# Streaming Szenario zur Unterstützung der Ankündigungen von Paketen im Logistikbereich

LOTAIRE TCHAMADEU TIAPPI, Hamburg University of Sciences

Stream-Computing wird immer mehr zu einem beliebten Paradigma, da es die Echtzeitversprechung von Datenanalysen ermöglicht. Besondern in der Logistik Branche mit mehr als 10 Million durchschnittlichen Sendungen pro Tag bei großen Paketdienstleistern lässt sich schon oft bedenken, wie man einen besseren Überblick über das Geschäftsmodell haben kann, um die Pakete besser zu steuern und wettbewerbsfähig zu bleiben. Doch mit einer unendlichen Anzahl von Lösungen lässt es sich nicht einfach die richtige Technologie für sein Geschäftsfeld vom Anfang an erkennen. In dieser Arbeit zeigen wir konstruktiv, wie man einen Streaming Prototyp aufbaut. Speziell für den Bereich der Logistik fassen wir Berichtergebnisse über mehre Benchmarking-Frameworks zusammen und entwerfen progressiv einen stabilen Prototyp für einen Paktdienstleisterbedarf nach dem Informationsgewinn aus den Berichten.

CCS Concepts: • **Computer systems Big Data**; *Big Data und Paketdienstleister*

**KEYWORDS**

Apache Spark, Apache Flink, Apache Kafak, Apache Storm, Benchmarking Big data, Logistik und Streaming, Big data Prototyp

## 1 INTRODUCTION

Die Digitalisierung ist heutzutage ein Begriff, dem viele von uns nicht mehr den Rücken zukehren würden. Darüber wird fast überall gesprochen. Sowohl in fachlichen Zeitschriften, in der Uni, als auch in Unternehmen ist der Begriff stark präsent und steuert unsere Wirtschaft. Wer wettbewerbsfähig sein will, sollte sich mit dem Thema Digitalisierung allgemein und Industrie 4.0 beschäftigen. Dies bedeutet, dass die Unternehmen die jeweiligen Eingaben oder Daten digital erfassen und die Information für ihre Geschäftszwecke aus den Daten extrahieren müssen. Die Art und Weise, wie die vielfältigen Daten mit großem Volumen und so schnell wie möglich verfasst werden, versteht man unmittelbar im Begriff der Big Data. Hierunter versteht man den Umfang, die Vielfältigkeit, die Wahrhaftigkeit der Daten und die Geschwindigkeit, indem die Daten gesammelt werden [1].

Doch die Unternehmen bemühen sich und wollen eine Big Data Lösung und vor allem eine Streamings Lösung, um wettbewerbsfähig zu werden. Die Lösungen variieren immer, je nach Arbeitsumfeld des Unternehmens. Besonders im Bereich der Logistik stellt man sich noch die Frage, ob man einen Streamingprototypen einsetzen soll, um die Paketinformationen zu verstehen, die Nachfrage auf dem Markt besser zu überdecken oder das Risiko des Paketverlusts zu vermeiden. Hier besteht immer noch der Wunsch nach einer stabilen Streaminganlage, wobei die stabilen Streaming-Infrastrukturen erst mit einem Prototyp anfangen würden. Ein Prototyp ist intuitiv definiert als erster Implementierungsvorgang einer Idee, der als konzeptionelle - oder technische Umsetzung gesehen werden kann. Man wird frühzeitig Feedback bezüglich der Eignung eines Lösungsansatzes bekommen, dann Schritt für Schritt den Prototyp verbessern bis ein stabiles Konzept oder eine technische Implementierung der Idee entsteht. Also bleibt es nicht einfach für viele Unternehmen den richtigen Prototyp je nach Herausforderung des Geschäftsbereichs zu finden.

Schwerpunkt dieser Arbeit ist es zu wissen, wie ein Prototyp einer Streamingsanlage bei den Paketdienstleistern im Bereich der Logistik aussehen würde? Welche Parametrisierung soll bei der Auswahl der Technologien betrachtet werden? Diese Arbeit beschränkt sich auf das Vorgehen bei dem Aufbau eines Streamingprototypen für den Logistikbereich, unter anderem auf das Benchmarking, das in dieser Arbeit nicht von uns durchgeführt ist. Aber die sind die Berichten- Ergebnisse von verschiedene Benchmarking, die in 2 Papers durchgeführt wurden.

Die Motivation dieser Arbeit kann zwei Perspektiven haben: zum einen sollen in der Logistik die Fachkräfte unterstützt werden, um die Wahl über eine bestimmte Big Data Technologie zu erleichtern oder Hilfe aufzeigen, um sich den Technologien anzuvertrauen, die für den Streamingprototypen in dem Bereich geeignet sind. Dazu kommt noch der wirtschaftliche Vorteil, dass die Logistik schneller als vorher an ihre Daten kommt. Diese Daten können mit Hilfe der Echtzeitverarbeitung analysiert werden, was eine neue Dimension der Datenanalyse bevorzugt. Damit können die Betrugsversuche von Zustellern schnell erkannt werden und zwar am selben Tag und in derselben Minute, wo der Betrugsversuch passiert ist.

# 2 Bisherige Arbeit

Viele Ansätze wurden für verschiedene Anlässe für den Streamingprototypen vorgeschlagen. Normalerweise umfasst ein Streamingprototyp drei Arten von Akteuren: Sensor, Producer und Verbraucher. Der Sensor besteht aus Einheiten auf der ersten Ebene der Streamingarchitektur, die Messungen als Eingangsdaten erzeugen. Die Messungen werden dann von einer Distributed Streaming Platform gesammelt, die die Daten bei Aufforderung an die Verbraucher übermittelt.
Für die Echtzeitbearbeitung von Daten und Hochleistungsrechnern wurden mehrere Patterns für Big Data Architekturen vorgeschlagen, wobei eine Architektur die grundlegende Anordnung und Konnektivität von Teilen eines Systems modelliert [2].
Mohiuddin Solaimani, Mohammed Iftekhar, Latifur Khan u.a präsentierten schon ein neuartiges, generisches, in Echtzeit verteiltes Anomalie-Erkennungs-Framework für Multi-Source-Stream-Daten. Sie haben sich nur auf den Ansatz mit Apache Spark beschränkt [3], ein Framework für Echtzeit Datenverarbeitung.
Eine andere Vorstellung eines Streamingprototypen wird von Yi-Hsin Wu, Sheng-De Wang, Li-jung chen und Cheng-Juei Yu verstärkt. Sie zeigen, wie man ein Framework für ein Streaming-Analysesystem für Daten- und Analyseworkflows entwickeln kann, um Halbleiter-Domain-Anwendungen zu unterstützen[4]. Mit anderen Worten entwickeln sie ein Streaming-Analytik-System, mit dem die stabile Produktionseffizienz mehrerer Produktionslinien gleichzeitig bewertet werden kann. Sie zeigen aber nicht, aus welchen Kriterien oder Eigenschaftswerten man sich für eine Big Data Technologie entscheiden soll.
Diese Arbeit unterscheidet sich von vorherigen Arbeiten, indem wir Ihnen zeigen, wie man ganz einfach, die richtigen Frameworks aussuchen und diese zusammen integrieren kann, um die minimale Herausforderung eines Streamingprototypen im Logistikbereich abzudecken. Dazu zeigen wir Ihnen die wichtigen Kriterien, die man betrachten soll, wenn man zu dem Punkt kommt, dass Sie sich selbst für eine Big Data Technologie entscheiden müssen.

# 3 Methode

Bei der Neukonzeption einer Streaminganlage wird normalerweise mit low-fidelity Prototypen begonnen, die wenige Details sowohl in Bezug auf das Design als auch die Funktionalität enthalten. Unser Prototyp wird helfen, eine erste Vorstellung von der Streaminganlage bei den Paketdienstleistern zu bekommen und er zeichnet sich durch seine schnelle und somit kostengünstige Entwicklung aus. Wir werden für den Zweck folgende Schritte verfolgen:
Zuerst werden wir das Problem grob analysieren, indem wir über Herausforderungen berichten, vor denen die Paketdienstleister stehen. Diese werden uns helfen, unsere Streaming-Systemspezifikation zu definieren.

Danach landen wir im Hauptteil unserer Arbeit, wo wir den Systementwurf machen wollen. Dafür präsentieren wir die Systemkernkomponente einer Streaminganlage. Nach der Vorstellung der Systemkomponente stellen wir tabellarisch aus der Literatur die erwarteten Eigenschaften je nach Framework dar. Die helfen uns, die jeweilige Frameworksleistung zu bewerten. In demselben Schritt präsentieren wir eine Literaturübersicht über das Benchmarking, das von verschiedenen Autoren erstellt wurde, um die Abweichung zwischen den Leistungswerten von verschiedenen Frameworkstechnologien zu erfahren und zwar zwischen den in der Literatur beschriebenen Leistungswerten und den in den Benchmarking-Ergebnissen resultierten Leistungswerten.
Nach dem Gewinn der Erkenntnisse bei dem Streamings-Benchmarking treffen wir eine Entscheidung für bestimmte Frameworkskomponenten und bauen im nächsten Schritt unseren ersten Streamingprototyp. Dieser Schritt ist Bestandteil der Implementierung.
Zum Schluss stabilisieren wir progressiv unsere Streaminganlage im Rahmen des Betriebs und Wartung, indem wir uns mit dem Problem der Ausfallsicherheit beschäftigen usw.

# 4 Eigene Arbeit

## 4.1 Problemanalyse und Grobplanung

Heutzutage werden Verbraucher für ihre Einkäufe online aktiver als in normalen Läden. Sie bestellen online und lassen die Ware nach Hause schicken. Durch dieses Einkaufsmodell entsteht ein boomender E-Commerce. Digital Shopping wird immer präsenter und aufwändiger. In der realen Welt, der analogen Welt bleibt alles beim alten Geschäftsmodell: Sobald die Bestellung bei dem Kunden bestätigt wird, sollen die bestellten Produkte verpackt, transportiert und ausgeliefert werden. Durch diese Marktentwicklung haben Kurier-, Express- und Paketdienstleister (KEP) immer mehr zu tun: Die Zahlen aus der Statistik zeigen, dass im Jahr 2016 in Deutschland erstmals mehr als drei Milliarden Sendungen ausgeliefert wurden. Die Anzahl der Sendungen wird immer höher. Laut KEP-Verband wurden im vergangenen Jahr in Deutschland 3,16 Milliarden Sendungen auf die Reise geschickt. Zum Vergleich: Vor fünf Jahren waren es 2,47 Milliarden, vor zehn Jahren nur 2,12 Milliarden Sendungen. Der Branchenverband schätzt, dass die Zahl in den nächsten fünf Jahren auf 4,15 Milliarden steigen wird . Bei mehr als zehn Millionen Sendungen pro Tag werden ungefähr 347 Sendungen pro Sekunde erwartet, wenn ein achtstündiger Arbeitstag betrachtet wird[5]. Bei diesem boomenden E-Commerce ist die Digitalisierung mancher Arbeitsprozesse in der Logistik für die Paketzustellung sehr wichtig. Die Paketlieferung soll fehlerfrei ausgeführt werden, weshalb alle Paketinformationen zuverlässig erfasst und gesteuert werden müssen. Die Paketdaten können echtzeitig erfasst und analysiert werden. Das heißt, dass absichtliche Betrugsversuche von manchen Paketzustellern verhindert werden können, indem sie die IST-Geodaten der Pakete von den Zustellern bekommen haben und diese mit den SOLL-Geodaten abgeglichen werden, welche von den Zustellern nach der jeweiligen Paketzustellung mit Hilfe ihrer modernen Scanner gesendet werden. Der Paketverlust sowie die Transportschäden werden kulant abgewickelt. Aber dieser Verlust bleibt für viele dieser Unternehmen eine große Herausforderung. Es geht darum, den Verlust so gering wie möglich zu halten oder sogar zu beseitigen. Oft wollen selbst die Paketdienstleister kaum Zahlen über verlorengegangene Sendungen hergeben: in der Onlineausgabe der Frankfurter Neue Presse stand, dass es bei durchschnittlich drei von 10 000 Sendungen zu derartigen Unregelmäßigkeiten kommt kann. Das wäre so ungefähr 0,03 Prozent Verlust [6], was langfristig zu verhindern ist.

## 4.2 Systemspezifikation

Für die Unternehmen generell und besonders für den Logistikbereich sind selbst produktionsfertige Datenverarbeitungsframeworks komplizierte Softwarekomponenten. Im Allgemeinen müssen sie die folgenden Anforderungen erfüllen [7], um nützlich zu sein:
**Hohe Leistung**: Verarbeitung großer Datenmengen in kurzer Zeit oder sogar in Echtzeit. Es geht um ungefähr 250 Datensatz/s für 10 bis 15 Tausend Touren.
**Skalierbar:** Skalierbar mit den Datenbedürfnissen des Kunden.
**Fehlertolerant:** Hosts und Netzwerkverbindungen werden insbesondere in Cloud-Umgebungen heruntergefahren. Der Rahmen muss für solche Fehler mit minimaler Unterbrechung des Datenflusses und sicherlich ohne menschliches Eingreifen widerstandsfähig sein.
**Unterstützung mehrerer Workloads:** Datenverarbeitungsframeworks müssen in der Lage sein, mehrere Anwendungen zu unterstützen, die verschiedene Aufgaben ausführen und unterschiedliche Workloads verarbeiten.
**Ausfallsicherheit:** die Verfügbarkeit der Infrastruktur soll gesichert werden.
Die meisten Organisationen können es sich nicht leisten, ein maßgeschneidertes Datenverarbeitungsframework für jeden einzelnen Geschäftsbedarf zu erstellen.

### 4.2.1 KPIs oder Metriken zur Überwachung der Systemspezifikation

Ein System, das die oben genannten Anforderungen unterstützen kann, wird wie fast jedes System ein komplexes System sein. Es wird aus vielen Teilen bestehen, die für das Host-Management, das Prozessmanagement, das Job-Management, das Datenrouting, die Ressourcenzuweisung, die Wiederherstellung usw. zuständig sind. Es wird aus einem Cluster von Maschinen bereitgestellt, die verschiedene Funktionen erfüllen und Abhängigkeiten zu anderen Systeme haben. Systeme wie Storm, Spark und Flink fallen unter diese Kategorie.
Aber all das liegt außerhalb dessen, was den Entwicklern am meisten am Herzen liegt, nämlich
die KPIs und Metriken zu identifizieren, um die obig eingegebenen Systemspezifikationen zu überwachen. Der Grund ist, dass sich ein Fehler in einer dieser Komponenten letztlich auf die Anwendungen auswirkt. Dies wirkt sich auf die Datenverarbeitungsflüsse aus, die sich wiederum auf das Unternehmen, die Marke, SLA (Service-Level-Agreement) mit Kunden oder den Umsatz auswirken können. Daher ist es wichtig zu verstehen, wie solche Systeme überwacht werden. Bei der Überwachung von Datenverarbeitungsframeworks im Allgemeinen kann man im Rahmen der Systemspezifikation intuitiv Fragen definieren, für die man die Antworten finden will und daraus KPIs und Metriken ableiten.

| **Fragen** | **Metriken** | **KPIs** |
|---|---|---|
| verarbeiten wir Daten mit der erwarteten Rate?<br><br>verarbeiten wir Daten innerhalb des erwarteten Zeitrahmens?<br><br>Fehler / Qualität: Gibt es Probleme mit den produzierten Daten? | Leistung | Durchsatz Ereignis/s<br><br>Latenzzeit<br><br>Anzahl der Replikation<br><br>Batch Size in Kafka:<br><br>Nachrichtengröße |

| | | |
|---|---|---|
| Sind Systeme, die Input in mein gewähltes Framework (wie Kafka) liefern, gesund?<br><br>Sind die Systeme, von denen meine Anwendung abhängig ist, z. B. andere API-Endpunkte, gesund?<br><br>Ist Zookeeper gesund, wenn das Streamingframework für die Jobkoordination davon abhängt? | Verfügbarkeit | Mehrere Node Cluster<br><br>- Anzahl von Broker im Cluster:<br>- Anzahl von Zookeeper im Cluster |
| Sind meine Anwendungs-KPIs innerhalb der normalen Betriebsparameter?<br><br>Sind die Leistungsindikatoren für die gegebene Streaming-Topologie normal? | Qualität | Ist-Durchsatzrat >= Soll-Durchsatzrat<br><br>Geringe Latenzzeit<br><br>Höhe Durchsatzrat |
| Funktionieren die wichtigsten Systemmetriken normal? | Systemzustand des Knotens | CPU-Auslastung<br><br>- Festplattennutzung<br><br>- Speichernutzung<br><br>- Netzwerknutzung |

## 4.3 Entwurf

Bei dem Teil unserer Arbeit versuchen wir euch einen Einblick in einen Streamingprototypen zu beschaffen. Es geht mehr darum die Grundlagenkomponente des Streamingsprototypen abzubilden. Diese wird uns später helfen, den ersten Prototyp zu definieren. Der folgende Abschnitt zeigt ein exemplarisches Streamingaufbaumodell aus der Literatur.

### 4.3.1 Architektur und Komponente Prototyp

Bei dem Streaming geht es darum, am meisten Daten in Echtzeit zu erfassen und zu bearbeiten.
Bei dieser Echtzeit-Datenverarbeitung erwarten wir generell, dass die Daten unendlich aus einer dritten Quelle kommen. Diese kommenden Daten werden erfasst und während der Echtzeitbearbeitung analysiert, ausgefiltert oder sortiert usw. Nach dem Schritt der Echtzeitverarbeitung werden die Output-Daten für andere Systeme bereitgestellt und am meisten im richtigen Format. Nach fachlicher Sicht könnte man es mit ETL vergleichen [8], hier werden Daten extrahiert, transformiert und im anderen System geladen. ETL ist ein Data Warehousing-Prozess, bei dem die Daten aus der Quelldatenbank mithilfe bestimmter Transformationsregeln über die

extrahierten Daten migriert werden, wobei diese transformierten Daten zurück in die Zieldatenbank geladen werden. Streaming stellt aber eine Erweiterung dieses Prozesses dar, indem die Daten nicht nur aus einer Datenbank kommen können. Es könnten beliebige Datenquellen oder Datenformate sein. Es ist einfach wichtig, dass die Daten ohne Unterbrechung erfasst und in Echtzeit bearbeitet werden. Die folgende Abbildung zeigt im Rahmen der Lösung der Streaming-Analytics-Verarbeitung im Fertigungsbetrieb die Kernkomponente einer Streaminganlage. Die Abbildung bildet gleichzeitig alle Teilkomponenten als auch die Aktivitäten in jeder Teilkomponente ab.

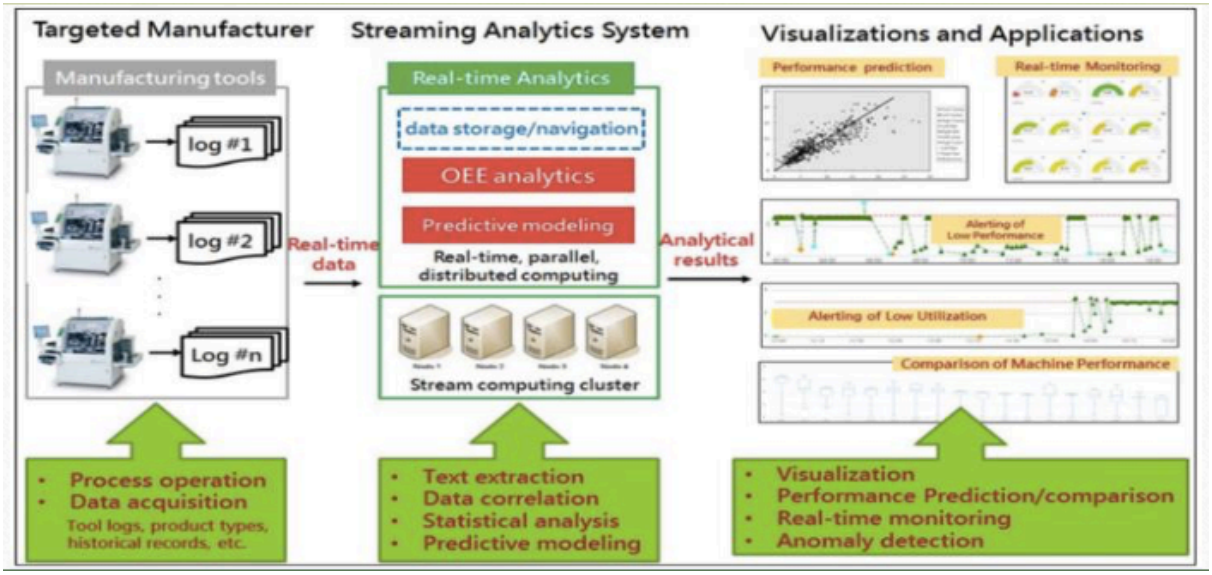


**Abbildung 1 Solution of streaming analytics processing in the manufacturing operation [9]**

### 4.3.2 Benchmarking über Systemkomponente

Die Verarbeitung von Streamingdaten hat aufgrund ihrer entscheidenden Auswirkungen auf eine Vielzahl von Anwendungsfällen wie Echtzeit-Handelsanalyse, Störungserkennung, intelligente Werbeplatzierung, Protokollverarbeitung und Metrikanalyse immer mehr Aufmerksamkeit auf sich gezogen. Um den wachsenden Anforderungen der Streaming-Datenverarbeitung gerecht zu werden, sind viele Computer-Engines entstanden. Es gibt jetzt mehrere Engines, die weithin angenommen werden, einschließlich Google Cloud Dataflow [10], Apache Storm, Apache Flink, Apache Spark, Apache Samba, Apache Apex und viele andere. Bei Yahoo ist zum Beispiel die Wahlplattform Apache Storm, die die intern entwickelte S4-Plattform [11] ersetzte. JStorm [12] wird stark genutzt, während Spark Streaming und Flink ebenfalls weit verbreitet sind.
Es besteht jedoch eine zunehmende Verwirrung darüber, welches Paket die besten Merkmale bietet und welches unter welchen Bedingungen besser funktioniert. Forscher haben Anstrengungen unternommen, um die Merkmale und die Leistung dieser populären Streaming-Verarbeitungsplattformen zu vergleichen [13], [14]. Die meisten dieser Arbeiten konzentrieren sich auf Feature-Vergleiche und die Auswertung von Durchsatz und Latenz durch Lichtgeschwindigkeitstests. Es geht für uns darum zu zeigen, wie man im Umfeld mit vielen Frameworks die Richtigen aussuchen kann.

#### 4.3.2.1 *Frameworks und Ihr Leistungseigenschaft aus Literatur*

Wir werden uns in diesem Abschnitt verschiede Benchmarking-Ergebnisse anschauen.
Dafür wollen wir die Information entnehmen, wie die verschiedenen Frameworks ihre Daten behandeln und wie ihre Leistungseigenschaften aussehen. Diese Eigenschaften sind für uns wichtig, da wir uns später durch einige Benchmarkings-Ergebnisse vergewissern können, dass wir das richtige Framework für unseren Streamingprototypen ausgesucht haben. Man kann in der folgenden Abbildung[16] feststellen, dass Apache Flink und Spark mit Sicht auf ihre Garantie eine niedrige Latenz und Durchsatz belegen können.

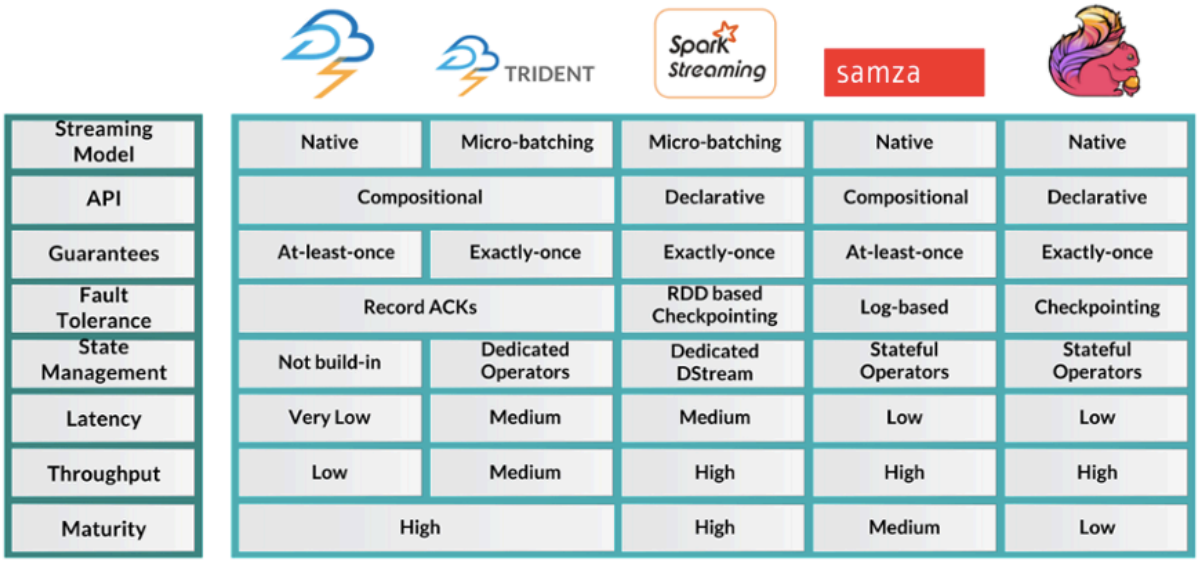


**Abbildung 2 Leistungseigenschaften Frameworks Benchmarking**

#### 4.3.2.2 *Erste Benchmarking Bericht*

Nachdem wir die obigen tabellarischen Leistungseigenschaften des jeweiligen Frameworks angeschaut haben, sehen wir nun ein implementiertes Benchmarking, um uns von obigen Architektureigenschaften überzeugen zu lassen. Uns ist hier wichtig zu wissen, dass die Leistungswerte aus der Literatur mit denen von den Autoren ausgeführten Benchmarking-Ergebnissen übereinstimmen. Unser erstes Benchmarking bezieht sich auf die Arbeit von Sanket Chintapalli, Derek Dagit, Bobby Evans, Reza Farivar u.v.m., die sich bereits mit dem Thema Streaming beschäftigt haben und die folgende Streaming-Benchmark entwerfen konnten [15]. In ihrer aufgebauten Pipeline gibt es eine Reihe von Werbekampagnen und eine Anzahl von Werbung für jede Kampagne. Die Aufgabe dieser Benchmark besteht darin, verschiedene JSON-Ereignisse von Kafka zu lesen, die relevanten Ereignisse zu identifizieren und relevante Ereignisse pro Kampagne in Redis zu speichern. Die Aktivitäten während des Streamings sind wie folgt in der Abbildung 3 definiert:
1) Die Nachrichten werden im JSON-Format von Kafka eingelesen.
2) Die JSON-Zeichenfolge werden deserialisiert.
3) Die irrelevanten Ereignisse werden ausgefiltert.
4) Das jeweilige Event (ad_id) wird mit korrektem Zeitstempel (Event_id) projektiert.

5) Jedes Ereignis (ad_id) wird mit der richtigen Kampagnen-ID vereinigt. Diese Information wird in Redis gespeichert.
6) Ein neues Ereignis wird in der Kampagne aufgenommen und mit seinem Zeitstempel in Redis gespeichert, in der das Fenster zuletzt in Redis aktualisiert wurde. Dieser Schritt muss spätere Ereignisse verarbeiten können.

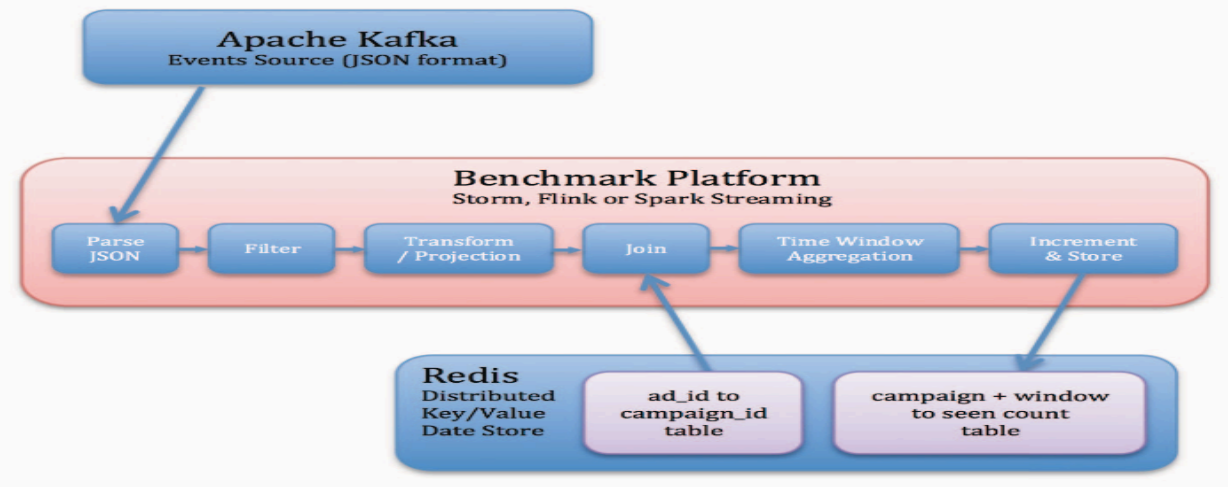


**Abbildung 3 Streaming Benchmark Design .**

Das Ergebnis dieses Benchmarkings hat ergeben, dass die Rate, mit der Kafka Datenereignisse in den Flink-Benchmark emittiert, von 50.000 Ereignissen / Sekunde bis 170.000 Ereignisse / Sekunde variiert.
Mit aktiviertem Acknowledgement hat Storm 0,11.0 mit 150.000 Ereignis/ s eine schreckliche Leistung gezeigt, was etwas besser als 0,10.0 ist, aber immer noch schlechter als Spark Streaming und Flink.
Spark Streaming konnte bei ausreichend hohem Durchsatz nicht mithalten, wenn die Stapelintervalleinstellung nicht geändert wurde. Bei 100.000 Ereignissen/ s stieg die Latenz stark an.
Die Ergebnisse in diesem Benchmarking zeigen, dass Storm und Flink bei einem ziemlich hohen Durchsatz eine wesentlich geringere Latenz aufweisen als Spark Streaming (dessen Latenz proportional zur Durchsatzrate ist). Die Begründung ist, dass Spark während seiner Leistung ziemlich empfindlich auf die Einstellung der Batch-Dauer reagiert.
Wenn wir uns Abbildung 3 ansehen, können wir sehen, dass Storm und Flink beide ziemlich linear reagieren. Dies liegt daran, dass diese beiden Systeme versuchen, ein eingehendes Ereignis zu verarbeiten, sobald es verfügbar ist. Auf der anderen Seite verhält sich das Spark-Streaming-System in einer schrittweisen Art und Weise, ein direktes Ergebnis seines Micro-Batch-Designs.

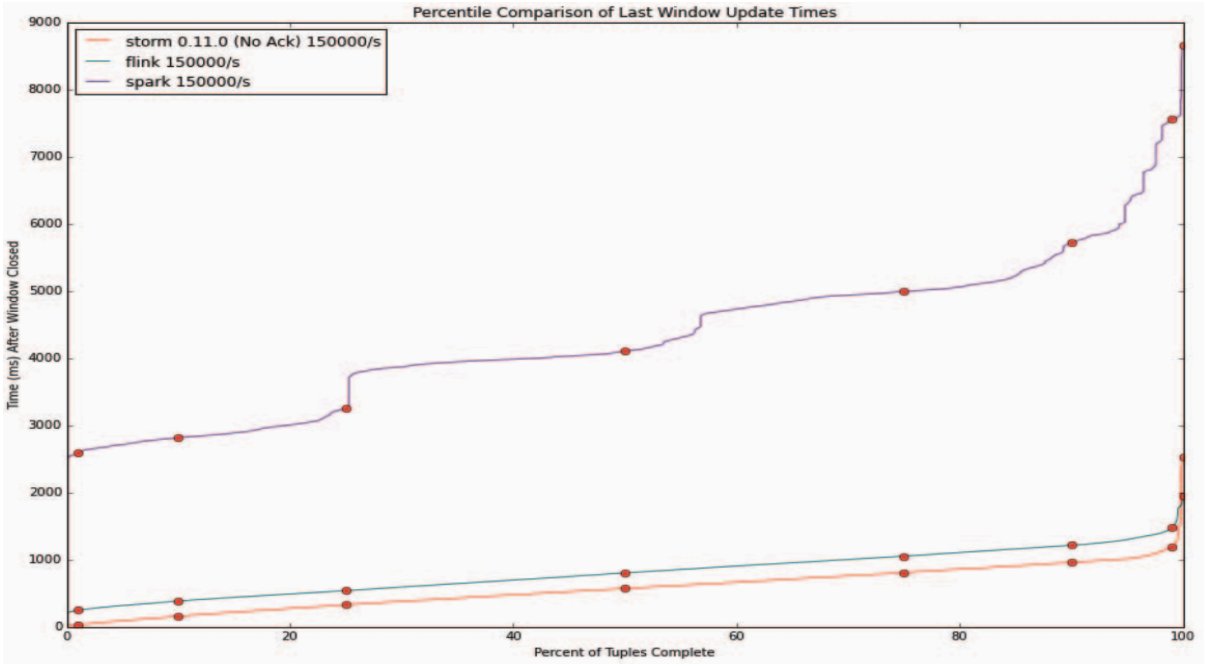


**Abbildung 4 Leistungsvergleich von Storm, Spark und Flink bei Fensterzeit der Datenverarbeitung[15]**

Der Durchsatz-Latenz-Graph für die verschiedenen Systeme ist vielleicht der aufschlussreichste, da er die Ergebnisse mit dieser Benchmark zusammenfasst. Wie in Abbildung 3 und 4 zu sehen ist, haben Flink und Storm eine sehr ähnliche Leistung. Von Spark Streaming, während es eine viel höhere Latenz hat, wird erwartet, dass es in der Lage ist, einen viel höheren Durchsatz durch Konfigurieren seines Stapelintervalls zu bewältigen.

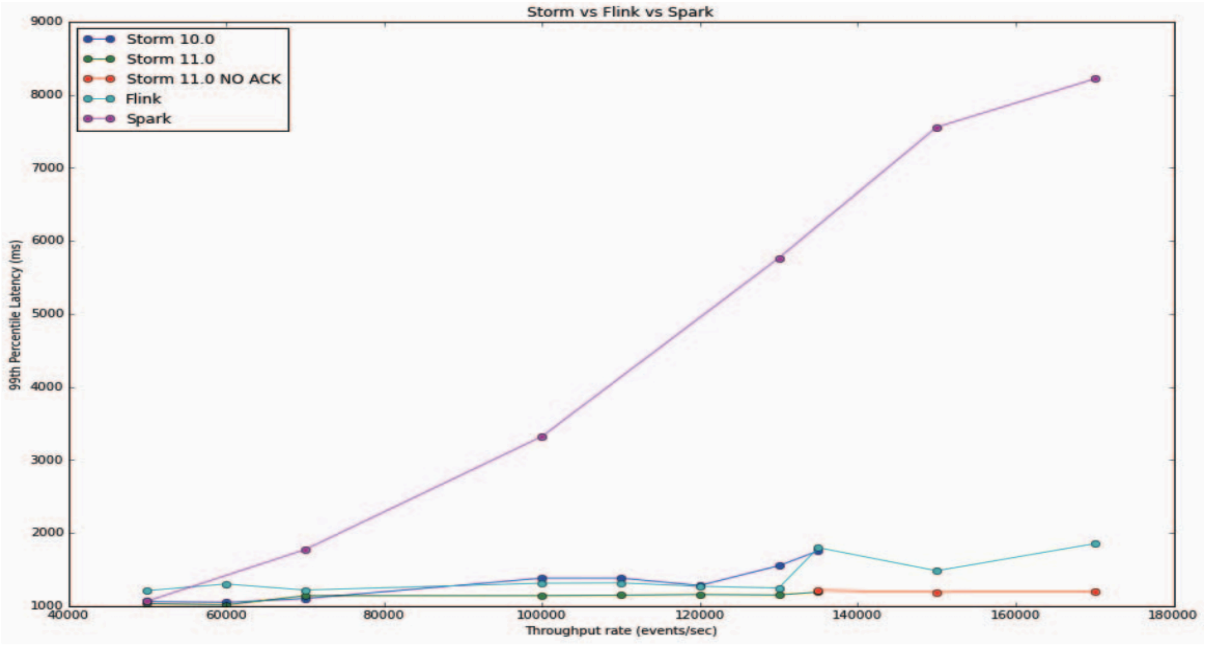


**Abbildung 5 Leistungsvergleiche von Storm, Flink und Spark Streaming im Hinblick auf die 99. Perzentil Latenz.[15]**

Wir stellen hier nach den beiden Schritten dieses Benchmarking fest, dass ein Streamingprototyp mit Apache Spark oder besser mit Apache Flink keine schlechte Idee wäre.

#### *4.3.2.3 Zweite Benchmarkingsbericht*

Nun gucken wir uns mal ein zweites Benchmarking bei anderen Konstellationen (Parametrisierung) an. Hier wird Storm und Flink verglichen. Es geht hier mehr um die Auswirkung der Technologie, die am meisten genutzt wird, um Datenstrom am Streamingframework zu binden: Apache Kafka. Wenn Sie nicht mit Kafka vertraut sind, handelt es sich um ein skalierbares, fehlertolerantes Messaging-System, mit dem Sie verteilte Anwendungen erstellen und webbasierte Internetunternehmen wie LinkedIn, Twitter, AirBnB und viele andere unterstützen können. Die Komponente kann falsch auswirken, wie sein strategischer Ansatz vorgesehen ist. Die Autoren behaupten, dass sich Kafka bei einem falschen strategischen Ansatz auf einzelne Komponenten oder Verarbeitungsschritte negativ auswirken kann, die die Latenz der Streaminganlagen erhöhen und somit den Durchsatz verkleinern können. Paul Le Noac'h, Alexandru Costan und Luc Bouge´ konnten in ihrem Beitrag zeigen, wie sich ein anderes Vorgehen bei der Leistungsoptimierung auf die gesamte Streamverarbeitung auswirken kann. Wie in Abbildung 6 gezeigt, ist die Verarbeitungsleistung (d. H. Apache Flink) stark benachteiligt, wenn Kafka für die Datenquelle verwendet wird, im Gegensatz zu eigenständigen Bereitstellungen. Laut der Abbildung 6 kann Apache Flink einen Durchsatz von 15 Millionen Ereignissen/Sekunde haben, wenn andere Parameter berücksichtigt werden[17].

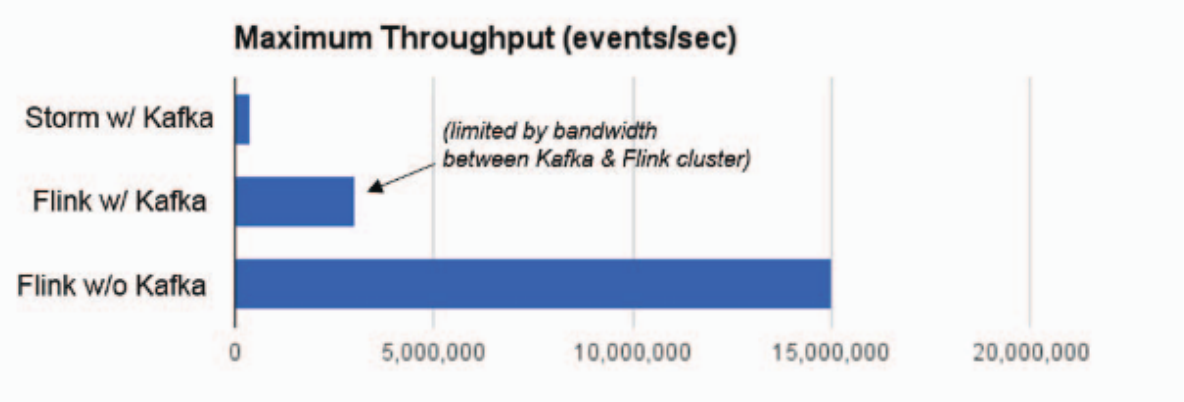


**Abbildung 6 Leistungsvergleich von Spark, Flink und Storm**

Dies deutet daraufhin, dass die Aufnahmephase zu einem Engpass in der gesamten Streaming-Pipeline werden kann. Die Gründe für die Leistungsverschlechterung können unterschiedlich sein: die Bandbreite, Parameterabstimmung, Anzahl der verwendeten Knoten usw. können für diese Einschränkung der Leistung verantwortlich sein. In dem Teil wollten wir erwähnen, dass es wichtig ist, die Auswirkungen der verschiedenen Parametereinstellungen auf Kafkas Gesamtleistung zu identifizieren. Somit stellen wir nach dem zweiten Benchmarking-Ergebnis fest, dass Flink die Lösung für uns wäre, unter einer guten Verwendung von Kafka, wenn wir die Daten so schnell wie möglich verarbeiten wollen.

# 5 Implementierung / Ergebnis

Generell ist es nicht einfach, sich für eines der beiden Tools zu entscheiden. Wenn man Real-Time Streaming braucht, aber auch Batch Processing, ist Flink eine ausgezeichnete Wahl. Ansonsten ist die Frage relevant, ob man auf die etablierte Technik setzt oder ein Early Adopter sein möchte.
Für Spark spricht, dass es stark auf dem Markt vertreten und in vielen Hadoop-Distributionen integriert ist und mehrere große Firmen hinter sich vereinen kann. Innovationen kommen mit jedem neuen Release dazu. So ist nicht zu befürchten, dass Spark bald die Puste ausgeht. Es gibt aber durchaus einige konzeptionelle Altlasten, die das Framework mit sich herumträgt.
Flink scheint nicht nur dort auf einem solideren Fundament aufgebaut zu sein, auch die API, das Monitoring, in vielen Bereichen wirkt Flink einfach frischer. Durch seine europäischen Wurzeln kämpft es aber durchaus um Akzeptanz bei den großen, in Nordamerika ansässigen Firmen. Somit haben wir zwei Basiskomponenten für unseren Streamingprototyp im Logistikbereich.
Unser Prototyp wird aus drei Komponenten bestehen:
Als erste Komponente setzen wir Apache Kafka, für den breiten Datenstrom und zwar aufgrund von diversen Arten von Daten: strukturierte (Paketinformationen(Scanner-daten)), semistrukturierte (E Mail oder SMS von Zusteller) und unstrukturierte (Tonaufnahme oder Videos von Zusteller, Sensor Daten von Lieferwagen)Daten, die integriert werden müssen.
Danach schließen wird die rechnerischen Frameworks (Apache Spark und Flink) mit Hilfe von Verbraucher-API von Kafka ein und können dann folgende ETL-Tätigkeiten ausführen: Daten ausfiltern, transformieren,

aggregieren usw. Hier konnte eine Memorydatenbank in Einsatz kommen, wenn es den Bedarf gibt, manche Objekte einige Minuten zu speichern.
Die Daten können nach der rechnerischen Operation in einer gezielten Datenbank wie MongoDB in einem Dockercontainer gespeichert werden, da MongoDB im allgemeinen für sofortige Abfragen verwendet werden kann und unterstützt in einfacher Weise die Geovisualisierung von Daten. Da wir die Geokoordinaten von Paketdaten in Echtzeit auf eine Landkarte anzeigen lassen wollen, wissen wir, dass er den schnellen Zugriff auf die geodaten ermöglicht. mehr, in welcher Region des Landes der betroffene Paketdienstleister aktiv ist. Das kann auch dem Paketdienstleiter helfen, seine Zustellungsstrategie zu steuern. Die Abbildung 7 zeigt uns vollständig, wie die obig beschriebenen Komponenten zusammengebaut sind. Wie haben den ersten Schritt für die Datenerfassung, den nächsten Schritt für die Echtzeitverarbeitung und zum Schluss die Daten den Schritt für die Datenvisualisierung.

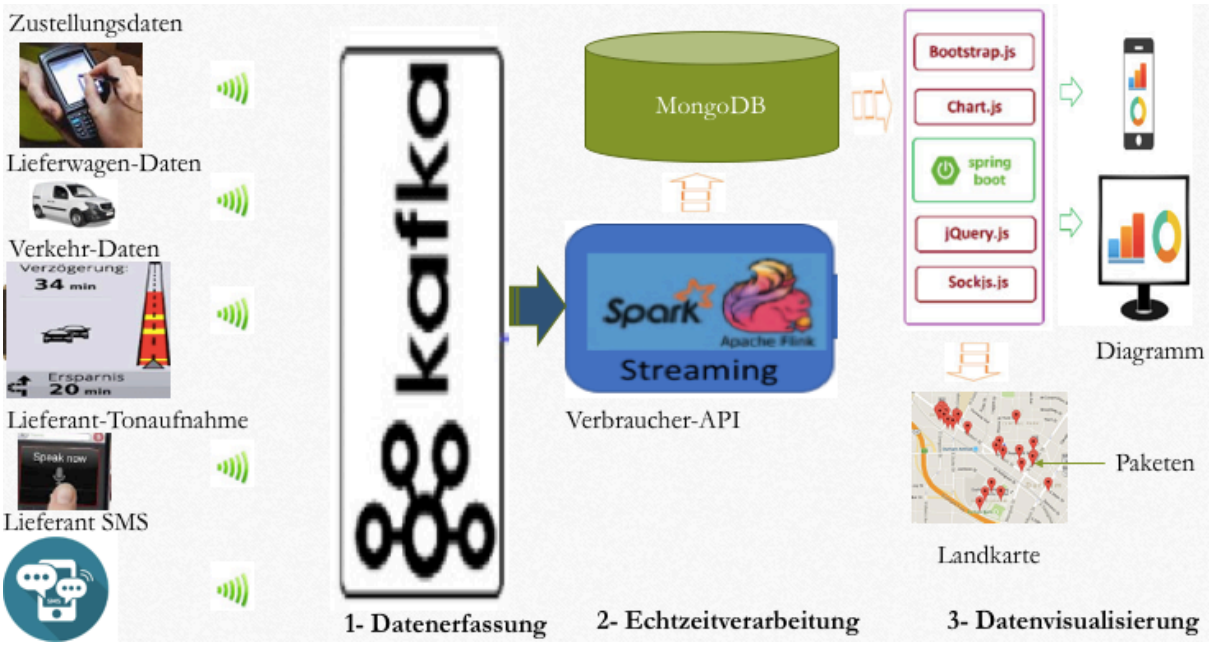


**Abbildung 7 unsere Streaming Prototyp**

# 6 Betrieb und Wartung

Es geht darum unser Prototyp im Betrieb zu bringen, um zu testen, ob er Protoytp stabil ist.
Baum genau Beobachtung dieser Prototyp, stellen wir während des Tests schnell fest, dass die Ausfallsicherheit nicht abgedeckt ist. Dieser Fall kann entstehen wenn man einziger Kafka Broker verwendet und der ausfällt. Wir decken die Lücke ab, in dem wir unseren Prototyp mit einem Kafka und Zookeeper Multi Node Cluster ergänzen, wobei jeder Rechner folgende Eigenschaften hat:

Xeon E3-1230-V2@3.30GHz CPU (4 cores w/ hyperthreading) and 32 GB. Die Daten können jetzt schnell auf alle Broker in unterschiedliche Server repliziert werden, was eine gute Lastverteilung für die Server ist, bei mehrere Tausend interessierte Verbrauche, die eine Daten haben wollen.

wie man in Abbildung 7 sieht.

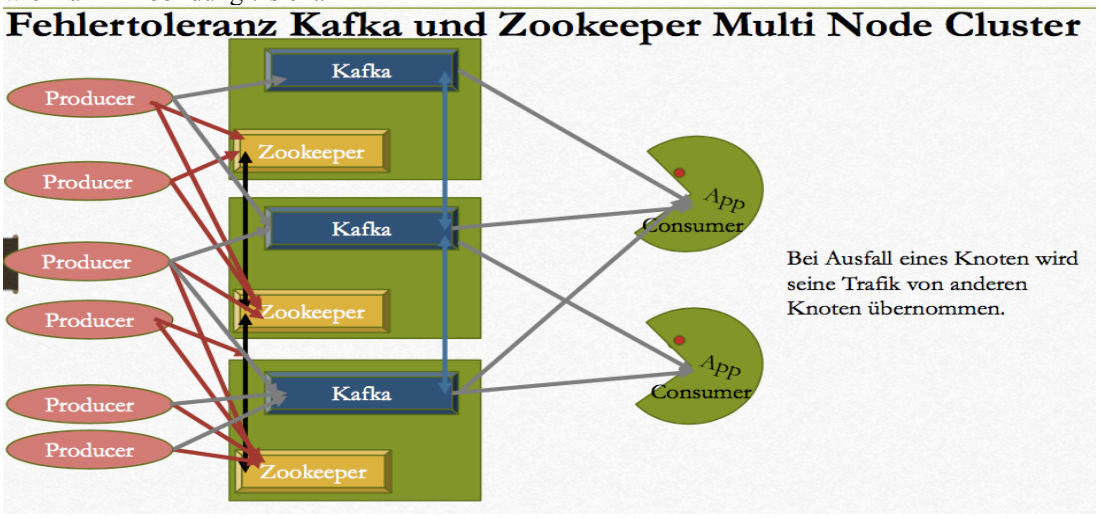


**Abbildung 8 Kafka und Zookeeper Cluster multti-Node**

Nach dem letzten Schritt sind wir immer noch nicht zufrieden, kafka kann noch besser optimiert werden. Das Problem ist, dass Kafka den Standardpartitionierungsmechanismus verwendet, um Daten zwischen Partitionen zu verteilen, aber im Falle eines Standardpartitionierungsmechanismus ist es möglich, dass einige Partitionen größer als andere sind. Nehmen wir an, wir haben 5 Partitionen und 40 GB Daten in Kafka eingefügt, dann könnte die Datengröße jeder Partition wie folgt aussehen:

Partition0 → 10 GB
Partition1 → 8 GB
Partition2 → 6 GB
Partition3 → 9 GB
Partition4 → 11 GB

Um also Daten einheitlich zu verteilen, müssten wir eine round-robin-eigene Partitionsklasse implementieren.
So können wir immer weiter versuchen eine Optimal Antworte zu geben, bei dein Fragen die im Abschnitt oben erstellt wurden, um Metriken-und KPIS- Werte der Systemspezifikation zu bewerten.

# 7 Schluss

Wir haben uns in dieser Arbeit mit dem Thema Streaming im Bereich der Logistik beschäftigt. Es ging darum, für die Anforderung der Logistik einen passende Streamings Prototypen zu definieren. Wir haben zu dem Zweck gezeigt, welche Anforderungen die Paketdienstleister generell haben, in dem wir die Informationen den vergangenen Onlineberichten zusammengefasst haben. Danach haben wir uns die Eigenschaften von verschiedenen Frameworks angeguckt, die gerade im Big Data Bereich im Thema Streaming, wie Apache Spark, Storm und Flink sehr präsent sind. Wir konnten uns danach für zwei Frameworks entscheiden, nachdem wir festgestellt haben, dass zwei unterschiedliche Benchmarking von unterschiedlichen Autoren in unterschiedlichen Arbeiten zu denselben Ergebnissen geführt haben und zwar die Priorisierung einer Streaming-Implementierung mit Hilfe von Apache Flink oder Apache Spark. Somit konnten wir einen Prototyp auf Basis von Apache Kafka, Spark und Flink definieren. Der Punkt, der aber offen bleibt, ist, ob man sich für eine stabile Technologie wie Spark entscheidet oder eine neue wie Flink, die noch manchmal ihre Grenzen aufzeigen kann. Wir raten zu Apache Flink für das Projekt, wo eine geringe Latenzzeit von Bedeutung ist.

[1] Abdelkarim Ben Ayed, u.a , “Big Data Analytics for Logistics and Transportation ”, International Conference on Advanced logistics and Transport (IEEE’ ICAl T 2015)
[2] International Organization for Standardization, “Industrial automation systems - requirements for enterprise-reference architectures and methodologies,” 2000.

[3] Mohiuddin Solaimani, Mohammed Iftekhar, Latifur Khan u.a, “Spark-based anomalie Detection Over Multi-Source VMware Performance Data In Real-time”, Symposium on Computational Intelligence in Cyber Security (IEEE’ CICS 2014)
[4] M. O’Neill, and P. Young, Developing Overall Equipment Effectiveness Metrics for Prototype Precision Manufacturing, PhD Thesis, Dublin City University, January 2011.
[5] https://www.ksta.de/wirtschaft/paketdienste-deutsche-verschicken-10-millionen-sendungen-pro-tag-27835854 online (19/08/2018)

[6] http://www.fnp.de/nachrichten/politik/Frust-ueber-verlorene-Pakete-waechst;art673,2870366 online (19/08/2018)

[7] https://www.opsclarity.com/monitoring-real-time-data-pipelines/ online(19/08/2018)
[8] Kunal Sharma, Dr. Vahida Attar, “Generalized Big Data Test Framework for ETL Migration”, International Conference on Computing, Analytics and Security Trends (IEEE’ CAST 2016)
[9] Yi-Hsin Wu and Sheng-De Wang, Li-Jung Chen and Cheng-Juei Yu, “Streaming Analytics Processing in Manufacturing Performance Monitoring and Prediction”, International Conference on Big Data (IEEE’ BIGDATA 2017)
[10] Akidau, R. Bradshaw, C. Chambers, S. Chernyak, R. J. Ferna ´ndez-Moctezuma, R. Lax, S. McVeety, D. Mills, F. Perry, E. Schmidt, et al., “The dataflow model: a practical approach to balancing correctness, latency, and cost in massive-scale, unbounded, out-of-order data process- ing,” Proceedings of the VLDB Endowment, vol. 8, no. 12, pp. 1792– 1803, 2015.
[11] L. Neumeyer, B. Robbins, A. Nair, and A. Kesari, “S4: Distributed stream computing platform,” in Data Mining Workshops (ICDMW), 2010 IEEE International Conference on, pp. 170–177, IEEE, 2010.
[12] “JStorm Project.” http://github.com/alibaba/jstorm/.
[13] “Which Stream Processing Engine: Storm, Spark, Samza, or Flink?.” http://www.altaterra.net/blogpost/288668/225612/Which-Stream- Processing-Engine-Storm-Spark-Samza-or-Flink/.
[14] “High-throughput,low-latency,andexactly-oncestreamprocessingwith Apache Flink..” http://data-artisans.com/high-throughput-low-latency- and-exactly-once-stream-processing-with-apache-flink/.
[15] Sanket Chintapalli, Derek Dagit, Bobby Evans u.a, “Benchmarking Streaming Computation Engines: Storm, Flink and Spark Streaming”, International Parallel and Distributed Processing Symposium Workshops (IEEE’ 2016)

[16] https://www.cakesolutions.net/teamblogs/topic/apache-storm online (15.08.2018)

[17] Paul Le Noac’h, Alexandru Costan und Luc Bouge´, “A Performance Evaluation of Apache Kafka in Support of Big Data Streaming Applications”, International Conference on Big Data (IEEE’ BIGDATA 2017)