Towards Process Mining on Kafka Event Streams

Maxim Vidgof[1]

[1]Vienna University of Economics and Business (WU Wien), Welthandelsplatz 1, 1020 Vienna, Austria

Abstract
Process mining is a family of techniques to analyze business processes based on the event logs produced during their execution. While most process mining techniques rely on readily available event logs, extracting them poses challenges and limits their availability and usability. As event-driven architecture is gaining momentum, event streaming platforms like Apache Kafka can solve this problem. This paper proposes an architecture that allows using Kafka both as message broker for the running process as well as data store directly enabling process mining, including streaming process mining on the same dataset.

Keywords
Process mining, Event stream, Apacke Kafta

1. Introduction

Process mining can help organizations to monitor processes, find bottlenecks and improvement potential. However, as information systems involved in the business processes are often backed by relational databases without a notion of event, extracting the data in a format suitable for process mining is challenging. With the emergence of event-driven architecture as new state-of-the-art for building loosely coupled information systems, opportunities arise to use event streams both for communication between the components of such systems and for process mining simultaneously.

In this paper, a data architecture for event streams is proposed. This architecture allows to structure event data in such way that it can be used both by automated services executing the process as well as by process mining tools, including online process monitoring. The rest of the paper is structured as follows: Section 2 provides the necessary background, Section 3 presents the proposed architecture, Section 4 describes related work and Section 5 concludes the paper.

2. Background

2.1. Process mining

Process mining is a family of techniques to analyze business processes based on event logs produced during their execution [1]. These techniques can further be divided into *discovery* – extracting process models from event logs, *conformance* – comparing an event log with the process model and detecting deviations, and *enhancement* – extending or improving an existing

ZEUS'24, February 29–March 1, 2024, Ulm, Germany
maxim.vidgof@wu.ac.at (M. Vidgof)
https://nm.wu.ac.at/nm/vidgof (M. Vidgof)
0000-0003-2394-2247 (M. Vidgof)

process model using information from an event log [2]. Event logs are collections of timestamped event records of execution of work items or other process-relevant events, recorded by BPMSs and enterprise systems such as CRM and ERP. Event logs are assumed to contain data related to a single process, and each event in the log must refer to a single process instance or case as well as to an activity. Events can also contain additional information such as resource or cost.

While process mining techniques rely on event logs, extracting them from the information systems supporting the process is not trivial. If a process is orchestrated by a BPMS, event data can be easily extracted in the right format [3]. In other cases, however, the only source of data are relational databases, whose primary task is to store the output of process activities rather than events themselves. Moreover, the unique case identifier that is required may not be stored across all systems, and in some cases business processes cannot have a single case identifier, which led to emergence of object-centric process mining [4].

2.2. Streaming Process Mining

Streaming process mining refers to a set of techniques and tools dealing with processing streaming event data [5]. As opposed to other process mining techniques that analyze static event logs, streaming process mining captures and analyzes events as they happen, in near-real time. Typical use cases include conformance checking to immediately detect violations and counteract timely, as well as process discovery. A major complication in streaming scenario lies in the limited amount of computational resources, especially memory, making impossible to employ traditional process mining techniques relying on complete event logs. Instead, different techniques like sliding window, problem reduction, offline (pre-)computation as well as hybrid approaches are applied, leading to efficient processing at the cost of data completeness.

2.3. Event Sreaming and Apache Kafka

Event streaming is the practice of capturing data in real-time from different sources in form of streams of events, storing these streams durably for later retrieval, processing and reacting to these streams in real-time and retrospectively, as well as routing these events to specified destinations [6]. Event streaming allows to loosely couple different systems by using a pub/sub model. Apache Kafka[1] is an event streaming platform allowing to *publish* and *subscribe* to streams of events, *store* them durably and *process* them as they occur or retrospectively.

An *event* in Kafka has a *key*, *value*, *timestamp* and optional metadata. The events are organized and stored in *topics*. A topic can have multiple producers and multiple consumers. In contrast to traditional messaging systems, the events are not deleted after consumption. This allows multiple consumers to process the events at their own pace, as well as read the entire topic from the beginning. For each topic, the retention period after which the events are deleted can be set. Importantly, this retention period can also be set to infinity, allowing for permanent storage of events. To allow for scalability, topics are split into *partitions*, which are scattered across multiple brokers. Events with the same key are always written to the same partitions, and Kafka guarantees the order preservation for the events in the same partition.

[1]https://kafka.apache.org

3. Proposed Architecture

This section presents the architecture for a Kafka-based system supporting process mining. Section 3.1 gives an overview of the system architecture. Section 3.2 describes the topic architecture, i.e. how events are split into topics. Section 3.3 describes the proposed serialization format of events in Kafka. Section 3.4 discusses the role of a BPMS in the proposed architecture, and Section 3.5 explains how process mining can be performed.

3.1. Overview

The proposed system architecture is shown in Figure 1. The central element is the *Kafka* event streaming platform that connects the other components. Importantly, Kafka serves as a communication backbone for the *Services* responsible for executing the tasks in the process. Depending on the environment where such system is deployed, there might already be a *BPMS* present, in which case it also should communicate with Kafka. Essentially, Kafka serves as the single source of truth for all traditional *process mining* components and for *streaming process mining*, which might have different data requirements.

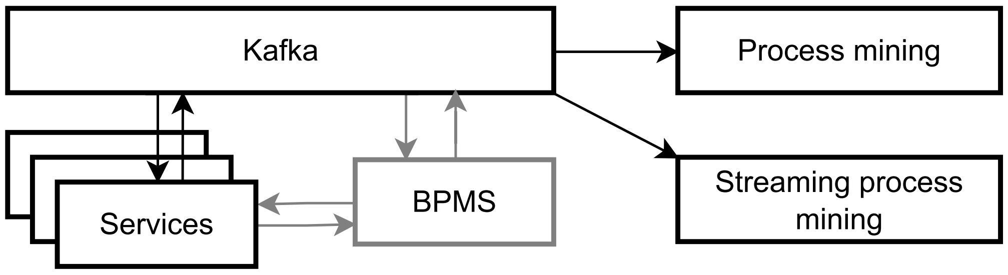

Figure 1: Proposed architecture.

3.2. Topic Architecture

In Kafka, event streams are stored in topics. The number and semantics of the topics is not predefined and should be selected according to the domain specifics. This consideration is crucial since Kafka only guarantees order preservation for events within one partition of the same topic. Thus, the events that need strict order, e.g. events within one case, should be stored in one partition. This is achieved by assigning the same *key* to all these events, e.g. a case ID. In this section, four strategies are proposed.

Case ID as topic. In this case, for each case that has started, a new topic has to be created in the broker. All further events referencing this case ID should be sent to this topic. Consumers should regularly poll the broker for new topics and subscribe to new topics that appear. Each case topic should have only one partition to ensure the event order within the case. Such architecture would, however, suffer from a number of other issues. First, the existing system supporting the process will have to be re-designed to accommodate for the new topics. Also, having such number of topics will introduce unnecessary overhead, especially as the number of old inactive topics grows. Finally, while such architecture is very much oriented on traditional event logs with single case ID, object-centric logs are not supported.

Activity as topic. This architecture most closely resembles known microservice best practices. Each microservice (or other system) responsible for a task writes the corresponding events in its topic. The events are later picked up by the services relying on them. Case IDs can be used as event keys, allowing for simple partitioning. For object-centric processes, the object ID of some object involved in the activity can be used. A benefit of such strategy is the minimality of change necessary in the system if it already followed event-driven architecture. In the ideal case, the system can remain untouched (except the schema update, see Section3.3) and additional consumers can be seamlessly added. Events having the same key (case or object ID) will benefit from consistent partitioning and total order within partitions. A downside of such system is, however, the partial view that every topic would have on the execution. The streams will have to be aggregated (using Kafka Streams API or processing outside of Kafka), and event correlation will be crucial. However, it might be solved with the event keys and timestamps provided by Kafka producer or broker.

Process as topic. Another approach is to store all events related to one business process in one topic. Case IDs will be used as keys for the traditional processes. The keys for the processes with no case ID are subject to discussion, however, it is reasonable to use the ID of some object and perform event correlation outside Kafka. The activity label must be included in the event value, and the consuming services will not only have to monitor the right topic but also make sure to only process events with the correct activity label. A clear benefit is that this setting is the closest to the desired output event log. Each topic will simply contain all executions of a process and can almost directly be used in process mining. A downside is the additional overhead on the (micro-)services that will have to process all events and filter them based on the activity label.

Single topic. A variation of the previous strategy, which can be especially useful in scenarios where borders of business processes are unclear. In this case, all events are put into the same topic. The event key for partitioning has to be defined using domain knowledge. A benefit of such strategy is the simplicity of implementation, as it only minimally uses Kafka as a universal message broker. However, in this setup the (micro-)services will suffer from even more filtering overhead. In addition, such single topic can quickly become a data swamp, especially if infinite retention is selected.

3.3. Serialization Format

Kafka itself is schema-less and stores the events as raw bytes, however, it is good practice to use common data exchange formats like JSON[2] , protobuf[3] or Avro[4] . In process mining, on the other hand, XES[5] and OCEL [7] are accepted standards. XES is used for exchanging event logs having a case ID, and uses XML as serialization. However, recently JXES [8] was proposed to support serialization of XES event logs in JSON. OCEL is used for event logs not having a single

[2]https://www.json.org/
[3]https://protobuf.dev/
[4]https://avro.apache.org/
[5]https://xes-standard.org/

case ID, it offers JSON serialization[6] included in the standard (along with the XML and SQLite variants).

This paper proposes using JXES and OCEL JSON as the event formats. It must be noted that OCEL objects cannot be stored in the event streams but rather will have to be reconstructed from the events that created or referenced used them at the event log construction stage.

The payload, or the value of the events should contain all the necessary event attributes. For the traditional logs, it is recommended to store at least `concept:name`, case ID attribute and `time:timestamp` of the producing application. While some of the attributes can be inferred from Kafka events depending on the chosen topic architecture, it improves the readability and unifies the approach regardless of the topic architecture. Note that the case ID on the event level has no standard attribute in XES, thus a custom one, e.g. `caseid` has to be used. Additional attributes should be stored in the same way as in XES event logs. For the object-centric processes, in addition, the referenced objects with their IDs and relevant attributes must be included in the event value. While not all of the stored attributes are necessary for the services processing the events, e.g. case ID, such services should be configured to simply copy these attributes to their output events.

3.4. Role of a BPMS

The proposed architecture can be used both in conjunction with a BPMS as well as without it. It can be distinguished between the following deployment scenarios.

BPMS-centric. In these scenarios, BPMS plays a central role in orchestrating a process. The events have a single case identifier, and external services are heterogeneous, requiring not only event-driven communication but also other methods, such as REST, Long polling and Java API. In these scenarios, a BPMS indeed serves as a single source of truth for process mining, and it would be challenging to meaningfully extract the process data from other sources. However, the events can be constantly extracted from the BPMS into Kafka to enable streaming process mining with minimal system changes.

BPMS-aided. If a BPMS is used to orchestrate a fully-automated process with event-driven services, this theoretically means both the BPMS and Kafka can be used as the primary event store. However, it may be still beneficial to use the proposed event serialization format and topic architecture to off-load data extraction tasks to Kafka. In addition, such scenarios can profit from streaming process mining if the proposed architecture is used.

No-BPMS. For other scenarios, where no BPMS is present, the proposed architecture becomes crucial for enabling process mining. If the process is composed of loosely coupled (micro-)services and systems, having a unified source of data is the only alternative to the complicated and error-prone process of extracting and correlating events (with possibly incomplete information) from heterogeneous internal storages of the respective systems.

3.5. Process Mining

Infinite retention allows to use Kafka both as message broker for the automated services as well as storage for historic data. The proposed architecture directly enables process mining on

[6]https://www.ocel-standard.org/specification/formats/json/

event streams. The last component that is needed for this is a Kafka consumer that connects to the respective topic(s) and extracts the events. If *activity as topic* architecture is chosen, this component also has to order the events using the timestamps provided by Kafka. The extraction of events is straightforward due to the standardized serialization format in the event streams. It must be noted that object-centric event logs might require additional event correlation efforts. The extracted event logs can be easily output as XES or OCEL logs or directly used by a (streaming) process mining tool.

4. Related Work

Apache Kafka has been used in projects related to various topics such as process mining, digital twins, IoT, and software development. However, these projects merely use Kafka as a middleware to exchange data between components. To the best of the author's knowledge, no paper shows explicit considerations about the format of the data and the strategy of using Kafka topics.

[9] uses Kafka as middleware for communication between components of a Digital Twins Cloud Platform. In [10], Kafka is a crucial part of a distributed architecture for real-time monitoring of Roll on/ Roll off (RoRo) terminals. This architecture combines Apache Kafka, Apacke Spark[7] and ProM[8] platforms to perform process mining on the events recorded by the terminals. Kafka is used to transport the event logs in format of streams of events from their source – RoRo terminals – to Apache Spark cluster responsible for the online processing of the events. While it is mentioned that the data source are XES event logs and that the ProM framework is used for process mining, the exact specifics of the data exchange format and schema are not provided.

[11] proposes a framework that applies event streaming to environmental data. A major difficulty in this domain is the incompatibility of data producers and data consumers due to different data formats. The framework solves this problem by using Kafka Connect APIs to receive data from heterogeneous sources and perform lightweight transformations to bring the data into unified format and prepare them for further processing. In [12], Kafka is used to collect events from a CI/CD pipeline of an application. These events are then transformed and output in a format suitable for process mining. [13] presents an IoT-enriched event log for process mining research in smart factories. It uses Kafka to collect additional data from the machines in the smart factory. This data together with the domain ontology is then used to enrich the log produced by the Workflow Management System. It enabled correcting the timestamps and reconstructing unrecorded events.

Camunda process orchestration platform has already developed own Kafka connectors[9] and provides examples of using them for process automation [14]. More broadly, [15] shows how a workflow engine can be used with event streaming platform. However, also here no recommendation on data architecture is provided.

[7]https://spark.apache.org/
[8]https://promtools.org/
[9]https://docs.camunda.io/docs/components/connectors/out-of-the-box-connectors/kafka/

5. Conclusion

Process mining can help organizations gain insights into their processes and find improvement potentials. Data extraction, however, poses significant difficulties as not all systems supporting business processes store the data in a format suitable for process mining. With the advent of event-driven architecture, where event streams are used to communicate between loosely coupled components, these difficulties can be solved by using data architecture suitable for process mining.

In this paper, a system architecture is proposed that uses Kafka event streams both as a platform for communication between systems responsible for the execution of a business process (e.g. BPMS, ERP systems and automated (micro-)services) and as a data storage for online and offline process mining. To this end, this paper proposes data architecture for Kafka topics, event serialization format, and a process mining component connecting the system to state-of-the-art process mining tools and techniques. In addition, it discusses the possible role of a BPMS in such setting.

Future work involves implementing the proposed architecture as well as evaluating its usefulness in simulated and real-life business processes.

References

[1] M. Dumas, M. L. Rosa, J. Mendling, H. A. Reijers, Fundamentals of Business Process Management, Second Edition, Springer, 2018.

[2] W. M. van der Aalst, Process Mining: Data Science in Action, Second Edition, Springer, 2016.

[3] A. Berti, W. M. P. van der Aalst, D. Zang, M. Lang, An open-source integration of process mining features into the camunda workflow engine: Data extraction and challenges, in: C. D. Ciccio, B. Depaire, J. D. Weerdt, C. D. Francescomarino, J. Munoz-Gama (Eds.), Proceedings of the ICPM Doctoral Consortium and Tool Demonstration Track 2020 co-located with the 2nd International Conference on Process Mining (ICPM 2020), Padua, Italy, October 4-9, 2020, volume 2703 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2020, pp. 23–26. URL: https://ceur-ws.org/Vol-2703/paperTD2.pdf.

[4] W. M. P. van der Aalst, Object-centric process mining: Dealing with divergence and convergence in event data, in: P. C. Ölveczky, G. Salaün (Eds.), Software Engineering and Formal Methods - 17th International Conference, SEFM 2019, Oslo, Norway, September 18-20, 2019, Proceedings, volume 11724 of *Lecture Notes in Computer Science*, Springer, 2019, pp. 3–25. URL: https://doi.org/10.1007/978-3-030-30446-1_1. doi:10.1007/978-3-030-30446-1_1.

[5] A. Burattin, Streaming process mining, in: W. M. P. van der Aalst, J. Carmona (Eds.), Process Mining Handbook, volume 448 of *Lecture Notes in Business Information Processing*, Springer, 2022, pp. 349–372. URL: https://doi.org/10.1007/978-3-031-08848-3_11. doi:10.1007/978-3-031-08848-3_11.

[6] Kafka 3.6 documentation, https://kafka.apache.org/documentation/, 2023. Accessed: 14.01.2024.

[7] A. F. Ghahfarokhi, G. Park, A. Berti, W. M. P. van der Aalst, OCEL: A standard for object-centric event logs, in: L. Bellatreche, M. Dumas, P. Karras, R. Matulevicius, A. Awad, M. Weidlich, M. Ivanovic, O. Hartig (Eds.), New Trends in Database and Information Systems - ADBIS 2021 Short Papers, Doctoral Consortium and Workshops: DOING, SIMPDA, MADEISD, MegaData, CAoNS, Tartu, Estonia, August 24-26, 2021, Proceedings, volume 1450 of *Communications in Computer and Information Science*, Springer, 2021, pp. 169–175. URL: https://doi.org/10.1007/978-3-030-85082-1_16. doi:10.1007/978-3-030-85082-1_16.

[8] M. B. S. Narayana, H. Khalifa, W. M. P. van der Aalst, JXES: JSON support for the XES event log standard, CoRR abs/2009.06363 (2020). URL: https://arxiv.org/abs/2009.06363. arXiv:2009.06363.

[9] G. I. Radchenko, A. B. A. Alaasam, A. Tchernykh, Micro-workflows: Kafka and kepler fusion to support digital twins of industrial processes, in: A. Sill, J. Spillner (Eds.), 2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion, UCC Companion 2018, Zurich, Switzerland, December 17-20, 2018, IEEE, 2018, pp. 83–88. URL: https://doi.org/10.1109/UCC-Companion.2018.00039. doi:10.1109/UCC-COMPANION.2018.00039.

[10] M. A. Mhand, A. Boulmakoul, H. Badir, Scalable and distributed architecture based on apache spark streaming and prom6 for processing roro terminals logs, in: Proceedings of the New Challenges in Data Sciences: Acts of the Second Conference of the Moroccan Classification Society, 2019, pp. 1–4.

[11] A. K. Akanbi, Estemd: A distributed processing framework for environmental monitoring based on apache kafka streaming engine, CoRR abs/2104.01082 (2021). URL: https://arxiv.org/abs/2104.01082. arXiv:2104.01082.

[12] A. F. Nogueira, M. Z. Rela, Monitoring a CI/CD workflow using process mining, SN Comput. Sci. 2 (2021) 448. URL: https://doi.org/10.1007/s42979-021-00830-2. doi:10.1007/S42979-021-00830-2.

[13] L. Malburg, J. Grüger, R. Bergmann, An iot-enriched event log for process mining in smart factories, CoRR abs/2209.02702 (2022). URL: https://doi.org/10.48550/arXiv.2209.02702. doi:10.48550/ARXIV.2209.02702. arXiv:2209.02702.

[14] N. Deehan, Orchestration with camunda and kafka confluent cloud, https://camunda.com/blog/2023/11/orchestration-camunda-kafka/, 2023. Accessed: 14.01.2024.

[15] B. Rücker, Process automation and apache kafka, https://page.camunda.com/wp-process-automation-and-apache-kafka, 2022. Accessed: 14.01.2024.