Scalable and Reliable Complex Event Processing on Event Streams

Bachelor's Thesis / T3300

for the study program
Computer Science

at the
Baden-Wuerttemberg Cooperative State University Stuttgart

by
Dominik Stiller

Submission Date	September 7, 2020
Project Period	12 Weeks
Company	DXC Technology
Corporate Supervisor	Dipl.Ing. Bernd Gloss
University Supervisor	Prof. Dr. Dirk Reichardt
Matriculation Number, Course	4369179, TINF17A

Declaration of Authorship

I hereby declare that the thesis submitted with the title *Scalable and Reliable Complex Event Processing on Event Streams* is my own unaided work. All direct or indirect sources used are acknowledged as references.

Neither this nor a similar work has been presented to an examination committee or published.

Sindelfingen	Septermber 7, 2020	
Place	Date	Dominik Stiller

Confidentiality Clause

This thesis contains confidential data of *DXC Technology*. This work may only be made available to the first and second reviewers and authorized members of the board of examiners. Any publication and duplication of this thesis–even in part–is prohibited.

An inspection of this work by third parties requires the expressed permission of the author, the project supervisor, and *DXC Technology*.

Abstract

Real-time computer vision applications with deep learning-based inference require hardware-specific optimization to meet stringent performance requirements. Frameworks have been developed to generate the optimal low-level implementation for a certain target device based on a high-level input model using machine learning in a process called autotuning. However, current implementations suffer from inherent resource utilization inefficiency and bad scalability which prohibits large-scale use.

In this paper, we develop a load-aware scheduler which enables large-scale autotuning. The scheduler controls multiple, parallel autotuning jobs on shared resources such as CPUs and GPUs by interleaving computations, which minimizes resource idle time and job interference. The scheduler is a key component in our proposed Autotuning as a Service reference architecture to democratize autotuning. Our evaluation shows good results for the resulting inference performance and resource efficiency.

Contents

List of Figures VI

List of Tables VII

List of Source Codes VIII

1 Introduction 1

2 Background 2
- 2.1 Streaming Data . . . 2
- 2.2 Stream Processing . . . 2
- 2.3 Stream Transport . . . 4
- 2.4 Event-Driven . . . 4
- 2.5 Complex Event Processing . . . 5

3 Design 6
- 3.1 Architecture . . . 6

4 Implementation 7
- 4.1 Geoaggregation . . . 7

5 Evaluation 8

6 Conclusion 9

Bibliography 10

List of Figures

List of Tables

List of Source Codes

1 Introduction

problem: Scalability + Stateful Processing (-> required for detection of complex events)

2 Background

batch processing for a couple of years

easy to reason about completeness since it is bounded

latency usually not an issue

2.1 Streaming Data

properties: unbounded, unordered, varying event time skew

stream vs table

concepts of event time and processing time

analysis of data based on when they are observed as opposed to when they occur is usually not sufficient

need to handle event time separately

need to define measure of completeness to maximize correctness

different methods for handling late data: retract old results, separate output, dismiss

tradeoff completeness vs latency

2.2 Stream Processing

short history

stream processing patterns [1, p. 35]:

Time-Agnostic very simple because no reasoning about time, only logic-based on single record, like filtering or inner join

Approximation complicated algorithms like streaming k-means, some with provable error bounds

Windowing chopping up stream into bounded datasets

partition by key

batch can be processed with streaming system

explanation of a true streaming use case [1, p. 386]

[1]

[2]

[3]

2.2.1 Windowing

division of unbounded stream in bounded segments

can be arbitrary, but usually time or count

will focus on time, since count is effectively processing time

fixed, sliding, session, fixed is special case of sliding window

based on processing time: simple and perfect measure of completeness, applicable in many cases where observation time is desired

based on event time: required when event time is desired, requires more buffering than processing time, usually no perfect measure of completeness, therefore based on heuristic

triggers

watermarks as heuristic, show different options

difference between watermark and allowed lateness

bounded out of orderness, ascending...

several aggregations over window

2.3 Stream Transport

message queue (Rabbitmq), often ephemeral

plain socket stream

pubsub

append-only immutable log with persistency

Motivation [2, p. 29 f.]:

- Loose coupling
- Scalability
- Schema flexibility

present Kafka

2.4 Event-Driven

event types and definitions

event type vs event instance

event happens in an instant

complex events are multiple events in correlated according to a pattern (have a duration)

composite event would be more fitting, but complex event is prevailing term

event driven types

event notification

event sourcing

event-carried state transfer

geoevents

2.5 Complex Event Processing

pattern recognition performed on event streams

seit sql:2016 auch iso standard

selection of events to evaluate by window or consumption mode

3 Design

go through design considerations

3.1 Architecture

separate clusters for ingest, streaming, processing and ui

performance considerations:

- large sliding window with short period requires lots of memory
- High allowed lateness increases time until records can be garbage collected
- Accumulation functions only need to store a single value instead of all like in process function

decoupling of ingestion and processing with persistent event log in between has benefits

- handle backpressure without data loss
- decouple ingest and processing -> other processing possible
- replay in case of failure because not ephemeral

processing latency reasons:

reasons for latency: https://flink.apache.org/news/2019/02/25/monitoring-best-practices.html#monitor latency

configuration can tradeoff latency vs throughput

not the same as stream latency caused by waiting for watermark

capacity planning: https://www.ververica.com/blog/how-to-size-your-apache-flink-cluster-general-guidelines

4 Implementation

2 variants

- sampling with regular events

- predict with dead reckoning to decrease event amount, 1d state space of trams is easily predictable

monitoring

4.1 Geoaggregation

Division in cells

late data handling

watermark bounded out of orderness time vs allowed lateness is a tradeoff between latency and recomputation effort

if only interested in latest window results: allowed lateness = window evaluation time

late side output if fine-grained handling required

but often if delayed: delayed much longer than allowed lateness, e.g. if bus is in tunnel instead of just small transmission delay

5 Evaluation

move to confluent with schema registry for production

measure log size with json vs binary protobuf

latency: latency is the delay between the creation of an event and the time at which results based on this event become visible (https://flink.apache.org/news/2019/02/25/monitoring-best-practices.html#monitoring-latency)

6 Conclusion

Bibliography

[1] T. Akidau, S. Chernyak, and R. Lax, *Streaming systems: The what, where, when, and how of large-scale data processing*, First edition. Sebastopol CA: O'Reilly, 2018, ISBN: 1491983876.

[2] M. Kleppmann, *Making Sense of Stream Processing*, 1st edition. O'Reilly Media, Inc, 2016, ISBN: 9781491937280. [Online]. Available: `https://www.oreilly.com/data/free/files/stream-processing.pdf`.

[3] Hedtstück, *Complex Event Processing: Verarbeitung von Ereignismustern in Datenströmen*, ser. eXamen.press. Berlin: Springer Berlin Heidelberg, 2017, ISBN: 9783662534502. DOI: `10.1007/978-3-662-53451-9`. [Online]. Available: `https://link.springer.com/content/pdf/10.1007%2F978-3-662-53451-9.pdf`.