TODO: globaleventprognosis

Phillip Ginter
Informatik (IN)
Hochschule Furtwangen
78120 Furtwangen, Deutschland
phillip.ginter@hs-furtwangen.de

Daniel Schönle
Informatik (IN)
Hochschule Furtwangen
78120 Furtwangen, Deutschland
daniel.schoenle@hs-furtwangen.de

***Zusammenfassung*—aaa**
***Index Terms*—wikipedia, prognosis, global event**

I. Einleitung

Wikipedia ist eine freie Online-Enzyklopädie, mit dem Ziel „eine frei lizenzierte und hochwertige Enzyklopädie zu schaffen und damit lexikalisches Wissen zu verbreiten“ [1]. Sie umfasst 74 Millionen Artikel, die von eine Community von 137.571 Nutzer erstellt und geprüft wurden. Seit 2001 wurden die Artikel 877.073.914 mal editiert, dies entspricht etwa 4.000.000 Edits pro Monat. [2]

Aufgrund des kollaborativen Erstellungsprozesses können die Nutzer autonom und dynamisch Artikel ändern und erstellen. [3] Die Gründe weswegen die Nutzer aktiv werden variieren, einer davon ist die Dokumentation globaler Vorfälle. Dies können unter Anderen politische Veränderungen, Naturkatastrophen, sportliche Ereignisse oder Entwicklungen in der Vita prominenter Personen sein. Eine wichtige gemeinsame Eigenschaft ist eine hohe Relevanz für einen großen Nutzerkreis, der sich in vielen Aktivitäten, sogenannten Edits, niederschlägt. Interessant ist dabei auch welche Untermenge sich aus den aktiven Nutzer dafür bildet. Offensichtliche gemeinsame Eigenschaften dieser Nutzermengen sind geographische Verortung, Sprache, Kompetenzen und Interessen. [4]

Eine Untersuchung der Aktivitäten der Nutzer über Zeiträume hinweg und mittels statistisch gestützter Analyse kann weitere Zusammenhänge herstellen. Dafür gibt es zwei Herangehensweisen; eine Analyse gespeicherter Aktivitäten, sowie eine Beobachtung in Echtzeit. Georgescu et. al. [4] haben in 'Extracting Event-Related Information from Article Updates in Wikipedia' ersteres durchgeführt, worauf im Abschnitt 'Verwandte Arbeiten' eingegangen wird.

Für eine Analyse in Echtzeit, kann die Folge von Nutzer-Aktivitäten als Stream implementiert werden und die Aktivitäten an sich als Events. Wikipedia bietet eine entsprechende Schnittstelle an, den Edit-Events-Stream. In einer Event-Driven-Architecture können durch Filtern, Aggregieren und Gruppieren der Events Muster erkannt werdern und die Zusammenhänge repräsentieren. Die gewonnenen Zusammenhänge können wiederum als weitere Events in den Verarbeiteten Stream eingefügt werden, welche wiederum selbst zu Mustern zusammengesetzt werden können. Eine stufenweise Verdichtung der Information, mit jeder weiteren Erkennung von Mustern, ist die Folge. Das Resultat sind Zusammenhänge die denen eines Use Case entsprechen.

- Wir betrachten nur die Metadaten (Zeitstempel, Autor, ...) und nicht den Inhalt der Änderung Änderung (z. B. textuelle Änderung).
- ...

Wie sieht so ein Burst of Wikipedia-Edits aus 1?

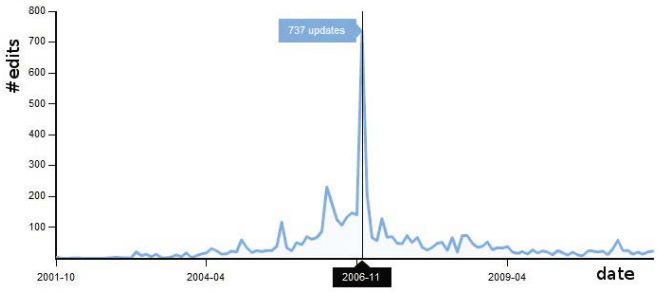

Abbildung 1: Donald Rumsfeld's Rücktritt führte zu einem Burst an Autoren, die einen Wikipedia-Edit vornahmen [4].

II. Verwandte Arbeiten

A. Event-Driven Detection

Weiler et. al. [5] evaluieren fünf State of the Art Techniken zur Erkennung von noch unbekannten Ereignissen.Niagarino wird als Implementation verwendet, eine Plattform die Apache Storm ähnelt. Die Techniken wurden als Anfragen umgesetzt, wie in 2 dargestellt. Twitter-Events dienen als Datenbasis, die einem Pre-Processing gefiltert werden. Dabei werden Retweets entfernt und der Inhalt der restlichen Tweets gesäubert, es verbleiben bibliographisch erkannte englische Wörter.

- TopN
  Jedem Wort wird ein Wert zugewiesen, der auf dem Inverse-Document-Frequency des Zeitfenster beruht. Nur die N-Wichtigtsten Wörter verbleiben als Event.
- Latent Dirichlet Allocation (LDA)
  Ist ein hierarchisches Bayes-Modell, das die Variation des Vokabulars in einer Gruppe von Dokumenten bewertet. Für jedes Zeitfenster extrahiert LDA vermutete Ereignisse, die durch Ausdrücke beschrieben werden.

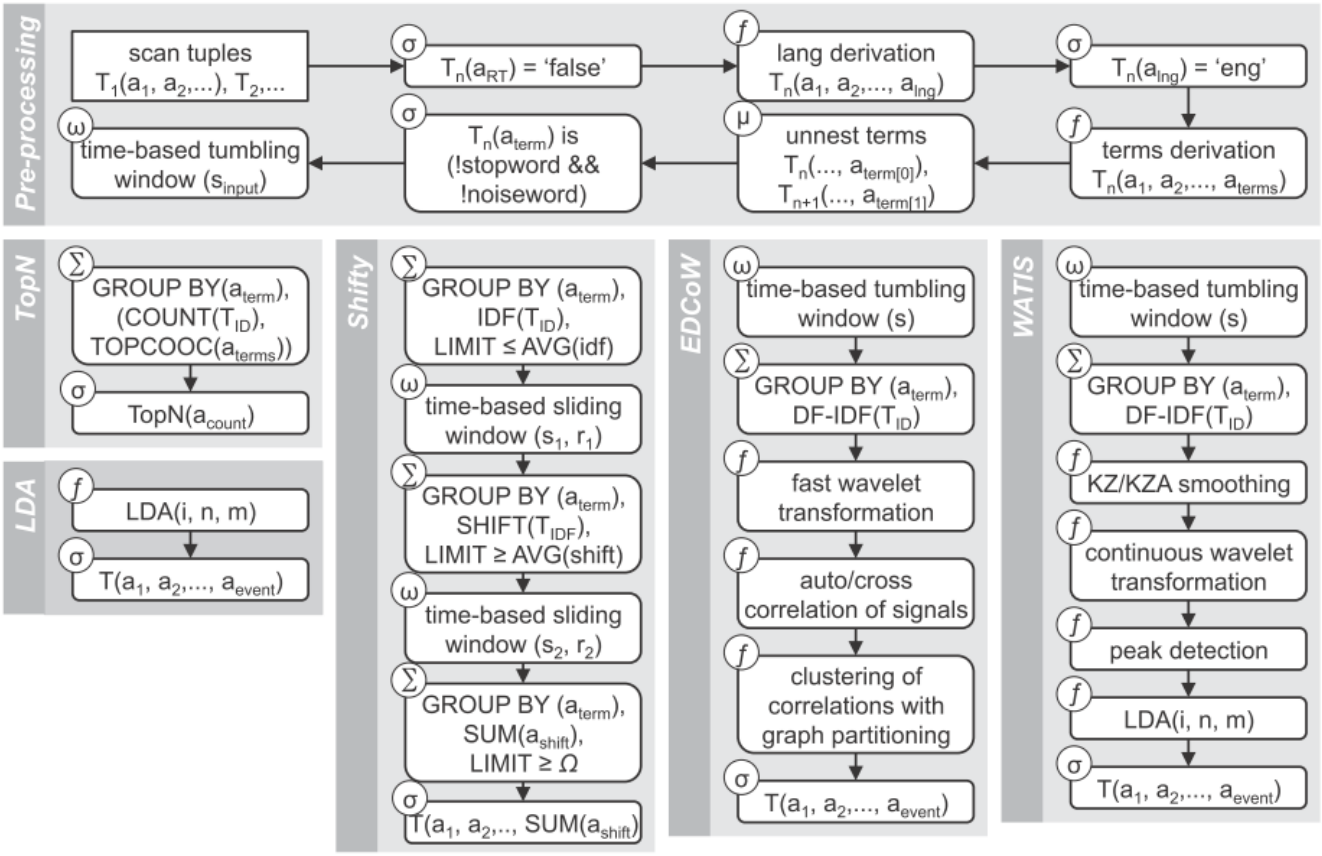

Abbildung 2: Niagarino Anfragen der fünf Erkennungs-Methoden

- Shifty
  Dabei werden die Veränderungen von TopN bewertet die sich zwischen zwei Zeitfenstern ereignen.
- Event Detection with Clustering of Wavelet-based Signals (EDCoW)
  Der Stream wird Batches aufgeteilt, die dann per Wavelet-Analyse zu einem weiteren Event zusammengefasst werden, in dem Wörter mit erkannten Bursts stehen. Davon werden dann unwichtige Wörter entfernt. Durch Clustering erhält das Ergbnis Burst-Events die mit zwei Termen beschreiben werden.
- The Wavelet Analysis Topic Inference Summarization (WATIS)
  Eine Weiterentwicklung des (EDCoW), bei dem zu jedem Burst fünf Terme geliefert werden.

B. Burst Detection

Da sich die Erkennung globaler Events mittels Esper als zumindest unvollständig erwies, wurden Methoden untersucht um das Verfahren zu Ergänzen oder zu Ersetzen.

Globale Events haben unter Anderem auch die Eigenschaft eine erhöhte Aktivität im Edit-Event-Stream zu verursachen. Erhöhte Aktivität (Bursts) ist ein Phänomen das bereits seit Jahrzehnten untersucht wird, daher existieren eine Reihe von Theorien und abgeleiteten Algorithmen. Dabei ist die zeitnahe Erstellung von Ergebnissen wichtig, weswegen auf eine wenig aufwändige Verarbeitung hin optimiert wird. Bursts besitzen eine Reihe an Attributen, die abhänig vom Use Case defininert werden. Dazu zählt der Zeitraum der jeweils auf das Auftreten eines Bursts untersucht wird, die Intensistät und die Verteilung der Intensistät auf den betrachteten Zeitraum.

Generell kann eine Datentstrom auf zwei Methoden betrachtet werden. Beim Point Monitoring wird nur das aktuelle Ereigniss betrachetet. Liegt der Wert eines Attributs in einen vordefinierten Bereich oder überschreitet es einen Schwellwert wird ein Burst erkannt. Beim Aggregate Monitoring werden über einen Zeitraum (Windows) hinweg Ereignisse aggregiert. Dabei werden drei Zeitfenstertypen unterschieden. Beim Landmark Window wird der Zeitraum durch einmal fest gelegte Zeitpunkte definiert. Das Sliding Window hingegen bewegt sich durch die Zeit. Dabei wird dessen Größe sowie das Interval mit dem es sich bewegt festgelegt. Die Paramter können als Zeitangaben oder als Zählangaben erfolgen. Somit kann ein Fenster entweder nach einer gewissen Zeit neu erstellt werden oder nach einer bestimmten Anzahl an Ereignissen. Beim Damped Window werden zudem jedem Ereignis Gewichtungen erteilt, je länger

ein Ereignis zurückliegt desto weniger Gewicht bekommt es mit Fortschreiten der Zeit. [6]

Eine spezielles Modell entwickeln Zhu et. al. in 'Efficient Elastic Burst Detection in Data Streams' [6] mit dem Elastic Window, bei dem die Größe der Zeitfenster variiert. Somit können die Bursts dynamischer durch deren Characterisik und unabhäniger von ihrer Dauer erkannt werden, die oftmals nicht im Vorfeld festgelegt werden kann. Dafür wird ein Wavelet-Tree vom Stream erzeugt, bei dem die Wavelet-Kooefizienten den Aggregaten der Fenster entsprechen. Wie in Abbildung **??** zu sehen ist, wird zu Beginn der Stream in Fenster unterteilt, die disjunkt nebeneinander liegen. Je Fenster wird ein Aggregat erstellt, in der Regel werden dazu die Ergeinisse eines bestimmten Typs oder mit gleichen Attributen gezählt. Diese Aggreagte bilden nun selbst wiederum einen Stream. Das Verfahren wird wiederholt, vorrausgesetzt es befinden sich entsprechende Ereignisse bzw. Aggregierte Ereignisse im Stream. Mit jedem Schritt wird der betrachtete Zeitabschnitt größer und man bewegt in das nächst höhere Level Richtung Wurzel des Baums. Zu jedem Level wird ein Schwellwert angegeben, bei dessen Überscheitung ein Burst ausgelöst wird.

Yuan et. al. entwickeln in 'Online Burst Detection Over High Speed Short Text Streams' [7] ein Verfahren das auf einem Wavelet-Tree basiert. Dieser kann auch als Pyarmide betratchtet werden, mit den Levels die dem Aggregierungsgrad entsprechen. Da die Burst-Detection sich auf den Schwellwert des jeweiligen Lebels stützt, muss dieser zu beginnder Analyse feststehen. Das kann eine Herrausforderung sein, denn damit muss auch die Menge an Ereignissen im Stream eingeschätzt werden können. Um diese Problem zu lösen werden keine Absoluten Angaben der Aggregatewerte verwendet, wie zum Beispiel deren Anzahl. Stattdessen wird angegeben, wieviele relevante Events aggregiert wurden im Verhältnis zu allen aufgetretenen Events im betroffenen Zeitraum. Abbildung 3. Eine weitere Komprimierung wird durch die Slope Pyarmid, wie in Abbildung 5 dargestellt, erreicht. Dabei wird der Fokus auf die Veränderung von Fenster zu Fenster gelegt. Dazu wird Verhältniswert aus der Ratio Pyamid herangezogen; der Wert des vorhegehenden Fensters wird mit dem des aktuellen Fenster dividiert. [7]

Besondere Anforderungen stellt die Realtime-Burst Detection in Verbindung mit der gleichzeitigen Betrachtung mehrerer Fenstergrößen. Übliche Burst-Detektionsverfahren sind für eine Echtzeiterkennung nicht schnell genug, daher wurde von Ebina et. al. [8] Burst-Erkennungsmethode weiter entwickelt. Das Ziel ist dabei die Berechnungszeit zu reduzieren, indem redundante Datenaktualisierungen wie bei Yuan et. al. vermieden werden. Dazu werden die letzen Zellen der Slope Pyarmid auf effizientere Weise berechnet. [8]

Je nach Use Case sind auch aufwändigere Analysen wie in 'Event Detection with Burst Information Networks' [9] notwendig.

C. Social Media Analyse

Eine ganze Reihe von Arbeiten beschäftigt sich mit der Analyse der Aktivitäten der Wikipedia-Nutzer.

- Extracting Event-Related Information from Article Updates in Wikipedia [4]
- Ongoing events in Wikipedia: a cross-lingual case study [10]
- How much is Wikipedia Lagging Behind News? [11]
- Event analysis in social multimedia: a survey [12]
- A cloud-enabled automatic disaster analysis system of multi-sourced data streams: An example synthesizing social media, remote sensing and Wikipedia data [13]

III. Streaming Data

Das Ziel unserer Aufgabenstellung ist die Verarbeitung von Streaming-Daten in Echtzeit. Die Streaming Data-Architektur von Psaltis [14] ist für diese Art von Problem konzipiert und bildet die Grundlage für unsere Architektur. Nach der kurzen Vorstellung der Streaming Data-Architektur von Psaltis, zeigen wir unsere eigene konkrete Umsetzung und vergleichen eingesetzten Technologien mit Alternativen.

A. Streaming Data-Architektur nach Psaltis

In Abbildung 6 sind die Komponenten der Streaming Data-Architektur, wie Psaltis [14] sie entwickelt hat, abgebildet. Der Collection Tier ist der Einstiegspunkt, der Daten in das System bringt. Unabhängig von dem verwendeten Protokoll, werden die Daten mithilfe eines dieser Patterns übertragen [14]:

- Request/response pattern
- Publish/subscribe pattern
- One-way pattern
- Request/acknowledge pattern
- Stream pattern

Bei der Datenquelle kann es sich sowohl um von Hardware und auch von Software generierte Events handeln. Beispiele hierfür sind Temperatur, Lautstärke oder auch Browser-Clicks.

Um die Daten vom Collection Tier auf den Rest der Pipeline zu verteilen wird ein Messaging Queueing Tier implementiert. Das Message Queueing Model ist ein Modell zur Interprozesskommunikation. Anwendungen schicken Nachrichten an eine Nachrichtenschlange, von der andere Anwendungen diese abholen können [15]. Dadurch werden die eingesetzten Systeme voneinander entkoppelt und die Kommunikation findet nicht durch direkte Aufrufe, sondern über die Queue statt [14]. Im vorliegenden Fall wird der Collection Tier vom Rest der Pipeline entkoppelt.

Im Analysis Tier werden auf die Streams des Message Queueing Tiers kontinuierlich Queries angewandt um Muster in den Daten zu erkennen. Hier können Event Stream Processing-Systeme (ESP) wie z. B. Apache Flink, Apache Spark oder Apache Storm oder auch Complex Event Processing-Systeme (CEP) wie z. B. Esper oder Apache Samza eingesetzt werden. [14] In ESP wird die Verarbeitungslogik imperativ umgesetzt, wohingegen in CEP einen deklarativen Ansatz verfolgt. Die Event Processing Languages (EPL) sind

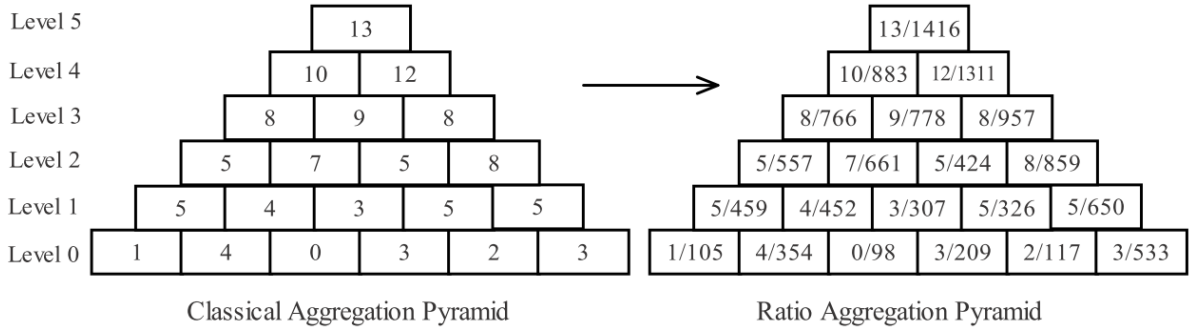

Abbildung 3: Von der klassischen 'Aggregataion Pyramid' zur 'Ratio Aggreation Pyramid' [7]

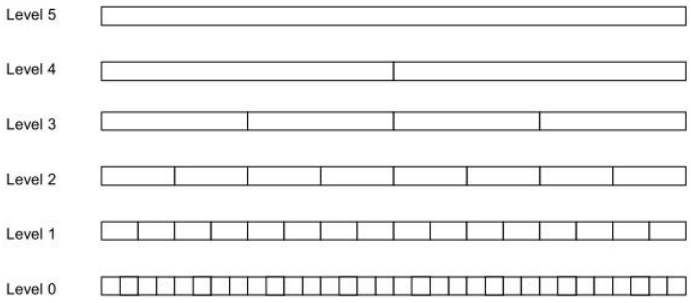

Abbildung 4: Wavelet Tree [6]

Sprachen, die über Sequenz-, Konjunktions-, Disjunktions- und Negationsoperatoren verfügen und für CEP entwickelt wurden [16]. Als Basis der Operatoren werden Windows eingesetzt. Ist die Analyse vorbei und die Ergebnisse stehen fest, können diese verworfen werden, zurück in die Streaming-Plattform gespeichert werden, für eine Echtzeitnutzung oder die Stapelverarbeitung gespeichert werden [14]. Wie der Abbildung 6 zu entnehmen ist, schlägt Psaltis hierfür eine In-Memory-Datenbank (IMDB), einen Data Access Tier und ein Long Term Storage vor.

B. Unsere Streaming Data-Architektur

In diesem Kapitel geht es um die Frage, welche Technologie wir in dem jeweiligen Tier einsetzen und wieso wir uns dafür entschieden haben. Technische Details, die die Implementierung betreffen, erläutern wir im nächsten Kapitel. TODO: Quellen zu den einzelnen Punkten/Systemen

- *Collection Tier.* Als Datenquelle haben wir Daten von Wikipedia. Konkret handelt es sich um die RecentChanges, also die aktuellen Änderungen an Wikipedia-Einträgen. Details hierzu sind im nächsten Kapitel beschrieben. Wir entschieden uns für Wikipedia als Datenquelle, weil es eine offene Plattform ist, die einen freien Zugang zu allen Daten gibt.
- *Messaging Queuing Tier.* Wir nutzen Kafka als Messaging-System, weil es skalierbar ist, einen hohen Datendurchsatz ermöglicht, Real-Time-Messaging unterstützt, eine einfache Client-Anbindung ermöglicht und Events standardmäßig persistiert. Der letzte Punkt ist besonders relevant, da für die Entwicklung der Regeln eine statische Datenmenge einfacher zu handhaben ist.
- *Analysis Tier.* Die Analyse der Wikipedia-EditEvents sollte deklarativ erfolgen. Dadurch erhofften wir uns in kurzer Zeit eine Vielzahl an Regeln zu testen und so ein gutes Verständnis für die Daten zu erhalten. Aufgrund dieser Vorgabe entschieden wir uns für Esper. Es bietet eine CEP-Implementierung, die nach einem regelbasierten Ansatz (somit deklarativ) arbeitet. Ein weiterer Punkt, der für Esper sprach, war zum einen die Esper Processing Language und die Implementierung der Anwendung in Java.
- *In-Memory Data Store, Long Term Storage und Data Access Tier.* Für diese drei Komponenten haben wir selbst keine Implementierung vorgesehen. Im Ausblick geben wir hierzu Ideen zu möglichen Erweiterungen.

IV. Prototyp

Zur Lösung der Aufgabenstellung haben wir einen lauffähigen Prototypen entwickelt, der die Machbarkeit demonstriert. Dafür haben wir die im vorhergehenden Kapitel genannten Technologien eingesetzt. Die Details zu den jeweils entstandenen Anwendungen stellen wir in diesem Kapitel vor.

Abbildung 7 zeigt die Teilsysteme und deren Interaktion untereinander.

A. Collection Tier: Wikipedia

Wie zuvor beschrieben, stützt sich unser System auf Daten von Wikipedia. Wir nutzen den RecentChanges-Stream, der Events zu neu erstellten, aktualisierten und gelöschten Wikipedia-Seiten enthält[1]. Der Quellcode 1 zeigt einen Ausschnitt eines solchen Events. Es sind nur die Attribut abgebildet, die in einem der nachfolgenden Teilsysteme relevant sind.

```
{
    "bot": false,
    "comment": "/* Politischer Werdegang
       */",
    "id": 269108829,
    "meta": {
```

[1] https://www.mediawiki.org/wiki/Manual:RCFeed

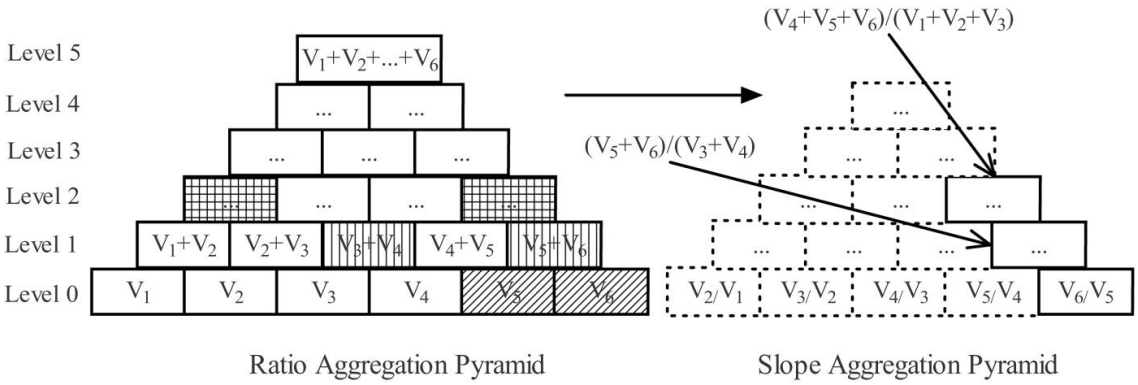

Abbildung 5: Berechnungsbeispiel zur 'Slope Pyramid' [7]

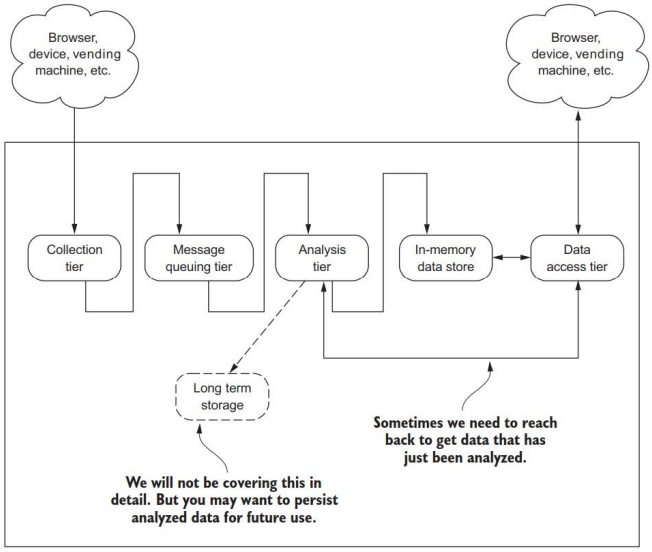

Abbildung 6: Streaming Data-Architektur [14]

```
        "uri": "https://de.wikipedia.org/
            wiki/Ingeborg_H%C3%A4ckel",
        "partition": 0,
        "offset": 1329388977
    },
    "timestamp": 1547910734,
    "title": "Ingeborg H\"ackel",
    "user": "Eszet2000",
    "wiki": "dewiki"
}
```

Quellcode 1: Wikipedia RecentChange-Event

Die beiden Parameter `offset` und `uri`, innerhalb des `meta`-Objektes, stammen aus einem Kafka-System und dienen der Wiederaufnahme eines abgebrochenen Streams. Wikipedia setzt intern Apache Kafka als Messaging-System ein. Nach außen nutzt Wikipedia EventStreams. Das ist ein Webservice der kontinuierliche Datenströme mit strukturierten Daten über HTTP sendet[2]. Die Basis-Technologie dessen, ist das Server-Sent Event (SSE) Protokoll. Der RecentChanges-Stream ist ein solcher Stream und kann über eine Client-Bibliothek konsumiert werden.

Wie im vorherigen Kapitel beschrieben, werden Daten vom Collection Tier mithilfe eines der aufgelisteten Pattern übertragen. SSE funktioniert nach dem One-way pattern, indem eine HTTP-Verbindung aufgemacht wird um Nachrichten zu empfangen [17].

B. Implementierungsdetails zum Messaging Queuing Tier: Kafka

Im Messaging Queuing Tier setzen wir Apache Kafka in der Version 2.1 ein, um die im Collection Tier beschriebenen Wikipedia-Events in unser eigenes Messaging-System zu überführen. Hierfür haben wir eine Java-Anwendung entwickelt, in der die folgenden Schritte nacheinander ausgeführt werden:

1) *Kafka Initialisierung.* Den Host des Bootstrap-Servers setzen um eine Verbindung zu erzeugen. Als Key- und Value-Serialisierer setzen wir jeweils den `StringSerializer` von Kafka ein. Das heißt, die Events werden als JSON-String in das Topic `wikiEdit` eingespeist. Zum Senden von Events erzeugen wir ein Producer-Objekt mit dem passenden Typ `Producer<String, String>`. An dieser Stelle war die Überlegung, anstelle eines String-Serialisierers für den Wert des Producers, einen eigenen Serialisierer für die WikipediaEditEvent-Klasse einzusetzen. Wir entschieden uns aber für den String-Serialisierer, da Wikipedia die Events auch als JSON-String überträgt und wir in Kafka selbst keine weiteren Operationen an den Daten vornehmen. Eine mögliche Operation könnte ein Filter sein. Aber das einzige Ziel von Kafka soll

[2]https://wikitech.wikimedia.org/wiki/EventStreams

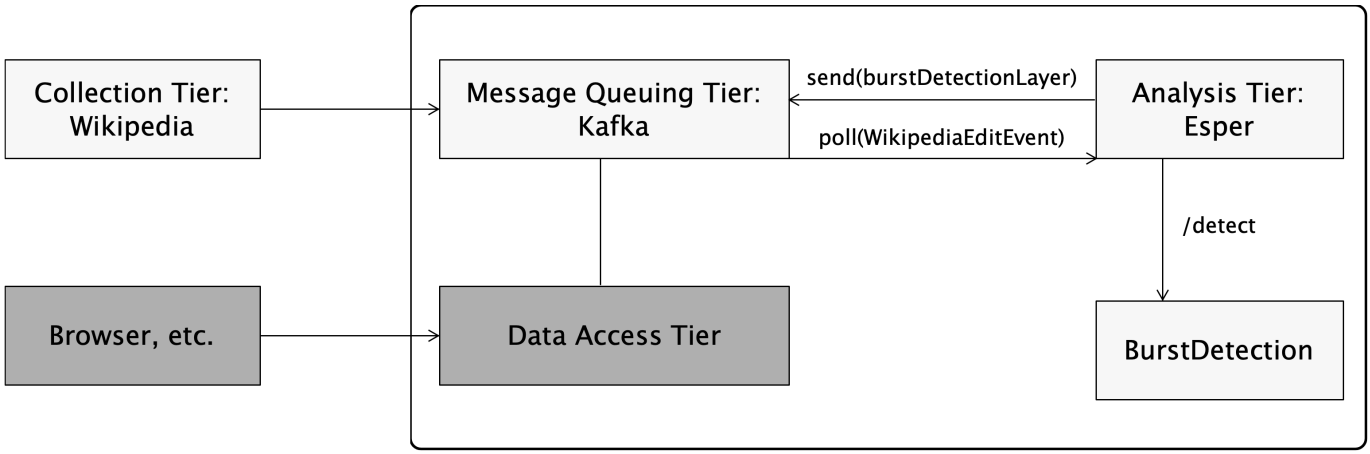

Abbildung 7: Architekturübersicht

die Persistierung und Weitergabe von Daten sein und dafür ist keine Serialisierung in einem bestimmten Typen notwendig. Außerdem verschiebt es Komplexität aus der Kafka-Anwendung in die Consumer-Anwendungen.

2) *Erzeugung eines EventHandlers.* Für das Empfangen von EventSource-Nachrichten nutzen wir die Java-Bibliothek *okhttp-eventsource*[3]. Zur Verarbeitung der Events `onOpen`, `onClose`, `onMessage`, `onComment` und `onError` muss das Interface `EventHandler` von *okhttp-eventsource* implementiert werden.
3) *Erzeugung und Starten einer EventSource.* Mit der Stream-URI der Wikipedia-EventSource (für die RecentChanges ist das: https://stream.wikimedia.org/v2/stream/recentchange) und des implementierten EventHandler-Interfaces kann ein `EventSource`-Objekt erzeugt werden. Das Objekt dient dem Starten und Beenden eines EventSource-Streams. Die Daten werden dann, wie zuvor beschrieben, durch das SSE-Protokoll von Wikipedia an die Anwendung gesendet.
4) *Beim Eintreffen eines Events: Senden einer Nachricht in ein Kafka-Topic.* Tritt ein Wikipedia-Event auf, wird die `onMessage`-Methode des implementierten EventHandlers-Interface aufgerufen. Der zweite Parameter enthält die Daten des aufgetretenen Events. Der Zugriff auf die als JSON-String codierte Nachricht erfolgt über die `getDate()`-Methode. Diese Daten sendet die Anwendung, über den zuvor erzeugten Producer, ohne eine weitere Verarbeitung in das Kafka-Topic `wikiEdit`.

Die Konfiguration der Kafka-Topics ist sehr einfach gehalten, da es sich bei der Anwendung nur um einen Prototypen handelt. Wir haben ein Topic mit dem Namen `wikiEdit`. Da die Anwendung auf nur einem Server läuft, setzen wir auch nur eine Partition ein und haben keine Replikation. Eine Skalierung der Kafka-Anwendung auf mehrere parallelarbeitende Server ist jedoch nicht ausgeschlossen für eine Produktivanwendung.

C. Implementierungsdetails zum Analysis Tier: Esper

In unserer Esper-Anwendung, die das Hauptsystem des Analysis Tier ist, nutzen wir Esper in Version 7.1 als Complex Event Processing-Werkzeug. Zur Verarbeitung der Wikipedia-Events haben wir zwei Lösungen umgesetzt, da die erste Lösung nicht zum Erreichen der Ziele führte. Für ein besseres Verständnis geben wir einen kurzen Überblick über den gemeinsamen Aufbau beider Lösungen. Danach beschreiben wir die Details und Unterschiede der jeweiligen Lösungen und analysieren diese hinsichtlich der Zielerfüllung. In unserer Esper-Anwendung sind wir im Allgemeinen wie folgt vorgegangen:

1) *Esper Initialisierung.* Die Initialisierung von Esper besteht aus der Erzeugung einer `Configuration`, der Erstellung und dem Starten von EPL-Statements, sowie dem Erzeugen von Listener-Klassen.
2) *Kafka initialisieren und starten.* Um die Daten aus dem Messaging Queuing Tier zu bekommen haben wir einen `KafkaConsumer<String, String>` implementiert. Wir pollen in einer Endlosschleife alle 10 Millisekunden die Daten vom Kafka-Topic `wikiEdit`. Bei den empfangenen Daten handelt es sich um einen String, der JSON enthält. Mithilfe der Gson-Bibliothek[4] konvertieren wir die JSON-Strings in Java-Objekte. Die daraus resultierenden Java-Objekte senden wir wiederum in das Esper-System, damit darauf die EPL-Statements angewandt werden können.
3) *Aktion ausführen, sobald das Muster erfüllt ist.* Ist das Muster erfüllt, wird die `update`-Methode der Listener-Klasse aufgerufen. Es werden die alten und neuen Events übergeben. Daraus kann eine Aktion erfolgen, z. B. dass erzeugen eines neuen komplexen Events.

1) Lösung 1: In unserer ersten Lösung war es unser Ziel, relevante Events durch die Modellierung geeigneter Ereignistypen zu detektieren. Ein relevantes Event besteht für

[3]https://github.com/launchdarkly/okhttp-eventsource

[4]https://github.com/google/gson

uns aus einer definierten *Anzahl an Benutzer* die in einem definierten *Zeitraum* eine definierte *Menge an Bearbeitungen* an einer Wikipedia-Seite vornehmen. Zur weiteren Konkretisierung zeigt Quellcode 2 das erste EPL-Statement. Hierbei werden 5 aufeinanderfolgende `WikipediaEditEvent`, die kein Bot sind, von unterschiedlichen Benutzern stammen und die gleiche URI aus der deutschsprachigen Wikipedia haben. Desweiteren wird mit der `checkWithin`-Methode geprüft, ob die Events `w2` bis `w5` jeweils den definierten Zeitabstand von 600 Sekunden nicht überschreiten. In Esper werden diese Methoden Single-Row-Functions genannt. Diese müssen in der `Configuration` angegeben werden und es handelt sich dabei um eine einfache Java-Methode, die einen `Boolean` als Rückgabetyp hat.

```
insert into WikipediaEditEventFive
select w1, w2, w3, w4, w5
from pattern[
    every
    w1=WikipediaEditEvent(bot = false,
       wiki='dewiki') ->
    ...
    w5=WikipediaEditEvent(w5.meta.uri =
       w1.meta.uri, bot = false, user <>
       w1.user, user <> w2.user, user <>
       w3.user, user <> w4.user,
       checkWithin(w1, w5, 600))
]
```

Quellcode 2: Lösung 1: EPL-Statement 1

Der Zugriff auf die Attribute der `WikipediaEditEvent`-Klasse erfolgt über Getter und Setter, die im Klassendiagramm der Abbildung 8 zur besseren Lesbarkeit nicht angegeben wurden. Ist das Muster aus dem Quellcode 2 erfüllt, wird ein neues Event des Typs `WikipediaEditEventFive` eingefügt. Bei der Erzeugung werden dem Konstruktor die selektierten Objekte `w1` bis `w5` übergeben.

Das zweite EPL-Statement ist von der Funktionsweise her analog zum ersten. Das Muster besteht jedoch nicht aus 5 aufeinanderfolgenden `WikipediaEditEvent`, sondern aus 4 aufeinanderfolgenden `WikipediaEditEventFive` die in einem Zeitraum von 20 Minuten aufgetreten sind.

Die beiden Zahlen, 4 und 5 sind durch Ausprobieren ermittelt worden. Das zeigt sehr deutlich, dass das weder eine effiziente noch eine sehr erfolgreiche Lösung für unsere Aufgabe ist. Das Problem liegt an den sehr unterschiedlichen Menge an Bearbeitungen. Denn es gibt Wikipedia-Seiten, die dauerhaft einer großen Menge an Aktualisierungen unterworfen sind und wiederum andere Wikipedia-Seiten, die nur sehr sporadisch aktualisiert werden. Aus diesen Gründen ist es uns mit dieser Herangehensweise nicht gelungen eine Vereinheitlichung, für relevante Events, über alle Wikipedia-Seiten hinweg zu finden.

2) Lösung 2: Die Probleme der Lösung 1 führten dazu, dass wir einen Ansatz verfolgten, bei dem die Betrachtung einzelner Seiten mehr im Fokus steht. In Lösung 2 sind wir

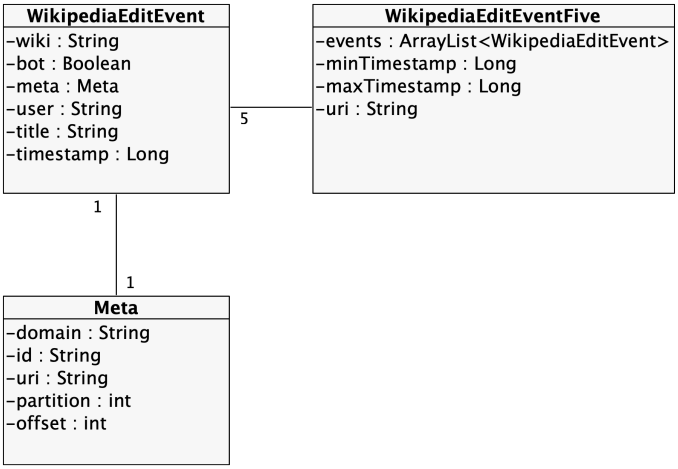

Abbildung 8: Klassendiagramm der zwei Ereignistypen WikipediaEditEvent, WikipediaEditEventFive und der Helferklasse Meta

ein Stück weg vom Website Activity Tracking, hin zur Anomalieerkennung gegangen. Das zeigt sich auch in der Architektur aus Abbildung 7, in der neben der Esper-Anwendung auch ein Aufruf eines Burst-Detection-Algorithmus abgebildet ist.

Konkret sieht die Implementierung so aus, dass wir 100 Events einer URI aggregieren und darauf wenden wir den Burst-Detection-Algorithmus an. Das ermöglicht es uns Anomalien – ein starker Anstieg der Anzahl der Aktualisierungen – einzelner URIs zu betrachten. Der Quellcode 3 zeigt das EPL-Statement hierzu. Es wird eine URI-Partitionierung mithilfe eines Kontexts erstellt. Damit wird das Muster zur Aggregation der 100 Events je URI, mit einem Batch-Window, umgesetzt.

```
create context GroupByUri partition by
   meta.uri from WikipediaEditEvent

context GroupByUri
select *
from WikipediaEditEvent(bot=false , wiki
   ='dewiki').win:length_batch(100)
```

Quellcode 3: Lösung 2: EPL-Statement 2

Wird das Muster erkannt, werden die Timestamps der 100 selektierten Events an die REST-Schnittstelle des BurstDetection-Service gesendet. Dieser wendet einen Burst-Detection-Algorithmus auf die Daten an und gibt ein Array mit "Ebenen"zurück. Die Ebene 0 entspricht hierbei dem gesamten Datensatz von 100 Events. Je höher die Ebene, desto stärker und klarer abgegrenzt ist die Anomalie bzw. der Burst. Für uns hat sich gezeigt, dass Events aber der Ebene 2 einem relevanten Ereignis sehr nahe kommen.

Betrachtet man die 100 Events der Wikipedia-Seite der Handball-Weltmeisterschaft der Männer 2019[5] in dem Zeit-

[5]https://de.wikipedia.org/wiki/Handball-Weltmeisterschaft_der_M%C3%A4nner_2019

raum vom 25. Juni 2018 12:44 Uhr bis zum 12. Januar 2019 17:06 Uhr, ergibt sich folgende Ausgabe:

```
[[0, 1529923440000, 1547309160000], [1.0,
    1546853700000, 1547309160000], [2.0,
   1547136540000, 1547243280000]]
```

Quellcode 4: Ebene 0 1 und 2 der Burst-Detectionen

Die Ebene 0 entspricht hier allen 100 Events, die Ebene 1 geht vom 07. Januar 2019 10:35 Uhr bis zum 12. Januar 2019 17:06 Uhr und die Ebene 2 geht vom 10. Januar 2019 17:09 Uhr bis zum 11. Januar 2019 22:48 Uhr. Je höher die Ebene, desto kleiner und auch relevanter wird ein Bereich. Relevant heißt, dass in diesem Bereich mehr Events als sonst aufgetreten sind – ein Burst. Das Beispiel zeigt also, dass wir durch die Lösung 2 für einzelne Wikipedia-Seiten relevante Ereignisse ableiten können.

D. BurstDetection-Implementierung

Die Burst-Detection ist in Python geschrieben. Es handelt sich hierbei um ein REST-Interface mit einem Endpunkt, der beim Aufruf lediglich als Wrapper für den Aufruf der Burst-Detection fungiert. Die Burst-Detection wird durch ein zusätzliches Python-Package eingefügt und ist auch austauschbar.

V. Anwendungsfälle

A. Erkennung globler Events

Die Erkennung global relevanter Events wurde bereits in **??** beschreiben.

Der Fokus liegt dabei auf der Verarbeitung der Events in Echtzeit, dem Erkennen von Ereignismustern im Wikipedia-Edit-Eventsstrom und dem generieren von komplexen Events. Eine komplexe Datenanalyse ist nicht vorgesehen, statt dessen wird durch eine einfache Mustererkennung genutzt. Dabei werden Events innerhalb von Zeitfenstern aggregiert und in Beziehung zueinander gesetzt um Rückschlüsse auf 'Global Events' zu ziehen.

Die Zielsetzung, den Fokus auf das Erkennen von Ereignismustern in Wikipedia-Edit-Eventsstrom zu setzen und dabei auf komplexe Datenanalyse zu verzichten, schränkt die Anzahl erkanner 'Global Events' ein.

In der Realtime-Analyse kann ein Hinzuziehen externer Informationen nicht erfolgen. Deswegen sind Abfragen externer Quellen, eine Textanalyse der Änderungen auf die in den Edits verweisen wird, eine Verortung der Seite in der Wikipedia-Hierarchie und das Parsen externer Web-Seiten zum Zweck der weiteren Informationsgewinngung nicht möglich.

B. Entworfene Anwendungsfälle

Die Mustererkennung im Wikipedia-Edit-Stream kann für weitere Anaylsen verwendet werden. Exemplarisch wurden weitere Anwendungsfälle entwickelt, die sich im Umfeld der Wikipedia-Nutzer umsetzen lassen. Die vorgestellen Use Cases leiten sich aus Problemen kollaborativer Texterstellung ab,

- Edit-Wars Detection
  Bei Edit-Wars stimmen die Vorstellungen über den Inhalt des Artikels der Autoren nicht überein. Die Folge sind wiederholtes gegenseitiges Revidieren der Änderungen, häufig in hoher Frequenz. [18]
  Daraus entstehen folgende Use Cases: Erkennen von aktiven Edit-Wars. Erkennen von Situationen die zu Edit-Wars führen. Identifizieren von Nutzern die im indirekten Zusammenhang mit Edit-Wars stehen.
- Fraud Detection
  Unter Vandalismus wird im Kontext der Wikipedia die Änderung von Textinhalten oder Bildern durch Benutzer mittels Einstellung offensichtlich unsinniger oder beleidigender, diffamierender, vulgärer oder obszöner Inhalte verstanden. Vandalismus wird typischerweise durch unangemeldete Benutzer verübt. [18]
  Use Cases: Ab wann gilt eine Änderung als Vandalismus? Gibt es Muster, die sich vor einem Vandalismus ereignen?
- Nutzerkreise
  Im Zentrum stehen Nutzer die untereinander in Verdindung stehen.
  Use Cases: Gibt es Nutzergruppen die ähnliche thematische Expertise aufweisen? Bilden sich auch Metagruppen mit noch unbekannten Gemeinsamkeiten?
- Machtprozesse
  Wie in allen Organisationen entstehen auch in der Wikipedia Machtstrukturen.
  Use Cases:Bildet sich die Organisationshierarchie in den Edits ab? Läßt sich eine Tendenz zur Abschottung durch Selbstergänzung der Experten erkennen? [3]
- Request Prediction
  Die Menge der Nutzer-Anfragen an die Wikipedia-Serverinfrastruktur variiert vermtulich.
  Use Cases: Welche Muster lassen sich in der Nutzerlast erkennen? Läßt sich die Nutzerlast vorhersagen?

VI. Ergebnisse

- Die Anzahl und Verteilung der Änderungen an Wikipedia-Seiten sind zu verschieden, um daraus ein allgemeingültiges Muster für relevante Events abzuleiten. Da ein relevantes Ereignis auch nicht auf jeder Wikipedia-Seite bestimmten gesetzmäßgkeiten folgt, spielt die Betrachtung der vergangenen Events eine große Rolle für uns. Erst dadurch kann mithilfe eines Burst-Detection-Algorithmus ein relevantes Ereignis für einzelne Wikipedia-Seiten ermittelt werden. - Der Einsatz von Esper im Analysis Tier, war für die erste Lösung die richtige Wahl. Wie sich jedoch gezeigt hat, konnten wir damit nicht die gewünschten Ergebnisse erzielten. Die zweite Lösung war jedoch in Esper nicht in vollem gewünschten Umfang umsetzbar. Das Ziel, die Analyse von relevanten Ereignissen in Echtzeit umzusetzen, mussten wir auf die Betrachtung der 100 neuesten Events beschränken. Denn Esper ermöglicht keinen Zugriff auf vergangene Events, wie es beispielsweise in Apache Flink, FlinkCEP mit (TODO: wie heißt das Teil?) möglich ist.

Vielleicht geht es auch, wenn man ein Batch-Window mit der Länge von 1 Woche anlegt und jedem Tag ein neues startet

VII. Diskussion

aa

VIII. Ausblick

- Die Lösung 2 aus Kapitel IV wurde nur auf deutschsprachige Wikipedia-Seiten und einen Zeitraum von 2 Tagen, in denen Wikipedia-Events gesammelt wurden, angewandt. Das hatte den Vorteil, dass die Menge an Events und auch die zu erwartenden Ergebnisse überschaubar blieben und eine Validierung praktikabler machte. Es bleibt somit offen, die Anwendbarkeit auf einen größeren Zeitraum und eine größere Menge an Events auszudehnen und die Testresultate zu validieren. - Desweiteren könnten auf Basis der gewonnen Erkenntnissen im Analysis Tier – die Ebenen der Burst-Detection – weitere Analysen durchgeführt werden. Zum Einen ist es eine Möglichkeit, für jede URI die Ebenen wieder zurück in ein Kafka-Topic zu senden und diese damit anderen Analysis Tier-Anwendungen zur Verarbeitung bereitzustellen. Zum Anderen könnten die Ebenen über den Data Access Tier von außen aufrufbar gemacht werden. Ein konkretes Szenario hierfür ist eine Webseite die für einzelne Wikipedia-Webseiten – auf Basis der Ebenen – eine Timeline mit den relevantesten Ereignissen darstellen kann.

IX. Beiträge der Autoren

Phillip Ginter und Daniel Schönle haben gleichermaßen zu dieser Arbeit beigetragen und sind Erstautoren.

Literatur

[1] Wikinews, "Interview mit jimmy wales: Wie geht es weiter mit wikipedia?" [Online]. Available: https://de.wikinews.org/wiki/Special:PermanentLink/577184%3Ftitle%3DInterview_mit_Jimmy_Wales:_Wie_geht_es_weiter_mit_Wikipedia%3F

[2] Wikipedia, "Statistics." [Online]. Available: https://en.wikipedia.org/wiki/Special:Statistics

[3] ——, "Wikipedia." [Online]. Available: https://de.wikipedia.org/wiki/Wikipedia

[4] M. Georgescu, N. Kanhabua, D. Krause, W. Nejdl, and S. Siersdorfer, "Extracting event-related information from article updates in wikipedia," in *Advances in Information Retrieval*, P. Serdyukov, P. Braslavski, S. O. Kuznetsov, J. Kamps, S. Rüger, E. Agichtein, I. Segalovich, and E. Yilmaz, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 254–266.

[5] A. Weiler, M. Grossniklaus, and M. H. Scholl, "An evaluation of the run-time and task-based performance of event detection techniques for twitter," *Information Systems*, vol. 62, pp. 207–219, 2016.

[6] Y. Zhu and D. Shasha, "Efficient elastic burst detection in data streams," in *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '03. New York, NY, USA: ACM, 2003, pp. 336–345. [Online]. Available: http://doi.acm.org/10.1145/956750.956789

[7] Z. Yuan, Y. Jia, and S. Yang, "Online burst detection over high speed short text streams," in *International Conference on Computational Science*. Springer, 2007, pp. 717–725.

[8] R. Ebina, K. Nakamura, and S. Oyanagi, "A real-time burst detection method," in *Tools with Artificial Intelligence (ICTAI), 2011 23rd IEEE International Conference on*. IEEE, 2011, pp. 1040–1046.

[9] T. Ge, L. Cui, B. Chang, Z. Sui, and M. Zhou, "Event detection with burst information networks," in *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, 2016, pp. 3276–3286.

[10] S. Gottschalk, E. Demidova, V. Bernacchi, and R. Rogers, "Ongoing events in wikipedia: a cross-lingual case study," in *Proceedings of the 2017 ACM on Web Science Conference*. ACM, 2017, pp. 387–388.

[11] B. Fetahu, A. Anand, and A. Anand, "How much is wikipedia lagging behind news," in *Proceedings of the ACM Web Science Conference*. ACM, 2015, p. 28.

[12] X. Liu, M. Wang, and B. Huet, "Event analysis in social multimedia: a survey," *Frontiers of Computer Science*, vol. 10, no. 3, pp. 433–446, 2016.

[13] Q. Huang, G. Cervone, and G. Zhang, "A cloud-enabled automatic disaster analysis system of multi-sourced data streams: An example synthesizing social media, remote sensing and wikipedia data," *Computers, Environment and Urban Systems*, vol. 66, pp. 23–37, 2017.

[14] A. Psaltis, *Streaming Data: Understanding the Real-time Pipeline*. Manning Publications, 2017.

[15] J. Gray, *Interprocess Communications in Linux*. Prentice Hall PTR, 2003.

[16] U. Hedtstück, *Complex Event Processing: Verarbeitung von Ereignismustern in Datenströmen*, ser. eXamen.press. Springer Berlin Heidelberg, 2017.

[17] I. Hickson, "Server-sent events," October 2015 (Zugegriffen am: 04.02.2019). [Online]. Available: https://www.w3.org/TR/eventsource/

[18] Wikipedia, "Probleme kollaborativer texterstellung." [Online]. Available: https://de.wikipedia.org/wiki/Wikipedia#Probleme%20kollaborativer%20Texterstellung