Version MILESTONE 5 (MS05)
letzte Änderung: 26. Mai 2014

BEUTH HOCHSCHULE
FÜR TECHNIK
BERLIN
University of Applied Sciences

Vergleich von Streamingframeworks: STORM, KAFKA, FLUME, S4

vorgelegt von

Eduard Bergen

Matrikel-Nr.: 769248

dem Fachbereich VI – Informatik und Medien –
der Beuth Hochschule für Technik Berlin vorgelegte Masterarbeit
zur Erlangung des akademischen Grades
Master of Science (M.Sc.)
im Studiengang
Medieninformatik-Online (Master)

Tag der Abgabe 27. Oktober 2014

1. Betreuer	Herr Prof. Dr. Edlich	Beuth Hochschule für Technik
Gutachter	Herr Prof. Knabe	Beuth Hochschule für Technik

Kurzfassung

Mit der enormen Zunahme von Nachrichten durch unterschiedliche Quellen wie Sensoren (RFID) oder Nachrichtenquellen (RFD newsfeeds) wird es schwieriger Informationen beständig abzufragen. Um die Frage zu klären, welcher Rechner am häufigsten über TCP frequentiert wird, werden unterstützende Systeme notwendig. An dieser Stelle helfen Methoden aus dem Bereich des Complex Event Processing (CEP). Im Spezialbereich Stream Processing von CEP wurden Streaming Frameworks entwickelt, um die Arbeit in der Datenflussverarbeitung zu unterstützen und damit komplexe Abfragen auf einer höheren Schicht zu vereinfachen.

Abstract

Inhaltsverzeichnis

Abbildungsverzeichnis

Kapitel 1

Einführung

> Social media streams, such as Twitter, have shown themselves to be useful sources of real-time information about what is happening in the world. Automatic detection and tracking of events identified in these streams have a variety of real-world applications, e.g. identifying and automatically reporting road accidents for emergency services. [MMO+13]

Im Internet steigt das Angebot zu unterschiedlichen Informationen rapide an. Gerade in Deutschland wächst das Datenaufkommen, wie die Studie der IDC [Dig14, S. 2-3] zeigt, exponentiell. Dabei nimmt ebenfalls das Interesse an wiederkehrenden Aussagen über die Anzahl bestimmter Produkte, die Beziehungen zu Personen und die persönlichen Stimmungen zueinander zu. So wird in [Dat14] eine interaktive Grafik zum Zeitpunkt der Ansprache zur Lage der Union des Präsidenten der USA angezeigt. Je Zeitpunkt und Themenschwerpunkt wird in der Ansprache zeitgleich die Metrik Engagement zu den einzelnen Bundesstaaten aus den verteilten Twitternachrichten berechnet ausgegeben.

In der Infografik [Jam14b] von Josh James, Firma Domo wird ein Datenwachstum von 2011 bis 2013 um 14,3% veranschaulicht. Es werden unterschiedliche Webseiten vorgestellt. Dabei werden unterschiedlichen Arten von Daten, die pro Minute im Internet erzeugt werden gezeigt. In der ersten Fassung [Jam14a] waren es noch 2 Millionen Suchabfragen auf der Google-Suchseite [Goo14]. Die zweite Fassung gibt über 4 Millionen Suchanfragen pro Minute an. In Facebook [Fac14] konnten in der Fassung mehr als 680 Tausend Inhalte getauscht werden. In der zweiten Fassung werden mehr als 2,4 Millionen Inhalte pro Minute getauscht.

Um die Sicherheit bei Verlust einer Kreditkarte zu erhöhen und gleichzeitig die höchste Flexibilität zu erhalten, gibt es im Falle eines Schadens bei der von unterschiedlichen Orten gleichzeitig eine unerwünschte Banküberweisung stattfindet, für die Bank die Möglichkeit, die Transaktion aufgrund der Positionserkennung zurückzuführen [SÇZ05, S. 3, K. Integrate Stored and Streaming Data].

Mit steigenden Anforderungen, wie in der Umfrage [Cap14, S. 8] durch schnellere Analyse, Erkennung möglicher Fehler und Kostenersparnis dargestellt, und damit einem massiven Datenaufkommen ausgesetzt, kann die herkömmliche Datenverarbeitung [CD97, S. 2, K. Architecure and End-to-End Process] durch das Zwischenlagern der Daten in einem Datenzentrum keine komplexen und stetigen Anfragen zeitnah beantworten [MMO+13, S. 2 K. Related Work: Big Data and Distributed Stream Processing]. Damit müssen Nachrichten, sobald ein Nachrichteneingang besteht, sofort verarbeitet werden können. Allen Goldberg stellt in [GP84, S. 1, K. Stream Processing Example] anhand eines einfachen Beispiels Stream processing zu deutsch

Verarbeitung eines Nachrichtenstroms ausgehend von loop fusion [GP84, S. 7, K. History] vor. Da Allen Goldbergs Beschreibung zu Stream processing in die Ursprünge geht, soll ein einfaches Modell eines Stream processing Systems für die weitere Betrachtung als Grundlage dienen.

So wird in [AAB+05, S. 2, K. 2.1: Architecture] die distributed stream processing engine Borealis vorgestellt und als große verteilte Warteschlangenverarbeitung beschrieben. Die Abbildung [AAB+05, S. 3, A. 1: Borealis Architecture] zeigt eine Borealis-Node mit Query processor in der Abfragen verarbeitet werden. Eine Borealis-Node entspricht einem Operator, in dem laufend Datentupel sequentiell verarbeitet werden. Mehrere Nodes sind in einem Netzwerk verbunden und lösen dadurch komplexe Abfragen. Damit die Komplexität, die Lastverteilung und somit die Steigerung der Kapazität für die Entwicklung von neuen Anwendungen vereinfacht werden, wurden Streaming frameworks entwickelt. Streaming frameworks stellen auf einer höheren Abstraktion Methoden zur Datenverarbeitung bereit.

Bisher werden einzelne Streaming frameworks separat in Büchern oder im Internet im Dokumentationsbereich der Produktwebseiten vorgestellt. Dabei werden vorwiegend Methoden des einzelnen Streaming frameworks erläutert und auf weiterführenden Seiten vertieft. Als Software Entwickler wird der Nutzen für die Streaming frameworks nicht sofort klar. Zum Teil sind die Dokumentationen veraltet, in einem Überführungsprozess einer neuen Version oder es fehlt ein schneller Einstieg mit einer kleinen Beispielanwendung.

In dieser Arbeit soll es eine Übersicht mit Einordnung und Spezifikation über die einzelnen Streaming frameworks Apache Storm [Mar13], Apache Kafka [KNR13], Apache Flume [PMS13] und Apache S4 [GJM+13] geben. Dabei werden außerdem die Streaming frameworks diskutiert und verglichen.

Da viele Begriffe aus dem englischen Sprachraum kommen, wird der englische Begriff kurz erläutert und im weiterführenden Text kursiv gekennzeichnet. Kapitel werden eingeleitet und ausgeleitet.

Die Arbeit ist in zwei Bereiche geteilt. Im ersten Bereich werden Grundlagen geschaffen, es wird analysiert und die Streaming frameworks werden vorgestellt. Im zweiten Bereich werden die Streaming frameworks diskutiert und verglichen. Dabei wird ein praxisnaher Anwendungsfall vorgestellt. Und im Schlussteil wird zusammengefasst, die Erkenntnis vorgestellt und ein Ausblick gegeben.

Das erste Kapitel befasst sich mit der Einführung und im zweiten Kapitel werden die Grundlagen geschaffen. Dabei werden die Grundbegriffe und die zum Einsatz notwendige Technologie vorgestellt, eingeordnet und zusammengefasst. Im dritten Kapitel findet eine Analyse in Verbindung der gewonnenen Grundlagen statt. Kapitel Vier stellt einen großen Teil dar. Darin werden die einzelnen Streaming frameworks vorgestellt. Das Kapitel endet mit einer Zusammenfassung. In Kapitel Fünf wird ein Anwendungsfall vorgestellt, in dem die Streaming framewoks im Einsatz gezeigt werden. Das sechste Kapitel knüpft an das vorangegangene an und stellt die Diskussion und den Vergleich. Das letzte Kapitel Sieben enthält die Schlussbetrachtung mit einer Zusammenfassung, einer Erkenntnis und einem Ausblick. In Anhang A sind zusätzliche Inhalte und Quelltexte hinterlegt.

Kapitel 2

Grundlagen

Im folgenden Kapitel werden die Begriffe Event, Stream, Processing aus der Informatik im Bereich der verteilten Systeme erläutert und in einen Zusammenhang zu Streaming frameworks gebracht. Dabei wird ein Grundkonzept für eine streambasierte Nachrichtenverarbeitung vorgestellt. Im weiteren Verlauf und maßgeblich in Kapitel 4 wird stets auf das Grundkonzept Bezug genommen. In der Einführung wurde die stream processing engine Borealis [AAB+05] als ein einfaches Modell eines Stream processing-Systems erwähnt. Zuerst werden im Unterkapitel 2.1 die wesentlichen Fachbegriffe vorgestellt. Anschließend wird im Unterkapitel 2.2 ein Zeitbezug zu verwandten Technologien gegeben und die Streaming frameworks aus Kapitel 4 werden eingeordnet. Das Kapitel 2 endet mit einer Zusammenfassung und leitet in das Kapitel 3 ein.

2.1 Grundbegriffe

Ein großer Teil der verwendeten Grundbegriffe sind in [TvS07] definiert. An dieser Stelle werden nur die wesentlichen Grundbegriffe vorgestellt. Ein Verteiltes System wird von Andrew S. Tanenbaum und Maarten van Steen in [TvS07, S. 19, K 1.1] grob definiert:

> Ein verteiltes System ist eine Ansammlung unabhängiger Computer, die den Benutzern wie ein einzelnes kohärentes System erscheinen.

Verteilte Systeme bestehen also laut [TvS07] aus unabhängigen Komponenten und enthalten eine bestimmte Form der Kommunikation zwischen den Komponenten. Informationen werden zwischen Sender und Empfänger über ein Signal ausgetauscht. Dazu hat Claude E. Shannon in [Sha48, S. 2, A. 1] ein Diagramm eines allgemeinen Kommunikationssystems vorgestellt. In der genannten Abbildung wird das Signal in einem Kanal codiert übertragen. Dabei ist das Signal einem Umgebungsrauschen ausgesetzt. Durch Einsatz geeigneter Kodierverfahren in Übertragungsprotokollen können Übertragungsfehler festgestellt und behoben werden. Im schlimmsten Fall wird eine fehlerhaft übertragene Nachricht zum Beispiel innerhalb des Transmission Control Protocol (TCP) auf OSI Schichtebene 4 in [Uni94, S. 40, K. 7.4.4.6 Data transfer phase] neu übertragen. Der Kanal ist das Medium in [Sha48], um die Nachricht zu übertragen. Tanenbaum und van Steen beschreiben in [TvS07, S. 184, K. 4.4.1] ein kontinuierliches Medium Temperatursensor gegenüber einem diskreten Medium Quelltext als zeitkritisch zwischen Signalen. Shannon beschreibt in [Sha48, S. 3 und S. 34] ein kontinuierliches System als:

> A continuous system is one in which the message and signal are both treated as continuous functions, e.g., radio or television. [...] An ensemble of functions is the appropriate mathematical representation of the messages produced by a continuous source (for example, speech), of the signals produced by a transmitter, and of the perturbing noise. Communication theory is properly concerned, as has been emphasized by Wiener, not with operations on particular functions, but with operations on ensembles of functions. A communication system is designed not for a particular speech function and still less for a sine wave, but for the ensemble of speech functions.

Ein Stream oder ein Datastream ist damit eine Folge von Signalen. Einem Signal entspricht ein Event und die Anwendung von Funktionen findet im Processing statt. Somit ist Event stream processing eine Signalfolgenverarbeitung in einem kontinuierlichen Medium. Weiterhin soll in diesem Zusammenhang von Event stream processing oder abgekürzt ESP benutzt werden.

Da zu Streams ebenfalls eine Paketierung von unterschiedlichen Substreams aus Audio, Video und Synchronisierungsspezifikation verstanden wird, wie in [TvS07, S. 191, letzter Absatz] mit dem Kompressionsverfahren 2 und 4 für Audio und Video Übertragung der Motion Pictures Expert Group (MPEG) gezeigt, soll an dieser Stelle keine tiefergehende Untersuchung in den Zusammenschluss unterschiedlicher Algorithmen zur Komprimierung der Substreams in einen Stream erfolgen.

Weiterhin beschreibt Muthukrishnan in [Mut10] (2010) mehrere Forschungsrichtungen in Datastreams. Darunter werden „[...] theory of streaming computation [...], data stream management systems [...], theory of compressed sensing [...],“ [Mut10, S. 2, Absatz 2] aufgezählt. In *streaming computation* wird an geringen Zugriffszeiten während mehrfachem Zugriff auf gegebener Datennachrichten geforscht. Mit einem *data stream management system* soll ein Zugriff, durch Einsatz von speziellen Operatoren auf nicht endende Datenquellen möglich sein. Und in der *theory of compressed sensing* wird nach geringen Zugriffsraten zum Aufteilen in Signalmustern unterhalb der Nyquist-Rate geforscht. So findet Streaming in der Signalverwaltung, Signalverarbeitung und Signaltheorie eine Anwendung.

Während Streams auf einem Prozessorsystem verarbeitet werden können, muss eine hohe Kapazität von Daten auf einem oder mehreren Multiprozessorsystemen in einer geringen Latenz verteilt berechnet werden können. Tanenbaum und van Steen stellen die Grundlagen der Remote Procedure Call (RPC)-Verwendung in [TvS07, S. 150, K. 4.2.1] vor. Abstraktionen der Schnittstelle zur Transportebene, wie diese auf OSI Ebene 4 durch TCP angeboten werden, bilden dabei eine Vereinfachung um Funktionen mit übergebenen Parametern auf entfernten Rechnern aufzurufen. Nach der entfernten Berechnung wird das Ergebnis sofort an den Client zurückgeschickt. Dabei ist der Client bei einem synchronen Nachrichtenmodell blockiert bis der Server geantwortet hat. Sobald die Berechnung durchgeführt wurde, wartet im asynchronen Nachrichtenmodell der Client nicht und wird erst nach Abschluß der Berechnung am Server vom Server informiert. Währenddessen können weitere Anfragen durch den Client auf den Server erfolgen.

Wie in [TvS07, S. 170, K. 4.3.2] vorgestellt, wurde durch den Einsatz von Warteschlangensystemen ein zeitlich lose gekoppelter Nachrichtenaustausch zwischen Sender und Empfänger möglich. Der Empfänger entscheidet selbst wann und ob eine Nachricht eines Senders von der Warteschlange abgeholt wird. Zusätzlich entsteht die Möglichkeit des Warteschlangensystems Nachrichten zwischenzuspeichern. Im Gegensatz zu RPCs haben Nachrichten in Warteschlangensystemen eine Adresse und können beliebige Daten enthalten.

Mehrere Server in einem Verbund bilden ein Cluster. In einem Cluster übernehmen einzelne Rechner-Knoten die Berechnung. Außerhalb der Rechner-Knoten gibt es einen Master-Knoten mit dem die Rechenaufgaben auf die Rechner-Knoten verteilt werden. Dazu wird von Tanenbaum und van Steen in [TvS07, S. 35, A. 1.6] ein Cluster-Computersystem in einem Netzwerk gezeigt. Diese Prinzip wird auch in den Streaming frameworks eingesetzt. In dem Kapitel 4 werden die einzelnen Frameworks im Detail vorgestellt. Die Streaming frameworks selbst bieten dabei ähnlich wie es bei den RPCs der Fall ist, eine Abstraktionsschicht um die Datenverarbeitung für den Entwickler zu vereinfachen. Dazu werden abstrakte Primitive und Operatoren für die Anwendung auf einem unterliegenden Cluster bereitgestellt.

Es wurden die Grundbegriffe eines Streaming Frameworks vorgestellt und eingeordnet. In Kapitel 2.2 wird ein Basis Streaming Framework als Modell vorgestellt. Außerdem werden die Streaming Frameworks Storm, Kafka, Flume und S4 kurz zum Basis Streaming Framework Modell in Zusammenhang gebracht. Die Technologie die dazu zum Einsatz notwendig ist, wird in eigenen Unterkapiteln vorgestellt.

2.2 Technologie

Mit den gewonnen Grundbegriffen werden in diesem Kapitel ein Modell eines Basis Streaming framework vorgestellt. Zuerst wird das Grundmodell und deren Komponenten gezeigt und beschrieben. Anschließend werden die Straming frameworks Storm, Kafka, Flume und S4 mit dem Modell des Streaming frameworks in eine Beziehung gebracht und erläutert. Das Unterkapitel 2.2 endet mit weiteren Komponenten für Streaming frameworks und leitet in das Unterkapitel Zusammenfassung ein.

Ein Basis Modell für Streaming frameworks soll durch eine Stream Processing Engine (SPE) Aurora/Borealis [ACc+03] veranschaulicht werden. Im weiteren Verlauf wird zur Vereinfachung das Schlagwort *Aurora* anstatt SPE Aurora/Borealis verwendet. So besteht ein Modell in [ACc+03, S. 2, Abb. 1 Aurora system model] aus ankommenden Daten, den *Input data streams*, aus ausgehenden Daten, dem *Output to applications* und aus wiederkehrenden Abfragen, den *Continous queries*. In Abbildung 2.1 wird ein Modell als Grundlage für weitere Betrachtungen zu Streaming frameworks vorgestellt. Dabei wird ein azyklisch gerichteter Graph im Zentrum als Verarbeitungseinheit mit linker und rechter Datenflussüberführung gezeigt.

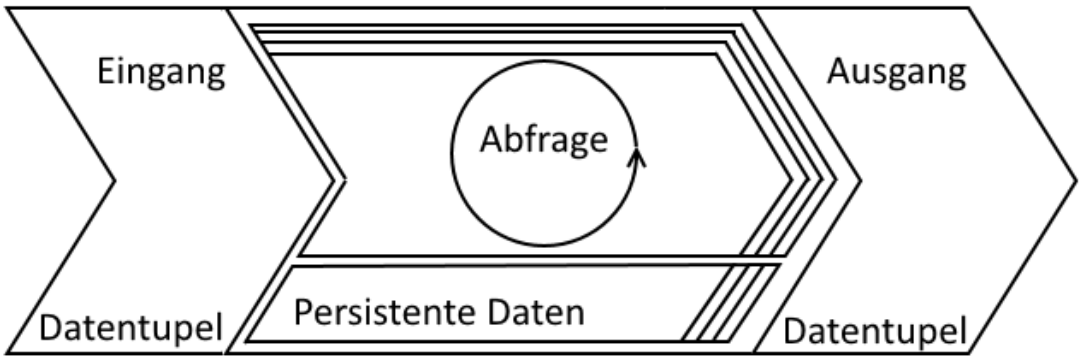

Abbildung 2.1: Exemplarische Darstellung eines Basis Modells für Streaming frameworks

Dabei sind *Input data streams* eine Sammlung von Werten und werden von *Aurora* als eindeutiges Tupel mit einem Zeitstempel identifiziert. Innerhalb von *Aurora* können mehrere *Continous*

queries gleichzeitig ausgeführt werden. Abbildung 2.1 stellt in der Mitte der Grafik zwischen Eingang und Ausgang der Datentupel mehrschichtige Ebenen als Repräsentation für mehrere *Continous queries* dar.

Ein *Continous query* besteht aus *boxes* und *arrows*. *Boxes* sind Operatoren um ankommende Datentupel in ausgehende Datentupel zu überführen. Durch die *Arrows* wird eine Beziehung zwischen den *Boxes* hergestellt. Ein komplexer Beziehungsgraph ist in eine Richtung gerichtet, enthält keine Zyklen, hat mehrere Startknoten und einen Endknoten. Für die weitere Datenverarbeitung können in einem *Continous query* zusätzlich persistente Datenquellen in einer *Box* zur Transformation von Datentupeln hinzugefügt werden. Dazu wird in Abbildung 2.1 unterhalb der *Continous queries* ein mehrschichtiger separater Bereich für die persistenten Daten dargestellt. Im Endknoten des azyklisch gerichteten Graphen werden die transformierten Datentupel für weitere Anwendungen als Ausgabestrom von Datentupeln bereitgestellt. Die Abbildung 2.2 stellt beispielhaft einen azyklisch gerichteten Graphen dar. Der dargestellte Graph enthält zwei Eingangdatenquellen. Die obere Datenquelle wird zeitlich kurz vor dem Endknoten mit der unteren Datenquelle kombiniert. Die unter Datenquelle enhält wird nach der zweiten Transformation in zwei neue Datenquellen aufgespalten. Nach drei Transformationen wird die mittlere Datenquelle in die oberer Datenquelle zeitlich an der sechsten Transformation überführt. Die untere Datenquelle wird mit der oberen Datenquelle an der siebten Transformation verbunden. Die letzte Transformation bildet den Endknoten.

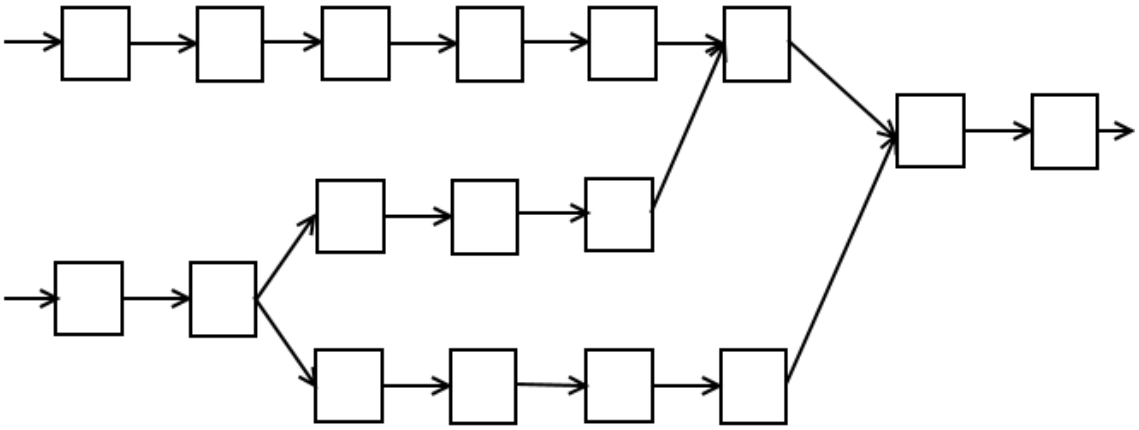

Abbildung 2.2: Darstellung eines azyklisch gerichteten Graphen

Das Datenmodell in *Aurora* besteht aus einem *Header*, dem Kopfbereich und Data, dem Datenteil als Tupel. Der Header in einem Basismodell besteht aus einem Zeitstempel. Mit dem Zeitstempel wird das Datenpaket eindeutig identifiziert und wird für das Monitoring in Quality of Service (QoS) als einen Dienst für die Güte eingesetzt. Im Gegensatz zu *Aurora* wird in der auf *Aurora* basierten Weiterentwicklung *Borealis* ein Vorhersagemodell für QoS zu jedem Zeitpunkt in einem Datenfluss möglich. Dazu wird jedem Datentupel ein Vector of Metrics (VM) hinzugefügt. Ein VM besteht aus weiteren Eigenschaften wie zum Beispiel Ankunftszeit oder Signifikanz. In [AAB+05, S. 3, Kap. 2.4 QoS Model] werden Vector of Metrics vorgestellt.

In *Borealis* gibt es statuslose Operatoren und Operatoren mit einem Status. Statuslose Operatoren sind Filter, Map und Union. Mit dem Filter kann eine Datenquelle nach bestimmten Bedingungen neue Datenquellen erzeugen. Der Map-Operator kann bestimmte Datentupel in einer Datenquelle transformieren wie zum Beispiel durch anreichern von Informationen. Mit dem Union-Operator können mehrere Datenquellen in eine Datenquelle zusammengeführt werden. Dazu wird ein Zwischenspeicher in der Größe $n + 1$ benutzt. In [Tea06, S. 9, Abb. 3.1 Sample outputs from stateless operators] wird eine Übersicht über die drei Operatoren Filter, Map und Union anhand eines konkreten Beispiels dargestellt. Operatoren mit einem Status wie

Join und *Aggregate* werden in [Tea06, S. 9, Kap. 3.2.2 Stateful Operators] als Berechnungen von speziellen Zeitfenstern, dem *window*, die mit der Zeit mitbewegen erläutert. In [Tea06, S. 10, Abb. 3.2 Sample output from an aggregate operator] wird ein Schaubild zum Operator *Aggregate* mit der Funktion *group by*, *average* und *order* in einem *window* gezeigt. Dabei werden eingehende Datenquellen mit einem Schema nach Zeit, Ort und Temperatur in einem Zeitfenster von einer Stunde gruppiert nach Raum, gemittelt nach Temperatur und sortiert nach Zeit in eine ausgehende Datenquelle transformiert. In Abbildung 2.3 wird ein Abfrage-Diagramm in einer Stream Processing Engine dargestellt. Es werden zwei Sensoren S1 und S2 mit dem Union-Operator in einen Stream zusammengeführt. Der Stream wird von zwei Aggregate-Operatoren in einem Zeitfenster von 60 Sekunden getrennt und jeweils mit einem Filter-Operator reduziert. Durch den Join-Operator werden beide Stream in einem Zeitfenster von 60 Sekunden zu einem dritten Stream transformiert.

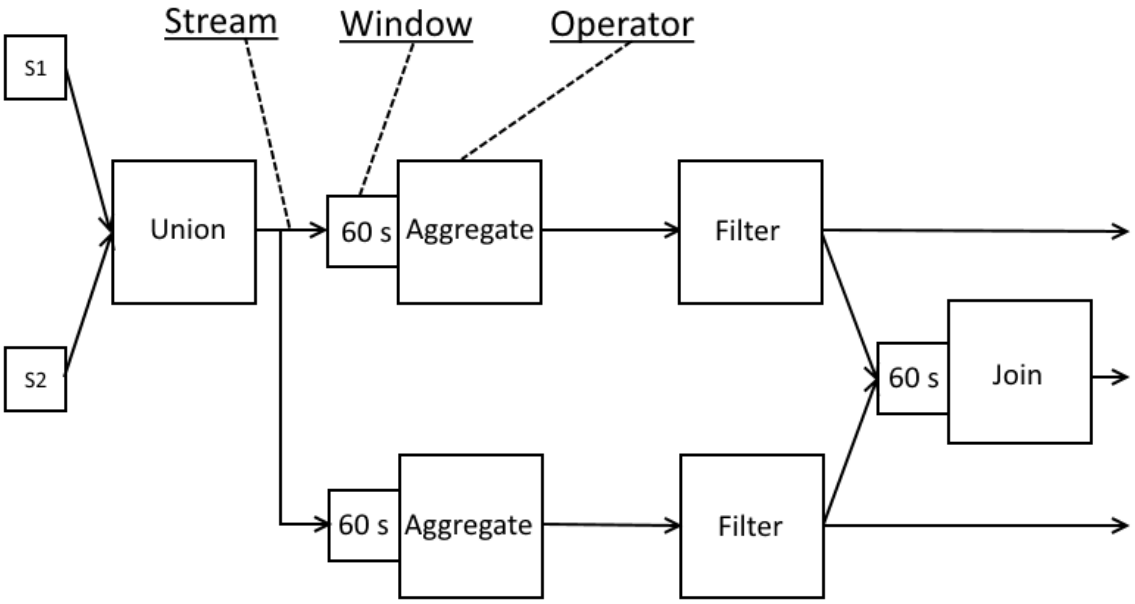

Abbildung 2.3: Diagramm einer Abfrage in einer Stream Processing Engine

Datentupel in *Aurora* können aufgrund von technischen Fehlern, wie zum Beispiel Sensorausfall oder doppelter Parametrierung von mehreren Sensoren durch Hinzufügen, Löschen oder Aktualisieren verschiedene Versionen annehmen. Daher wurde das Datenmodell in *Borealis* im *Header* um einen Revisionstyp und einem Index erweitert. In separaten Speichern den Connection Points (CPs) werden die Revisionen der Datentupel als Historie gehalten. Die CPs sind direkt an einer Datenquelle angeschlossen. Operatoren können auf die CPs durch die Identifikatoren im *Header* auf benötigte Datentupel in der Historie zugreifen.

In den Streaming frameworks Storm, Kafka, Flume und S4 wird eine ähnliche Architektur wie sie im Referenzmodel von *Aurora* und *Borealis* vorgestellt wurde benutzt. Zwischen den Streaming frameworks gibt es trotzdem Unterschiede. Das Referenzmodell von *Aurora* und *Borealis* soll dem Verständnis bei der Vorstellung der Streaming frameworks im Kapitel 4 dienen und die Unterschiede aufzeigen. Mit der Einführung der Grundbegriffe und eines Referenzmodells soll nun das Kapitel Grundlagen im Unterkapitel 2.3 zusammengefasst werden.

2.3 Zusammenfassung

Im Kapitel Grundlagen wurden die Streaming frameworks in die Bereiche der Informationsverarbeitung in verteilten Systemen, der Signaltheorie und der wiederkehrenden Berechnung von Daten in Datenströmen eingeordnet. Dabei wurden aktuelle Forschungsbereiche aufgezeigt und es wurde ein Referenzmodell als Grundlage dargestellt. In der Beschreibung des Referenzmodells wurden die Primitive und die komplexen Operatoren vorgestellt. Weiterhin wurde ein Abfrage-Diagramm anhand eines azyklisch gerichteten Graphen gezeigt. Mögliche Fehlererkennung durch Qualitätssicherungsmaßnahmen wurden durch Vector of Metrics angesprochen. Fehlererkennungsmechanismen und Gütesicherung werden im Einzelnen im Kapitel 4 aufgezeigt. Bevor die Streaming frameworks vorgestellt werden, wird in Kapitel 3 die Umgebung und der Markt analysiert.

Kapitel 3

Analyse

Kapitel 4

Vor- und Gegenüberstellung Streaming Frameworks

Kapitel 5

Anwendungsfall und Prototyp

Kapitel 6

Auswertung

6.1 Benchmark Ergebnisse

6.2 Erkenntnis

Kapitel 7

Schlussbetrachtung

7.1 Zusammenfassung

7.2 Einschränkungen

7.3 Ausblick

Verzeichnisse

Literaturverzeichnis

[AAB+05] Abadi, Daniel J, Yanif Ahmad, Magdalena Balazinska, Ugur Cetintemel, Mitch Cherniack, Jeong-Hyon Hwang, Wolfgang Lindner, Anurag Maskey, Alex Rasin, Esther Ryvkina et al.: *The Design of the Borealis Stream Processing Engine.* In: *CIDR*, Band 5, Seiten 277–289, 2005.

[ACc+03] Abadi, Daniel J., Don Carney, Ugur Çetintemel, Mitch Cherniack, Christian Convey, Sangdon Lee, Michael Stonebraker, Nesime Tatbul und Stan Zdonik: *Aurora: A New Model and Architecture for Data Stream Management*. The VLDB Journal, 12(2):120–139, August 2003.

[Cap14] Capital, KPMG: *Going beyound the data: Achieving actionable insight with data and analytics*, Januar 2014.

[CD97] Chaudhuri, Surajit und Umeshwar Dayal: *An overview of data warehousing and OLAP technology*. SIGMOD Rec., 26(1):65–74, März 1997.

[Dig14] Digital, EMC: *Digital universe around the world*, April 2014.

[GP84] Goldberg, Allen und Robert Paige: *Stream processing*. In: *Proceedings of the 1984 ACM Symposium on LISP and functional programming*, LFP '84, Seiten 53–62, New York, NY, USA, 1984. ACM.

[MMO+13] McCreadie, Richard, Craig Macdonald, Iadh Ounis, Miles Osborne und Sasa Petrovic: *Scalable distributed event detection for Twitter*. In: *2013 IEEE International Conference on Big Data*, Seiten 543–549. IEEE, October 2013.

[Mut10] Muthukrishnan, S.: *Massive data streams research: Where to go*. Technischer Bericht, Rutgers University, March 2010.

[SÇZ05] Stonebraker, Michael, Uğur Çetintemel und Stan Zdonik: *The 8 requirements of real-time stream processing*. ACM SIGMOD Record, 34(4):42–47, 2005.

[Sha48] Shannon, Claude E.: *A Mathematical Theory of Communication*. The Bell System Technical Journal, 27:379–423, 623–656, July, October 1948.

[Tea06] Team, Borealis: *Borealis application programmer's guide*. Technischer Bericht, Brown University, May 2006.

[TvS07] Tanenbaum, Andrew S. und Maarten van Steen: *Verteilte Systeme*. PEARSON STUDIUM, 2., Aufl. Auflage, 2007.

[Uni94] Union, International Telecommunication: *Information technology – Open Systems Interconnection – Basic Reference Model: The basic model*, 1994.

Internetquellen

[Dat14] Data, Twitter: *State of The Union address 2014 URL: http://twitter.github.io/interactive/sotu2014/#p1*, May 2014. Abgerufen am 12.05.2014.

[Fac14] Facebook: *Facebook URL: https://www.facebook.com/*, May 2014. Abgerufen am 12.05.2014.

[GJM+13] Gopalakrishna, Kishore, Flavio Junqueira, Matthieu Morel, Leo Neumeyer, Bruce Robbins und Daniel Gomez Ferro: *S4 distributed stream computing platform. URL: http://incubator.apache.org/s4/*, June 2013. Abgerufen am 30.06.2013.

[Goo14] Google: *Google Search URL: https://www.google.de/*, May 2014. Abgerufen am 12.05.2014.

[Jam14a] James, Josh: *Data Never Sleeps 1.0 URL: http://www.domo.com/blog/wp-content/uploads/2012/06/DatainOneMinute.jpg*, May 2014. Abgerufen am 12.05.2014.

[Jam14b] James, Josh: *Data Never Sleeps 2.0 URL: http://www.domo.com/blog/wp-content/uploads/2014/04/DataNeverSleeps_2.0_v2.jpg*, May 2014. Abgerufen am 05.05.2014.

[KNR13] Kreps, Jay, Neha Narkhede und Jun Rao: *Apache Kafka is publish-subscribe messaging rethought as a distributed commit log. URL: http://kafka.apache.org/*, June 2013. Abgerufen am 29.06.2013.

[Mar13] Marz, Nathan: *Storm is a distributed realtime computation system. URL: https://github.com/nathanmarz/storm/wiki/Home/*, June 2013. Abgerufen am 29.06.2013.

[PMS13] Prabhakar, Arvind, Prasad Mujumdar und Eric Sammer: *Apache Flume distributed, reliable, and available service for efficientliy operating large amounts of log data. URL: http://flume.apache.org/*, June 2013. Abgerufen am 29.06.2013.

Abbildungsverzeichnis

Anhang A

Zusätze

A.1 Abkürzungen

CP	Connection Point 9
ESP	Event Stream Processing 6
MPEG	Motion Pictures Expert Group 6
QoS	Quality of Service 8
RPC	Remote Procedure Call 6, 7
SPE	Stream Processing Engine i, 7, 9
TCP	Transmission Control Protocol 5, 6
VM	Vector of Metrics 8, 10