API Monitoring mit Predictive Analytics

Ausarbeitung Grundseminar, M-INF

Björn Baltbardis
Master Studiengang Informatik
Hochschule für Angewandte Wissenschaften Hamburg
E-Mail: bjoern@baltbardis.de

I. Motivation

Das soziale Businessnetzwerk *XING* hat mehr als 14 Millionen Mitglieder. Täglich generieren sie zusammen etwa die gleiche Anzahl an API-Anfragen. Zu Lastzeiten können dort so deutlich mehr als zweihundert Anfragen pro Sekunde eingehen.
Bei diesem großen Datenaufkommen ist es gleichermaßen notwendig und herausfordernd, auftretende Fehler, sowie Missbrauch des Systems angemessen zu erkennen, noch bevor ein Schaden entsteht.

Für jede der REST-Anfragen (Representational State Transfer) an einen der Produktivserver des Netzwerks wird im aktuellen System eine interne Nachricht über einen *Message Broker* publiziert. Zur Fehlerprävention, Optimierung und besonders der Missbrauchserkennung sollen diese Nachrichten abonniert und ausgewertet werden. Die Erkennung von Missbrauch entspricht in diesem Sinne vor allem einer Bestimmung von Anomalien, d.h. Mustern die nicht den Erwarteten entsprechen. Sie liegen beispielsweise vor, wenn der Zugang der API unbefugt, im falschen Namen durch Dritte genutzt wird und damit das Verhalten eines angemeldeten Nutzers plötzlich deutlich von seinem Bisherigen abweicht.

II. Zielsetzung

Aus informatischer Sicht ergeben sich mehrere Fragestellungen, die aufgrund der umfangreichen Datenbasis der Thematik *Big Data* unterzuordnen sind. Je nach Analyseverfahren gilt es zunächst eine geeignete Form der Speicherung auszumachen. Auch die Analyse kann sehr aufwändig werden und benötigt ggf. ein Cluster zur Ausführung. Im Folgenden sollen deshalb auch Echtzeitverfahren mit klassischen MapReduce-Verfahren verglichen werden.

Als Basis für MapReduce-Analysen hat sich das *Hadoop Framework* zum de facto Standard entwickelt. Es persistiert die zu verarbeitenden Daten und Ergebnisse verteilt im Cluster und bietet darauf aufbauend parallele MapReduce-Verfahren an. Mit den Apache Projekten *Mahout*, *Pig*, *Hive*, *Spark*, *Drill* und vielen weiteren Komponenten gibt es mittlerweile eine Menge zusätzlicher, teilweise auch kombinierbarer Systeme, die Hadoop als Plattform nutzen. Sie sollen zunächst betrachtet, verglichen und auf Ihre Eignung für die genannte Problemstellung überprüft werden.

Auch die Echtzeit-Auswertung der Daten mit Hilfe eines Information Flow Processing (IFP) Frameworks wird in Betracht gezogen und im Rahmen dieser Arbeit untersucht.

Abschließend ist die Frage zu beantworten, auf welche Weise die gewonnenen Analysedaten sinnvoll in Unternehmensprozesse integriert werden können und wie hoch die Erkennungs- bzw. Fehlerraten ausfallen.

III. Datenermittlung

Grundlage für jede Datenanalyse bilden die auszuwertenden Daten. Es gilt folglich zunächst dem Analyseteil eine Auslese der geeigneten Daten schnell und zugänglich zur Verfügung zu stellen. Wie das geschieht, verdeutlicht folgendes Szenario:

Beim Eintreffen einer Nutzeranfrage an einen der API Server – beispielsweise die Abfrage der eigenen Kontakte (`/v1/users/me/contacts`) – wird diese wie gewohnt beantwortet. Gleichzeitig aber wird eine interne Nachricht publiziert, welche Metadaten der Nutzeranfrage enthält (s. Abb. 1).

```
{
    "duration" : 0.314159265359,
    "recorded_at" : "2015-02-28T12:48:31+02:00",
    "method": "GET",
    "path_info": "/v1/users/me/contacts",
    "host": "xws-15.xxx.com",
    "params": {
      "limit": "100",
      "offset": "0",
    },
    "status": 200,
    "size": 1234,
    "ip_address": "XXX.XXX.XXX.XXX",
    "consumer_key": "123XXXXXXXXXXXXXXX",
    "access_token": "ABCXXXXXXXXXXXXXXX"
}
```

Abb. 1. Metadaten einer HTTP-Anfrage als JSON-Nachricht

Die abgebildete Nachricht beinhaltet dabei vor allem ein Anfragedatum `recorded_at`, Daten über die HTTP-Anfrage (`method`, `path_info`, `params`), Account-Informationen des Anfragenden (`consumer_key`, `access_token`), den HTTP-Statuscode der Antwort und die IP Adresse (`ip_address`) des Anfragenden.

Nachrichten dieses Formats sind Grundlage für die nachfolgenden Auswertungsmechanismen. Sie können entweder in Echtzeit analysiert oder zunächst persistiert und anschließend analysiert werden.

IV. Erkennungsmetriken

Die zu automatisierenden Analyseverfahren sollen nachfolgend zunächst in der Theorie betrachtet werden. Das Hauptaugenmerk gilt dabei der Erkennung von möglichem Missbrauch und der Früherkennung von Fehlern.

A. *Missbrauchserkennung*

Die Authentifizierung eines Nutzers geschieht grundsätzlich über *OAuth* (vgl. Abb. 2). Eine externe Applikation (ein *Consumer*) sendet dabei eine Zugriffsanfrage für einen bestimmten Nutzer und festgelegte Zugriffsrechte an den Server. Der Nutzer erteilt dem Server die Erlaubnis, der Applikation den Zugriff auf seine Daten zu gestatten. Die Applikation kann nun ihre eindeutige ID (den *Consumer Key*) durch einen zeitlichen begrenzten Zugriffstoken, den sog. *Access Token*, eintauschen. Er ist eine eindeutige Abhängigkeit gegenüber einer spezifischen Applikation und einem bestimmten Benutzer.

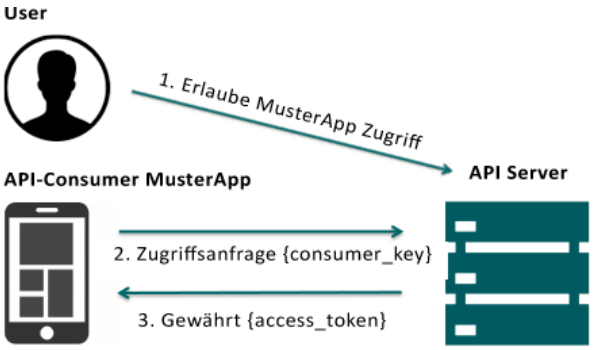

Abb. 2. Ablauf-Schema der OAuth-Authentifizierung

Der OAuth Mechanismus birgt allerdings die Risiken, dass der Consumer Key oder der Access Token gestohlen werden kann. Der Diebstahl eines Access Tokens könnte für den einzelnen Benutzer einen größeren Schaden bedeuten, da er zusammen mit dem Consumer Key Direktzugriff auf seine Daten erlaubt.

Doch auch der alleinige Diebstahl des Consumer Keys bedeutet bereits ein nicht zu unterschätzendes Risiko. In Applikationen (*Apps*) für Mobiltelefone beispielsweise muss dieser Schlüssel meist hartcodiert hinterlegt werden. Durch sog. *Reverse Engineering* lässt er sich, je nach Verschleierungsaufwand des Entwicklers, mehr oder weniger leicht auslesen. Er bietet zwar zunächst keinen Direktzugriff auf Nutzerdaten, erlaubt dem Angreifer aber sich als die bestohlene Applikation auszugeben. Hieraus können folgende Angriffsszenarien entstehen:

1) Verschleierung der API Nutzung
Applikationen werden zur Nutzung der API manuell freigeschaltet. Jemand der gar nicht zur Nutzung berechtigt ist, könnte sich durch den Diebstahl eines Consumer Keys Zugang beschaffen.

2) Umgehung von Throttling
Weiterhin sind Applikationen in der Anzahl ihrer Anfragen pro Zeitintervall limitiert, bevor beim überschreiten bestimmter Grenzwerte ein sog. *Throttling* (eine temporäre Sperre) greift. Durch die Nutzung fremder Consumer Keys kann dieser Sicherheitsmechanismus umgangen werden. Große Datenmengen können in der Folge einfacher abgeschöpft werden.

3) API Nutzung im falschen Namen
Die API könnte bewusst im Namen einer anderen Applikation genutzt werden, um beispielsweise Fehler zu generieren und die Applikation zu diskreditieren.

4) Throttling Manipulation
In Anlehnung an 2) und 3) ist auch denkbar, dass ein Angreifer die Throttling-Werte für Applikationen durch eine große Anzahl von Anfragen in die Höhe treibt, um eine temporärere Sperre dieser zu erreichen. Die echte Applikation wäre nicht mehr nutzbar. Die Ausfälle selbst können einen hohen Schaden verursachen, aber auch die aus dem Ausfall resultierende Rufschädigung stellt ein Risiko für die beteiligten Unternehmen dar.

5) Rechtemissbrauch
Einige Applikationen bekommen über ihre Consumer Keys besondere Rechte zuteil, können beispielsweise nicht-öffentliche Anfragen nutzen, oder haben andere besondere Vorzüge. Ein Angreifer bringt sich durch Diebstahl des Consumer Keys in die selbe Position.

Um die zahlreichen Angriffsszenarien frühzeitig zu erkennen, sollen die gewonnenen Daten schnell und zuverlässig auf spezifische Merkmale analysiert werden. Grundannahme ist, dass die Aufrufschemata einer Anwendung meist gleich bleiben, denn der Algorithmus, nachdem unterschiedliche REST Aufrufe ausgeführt werden, bleibt weitestgehend unverändert. Das Nutzverhalten kann nur innerhalb festgelegter Rahmen variieren. Anomalien können so beispielsweise an den folgenden Merkmalen erkannt werden:

- Abweichende Reihenfolge mehrerer API-Aufrufe
- Abweichende zeitliche Aspekte, wie beispielsweise Pausenlängen zwischen den einzelnen Anfragen
- Abweichende Parameter der Aufrufe
- Aufrufe bislang ungenutzter Ressourcen oder gar von Ressourcen, für die eigentliche keine Berechtigung vorliegt
- Plötzlicher, für die jeweilige Applikation untypischer, Anstieg von Anfragen

B. *Fehlerprävention*

Auch die frühzeitige Erkennung von Fehlern soll durch das Analyseverfahren bewerkstelligt werden. Das frühzeitige Auffinden dieser kann sicherheitsrelevant sein und beim berichtigen experimenteller Features helfen. Fehler sind in diesem Sinne REST-Anfragen, die als Ergebnis einen HTTP-Fehlercode zurückgeben. Es ist zu unterscheiden zwischen 500er Serverfehlern und den 400er Fehlern, welche auf eine Fehlnutzung hindeuten. Sie reichen in der Praxis von klassischen *404 Not Found* und *403 Forbidden* Fehlern, über Fehler, die die Überschreitung maximaler Textlängen deutlich machen, bis hin zu fachlich komplexeren Fehlern, die auf das Überschreiten der eigenen Limits für private Nachrichten hinweisen.

Täglich können einige tausend fehlerhafte Anfragen als „normal“ abgetan werden. Sie zeugen von Timeouts durch verbesserungswürdiges *Loadbalancing* oder etwa Nutzerfehler. Analytisch interessant sind vor allem plötzliche Anstiege der Fehlerrate (s. Abb. 3)

Abb. 3. Plötzlicher Anstieg der Fehlerrate

Das zu wählende Analyseverfahren soll damit zum einen in der Lage sein, plötzliche Anstiege schnell zu erkennen und zum anderen auch langfristig gehäuft auftretende Fehler zu ermitteln.

V. Daten Analyse

Zentraler Gegenstand der Anomalieerkennung ist die Analyse der aktuellen Daten. Sie kann „offline“, d.h. verteilt und asynchron gegenüber der Datengenerierung stattfinden, oder aber „online“ bzw. in Echtzeit [7]. Auch Hybridvarianten sind möglich. Zunächst sollen die beiden Ansätze aber getrennt betrachtet werden.

A. *Verteilte Auswertung*

Hadoop ist das wohl am weitesten verbreitete Framework zur verteilten Verarbeitung großer Datenmengen. Mit seinem *Hadoop Distributed File System* (HDFS) bringt es ein eigenes, hochverfügbares Dateisystem zur Speicherung sehr großer Datenmengen mit. Diese können mittels paralleler Map-Reduce-Anfragen vergleichsweise schnell analysiert werden, wenngleich der Batch Processing Ansatz von Hadoop durch interne Vorbereitung und Scheduling zu höheren Latenzen führt.

1) *Hadoop Komponenten*

Das Framework lässt sich durch viele verschiedene Erweiterungen ergänzen. Einige wichtige seien im Folgenden genannt:

a) ***Apache Hive*** erweitert Hadoop um die SQL ähnliche Abfragesprache „HiveQL“ (Bsp. s. Abb. 4), wobei die Verwendung eigener MapReduce Blöcke weiterhin möglich ist. Hive bietet keine Echtzeitfunktionalität und ist aufgrund der Hadoop Eigenheiten relativ langsam. Ein Query kann bereits bei relativ geringen Datenmengen von wenigen hundert Megabyte einige Minuten in Anspruch nehmen [1].

```
SELECT a.foo FROM invites a WHERE a.bar='1337';
```

Abb. 4. HiveQL: Einfaches SELECT-Beispiel

b) ***Apache Pig*** bzw. die zugehörige Sprache *Pig Latin* ist eine Hochsprache, dessen programmierte Datenflüsse in MapReduce Sequenzen kompiliert wird. Sie lassen sich lokal oder mittels eines Hadoop Clusters ausführen. Erlaubt sind Operationen der relationalen Algebra (z.B. `join`, `filter`, `project`), sowie funktionale Operatoren wie Map/Reduce (vgl. Bsp. Abb. 5) [2].

```
raw = LOAD 'example.log' USING PigStorage('\t') AS
      (user, time, query);
clean1 = FILTER raw BY org.apache.pig.tutorial.
      NonURLDetector(query);
clean2 = FOREACH clean1 GENERATE user, time,
      org.apache.pig.tutorial.ToLower(query) as
      query;
```

Abb. 5. Pig Latin: Dateflussbeispiel mit FILTER

c) ***Apache Drill*** ist eine ebenfalls SQL basierte Query-Engine für Hadoop und NoSQL. Sie kann mit semistrukturierten Daten arbeiten und erlaubt beispielsweise die direkte Integration von .csv / .json Dateien in den Datenbestand (s. Abb. 6). Die Drill-Query-Engine baut auf Apache Hive auf und möchte durch geringe Latenzen und volle ANSI SQL Rückwärtskompatibilität überzeugen [3].

```
SELECT * FROM cp.`employee.json`;
```

Abb. 6. Drill: Beispiel für die Integration einer .json Datei in ein Query

d) ***Apache Mahout*** ist eine skalierbare *Machine-Learning-Engnine*, die auf Hadoop aufbaut. Viele bekannte Algorithmen sind bereits integriert und für die verteilte Ausführung optimiert. Beispiele sind Klassifikations-algorithmen wie *Naive Bayes*, *Random Forest*, *Hidden Markov Models* (nur lokal) und *Multilayer Perceptron* (nur lokal), sowie die Clustering-Algorithmen *k-Means, Fuzzy k-Menas, Streaming k-Means* und das *Spectral Clustering* [4].

2) *Clustering*

Nach *Chandola u.a.* unterscheidet man in bei den Verfahren der Anomalieerkennung zwischen der *Klassifikation*, dem *Clustering*, dem *Nearest Neighbor Verfahren*, *statistischen Verfahren* und anderen, beispielsweise visuellen Verfahren [7]. In der Arbeit „Online Performance Anomaly Detection“ [14] wurde beispielsweise erfolgreich auf statistische und visuelle Verfahren zurückgegriffen. Die Komplexität der Missbrauchserkennung verbietet allerdings den alleinigen Gebrauch dieser Verfahren. Besonders interessant ist deshalb der zuvor genannte Zweig des verteilten Clustering zur Erkennung von Anomalien. Nachfolgend sollen die Grundverfahren des Clustering vorgestellt werden [8]:

a) *Partitionierendes Clustering*

Dieses Verfahren bedingt eine zuvor bekannte Zahl von Clustern, in welchen anschließend Clusterzentren bestimmt werden. Eine Fehlerfunktion wird minimiert, während das Zentrum - meist durch Mittelwertbildung - neu berechnet wird, bis sich die Zuordnungen nicht mehr ändern [8] (vgl. Abb. 7).

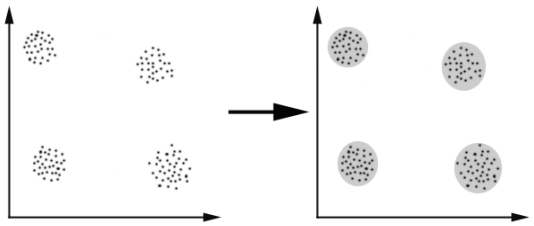

Abb. 7. Partitionierendes Clustering mit 4 Clustern und zwei Merkmalsdimensionen.

b) *Hierarchisches Clustering*

Die Distanzen einzelner Datensätze zueinander werden in Hierarchien überführt, dessen Distanzen zueinander ebenfalls in der Hierarchie abgebildet werden. Es entsteht ein Baum, d.h. eine hierarchische Struktur von Gruppen [8].

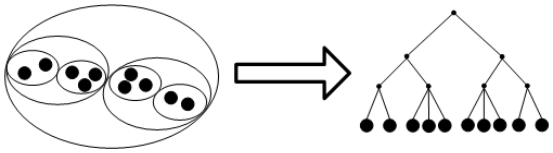

Abb. 8. Hierarchische Überführung beim hierarchischen Clustering

c) Dichtenbasiertes Clustering

Bei dichtebasiertem Clustering werden Cluster als Objekte in einem d-dimensionalen Raum betrachtet, welche dicht beieinander liegen, getrennt durch Gebiete mit geringerer Dichte. Nicht zentrische Cluster könnnen so deutlich besser erfasst werden, als bei anderen Verfahren [8] (vgl. Abb. 9).

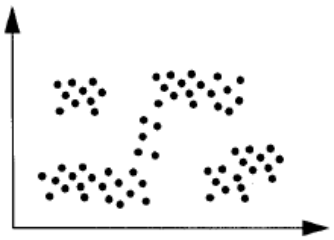

Abb. 9. Dichtenbasiertes Clustering ist flexibler gegenüber nicht zirkulären Clustern

Ein Beispiel für ein dichtenbasiertes Clustering ist das Mean-Shift Clustering. Es ist besonders effektiv und bewies in der Arbeit „*A scalable, non-parametric anomaly detection framework for Hadoop*“ von Yu u.a., seine hohe Erkennungsperformanz gegenüber den anderen Verfahren (vgl. Abb. 10).

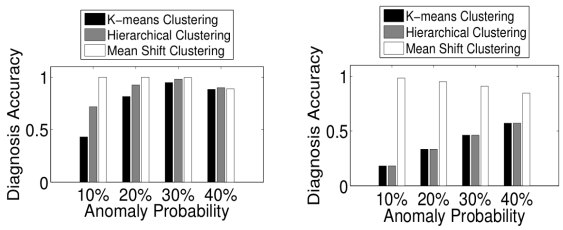

Abb. 10. Die Genauigkeit des Mean-Shift Algorithmus steigt mit der Anzahl vorhandener Anomalien (links: eine, rechts: vier) [9]

Durch die bereits genannte Machine-Learning-Engine *Mahout* könnte auf verteilte Verarbeitung in einem Hadoop Cluster zurückgegriffen werden. Performanz und Genauigkeit dieses Algorithmus bilden einen vielversprechenden Ansatz.

B. Online Auswertung

Die Echtzeitanalyse eintreffender Daten entspricht dem sog. *Information Flow Processing* (IFP). IFP ist ein Begriff der durch die Arbeit „*Processing flows of information: From data stream to complex event processing*“ geprägt wurde [10]. Sie fässt die zwei Forschungszweige des *Data Stream Processing* (DSP) und des *Complex Event Processing* (CEP) mit einem vereinheitlichten Vokabular zusammen und vergleicht sie einander.

Gemeinsame Grundaufgabe von IFP-Systemen ist demnach die kontinuierliche Auswertung und Kontrolle von Informationsflüssen in Echtzeit. Sie kommen bei großen Datenmengen (z.B. Sensornetzwerke) oder zeitkritischen Prozessen zum Einsatz (z.B. Finanzdaten oder Verkehrs-überwachung) [10]. Klassische Datenbank Management Systeme (DBMS) sind auf Persistenz und Indexierung angewiesen, ehe sie zu Analysezwecken genutzt werden können. Ein erster Schritt in Richtung IFP Systeme geschah mit den *Active Database Systems* genannten Systemen [10], sie sondern Benachrichtigungen in vordefinierten Situationen (Regeln) ab. Regeln werden nach dem ECA Schema (*Event, Condition, Action*) definiert.

Als Weiterentwicklung entstanden die zuvor genannten Ansätze des DSP und CEP, welche nachfolgend näher betrachtet werden sollen.

1) Data Stream Processing

Data Stream Management Systeme (DSMS) erhalten fortwährend generierte Daten als Input. Die Daten müssen nicht persistiert werden und liegen i.d.R. im Hauptspeicher, was einen Performanzvorteil bedeutet [11]. DSMS sind deshalb besonders gut zur Auswertung großer Datenmengen geeignet. Sie erlauben nur sequentiellen Zugriff und beständige Queries, gegenüber wahlfreiem Zugriff und Einzelabfragen bei klassischen DBMS [12]. Aufgrund der möglicherweise sporadischen Ankunft von Daten sind Abfragen nicht planbar.

2) Complex Event Processing

Beim Complex Event Processing (CEP) wird der eintreffende Fluss an Informationen als Folge von Ereignissen betrachtet. CEP entstand aus der *publish-subscribe* Domäne, indem das konventionelle Filtern von Ereignissen um die Einbeziehung bereits vergangener Ereignisse ergänzt wurde [10]. Aufgabe von CEP Systemen ist, atomare Events der Observer zu immer höherwertigen (*Situationen*) zu kombinieren, welche anschließend externen Konsumenten zugänglich gemacht werden können [13] (vgl. Abb. 11).

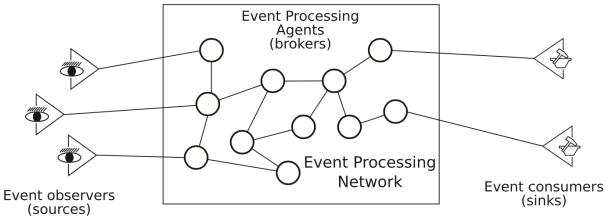

Abb. 11. Schematischer Ablauf des Complex Event Processing

3) Information Flow Processing Systeme

Wie zuvor erwähnt, wird mit dem Ansatz des Information Flow Processing versucht, die Vorteile von DSP und CEP zusammenzuführen, sowie ein generalisiertes Vokabular zu schaffen. Statt den Begriffen *event, stream* und *data,* werden die eintreffenden Daten als Informationen und Fluss bezeichnet [10]. Typische Anforderungen an IFP Systeme sind Echtzeit-verarbeitung, eine Abfragesprache zur Beschreibung der Informationsverarbeitung und Skalierbarkeit. Nachstehend sei die innere Funktionsweise von IFP Systemen grob skizziert:

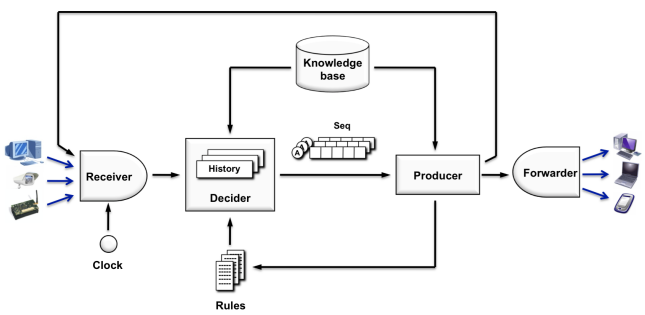

Abb. 12. Schematischer Ablauf innerhalb von IFP Systemen

Wie Abb. 12 zeigt, gibt es zunächst den sog. *Receiver*, welcher Informationen annimmt und (optional getaktet) an den sog. *Decider* weiterleitet. Gemeinsam mit dem *Forwarder* implementiert der Receiver externe Protokolle und stellt eine Kommunikationsbrücke dar. Im *Decider* treffen Informationen und zuvor festgelegte Regeln aufeinander. Trifft eine Regel zu, werden die Informationen an den *Producer* weitergeleitet. Auch ältere Informationen (*History*) können bei Anwendung der Regeln berücksichtigt werden. Der *Producer* nimmt nun ggf. eine Aufarbeitung der Informationen vor und wendet zuvor festgelegte Transformationen an. Die Informationen werden an den *Forwarder* weitergeleitet und anschließend an externe Konsumenten verteilt. Einige Systeme erlauben dem *Producer* darüber hinaus zur Laufzeit Änderungen an den Regeln vorzunehmen. Ebenso können Daten rekursiv an den *Receiver* zurückgegeben werden – ähnlich wie es die Netzwerke beim CEP erlauben. Schließlich können Informationen optional durch den *Producer* gespeichert werden und dem *Decider* über die sog. *Knowledge Base* zur Einbeziehung bei Anwendung der Regeln zur Verfügung gestellt werden [10].

IFP Systeme (IFPS) können sehr vielfältig sein, sie unterscheiden sich in einigen zentralen Merkmalen voneinander, die nachfolgend beschrieben sind [10]:

a) Verteilung – Ist das System zentralisiert oder verteilt? Gibt es mehrere Komponenten? Welche Komponente hält die Informationen vor, welche die Regeln?

b) Interaktion – Das Interaktionsmodell teilt sich auf in das *Observation Model* zum Datenabgriff von den Quellen, das *Forwarding Model*, welches die interne Kommunikation mehrerer Komponenten beschreibt und das *Notification Model* zur Abgabe der Informationen an externe Konsumenten. Für jedes der drei Modelle gibt es grundsätzlich jeweils einen *Push*- oder *Pull*-Ansatz, d.h. die Komponente kann andere Komponenten bei ‚Neuigkeiten' aktiv benachrichtigen, oder verhält sich selbst passiv und wird befragt.

c) Daten – Das Datenmodell beschreibt die interne Darstellung, sowie den Umgang mit Daten. Während ein DSMS beispielsweise von einer *Manipulation von Daten* ausgeht, spricht mit beim CEP von einer *Akkumulation von Ereignissen*. Weiterhin kann die eigentliche Darstellung in Tupeln (typisiert und nicht typisiert), in Form von Objekten und beispielsweise als XML stattfinden. Das Datenmodell kann natürlich Einfluss auf andere Merkmale eines IFPS, wie etwa die Sprache haben.

d) Zeit – Die Beziehung zwischen Informationen wird in IFPS üblicherweise zeitlich ausgedrückt, konkret in einer *passierte-vor-Beziehung*. Es gibt vier grundlegende Ansätze:

- *stream only:* Bei DSMS wird häufig auf diesen Anasatz gesetzt, hierbei wird alleine die Reihenfolge der eintreffenden Informationen berücksichtigt.
- *absolute:* Absolute Zeitstempel werden verwendet. Dies bringt Probleme wie out-of-order-arrivals mit sich, bei denen Nachrichten zu spät, d.h. nach zeitlich später abgesendeten Nachrichten eintreffen.
- *causal:* Ereignisse bzw. Informationen werden als kausal zusammenhängend betrachtet. Ereignis A ist eine Folge von Ereignis B.
- interval: Ereignisse werden anhand von Intervallen (Start- & Enddatum) in Beziehung gesetzt.

e) Regeln – Das Regelmodell beschreibt welche Transformationsregeln und Erkennungsregeln angewendet werden können.

f) Sprache – Das Sprachmodell beschreibt die Sprache der Regeln. Bei den Transformationsregeln unterscheidet man zwischen *deklarativen Sprachen* (erwartete Ergebnisse werden beschrieben, meist SQL-ähnlich) und den *imperativen Sprachen* (meist primitive Operationen). Erkennende Regeln dienen zur Auswahl der gewünschten Informationen. Sie erlauben meist logische Operatoren, zeitliche Verweise und Vergleiche, sowie Inhaltsfilter.

g) Consumption Policy – Die CP. beschreibt den „Verbrauch" von Informationen. Wie wird mit einzelnen Informationen umgegangen, wenn sie Teil eines erkannten Musters sind? Können sie für weitere benutzt werden und dürfen mehrfache Benachrichtigungen (*fire*) abgesondert werden?

In der Arbeit „*Processing flows of information: From data stream to complex event processing*" [10] wurden weit über 30 Syteme anhand der genannten Merkmale verglichen. Man kam zu dem Schluss, dass verschiedene Systeme sich für verschiedene Anforderungen jeweils besser oder schlechter eignen. Sie fokussieren andere Aspekte, weshalb es Teil des Grundprojektes werden soll, diese näher zu analysieren und das, für die beschriebenen Anforderungen geeignetste, System zu finden.

Besonders vielversprechend scheinen die Systeme *SASE* bzw. dessen Weiterentwicklung *SASE+* und *TESLA / T-Rex*. Sie fallen durch ein vielfältiges Aufgabenspektrum, eine relativ hohe Verbreitung und gute Community-Unterstützung auf. Die Syntax ist ferner SQL ähnlich und damit sehr zugänglich. Nachfolgendes Beispiel soll den grundsätzlichen Aufbau einer Erkennungsregel verdeutlichen:

```
1 Define Fire(area: string, measuredTemp: double)
2 From   Smoke(area=$a) and
3        last Temp(area=$a and value>45)
4        within 5 min. from Smoke
5 Where  area=Smoke.area and
6        measuredTemp=Temp.value
```

Abb. 13. TESLA / T-Rex Beispiel für einen Rauchmelder

Zeile 1 der Abb. 13 beschreibt die auszuführende Aktion beim Erkennen eines Musters (Fire). Die Zeilen 2 und 3 stellen die eigentliche Erkennungsregel dar. Ist Rauch für den Bereich *area* vorhanden und ist die letzte Temperatur des Bereichs > 45 °C, gibt es einen Treffer. Statt ‚last' könnte auch die Schlüsselwörter each, first within, n-first und n-last verwendet werden. Zeile 4 begrenzt den Referenzzeitraum für Smoke auf die letzten fünf Minuten. Die letzten beiden Zeilen 5 und 6 dienen zur Umbennenung und synchronisation mit den Parametern area und measureTemp [10].

C. Hybrid-Lösung

Die vorgestellten Ansätze der verteilten- und Onlineauswertung bieten für sich bereits viele Möglichkeiten eine solide Anomalieerkennung zu entwickeln. Es existieren aber auch hybride Ansätze, welche Vorteile aus beiden Welten vereinen sollen.

Allem voran genannt ist hier die sog. *Lambda architecture*, ursprünglich von Nathan Marz 2012 das erste Mal beschrieben. Die λ-Architektur veranschaulicht die erfolgreiche Zusammenarbeit von *Batch-* und *Streamprocessing* Ansätzen in einem einzelnen System. Man teilt das Gesamtsystem in drei Ebenen auf: den *Batch Layer*, den *Serving Layer* und den *Speed Layer* (vgl. Abb. 14).

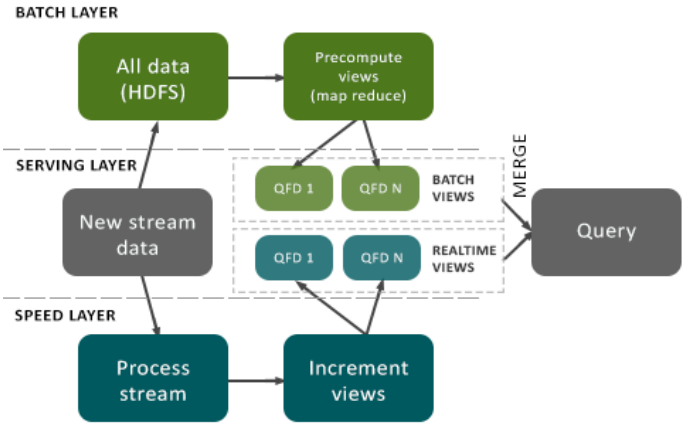

Abb. 14. λ-Architektur, Bild nach [16]

Eintreffende Daten werden gleichzeitig auf den *batch layer* und den *speed layer* verteilt. Auf dem *batch layer* werden die Daten mittels Hadoop persistiert. Eine Änderung der Daten stößt eine Neuberechnung der *batch views* an. Dies sind beständige Anfragen, die hier als *Question Focused Datasets* (QFDs) bezeichnet werden. Die *batch views* werden auf Basis des gesamten Datenbestandes neu berechnet, was je nach Datenbestand und Komplexität einige Stunden in Anspruch nehmen kann [16].

Ebenso wie der *batch layer* erhält auch der *speed layer* neu eintreffende Daten. Er wird beispielsweise mit Apache Storm umgesetzt, dient zur Kompensation der Hohen Latenz des *batch layers* und berechnet inkrementelle *realtime views* für das Delta der Daten, die noch nicht in den *batch views* berücksichtigt sind [15]. Hat der *batch layer* die Berechnung für neue Daten abgeschlossen, werden die Daten des *speed layers* verworfen.

Im *serving layer* kann dann beispielsweise auf die Apache Datenbank HBase zurückgegriffen werden, um die Daten und Views des speed layers zwischen zu speichern. HBase ist dabei besonders gut für Daten geeignet, die selten geändert aber häufig ergänzt werden. So lassen sich mit HBase Milliarden von Datensätzen verteilt und effizient verwalten. Mittels der Hadoop Erweiterung *Cloudera Impala* können die Daten aus *batch views* und *realtime view* nun zusammengeführt werden und ergeben einen einheitlichen, strukturierten Satz an Daten [5].

Das vorgestellte Apache Storm ist als verteiltes Echtzeit-Datenverarbeitungssystem Kernstück der λ-Architektur [6]. Es ist direkt kompatibel mit verschiedenen Message-Brokern und kann so beispielsweise direkt die Eingangs genannten Statusnachrichten abonnieren.

Verwendung findet Apache Storm beispielsweise in der Arbeit „*Scalable hybrid stream and hadoop network analysis system*" [17]. Dort wurden, zur Analyse des Netzwerkverkehrs einer Bildungseinrichtung, ebenfalls AMQP Nachrichten abonniert und in Storm eingespeist. Gleichzeitig wurden die Daten mit Hadoop zur ergänzenden MapReduce-Analyse persistiert. Diese Entscheidung wurde getroffen, da sonst zu viele CEP Regeln für Storm entstanden wären und die Echtzeit-Performanz gelitten hätte. Die Auswertung der Hadoop und Storm Daten geschah aber, anders als bei der λ-Architektur vorgeschlagen, manuell und getrennt voneinander. Hier scheint Potential für das Echtzeit-Query-Tool Impala zu sein, welches die Ergebnisse beider Layer zusammenführen kann.

VI. Risikien

Bei dem beschriebenen Vorhaben sind Datenschutzbedenken nicht von der Hand zu weisen und auch falsche Handlungen aufgrund von sog. *False-Positives* müssen ausgeschlossen werden können. Bei der Auswertung der Daten ist deshalb auf eine möglichst weitreichende Anonymisierung dieser zu achten. Ziel ist es schließlich nicht das Verhalten von Einzelpersonen zu analysieren, sondern das von Consumern, das heißt Applikationen oder aufrufende Websites.

Eine Auseinandersetzung mit der Qualität der zu entwickelnden Metrik, sowie der Umgang mit *False-Positives* ist ein Aspekt, der im Rahmen des Grundprojekts geklärt werden soll.

VII. Zusammenfassung

Dieses Dokument sollte einen ersten Einblick in die technisch mannigfaltigen Möglichkeiten zur Analyse, der Anomalieerkennung und der Vorhersage bei großen Datenbeständen, wie etwa einer viel genutzten, öffentlichen REST Schnittstelle geben. Es wurde gezeigt welche Motivation den Autor umtreibt, welche informatischen Fragestellungen entstehen und welche unternehmerische Relevanz diese Thematik inne hat. Ferner wurden fachliche Metriken zur Erkennung einer Anomalie beschrieben und elementare Algorithmen zum Finden dieser vorgestellt. Schließlich wurde mit den Verteilten- und Echtzeitsystemen gezeigt, welche unterliegenden Systeme zur Analyse genutzt werden könnten. Es wurde mit der Lambda-Architektur auch ein hybrider Ansatz behandelt, der vielversprechende Möglichkeiten bietet.

VIII. Ausblick und Ziele im Grundprojekt

Die während dieser Arbeit gesammelten Informationen sollen im Grundprojekt als Basis für einen Vergleich und eine tiefergehende Untersuchung der verschiedenen technischen Ansätze dienen. Vor allem der hybride Ansatz nach der Lambda-Architektur soll nachempfunden und getestet werden. Aber auch die Metriken als theoretischer Teil der Thematik sollen vertieft betrachtet und erweitert werden. Zusammen mit dem genannten Unternehmen soll so bereits im Grundprojekt eine nähere Idee davon entstehen, wie die Daten gezielt zur Anomalieerkennung eingesetzt werden können. Überdies sollen konkrete Datenstrukturen und Abfragen zur Erkennung von Anomalien entstehen. Im Hauptprojekt sollen diese dann vertieft und mit Produktivdaten getestet und ausgewertet werden. Außerdem gilt es sinnvolle Unternehmensprozesse in Form von Handlungshinweisen beim Auftreten eines Befundes zu modellieren.

Literatur

[1] Apache Hive, https://cwiki.apache.org/confluence/display/Hive/, Abruf 28.02.2015

[2] Apache Pig, https://github.com/apache/pig/, Abruf 28.02.2015

[3] Apache Drill, http://drill.apache.org/, Abruf 28.02.2015

[4] Apache Mahout, http://mahout.apache.org/, Abruf 28.02.2015

[5] Cloudera Impala, http://www.cloudera.com/content/cloudera/en/products-and-services/cdh/impala.html, Abruf 28.02.2015

[6] Apache Storm, http://hortonworks.com/hadoop/storm/, Abruf 28.02.2015

[7] Varun Chandola, Arindam Banerjee, and Vipin Kumar. 2009. Anomaly detection: A survey. *ACM Comput. Surv.* 41, 3, Article 15 (July 2009), 58 pages.

[8] Jürgen Cleve and Uwe Lämmel, Data Mining, De Gruyter Oldenbourg, 978-3486713916, 2014

[9] Li Yu and Zhiling Lan. 2013. A scalable, non-parametric anomaly detection framework for Hadoop. In Proceedings of the 2013 ACM Cloud and Autonomic Computing Conference (CAC '13). ACM, New York, NY, USA

[10] Gianpaolo Cugola and Alessandro Margara. 2012. Processing flows of information: From data stream to complex event processing. ACM Comput. Surv. 44, 3, Article 15 (June 2012)

[11] Arasu, A., Babcock, B., Babu, S., Datar, M., Ito, K., Nishizawa, I., Rosenstein, J., and Widom, J. 2003. Stream: The stanford stream data manager. IEEE Data Eng. Bull. 26.

[12] Vorlesung: Distributed Data Management, SS2013, Dr. Ing. Sebastian Michel, TU Kaiserslautern

[13] Luckham, D. C. 2011. The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems. Addison-Wesley Longman Publishing Co., Inc., Boston, MA.

[14] Bielefeld, Tillmann Carlos. Online performance anomaly detection for large-scale software systems. Diss. Kiel University, 2012.

[15] Marz, Nathan, and James Warren. Big Data: Principles and best practices of scalable realtime data systems. Manning Publications Co., 2015.

[16] The Lambda architecture: principles for architecting realtime Big Data systems, http://jameskinley.tumblr.com/post/37398560534/the-lambda-architecture-principles-for, Abruf 28.02.2015

[17] Vernon K.C. Bumgardner and Victor W. Marek. 2014. Scalable hybrid stream and hadoop network analysis system. In Proceedings of the 5th ACM/SPEC international conference on Performance engineering (ICPE '14). ACM, New York, NY, USA