Version MILESTONE 4 (MS04)
letzte Änderung: 19. 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 gesprochen 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 MPEG gezeigt, soll an dieser Stelle keine tiefergehende Untersuchung in den Zusammenschluss unterschiedlicher Algorithmen zur Komprimierung der Substreams in einen Stream erfolgen.

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. Im asynchronen Nachrichtenmodell wartet der Client nicht und wird erst vom Server informiert, sobald die Berechnung durchgeführt wurde. Währenddessen können weitere Anfragen durch den Client 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 RPC haben Nachrichten in Warteschlangensystemen eine Adresse und können beliebige Daten enthalten.

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 Aggregate für die Anwendung auf einem unterliegenden Cluster bereitgestellt.

## 2.2 Technologie

## 2.3 Zusammenfassung

# 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

# 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.

[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.

[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.

[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.

# Anhang A

# Quelltext zum Prototyp