# Scalability Benchmarking of Stream Conformance Checking Algorithms

Implementation and Comparison

Benedikt Marc Masuhr

Bachelor thesis
September 2024

Software Engineering Group
Department of Computer Science
Kiel University

Advised by
Prof. Dr. Wilhelm Hasselbring
M. Sc. Hendrik Reiter

**Selbstständigkeitserklärung**

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Weiterhin erkläre ich, dass die digitale Fassung dieser Arbeit, die dem Prüfungsamt per E-Mail zugegangen ist, der vorliegenden schriftlichen Fassung entspricht.

Kiel,

# Abstract

Businesses often specify the orderly execution of their processes through process models. Conformance checking, a branch of Process Mining, compares the actual observed behaviour with the prescribed behaviour in the process model. Many conformance checking techniques aim to accomplish this. In modern times businesses are interested in the detection of deviations between execution of a process and the prescribed model in real time. Consequently, conformance checking techniques are required that work in an online, or in-vivo, setting. Many such online conformance checking algorithm exists with different results. In this paper, we aim to establish a fundamental basis for benchmarking online conformance checking algorithms. We implemented and tested three of the most cited online techniques and tested them based on our benchmarks. The benchmarks show that the horizontal scalability of the tested algorithms varies, while the results indicate one specific algorithm.

# Contents

# List of Figures

# List of Tables

Chapter 1

# Introduction

Process mining is the domain of monitoring, validating and improving business processes [Aal11]. Process mining has three main disciplines: Process discovery, process enhancement and conformance checking. Process discovery is concerned with discovering business process models extracted from information available in today's information system, e.g. event logs. There exists a multitude of algorithms that extract information from a process log and formulate a fitting process model [Aal16].

Process enhancement focuses on optimising business process models and their execution in real life. Enhancement achieves this goal through two different types of process enhancement: process extension and process improvement. Both techniques ensure that the process model is as precise as possible, clearly defining the behaviour that is allowed and not allowing for any illegal behaviour. The model is also being modified to fit with what is realistically possible in real execution regarding time constraints and resource consumption, e.g. workforce. Lastly, it improves current process models through replacing or forbidding execution that leads to inefficient performance, while trying to build efficient process models [Leo22].

Conformance checking takes the event log and the process model as input. Where the event log represents the is-state of the business execution, the business process model represents the target state of the process execution. Conformance checking compares them with each other to assess how much of the event log adheres to the prescribed process model [Aal12].

Nevertheless, event logs need to collect all the information about the execution of the cases, before the cases can be verified. This consequently results in the identification of discrepancies at a later stage than would be optimal, i.e. when they occur. The ability to identify deviations from a process in real-time is becoming increasingly relevant, as regulations from agencies become more demanding, and businesses become more focused on optimising their processes. Deviations from the process model must be identified in real time in highly critical environments, e.g. healthcare institutions or airports. It is therefore necessary to develop conformance checking algorithms that do not rely on the completed event log before the recorded behaviour is compared to the prescribed one. Such algorithms are referred to as online conformance checking algorithms. The distinction between online and non-online algorithms lies in their capacity to process each execution of a case in real time as it is recorded. Since the source of events, in an online setting, is an active event stream, instead of an event log that is completed in advance, the algorithm must monitor all activities recorded in a case to ascertain the sequence in which the process was executed for each case. The difficulty lies in the fact that the completion of a case from the most recent recorded activity is unknown, preventing a comprehensive examination of the case.

# 1. Introduction

This paper aims to test different popular online conformance checking algorithms. These algorithms were chosen based on the frequency of citations they have garnered. Our goal is to create a foundation to benchmark different online conformance checking algorithms on parameters that are relevant to their performance regarding an event stream and their scalability. We aim to create tests that favour only the algorithm that scales best on the given task. For that, we create several benchmark tests with increasing intensity and evaluate which algorithm performs best in which task.

The remainder of this paper is structured as follows. In Chapter 2 the fundamental principles underlying our online conformance checking algorithms are introduced and explained. Chapter 3 presents a review of the literature and methods that were not incorporated into this work but were deemed relevant nevertheless. In Chapter 4 we present the implementation of our online conformance checking algorithm and provide an explanation of its functionality. Subsequently, a series of tests is conducted in Chapter 5. Furthermore, the results of the aforementioned tests are subjected to analysis and visualisation. The challenges that occurred during implementation as well as a set of assumptions, made prior to running the algorithms, are presented in Chapter 6.1. The findings are summarised in Chapter 7 and a series of potential avenues for future research are proposed.

Chapter 2

# Background

For this paper, it is assumed that the technical process description is a Petri net. While alternative approaches to technical process description are frequently employed, the graphical representation method is particularly prevalent [DSM+19]. A variety of graphical representations are available for consideration including, but not limited to, UML activity diagrams, BPMN, Heuristics nets, Directly-Follows Graphs and EPC. However, Petri nets are among the oldest and most thoroughly investigated process model languages [Aal16]. Moreover, numerous models can be translated from one representation to another without the loss of information. Concerning the implemented algorithms we only need information extracted from the model, except for the Token-based replay algorithm, which needs the Petri net itself. The Token-based replay necessitates a sound workflow Petri net. The information derived from Petri nets and the Petri nets themselves are discussed in this chapter.

## 2.1 Petri net

The definition of a Petri net is described in [Mur89] and is regularly repeated throughout the literature. A full Petri net with markings is defined by [Mur89] as:

**2.1.1 Definition** (Petri net)**.** Petri net is a 5-tuple $P_N = (P, T, F, W, M_0)$ where:
$P = \{p_1, p_2, p_3, \cdots, p_n\}$ as all places within the Petri net
$T = \{t_1, t_2, t_3, \cdots, t_m\}$ as all transitions included in the Petri net
$F \subseteq (P \times T) \cup (T \times P)$ as the arcs between places and transitions
$W : F \to \mathbb{N}_0$ is an optional weight function on places and transitions,
$M_i : P \to \mathbb{N}_0$ is the initial marking positioned in the start place of the Net,
$M_f : P \to \mathbb{N}_0$ is the final marking positioned in the end place of the Net,
$P \cap T = \varnothing$ and $P \cup T \neq \varnothing$

A Petri net structure $N = (P, T, F, W)$ without any specific initial marking is denoted as $N$. A Petri net with the given initial marking is denoted as $P_N = (N, M)$.

**2.1.2 Definition** (Workflow Petri net)**.** Introduced in [VAN98], a Petri net $P_N$ is a Workflow Petri net $P_W$ if the following is true:

▷ It possesses a special place $p_o$ that acts as the only source of tokens for the Petri net $P_N$.

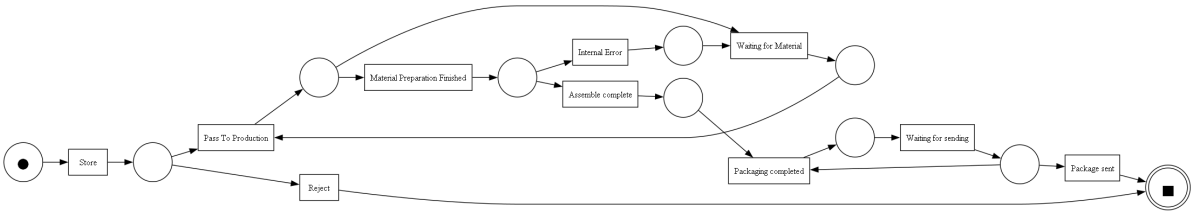



**Figure 2.1.** An example Petri net created with pm4py library [BZS23]

▷ There is another special place $p_s$ which is the sole sink of the Petri net $P_N$ where all tokens end.

▷ Any and all transitions $t^*$ can be reached from place $p_o$ as well as lead to place $p_s$. The resulting Petri net $P_N$ is considered a strongly connected Petri net.

A transition $t_n$ is active or able to "fire" if each input place $p$ leading to the transition $t_n$ contains the number of markings described by $w(p, t_n)$ where $w(p, t_n)$ is the weight of the arc from the previous place $p$ to transition t.
Activated or "fired" transition $t_n$ first consumes all tokens in previous places p. The transition $t_n$ then produces markings for all the following places p. The markings in each following place $p$ depend on $w(t_n, p)$ where $w(t_n, p)$ is the weight of the arc from the transition $t_n$ to the place $p$.

An exemplary Petri net is illustrated in Figure 2.1. The Petri net illustrates the concept of places and transitions. Transitions are represented by squares which are labelled with a designated name and an internal ID. The sole attribute of a place is an internal ID. It can be observed that the transition designated as "Store" is executed initially, immediately following the start place. The initial activity designated as the "start activity" generates a token that is directed to the subsequent places associated with the transition. The named transition "Pass to Production" or the transition named "Reject" will consume the token and produce a token in its own subsequent place. It is proposed that each execution of the process should commence at the designated place "start place" with initial marking, and the marking is consumed by a following start transition, i.e. "Store", before transferring it to its successor place. This process will continue until a token has reached the final place which is the place from which no further transitions can consume a token. In the case of the Petri net shown in Figure 2.1 it is the place after the transition named "Reject" or after the transition named "Package sent". These transitions also constitute a set of final transitions as no further transitions will be performed after them.

## 2.2 Event Logs

In a business context, a process may be performed on numerous occasions, often concurrently. This is to say that while one process is still underway a new process of the same type may be initiated. In order to ascertain the precise execution of the process it is necessary to record

it in order to facilitate a comparison between the recorded behaviour and the intended process execution set out in the model. To distinguish between each individual execution a unique identifier designated as a case ID is assigned to each process execution. Upon the conclusion of each activity within the process, as outlined by the transitions within a Petri net, the completion of that activity is recorded in the event log accompanied by additional information. This information encompasses the process execution or case to which the activity belongs, a timestamp indicating the time at which the activity was successfully completed and any supplementary data or information deemed pertinent.

**2.2.1 Definition** (Events)**.** Let $\mathcal{C}$ denote the universe of all case identifiers. Let $A$ denote the universe of all activity identifiers. Let $\mathcal{T}$ denote the universe of all time references and let $\mathcal{R}$ denote the universe of all and any additional resources. Let $E$ denote the universe of all possible events $e \in E$ with $e = (c, a, \tau, r)$.

Activity $a$ was executed in the context of case $c$ at time $\tau$ with additional resource $r$. Such additional information can be represented by any means of required resource, e.g. by whom the activity was executed or what material was used. The concatenation of many such events builds an event log.

The recorded behaviour displayed in the event log will show the is-state of the process execution which can be validated against the target-process execution represented in the Petri net.

**2.2.2 Definition** (Event Log)**.** A tuple $(c, a, \tau, r) \in C \times A \times \mathcal{T} \times \mathcal{R}$ presents a single event within the event log. An event log $L$ is a finite sequence over $C \times A \times \mathcal{T} \times \mathcal{R}$. So that $L \subseteq (C \times A \times \mathcal{T} \times \mathcal{R})$.

A model event log can be observed in Table 2.1. Each incoming event is given a uniquely identifying event ID. Further, each event belongs to one case ID.

Moreover, each event is associated with a single case ID. A multitude of events contribute to the formation of a case which represents the completion of a full business process. The activity described in the event indicates the stage of the process at which the case is currently situated. All events recorded in the event log demonstrate the extent of process completion. The timestamp provides precise information regarding the point in time at which the activity has been completed and has been transitioned to the subsequent activity which is then ready for processing. The information attribute provides details of any additional data that may be of interest such as the resources required to complete the activity or the personnel or machinery involved in the process. It is important to note that event logs are designed in such a way that events are collected and cases are fully executed before any conformance checking can be performed. This gives rise to a number of inferences. In essence, the event log encompasses all essential details pertaining to a case, including the activities undertaken within the case and any deviations from the prescribed process. Notably, the beginning as well as the end of each case are also recorded. This allows for the execution of a fully accurate compliance check. Conversely, the initial step is to collate this data which may result in a

**Table 2.1.** Example event log excerpt

| Event ID | Case ID | Activity | Timestamp | Resource |
|---|---|---|---|---|
| 1 | case 1 | Store | 2024-08-17 16:08:13 | GoodsDelivery |
| 2 | case 1 | Pass To Production | 2024-08-17 16:12:32 | GoodsDelivery |
| 3 | case 1 | Waiting for Material | 2024-08-17 16:14:32 | MaterialPreparation |
| 4 | case 1 | Pass To Production | 2024-08-17 16:15:41 | GoodsDelivery |
| 5 | case 1 | Material Prep. Finished | 2024-08-17 16:15:41 | MaterialPreparation |
| 6 | case 2 | Store | 2024-08-18 11:15:49 | GoodsDelivery |
| 7 | case 3 | Store | 2024-08-19 21:12:22 | GoodsDelivery |
| 8 | case 3 | Reject | 2024-08-20 00:55:00 | GoodsDelivery |
| 9 | case 1 | Assembly complete | 2024-08-20 03:56:00 | AssemblyProcess |
| 10 | case 2 | Material Prep. Finished | 2024-08-23 07:58:37 | MaterialPreparation |
| 11 | case 2 | Waiting for sending | 2024-08-23 16:51:05 | Shipping |
| 12 | case 1 | Package sent | 2024-08-23 17:11:27 | Shipping |

lengthy delay between process execution and conformance check. However, as previously discussed in chapter 1 there are instances where it is crucial to identify deviations from the anticipated process in a prompt and real-time manner. This highlights the necessity for event streams which are analogous to event logs but, most notably, distinguish themselves from event logs by the fact that they are potentially infinite.

**2.2.3 Definition** (Event Streams)**.** Given the universe of observable events $e = (c, a, \tau, r) \subseteq C \times A \times \mathcal{T} \times \mathcal{R}$, an event stream $S$ of events $e$ is defined as an infinite sequence of observable events.

As infinite memory does not exist, it is only possible to memorise parts of the event stream at a time. Therefore, the beginning of some cases might not be accessible any more. Furthermore, as the stream continuously grows and adds new events for each point in time $\tau$ there is no further information about each case at time $\tau$ considering the uncertainty on how the case continues or whether the last recorded activity is the last activity in the case. The problem arises during conformance checking because offline techniques do not take these challenges into account. Therefore, conformance checks can only be performed based on the activities that have been observed for each case. We will present several techniques of online conformance checking in Chapter 4. These approaches require a deeper understanding which will be demonstrated in the following.

## 2.3 Behavioural Patterns

Behavioural patterns can be derived from the target business process. While behavioural patterns do not dictate a Petri net as they are independent of the exact process model, Petri

nets are one good example of displaying behavioural patterns.

[BZA+18] defined behavioural patterns formerly as:

**2.3.1 Definition** (Behavioural Pattern)**.** A set of the universe of all activity identifiers $A$ and a possible set of the universe of all control-flow relations $R$, where a behavioural pattern is defined as $r(x, y)$ with $x, y \in A$ being activities and $r \in R$, represents a control-flow relation. It is also common to write $r(x, y)$ as $xry$. A business process $B$ is a set of all possible behavioural patterns such that $B \subseteq R \times A \times A$ with $A$ denoting the universe of all activity identifiers and $R$ denoting the universe of all control-flow relations.

To visualise behavioural patterns the activity "Store" ($a$), seen in Figure 2.1, must be executed before the activity "Pass to Production" ($b$) and both must be executed before "Material Preparation Finished" ($c$). This gives us the behavioural patterns $a > b$ and $b > c$. Furthermore, observe in Figure 2.1 the existence of pattern $b > f$ by proceeding from "Pass to Production" ($b$) to the activity "Waiting for Material" ($f$). There are many such molecular behavioural patterns in each business process.

A business execution, as seen in real life, performs several different behavioural patterns in a row. The concatenation of such behavioural patterns can be seen as something similar to the alignments of a process model.

## 2.4 Alignments

Alignments can be visualised as many behavioural patterns concatenated together. These alignments are derived from a Petri net or similar representation of a business process, similar to behavioural patterns. Consider Figure 2.2 where a model process can be observed. After the first activity "a" in transition $t_1$ a decision in the process is made whether it proceeds to activity "b" in transition $t_2$ or to activity "c" in transition $t_3$. If the process followed activity "b" it will once again decide if activity "d" in transition $t_4$ or activity "e" in transition $t_5$ is being executed next. Following that, a decision on activity "h" in transition $t_8$ will be processed. If the process decides to go to activity "c" it will continue to perform both; activity "f" in transition $t_6$ as well as activity "g" in transition $t_7$. After this concurrent move of activity "f" and "g", activity "i" in transition $t_9$ will be executed. Transition $t_{10}$ will perform activity "j" if either of the previously explained paths has been followed. After processing activity "j" the business process will terminate. Exemplary alignments that fit the described Petri net can be observed in Table 2.2. The first alignment, alignment $\gamma_1$, explores an execution path of the business process. However, at activity two and activity five the alignment records deviations from the process dictated in the process model. These two deviations visualise the possible cost a trace alignment can record. The first deviation records a skipped activity. From activity "a" the model dictates that activity "b" or activity "c" must follow: $a > b$ or $a > c$. However, activity "d" is the next recorded activity thus it needs to skip activity "b" in this case and denotes a cost increase of one. The second deviation is an added activity, i.e. an activity not prescribed by the process model and added by the event stream through noise. For the second deviation, a cost increase of one is needed as well.

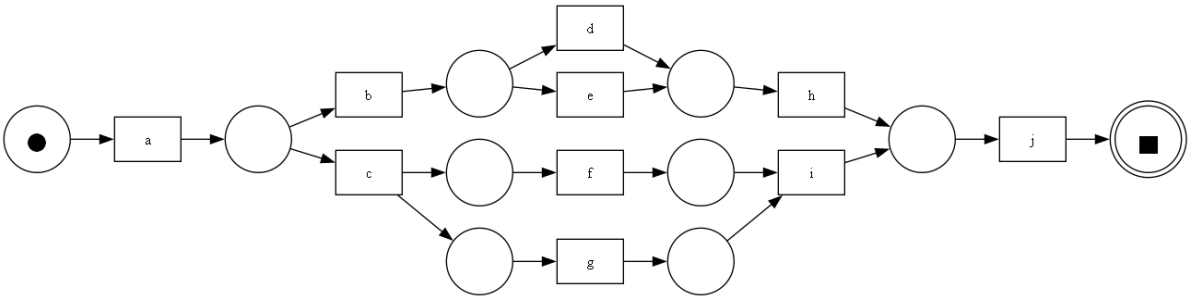



**Figure 2.2.** An example Petri net created with pm4py library [BZS23]

As there is no skip in prescribed activities, an activity that is not part of the business process is recorded, meaning activity $x \notin A$. Therefore the position of the trace can not move forward from transition $t_8$.

**2.4.1 Definition** (Alignments)**.** Let $A$ denote a set of all activities in the business process. Let $P_W$ be the Petri net over the business process. Let $\sigma \in A$ be a sequence of activities $a$ such that $\sigma = (a_1, \ldots, a_n)$. With $|\sigma| = n$ representing the number of activities in the sequence. If $n = 0$, then $\sigma$ is an empty sequence.
An alignment $\gamma \in (\sigma \times P_W)$ is:

- ▻ A legal combination of behavioural patterns $a_1 r a_2$ such that every activity in $\sigma$ can be executed properly in accordance with prior activities in $\sigma$ and to the referenced Petri net $P_W$, and
- ▻ The first activity in $\gamma$ describes an activity directly reachable from the initial marking without passing over any other named transitions. Furthermore, the last recorded activity in $\gamma$ depicts a named transition which is immediately followed by the place that holds the final marking of the Petri net $P_W$, i.e. an end activity.

$\Gamma$ is defined as the set of all allowed alignments between traces $\sigma$ and the Petri net $P_W$.

**Table 2.2.** Examples for trace alignments

$$\gamma_1 = \begin{array}{|c|c|c|c|c|c|} \hline \text{a} & \gg & \text{d} & \text{h} & \text{x} & \text{i} \\ \hline t_1 & t_2 & t_4 & t_8 & \gg & t_9 \\ \hline \end{array} \qquad \gamma_2 = \begin{array}{|c|c|c|c|c|c|} \hline \text{a} & \text{c} & \text{f} & \text{g} & \text{i} & \text{j} \\ \hline t_1 & t_3 & t_6 & t_7 & t_9 & t_{10} \\ \hline \end{array}$$

Alignments were first presented as a means for more robust methods against peculiarities of process models such as duplicate and invisible tasks [Adr14]. While these high-level deviation detections have improved low-level deviations, i.e. observed activities that are not allowed, according to the business process and the other way around, are also identified. As such they were rather effective as offline conformance checking metrics. However, they are not fit for online conformance checking as the alignment is required to be completed, i.e. it starts from the very beginning of the Petri net $P_W$ and ends at the Petri net's last named

transition. Conformance checking in online settings poses the challenge that a trace is not yet complete and more activities will be added to the trace and subsequently to the alignment in the future. As such a shorter alignment is required. One that enables growth or allows the trace only to show a sequence of allowed behaviour in the Petri net up until the last recorded activity. In short, the second part of definition 2.4.1 needs to be relaxed. However, there are many possible ways of reaching any given named transition in a Petri net. This results in the need to rank all prefix-alignments that reach the same transition between each other which is true for all transitions in the Petri net. The length is an obvious choice as a prefix-alignment is, much like a conventional alignment, just a sequence of moves or patterns. As such, the minimal cost for such a prefix-alignment $\overline{\gamma}$ in all traces $\overline{\sigma}_t$ that end at each transition $t$ is the prefix-alignment $\overline{\gamma}$ with the fewest patterns, i.e. $|\overline{\gamma}|$ is the minimum of all $\overline{\gamma} \in \overline{\sigma}_t$.

**2.4.2 Definition** (Prefix-Alignments)**.** Let $\sigma \in A$ be a sequence of activities. Furthermore, let $P_W$ be a Petri net with initial and final marking $M_i$ and $M_F$. Let $\gamma \in \Gamma(N, \sigma, M_i, M_f)$ be optimal. If $\overline{\sigma}$ is a prefix of $\sigma$ and $\overline{\gamma} \in \overline{\Gamma}(N, \overline{\sigma}, M_i, M_f)$ is an optimal prefix-alignment, then the cost $\kappa(\overline{\gamma}) \leqslant \kappa(\gamma)$.

Consequently, the cost of the completed alignment $\gamma$ is always going to be underestimated with the cost of an optimal prefix-alignment. This is advantageous because once a prefix-alignment is recorded that has a nonzero cost, i.e. a non-optimal alignment, we can guarantee that a deviation from the business process execution has occurred.

For the performance of our online conformance algorithm test a different software, previously mentioned in chapter 1, is required. The underlying system that manages the deployment and ensures our test run is the software Kubernetes.

## 2.5 Kubernetes

Our algorithm will be deployed using Kubernetes and Theodolite, explained in section 2.7. It is an open-source platform designed to automate the deployment, scaling, and operation of containerised applications[1]. Our virtual online setting will be deployed in Pods provided and managed by Kubernetes.

A docker container runs one single application and communicates information to other containers if applicable. A multitude of containers build one Pod. A Pod represents one complete set of containers working together to perform one test.

Each Kubernetes Pod will host a replication of our experiment setting in multiple containers. In the context of this paper the applications constitute our online conformance checking algorithms, a load generator to simulate the event stream and Theodolite to monitor the performance of the conformance checking algorithms. Communication between the load generator and the conformance checking algorithm is handled by Kafka.

[1] `https://kubernetes.io/`, Accessed 19.09.2024

As Kubernetes provides a robust framework for orchestrating containerised applications to combat service discovery, load balancing and scaling, it is an optimal choice for our experiment. Upon starting our experiment Kubernetes will first create a Pods with the necessary containers. Secondly, it will boot up Theodolite which will, as explained in more detail in section 2.7, handle the deployment of the Kafka communication and start our own conformance checking algorithms.

Kubernetes also enables Theodolite to deploy and integrate new containers if needed continuously. This will be especially helpful as we aim to test the horizontal scaling of our algorithms, i.e. we test how the algorithm scales with an increasing workload if multiple instances of the same algorithm are deployed.

## 2.6 Kafka

The communication between each conformance checking algorithm and the event stream will be handled by Kafka for several reasons:

- Apache Kafka is a well event-streaming platform[2] known for its performances to distribute, in scalability and in being fault-tolerant. As the load is generated and sent to the conformance checking algorithm it increases. The scalability and fault-tolerance of the algorithms are key parts of the experiments.

- Kafka is compatible with our benchmark software Theodolite and deployment software Kubernetes.

- Kafka represents a publish-subscribe model. A publisher issues messages that all subscribed consumers can read. The role of the publisher inhabits the event stream producer while the role of the consumer is taken by the algorithms.

For these reasons Kafka is a reasonable choice for handling the generation of events in a simulated event stream and the reliable delivery to the conformance checking algorithm. The Producer creates events based on a load-generating YAML file that represents the Petri net. The actual load intensity is controlled by Theodolite. The Kafka consumer takes each event and calls the conformance checking algorithm to process the information in the event in accordance with the Petri net.

## 2.7 Theodelite

As a benchmark software Theodolite will be used. The definition of a benchmark is summarised by [Has21] as a standard tool for competitive evaluation and comparison of software in regards to characteristics such as performance, dependability or scalability. A benchmark

[2]Kafka, `https://kafka.apache.org/documentation`, Accessed: 20.09.2024

must motivate the comparison of the software itself in the research area. Further, the benchmark must be relatable to the real world, i.e. the benchmark should perform tests that mimic the behaviour the software is exposed to in actual practice. The measurements from the benchmark must be quantitative or qualitative and feasible by a human or a machine. In addition to these requirements for benchmarks [Has21] and [KAH+15] define a few key characteristics. These characteristics amount to reproducibility, usability, verifiability, and fairness. Reproducibility requires the benchmark to consistently produce similar results when it is run with the same test configurations. Usability ensures that others can also reproduce the benchmark without roadblocks. Verifiability constitutes the characteristic that a benchmark provides accurate results. The fairness attribute declares that different test configurations are allowed to compete on their values without any artificial limitations.
Theodolite is a framework for performing benchmark tests regarding the horizontal and vertical scalability of cloud-native applications [HH22]. Each one of our conformance checking algorithms will undergo a Theodolite benchmark test. During the benchmark testing these algorithms will be denoted as systems under test (SUT). Theodolite introduces three metrics for their framework: load intensity, service level objectives (SLO) and provisioned resources. Load intensity specifies the amount of data a system receives, i.e. how often the processing of data by the system under test is called. The better the SUT can handle these increasing data loads, i.e. at every new message the SUT can successfully process the message before the next one requires processing, the better it scales. In the framework of this paper the messages are events where the conformance checking algorithms are being tested to see how many events they can successfully process in, e.g. a minute or a second, disregarding events that have been created but not processed thus far. SLO are quality criteria that must be fulfilled by the system under test at every load intensity for the system under test to pass the load intensity. Provisioned resources are the amount of memory and CPU power a system under test has access to, as well as the number of instances of the system under test. Provisioned resources are split into two metrics that Theodolites tracks during benchmark tests.

Horizontal scaling tests the scalability of the system under test by varying the amount of Kubernetes Pods while the memory and CPU constraints stay the same.

Theodolite can determine the resource demand metric and the load capacity metric from these benchmarks defined by [HH22] as follows.

**2.7.1 Definition.** Let $L$ be the set of possible load intensities. Let $R$ be the set of possible resource kinds. Further, assume that an ordering between both set $L$ and $R$ exists. Let $\mathcal{S}$ be the set of all possible SLO $s \in \mathcal{S}$ as a Boolean-valued function $slo_s : L \times R \to \{false, true\}$ with $slo_s(l, r) = true$ if a system deployed with $r$ resource amounts does not violate SLO $s$ when processing load intensity $l$.

The resource demand metric is denoted as $demand : L \to R, definedas :$

$\forall l \in L : demand(l) = min\{r \in R \mid \forall s \in \mathcal{S} : slos(l, r) = true\}$

Likewise the resource capacity metric is denoted as $capacity : \mathcal{R} \to L$ and defined as:
$\forall r \in \mathcal{R} : capacity(r) = max\{l \in L \mid \forall s \in \mathcal{S} : slos(l, r) = true\}$

The load generator produces messages for the Kafka producer to release to the consumer or system under test. The task of this system under test is to process the messages from the producer as they are released. The lag depicts the messages that have been published but have yet to be processed by the consumer. Further, the lag trend is the growth in messages that the producer has released but the consumer has not processed thus far. The lag trend informs about the discrepancy between the speed with which the producer generates new messages and the speed of the system under test, or the consumer, with which it processes the messages. Theodolite measures the lag trend between producer and consumer making it an adequate choice for stream benchmark testing [HH21]. Furthermore, previous research by [Rei24] proved that Theodolite is well suited for performing benchmark tests on our process mining tasks such as conformance checking.

Chapter 3

# Literature Review

A plethora of different online conformance checking techniques exist. While most papers perform tests on their individual algorithm, some papers [WSA+22; BA21] also compare their algorithm with others. However, to the best of our knowledge, there is no research comparing the scalability of several online conformance checking algorithms without bias of the algorithm introduced in their paper or with tests based on benchmark characteristics. Paper exists that compares such algorithms, however, they only compare them based on characteristics such as methods or quality metrics [DSM+19].

Our conformance checking algorithms all embody the same method for memory cleanup. However, there is ample literature providing different methods. These methods are likely faster and better suited to determine which cases to keep and which to delete. Several different algorithms for finding frequent items are introduced in [CH09]. Through extensive testing [CH09] deduces that counter-based algorithms perform the best. The counter-based "SpaceSaving" algorithm introduced by [MAE04] was a clear favourite by [CH09]. "SpaceSaving" stores a $n$ amount of pairs (item, count) initialised by the first $n$ distinct items and their exact counts. Whenever a new item appears that is tracked the counter for that item is increased. If the next item is not monitored, it replaces the currently tracked item with the fewest occurrences. The count of the item that is now monitored is increased by one. This results in a space consumption of $O(n)$. Three different methods for efficient memory consumption are presented in [ZHD21]. These methods include limiting the saved trace to the newest behaviour exclusively limiting the number of cases tracked and a combination of both. A more complex method is introduced in [CSS+09]: The method explains forward delay. At the time when a new case is recorded a forward decay function determines the point at which the case will no longer be of relevance. An algorithm needs to be able to know at all times when a case is decayed. This might add more resource consumption and there is no guarantee that the case will truly be dispensable.

A more abstract method for memory keeping is introduced in [Vit85] via reservoirs. Deducted from the paper a reservoir with size n can be interposed between accepting an event and processing the activity with its case trace. The reservoir would hold recurring cases. All cases not in the reservoir have a greater chance of being erased from memory.

Chapter 4

# Online conformance checking algorithms

This section presents the three online conformance checking algorithms utilised in this paper. The first online conformance checking algorithm that will be implemented is the behavioural pattern algorithm which is founded upon the work of [BZA+18]. The second algorithm is based on the concept of prefix-alignment as outlined in [ZBH+17]. They improve the algorithm presented in [Adr14] based on alignments and adapt it to the online setting. The final algorithm is introduced in [BA21] and implemented in the PM4Py library of the Fraunhofer Institution [BZS23]. All three algorithms have been implemented in Python version 3.10. The deployment of our algorithms will be handled by Kubernetes in a cloud-based container setting. The scalability benchmark test will be performed using Theodolite software presented in [HH21]. The focus of the testing will be on the horizontal scaling of the implemented algorithms and the load lag. Horizontal scaling tests how the performance of the algorithms improves with an increasing number of deployments. The load lag describes the difference between the speed of incoming events from the stream and the speed of the algorithm processing each event. Apache Kafka will be used to handle the communication between the source for events, i.e. our event stream and the containers hosting our conformance checking algorithms.

## 4.1 Behavioural Pattern

The initial algorithm presented in this paper as outlined by [BZA+18] is not reliant on a particular model or representation of a process execution. The algorithm operates on behavioural patterns as its sole input. The aforementioned behavioural patterns are defined in definition 2.3. In the absence of prior knowledge regarding the behavioural patterns, they can be retrieved from any representation of a process model, e.g. a Petri net or BMPN. A number of methods for the extraction of these patterns can be found in [KKV03] and [MP95]. As these patterns are assumed and indeed are provided in our test setup through a Heuristics net, the readers are directed to these sources for further research if interested. These behavioural patterns define the sequence of activities that may be initiated following the completion of a preceding activity. The sourcing of the behavioural patterns is conducted in an offline setting prior to the launch of the online conformance checking process.

As event streams operate as an infinite sequence of events with some earlier events already out of memory [BZA+18] also consider warm start scenarios, i.e. traces that do not necessarily

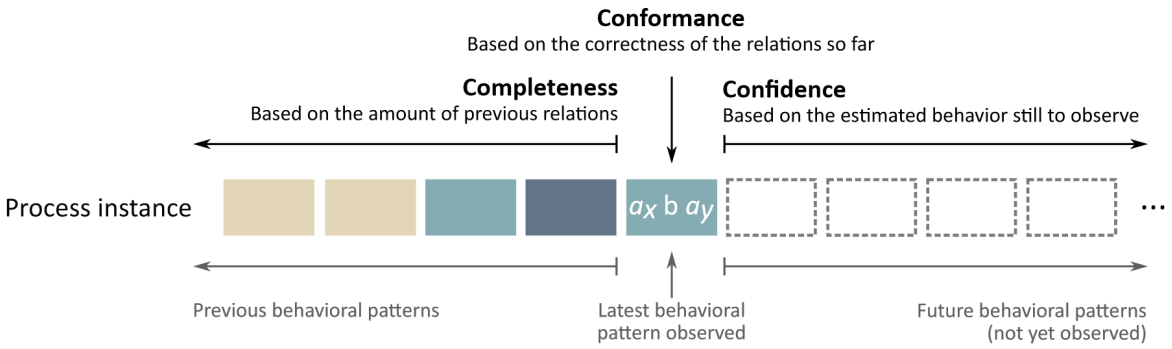



**Figure 4.1.** Three measures presented in [BZA+18]

record all previously completed activities. They decided to look at how many activities might have happened before the recorded activities in the trace. This leads [BZA+18] to consider three major parameters to assess the overall conformance of the considered trace w.r.t the model the trace must adhere to.

- Completeness: Displaying whether or not a trace starts closer to the business process's beginning. The more activity relations from the start of the business process to the recorded start in the trace are missing, the lower the completeness value is. The completeness value decreases as the margin for false activity relations increases. Respectively, if the trace starts at the very beginning of the business process, the completeness value is at its maximum value, which is one as there are no possible relations preceding the trace.

- Conformance: Demonstrating how much of the behaviour seen in the trace matches the proper business process execution according to the model. The more they match the higher the conformance score is. If the recorded process snippet in the trace perfectly matches the business process prescribed by the model, the trace conformance and its value are at their maximum, which is also one. If the behaviour observed in the trace however does not match any part of the business process prescribed by the model and is therefore not represented in the reachability graph, the conformance is at its minimum value, which is zero.

- Confidence: Predicting how likely it is for the trace to be completed in accordance with the business process foreseen by the reference model. Traces that are further from completion may trail from the supposed business process execution. This results in a higher statistical chance for deviation in the future. As such, traces which are far from business process completion are assigned a low confidence, while traces which near their potential completion are given a higher score. The value for confidence varies between zero and one. With one being the highest value meaning they finished, and zero meaning that the trace is furthest from completion and, therefore, has the highest uncertainty of completing conform.

Considering Figure 4.1 where these parameters are visualised. While completeness summarises what could have happened before all behaviour was recorded, conformance focuses on the behaviour recorded much like other existing online conformance checking techniques and, finally, confidence gauges how likely it is that the case finishes in conformity to the Petri net.

## 4.2 Behavioural Pattern Algorithm Implementation

For the accurate and fast processing of events our algorithm requires several sets of information. This information can be extracted from the set of behavioural patterns calculated beforehand. The following information are needed:

- The set $B$ of all behavioural patterns prescribed by the business process model.
- A dictionary $f_s$ that contains the fewest number of behavioural patterns for each activity in the prescribed business process to an end activity. This means that the minimum number of following behavioural patterns can be disclosed through a simple look-up of the current activity or behavioural pattern.
- For each behavioural pattern the dictionary $p_p$ contains the minimum number of distinct prescribed patterns which must be observed before any other activity is observed.

The algorithmic procedure for the online computation of the conformance checking is detailed in the following algorithm 1. In addition to the aforementioned data the algorithm requires the event stream. The algorithm initiates the construction of two dictionaries: The dictionaries $D_a$ and $D_i$. The $D_a$ dictionary is used to map incoming activities and behavioural patterns to a given case ID. The second dictionary designated as $D_i$ employs the case ID as the key and maps it to the number of incorrectly observed behavioural patterns.

In order to compute the online conformance checking metrics for each case, the following steps are calculated perpetually given that the stream of events is unbounded. Upon the registration of a new event, a check is conducted to identify whether this event constitutes the initial occurrence of the case. In the event that this is the situation, a new key-value pair is added to the $D_a$ dictionary and the minimal number of behavioural patterns prescribed for the observed activity is calculated. This computation has been performed in advance, allowing the information to be extracted from the dictionary $p_p$ in which it is stored. The use of dictionaries enables this operation to be completed in constant time.

Subsequently, the completeness measure is calculated in line 7. The completeness value quantifies in the interval [0,1] with 1 signifying the trace started at the beginning as intended by the business process model. The lower the value the greater the distance between the supposed start and the recorded start of the trace. This is the minimum between 1 and the total length of recorded behaviour, divided by 1 plus the minimum number of prescribed behavioural patterns required to reach the earliest observed behaviour within the trace. It was decided that the algorithm proposed in [BZA+18] should be modified, as the completeness

**Algorithm 1:** Adapted Behavioural Patterns Algorithm

**Input:** $B$, $S$, $p_p$, $f_s$

1 *Initialize map cases* $D_a$ ; /* Maps activities of same case together */
2 *Initialize map incorrect* $D_i$ ; /* Maps deviations per cases to integer costs */
3 **while** *True* **do**
4 $(c, a) \leftarrow S(e_i)$;
5 **if** $c \notin D_a$ **then**
6 $D_a \leftarrow (c, a)$;
7 $completenes(c) \leftarrow min\{1, \frac{|D_a|}{1+|p_p(c)|}\}$
8 **else**
9 $b \leftarrow D_a(c) \cup a$;
10 **if** $b \in B$ **then**
11 $D_a(c) \leftarrow D_a(c) \cup a$;
12 $confidence(c) \leftarrow 1 - \frac{f_s(b)}{max_{b' \in c} f_s(b)}$;
13 **else**
14 $D_i(c) \leftarrow D_i(c) + 1$;
15 $conformance(c) \leftarrow \frac{|D_a(c)|}{|D_a(c)|+D_i(c)}$;
16 **if** *size of* $D_a$ *and* $D_i$ *is close to max capacity* **then**
17 perform cleanup by removing oldest entries
18 **end**

value determines the distance from the start of the recorded behaviour. As this value remains constant when additional behavioural patterns are added to a case and is instead set at the point when the first behaviour is recorded, it was decided that this value should be calculated at this point.

In the event that a case has already been instantiated and contains recorded behaviour, the current event's activity and the most recent activity from the trace are taken and used to construct the behavioural pattern. Subsequently, the behavioural pattern is audited to establish whether it resides within the set $B$, comprising all permitted behavioural patterns as defined by the business model. If this is the case, the event activity is added to the trace corresponding to the case and the following metric calculation is performed.

In line 12 the result of the confidence is 1 subtracted by the number of minimal mandatory patterns from the behavioural patterns in our trace until completion $f_s(b)$ over the longest number of compulsory behavioural patterns in our trace. The confidence value ranges from [0,1], with 1 indicating that the trace has reached an end activity and thus is finished. A lower confidence value indicates a higher chance that the trace does not finish fit, w.r.t. the model.

In the event that our most recent behavioural pattern is not part of the set $B$ that entails all allowed behavioural patterns w.r.t the model the count of the incorrectly observed behavioural pattern for that case is instead increased.

The following step is the last check each trace undergoes before a new one is accepted. The

**Table 4.1.** Examples for trace alignments

$$\gamma_3 = \begin{array}{|c|c|c|c|c|} \hline \text{a} & \gg & \text{d} & \text{z} & \text{h} \\ \hline t_1 & t_2 & t_4 & \gg & t_8 \\ \hline \end{array} \qquad \gamma_4 = \begin{array}{|c|c|c|c|c|} \hline \text{a} & \text{c} & \text{f} & \gg & \text{i} \\ \hline t_1 & t_3 & t_6 & t_7 & t_9 \\ \hline \end{array}$$

conformance measure is calculated in line 15. This is the result of the length of the correctly observed trace being divided by the sum of all incorrectly observed behavioural patterns and the length of the correctly observed trace for the case. The conformance value is within the interval of [0,1]. A value of 1 indicates absolute conformity between the observed trace and the prescribed behaviour by the process model. Therefore, a lower conformance value indicates a greater discrepancy between observed and prescribed behaviour. It should be noted that the conformance value is only calculated based on observed behaviour while the completeness measure is based on prescribed but unobserved behaviour.

All the information saved and accessed by the algorithm is stored in dictionaries or sets. This allows access to the information stored in constant time regardless of the size of the dictionary or set. Further, our calculation is also performed in constant time making our algorithm suitable for online conformance checking.

## 4.3 Prefix Alignment

The second implemented algorithm in this paper is the prefix-alignment conformance checking algorithm. Alignment based conformance checking was first introduced in [Adr14] and has been adapted to work in an online setting in [ZBH+17]. Its principle idea is the same, i.e. calculating alignments. Two example alignments built from a trace of events are seen in Table 4.1. These alignments reference the Petri net in Figure 2.2. The alignments are built and played against the model to check whether they conform to the model or if deviation and, thus, cost arise.

Considering $\gamma_3$ we can observe that the trace has already recorded deviation. After activity "a" the next recorded activity is "d". However, as we can see in the Petri net in Figure 2.2 "d" is not an allowed activity to follow after "a". Therefore, we need to find an optimal prefix-alignment from "a" to "d". Such an optimal prefix-alignment is $\gamma_a = (b, d)$. Therefore, we can observe in our alignment $\gamma_3$ one skipped activity after activity "a". As new activities constantly come in and add to the cases the algorithm calculates the prefix-alignment based on all activities recorded from the first activity until the newest addition to the case. However, as not every activity might conform, an optimal way of determining the compliance of a new activity is needed. For this, [ZBH+17] assume a $\overline{a}$ heuristic algorithm that can compute approximate prefix-alignments from one activity to another in constant time. In this paper, however, some prefix-alignments are calculated beforehand. These pre-calculated prefix-alignments start at an activity and continue toward the end. This set ensures that at least one prefix-alignment from each activity reaches each succeeding activity. Given these conditions, a few steps can be performed to ensure that each incoming event is appropriate with reference

to the process model. Each activity within an event is either in accordance with the case whereby the subsequent activity is permitted to proceed following the established business process model or it is not. If it does, the activity forms our prefix-alignment with all our previously recorded activities for that case. In the event that the case does not conform to the established business process, for example, if the activity in question should occur at an earlier or later stage in the process but not in direct succession with the preceding activity within the case, a "skipped activity" is recorded. In other instances, the activity in question is not part of the overarching business process. This can result in a deviation from the established process whereby an activity is recorded that is not permitted under the circumstances. This additional activity may be attributed to an erroneous execution or simply the result of noise. Our implemented algorithm performs several checks to determine in which situation the incoming event lies and proceeds accordingly.

**Algorithm 2:** Adapted Incremental Prefix-Alignments

**Input:** $B$, $S$, $\overline{\gamma}(a_1, a_2) \mapsto \mathbb{N}_0$

*Initialize* $\gamma_{post}$ ; `/* Creates a map of suffix-alignments from activities */`
*Initialize map case* $D_\gamma$ ; `/* Maps activities of same case together */`
*Initialize map cost* $D_c$ ; `/* Maps cost to case as integer */`
**while** *True* **do**
  $(c, a) \leftarrow S(e_n)$;
  **if** $c \in D_\gamma$ **then**
    $\gamma \leftarrow D_\gamma(c)$;
    $D_\gamma \leftarrow \gamma$;
    **if** $\gamma + a$ *not an allowed prefix akignment* **then**
      $D_c \leftarrow D_c + \overline{\gamma}(\gamma, a)$;
  **else**
    $\gamma \leftarrow a$;
    $D_\gamma \leftarrow \gamma$;
    **if** $a \notin$ *start activities* **then**
      $D_c \leftarrow D_c + \overline{\gamma}(start\ activity, a)$;
  **if** *size of* $D_\gamma$ *and* $D_c$ *is close to max capacity* **then**
    perform cleanup by removing oldest entries
**end**

## 4.4 Prefix-Alignment Algorithm Implementation

In order for the continuous monitoring of events from the event stream and their compliance with the process model to become operational, the algorithm requires the reference process model as input. Furthermore, a function $\overline{\gamma}(a_1, a_2) \rightarrow \mathbb{N}_0$ is necessary to identify the optimal prefix-alignment between one activity within the process model and another activity. Thirdly, the event stream is necessary to provide the events that are processed. Once these conditions

are satisfied, the conformance checking can be initiated. Following the acceptance of a new event, the first step is to establish whether the event introduces a new case.

If the case is already established and contains a recorded trace, the new activity is added to the trace. The algorithm then verifies in line 9 if the new prefix-alignment imposes a new deviation from the process model, i.e. if the new activity is not fit with the previous activity in the trace. Under the assumption that a divergence has ensued, we calculate an optimal prefix-alignment between the newest activity in the trace and the activity from the event. The length of this prefix-alignment is noted down as deviation cost and saved in the corresponding dictionary.

Barring the event introduces a new case a new prefix-alignment is built based on the activity from the event. If the first activity in the case is not a start activity as prescribed by the process model we determine the optimal prefix-alignment from the start activity of the process model to the start activity of the prefix-alignment in the trace line 15. The length of the optimal prefix-alignment will be added to our dictionary which tracks the number of deviations in a prefix-alignment per case.

Identically to the algorithm 1 we perform a clean-up function when our storage memory reaches its maximum capacity.

To ensure that the algorithm is suitable for the online setting, we use dictionaries to map the incoming activities to case together as well as map the deviation to a case. In addition, we perform some preprocessing steps with respect to the function $\overline{\gamma}(a_1, a_2)$ by calculating some prefix-alignments beforehand.

## 4.5 Token Based Replay

Token-based replay is the third and last algorithm we consider in this paper. Token-based replay in the offline setting was one of the first ways to perform conformance checking between event logs and a reference process model, e.g. Petri net. The variant tested in this paper is based on [BA21]. They improved the original token-based replay introduced by [RA08]. As mentioned above, token-based replay requires a Petri net with final and initial marking as well as the event stream to perform compliance checks. Given the Petri net, token-based replay, as well as the other algorithms, also performs several steps in advance before running conformance checking on the event stream. The first step is to calculate a graph $G = (V, A)$ where the vertices $V$ are the places $P$ of the Petri net and $A \subseteq P \times P$. Following, to each arc $(p_1, p_2) \in A$ in the Petri net a transition $t(p_1, p_2)$ is associated to. With the defined graph a search algorithm travels along the graph $G$ searching for the shortest paths between nodes in that graph. The algorithm builds sequences of places $\langle p_1, \cdots, p_n \rangle$ that result in $(p_i, p_{i+1}) \in A$ for any $1 \leqslant i < n$. These sequences are then transformed into move sequences $\langle m_1, \cdots, m_{n-1} \rangle$ of transitions such that $m_i = (p_1, p_{i+1})$ for any $1 \leqslant i < n$. Given any marking $M$ such that $M(p_1) > 0$ and $M(p_2) = 0$, a marking $M^1$ where $M^1(p_2) > 0$ could be reached by the continuous activation of transitions in a sequence $\langle m_1, \cdots, m_n \rangle$ that is the shortest path on the graph $G$ between $p_1$ and $p_2$. Note that some places could still have

missing tokens during the activation of the sequence [BA21].

Acknowledging these sequences a trace can be replayed. Whenever a new activity from the event stream adds to a trace the entire trace is replayed by the algorithm. During replay, the algorithm tracks several variables. These variables include the amount of consumed token ($cd$), the number of produced token ($p$), the number of missing token ($mi$) and the number of remaining token ($r$). At the start of the replay it is assumed that the initial marking is placed by the Petri net which also increases $p$ by one. For each step in the trace the algorithm looks for a transition in the Petri net corresponding to an activity in the trace, at which time the following can happen:

- The transition corresponds to an activity in the trace and can be fired without the need to add markings. In this situation the activity sequence fits the Petri net.
- No shortest path between places $(p_1, p_2)$ could fire, consequently, a token is added to enable the firing of the transition.
- Not enough markings are produced in the replay of the trace thus far, meaning a token has to be inserted for the transition to fire.

For each activity in the trace a number of tokens is added in the output of said activity. This number is added to $p$. The amount of token consumed to fire the activity is added to $cd$. If a deviation occurred and an amount of tokens needed to be added for the activity to be able to fire, the amount added increases $m$ accordingly. Lastly, if the trace has not reached the final marking, i.e. the recorded process execution has reached the end of the Petri net and is therefore complete, the number of remaining tokens in the Petri net are increased to the value of $r$. After the replay is completed, these variables are used to calculate the fitness of the trace and the Petri net.

$f_{trace} = \frac{1}{2}(1 - \frac{mi}{cd}) + \frac{1}{2}(1 - \frac{r}{p})$

The function $f_{trace}$ is calculated after every replay, whenever a new activity is recorded and added to the trace. As incomplete traces and traces with many concurrent activities leave tokens at either the end of the trace or after activities that have not been executed, the remaining token value $r$ might record false positives and therefore lower the overall conformance score of the trace.

## 4.6 Similarities between the algorithms

All algorithms implement the same methods for accepting a new trace. A simple method is in place to ensure that the case ID value is valid, i.e. filled with an actual case ID. Furthermore, the activity is also being checked for a value. If a value is invalid the method logs an error. The reason for that is that an activity must have a corresponding case ID, else there is an activity without a corresponding case, and an event must have an activity, else the case logs nothing. If both keys in the event have values the event is passed to the individual algorithms that perform the actual online conformance checking. This greatly varies between all three algorithms as described in this chapter.

All three Online conformance checking algorithms presented in this paper implement the same method to perform cleanup if the storage memory reaches its maximum value. The token-based replay algorithm was adapted to implement the same method as the behavioural patterns and prefix-alignment algorithm. Additionally, the exact condition upon which this cleanup method is initiated is also the same. This was a deliberate design decision to prevent any unwarranted deviation from the algorithms that could affect runtime. The aim was to guarantee that the benchmark tests are not influenced by differences in storage maintenance and instead assess the actual algorithmic calculation.

Chapter 5

# Benchmarks

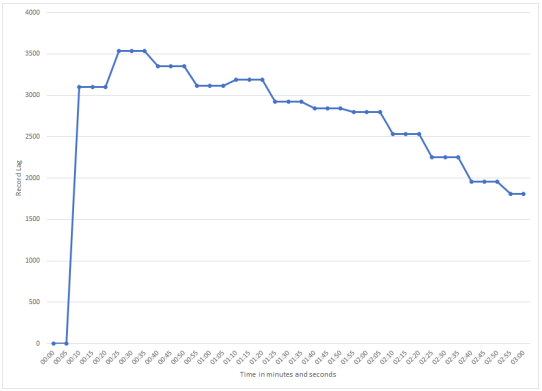
In this section, we explain the premise and setup of our benchmark tests. Following that, we present the results of the benchmark test and provide an analysis before concluding.

## 5.1 Experiment

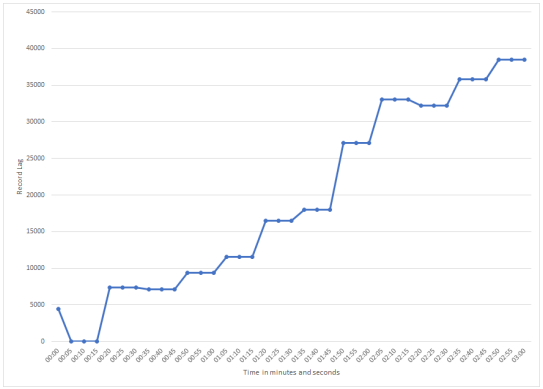
The benchmark tests were performed in a Kubernetes Pod on University servers. The Pods were given a restraint on their power access, such that the algorithms perform with a single CPU core with 2.1 GHz and 1 GB of RAM. A total of 12 benchmark tests were performed. All three algorithms were tested on both Petri nets seen in Figures A.1 and A.2. The algorithms were tested twice per Petri net, the first time with one deployment and the second time with three deployments, testing their horizontal scalability. Each test was split into two parts, with each part lasting 180 seconds, of which 45 seconds were reserved for warm-up. The warm-up period allowed the algorithm to perform all preprocessing steps required to perform their conformance checking technique on the stream and to ensure that each container had properly started.

The intensity of the event stream ranged from 500 events being generated per second to 1500 events. Theodolite started each benchmark test with 1000 events per second being generated. If the algorithm managed to process the events without the lag trend, i.e. how many more events are ready to be processed in $t_i$ then where there in $t_{i-1}$, the second part of each test would increase the workload to 1500 events per second. If the algorithm did not process the events fast enough and the lag trend exceeded the set amount the produced load would be decreased to 500 events per second. The threshold of lag increase was set to 15 events meaning that at the speed of between 500 and 1500 events per second virtually any lag increase resulted in the algorithm not passing the workload.

The event stream generated events based on two different Petri nets seen in Figure A.1 and in Figure A.2. We ensured that the Petri nets display different characteristics, i.e. some form of loops, decision gates, and concurrent activities. The Petri net seen in Figure A.1 implements two loops. These include going from the activity "Waiting for Material" back to the activity "Pass to Production", as well as going from the activity "Waiting for sending" to the activity "Packaging complete". The second Petri net seen in Figure A.2 includes a concurrency at the activities "Pay" and "Record", meaning they both need to be executed before the activity “Manufacture”. The results of our tests are presented below. We first present results related to only one instance of each algorithm running and being tested on both Petri nets seen in

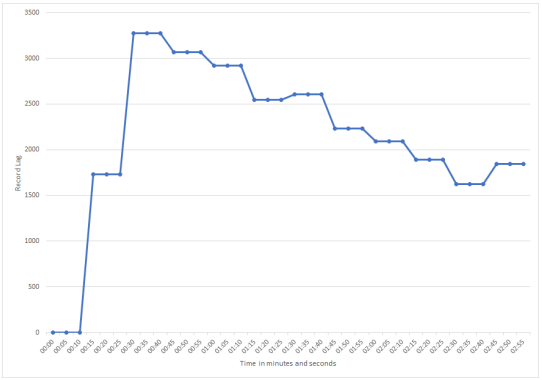



**(a)** Lag Record at 1000 events per second

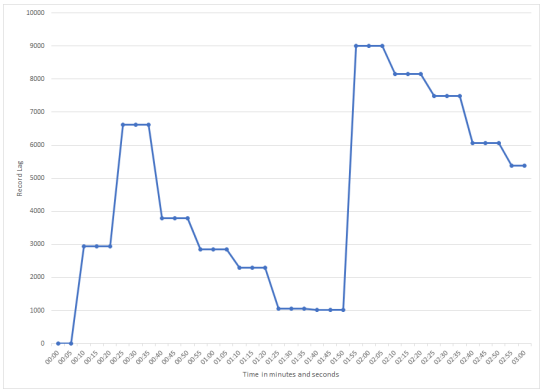



**(b)** Lag Record at 1500 events per second

**Figure 5.1.** Single Prefix-alignment algorithm tested on A.1




**(a)** Lag Record at 1000 events per second




**(b)** Lag Record at 1500 events per second

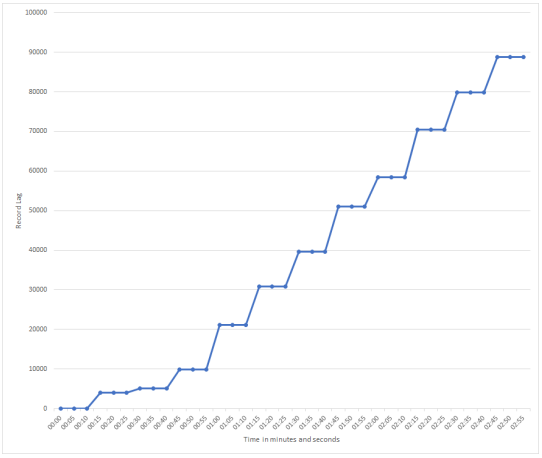
**Figure 5.2.** Single Behavioural Pattern algorithm tested on A.1

Figures A.1 and A.2. Later we present the results related to each algorithm having three instances running in different containers simultaneously.

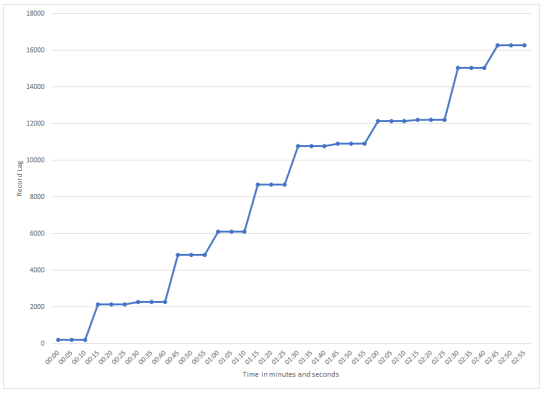
## 5.2 Evaluation

Due to the inconsistency of Kubernetes creating and closing the containers for the conformance checking algorithms and the event streams in order, we observe some spikes of sudden lag record at the beginning or the end of our graphs.

The spike at the beginning denotes an occasional late start of the event stream container by Kubernetes after the container with the conformance checking algorithm is already running. The spike at the end is often due to the fact that the container hosting the conformance checking algorithm is being terminated before the event stream, resulting in the continuous generation of events with a process present to process them.

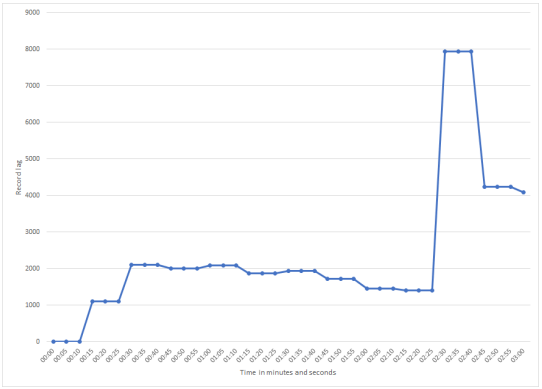



**(a)** Lag Record at 1000 events per second

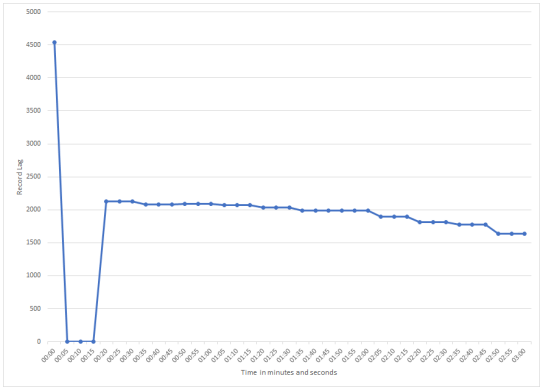



**(b)** Lag Record at 500 events per second

**Figure 5.3.** Single Token-based replay tested on A.1

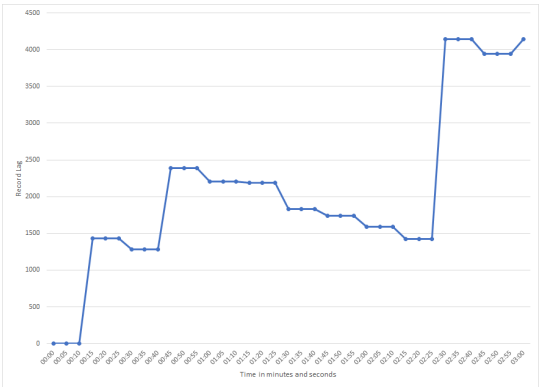



**(a)** Lag Record at 1000 events per second

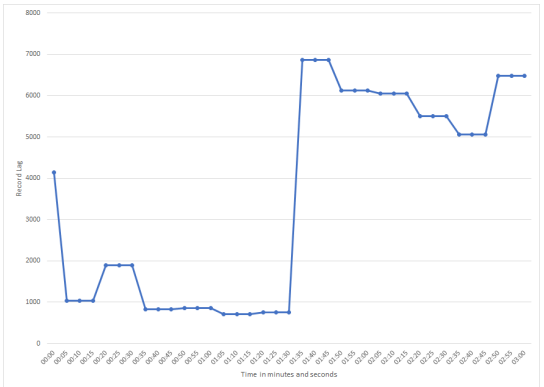



**(b)** Lag Record at 500 events per second

**Figure 5.4.** Single Prefix-alignment algorithm tested on A.2

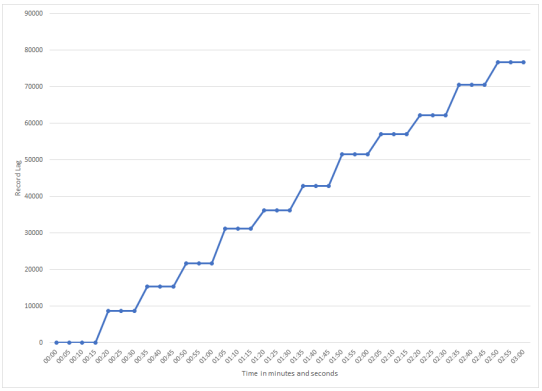



**(a)** Lag Record at 1000 events per second

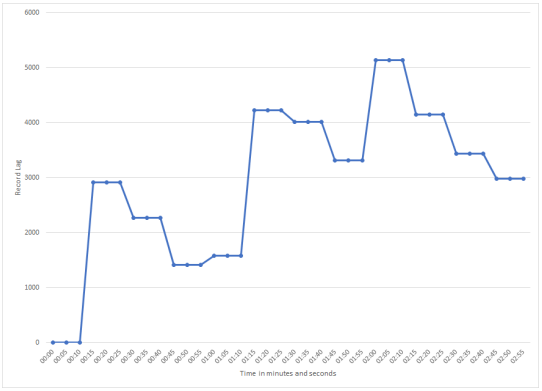



**(b)** Lag Record at 1500 events per second

**Figure 5.5.** Single Behavioural Pattern algorithm tested on A.2




**(a)** Lag Record at 1000 events per second




**(b)** Lag Record at 500 events per second

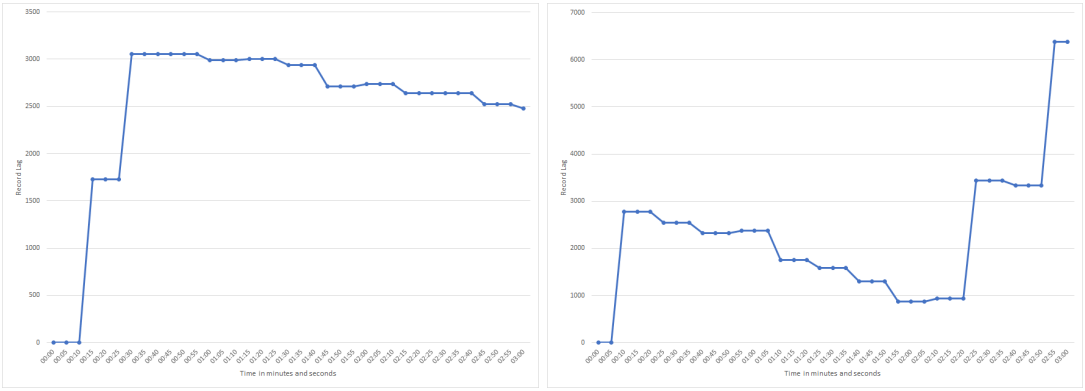
**Figure 5.6.** Single Token-based replay tested on A.2

We can observe in Figures 5.2a and 5.1a that the prefix-alignment and behavioural pattern algorithm handle the events coming from the event stream at a thousand messages per second quite well. As seen in Figure 5.3a the token-based approach already struggles, which it continues in a lower workload of only 500 messages per second as seen in Figure 5.3b. While the prefix-alignment and the behavioural pattern algorithms start to struggle at 1.5 thousand messages per second as seen in Figures 5.1b and 5.2b. The behavioural pattern algorithm still shows a clear trend downward in record lag, despite inexplicable spikes.

Behavioural pattern and the prefix-alignment algorithms performed worse on the Petri net seen in Figure A.2, which included concurrency. The token-based replay did not process the starting one thousand events per second based on the second Petri net either as seen in Figure 5.6a. However, the token-based replay algorithm showed more promising performance at 500 events per second. While still struggling, the algorithm managed to lower the record lag at times, which is visualised in Figure 5.6b. Behavioural patterns and prefix-alignment performed adequately on 1000 events per second based on the second Petri net, as Figures 5.5a and 5.4a show. However, the prefix-alignment algorithm failed the Theodolites benchmark test due to its enormous spike in record lag, and did not perform convincingly at 500 messages per second either (seen in Figure 5.4b). While 1500 events per second led to spikes in the record lag for the behavioural pattern algorithm observable in Figure 5.5b the algorithm still recovered after each spike and continued to process more events than the stream produced.

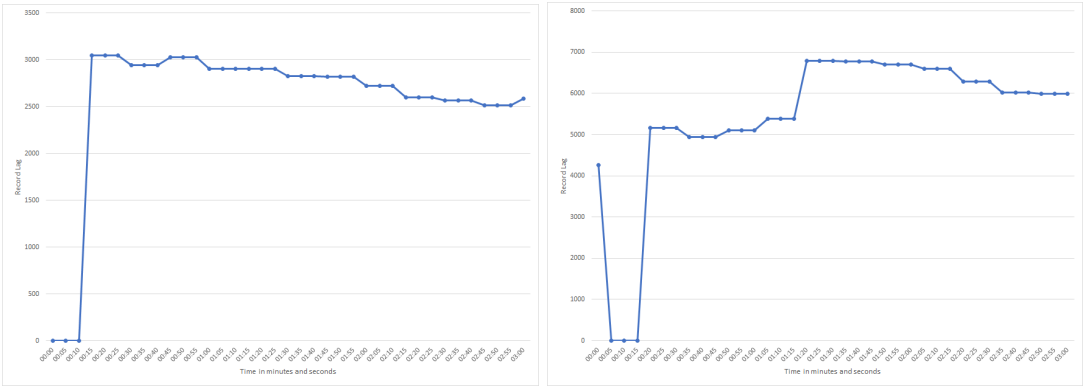
Following the benchmark test of each algorithm with a single instance, we performed each test again in the same constellation without changing the boundary condition of the individual tests but for the number of parallel running algorithms. We increased the number to three conformance checking algorithms of the same type running at the same time for each test.

Figures 5.7a and 5.10a show that the prefix-alignment algorithm handles the event stream of 1000 events per second on both Petri nets with a steady decline in record lag. Figure 5.7b also shows a decline, however, spikes towards the end rapidly increase the record lag. The



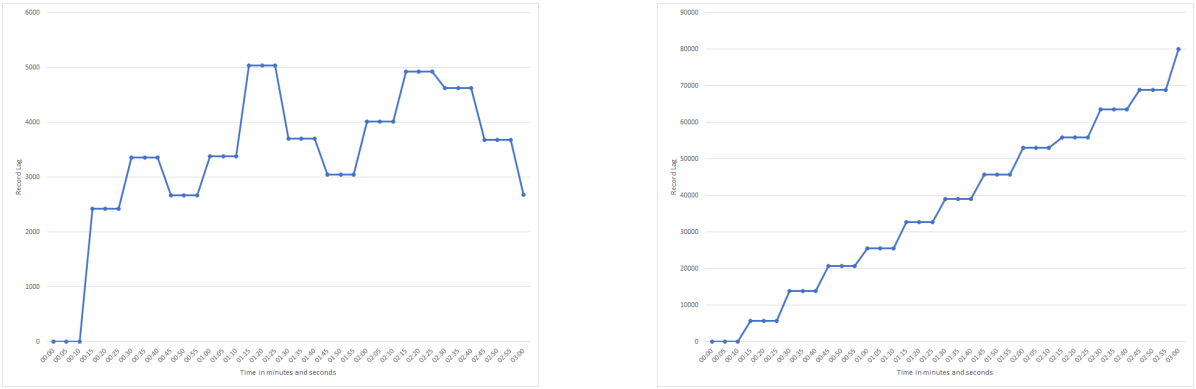

**(a)** Lag Record at 1000 events per second **(b)** Lag Record at 1500 events per second

**Figure 5.7.** Prefix Alignment with 3 Replica on Petri net A.1



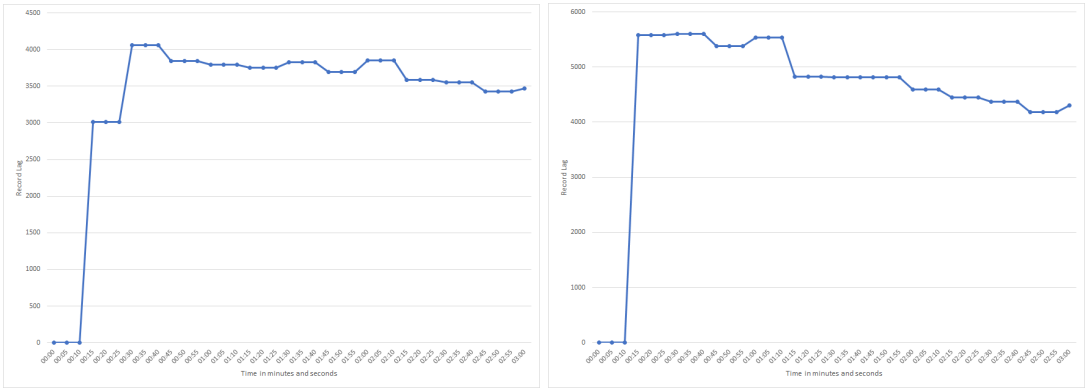

**(a)** Lag Record at 1000 events per second **(b)** Lag Record at 1500 events per second

**Figure 5.8.** Behavioural Patterns with 3 Replica on Petri net A.1



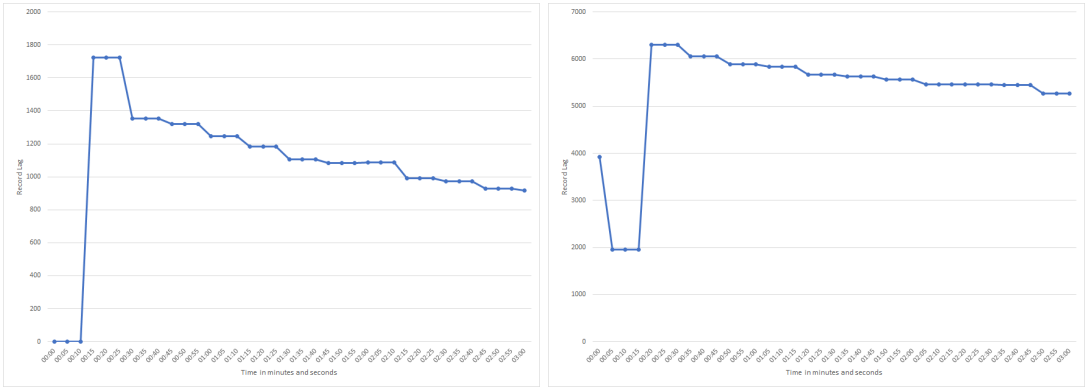

**(a)** Lag Record at 1000 events per second **(b)** Lag Record at 1500 events per second

**Figure 5.9.** Token-based replay with 3 Replica on Petri net A.1




**(a)** Lag Record at 1000 events per second **(b)** Lag Record at 1500 events per second

**Figure 5.10.** Prefix-alignments with 3 Replica on Petri net A.2




**(a)** Lag Record at 1000 events per second **(b)** Lag Record at 1500 events per second

**Figure 5.11.** Behavioural Patterns with 3 Replica on Petri net A.2

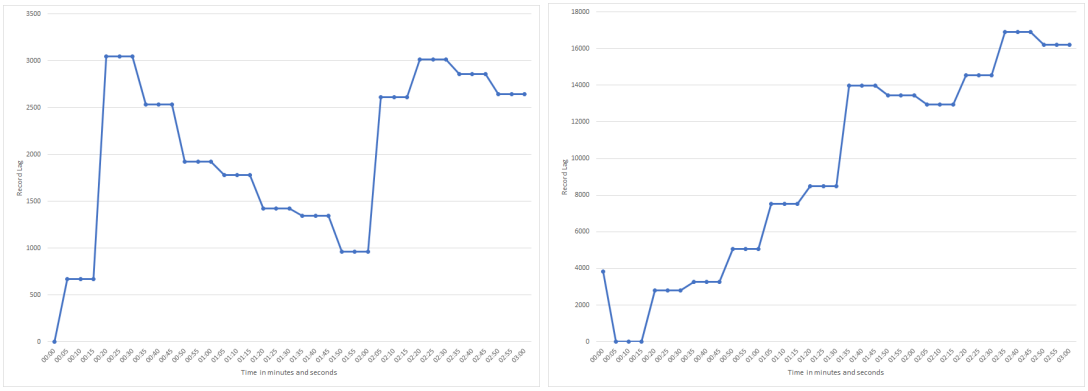



**Figure 5.12.** Token-based replay with 3 Replica on Petri net A.2

algorithm was not able to reduce the record lag in time, making it uncertain if it would have. Figure 5.10b shows a steady decline in record lag, showing that three parallel prefix-alignment containers can handle what a single container could previously not with the same resources.

Figures 5.8a and 5.11a both show, similarly to prefix-alignment, that the record lag continuously decreases. While Figures 5.8b and 5.11b follow the trend of the behavioural pattern algorithm being able to handle the tasked workload, Figure 5.8b does record one spike, that the behavioural pattern algorithm processes.

Token-based replay passes the Theodolite benchmark test at a workload of 1000 events per second when running three instances of the algorithm in parallel instances. This can be observed in Figure **??** where the record lag recedes despite recorded spikes. Figure 5.9a indicates that 1000 events per second might already be around the maximum workload the token-based replay algorithm can handle. This is shown by the fact that the record lag increases and subsides again throughout the benchmark, with no significant difference in record lag between the beginning and the end. Both Figures 5.9b and **??** support this by showing that token-based replay can not handle the increased workload of 15000 events per second.

## 5.3 Result

The token-based replay algorithm benefits the most from parallelism, significantly reducing the record lag buildup as well as increasing the number of processable events per second. The behavioural pattern algorithm showed the least improvement when conducted in a parallel run of three, only slightly varying in record lag. Prefix-alignments also benefited from parallelism. However, they benefited slightly more in reducing record lag than behavioural patterns did, as well as succeeding in the maximum imposed workload.

Table 5.1 summarises whether each algorithm handled each workload, i.e. how many events the event stream generated per second, in either Petri net. Prefix-alignment algorithm has been abbreviated to Prefix, while Behavioural Pattern has been abridged to BHP and Token-based replay has been shortened to TBR. Table 5.1a shows each algorithm with only

**Table 5.1.** Performance of algorithm on Petri nets with varying degrees of workload

| Algorithm | Petri net | Workload 500 | 1000 | 1500 |
|---|---|---|---|---|
| Prefix | A.1 | √ | √ | × |
| Prefix | A.2 | √ | √ | × |
| BHP | A.1 | √ | √ | √ |
| BHP | A.2 | √ | √ | × |
| TBR | A.1 | × | × | × |
| TBR | A.2 | × | × | × |

**(a)** Each algorithm singled instanced

| Algorithm | Petri net | Workload 500 | 1000 | 1500 |
|---|---|---|---|---|
| Prefix | A.1 | √ | √ | √ |
| Prefix | A.2 | √ | √ | √ |
| BHP | A.1 | √ | √ | √ |
| BHP | A.2 | √ | √ | √ |
| TBR | A.1 | √ | √ | × |
| TBR | A.2 | √ | √ | × |

**(b)** Each algorithm with 3 instances

one instance. The prefix-alignment algorithm was able to handle up to 1000 events per second on both Petri nets shown in Figures A.1 and A.2. In this case, it did not manage to pass the benchmark for a workload of 1500 events per second. The behavioural pattern algorithm has shown multiple spikes in record lag when performing 1500 events per second on the second Petri net, however, the first Petri net still showed rapidly decreasing record lag after a spike, proving that the algorithm can handle the workload. A single-instanced token-based replay algorithm could not handle any of the workloads we proposed on it.
Table 5.1b displays the result of each benchmark when running three instances of each algorithm in parallel. Prefix-alignments comfortably handled each workload in each Petri net, only showing one spike in Figure 5.7b. This showed an improvement over the single-instanced benchmark, where prefix-alignments did not succeed in a workload of 1500 events per second in either Petri net. Token-based replay greatly increased its performance due to parallelism. We can see in Figure 5.1b that the token-based replay is now capable of handling workloads of up to 1000 events per second. This is in contrast to the single-instanced benchmarks where token-based replay did not handle 500 events per second.

Behavioural patterns saw the smallest increase in performance by running three instances of the same algorithm in parallel. The parallelism of running the algorithm three times resulted in generally lower record lag throughout the benchmarks. It also resulted in the behavioural pattern algorithm confidently managing a workload of 1500 events per second on both Petri nets, only showing one spike in Petri net one seen in Figure 5.8b.

It stands to reason that the behavioural pattern algorithm performs the best regarding workload, out of the three online conformance checking algorithms presented in this paper. While the performance increase of the token-based replay algorithm with three instances was significant, it was still outperformed by the behavioural pattern algorithm. However, a general decrease in performance through parallelism was discovered, making it the right choice over single-instanced deployment. Considering the nature of event streams, that the beginning of a case might no longer be recorded, behavioural patterns also provide the added benefit that the event stream can already run and the conformance comparison of the trace to the Petri net is not negatively influenced, unlike with token-based replay and prefix-alignment. This leads to the behavioural pattern being the first choice of the three presented, with prefix-alignments being a close follow-up and token-based replay trailing behind.

Chapter 6

# Discussion

In this section limitations in the implementation of our online conformance checking algorithms are briefly discussed. After that, some basic assumptions of knowledge about the algorithms are defined and explained.

## 6.1 Limitations

Due to time constraints, an adjustment was made to the prefix-alignment algorithm such that prefix-alignments are now calculated based on behaviour, as opposed to token replay. This results in a less precise conformance measure when dealing with concurrent activities.

Further, the behavioural pattern algorithm is not fully optimised for convoluted Petri nets with a multitude of parallel activities. Both algorithms struggle with concurrency because they are both based on behavioural patterns which do not offer a precise way of verifying whether an activity has already passed another concurrent activity without further effort.

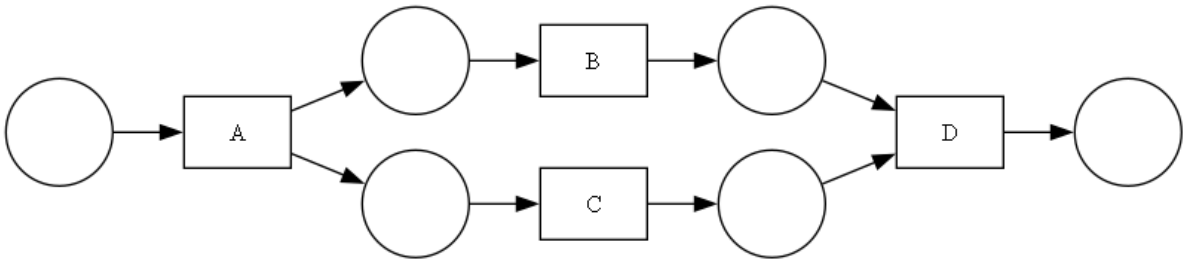



**Figure 6.1.** Short Petri net with one and gate, made with [BZS23]

Take the Petri net seen in Figure 6.1. Derived from this Petri net we can determine the set of all behavioural patterns comprised of $B = (a > b), (a > c), (b > d), (b > c), (c > b), (c > d)$. And the set of all alignments $\Gamma = (a, b, c, d), (a, c, b, d)$. Both these sets are rather unambiguous. The concurrent activities do not inflate the sets and allow for otherwise unwanted behaviour. However, should the Petri net display more concurrent behaviour such as seen in the Petri net seen in Figure 6.2, distinguishing between mere concurrency and deviation will become increasingly difficult. Take the set of behavioural pattern $B = (a > b), (a > c), (b > c), (b > f), (b > g), (b > i), (b > d), (b > e), (c > b), (c > d), (c > e), (c > h), (c > f), (c > g), (d < e), (d > h), (d > c), (d > f), (d > g), (d > i), (e > d), (e > h), (e > c), (e > f), (e > g), (e > h), (h > j), (h > c), (h > f), (h > g), (h > i), (f > g), (f > i), (f > b), (f > d), (f > e), (f >$

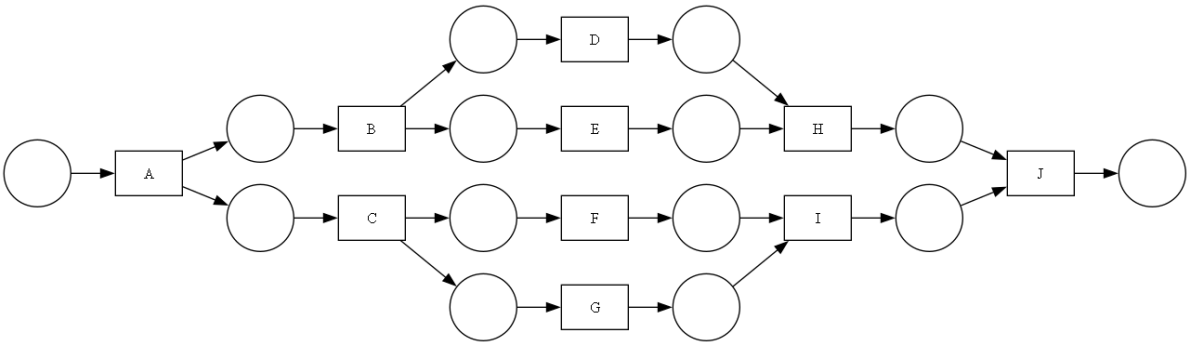



**Figure 6.2.** Petri net with a multitude of and gates, made in [BZS23]

$h), (g > f), (g > i), (g > b), (g > d), (g > e), (g > h), (i > h), (i > b), (i > d), (i > e), (i > j)$ where almost any behaviour is allowed due to concurrency. This leads to an inflated set of behavioural patterns, in which it is nigh impossible to distinguish whether an incoming new activity conforms with the process model seen in Figure 6.2. Behaviour allowed by this net is virtually anything that moves forward in the process.

Another hurdle we encountered were the possible deviation that could occur in an event stream, which an online conformance checking algorithm must be able to handle. Consequentially, the algorithms were designed to handle “skipped activities”. However, the method for accepting a new trace, which is the same for all algorithms, filters out all "added activity" deviations. Further, the simulated event stream in the benchmark tests is not designed to produce such "added activities" at random. As such, the algorithms are not tested on their ability to handle this kind of deviation. Nevertheless, they all possess a method for handling "added activities".

Concerning the potential range of process models we expect a model comprising unique activity names, ensuring that each activity from the event stream is unambiguously associated with one activity in the process model. Furthermore, we assume the availability of a Petri net for our token-based replay algorithm. Additionally, we anticipate possessing knowledge about the set $B$ of all behavioural patterns.

Chapter 7

# Conclusion

In this paper, we introduced three popular online conformance-checking algorithms. These algorithms were chosen based on their number of citations and relevance in the research field. We implemented and ensured fair testing conditions for all algorithms. Our benchmarks were designed to test the scalability of each algorithm regarding their ability to handle increasing workloads, i.e. how many events they could process per second. Benchmark tests were also performed to test how an increase in simultaneously running instances of the algorithm would show improved performance over a single instance of each algorithm. The results show that, albeit closely followed by the prefix-alignment algorithm, the behavioural pattern algorithm performs best. Further, running the algorithms in parallelism, i.e. having multiple instances running at the same time, shows an increase in performance across each algorithm, making it well worth it.

## 7.1 Future Outlook

As discussed in section 6.1 the prefix-alignment and the behavioural patterns algorithm both struggle with complex concurrency activities. A possible solution would be to include more information in the process model, such as time constraints which could exclude some patterns due to some activities taking as long as other activities take together. Another method is by moving along the Petri net tracking the tokens fired and checking if certain activities have already been executed before. The token-based replay alignment algorithm replays incoming events given all previous activities in the trace and is thus quite stable at handling concurrency as the algorithm looks for a way to reach the current activity based on the trace. The prefix-alignment algorithm presented in [ZBH+17] assumes an algorithm that searches for an optimal prefix-alignment $\overline{\gamma}$ in a trace when a new activity is accepted. In our implementation, we forfeited the token variant due to time restraints and utilised a method of finding optimal prefix-alignment using behavioural patterns. In future works, tokens or time restraints could be included, notice however that token replay is time consuming.

Regarding the prefix-alignment algorithm presented in [ZBH+17] the algorithm assumes a function $\overline{a}$ that heuristically calculates the optimal prefix between two activities that can not be executed in direct succession with reference to the model. This means that whenever a skip deviation occurs $\overline{a}$ calculates an optimal prefix from the current trace for the case to the new activity. Because the algorithm does not handle the warm start scenario and needs to be started before the stream begins either way, it might be feasible to calculate a set of optimal prefix-

alignments beforehand so that further time can be saved during the execution of the algorithm.

The confidence value of the behavioural pattern algorithm presented in [BZA+18] and implemented in section 4.1 only considers the length of the trace. This is calculated by taking the distance from the start of the trace to the end of the process model and the distance from the current end of the trace to the end of the process model. It does not, however, consider the amount of incorrectly observed behaviour, nor the ratio of correct observed behaviour to incorrect observed behaviour. As the confidence value aims at predicting the possibility that the conformance score remains stable the actual conformance of the current trace might be incorporated in the future for a more accurate prediction.

Further research can also be conducted to perform these tests with more online conformance checking algorithms, e.g. [SMR20; LBM+21; BC18; MCA14]. The algorithm benchmarked in this paper could also be imported, with possible improvements, into a standard framework, such as the ProM tool.
With more online conformance checking algorithms implemented, a deeper set of benchmark tests might be advantageous. Such benchmarks could include Petri nets with invisible transitions, i.e. transitions that do not correspond to an activity within the business process. Other option are to make the workload steps finer and lower the distance between each benchmark test.

Appendix A

# Petri Nets used for Tests

Below is the first Petri net displaying the process of the first patch of benchmark tests.

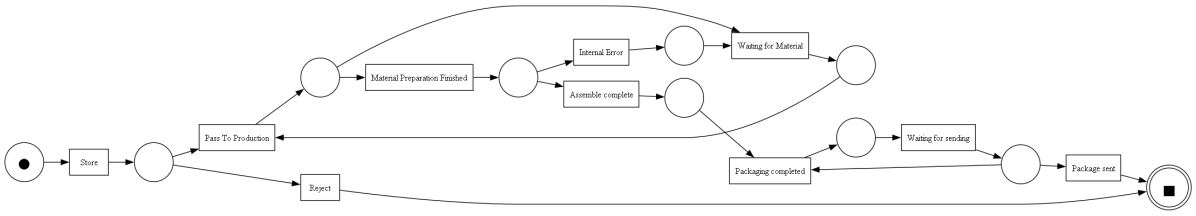



**Figure A.1.** Petri net used for the first set of tests using Theodolite

Following is the second Petri net displaying the underlying process that was used for the second patch of benchmark tests.

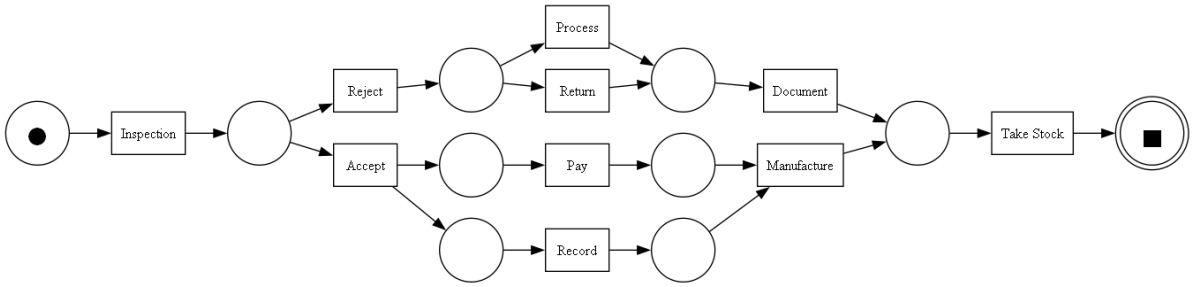



**Figure A.2.** Petri net used for the second set of tests using Theodolite

# Bibliography


[Aal11] Wil M. P. van der Aalst. *Process mining: discovery, conformance and enhancement of business processes*. Springer Berlin Heidelberg, 2011. ISBN: 9783642193453. DOI: 10.1007/978-3-642-19345-3.

[Aal12] Wil van der Aalst. "Process mining: overview and opportunities". In: *ACM Transactions on Management Information Systems* 3.2 (July 2012), pp. 1–17. ISSN: 2158-6578. DOI: 10.1145/2229156.2229157.

[Aal16] Wil van der Aalst. *Process mining*. Springer Berlin Heidelberg, 2016. ISBN: 9783662498514. DOI: 10.1007/978-3-662-49851-4.

[Adr14] A. Adriansyah. "Aligning observed and modeled behavior". English. Phd Thesis 1 (Research TU/e / Graduation TU/e). Mathematics and Computer Science, 2014. ISBN: 978-90-386-3574-3. DOI: 10.6100/IR770080.

[BA21] Alessandro Berti and Wil M. P. van der Aalst. "A novel token-based replay technique to speed up conformance checking and process enhancement". In: *Transactions on Petri Nets and Other Models of Concurrency XV*. Springer Berlin Heidelberg, 2021, pp. 1–26. ISBN: 9783662630792. DOI: 10.1007/978-3-662-63079-2_1.

[BC18] Andrea Burattin and Josep Carmona. "A framework for online conformance checking". In: *Business Process Management Workshops*. Springer International Publishing, 2018, pp. 165–177. ISBN: 9783319740300. DOI: 10.1007/978-3-319-74030-0_12.

[BZA+18] Andrea Burattin, Sebastiaan J. van Zelst, Abel Armas-Cervantes, Boudewijn F. van Dongen, and Josep Carmona. "Online conformance checking using behavioural patterns". In: *Business Process Management*. Springer International Publishing, 2018, pp. 250–267. ISBN: 9783319986487. DOI: 10.1007/978-3-319-98648-7_15.

[BZS23] Alessandro Berti, Sebastiaan van Zelst, and Daniel Schuster. "Pm4py: a process mining library for python". In: *Software Impacts* 17 (Sept. 2023), p. 100556. ISSN: 2665-9638. DOI: 10.1016/j.simpa.2023.100556.

[CH09] Graham Cormode and Marios Hadjieleftheriou. "Methods for finding frequent items in data streams". In: *The VLDB Journal* 19.1 (Dec. 2009), pp. 3–20. ISSN: 0949-877X. DOI: 10.1007/s00778-009-0172-z.

[CSS+09] Graham Cormode, Vladislav Shkapenyuk, Divesh Srivastava, and Bojian Xu. "Forward decay: a practical time decay model for streaming systems". In: *2009 IEEE 25th International Conference on Data Engineering*. IEEE, Mar. 2009, pp. 138–149. DOI: 10.1109/icde.2009.65.

[DSM+19] Sebastian Dunzer, Matthias Stierle, Martin Matzner, and Stephan Baier. "Conformance checking: a state-of-the-art literature review". In: *Proceedings of the 11th International Conference on Subject-Oriented Business Process Management*. S-BPM ONE'19. ACM, June 2019. DOI: 10.1145/3329007.3329014.

[Has21] Wilhelm Hasselbring. "Benchmarking as empirical standard in software engineering research". In: *Evaluation and Assessment in Software Engineering*. EASE 2021. ACM, June 2021, pp. 365–372. DOI: 10.1145/3463274.3463361.

[HH21] Sören Henning and Wilhelm Hasselbring. "Theodolite: scalability benchmarking of distributed stream processing engines in microservice architectures". In: *Big Data Research* 25 (July 2021), p. 100209. ISSN: 2214-5796. DOI: 10.1016/j.bdr.2021.100209.

[HH22] Sören Henning and Wilhelm Hasselbring. "A configurable method for benchmarking scalability of cloud-native applications". In: *Empirical Software Engineering* 27.6 (Aug. 2022). ISSN: 1573-7616. DOI: 10.1007/s10664-022-10162-1.

[KAH+15] Jóakim v. Kistowski, Jeremy A. Arnold, Karl Huppler, Klaus-Dieter Lange, John L. Henning, and Paul Cao. "How to build a benchmark". In: *Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering*. ICPE'15. ACM, Jan. 2015, pp. 333–336. DOI: 10.1145/2668930.2688819.

[KKV03] Victor Khomenko, Maciej Koutny, and Walter Vogler. "Canonical prefixes of petri net unfoldings". In: *Acta Informatica* 40.2 (Oct. 2003), pp. 95–118. ISSN: 1432-0525. DOI: 10.1007/s00236-003-0122-y.

[LBM+21] Wai Lam Jonathan Lee, Andrea Burattin, Jorge Munoz-Gama, and Marcos Sepúlveda. "Orientation and conformance: a hmm-based approach to online conformance checking". In: *Information Systems* 102 (Dec. 2021), p. 101674. ISSN: 0306-4379. DOI: 10.1016/j.is.2020.101674.

[Leo22] Massimiliano de Leoni. "Foundations of process enhancement". In: *Process Mining Handbook*. Springer International Publishing, 2022, pp. 243–273. ISBN: 9783031088483. DOI: 10.1007/978-3-031-08848-3_8.

[MAE04] Ahmed Metwally, Divyakant Agrawal, and Amr El Abbadi. "Efficient computation of frequent and top-k elements in data streams". In: *Database Theory - ICDT 2005*. Springer Berlin Heidelberg, 2004, pp. 398–412. ISBN: 9783540305705. DOI: 10.1007/978-3-540-30570-5_27.

[MCA14] Jorge Munoz-Gama, Josep Carmona, and Wil M.P. van der Aalst. "Single-entry single-exit decomposed conformance checking". In: *Information Systems* 46 (Dec. 2014), pp. 102–122. ISSN: 0306-4379. DOI: 10.1016/j.is.2014.04.003.

[MP95] K. L. McMillan and D. K. Probst. "A technique of state space search based on unfolding". In: *Formal Methods in System Design* 6.1 (Jan. 1995), pp. 45–65. ISSN: 1572-8102. DOI: 10.1007/bf01384314.

[Mur89] T. Murata. "Petri nets: properties, analysis and applications". In: *Proceedings of the IEEE* 77.4 (Apr. 1989), pp. 541–580. ISSN: 0018-9219. DOI: 10.1109/5.24143.

[RA08] A. Rozinat and W.M.P. van der Aalst. "Conformance checking of processes based on monitoring real behavior". In: *Information Systems* 33.1 (Mar. 2008), pp. 64–95. ISSN: 0306-4379. DOI: 10.1016/j.is.2007.07.001.

[Rei24] Hendrik K. Reiter. "Scalability benchmarking of the realtime heuristics miner implemented as a microservice architecture". https://oceanrep.geomar.de/id/eprint/59777/. MA thesis. Kiel University, 2024.

[SMR20] Florian Stertz, Juergen Mangler, and Stefanie Rinderle-Ma. *Temporal conformance checking at runtime based on time-infused process models*. 2020. DOI: 10.48550/ARXIV.2008.07262.

[VAN98] W. M. P. VAN DER AALST. "The application of petri nets to workflow management". In: *Journal of Circuits, Systems and Computers* 08.01 (Feb. 1998), pp. 21–66. ISSN: 1793-6454. DOI: 10.1142/s0218126698000043.

[Vit85] Jeffrey S. Vitter. "Random sampling with a reservoir". In: *ACM Transactions on Mathematical Software* 11.1 (Mar. 1985), pp. 37–57. ISSN: 1557-7295. DOI: 10.1145/3147.3165.

[WSA+22] Indra Waspada, Riyanarto Sarno, Endang Siti Astuti, Hanung Nindito Prasetyo, and Raden Budiraharjo. "Graph-based token replay for online conformance checking". In: *IEEE Access* 10 (2022), pp. 102737–102752. ISSN: 2169-3536. DOI: 10.1109/access.2022.3208098.

[ZBH+17] Sebastiaan J. van Zelst, Alfredo Bolt, Marwan Hassani, Boudewijn F. van Dongen, and Wil M. P. van der Aalst. "Online conformance checking: relating event streams to process models using prefix-alignments". In: *International Journal of Data Science and Analytics* 8.3 (Oct. 2017), pp. 269–284. ISSN: 2364-4168. DOI: 10.1007/s41060-017-0078-6.

[ZHD21] Rashid Zaman, Marwan Hassani, and Boudewijn F. van Dongen. *A framework for efficient memory utilization in online conformance checking*. 2021. DOI: 10.48550/ARXIV.2112.13640.