

Institut für Parallele und Verteilte Systeme

Universität Stuttgart
Universitätsstraße 38
D-70569 Stuttgart

Bachelorarbeit

Scalability Investigation of Event Processing Systems

Alexander Stifel

Studiengang: Softwaretechnik

Prüfer/in: Prof. Dr. rer. nat. Dr. h. c. Kurt Rothermel

Betreuer/in: M. Sc. Ahmad Slo

Beginn am: 19. März 2018

Beendet am: 19. September 2018

Kurzfassung

Das frühestmögliche Erkennen von nützlichen Informationen in Datenströmen und die damit zeitnah mögliche Reaktion auf sich ändernde Tendenzen spielt in vielen Anwendungsgebieten eine strategische Rolle. Das enorm zunehmende Datenvolumen erschwert jedoch das Ableiten von Zusammenhängen der Daten mit Hilfe von Complex-Event-Processing-Systemen in Echtzeit.

Für die effiziente Verarbeitung dieser Datenströme bieten aktuelle Complex-Event-Processing-Systeme unterschiedliche Parallelisierungstechniken an. Eine Möglichkeit, um die Leistungsfähigkeit zu erhöhen, ist die verteilte Ausführung auf einem Rechnerverbund in dem weitere Knoten für die Verarbeitung zur Verfügung gestellt werden und das System den Datenstrom auf die verfügbaren Knoten verteilen kann.

In dieser Arbeit wird die Auswirkung des Parallelisierungsgrads auf den Durchsatz von Complex-Event-Processing-Systemen untersucht. Insbesondere wird das eigenentwickelte Complex-Event-Processing-System mit einem auf dem Markt verfügbaren System verglichen. Für die Untersuchung werden für Ereignisverarbeitungssysteme geeignete Datensätze recherchiert und basierend darauf verschiedene Anwendungsszenarien entwickelt. Die erarbeiteten Anwendungsszenarien werden für die beiden Systeme implementiert und schließlich wird die Auswirkung des Parallelisierungsgrads experimentell überprüft und ausgewertet.

Inhaltsverzeichnis

1. Einleitung	15
2. Einführung in das Complex-Event-Processing	17
2.1. Ereignis und Ereignisstrom	17
2.2. Komplexe Ereignisse	17
2.3. Complex-Event-Processing	18
2.4. Aufbau von Complex-Event-Processing-Systemen	18
2.5. Datenparallelisierung	20
2.6. Fenster	21
3. Bestehende Arbeiten	23
4. Complex-Event-Processing-Systeme	25
4.1. DCEP	25
4.2. Existierende Complex-Event-Processing-Systeme	27
4.3. Gegenüberstellung	29
4.4. Auswahl	29
5. Datensätze und Anwendungsszenarien	31
5.1. Sensordaten am Beispiel eines Fußballspiels	31
5.2. Öffentliche Aktivitätsdaten der Entwicklerplattformen GitHub	33
5.3. Börsenkurse von Aktiengesellschaften	35
5.4. Nummernschilderkennung am Beispiel von Fahrzeugbildern	36
6. Ergebnisse und Auswertung	39
6.1. Experiment	39
6.2. Durchführung der Experimente	40
6.3. Auswertung der Experimentreihen	45
7. Zusammenfassung und Ausblick	47
Literaturverzeichnis	49
A. Anhang	51
A.1. Installationshinweis	51
A.2. Google-Suchen	51
A.3. Skripte zum Generieren und Bereinigen der Datensätze	51

Abbildungsverzeichnis

2.1.	Betrachtung von Ereignissen auf unterschiedlichen Abstraktionsebenen [BD15]. . .	18
2.2.	Zyklische Verarbeitung von Ereignissen in CEP-Systemen - Nach Balkesen [BDWT13].	19
2.3.	Ablaufbasierte Parallelisierung eines Operators - Nach Balkesen [BDWT13]. . .	20
2.4.	Zeitfenster mit einem festen Zeitraum ohne Verschiebung	21
2.5.	Zeitfenster mit einem festen Zeitraum mit Verschiebung	22
4.1.	Parallelisierungs- und Datenflussmodell von DCEP.	25
4.2.	Datenfluss- und Programmiermodell von Apache Flink [Fli18a].	28
5.1.	Schematische Darstellung des Spielfeldes mit Koordinaten.	31
5.2.	Häufigkeit der unterschiedlichen Ereignisarten.	34
5.3.	Auswertung des Datensatzes auf Basis von Benutzername und Repository. . .	35
5.4.	Exemplarische Fahrzeugbilder für die Kennzeichenerkennung [Mar17].	37
6.1.	Durchsatzanalyse für den Datensatz 5.1 für die Systeme DCEP und Apache Flink.	41
6.2.	Durchsatzanalyse für den Datensatz 5.2	42
6.3.	Durchsatzanalyse für den Datensatz 5.3 für die Systeme DCEP und Apache Flink.	42
6.4.	Durchsatzanalyse für den Datensatz 5.4 für die Systeme DCEP und Apache Flink.	43

Tabellenverzeichnis

4.1. Gegenüberstellung von CEP-Systemen	29
6.1. Systemspezifikationen des Rechnerverbundes	39

Verzeichnis der Listings

5.1. Datenschema der Sensordaten im Rohdatensatz	32
5.2. Grundstruktur von allen Ereignissen der GitHub Plattform [Git18a].	33
5.3. Verkürzte Datenstruktur von GitHub Ereignissen.	33
5.4. Datenschema von Kursdaten.	36
5.5. Datenschema von Verkehrsüberwachungsdaten.	37
6.1. Ordnerstruktur auf dem Rechnerverbund	40

Abkürzungsverzeichnis

CEP Complex-Event-Processing. 17

CEP-System Complex-Event-Processing-Systeme. 18

DBMS Datenbankmanagementsystem. 18

DCEP Distributed-Complex-Event-Processing. 25

DEBS Distributed Event-Based Systems. 31

EPL Esper and Event Processing Language. 28

IEX Investors Exchange. 35

1. Einleitung

Die heutige Welt ist geprägt durch die Allgegenwart und die zunehmende Anzahl von Sensoren sowie intelligenten Geräten, wobei ein Ende nicht in Sicht ist. Bereits heute existieren schon mehr intelligente Geräte als Menschen und es wird prognostiziert, dass im Jahre 2020 50 Milliarden dieser Geräte im Umlauf sein werden [Eva11]. Die enorme Zunahme des Datenvolumens spiegelt sich in vielen Anwendungsgebieten wieder, ob in der Digitalisierung der Industrie, in sozialen Netzwerken durch die steigende Anzahl von Nutzern oder im privaten Sektor durch die vermehrte Vernetzung von physischen Geräten mit dem Internet.

In allen Anwendungsgebieten existiert jedoch die Notwendigkeit diese Daten in Echtzeit zu verarbeiten um frühestmöglich nützliche Informationen zu extrahieren um somit Wettbewerbsvorteile zu schaffen. Besonders in der Industrie ist es von strategischer Bedeutung schnell auf sich ändernde Tendenzen zu reagieren, da es ein entscheidender Faktor für den Erfolg oder Misserfolg eines Unternehmens sein kann.

Ein zentrales Problem bei der Echtzeitverarbeitung von Datenströmen ist die Erkennung von Ereignismustern. Betrachtet man die einzelnen Ereignisse der Datenströme besteht in vielen Fällen kein fachlicher Zusammenhang. Erst die Kombination aus unterschiedlichen Ereignissen und deren zeitlicher Reihenfolge ermöglicht das Ableiten von komplexen Situationen.

Complex-Event-Processing adressiert genau dieses Problem und ist ein effektives Paradigma zur Verarbeitung und Erkennung von Mustern in kontinuierlich eingehenden Datenströmen. Um den Durchsatz (Ereignisse pro Sekunde) von CEP-Systemen zu erhöhen eignet sich ein verteilter und parallelisierter Ansatz bei dem ein System auf einem Rechnerverbund ausgeführt und der Datenstrom verteilt verarbeitet wird. Auf dem Markt existieren mehrere Complex-Event-Processing Systeme von unterschiedlichen Anbietern die sich in ihrer Funktionsweise und verteilten Ausführung unterscheiden. Um auf enorme Datenmengen effizient und in Echtzeit reagieren und diese verarbeiten zu können, muss das CEP-System mit dem Rechnerverbund horizontal skalieren. Durch das Hinzufügen von weiteren Knoten im Rechnerverbund kann der eingehende Ereignisstrom in Datenblöcke unterteilt werden und von mehreren Knoten verarbeitet werden, womit der Durchsatz des Systems erhöht werden kann. Diese Aufteilung des Ereignisstroms in Datenblöcke kann schlüsselbasiert [GRGH17] oder stapelbasiert [BDWT13] erfolgen.

In Rahmen dieser Arbeit wird die Auswirkung des Parallelisierungsgrads auf den Durchsatz von verteilten Complex-Event-Processing-Systemen untersucht. Insbesondere wird das eigenentwickelte Complex-Event-Processing-System mit einem auf dem Markt verfügbaren System verglichen. Dazu werden der Durchsatz und die Anzahl der Knoten im Rechnerverbund als Messwerte herangezogen. Für die Untersuchung werden für Ereignisverarbeitungssysteme geeignete Datensätze recherchiert und basierend darauf verschiedene Anwendungsszenarien entwickelt. Die erarbeiteten Anwendungsszenarien werden für die beiden Systeme implementiert und schließlich wird die Auswirkung des Parallelisierungsgrads experimentell überprüft und ausgewertet.

Die vorliegende Arbeit ist in folgender Weise gegliedert: In Kapitel 2 werden die verwendeten Grundbegriffe betreffend dieser Arbeit beschrieben. In Kapitel 3 werden bereits vorhandene Forschungsarbeiten in diesem Gebiet aufgezeigt. Zu Beginn wird in Kapitel 4 das eigenentwickelte CEP-System, das als Grundlage für die Arbeit dient, dargestellt. Im Anschluss daran werden auf dem Markt verfügbare Systeme vorgestellt und miteinander verglichen. Im darauffolgenden Kapitel 5 werden für CEP-Systeme geeignete Datensätze recherchiert und passende Anwendungsszenarien erarbeitet. In Kapitel 6 werden alle relevanten Informationen für das Experiment beschrieben und die Durchführung und Auswertung des Experiments aufgezeigt. Schließlich werden in Kapitel 7 die Ergebnisse der Arbeit zusammengefasst und ein Ausblick auf weitere Forschungen gegeben.

2. Einführung in das Complex-Event-Processing

Im ersten Abschnitt werden einige Grundbegriffe im Zusammenhang mit Complex-Event-Processing (CEP) erläutert, um einen Einblick in die Thematik zu erhalten.

2.1. Ereignis und Ereignisstrom

Allgemein bezieht sich ein Ereignis (engl. event) auf eine konkrete Veränderung in der realen oder virtuellen Welt. Ereignisse enthalten für die Einordnung und Weiterverarbeitung Metadaten und Informationen zum Ereigniskontext [BD15]. In den Metadaten befindet sich meist Informationen wie Ereignistyp, Ereignisquelle sowie ein Zeitstempel der den Zeitpunkt des Auftretens repräsentiert. Der Ereigniskontext beinhaltet die eingetretene reale oder virtuelle Veränderung wie zum Beispiel die gemessene Temperatur eines Sensors oder eine Interaktion in sozialen Netzwerken.

Ereignisse können potenziell kontinuierlich erzeugt werden, so erzeugt beispielhaft ein Temperatursensor fortlaufend und in regelmäßigen Zeitabständen einen Messwert, was zur einer unendlichen Folge von Ereignissen führen kann. Diese Folge ist eine linear geordnete Sequenz von kontinuierlich eintreffenden Ereignissen und wird als Ereignisstrom (engl. event stream) bezeichnet [Luc02].

2.2. Komplexe Ereignisse

Komplexe Ereignisse werden von unterschiedlichen Ereignissen aus dem Ereignisstrom abgeleitet und repräsentieren ein erkanntes Muster im Ereignisstrom. Einfache Muster können einzelne Ereignisse oder boolesche Kombinationen im Ereignisstrom sein, jedoch können auch komplexere Muster mit zusätzlichen Abhängigkeiten geprüft werden [BD15].

Die Abbildung 2.2 visualisiert den Abstraktionsgrad von Ereignissen und komplexen Ereignissen und zeigt, dass komplexe Ereignisse sich auf einer höheren Abstraktionsebene befinden und eine fachliche Aussagekraft besitzen.

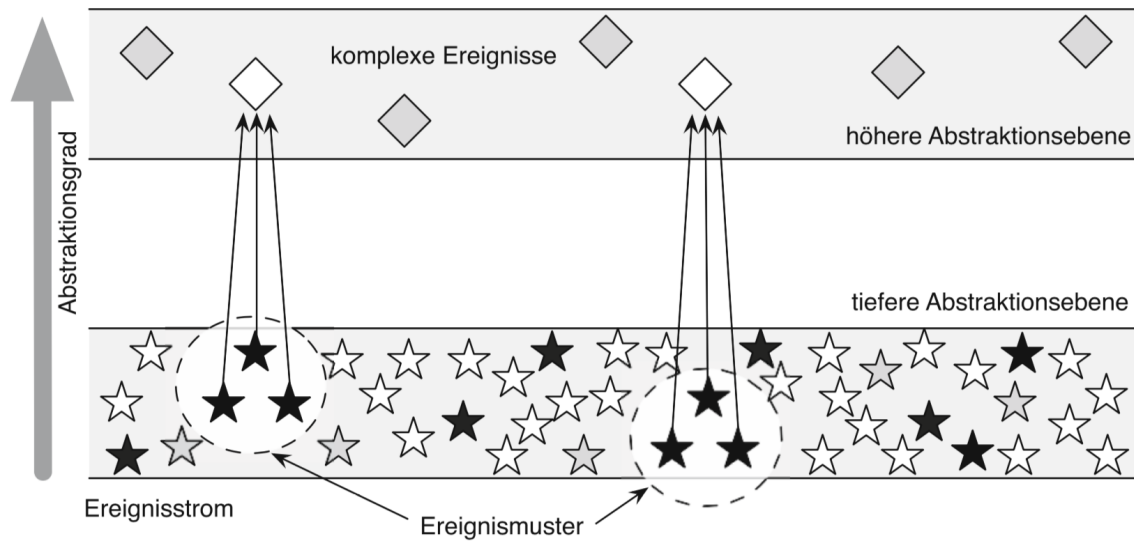


Abbildung 2.1.: Betrachtung von Ereignissen auf unterschiedlichen Abstraktionsebenen [BD15].

2.3. Complex-Event-Processing

Der Begriff CEP wurde erstmal von David Luckham erwähnt und umfasst eine Menge von Techniken, Methoden und Werkzeugen mit denen Muster modelliert und auf Datenströme angewandt werden können [Luc02]. Im Gegensatz zu herkömmlichen Datenbankmanagementsystem (DBMS), bei denen eine Abfrage auf gespeicherten Daten ausgeführt wird, werden die Daten auf gespeicherten Abfrage ausgeführt. CEP fokussiert sich auf das Auswerten, Steuern und Verarbeiten von Datenströmen und speichert den Ereignisstrom nicht dauerhaft ab.

2.4. Aufbau von Complex-Event-Processing-Systemen

Complex-Event-Processing-Systeme (CEP-System) sind speziell auf die Verarbeitung von Ereignissen ausgerichtet und führen zyklisch drei Ausführungsschritte durch [BDWT13]. Im ersten Schritt erzeugen unterschiedliche Quellen Ereignisse und senden diese an das System. Die empfangenen Daten werden dann im nächsten Schritt in Operatoren verarbeitet. Hierzu werden die zuerkennenden Muster modelliert und als Abfrage hinterlegt. Die Operatoren prüfen die eintreffenden Ereignisse auf die beschriebenen Muster und leiten gefundene Muster an die Ereignisbehandlung weiter. Bei der Ereignisbehandlung werden die entdeckten komplexen Ereignisse für weitere Anwendungen bereitgestellt. Die zyklische Verarbeitung von Ereignissen in CEP-Systemen wird in Abbildung 2.2 dargestellt.

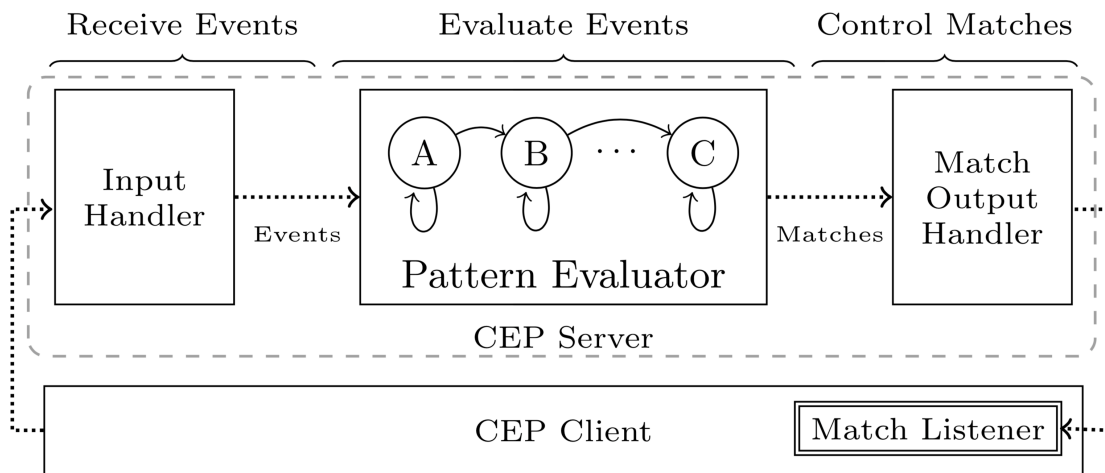


Abbildung 2.2.: Zyklische Verarbeitung von Ereignissen in CEP-Systemen - Nach Balkesen [BDWT13].

2.4.1. Operatorgraph

CEP-Systeme können mehrere miteinander verknüpfte Operatoren für die Mustererkennung beinhalten. Die Verbindung und Anordnung der einzelnen Operatoren können durch einen gerichteten azyklischen Graphen (Operatorgraph) dargestellt werden.

2.5. Datenparallelisierung

Die einfachste Form von CEP-Systemen baut auf einem zentralisierten Ansatz auf und schränkt somit die Skalierbarkeit des Systems ein. Ein zentralisierter Ansatz kann jedoch bei folgenden Situationen zu Problemen in der Leistung führen:

1. Behandlung einer großen Anzahl von Abfragen.
2. Behandlung einer großen Anzahl von Ereignissen
3. Komplexe Abfragen die enorme Ressourcen benötigen.

Um diese Probleme zu bewältigen eignet sich die Datenparallelisierung als Parallelisierungstechnik [MTR+17]. Hierzu werden die Operatoren aus dem Operatorgraphen auf mehrere identische Kopien, sogenannte Operator-Instanzen, repliziert. Der Ereignisstrom wird dann auf die einzelnen Instanzen verteilt und kann somit getrennt voneinander auf Muster untersucht werden. Abbildung 2.3 veranschaulicht den Ansatz der Datenparallelisierung, durch die Replezierung von Operatoren, in CEP-Systemen schematisch. Dieser verteilter Ansatz eignet sich für die Ausführung auf einem Rechnerverbund und ermöglicht eine horizontale Skalierung.

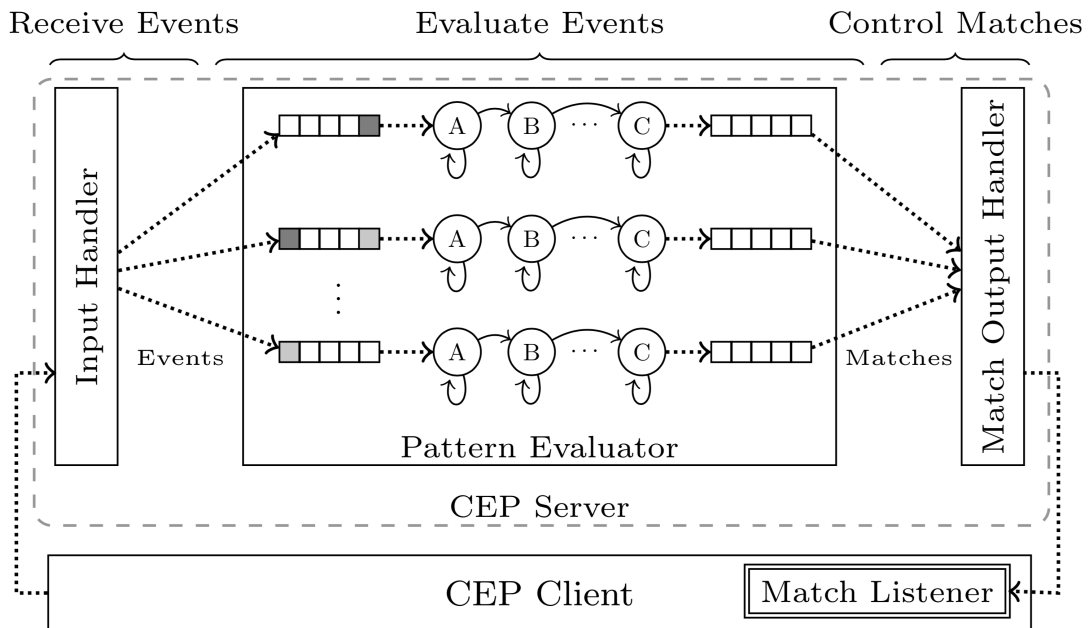


Abbildung 2.3.: Ablaufbasierte Parallelisierung eines Operators - Nach Balkesen [BDWT13].

2.6. Fenster

Wie bereits in Abschnitt 2.1 erwähnt, kann ein Ereignisstrom aus einer unendlichen Folge aus Daten bestehen. Um jedoch die Mustererkennung effizient und zeitnah in den Operatoren durchzuführen, wird nur ein Ausschnitt der Daten berücksichtigt. CEP bezeichnet diese generierten Ausschnitte als Fenster und beschreibt zusätzlich unterschiedliche Ansätze zur Aufteilung.

Die zwei gängigsten Arten von Fenstern, die den Ereignisstrom einschränken, sind Längen- und Zeitfenster, die eintreffende Ereignisse werden nach dem FIFO-Prinzip (First In First Out) verarbeitet. Bei *Längenfenstern* wird nur eine begrenzte Anzahl von Ereignissen berücksichtigt. *Zeitfenster* berücksichtigen hingegen alle Elemente die in einem definierten Zeitraum auftreten.

Da die Operatoren bei der Mustererkennung nur die im Fenster verfügbaren Elemente verwenden, können Muster die sich in zwei aufeinanderfolgenden Fenstern befinden nicht entdeckt werden. Um den Suchraum zu erweitern, können mehrere Fenster mit einem Verschiebungsfaktor erzeugt werden. Hierbei ist zu beachten, dass sich Ereignisse in unterschiedlichen Fenstern befinden können.

Die Abbildungen 2.4 und 2.5 visualisieren welche Ereignisse in Fenstern aufgenommen werden und zeigen, dass sich durch die Verschiebung (*window slide*), Fenster überlappen und sich somit Ereignisse in unterschiedlichen Fenstern befinden.

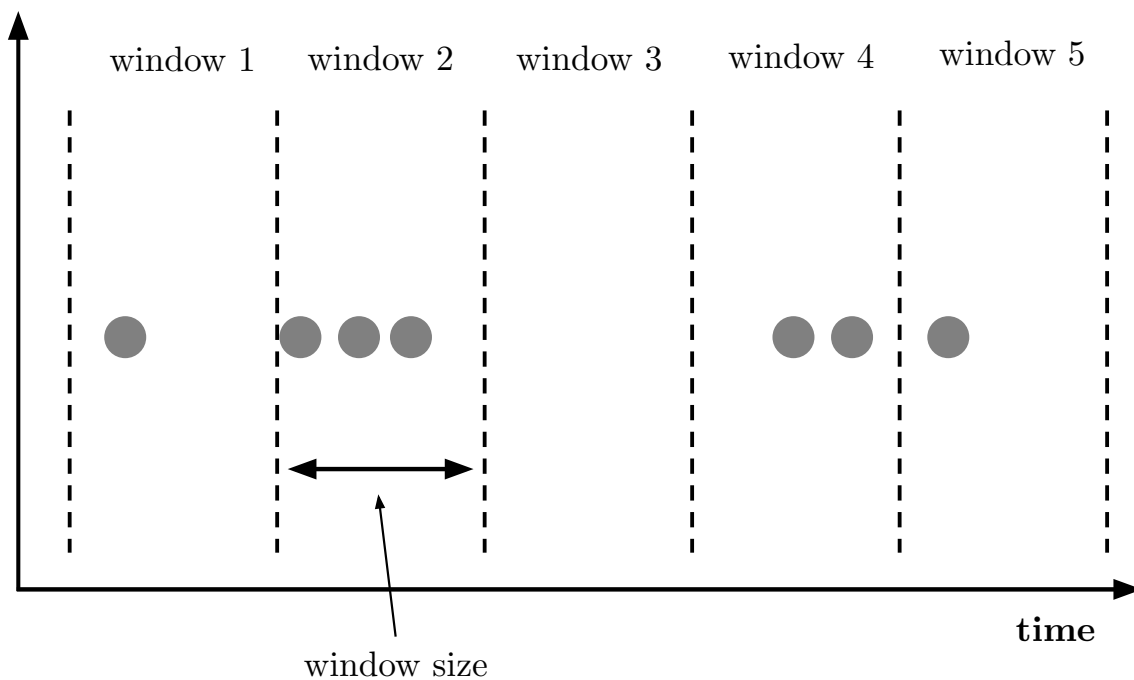


Abbildung 2.4.: Zeitfenster mit einem festen Zeitraum ohne Verschiebung

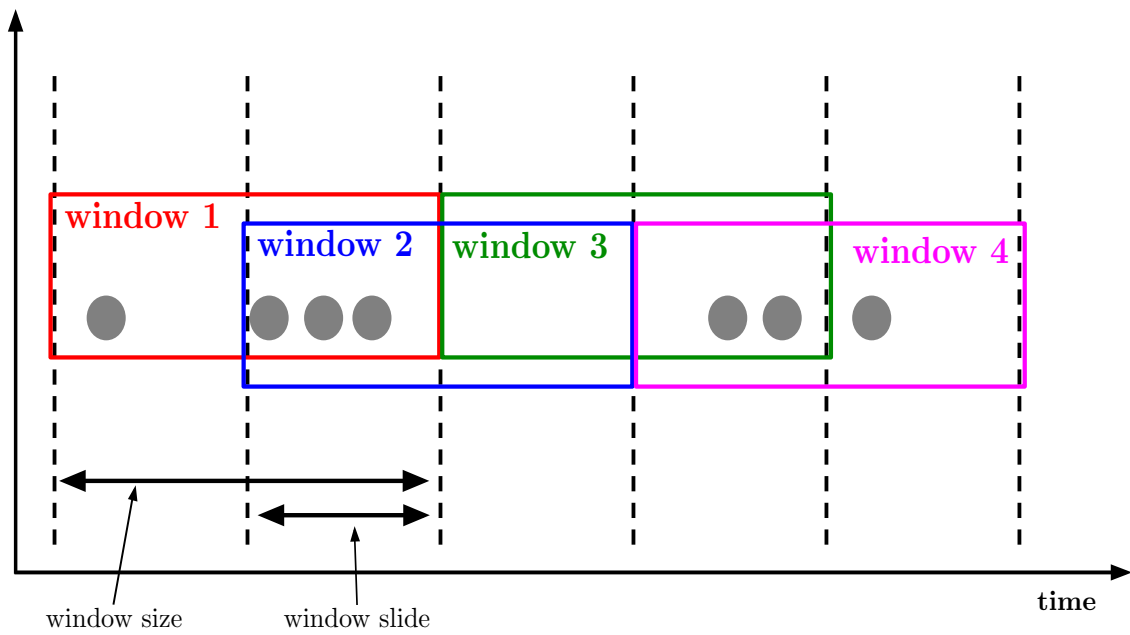


Abbildung 2.5.: Zeitfenster mit einem festen Zeitraum mit Verschiebung

3. Bestehende Arbeiten

Nach der erstmaligen Erwähnung von David Luckham wurden im Kontext zu CEP mehrere wissenschaftliche Arbeiten veröffentlicht, wodurch ein Forschungsgebiet entstanden ist. Untersuchungen bezüglich der Performanz und dem horizontalen Skalierungsverhalten sind in der wissenschaftlichen Literatur jedoch weitgehend unerforscht, da im derzeit größten Markt für CEP (Algorithmic Trading) große Geheimhaltung herrscht. [Inf18].

Es existieren mehrere Veröffentlichungen, bei denen der Fokus der Forschungen auf unterschiedliche Teilgebiete beschränkt ist. Die Auswirkungen von unterschiedliche Parallelisierungstechniken auf den Durchsatz wurden bereits untersucht [BDWT13]. Die Untersuchungen beschränkten sich jedoch auf einen synthetischen Datensatz und auf ein selbstentwickeltes System. Zudem wurde die Skalierbarkeit von auf dem Markt verfügbaren Systemen untersucht [LLD16] [GRGH17] wobei hier die Mustererkennung nicht berücksichtigt wurde. Darüber hinaus konnte bestätigt werden, dass der Durchsatz stark von der modellierten Anfrage beeinflusst wird [MBM09]. Bisher wurde in keiner der veröffentlichten Arbeiten die horizontale Skalierbarkeit von mehreren Systemen mit Hilfe von realen Datensätzen untersucht.

In dieser Arbeit hingegen wird die Auswirkung des Parallelisierungsgrads auf den Durchsatz bei der Mustererkennung auf Basis von realen Datensätzen untersucht. Hierzu wurden unterschiedliche Anwendungsszenarien entwickelt, diese in die jeweiligen Systeme integriert und experimentell überprüft und ausgewertet.

4. Complex-Event-Processing-Systeme

In diesem Kapitel wird zunächst das selbstimplementierte CEP-System genauer beschrieben. Anschließend folgt eine detaillierte Beschreibung von bestehenden Lösungen und darauf aufbauend wird eine Vergleichstabelle mit den wichtigsten Eigenschaften vorgestellt. Der Begriff DCEP wird in dieser Arbeit als Synonym für das selbstentwickelte CEP-System verwendet.

4.1. DCEP

Distributed-Complex-Event-Processing (DCEP) wurde vom Institut für Parallele und Verteilte Systeme der Universität Stuttgart entwickelt und ist ein generisches CEP-System, das die Datenparallelisierung als Parallelisierungstechnik verwendet. Das System ist in Java implementiert und besteht aus den vier Teilprojekten *Source*, *Splitter*, *Instance* und *Merger*.

Das Source Projekt ist für die Generierung und Weiterleitung von Ereignissen zuständig indem es Ereignisse aus Dateien ausliest und mit einer definierten Frequenz an den Splitter sendet. Der Splitter empfängt diese Ereignisse und verteilt geöffnete Fenster an die Operatoren. Operator-Instanzen prüfen bei eingehenden Ereignissen ob eine komplexe Situation vorliegt und senden im Fall der Erkennung ein komplexes Ereignis an den Merger. Die Mustererkennung in den Operatoren wird programmatisch mit Java umgesetzt. Der Merger dient als zentrale Instanz in der alle erkannten komplexen Ereignisse zusammenfließen und für die weitere Verarbeitung bereitstehen. Abbildung 4.1 veranschaulicht die grundlegende Funktionsweise des Systems und zeigt zusätzlich das Datenflussmodell.

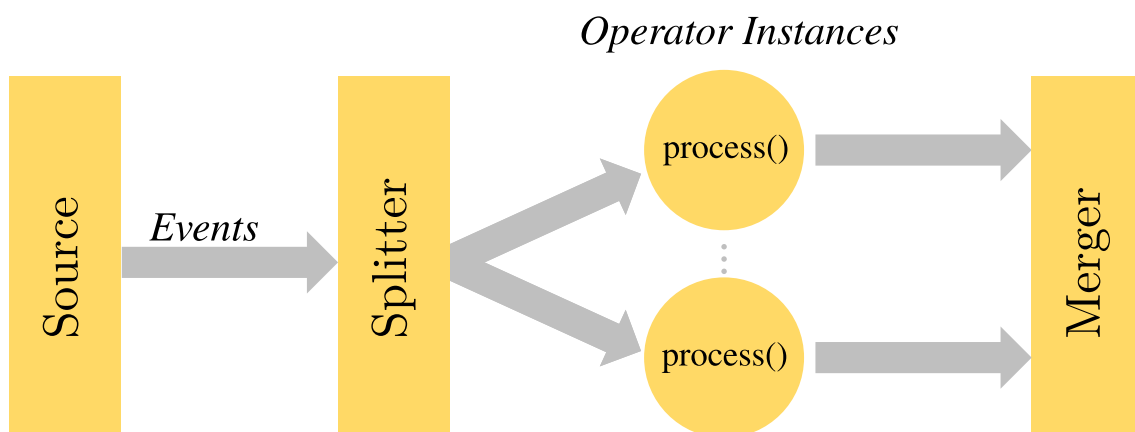


Abbildung 4.1.: Parallelisierungs- und Datenflussmodell von DCEP.

4. Complex-Event-Processing-Systeme

Alle Teilprojekte können auf unterschiedlichen Knoten ausgeführt werden. Um den Parallelisierungsgrad zu erhöhen kann die Anzahl der Operator-Instanzen erhöht werden. Der Splitter erkennt hinzugefügte Operatoren und berücksichtigt diese bei der Verteilung des Ereignisstroms.

4.2. Existierende Complex-Event-Processing-Systeme

Mittlerweile existiert ein breites Spektrum von kommerziellen und öffentlichen CEP-Systemen, die aus der akademischen oder kommerziellen Forschung entstanden sind [Dor17]. Das Grundkonzept aller Systeme ähnelt sich sehr und unterscheidet sich meist nur in wenigen Punkten.

Auswahlkriterien

Für die Auswahl geeigneter Complex-Processing-Systeme werden fünf Kriterien angewandt. (1) Das System verfügt über die grundlegenden Funktionen um Ereignisse zu generieren, zu verarbeiten und komplexe Ereignisse abzuleiten. (2) Das System besitzt eine umfassende Dokumentation, die es ermöglicht die Funktionweise nachzuvollziehen. (3) Das System ermöglicht es die Erkennung von komplexen Ereignissen programmatisch umzusetzen. (4) Das System unterstützt eine parallelisierte Ausführung auf einem Rechnerverbund. (5) Das System sollte einen für den Markt ausreichenden Softwarestand besitzen.

Mir ist bewusst, dass diese Kriterien sehr vage sind und der Auswahlprozess stark abhängig von der individuellen Erfahrung sowie der subjektiven Beurteilung ist. Aus diesem Grund werden die wichtigsten Auswahlkriterien so transparent wie möglich dargestellt.

- (1) Funktionalität.
- (2) Dokumentation.
- (3) Erkennung von komplexen Ereignissen.
- (4) Ausführung auf einem Rechnerverbund.
- (5) Entwicklungsstadium.

Um existierende CEP-Systemen zu finden wurde eine Google-Suche, welche im Anhang (siehe A.2.1) aufgeführt ist, durchgeführt. Nach Anwendung der Auswahlkriterien werden im folgenden Abschnitt fünf Systeme detailliert vorgestellt.

Apache Flink

Apache Flink ist eine Plattform für die Stream- und Batch-Datenverarbeitung [Fli18a], die für eine parallele und verteilte Ausführung auf einem Rechnerverbund konzipiert wurde. Der Quellcode ist frei zugänglich und wird aktuell in der Version 1.6 unter der Apache Lizenz 2 vertrieben. Apache Flink bietet dem Entwickler zwei Möglichkeiten, um komplexe Ereignisse in Ereignisströmen zu erkennen, an. Erstens können mit Hilfe der bereitgestellten CEP-Bibliothek Muster auf einer höheren Abstraktionsebene beschrieben und gefunden werden [Fli18b]. Die zweite Möglichkeit ist eine rein programmatische Beschreibung mit Java oder Scala. Die Ausführung auf einem Rechnerverbund und die Funktionalität von Apache Flink sind vollständig dokumentiert. Für eine parallele Ausführung benötigt Apache Flink einen schlüsselbasierten Datenstrom [Fli18a].

Abbildung 4.2 bildet das Datenflussmodell innerhalb Apache Flink ab, wobei die Source Operatoren für das Erzeugung von Ereignissen und die Transformation Operatoren für das Finden von komplexen Ereignissen, zuständig sind.

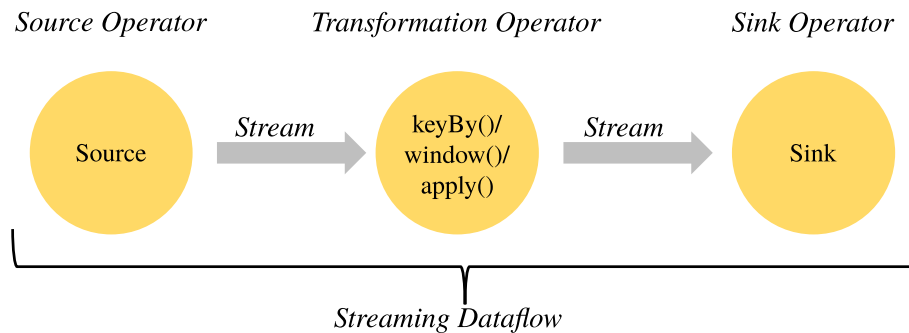


Abbildung 4.2.: Datenfluss- und Programmiermodell von Apache Flink [Fli18a].

Apache Storm

Apache Storm ist eine quelloffene in Java implementierte Plattform für die Analyse von Datenströmen und aktuell in der Version 1.2 verfügbar [Sto]. Mit Hilfe von Bolts und Spouts können für die Ereignisströme CEP-Regeln modelliert werden. Die definierten Anfragen können parallel auf einem Rechnernetz ausgeführt werden. Des Weiteren bietet Apache Storm eine integrierte Fehlerbehandlung an, die im Ausfall Knoten neustartet. Die Funktionalitäten, wie zum Beispiel die Verwendung von Fenster, Bolts und Spouts sind ausführlich dokumentiert.

Esper

Esper ist ein CEP-System aus dem Hause EsperTech und wird in einer kostenlosen und kommerziellen Version angeboten [Esp18]. Komplexe Ereignisse werden mit der SQL-ähnlichen Sprache Esper and Event Processing Language (EPL) beschrieben. Zusätzlich werden unterschiedliche Fensterarten zur Einschränkung des Datenstroms bereitgestellt. Die freie Version unterstützt keine verteilte Ausführung auf einem Rechnernetz, hierfür muss die kommerzielle Version erworben werden.

Siddhi

Siddhi ist eine Java-Bibliothek die es ermöglicht Muster in Datenströmen zu erkennen und unterschiedliche Aktionen auszuführen [Sid18]. Der Quellcode und die Dokumentation sind frei zugänglich und werden unter der Apache Lizenz 2 vertrieben. Die Beschreibung von komplexen Ereignissen erfolgt mit einer eigenentwickelten SQL-Sprache (Siddhi Streaming SQL). Siddhi unterstützt zudem eine verteilte Ausführung und ist bereits im kommerziellen Einsatz [Sid18].

Heron

Heron ist eine von Twitter entwickelte echtzeitfähige, verteilte und fehlertolerante Streaming-Plattform und ist der direkte Nachfolger von Apache Storm [Her18]. Für die Migration von Apache Storm CEP-Regeln sind keine Programmcodeänderungen erforderlich, da die Heron Schnittstelle

rückwärtskompatibel mit der von Apache Storm ist. Bei der Entwicklung von Heron wurde architektonische Verbesserungen von Storm umgesetzt, die eine performantere verteilte Ausführung und bessere Skalierbarkeit ermöglichen. Der Quellcode ist frei zugänglich und wird aktuell in der Version 0.17.5 unter der Apache Lizenz 2 vertrieben.

4.3. Gegenüberstellung

Für einen besseren Überblick, über die zuvor vorgestellten Systeme, werden in diesem Abschnitt die Systeme gegenübergestellt. Hierbei wird auf die bereits erwähnten Auswahlkriterien eingegangen und jedes System getrennt bewertet. Die folgende Tabelle 4.1 stellt eine Übersicht über grundlegende Funktionen verschiedener Systeme dar.

	DCEP	Apache Flink	Apache Storm	Esper	Siddhi	Heron
Längen- und Zeitfenster mit Verschiebung	✓	✓	✓	✓	✓	✓
Entwicklungsstadium (Version)	Prototyp	1.6	1.2	7.1	3.0	0.7
Vollständige Dokumentation	✗	✓	✓	✗()	✓	✓
Ausführung auf Rechnerverbund	✓	✓	✓	✓	✓	✓
Open Source	✓	✓	✓	✗	✓	✓
Beschreibung von komp. Ereignissen	Java	Java / CEP-API	Java	EPL	Siddhi-SQL	Java

Tabelle 4.1.: Gegenüberstellung von CEP-Systemen

4.4. Auswahl

Die vorgestellten Systeme sind potentiell für das folgende Experiment geeignet, jedoch können mit Hilfe der Gegenüberstellung, Systeme ausgeschlossen werden.

Ziel dieser Arbeit ist es, die Skalierbarkeit von CEP-Systemen zu untersuchen. Da Esper nur in der kommerziellen Version eine verteilte Ausführung unterstützt, wird es nicht im Experiment berücksichtigt. Apache Storm wird im Experiment außer Acht gelassen, da es als Basis für die Entwicklung von Heron diente. Die Erkennung von komplexen Ereignissen erfolgt in DCEP ausschließlich in Java. Systeme die für das Experiment in Frage kommen, sollten diese Möglichkeit ebenfalls unterstützen. Da Siddhi nur eine selbstentwickelte SQL-Sprache bereitstellt, ist eine Migration notwendig und ein ähnlicher Programmablauf nicht garantiert, somit kann es ausgeschlossen werden. Heron und Apache Flink bieten beide eine verteilte Ausführung an jedoch benötigt man bei Heron einen höheren Konfigurationsaufwand. Apache Flink ermöglicht eine verteilte Ausführung mit Hilfe von bereitgestellten Skripten.

Aus diesen Gründen wird nur Apache Flink für das Experiment in Betracht gezogen. Ein weiterer Punkt, der für Apache Flink spricht, ist die unterschiedliche parallele Verarbeitung der Daten. Bei DCEP erfolgt dies stapelbasiert, indem gesamte Fenster von unterschiedlichen Instanzen verarbeitet werden, wohingegen Apache Flink einen schlüsselbasierten Ansatz verfolgt.

5. Datensätze und Anwendungsszenarien

In diesem Kapitel werden die Anwendungsszenarien für die jeweiligen Datensätze genauer vorgestellt. Hierfür wird jeder Datensatz beschrieben und aufgezeigt wie man ihn erstellt. Die aufgeführten Datensätze dienen als Grundlage für das folgende Experiment.

5.1. Sensordaten am Beispiel eines Fußballspiels

Dieser Datensatz wurde von der Distributed Event-Based Systems (DEBS)-Konferenz 2013 im Zuge der jährlichen Grand Challenge bereitgestellt und besteht aus Positions- und Geschwindigkeitsmessungen von Spielern während eines Fußballspieles [DEB13]. Das Testspiel wurde auf einem halben Spielfeld zwischen zwei Mannschaften mit jeweils 8 Spielern aufgezeichnet und dauerte 60 Minuten, wobei zwei Abschnitte mit jeweils 30 Minuten gespielt wurden [MZJ13].

Um die Position der Spieler und des Balles auf dem Spielfeld zu bestimmen kann das Spielfeld als Rechteck mit den Koordinaten $(0, 33965)$, $(-50, -33960)$, $(52489, -33939)$ und $(52477, 33941)$ betrachtet werden. In Abbildung 5.1 wird das Spielfeld schematisch mit Koordinaten für die Ecken und Torlinien beider Mannschaften dargestellt.

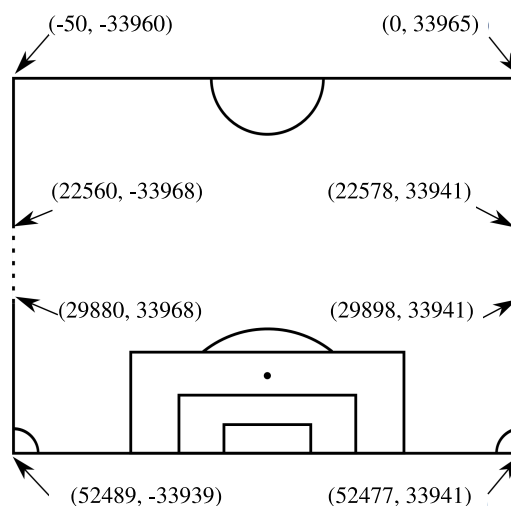


Abbildung 5.1.: Schematische Darstellung des Spielfeldes mit Koordinaten.

Listing 5.1 Datenschema der Sensordaten im Rohdatensatz

```
sid          // Eindeutige Sensorkennung
ts           // Zeitstempel
x,y,z       // Sensorkoordinaten
|v|,        // Geschwindigkeit
|a|,        // Beschleunigung
vx , vy , vz // Richtungsvektor
ax , ay , az // Beschleunigungsvektor
```

Der Schiedsrichter und die Spieler wurden während der Aufzeichnung mit zwei Sensoren in ihren Schienbeinschützern ausgestattet, die mit einer Frequenz von 200 Ereignissen pro Sekunde Daten verschickten. Die Torhüter wurden zusätzlich mit zwei Sensoren in ihren Handschuhen ausgestattet. In jedem Ball wurde ein Sensor mittig angebracht, der 2000 Positions- und Geschwindigkeitsänderungen pro Sekunde registrierte.

Auflistung 5.1 zeigt den Aufbau des Rohdatensatzes. Der Wert *sid* ist die eindeutige Kennung des Sensors, der die Positions- und Bewegungswerte berechnet. Der Zeitstempel *ts* repräsentiert den Zeitpunkt der Messung in Pikosekunden. Die Werte *x*, *y* und *z* beschreiben die Position im Spielfeld des Sensors. Geschwindigkeit und Beschleunigung des Sensors können mit $|v|$ und $|a|$ ermittelt werden.

Nach Ablauf der regulären Spielzeit entstand ein Datensatz mit rund 50 Millionen Einträgen der zur Auswertung bereitsteht.

5.1.1. Anwendungsszenario Ballbesitz

Ein Ball ist in Besitz eines Spielers, wenn er weniger als einen Meter von ihm entfernt ist. Durch Pässe oder Balleroberungen kommt es bei Fußballspielen regelmäßig zu einem Ballwechsel. Ziel dieser Anfrage ist es einen Ballwechsel und somit einen neuen Ballbesitz zu erkennen. Hierfür wird kontinuierlich die Position des Balles mit allen Spielern abgeglichen. Besitzt ein anderer Spieler einen geringeren Abstand zum Ball wird das als Ballwechsel erkannt und als komplexes Ereignis wahrgenommen.

5.1.2. Anwendungsszenario Spieler am Ball

Bei diesem Szenario werden die Abstände aller Spieler zum Ball ermittelt und die Spieler mit dem geringsten Abstand hervorgehoben. Durch die ständige Bewegung des Balles ändern sich folglich auch die Abstände der Spieler zum Ball. Ziel dieser Anfrage ist es Änderungen der Abstände zu erkennen.

Listing 5.2 Grundstruktur von allen Ereignissen der GitHub Plattform [Git18a].

```
[
  {
    "type": "Event",
    "public": true,
    "payload": {},
    "repo": {
      "id": 3,
      "name": "octocat/Hello-World",
      "url": "https://api.github.com/repos/octocat/Hello-World"
    },
    "actor": {
      "id": 1,
      "login": "octocat",
      "gravatar_id": "",
      "avatar_url": "https://github.com/images/error/octocat_happy.gif",
      "url": "https://api.github.com/users/octocat"
    },
    "org": {
      "id": 1,
      "login": "github",
      "gravatar_id": "",
      "url": "https://api.github.com/orgs/github",
      "avatar_url": "https://github.com/images/error/octocat_happy.gif"
    },
    "created_at": "2011-09-06T17:26:27Z",
    "id": "12345"
  }
]
```

Listing 5.3 Verkürzte Datenstruktur von GitHub Ereignissen.

```
eventType // Ereignisart
user      // Benutzername
repo     // Name des GitHub Repositoriums
ts       // Zeitstempel
```

5.2. Öffentliche Aktivitätsdaten der Entwicklerplattformen GitHub

GitHub ist ein weltweiter Onlinedienst mit mehr als 25 Millionen Benutzern und stellt über 70 Millionen Repositorien bereit [Git17]. Ein großer Teil der Repositorien ist öffentlich einsehbar und wird täglich aktualisiert [Git17]. GitHub unterstützt aktuell 32 unterschiedliche Ereignisarten, die von Dateiänderungen, Kommentaren bis hin zur Erstellung neuer Projekte reichen [Git18a]. Github Archive speichert jede dieser Interaktionen der Plattform und stellt diese zur Verfügung [Git18b].

Alle Ereignisse besitzen die selbe Datenstruktur (siehe Listing 5.2) und unterscheiden sich lediglich durch die Felder *type* und *payload*. Der Datensatz kann mit dem angefügten Skript A.3.1 mit Angabe von Tag Monat und Jahr heruntergeladen werden.

Um die überflüssigen Informationen aus den Datensätzen zu entfernen wird das Skript A.3.2 verwendet. Nach der Filterung stellt Listing 5.3 schematisch die optimierte Datenstruktur dar. Das Feld *eventType* kennzeichnet den Ereignistyp des Datensatzes und spiegelt eine der von GitHub verfügbaren Interaktionen wieder. Der eindeutige Benutzername wird mit dem Feld *user* hinterlegt. Das geänderte Repository wird im Wert *repo* festgehalten. Im Feld *ts* wird der Zeitpunkt der Interaktion gespeichert.

5.2.1. Auswertung des Datensatzes

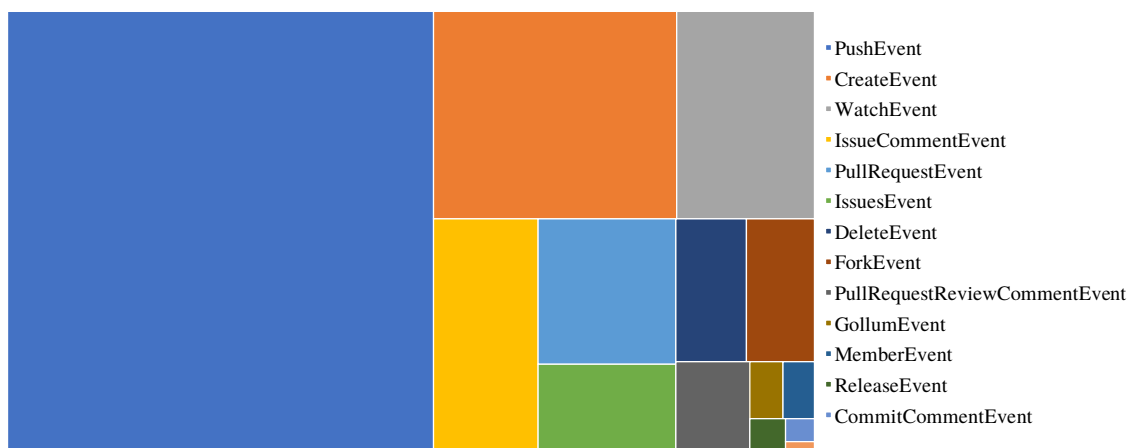


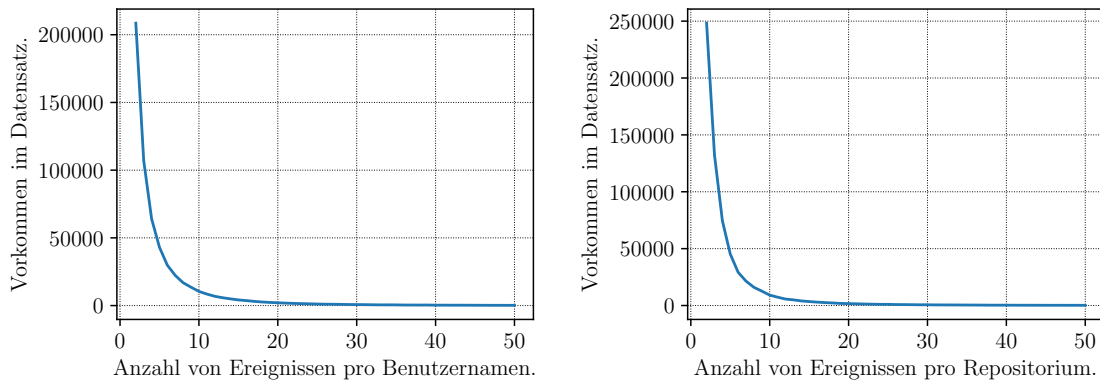
Abbildung 5.2.: Häufigkeit der unterschiedlichen Ereignisarten.

Um komplexe Ereignisse zu modellieren wurde zu Beginn der Datensatz exemplarisch für die Tage 19.03.2018 und 20.03.2018 mit den Skripten A.3.1, A.3.2 heruntergeladen und bereinigt. Daraufhin wurde der Datensatz anhand von Ereignisarten, Benutzernamen und Repositorien untersucht. Die Abbildung 5.2 visualisiert die Auswertung des bereinigten Datensatzes und zeigt die Häufigkeit des Auftretens der unterschiedlichen Ereignisarten. Der exemplarische Datensatz besteht aus 3334291 Einträgen und die drei am häufigsten auftretenden Ereignisarten sind *PushEvent* mit 1760415 gefolgt von *CreateEvent* mit 468873 und *WatchEvent* mit 266400. Dieser Datensatz wird auch in Kapitel 6 verwendet.

Die Abbildungen 5.3b und 5.3a veranschaulichen die Anzahl der Ereignisse pro Benutzername und pro Repository. Aus den Abbildungen ist zu entnehmen, dass das Vorkommen im Datensatz stark mit der Anzahl der von unterschiedlichen Ereignissen abnimmt.

5.2.2. Mustererkennung mit Hilfe von regulären Ausdrücken

Für das Finden von komplexen Ereignissen wurde eine generische Schnittstelle implementiert, die auf Grundlage von aufeinander folgenden Ereignisarten definierte Muster erkennt. Möchte man beispielsweise ein Muster finden, in dem ein Ereignis von Typ 2 auf ein Ereignis von Typ 1 folgt, muss man die Suche wie untenstehend angeben. Die Schnittstelle generiert dynamisch anhand der Suchanfrage einen regulären Ausdruck.



(a) Anzahl von Ereignissen pro Benutzernamen.

(b) Anzahl von Ereignissen pro Repositoryum.

Abbildung 5.3.: Auswertung des Datensatzes auf Basis von Benutzernamen und Repositoryum.

Suchmuster:	Typ_1->Typ_2
Regulärer Ausdruck :	(Typ_1)+.*(Typ_2)+.*

5.2.3. Anwendungsszenario auf Basis der Benutzernamen

Aus der Auswertung in Abschnitt 5.2.1 folgt, dass 582922 eindeutig identifizierbare Benutzer auf der Plattform agierten. Für dieses Szenario werden alle Ereignisse basierend auf den Benutzernamen untersucht, dazu können einfache Suchmuster und komplexere Suchmuster verwendet werden.

Einfaches Muster :	PushEvent->WatchEvent
Komplexes Muster :	PushEvent->CreateEvent->WatchEvent->IssuesEvent->PullRequestEvent

5.2.4. Anwendungsszenario auf Basis der Repositoryen

Bei folgendem Szenario wiederum werden alle Ereignisse basierend auf den Repositoryen untersucht. Der exemplarische Datensatz beinhaltet 647164 Repositoryen. Auch hier kann ein komplexes und ein einfaches Suchmuster angewandt werden.

Einfaches Muster :	PushEvent->WatchEvent
Komplexes Muster :	PushEvent->ForkEvent->WatchEvent->IssueCommentEvent->PullRequestEvent

5.3. Börsenkurse von Aktiengesellschaften

Für diesen Datensatz wird die öffentliche Schnittstelle von Investors Exchange (IEX), einer Börsenplattform mit Sitz in den Vereinigten Staaten, verwendet [IEX18]. Die Verwendung von Kursdaten findet im Bereich von CEP häufige Anwendung [ABNS06].

Listing 5.4 Datenschema von Kursdaten.

```
sym      // Symbol
open     // Eröffnungskurs
high     // Kurshoch
low      // Kurstief
close    // Schlusskurs
ts       // Zeitstempel
```

IEX stellt Aktienkurse minutengenau für 8675 Unternehmen 30 Kalendertage rückwirkend bereit [IEX18]. Das Skript A.3.3 ermöglicht das Herunterladen von Kursdaten für mehrere Unternehmen im JSON-Format.

Um eine einheitliche Datenstruktur zu verwenden, wird mit dem Skript A.3.2 der Datensatz, wie in Listing 5.4 beschrieben, umgewandelt. Das Feld *sym* spiegelt den Unternehmensnamen in verkürzter Schreibweise wieder. Die Werte *open*, *high*, *low*, und *close* beschreiben die unterschiedlichen Kurswerte für den angefragten Zeitraum. Im Feld *ts* wird der Zeitpunkt der Veröffentlichung gespeichert.

5.3.1. Beispielszenario: Auswertung Börsenwerten mit Hintergrundinformationen

Für die Auswertung von Kursdaten gibt es bereits eine Vielzahl von definierten Mustern [Dow00]. Jedoch basieren diese Muster rein auf den Aktienwerten der jeweiligen Unternehmen und betrachten keine weiteren Informationen. Dieses Szenario untersucht nicht nur die Aktienwerte sondern auch zusätzliche Informationen der gelisteten Unternehmen. Hierzu bietet IEX eine Schnittstelle an mit Hilfe derer unter anderem Informationen zu Schulden, Vermögen und dem Unternehmensbereich abgefragt werden können. Mit diesem Hintergrundwissen zum Unternehmen lassen sich bessere Handlungsstrategien erstellen [TRP12].

Ziel dieser Anfrage ist es Unternehmen im Technologiesektor zu finden, die in den letzten 60 Minuten steigende Aktienwerte vorzeigen. Zusätzlich darf die Schuldenhöhe nicht das Vermögen überschreiten.

5.4. Nummernschilderkennung am Beispiel von Fahrzeugbildern

In diesem Datensatz wird eine Verkehrsüberwachung simuliert, die jedes vorbeifahrende Fahrzeug fotografiert und die gefahrene Geschwindigkeit misst. Für die Fahrzeugbilder wird eine öffentlich zugängliche Quelle verwendet [Mar17]. Wie aus Abbildung 5.4 zu entnehmen ist, handelt es sich um Direktaufnahmen von Fahrzeugen. Die Geschwindigkeit wird zufällig zwischen 50-80 Kilometern pro Stunde generiert.

Mit dem Skript A.3.4 kann ein synthetischer Datensatz beliebiger Größe erzeugt werden. Die Struktur der erzeugten Einträge folgt der Datenstruktur in Listing 5.5. Das Feld *type* beschreibt die Ereignisart. Für eine eindeutige Identifizierung der Ereignisse steht der Wert *id* bereit. Die randomisierte Geschwindigkeit wird im Feld *speed* gespeichert. Das für die Nummernschilderkennung benötigte Fahrzeugbild wird im Feld *image* hinterlegt.



(a) Beispielhaftes Fahrzeugbild von [Mar17].



(b) Beispielhaftes Fahrzeugbild von [Mar17].



(c) Beispielhaftes Fahrzeugbild von [Mar17].

Abbildung 5.4.: Exemplarische Fahrzeugbilder für die Kennzeichenerkennung [Mar17].

Listing 5.5 Datenschema von Verkehrsüberwachungsdaten.

```
type      // Ereignisart
id        // Eindeutige Identifikationsnummer
speed     // Geschwindigkeit
image     // Pfad zum Fahrzeugbild
```

5.4.1. Nummernschilderkennung

Um Nummernschilder zu erkennen müssen mehrere aufwendige Algorithmen angewandt werden [Mar17]. Zu Beginn muss das Kennzeichen lokalisiert und normalisiert werden, um im Anschluss die einzelnen Zeichen zu erkennen. Diese Algorithmen sind bereits von der öffentlichen Bibliothek *JavaANPR* implementiert [Mar17]. Beim Testen des Erkennungsprozesses variiert die benötigte Zeit zwischen 50 und 300 Millisekunden pro Fahrzeugbild.

5.4.2. Anwendungsszenario: Geschwindigkeitsüberprüfung mit Nummernschilderkennung

Das Ziel dieser Anfrage ist es das Nummernschild bei Geschwindigkeitüberschreitung zu erkennen. Hierzu wird ein Schwellwert für die erlaubte Geschwindigkeit vorgegeben und bei jedem Ereignis geprüft ob eine Überschreitung vorliegt. Im Falle einer Geschwindigkeitüberschreitung wird versucht das Nummernschild zu ermitteln. Wird das Kennzeichen erfolgreich entziffert wird dieses mit der Geschwindigkeitüberschreitung als komplexes Ereignis erkannt.

6. Ergebnisse und Auswertung

Im folgenden Kapitel werden die durchgeführten Experimente überprüft und ausgewertet. Hierzu werden zu Beginn der Experimentablauf und die Testumgebung detailliert beschrieben. Im Anschluss folgt die Auswertung der Experimentreihen.

6.1. Experiment

Das Ziel des Experiments ist es, die Auswirkung des Parallelisierungsgrads auf den Durchsatz zu untersuchen. Der Durchsatz ist definiert als die Anzahl der Ereignisse die pro Sekunde vom System verarbeitet werden können. Jedes Beispielszenario aus Kapitel 5 wird mit unterschiedlichen Parallelisierungsgraden ausgeführt. Für das Experiment werden das selbstentwickelte System (siehe Abschnitt 4.1) und Apache Flink (siehe Abschnitt 4.2) berücksichtigt.

6.1.1. Testumgebung

Als Testumgebung wurde der Rechnerverbund *netfe*, der vom Institut bereitgestellt wurde, verwendet. Der Verbund besteht aus sieben Knoten die CentOS in der Version 6.4 als Betriebssystem installiert haben. Die Systemspezifikationen der einzelnen Knoten können aus der Tabelle 6.1 entnommen werden.

Knoten	Anzahl an Prozessoren	Prozessor-Taktung	Verfügbarer Direktzugriffsspeicher
netfe	16	1600 Megahertz	24 Gigabyte
compute-0-0	8	1600 Megahertz	24 Gigabyte
compute-0-1	8	1600 Megahertz	24 Gigabyte
compute-0-2	8	1600 Megahertz	24 Gigabyte
compute-0-3	8	1600 Megahertz	24 Gigabyte
compute-0-4	8	1600 Megahertz	24 Gigabyte
compute-0-5	8	1600 Megahertz	24 Gigabyte

Tabelle 6.1.: Systemspezifikationen des Rechnerverbundes

Alle für das Experiment benötigten Dateien sind in einer hierarchischen Struktur, wie in Tabelle 6.1 dargestellt, abgelegt. Konfigurationsdateien befinden sich innerhalb der jeweiligen Systemordner und die Datensätze werden separat bereitgestellt. Im Ordner *results* werden die Ergebnisse der Experimentreihen gespeichert.

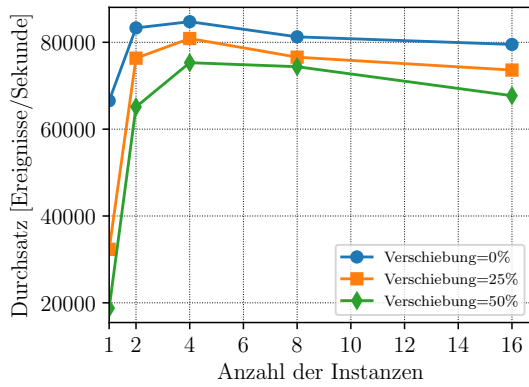
Listing 6.1 Ordnerstruktur auf dem Rechnerverbund

```
|-- data/
    |-- soccer/
    |-- github/
    |-- image_processing/
    |-- stock/
|-- dcep/
    |-- config/
|-- flink/
    |-- config/
|-- results/
```

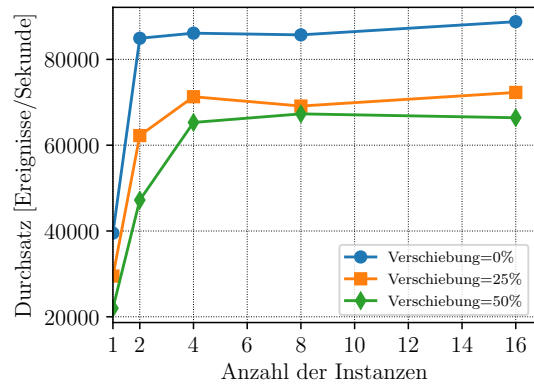
6.2. Durchführung der Experimente

Jedes beschriebene Anwendungsszenario (siehe Kapitel 4) wird im Experiment durchgeführt. Um einen gemittelten Wert für den Durchsatz zu erhalten, wird jedes Experiment zehnmal ausgeführt. Um Abweichungen zu berücksichtigen wird der Durchschnittswert mit Hilfe der zehn Messungen gebildet. Zu Beginn jedes Experiments wird das System ohne Parallelisierung ausgeführt. Die weiteren Durchläufe werden mit den Parallelisierungsgraden 2, 4, 8 und 16 ausgeführt. Um den Ereignisstrom einzugrenzen wird für jede Experimentreihe ein Längenfenster verwendet. Die Auswirkung durch eine Verschiebung der Längenfenster wird zusätzlich mit 0%, 25% und 50% getestet.

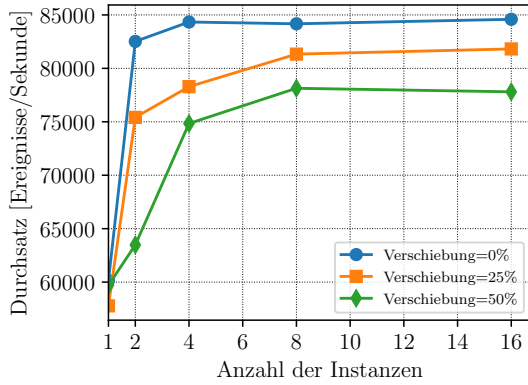
6.2. Durchführung der Experimente



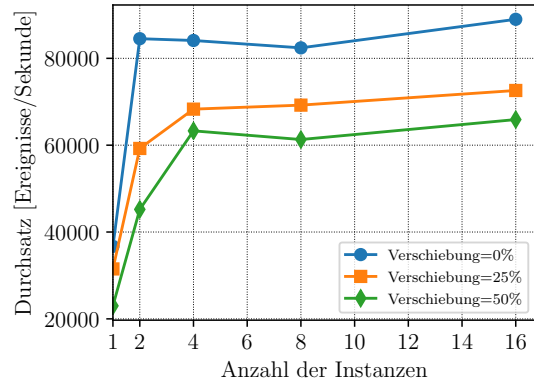
(a) Skalierungsverhalten von DCEP für Anwendungs-szenario 1.



(b) Skalierungsverhalten von Apache Flink für Anwendungsszenario 1.



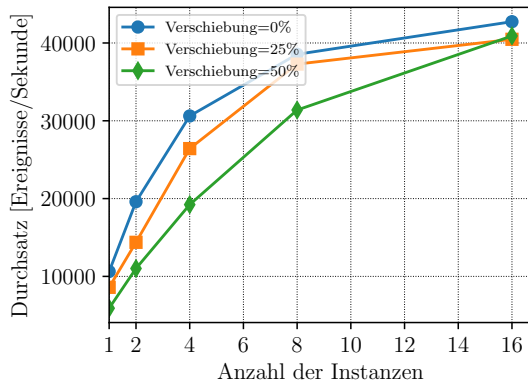
(c) Skalierungsverhalten von DCEP für Anwendungs-szenario 2.



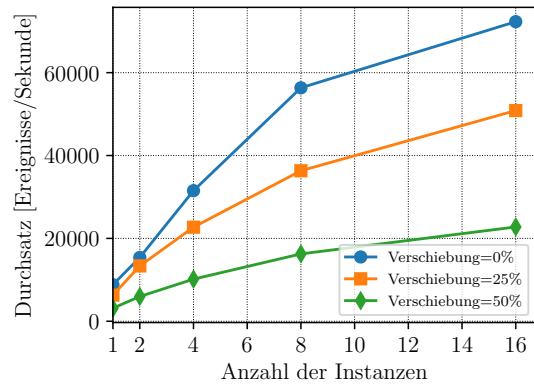
(d) Skalierungsverhalten von Apache Flink für Anwendungsszenario 2.

Abbildung 6.1.: Durchsatzanalyse für den Datensatz 5.1 für die Systeme DCEP und Apache Flink.

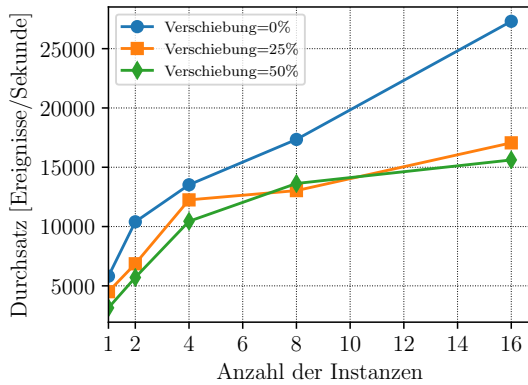
6. Ergebnisse und Auswertung



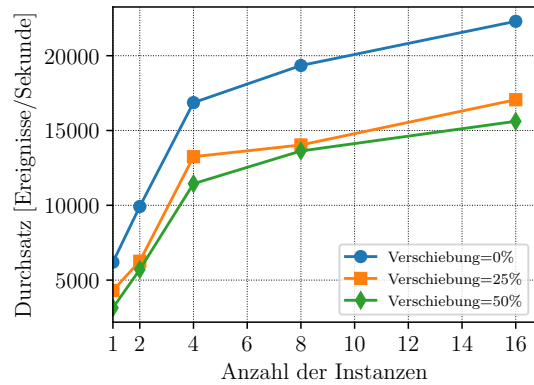
(a) Skalierungsverhalten von DCEP für Anwendungs-szenario 1.



(b) Skalierungsverhalten von Apache Flink für An-wendungsszenario 1.

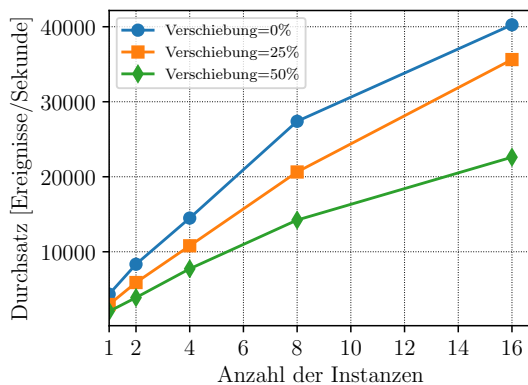


(c) Skalierungsverhalten von DCEP für Anwendungs-szenario 2.

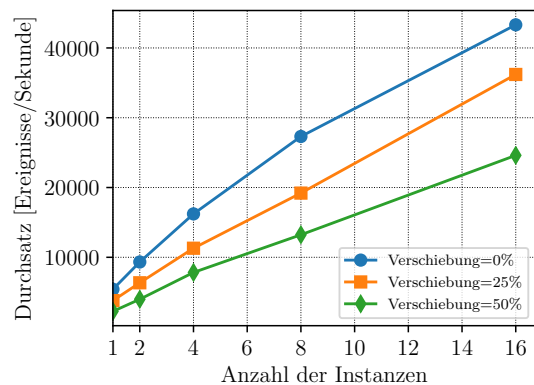


(d) Skalierungsverhalten von Apache Flink für An-wendungsszenario 2.

Abbildung 6.2.: Durchsatzanalyse für den Datensatz 5.2

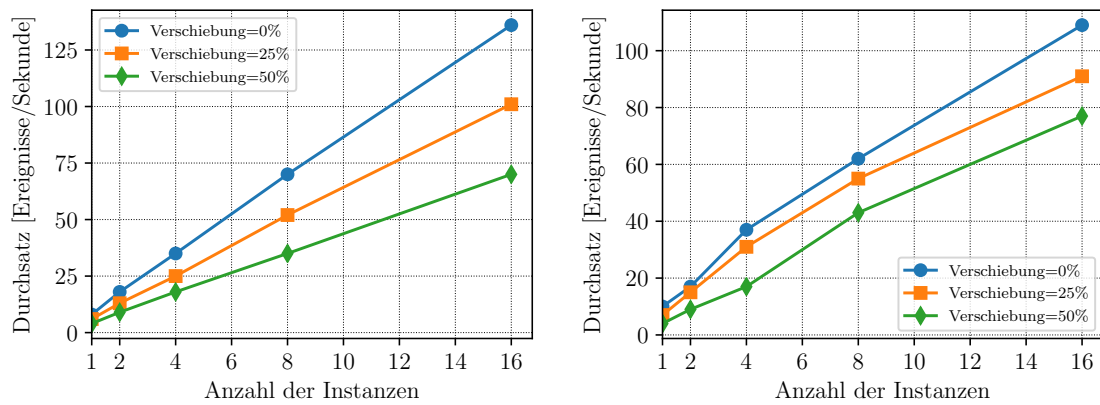


(a) Skalierungsverhalten von DCEP für Anwendungs-szenario 1.



(b) Skalierungsverhalten von Apache Flink für An-wendungsszenario 1.

Abbildung 6.3.: Durchsatzanalyse für den Datensatz 5.3 für die Systeme DCEP und Apache Flink.



(a) Skalierungsverhalten von DCEP für Anwendungs- (b) Skalierungsverhalten von Apache Flink für Anwendungsszenario 1.

Abbildung 6.4.: Durchsatzanalyse für den Datensatz 5.4 für die Systeme DCEP und Apache Flink.

6. Ergebnisse und Auswertung

Experimentreihe 1

Für die erste Experimentreihe wurden die Sensordaten (siehe Abschnitt 5.1) verwendet. Folgende Konfigurationen wurden für die einzelnen Anwendungsszenarien eingesetzt. Das Skalierungsverhalten ist Abbildung 6.1 dargestellt.

```
Konfiguration für Anwendungsszenario 1
Anzahl der Ereignisse : 20000000
Ballbeschleunigung : 1000000
Fenstergröße : 100000
```

```
Konfiguration für Anwendungsszenario 2
Anzahl der Ereignisse : 20000000
Suchmuster : 5 Spieler
Fenstergröße : 100000
```

Experimentreihe 2

Die zweite Experimentreihe verwendete den in Abschnitt 5.2 beschriebenen GitHub Datensatz. Die Ergebnisse der zweiten Experimentreihe mit den folgenden Konfigurationen werden in Abbildung 6.2 dargestellt.

```
Konfiguration für Anwendungsszenario 1
Anzahl der Ereignisse : 10000000
Suchmuster : PushEvent->ForkEvent->WatchEvent->IssueCommentEvent->PullRequestEvent
Fenstergröße : 100000
```

```
Konfiguration für Anwendungsszenario 2
Anzahl der Ereignisse : 10000000
Suchmuster : CreateEvent->PushEvent->WatchEvent
Fenstergröße : 35000
```

Experimentreihe 3

Bei der dritten Experimentreihe wurden die Kursdaten von Aktien (s. Abschnitt 5.3) verwendet. Abbildung 6.3 veranschaulicht die Ergebnisse mit folgender Konfiguration.

```
Konfiguration für Anwendungsszenario 1
Anzahl der Ereignisse : 1000000
Suchmuster : Steigender Wert und Technologiesektor und Vermögen > Schulden
Fenstergröße : 1000
```

Experimentreihe 4

Der Datensatz 5.4 für die Nummernschilderkennung anhand von Fahrzeugbildern diene als Grundlage für die Experimentreihe 4. Die Ergebnisse der Experimentreihe werden in Abbildung 6.4 dargestellt. Für die Experimentreihe wurde die folgende Konfiguration genutzt.

Konfiguration für Anwendungsszenario 1	
Anzahl der Ereignisse	: 50000
Suchmuster	: Geschwindigkeitsüberprüfung mit Nummernschilderkennung
Erlaubte Geschwindigkeit	: 49
Fenstergröße	: 100

6.3. Auswertung der Experimentreihen

Bei der Auswertung ist zu beachten, dass beide Systeme einen unterschiedlichen Ansatz für die Verteilung des Ereignisstroms besitzen. DCEP verfolgt einen stapelbasierten Ansatz, wohingegen Apache Flink einen schlüsselbasierten Ansatz verfolgt. Durch die Experimentreihen kann gezeigt werden, dass sich beide Ansätze für eine verteilte Ausführung eignen und skalieren.

Durch Verschiebung der Längfenster erhöht sich die Anzahl der Fenster und somit auch der Aufwand für die Systeme. Dieses Verhalten veranschaulichen die Abbildungen in diesem Kapitel. Aus den Abbildungen 6.2, 6.3 und 6.4 ist zu entnehmen, dass bei beiden Systemen der Durchsatz durch das Hinzufügen von Knoten im Rechnernetz steigt. Beide Systeme skalieren für die Experimentreihe 4 nahezu linear und nutzen die neu hinzugefügten Knoten effizient aus. Für den ersten Datensatz steigt der Durchsatz, bei beiden Systemen, jedoch nur bis zu zwei Knoten und steigt danach nicht mehr. Grund dafür ist jeweils der Verteilungsalgorithmus der Systeme. Die Operatoren verarbeiten die Fenster schneller als das System den Ereignisstrom verteilen kann. Für die Datensätze 2, 3 und 4 benötigen die Operatoren mehr Zeit für die Mustererkennung wodurch das System ausreichend Zeit für die Verteilung der Last auf mehrere Knoten hat.

7. Zusammenfassung und Ausblick

Das Ziel dieser Arbeit war es, die Auswirkung des Parallelisierungsgrads von CEP-Systemen bezüglich des Durchsatzes zu untersuchen. Dazu wurde in Kapitel 2 auf Begriffe eingegangen, welche für die Mustererkennung in Ereignisverarbeitungssystemen von Bedeutung sind. In Kapitel 3 wurden bereits bestehende wissenschaftliche Arbeiten vorgestellt.

In Kapitel 4 wurde die Funktionsweise des eigenentwickelten CEP-Systems, DCEP (siehe Abschnitt 4.1), aufgezeigt. Mit Hilfe von Auswahlkriterien wurden bereits existierende Systeme ermittelt und evaluiert. Zur Evaluation wurden fünf bereits auf dem Markt verfügbare CEP-Systeme herangezogen und geprüft, ob sich diese zum Vergleich mit dem selbstentwickelten System eignen. Die Evaluierung ermöglichte die Auswahl eines, für die Experimentreihen geeignetes, Systems.

In Kapitel 5 wurden geeignete Datensätze für das Experiment recherchiert und basierend darauf verschiedene Anwendungsszenarien entwickelt. Für die Durchführung des Experimentes wurden zunächst die erarbeiteten Anwendungsszenarien in beiden Systemen umgesetzt. Hierzu wurde versucht möglichst identische Implementierungen für beide Systeme zu erstellen, um eine vergleichbare Ausgangssituation zu ermöglichen. Dennoch mussten die Eigenarten des jeweiligen Systems berücksichtigt werden.

In Kapitel 6 wurden die Ergebnisse der Untersuchung und deren Auswertung vorgestellt. Mit Hilfe der Experimentreihen konnte die Auswirkung des Parallelisierungsgrads auf den Durchsatz von verteilten Complex-Event-Processing-Systemen gezeigt und gegenübergestellt werden.

Inbesondere konnte gezeigt werden, dass das eigenentwickelte Complex-Event-Processing-System, durch das Hinzufügen von weiteren Knoten im Rechnernetz einen höheren Durchsatz ermöglicht und somit horizontal skaliert. Bei zeitintensiven Abfragen in den Operatoren verteilt das selbstentwickelte System die Berechnungen auf die verfügbaren Knoten und skaliert nahezu linear.

Ausblick

Die in dieser Arbeit entwickelten Datensätze und Anwendungsszenarien können als Grundlage für weitere Untersuchungen von CEP-Systemen verwendet werden. Die Datensätze spiegeln, wie in der Einleitung erwähnt, enorme sowie reelle Datenströme wieder.

Im Rahmen dieser Arbeit konnte die horizontale Skalierbarkeit lediglich für das eigenentwickelte und ein weiteres System untersucht werden. Daher bietet sich eine Untersuchung mit weiteren Systemen, mit den erarbeiteten Anwendungsszenarien, für zukünftige Arbeiten an.

Des Weiteren eignen sich die vorgestellten Datensätze auch für Langzeitexperimente, um zum Beispiel die Stabilität, Fehlerverhalten oder Optimierungen der Systeme zu überprüfen.

Literaturverzeichnis

- [ABNS06] A. Adi, D. Botzer, G. Nechushtai, G. Sharon. „Complex Event Processing for Financial Services“. In: *2006 IEEE Services Computing Workshops*. IEEE, Sep. 2006. DOI: [10.1109/scw.2006.7](https://doi.org/10.1109/scw.2006.7) (zitiert auf S. 35).
- [BD15] R. Bruns, J. Dunkel. *Complex Event Processing*. Springer Fachmedien Wiesbaden, 2015. DOI: [10.1007/978-3-658-09899-5](https://doi.org/10.1007/978-3-658-09899-5) (zitiert auf S. 17, 18).
- [BDWT13] C. Balkesen, N. Dindar, M. Wetter, N. Tatbul. „RIP“. In: *Proceedings of the 7th ACM international conference on Distributed event-based systems - DEBS '13*. ACM Press, 2013. DOI: [10.1145/2488222.2488257](https://doi.org/10.1145/2488222.2488257) (zitiert auf S. 15, 18–20, 23).
- [DEB13] DEBS. *DEBS 2013 Grand Challenge: Soccer monitoring – DEBS.org*. 2013. URL: <http://debs.org/debs-2013-grand-challenge-soccer-monitoring/> (zitiert auf S. 31).
- [Dor17] M. Dorner. *Complex-Event-Processing-Systems*. 2017. URL: <https://github.com/michaeldorfner/CEP-Systems> (zitiert auf S. 27).
- [Dow00] E. Downs. *7 Chart Patterns That Consistently Make Money*. Marketplace Books, Inc., 2000. ISBN: 1-883272-61-0. URL: <https://www.amazon.com/Chart-Patterns-That-Consistently-Money/dp/1883272610?SubscriptionId=AKIAIOBINVZYXZQZ2U3A&tag=chimbori05-20&linkCode=xm2&camp=2025&creative=165953&creativeASIN=1883272610> (zitiert auf S. 36).
- [Esp18] EsperTech. *Esper - EsperTech*. 2018. URL: <http://www.espertech.com/esper/> (zitiert auf S. 28).
- [Eva11] D. Evans. „The Internet of Things: How the Next Evolution of the Internet is Changing Everything“. In: 1 (Jan. 2011), S. 1–11 (zitiert auf S. 15).
- [Fli18a] A. Flink. *Apache Flink: Stateful Computations over Data Streams*. 11. Sep. 2018. URL: <https://flink.apache.org/> (zitiert auf S. 27, 28).
- [Fli18b] FlinkCEP. *FlinkCEP - Complex event processing for Flink*. 2018. URL: <https://ci.apache.org/projects/flink/flink-docs-release-1.6/dev/libs/cep.html> (zitiert auf S. 27).
- [Git17] GitHub. *GitHub Octoverse 2017 | Highlights from the last twelve months*. 2017. URL: <https://octoverse.github.com/> (zitiert auf S. 33).
- [Git18a] GitHub. *Event Types & Payloads | GitHub Developer Guide*. 2018. URL: <https://developer.github.com/v3/activity/events/types/> (zitiert auf S. 33).
- [Git18b] GitHub. *GH Archive*. 2018. URL: <https://www.gharchive.org/> (zitiert auf S. 33).
- [GRGH17] D. García-Gil, S. Ramírez-Gallego, S. García, F. Herrera. „A comparison on scalability for batch big data processing on Apache Spark and Apache Flink“. In: *Big Data Analytics 2.1* (März 2017). DOI: [10.1186/s41044-016-0020-2](https://doi.org/10.1186/s41044-016-0020-2) (zitiert auf S. 15, 23).

- [Her18] Heron. *Heron*. 2018. URL: <https://apache.github.io/incubator-heron/> (zitiert auf S. 28).
- [IEX18] IEX. *IEX Group | IEX API*. 2018. URL: <https://iextrading.com/developer/docs/#stocks> (zitiert auf S. 35, 36).
- [Inf18] G. für Informatik (GI). *Complex Event Processing (CEP) - Gesellschaft für Informatik e.V.* 2018. URL: <https://gi.de/informatiklexikon/complex-event-processing-cep/> (zitiert auf S. 23).
- [LLD16] M. A. Lopez, A. G. P. Lobato, O. C. M. B. Duarte. „A Performance Comparison of Open-Source Stream Processing Platforms“. In: *2016 IEEE Global Communications Conference (GLOBECOM)*. IEEE, Dez. 2016. DOI: 10.1109/glocom.2016.7841533 (zitiert auf S. 23).
- [Luc02] D. Luckham. *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems*. Addison-Wesley Professional, 2002. ISBN: 0201727897. URL: <https://www.amazon.com/Power-Events-Introduction-Processing-Distributed/dp/0201727897?SubscriptionId=0JYN1NVW651KCA56C102&tag=techie-20&linkCode=sm2&camp=2025&creative=165953&creativeASIN=0201727897> (zitiert auf S. 17, 18).
- [Mar17] O. Martinsky. *JavaANPR - Automatic Number Plate Recognition System for Java*. 2017. URL: <https://github.com/oskopek/javaanpr> (zitiert auf S. 36, 37).
- [MBM09] M. R. N. Mendes, P. Bizarro, P. Marques. „A Performance Study of Event Processing Systems“. In: *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2009, S. 221–236. DOI: 10.1007/978-3-642-10424-4_16 (zitiert auf S. 23).
- [MTR+17] R. Mayer, M. A. Tariq, K. Rothermel, M. Gräber, U. Ramachandran. „SPECTRE“. In: *Proceedings of the 18th ACM/IFIP/USENIX Middleware Conference on - Middleware '17*. ACM Press, 2017. DOI: 10.1145/3135974.3135983 (zitiert auf S. 20).
- [MZJ13] C. Mutschler, H. Ziekow, Z. Jerzak. „The DEBS 2013 grand challenge“. In: *Proceedings of the 7th ACM international conference on Distributed event-based systems - DEBS '13*. ACM Press, 2013. DOI: 10.1145/2488222.2488283 (zitiert auf S. 31).
- [Sid18] Siddhi. *Siddhi*. 2018. URL: <https://wso2.github.io/siddhi/> (zitiert auf S. 28).
- [Sto] A. Storm. *Apache Storm*. URL: <http://storm.apache.org/> (zitiert auf S. 28).
- [TRP12] K. Teymourian, M. Rohde, A. Paschke. „Knowledge-based processing of complex stock market events“. In: *Proceedings of the 15th International Conference on Extending Database Technology - EDBT '12*. ACM Press, 2012. DOI: 10.1145/2247596.2247674 (zitiert auf S. 36).

Alle URLs wurden zuletzt am 16. 09. 2018 geprüft.

A. Anhang

A.1. Installationshinweis

Der beigefügte Datenträger beinhaltet neben dieser Arbeit zusätzlich den vollständigen DCEP-Quellcode und die Implementierungen aller Anwendungsszenarien für die Systeme DCEP und Apache Flink. Darüber hinaus befindet sich darauf eine Anleitung, die alle notwendigen Schritte für die lokale und verteilte Ausführung beschreibt. Mit Hilfe der Konfigurationsdateien für die jeweiligen Anwendungsszenarien können die Experimentreihen wiederholt werden. Des Weiteren befinden sich Skripte zum Generieren und Bereinigen der Datensätze auf dem Datenträger.

A.2. Google-Suchen

A.2.1. Complex-Event-Processing-Systeme

In Abschnitt 4.2 werden CEP-Systeme als Evaluationsobjekte gesucht. Zur Bestimmung von existierenden Systemen wird die folgende Suche durchgeführt.

Eine Google-Suche mit den Schlüsselbegriffen *cep*, *complex event processing*, *stream processing* und *cep system* liefert am 05. Juni 2018 zwischen 10:20 und 10:30 Uhr /, 8110 Resultate. Die genaue Such-URL lautet <https://www.google.de/search?q=%22cep%22+%22complex+event+processing%22+%22stream+processing%22+%22cep+systems%22&oq=%22cep%22+%22complex+event+processing%22+%22stream+processing%22+%22cep+systems%22&>.

A.3. Skripte zum Generieren und Bereinigen der Datensätze

In Kapitel 5 werden für CEP-Systeme geeignete Datensätze vorgestellt. Zur Generierung und Bereinigung der Datensätze wurden folgende Skripte erarbeitet. Jedes Skript beinhaltet in der ersten Zeile einen beispielhaften Aufruf.

A.3.1. GitHub Datensatz generieren

```
#Usage: ./sript.sh <year> <month> <day>
for i in {0..23} #Downloading dataset
do wget "http://data.gharchive.org/$1-$2-$3-$i.json.gz" "-P$1-$2-$3" "-q"
done

gunzip "$1-$2-$3/*" #Unzip dataset
```

A. Anhang

```
cat "$1-$2-$3/"* > "github-$1-$2-$3.json" #Merge dataset
```

A.3.2. GitHub Datensatz bereinigen

```
//Usage: ./sript.js --i=<pathToInput> --o=<pathToOutput>
const fs = require('fs');
const argv = require('yargs').argv
const readline = require('readline');

processFile(argv.i, argv.o)

function processFile(inputFile, outputFile) {
  instream = fs.createReadStream(inputFile),
  outstream = new(require('stream'))(),
  rl = readline.createInterface(instream, outstream);

  rl.on('line', function (line) {
    if (line != undefined) {
      obj = JSON.parse(line)
      string = obj.type + "," + obj.actor.login + "," + obj.repo.name + "," + Date.parse(obj
.created_at);
      fs.appendFileSync(outputFile, string + '\r\n');
    }
  });

  rl.on('close', function (line) {});
}
```

A.3.3. Börsendatensatz generieren

```
//Usage: ./sript.js --d=<YYYYMMDD> --i=<pathToSymbols> --o=<pathToOutput>
const fs = require('fs');
const moment = require('moment');
const argv = require('yargs').argv
const axios = require('axios');

var lineReader = require('readline').createInterface({
  input: require('fs').createReadStream(argv.i)
});

lineReader.on('line', function (symbol) {
  (async() => {
    try {
      const stock = await axios('https://api.iextrading.com/1.0/stock/' + symbol + '/chart/
date/' + argv.d);
      stock.data.forEach(obj => {
```

```

        if (obj.open !== undefined && obj.high !== undefined && obj.low !== undefined && obj.close !== undefined) {
            var string = moment(obj.date + "-" + obj.minute, 'YYYYMMDD-hh:mm').unix() + ","
                + symbol + "," + obj.open + "," + obj.high + "," + obj.low
                + "," + obj.close + ","
            fs.appendFileSync(argv.o, string + '\r\n');
        }
    });

    } catch (e) {}
    })();
});

```

A.3.4. Verkehrsüberwachungsdaten generieren

```

//Usage: ./sript.js --n=<numberOfEvents> --i=<numberOfImages> --o=<pathToOutput>
const argv = require('yargs').argv
const fs = require('fs');
var id = 1;

for (let i = 1; i <= argv.n; i++) {
    let event = ({
        type: 'TrafficEvent',
        id: i,
        speed: Math.floor(Math.random() * 31) + 50,
        image: 'vehicle_' + id + '.jpg'
    });

    fs.appendFileSync(argv.o, JSON.stringify(event) + '\r\n');
    if (id == argv.i) { id = 0; }
    id++;
}

```


Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben. Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

Ort, Datum, Unterschrift