Conversely, regarding the non-cumulative property computation

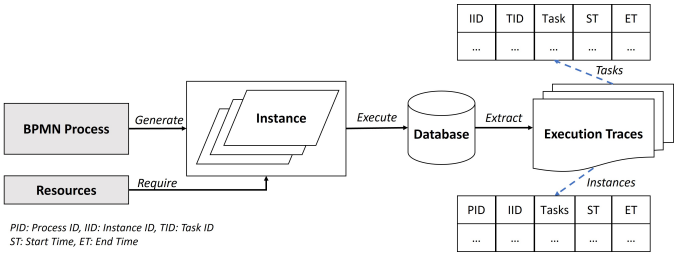


Figure 3: Overview of the process monitoring

mode, we only consider the execution traces involved in the recent past, that is, for a given period. In this paper, we mainly consider the latter case.

a) Sliding Windows

To allow for efficient and scalable online monitoring, we use sliding windows. To do this, in a running BPMN process, we give a *checkpoint* parameter, which refers to the point in time when we perform a new computation of the properties. Whenever we encounter a *checkpoint*, we retrieve all the execution traces for the last period from this point in time. Another parameter is known as *look-ahead time*, which is the width of the window used to indicate for how long the data has been observed in the past.

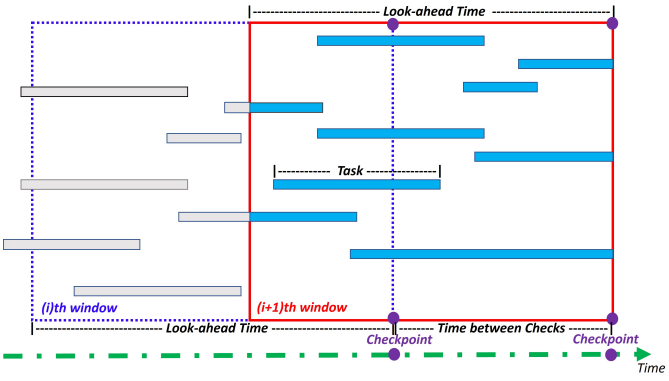


Figure 4: Overview of sliding windows

Figure 4 shows the computation process from the $(i)th$ window shift to the $(i+1)th$ window.

b) Resource Usage

Resource usage is the average usage of a single replica of a resource R used by a process during its execution for a given period.

Equation 1 defines the resource usage of a resource R. It is the total execution time of all the tasks $\sum_{i=0}^{n} Dtask_i$ that use the resource R within a given period td divided by the number of replicas R_{number} of the resource and the given time td (*look-ahead time*). To compute this, we first filter the tasks using the resource during a given period. Then we compute the result by using Equation 1.

$$U_R := \frac{\sum_{i=0}^{n} Dtask_i}{R_{\text{number}} \times td} \tag{1}$$

Algorithm 1 Obtaining execution traces of tasks

Inputs: A process ID (pid), a timestamp (ts), and a time duration (td)

Output: A set of execution traces of tasks ($\mathcal{T}_{tasks}$)

1: $\mathcal{I} := \emptyset, \mathcal{T} := \emptyset, \mathcal{T}_{tasks} := \emptyset, \mathcal{T}_{temp} := \emptyset$
2: $\mathcal{I} := getInstances(pid)$
3: **for all** $I \in \mathcal{I}$ **do**
4: $\quad \mathcal{T} := I.computeSortedTasks()$
5: $\quad$ **if** $ts - td \leq I.\text{startTime}$ and $I.\text{endTime} \leq ts$ **then**
6: $\quad\quad \mathcal{T}_{tasks} := \mathcal{T}_{tasks} \cup \mathcal{T}$
7: $\quad$ **else if** $I.\text{startTime} < ts - td$ and $I.\text{endTime} \leq ts$ **then**
8: $\quad\quad$ **for all** task $T \in \mathcal{T}$ **do**
9: $\quad\quad\quad$ **if** $T.\text{startTime} < ts - td$ and $ts - td < T.\text{endTime}$ **then**
10: $\quad\quad\quad\quad T.\text{startTime} := ts - td$
$\quad\quad\quad\quad \mathcal{T}_{temp} := \mathcal{T}_{temp} \cup \{T\}$
11: $\quad\quad\quad$ **end if**
12: $\quad\quad$ **end for**
13: $\quad\quad \mathcal{T}_{tasks} := \mathcal{T}_{tasks} \cup \mathcal{T}_{temp}, \mathcal{T}_{temp} := \emptyset$
14: $\quad$ **else if** $ts - td < I.\text{startTime}$ and ($ts \leq I.\text{endTime}$ or $I.\text{endTime}$ is NaN) **then**
15: $\quad\quad$ **for all** task $T \in \mathcal{T}$ **do** ▷ NaN means unfinished.
16: $\quad\quad\quad$ **if** $T.\text{endTime}$ is NaN or $ts \leq T.\text{endTime}$ **then**
17: $\quad\quad\quad\quad T.\text{endTime} := ts$
18: $\quad\quad\quad$ **end if**
19: $\quad\quad$ **end for**
20: $\quad\quad \mathcal{T}_{tasks} := \mathcal{T}_{tasks} \cup \mathcal{T}$
21: $\quad$ **end if**
22: **end for**
return $\mathcal{T}_{tasks}$

Algorithm 1 describes the execution process for obtaining the execution traces of tasks. Its inputs are a process ID, a current timestamp (*checkpoint*), and a period (*look-ahead time*). It returns a set of execution traces of the task. This is a filtering algorithm that iterates over different instances of a given process and then determines in turn whether the start time and/or the end time of all the tasks in these instances fall within a given time duration. In the end, it stores the tasks that satisfy the condition into the set of returned values.

Let us describe Algorithm 1 in more details. First, all the instances corresponding to a given process are retrieved using the function $getInstances()$, and stored in a set of instances $\mathcal{I}$ (line 2). Next, the algorithm traverses each instance of this set, retrieves the tasks in that instance according to the function $computeSortedTasks()$, and stores them in a set of tasks $\mathcal{T}$. Afterwards, the algorithm determines the status of the current instance by its start and end time. When its start and end times are in the given time range $(ts - td)$, all its tasks are added to the returned set of tasks (lines 5 to 6). When its start time is earlier than $(ts - td)$, but its end time is earlier than the end timestamp ts, the algorithm iterates through all its tasks, and when the end time of the task is greater than $(ts - td)$, the start

time of this task is changed to the time point $(ts - td)$ and this task is added to the returned set (lines 7 to 13). In the last case, when the start time of the current instance is later than $(ts - td)$ and the instance has not ended before ts, the algorithm iterates through its tasks and changes the end time of the related task to the end timestamp. Similarly, the task is added to the set of execution traces of tasks (lines 14 to 21). When all instances have been traversed, the algorithm terminates and returns the execution traces of the associated tasks.

The time complexity of this algorithm is $\mathcal{O}(n \times m)$, where n is the number of instances of the given process and m is the number of tasks in each instance.

c) Average Execution Time (AET)

In this work, the average execution time (*AET*) refers to the average execution time per instance of the completed instances $\mathcal{T}_{insts}$ within a given time duration td.

The computation of the AET property is divided into two main parts. First, we obtain the execution traces of the completed instances for the given period. Then, we compute the average execution time for each of these instances.

Algorithm 2 describes the process of obtaining completed instances within a specified sliding window of time for a particular process. The algorithm's inputs are the process's ID pid, a timestamp ts, and a time duration td (look-ahead time). This timestamp is the moment when the execution of the algorithm starts. First, an empty set is initialised to store the returned results. Then, function $getInstances(pid)$ fetches all the instances of the process (line 2). The algorithm iterates over these instances and determines if the completion time of the instance is within the given duration. If the condition is satisfied, we add this instance to the resulting set (lines 4 to 6). When the traversal is completed, the algorithm returns a set of completed instances $\mathcal{T}_{insts}$. The time complexity of this algorithm is $\mathcal{O}(n)$, where n is the number of instances.

Algorithm 2 Obtaining completed instances

Inputs: A process ID (pid), a timestamp (ts), and a time duration (td)
Output: A set of completed instances ($\mathcal{T}_{insts}$)

1: $\mathcal{I} := \emptyset, \mathcal{T}_{insts} := \emptyset$
2: $\mathcal{I} := getInstances(pid)$
3: **for** each $I \in \mathcal{I}$ **do**
4: **if** $ts - td \leq I.\text{endTime} \leq ts$ **then**
5: $\mathcal{T}_{insts} := \mathcal{T}_{insts} \cup \{I\}$
6: **end if**
7: **end for**
return $\mathcal{T}_{insts}$

Given a set of completed instances returned by the previous algorithm, Equation 2 shows how to compute their *AET*, which can be expressed based on this set of instances $\mathcal{T}_{insts} = \{I_1, I_2, ..., I_n\}$, where I_i represents the i^{th} instance, as follows:

$$AET := \frac{1}{n} \sum_{i=1}^{n} (I_i.\text{endTime} - I_i.\text{startTime}) \quad (2)$$

Here, $\sum_{i=1}^{n} (I_i.\text{endTime} - I_i.\text{startTime})$ denotes the sum of execution times for all instances, and $\frac{1}{n}$ represents the average of the total execution time, resulting in the *AET* of a single instance.

d) Execution Cost

Regarding the total execution cost, we only need to count the cumulative sum of the total costs per resource used during the execution of the process.

Equation 3 describes how to calculate the execution cost, where n is the number of different resources used in the process, t_{active} indicates the execution time of the active state of the resource R, and $t_{inactive}$ refers to the execution time of the inactive state of the resource.

$$T_{cost} := \sum_{i=1}^{n} (R(i)_{cost}.active \times t_{active} + R(i)_{cost}.inactive \times t_{inactive}) \quad (3)$$

3.4 Dynamic Resource Allocation

This section describes the algorithm for dynamic resource allocation (Algorithm 3). The algorithm is a core component belonging to the controller part. The main idea of the algorithm is that the number of each resource is adjusted by computing its usage at runtime.

The inputs of this algorithm are a resource usage of resource R, a duration t, a number n that indicates the number of replicas of the resource that can be adjusted each time, and a resource trace $\mathcal{T}_{resource}$, which is used to record the change in the number of resource replicas. Algorithm 3 runs periodically or when significant resource usage changes occur. It adjusts resources for all running BPMN instances. U_R represents the overall resource usage across instances, and n is the number of replicas added/removed per resource.

Algorithm 3 Dynamic resource allocation

Inputs: U_R Resource usage of resource R, a duration t, a number n, a resource trace $\mathcal{T}_{resource}$
Output: A resource trace $\mathcal{T}'_{resource}$

1: $currentTime$:= $getCurrentTime()$
2: $\mathcal{T}'_{resource} := \mathcal{T}_{resource}$
3: **if** $currentTime$ - $R_{\text{TC}} \geq t$ **then**
4: **if** $R_{\text{BU}}.max \leq U_R$ **then**
5: R_{number} := R_{number} + n
6: $\mathcal{T}'_{resource}$:= $\mathcal{T}'_{resource} \cup \{(currentTime, R_{\text{number}})\}$
7: **else if** $U_R \leq R_{\text{BU}}.min$ **then**
8: **if** $R_{\text{number}} - n > 0$ **then**
9: R_{number} := $R_{\text{number}} - n$
10: $\mathcal{T}'_{resource}$:= $\mathcal{T}'_{resource} \cup \{(currentTime, R_{\text{number}})\}$
11: **end if**
12: **end if**
13: **end if**
return $\mathcal{T}'_{resource}$

The details of Algorithm 3 are as follows. The algorithm first obtains the current time through function $getCurrentTime()$ and then determines whether the resource is in the period (R_{TC}) based on the time of the last quantity adjustment of the resource (line 3). The algorithm is interrupted if the resource is still in the R_{TC} period. Otherwise, we determine whether the current resource utilisation exceeds the resource's minimum or maximum utilisation constraint. If the resource's usage exceeds its maximum set usage ($R_{\mathrm{BU}}.max$), the algorithm adds n replicas of it (lines 4 to 6). Similarly, if the resource's usage goes lower than its minimum set usage ($R_{\mathrm{BU}}.min$), the number of replicas is reduced by n (lines 7 to 11). The algorithm does not perform any other operation if the current resource utilisation is within the given minimum and maximum interval. The time complexity of Algorithm 3 is $\mathcal{O}(1)$.

4. Prediction

In this section, we describe why and how we can incorporate predictive analytics to further improve the dynamic resource allocation strategy. Section 4.1 introduces the main idea. Section 4.2 describes the predictive monitoring component. The dynamic resource allocation algorithm integrating predictive monitoring is presented in Section 4.3.

4.1 Overview

The dynamic resource allocation strategy introduced in the former section works as follows. When the resource usage of any resource R is outside its usage interval (R_{BU}), the controller adjusts the number of replicas of that resource at runtime. However, when the controller changes the number of resources, the resource usage is already too low or too high with respect to the given interval. One could say that the decision is thus taken somehow too late. The idea of relying on predictive analytics is to reduce and even avoid this late decision. By adding a predictive analytics component to the approach, the idea is to compute in advance the properties, in particular those related to resource usage, and then detect also in advance that a resource usage will move outside of the given usage interval R_{BU}. Consequently, the controller can rely on these predicted values to adjust with anticipation the number of replicas of some specific resources.

4.2 Predictive Analytics

Predictive analytics consists of the following elements: input data, data processing, building the model, evaluating it, and using the trained predictive model. The predictive model uses both historical data and real-time observations. It is initially built with historical data and continuously updated with new observations to stay accurate and responsive to changes. Figure 5 gives an overview of the prediction analytics steps that we will now detail.

a) Input Data

The primary input data corresponds to the execution traces generated by the multiple executions of the process, which are obtained by the monitoring component. In this paper, the input

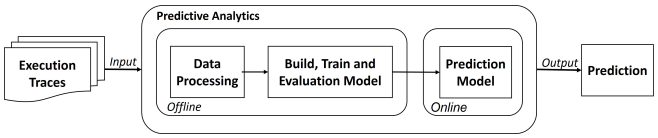


Figure 5: Overview of predictive analytics

data corresponds to the execution traces of resource utilisation generated after quantitative analysis of the data obtained by monitoring. This data type is a tuple (timestamp, resource R, resource usage U_R).

b) Data Processing

This step consists of two parts. The first part deals with data extraction, where we need to determine the input data size for executing historical resource usage traces. This requires to specify whether only the latest data or data from a previous period is needed. The second part involves processing the execution traces for each resource, which allows us to obtain the execution traces specific to each resource. For instance, data regarding resource R is represented as a tuple (timestamp, resource usage U_R).

c) Building Model

Depending on the item to be predicted, various predictive models can be considered. The choice of the model needs to be defined according to the input data type. For example, real-time numerical data generated at runtime can be seen as time series data. In the data processing phase, time series data can be transformed into supervised problems using specific algorithms. In this case, a deep learning model could be chosen to solve this regression problem. It is noteworthy that each type of resource is associated with a different model. That is, if there are n different resources in the process, n different forecasting models are used to predict their usage accurately. This is because each resource may behave differently, and these differences need to be considered to make accurate predictions.

Algorithm 4 Lag observation algorithm - transform time series data into supervised learning problem

Inputs: Time series data $T(x_1, x_2, ..., x_T)$, lag time steps n for each observation x_t, and future time steps N for prediction.

Output: Transformed supervised learning problem dataset (X, y).

1: $X \leftarrow [], y \leftarrow []$ ▷ Initialize input data X and label data y.
2: **for** $i = n$ to $T - N$ **do**
3: Use $x_{i-n}, \ldots, x_{i-1}$ as input feature vector x_i.
4: Use x_{i+N} as label y_i.
5: X.append(x_i) ▷ Append x_i to input data X.
6: y.append(y_i) ▷ Append y_i to label data y.
7: **end for**
return dataset (X, y).

The lag observation algorithm (Algorithm 4) is a technique for transforming time series data into a supervised learning problem. The algorithm takes three inputs: the time series data

$T(x_1, x_2, ..., x_T)$, the lag time steps n for each observation x_t, and the future time steps N for prediction. The algorithm initializes two empty arrays X and y to store the input features and corresponding labels, respectively (line 1). The loop starts at the nth observation and goes up to the $(T-N)$th observation. For each observation x_i, the algorithm selects the n lagged observations before it as the input feature vector x_i (line 3) and the Nth observation after it as the corresponding label y_i (line 4). It then appends the input feature vector x_i to the input data array X and the corresponding label y_i to the label data array y (lines 5 to 6). Finally, the algorithm returns a transformed dataset (X, y) of supervised learning problem. The time complexity of this algorithm is $\mathcal{O}(T \times n)$, where T is the length of the time series data, n is the number of lag time steps.

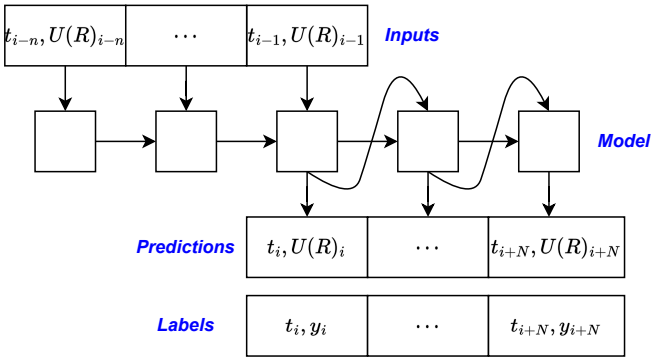


Figure 6: Overview of model training

Figure 6 depicts the process of model training based on the data generated by Algorithm 4.

d) Evaluation

Given that resource utilisation forecasts represent continuous data over a relatively short period of time, we use the Root-Mean Squared Error (RMSE) equation, which is a measure to perform quantitative deviations. This equation evaluates the difference between the predicted and actual values when we predict numerical values. The formulas of $RMSE$ is $\sqrt{\frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{n}}$, where y_i indicates the real value, $\hat{y}_i$ indicates the predicted value, and n represents the number of sample data.

4.3 Dynamic Resource Allocation Using Predictive Analytics

Figure 7 shows an overview of our dynamic resource allocation strategy combined with predictive analytics.

Predictive analytics is introduced as follows. After we have gone through the quantitative analysis of the process, the resulting values for the different properties (in particular resource usage) are used as input data. We then perform data processing by, for example, cleaning the data or changing the format of the data to satisfy the input part of the prediction model. With this offline data, the predictive model is trained and, based on the evaluation function, a converged model is finally generated. When some new real-time resource

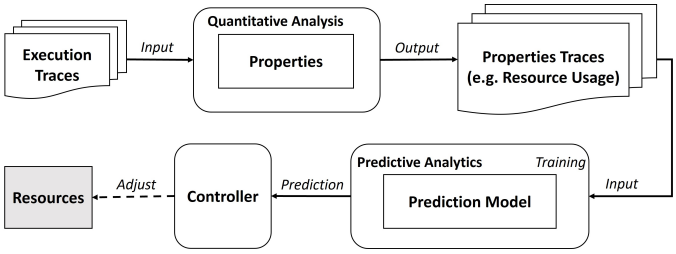


Figure 7: Resource allocation strategy with predictive analytics

utilisation data is fed into the model, the model returns a prediction of resource usage at some point in the future (e.g., the value of the next checkpoint), and the controller adjusts the number of replicas of the relevant resource based on the values returned by the prediction, thus making resource adjustments in advance. To prevent overfitting and ensure robustness in volatile environments, we use techniques like regularisation, cross-validation, and continuous model updates, ensuring that our approach remains effective in dynamic and large-scale business process environments.

5. Tool Support

The tool we implemented automates all the steps of our approach. It consists of the following components: BPMN process and resource modelling, monitoring, quantitative analysis, predictive analytics, and controller.

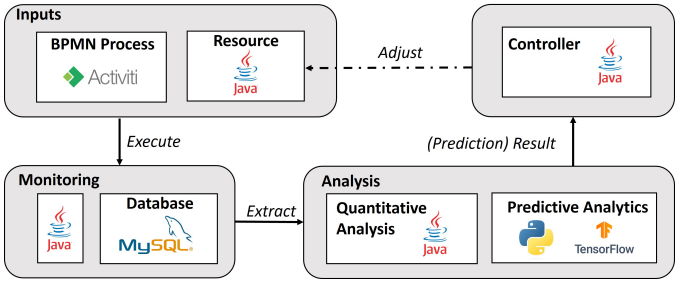


Figure 8: Overview of tool support

Figure 8 shows an overview of tool support. The BPMN process is designed using the open-source software *Activiti*[1], which is a BPMN engine supporting process automation. We extended it to describe tasks, including additional parameters such as execution time and resources. As far as monitoring is concerned, data is stored using a *Mysql* database, and extraction of the execution traces is implemented in Java. Quantitative analysis, the controller component, and the real-time visualisation of data were implemented in Java. Finally, regarding predictive analytics, the predictive model is implemented using the Python deep learning framework *Tensorflow*[2], where the Long Short-Term Memory (LSTM) [12] model is chosen as deep learning model.

[1]https://www.activiti.org/
[2]https://www.tensorflow.org/

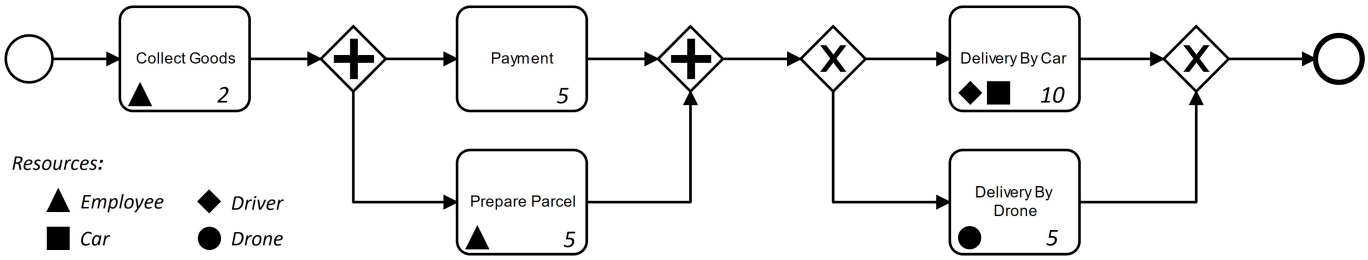


Figure 9: BPMN process of goods delivery

6. Case Study

This section focuses on a BPMN process for illustrating our approach. More specifically, we present experimental results on this process for the two dynamic resource allocation approaches presented in this paper (without and with the predictive analytics component).

6.1 Process Example

For illustration purposes, we present a process describing how clients can deliver goods via an external service (a mail office, for instance). This process is described in Figure 9. First of all, an employee collects the goods brought by a client. Then, in parallel, the client pays for the delivery service, and an employee prepares a parcel. The company can deliver the parcel using a car or drone (depending on the distance, for example). Beyond the required resources appearing in the figure, we can also see times (expressed as durations) associated with tasks. For example, the average time for preparing a parcel is five units (e.g., 5 minutes). We also assume that the probability of delivering by car or drone is the same (0.5).

6.2 Experimental Evaluation

This experiment aims at comparing the two different approaches, one of which does not use the predictive analytics component, and the other which does. It uses the same workload where the same seed is used to control the values of the random numbers generated. The simulation executes 1000 instances of the process. We consider that the utilisation interval of each resource should range in $[70\%, 90\%]$, and for each resource, its initial number is one replica. The cost of using the resource is 40 (euros or any other currency) per time unit when active and 10 per time unit when inactive. When the resource's utilisation percentage goes outside of its given interval, the resource is increased or decreased by one replica, where the number of replicas cannot be less than one. The size of the sliding window is 60 time units, and the utilisation of each resource is recalculated every 10 time units.

a) Dynamic Resource Allocation

Figure 10 describes the dynamic resource allocation approach without predictive analytics. It contains four sub-figures: 1) Figure 10 (a) gives the average utilisation rate of each resource replica. 2) Figure 10 (b) shows the average execution time of each completed instance within the given window. 3) Figure 10 (c) describes the change in the number of resource replicas. 4) Figure 10 (d) presents the cost of the whole process execution. Figure 10 (a) and Figure 10 (c) show that the goal of this work is to adjust the number of replicas of different resources according to whether its usage is within a given interval or not. For example, for the *Drone* resource, a new replica is added when its current utilisation exceeds 90%. One can also see the peak points of the average execution time shown in Figure 10 (b). One can thus deduce the correspondence between these peaks and the number of resource replicas shown in Figure 10 (c). For example, the number of *Employee* resources is adjusted to 4, and just before this, the average execution time reaches a peak.

b) Approach with Predictive Component

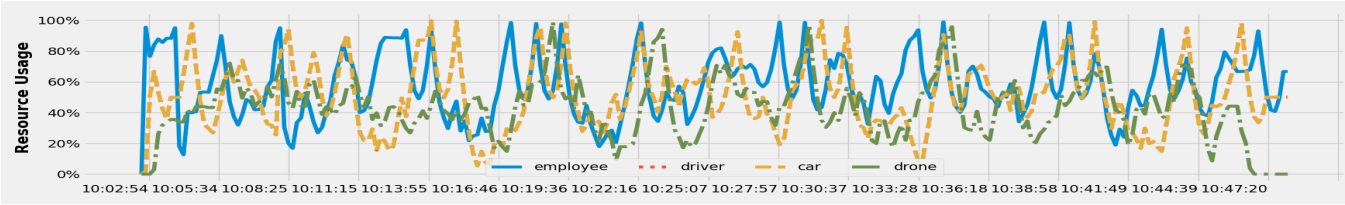
Regarding the prediction model structure of this experiment, we use a three-layer *LSTM* nested model, followed by two *Dense* layers. This model is trained based on offline resource usage data, after which it is trained continuously (e.g., every 100 new resource usage data). In the current experiment, the model predicts usage at the next checkpoint based on usage data for the last five checkpoints. In the example, there are four different resources, and we create separate prediction models for each resource, but for convenience, they have the same model structure.

Figure 11 shows the experimental results of the dynamic resource allocation approach using the predictive analytics component. The allocation works as described in the previous section: by using the predictive model, the values of resource usage are computed in advance, thus allowing the controller to adjust the number of resources with anticipation. It is worth mentioning that since different resources use different prediction models, the *Car* and *Driver* resources, which were initially associated together no longer have the same values for their usage all the time as they had before.

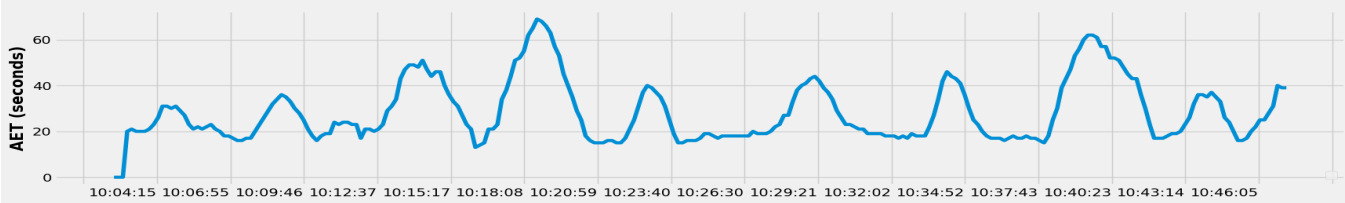
c) Discussion

Figure 12 and Table I show the results of the comparison between these two approaches, where (1) represents the initial dynamic resource allocation strategy and (2) corresponds to the strategy with the predictive analytics component.

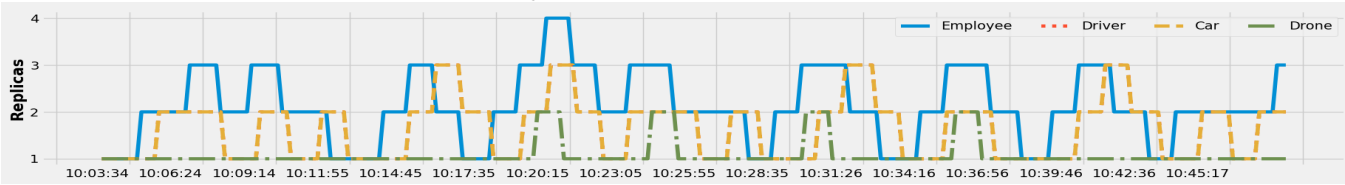
Figure 12 represents the density distribution of resource usage for both approaches. It uses the kernel density estimation (KDE) method, which applies kernel smoothing for probability density estimation. The main idea is to compare the values of the density distribution of the resource usage for the same



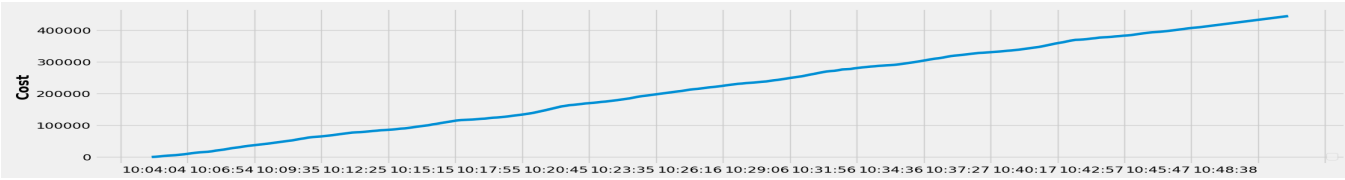
(a) Resource Usage



(b) Average Execution Time (AET) of instances



(c) Number of replicas of resources



(d) Cost of process execution

Figure 10: Results without predictive analytics

resource under both approaches. In particular, we evaluate the values of their density distribution within a given interval $\mathcal{V}$ of desired usage (i.e., $[70\%, 90\%]$). This figure allows us first to compare the density results for the approach (1) (without) and approach (2) (with predictive analytics). One can see that approach (2) has a higher density of resource usage over the interval $\mathcal{V}$. More precisely, resource *(d) Drone* has a more significant enhancement, followed by resources *(b) Driver* and *(c) Car*, and finally resource *(a) Employee*. The descending order of their enhancement rates is: *(d)* $>$ *(b)* $\simeq$ *(c)* $>$ *(a)*. Second, the KDE results also show that the variation in approach (2) is relatively smooth and uniform, which can be interpreted as a more balanced distribution of resource usage.

Property \ Approach	(1)	(2)	Results
AET	28.67	24.44	14.75% ↓
Total Cost	420798	409936	2.58% ↓

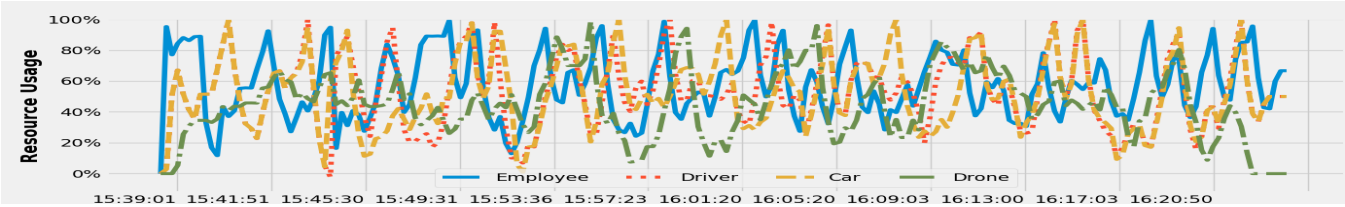
TABLE I: Experimental results: properties

Table I shows the results of these two approaches on some properties (the AET of a single instance and the total cost). By using the approach that includes a predictive component, the average execution time is significantly reduced by about 15% on the current process. As for the total cost of the process execution, it is reduced by about 3% thanks to the predictive analytics approach. The reduction in costs is explained as follows: as the average execution time is reduced, the global execution time of all the instances is reduced. Therefore, the inactive time of the resources is reduced, thus reducing the total cost associated with the process.

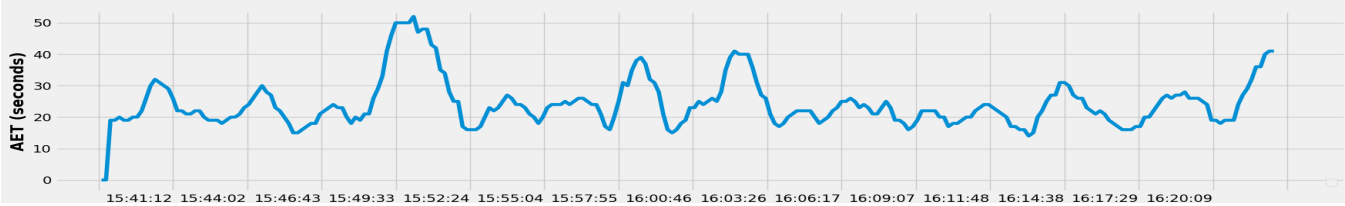
d) Additional Remarks

In this work, the predictive model was trained by using real-time data on resource utilisation. It is worth noting that, although we use the LSTM model in our experiments, our approach is generic and any kind of predictive model can be used instead. Our goal was not to choose the best predictive model but to show how machine learning and predictive analytics could be used to improve resource allocation in the context of executable business processes.

Dynamic allocation resource is not a new problem and many solutions have been proposed for it. In this paper, we have a specific focus on this problem for business processes. There are many specificities (workflow-based models, execution

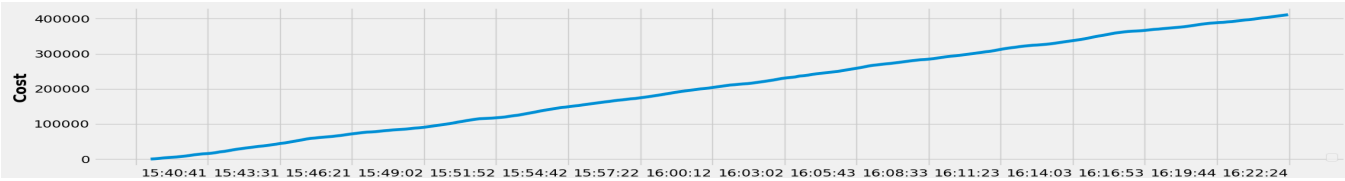


(a) Resource Usage



(b) Average Execution Time (AET) of instances

(c) Number of replicas of resources



(d) Cost of process execution

Figure 11: Results with predictive analytics

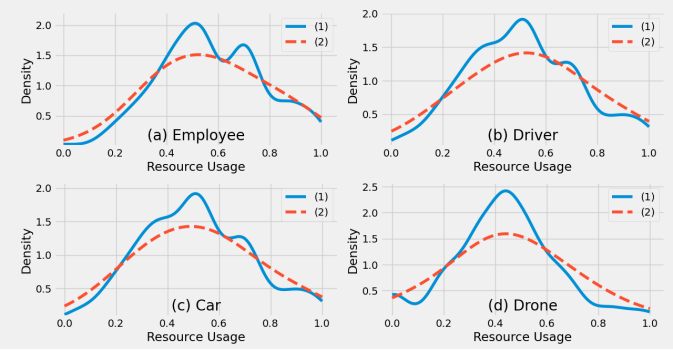


Figure 12: Experimental results: resources

probabilities, workloads, sliding windows, etc.) which make very difficult to carry out experiments for comparing our solution with other existing approaches (out of the business process domain). In contrast, these results are useful for other classic problems existing for business processes such as process optimisation, process refactoring or process performance.

7. Related Work

This section focuses successively on three main topics in business processes: monitoring process, allocation of resources, and predictive analytics.

a) Monitoring

The purpose of process monitoring is to track the execution of running processes. These techniques are able to automatically monitor the execution of processes, thus facilitating resource analysis. The authors in [13] propose to extend the BPMN process notation so it can be monitored at runtime. Similarly, in [14], [15], the authors describe how BPMN can be monitored by incorporating runtime techniques. This approach, which involves cumulative calculations, may experience reduced efficiency in the context of big data environments. In contrast, our approach, which relies on sliding windows for non-cumulative computing, can address this issue effectively. Additionally, there exists predictive business process monitoring, which focuses on a particular problem of prediction (e.g. [16]–[19], etc.). However, these studies do not focus on the monitoring itself, such as how to extract information or how to quantitatively compute properties in real time for ongoing instances or tasks.

b) Resource Allocation

The authors in [20] propose an automated analysis approach for evaluating and measuring business process execution time and resource occupancy given a workload and a provisioning

strategy. [21] focuses on human resource allocation and the authors survey research in the domain of human resource allocation in Business Process Management and process mining. Some studies claim that resource allocation can be considered as an optimisation problem. The authors in [22] propose to assign the same resources to specific tasks and to prioritise specific tasks to optimise their order of execution [23]. [24] provides a solution to the resource allocation problem based on multi-objective optimisation algorithms. In contrast, our dynamic resource allocation approach applies on running processes without constraints on resource types. Compared to [25], we enhance BPMN process optimisation by integrating predictive analytics with dynamic, real-time resource allocation to improve scalability, resource utilisation, and adaptability to fluctuating workloads.

c) Predictive Analytics

Prior to the emergence of deep learning techniques, traditional machine learning methods were predominantly utilised in predictive analytics research as observed in studies such as [7] or [8], which rely on Hidden Markov Models. However, the development of deep learning techniques has resulted in an increasing number of diverse neural network models. Furthermore, there is a growing trend to incorporate predictive analytics into business process development.

In the field of business process runtime prediction, [26] provides an overview of the prevalent methods. [27] introduces the utilisation of deep learning techniques for content prediction related to business processes. Various prediction goals have been explored in these studies, including the next task [9], [10], [16], [17], [28], the execution order of tasks within a running instance [17], the start timestamp of the next task [10], [16], or the remaining execution time of an instance [29]. Moreover, a wide range of prediction models have been used. For instance, [28] utilised Convolutional Neural Networks (CNNs) as a predictive model, DFNN (Deep Feedforward Network) was used in [30], or GRU (Gated Recurrent Unit) was utilised in [31]. However, the LSTM network model is the one receiving most attention in the context of business processes, see, e.g., [9], [10], [16]. In [9], the focus primarily lies in predicting the next activity, while [10], [16] additionally predict the process execution time and the time of the next event. Additionally, several novel neural network models, such as the attention mechanism, have been applied in the domain of business processes. As an example, [18], [19], [32], [33] have specifically focused on predicting the next event or task, whereas [19] extended their work to include the prediction of the remaining execution time of an instance.

All the previous works aim at predicting the next task, the sequence of tasks for a running instance, or the time of the next task. They do not address the prediction of resource usage in business processes, whereas our primary focus lies in predicting the resource usage, enabling us to allocate resources intelligently between running processes. By predicting resource usage, we can proactively make well-informed decisions regarding resource allocation adjustments, resulting in substantial gains when it comes to optimising business processes.

8. Conclusion

In this paper, we have presented an innovative approach to optimise the resource allocation within BPMN processes at runtime. To do so, the BPMN process is first monitored during its execution and then the produced execution traces are analysed. These traces allow us to compute several critical quantitative properties of the process, including resource usage and average execution time. Finally, based on the results of these properties, the number of replicas of different resources is dynamically adjusted. A preliminary strategy modifies resource numbers based on current usage percentages falling outside predefined utilisation intervals, often resulting in delayed adjustments. To avoid this delayed decision issue, we have incorporated a predictive analytics component, which can predict resource usage in advance by training a predictive model from historical execution traces. This component improves our approach by dynamically adjusting the number of resources with anticipation. These ideas were implemented into a toolchain that we applied to several realistic process scenarios.

The main perspective of this work aims at refining the predictive analytics component by exploring alternative predictive models, and thus providing more accurate predictions. These enhanced prediction values would allow us to obtain better gains in terms of average execution time and total costs associated with the multiple executions of the process. By refining the predictive analytics component, the goal is to better optimise resource allocation and improve the overall performance of the process.

Acknowledgements

This work was supported by (i) the Région Auvergne-Rhône-Alpes within the "*Pack Ambition Recherche*" programme, and by (ii) the French National Research Agency in the framework of the "*France 2030*" program (ANR-15-IDEX-0002), and the LabEx PERSYVAL-Lab (ANR-11-LABX-0025-01).

References

[1] C. Cabanillas, "Process- and Resource-Aware Information Systems," in *Proc. of EDOC'16*. IEEE Computer Society, 2016, pp. 1–10.

[2] J. Xu, C. Liu, X. Zhao, and Z. Ding, "Incorporating Structural Improvement into Resource Allocation for Business Process Execution Planning," *Concurr. Comput. Pract. Exp.*, vol. 25, no. 3, pp. 427–442, 2013.

[3] R. M. Dijkman, J. Hofstetter, and J. Koehler, *Business Process Model and Notation - BPMN 2011*, ser. LNBIP. Springer, 2011, vol. 95.

[4] W. Zhao, H. Liu, W. Dai, and J. Ma, "An Entropy-based Clustering Ensemble Method to Support Resource Allocation in Business Process Management," *Knowl. Inf. Syst.*, vol. 48, no. 2, pp. 305–330, 2016.

[5] W. Zhao, Q. Zeng, G. Zheng, and L. Yang, "The Resource Allocation Model for Multi-Process Instances Based on Particle Swarm Optimization," *Information Systems Frontiers*, vol. 19, no. 5, pp. 1057–1066, 2017.

[6] A. Kumar, W. Van der Aalst, and H. Verbeek, "Dynamic Work Distribution in Workflow Management Systems: How to Balance Quality and Performance," *J. Manag. Inf. Syst.*, vol. 18, no. 3, pp. 157–194, 01 2002.

[7] G. T. Lakshmanan, D. Shamsi, Y. N. Doganata, M. Unuvar, and R. Khalaf, "A Markov prediction model for data-driven semi-structured business processes," *Knowl. Inf. Syst.*, vol. 42, no. 1, pp. 97–126, 2015.

[8] B. Letham, C. Rudin, T. H. McCormick, and D. Madigan, "Interpretable classifiers using rules and Bayesian analysis: Building a better stroke prediction model," *The Annals of Applied Statistics*, vol. 9, no. 3, 2015.

[9] J. Evermann, J.-R. Rehse, and P. Fettke, "Predicting process behaviour using deep learning," *Decis. Support Syst.*, vol. 100, pp. 129–140, 2016.

[10] N. Tax, I. Verenich, M. L. Rosa, and M. Dumas, "Predictive Business Process Monitoring with LSTM Neural Networks," in *Proc. of CAiSE'17*, ser. LNCS, vol. 10253. Springer, 2017, pp. 477–492.

[11] Y. Falcone, G. Salaün, and A. Zuo, "Semi-automated modelling of optimized BPMN processes," in *Proc. of IEEE SCC'21*. IEEE, 2021, pp. 425–430.

[12] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural computation*, vol. 9, no. 8, 1997.

[13] J.-P. Friedenstab, C. Janiesch, M. Matzner, and O. Muller, "Extending BPMN for Business Activity Monitoring," in *Proc. of HICSS'12*. IEEE Computer Society, 2012, pp. 4158–4167.

[14] Y. Falcone, G. Salaün, and A. Zuo, "Probabilistic model checking of BPMN processes at runtime," in *proc. of IFM'22*, ser. LNCS, vol. 13274. Springer, 2022, pp. 191–208.

[15] Y. Falcone, G. Salaün, and A. Zuo, "Probabilistic Runtime Enforcement of Executable BPMN Processes," in *Proc. of FASE@ETAPS'24*, ser. LNCS, vol. 14573. Springer, 2024, pp. 56–76.

[16] M. Camargo, M. Dumas, and O. González-Rojas, "Learning Accurate LSTM Models of Business Processes," in *Proc. of BPM'19*, ser. LNCS, vol. 11675. Springer, 2019, pp. 286–302.

[17] L. Lin, L. Wen, and J. Wang, "MM-Pred: A Deep Predictive Model for Multi-attribute Event Sequence," in *Proc. of SDM'19*. SIAM, 2019, pp. 118–126.

[18] A. Jalayer, M. Kahani, A. Pourmasoumi, and A. Beheshti, "HAM-Net: Predictive Business Process Monitoring with a hierarchical attention mechanism," *Knowl. Based Syst.*, vol. 236, p. 107722, 2022.

[19] Z. A. Bukhsh, A. Saeed, and R. M. Dijkman, "Processtransformer: Predictive Business Process Monitoring with Transformer Network," *arXiv preprint arXiv:2104.00721*, 2021.

[20] F. Durán, C. Rocha, and G. Salaün, "Resource Provisioning Strategies for BPMN Processes: Specification and Analysis using Maude," *J. Log. Algebraic Methods Program.*, vol. 123, p. 100711, 2021.

[21] M. Arias, R. Saavedra, M. R. Marques, J. Munoz-Gama, and M. Sepúlveda, "Human resource allocation in business process management and process mining: A systematic mapping study," *Management Decision*, vol. 56, no. 2, pp. 376–405, 2018.

[22] I. Barba, B. Weber, C. Del Valle, and A. Jiménez-Ramírez, "User recommendations for the optimized execution of business processes," *Data Knowl. Eng.*, vol. 86, pp. 61–84, 2013.

[23] M. Yaghoubi and M. Zahedi, "Resource allocation using task similarity distance in business process management systems," in *Proc. of ICSPIS'16*. IEEE, 2016, pp. 1–5.

[24] F. Durán, C. Rocha, and G. Salaün, "A rewriting logic approach to resource allocation analysis in business process models," *Sci. Comput. Program.*, vol. 183, 2019.

[25] F. Durán, Y. Falcone, C. Rocha, G. Salaün, and A. Zuo, "From Static to Dynamic Analysis and Allocation of Resources for BPMN Processes," in *Proc. of WRLA@ETAPES'22*, ser. LNCS, vol. 13252. Springer, 2022, pp. 3–21.

[26] A. Marquez-Chamorro, M. Resinas, and A. Ruiz-Cortés, "Predictive Monitoring of Business Processes: A Survey," *IEEE Trans. Serv. Comput.*, vol. 11, no. 6, pp. 962–977, 2018.

[27] E. Rama-Maneiro, J. C. Vidal, and M. Lama, "Deep Learning for Predictive Business Process Monitoring: Review and Benchmark," *IEEE Trans. Serv. Comput.*, vol. 16, no. 1, pp. 739–756, 2023.

[28] N. D. Mauro, A. Appice, and T. M. A. Basile, "Activity Prediction of Business Process Instances with Inception CNN Models," in *Proc. of AI*AI'19*, ser. LNCS, vol. 11946. Springer, 2019, pp. 348–361.

[29] N. A. Wahid, T. N. Adi, H. Bae, and Y. Choi, "Predictive Business Process Monitoring-Remaining Time Prediction using Deep Neural Network with Entity Embedding," *Procedia Computer Science*, vol. 161, pp. 1080–1088, 2019.

[30] J. Theis and H. Darabi, "Decay Replay Mining to Predict Next Process Events," *IEEE Access*, vol. 7, pp. 119 787–119 803, 2019.

[31] M. Hinkka, T. Lehto, and K. Heljanko, "Exploiting Event Log Event Attributes in RNN Based Prediction," in *Proc. of ADBIS'19*, ser. CCIS, vol. 1064. Springer, 2019, pp. 405–416.

[32] P. Philipp, R. Jacob, S. Robert, and J. Beyerer, "Predictive Analysis of Business Processes Using Neural Networks with Attention Mechanism," in *Proc. of ICAIIC'20*. IEEE, 2020.

[33] W. Ni, G. Zhao, T. Liu, Q. Zeng, and X. Xu, "Predictive Business Process Monitoring Approach Based on Hierarchical Transformer," *Electronics*, vol. 12, no. 6, p. 1273, 2023.