Oberseminar Datenbanksysteme - Aktuelle Trends

Fakultät Informatik, Mathematik und Naturwissenschaften
der Hochschule für Technik, Wirtschaft und Kultur Leipzig

Data Streams & Complex Event Processing

vorgelegt von
Erik Rohkohl
erik.rohkohl@stud.htwk-leipzig.de

Prof. Dr.-Ing. Thomas Kudraß

Leipzig, den 29. Juni 2017

Abstract. In fast allen Bereichen des Lebens trifft man auf verteilte IT-Systeme, ob im Unternehmenskontext, Verwaltung, Militär oder als Fabrik, die im Sinne von Industrie 4.0 völlig automatisiert produziert. Diese Systeme verbindet eine Gemeinsamkeit, sie senden eine Vielzahl von Nachrichten pro Sekunde durch ihr Netzwerk, um Status- und Prozessinformationen unter ihren Komponenten auszutauschen. *Complex Event Processing* (CEP) versucht diese Datenströme auszuwerten, zu steuern und zu verarbeiten, indem es Muster in den Ereignissen erkennt. Dafür fasst CEP eine Menge von Techniken, Methoden und Werkzeugen zusammen, mit denen Muster modelliert und diskrete Datenströme untersucht werden können (Luckham, 2002).

Nach einem einführenden Kapitel diskutiert Abschnitt 2 Anwendungsszenarien für CEP im Netzwerkmanagement, öffentlichen Raum und in der Finanzbranche. Der dritte Abschnitt erläutert Fenstertechniken für Datenströme, die Event-Algebra und als Vertreter für Anfragesprachen die Continuous Query Language. Wohingegen der letzte Abschnitt einen Überblick über die Technologie Apache Storm gibt.

1 Einführung

Nachrichten in einem verteilten IT-System stehen immer in Relation zu einem dedizierten Zeitpunkt, an dem sie versandt werden. Im Umfeld von CEP wird eine Nachricht in Abhängigkeit der Zeit als *Event* bezeichnet. Verschiedene *Event Producer* erzeugen Events in einem *Event Processing Network* (EPN), welche zum Beispiel eine Transaktion oder ein Sensordatum abbilden.
Unverarbeitete Events, oder auch rohe Events, bezeichnet diese Arbeit im Folgenden als *Simple Events*. Mithilfe einer *Event Query Language* (EQL) kann der Entwickler Muster modellieren, die eine CEP-*Engine* versucht in Datenströmen zu erkennen und deren Auftreten die CEP-Engine an einen *Event Consumer* meldet. Dieser Consumer reagiert automatisiert auf dieses Event.
Konnte die CEP-Engine ein Muster in einem Datenstrom bestehend aus mehreren Simple Events finden, abstrahiert sie diese zu einem *Complex Event*.

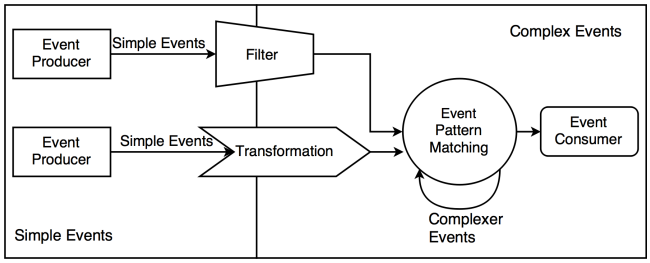

Abbildung 1: Aufbau eines CEP-Systems nach (Gad et al., 2013).

Abbildung 1 zeigt schematisch den Aufbau eines CEP-*Systems*. In ihm erzeugen Event-Producer einen Datenstrom aus Simple Events. Das CEP-System filtert und normiert diese Datenströme während des *Preprocessing*. Anschließend versucht die CEP-Engine mittels *Pattern-Matching* Muster in den Datenströmen zu identifizieren. Natürlich kann ein CEP-System auch Muster von Complex Events, also bereits abgeleiteten Events, untersuchen, diese Arbeit beschränkt sich jedoch auf die Mustererkennung in Sequenzen von Simple Events.

2 Anwendungsszenarien

Dieser Abschnitt untersucht Anwendungsszenarien für CEP aus den Bereichen Netzwerk- und Systemmanagement, Anwendungen für Smart Cities und Erkennung von Frauds im Finanzwesen.

2.1 Netzwerkmanagement

Die Motivation für den Einsatz von CEP-Tools im Netzwerkmanagement besteht darin, völlig automatisch durch Analyse des Datenverkehrs Angriffe, Overloads oder defekte Hardware zu erkennen. Dabei darf die Verarbeitung der CEP-Engine jedoch nicht so aufwendig sein, dass die Bandbreite nicht mehr der eigentlichen Anwendung genügt.
Mögliche Simple Events im Bereich Netzwerkmanagement sind zum Beispiel das Versenden von Datenpaketen, ein Host kommt zum Netzwerk hinzu oder verlässt es sowie ein ausfallender Dienst, der zum Abschied noch eine Nachricht sendet.
Gad, Kappes, Boubeta-Puig und Medina-Bulo beschreiben in ihrem Paper zur 9. *Advanced International Conference on Telecommunications* einen Ansatz, wie sie mithilfe von CEP *Denial of Service* (D.o.S.) Attacken in einem Netzwerk erkennen (Gad et al., 2013). Die Autoren haben dafür eine Event-Hierarchie modelliert, die Abbildung 2 veranschaulicht.

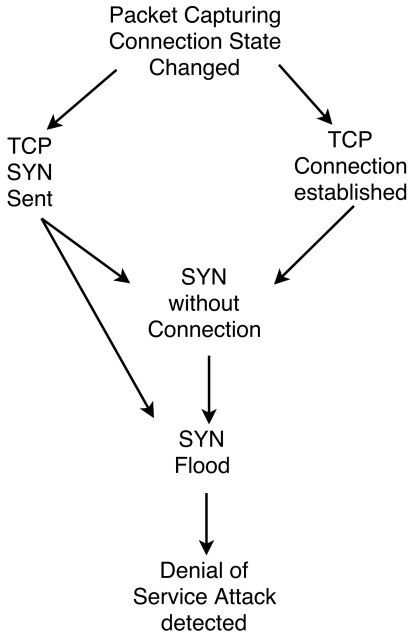

Abbildung 2: Event-Hierarchie zur Erkennung von Denial of Service Attacken (Gad et al., 2013).

Sie betrachten alle Datenpakete im Netzwerk als einen diskreten Datenstrom und führen auf ihm mittels der CEP-Engine Mustererkennung durch.

Der erste Schritt in der Event-Hierarchie besteht darin, alle Datenpakete zu filtern, nach jenen Paketen, die Informationen über eine bereits bestehende *Transmission Control Protocol* (TCP) Verbindung oder die Anfrage über den Verbindungsaufbau einer neuen TCP-Verbindung beinhalten. Diese zwei Ereignisse stellen in dem Modell die möglichen Typen von Simple Events dar.
Wenn auf eines dieser Simple Events keine gültige TCP-Verbindung folgt, leiten die Autoren das Complex Event *SYN without Connection* ab. Eine abgebrochene TCP-Verbindung allein ist noch kein Beweis für eine D.o.S Attacke. Tritt jedoch eine Flut, *SYN Flood* jenes komplexen Ereignisses auf, erkennen die Autoren eine solche Attacke. Ein entsprechender Event Consumer kann anschließend automatisiert auf diesen Angriff reagieren und zum Beispiel die IP-Adresse sperren.

2.2 Smart City

Smart Cities sind gekennzeichnet von einem hohen Grad an vernetzten, intelligenten IT-Systemen, die unstrukturierte und strukturierte Daten sowie Datenströme mit hohem Datenvolumen und geringer Speicherdauer erzeugen.
Moraru und Mladenic beschreiben einen Prototypen, mit dem sie am Beispiel von London Verkehrsereignisse mit kulturellen Veranstaltungen korrelieren und so Vorhersagen über das Entstehen von Verkehrsstörungen treffen (Moraru und Mladenic, 2012). Dafür haben sie Wetter- und Verkehrsdaten sowie Informationen über kulturelle Veranstaltungen gegen entsprechende Schnittstellen in regelmäßigen Intervallen abgefragt und so einen diskreten Datenstrom erzeugt.
Mithilfe des Machine Learning Toolkit *WEKA* der Universität von Waikato versuchten sie Muster in diesem Datenstrom zu erkennen und diese anschließend mit einer EQL zu modellieren, um zukünftige Datenströme entsprechend dieser Muster zu untersuchen.
Leider blieb das Paper ohne Ergebnis, da die Autoren keine statistisch signifikanten Zusammenhänge in den Daten fanden und so nur Vorhersagen unabhängig vom Verkehr treffen konnten.

2.3 Fraud Detection

Ein *Fraud* beschreibt Betrug, Unterschlagung oder durch Handlung von Mitarbeitern bedingte Vermögensverluste und *Fraud Detection* den automatisierten Vorgang, solche zu erkennen.
Im Umfeld eines Finanzdienstleisters sieht CEP fünf Ebenen der Verarbeitung von Datenströmen vor (Bass, 2006). Ebene 0 beschreibt die Vorverarbeitung des Datenstroms, welche die Daten normiert. Da das Datenvolumen pro Sekunde so hoch ist, dass die CEP-Engine nicht alle Simple Events auswerten kann, folgt in Ebene 1 das *Event Refinement*. In diesem Verarbeitungsschritt werden alle Simple Events markiert, die betrügerisch sein könnten, und anschließend entsprechend der Wahrscheinlichkeit einen Betrug zu enthalten in eine Rangfolge einsortiert. Die dritte Ebene das *Situation Refinement* bildet das eigentliche CEP ab. In dieser Phase werden die zuvor gefilterten Echtzeit Ereignisse auf entsprechende Muster untersucht. Wenn die CEP-Engine zum Beispiel das Simple Event *Login Success* durchläuft, ist dies noch lange keine betrügerische Aktivität. Falls dieses Simple Event zusammen mit einem Event vom Typ *Known Fraud IP* auftritt, leitet die CEP-Engine das Complex Event *Identity Theft* ab. Der Event-Consumer würde folglich automatisiert den Kunden benachrichtigen, dass jemand seine Identität gestohlen und sich in sein Benutzerkonto eingloggt hat.
Die letzten beiden Ebenen *Impact Assessment* und *Process Refinement* beschreiben, wie mittels CEP das Ausmaß des Fraud abgeschätzt und das Pattern-Matching der CEP-Engine mit den gewonnen Daten verbessert wird.

3 Event Query Languages

Dieser Abschnitt geht zunächst tiefer auf den Unterschied zwischen Simple und Complex Events ein, bevor es die algebraischen Strukturen untersucht, die Anfragen auf Datenströme abbilden. Anschließend zeigt er Beispiele der *Continuous Query Language* (CQL) als einen Vertreter für Anfragesprachen auf diskreten Datenströmen.

3.1 Eventbegriff

Die Ursache für ein Simple Event in einem ereignisgesteuerten IT-System liegt immer in der Änderung seines Zustands. Ein Simple Event beschreibt einen eindeutigen Zeitpunkt und keine Zeitdauer. Wohingegen Complex Events eine Menge von Simple Events umfassen, die einem zuvor modellierten Muster entsprechen. Deren Muster werden mithilfe einer EQL wie zum Beispiel CQL fixiert.
Die Datenströme in einem ereignisgesteuerten IT-System sind zwar nicht unendlich, reichen aber theoretisch bis zum Start des Systems zurück. Nun kann dieses System nicht alle Ereignisse persistent speichern und die CEP-Engine kann nur einen kleinen Ausschnitt verarbeiten, deshalb terminiert die CEP-Engine ein Fenster.

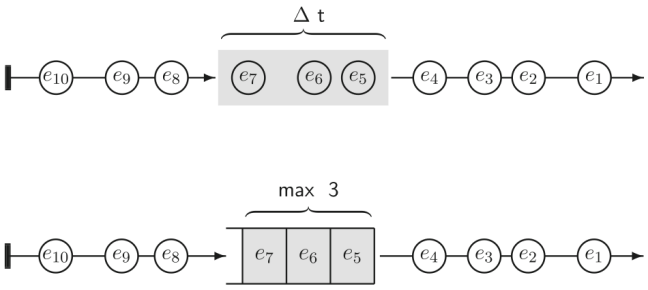

Abbildung 3: Gleitendes Zeit- und Längenfenster (Hedtstück, 2017).

Abbildung 3 zeigt die zwei unterschiedlichen Arten von Fenstern, die einen Ausschnitt des Datenstroms markieren. Das Zeitfenster fixiert eine feste Zeitspanne des Datenstroms, welche die CEP-Engine entsprechend der Zeitstempel auswertet. Im Gegensatz zum Zeitfenster umschließt das Längenfenster eine feste Anzahl von Ereignissen des Datenstroms.

3.2 Event-Algebra

Ausgangspunkt für die Event-Algebra ist, dass alle Events innerhalb des CEP-Systems durch eine Menge von Paaren, bestehend aus Ereignistyp und Zeitstempel beschrieben sind. Dem folgend ergeben sich für die Anfragen folgende Kernoperationen: Sequenz, Konjunktion, Disjunktion und Negation (Hedtstück, 2017).
Die zeitbezogene Sequenz untersucht die Reihenfolgen von Ereignistypen entsprechend einer Ordnungsrelation. Diese Ordnungsrelation ist für Simple Events gegeben durch die Kleinergleichrelation auf ihren Zeitstempeln. Diese Kernoperation ist so in der Lage Kausalitäten von Ereignissen zu modellieren.
Mit den Kernoperationen Disjunktion und Konjunktion kann eine Anfrage einen Datenstrom auf jene Folge von Ereignissen filtern die entweder ein Ereignisse oder alle Ereignisse der gewünschten Ereignistypen enthalten.
Ferner formuliert die Negation, dass in einem Datenstrom Ausschnitt ein Ereignis eines bestimmten Ereignistypen nicht vorkommen darf. Dieses Vorgehen beschreibt den fensterbasierten Ansatz. Ihm gegenüber steht der sequenzbasierte Ansatz, dieser fixiert, dass die Instanz eines Ereignistyps nicht innerhalb einer bestimmen Sequenz auftauchen darf.
Eine Mischform dieser beiden Ansätze sieht vor, dass ein Ereignis von Typ A nicht eintreten darf, bevor ein Ereignis von Typ B eingetreten ist.

3.3 Continuous Query Language

Die CQL ist eine deklarative Anfragesprache für diskrete Datenströme, sie modelliert Muster von Simple Events für das Pattern-Matching durch die CEP-Engine. CQL wurde im Rahmen des *STREAM* Projekts an der Stanford Universität entwickelt, ist jedoch keine standardisierte Anfragesprache (The STREAM Group, 2003).
Diese Anfragesprache stellt eine Erweiterung von der *Structured Query Language* (SQL) dar und setzt voraus, dass für jeden möglichen Ereignistyp eine entsprechende Relation vorliegt. Diese Relation speichert den Zeitstempel des Simple Events, sowie dessen Metadaten.
Das folgende Listing zeigt, wie mittels CQL eine Anfrage basierend auf einem Zeitfenster beschrieben wird.

```
SELECT Symbol, AVG(Price)
FROM StockTicketEvent
     [RANGE 60 SEC SLIDE 10 SEC]
WHERE Symbol = 'CSCO'
```

Diese Anfrage fixiert das Attribut Symbol und den durchschnittlichen Preis aller Ereignisse vom Typ *StockTicketEvent* bei denen das Attribut Symbol *CSCO* entspricht. Dabei terminiert diese Anfrage mit dem Schlüsselwort *Range* ein Zeitfenster, das

Ereignisse der letzten 60 Sekunden umfasst und das die CEP-Engine in jedem Schritt um zehn Sekunden weiterschiebt.
Die folgende Anfrage illustriert, wie CQL ein Längenfester mit dem Schlüsselwort *ROWS* beschreibt. Deren Längenfenster umspannt die letzten zehn Ereignisse bei denen das Symbol *CSCO* gesetzt ist, wobei die CEP-Engine dieses Fenster in jedem Verarbeitungsschritt um fünf Ereignisse versetzt.

```
SELECT Symbol, AVG(Price)
FROM StockTicketEvent
     [ROWS 10 SLIDE 5]
WHERE Symbol = 'CSCO'
```

4 Technologien

Auf dem Markt gibt es eine Vielzahl von verschiedenen Technologien, die für eine Anwendung ein CEP-System bereitstellen. Die Mehrheit dieser System ist aus universitären Projekten entstanden, wie das STREAM Projekt der Stanford Universität, die kaum eine Anwendung finden. Am Mark haben sich hauptsächlich das properitäre System von Oracle *Oracle Event Processing* und das open source Framework der Apache Foundation *Apache Storm* durchgesetzt.

4.1 Oracle Event Processing

Oracle's Event Processing bietet mit *Oracle CQL* eine Möglichkeit Anfragen zu formulieren, die sich stark an der CQL des Stanfort Projekts orientieren. Weiterhin stellt es eine *JDeveloper* Integration bereit, mit der der Benutzer sein EPN grafisch modellieren kann und eine eigene Plattform *Oracle Stream Explorer*, die Muster in Datenströmen sucht.
Trotz des reichen Funktionsumfangs von Oracle Event Processing hat sich Apache Storm als Quasi-Standard etabliert. Deshalb führt dieses Kapitel es nur der Vollständigkeit halber auf und untersucht Apache Storm im Detail.

4.2 Apache Storm

Apache Storm ist wie Oracle Event Processing eine verteilte Anwendung, die ein CEP-System implementiert. Dafür führt es eine Reihe von softwaretechnischen Abstraktionen ein. Ein *Tuple* beschreibt die zentrale Datenstruktur in diesem System, welches ein Ereignis programmiert abbildet. Ein *Stream* ist eine ungeordnete Folge von Tuple und abstrahiert somit einen Datenstrom. *Spouts* und *Bolts* stellen zum einen die Quellen eines Streams und zum anderen die logischen Datenverarbeitungseinheiten dar.
Der größte Nachteil von Apache Storm gegenüber Oracle Event Processing besteht darin, dass jenes keine Anfragesprache bereitstellt. Alle Anfragen und Operationen auf dem Datenstrom müssen innerhalb eines Bolts programmiert abgebildet werden, was zwar deutlich aufwendiger ist, dem Entwickler aber mehr Kontrolle gibt. Abbildung 4 veranschaulicht die Architektur eines Apache Storm Clusters, welches das verteilte Apache Strom ausführt.

4.2.1 Cluster

Im Mittelpunkt eines jeden Storm Clusters steht der *Nimbus*, dieser stellt den Master-Node dar und führt die programmiert beschriebene Topologie, die aus Spouts und Bolts besteht, aus. Ihm gegenüber stehen ein oder mehrere *Supervisors*, welche die Anweisungen des Nimbus ausführen und *Worker Processes* starten. Dabei kapselt ein Worker Process mehrere *Executor*, die jeweils für die Verwaltung von *Tasks* zuständig sind. Ein Task führt innerhalb des Storm Clusters die eigentliche Datenverarbeitung aus.

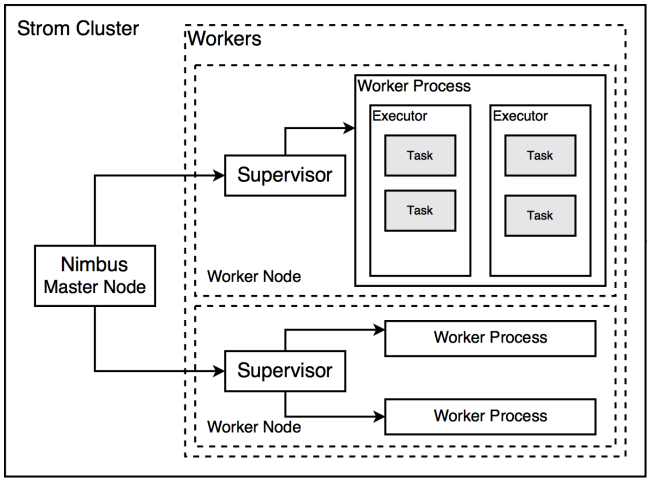

Abbildung 4: Aufbau eines Apache Storm Clusters (tutorialspoint, 2017).

Über eine Instanz des *ZooKeeper* Frameworks tau-

schen die einzelnen Knoten im Storm Cluster Daten untereinander aus.

4.2.2 Anwendungsbeispiel

Apache Storm stellt dem Entwickler zwei Interfaces *ISpout* und *IBolt* bereit, welche die softwaretechnischen Abstraktionen von Bolt und Spout darstellen. Die Methode *nextTuple()* des Spout-Interfaces beschreibt die zentrale Schnittstelle zwischen Quellen und den Datenverarbeitungseinheiten. Diese Methode wird vom Bolt aufgerufen sobald ein neues Tupel eintrifft. Wohingegen die Methode *execute()* des Bolt-Interfaces ein Tupel des Datenstroms verarbeitet.
Zum Beispiel kann ein Netzbetreiber Apache Storm einsetzen, um mittels CEP seine Verbindungsdaten auszuwerten. Dieser implementiert einen Spout so, dass der Spout bei dem Aufruf von nextTuple() eine Telefonverbindung an einen Bolt weiterleitet (tutorialspoint, 2017). Eine benutzerdefinierte Bolt-Implementierung zählt die Verbindungsdaten, welche durch ihren Sender und Empfänger eindeutig bestimmt sind.
Apache Storm's *TopologyBuilder* setzt jene Implementierungen zu einer Topologie zusammen. Diese Topologie kann der Netzbetreiber anschließend in einem Storm Cluster ausführen und in seinem Echtzeit Datenstrom die Anzahl aller Telefonverbindungen zählen. Das Listing 1 im Anhang zeigt, wie mit dem TopologyBuilder Instanzen von ISpout und IBolt zu einer ausführbaren Topologie zusammengefügt werden.

5 Fazit

Abschließend bleib abzuwarten, ob Complex Event Processing ein Trend ist, der sich durch setzen wird. Sicher ist zum heutigen Zeitpunkt, dass seine Technologien, Methoden und Werkzeuge ein großes Potential bieten, Kosten zu senken und Prozesse zu überwachen.
Es besteht in diesem Zusammenhang immer die Frage, welche Daten ein CEP-System speichert und auswertet. Wenn es sich um personenbezogene Daten handelt überwiegt der Schutz der Persönlichkeitsrechte der Prozessoptimierung.
Die in diesem Paper vorgestellten Technologien richten sich an kleine und mittelständische Unternehmen. Großkonzerne, deren ereignisgesteuerten IT-Systeme einen bestimmten Grad an Komplexität erreicht haben, implementieren sicherlich eigene und auf ihre Anforderungen zugeschnittene Lösungen.

Literaturverzeichnis

Tim Bass. *Fraud Detection and Complex Event Processing for Predictive Business*. CISSP, TIBCO Software Inc, 2006.

Ruediger Gad, Martin Kappes, Juan Boubeta-Puig, und Inmaculada Medina-Bulo. *Employing the CEP Paradigm for Network Analysis and Surveillance*, The Ninth Advanced International Conference on Telecommunications, 2013.

The STREAM Group. *stanfordstreamdatamanager*, Stanford University. URL `http://infolab.stanford.edu/stream/`.

Ulrich Hedtstück. *Complex Event Processing - Verarbeitung von Ereignismustern in Datenstromen*. Springer Vieweg, 2017.

David Luckham. *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems*. Addison-Wesley Professional, Birmingham, 2002.

Alexandra Moraru und Dunja Mladenic. *COMPLEX EVENT PROCESSING AND DATA MINING FOR SMART CITIES*, Artificial Intelligence Laboratory Jozef Stefan Institute, Slovenia, 2012.

tutorialspoint. tutorialspoint simple and easy learning: Apache Storm Tutorial. URL `https://www.tutorialspoint.com/apache_storm/`.

Anhang

```
public class CallAnalyser {
  public static void main(String[] args){

    //Erzeuge eine Konfiguration für Storm Cluster
    Config config = new Config();
    config.setDebug(true);

    //Füge der Topologie Quelle des Datenstroms hinzu
    TopologyBuilder builder = new TopologyBuilder();
    builder.setSpout("call-log-spout", new CallLogSpout());

    //Füge der Topologie Datenverarbeitungseinheit hinzu
    builder.setBolt("call-counter-bolt", new CallCounterBolt())
      .fieldsGrouping("call-log-creator-bolt", new Fields("call"));

    LocalCluster cluster = new LocalCluster();
    cluster.submitTopology("CallAnalyser", config, builder.createTopology());

    //Stopp die Topologie
    cluster.shutdown();
  }
}
```

Listing 1: Aufbau einer Apache Storm Topologie mit TopologyBuilder nach (tutorialspoint, 2017)