

BIG DATA

amazon.com[®]



Google

facebook

Lars Briem

(briem.lars@googlemail.com)

Duale Hochschule Baden Württemberg - Standort Karlsruhe

Motivation

Lambda Architektur

Datenmodell

Batch Schicht

Abfrage Schicht

Echtzeit Schicht

Fazit

Literatur / Quellen

Motivation

Lambda Architektur

Datenmodell

Batch Schicht

Abfrage Schicht

Echtzeit Schicht

Fazit

Literatur / Quellen

Webanalyse

Gegeben eine kleine Webanwendung mit verschiedenen Seiten. Führen Sie eine kleine Statistik, welche Seite wie oft aufgerufen wurden.

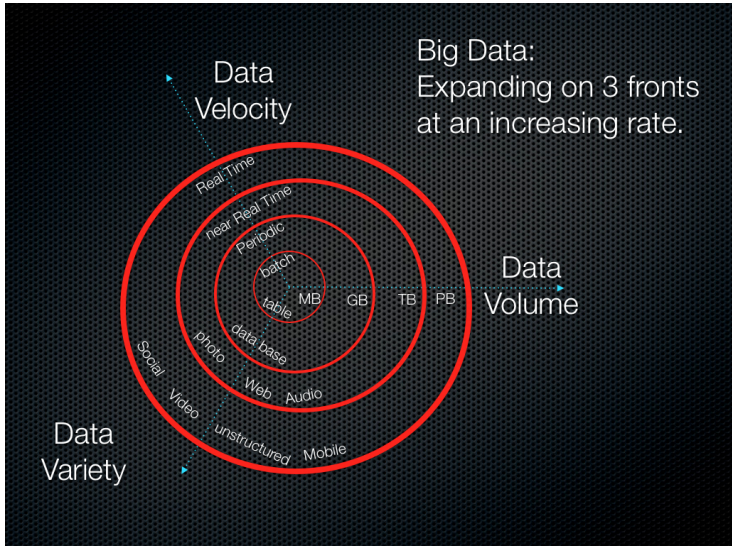
Einfachste Lösungsmöglichkeit?

Probleme im Beispiel

- ▶ Komplexität des Systems
- ▶ Inkrementelles System
 - ▶ Betriebsbedingte Komplexität
 - ▶ Konsistenz erzielen
- ▶ Fehlende Fehlertoleranz

⇒ Wie können Besucher anstelle von Seitenaufrufen gezählt werden?

Was ist Big Data



Anforderungen an ein Big Data System

- ▶ Robustheit / Fehlertoleranz
 - ▶ Ausfallsicherheit
 - ▶ Konsistenz
 - ▶ Menschliche Fehler
- ▶ Abfragen und Aktualisierungen in kurzer Zeit
 - ▶ Unterschiedliche Wartezeiten
- ▶ Skalierbarkeit
 - ▶ Linear
- ▶ Generalisierung

Anforderungen an ein Big Data System

- ▶ Erweiterungsmöglichkeit
- ▶ Ad hoc Abfragen
- ▶ Minimaler Wartungsaufwand
 - ▶ Einfache Komponenten
 - ▶ Keine Komplexität im Anwendungskern
- ▶ Möglichkeit zur Fehlersuche
 - ▶ Rückverfolgung wie Werte entstehen

Motivation

Lambda Architektur

Datenmodell

Batch Schicht

Abfrage Schicht

Echtzeit Schicht

Fazit

Literatur / Quellen

Grundgedanke

Gegeben: Menge von Daten

Gesucht: Antwort auf eine Abfrage

Grundgedanke

Gegeben: Menge von Daten

Gesucht: Antwort auf eine Abfrage

Abfrage = Funktion(alle Daten)

Grundgedanke

Gegeben: Menge von Daten

Gesucht: Antwort auf eine Abfrage

Batch View = Funktion(alle Daten)

Abfrage = Funktion(Batch View)

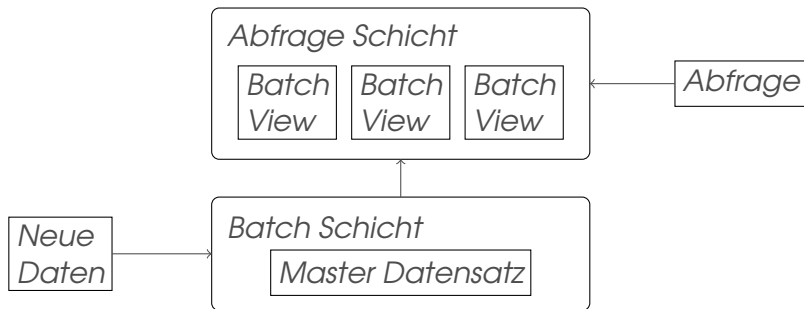
Grundgedanke

Gegeben: Menge von Daten

Gesucht: Antwort auf eine Abfrage

Batch View = Funktion(alles Daten)

Abfrage = Funktion(Batch View)



Eigenschaften die Batch + Abfrage Schicht erfüllen

- ▶ Robustheit / Fehlertoleranz
- ▶ Skalierbarkeit
- ▶ Generalisierung
- ▶ Erweiterungsmöglichkeit
- ▶ Ad hoc Abfragen
- ▶ Minimaler Wartungsaufwand
- ▶ Möglichkeit zur Fehlersuche

Probleme

- ▶ Berechnung von Batch View zeitintensiv
- ▶ Späte Berücksichtigung neuer Daten

Grundgedanke

Gegeben: Menge von Daten

Gesucht: Antwort auf eine Abfrage

Batch View = Funktion(alle Daten)

Abfrage = Funktion(Batch View)

Grundgedanke

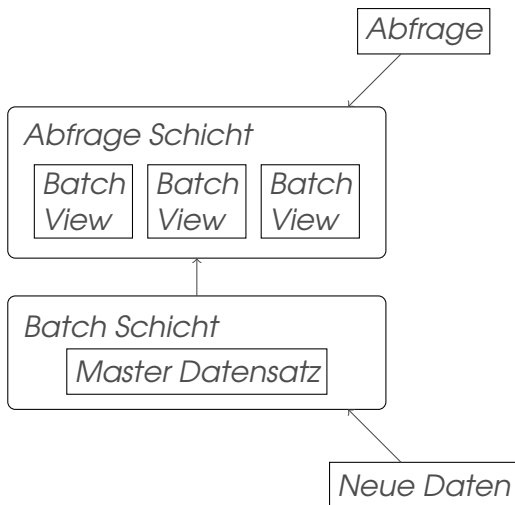
Gegeben: Menge von Daten

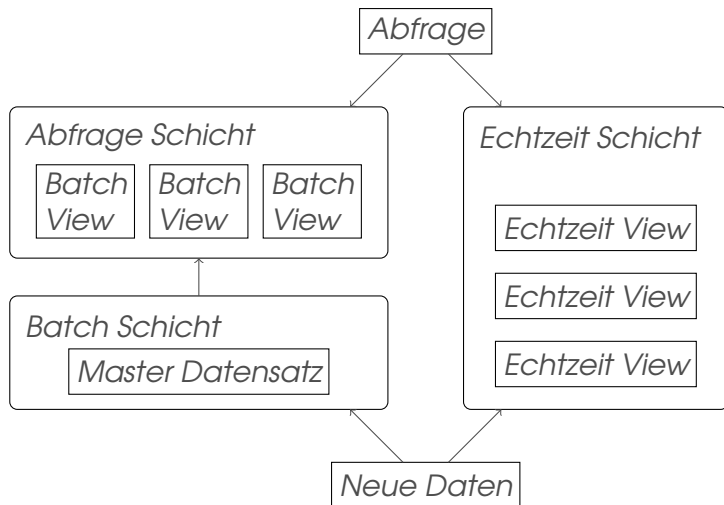
Gesucht: Antwort auf eine Abfrage

Batch View = Funktion(alle Daten)

Echtzeit View = Funktion(Echtzeit View, neue Daten)

Abfrage = Funktion(Batch View, Echtzeit View)





Lambda Architektur

Datenmodell

Batch Schicht

Abfrage Schicht

Echtzeit Schicht

Eigenschaften von Daten

Informationen	Ansammlung von Wissen
Daten	Nicht ableitbare Information. „Axiom“ der Datenverarbeitung.
Abfragen	Abfragen basierend auf allen Daten.
Views	Aus den Daten abgeleitete Informationen. Spezialisierte Sicht auf die Daten.

Eigenschaften von Daten

Informationen	Ansammlung von Wissen
Daten	Nicht ableitbare Information. „Axiom“ der Datenverarbeitung.
Abfragen	Abfragen basierend auf allen Daten.
Views	Aus den Daten abgeleitete Informationen. Spezialisierte Sicht auf die Daten.

3

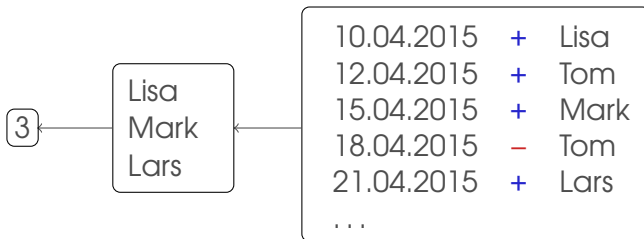
Eigenschaften von Daten

Informationen	Ansammlung von Wissen
Daten	Nicht ableitbare Information. „Axiom“ der Datenverarbeitung.
Abfragen	Abfragen basierend auf allen Daten.
Views	Aus den Daten abgeleitete Informationen. Spezialisierte Sicht auf die Daten.



Eigenschaften von Daten

Informationen	Ansammlung von Wissen
Daten	Nicht ableitbare Information. „Axiom“ der Datenverarbeitung.
Abfragen	Abfragen basierend auf allen Daten.
Views	Aus den Daten abgeleitete Informationen. Spezialisierte Sicht auf die Daten.



Wie **roh** sollen die Daten sein?

- ▶ So roh wie möglich
- ▶ Entfernen von nicht benötigten Daten
 - ▶ JavaScript
 - ▶ Stylesheet
 - ▶ ...
- ▶ Semantische Normalisierung reduziert Möglichkeiten
- ▶ Einfaches Parsen ist möglich

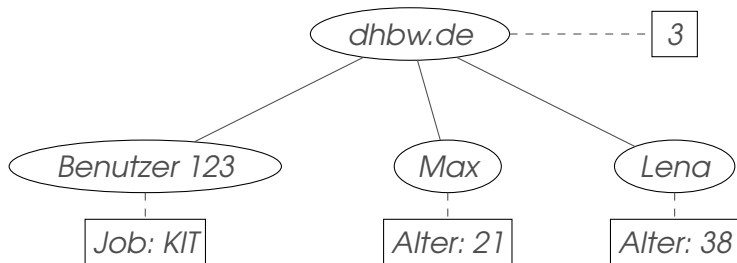
Unveränderlichkeit der Daten

- ▶ Kein „Update“ wie in inkrementellen Systemen
- ▶ Toleranz gegenüber menschlichen Fehlern
 - ▶ Menschen machen Fehler
 - ▶ Fehler können Daten zerstören
 - ▶ Unveränderliche Daten nicht löschtbar
- ▶ Einfaches Modell
 - ▶ Keine Indexierung nach Änderung (da keine Änderung)
 - ▶ Nur Möglichkeit zum Hinzufügen notwendig
- ▶ Zeitangabe bei jedem Eintrag (Timestamp)

- ▶ Fakt
 - ▶ Atomar
 - ▶ Mit Zeitangabe
 - ▶ Identifizierbar
- + Vergangenheit abfragbar
- + Fehlertolerant
- + Teilweise Informationen ausreichend
- + Kombination aus normalisierten und denormalisierten Daten

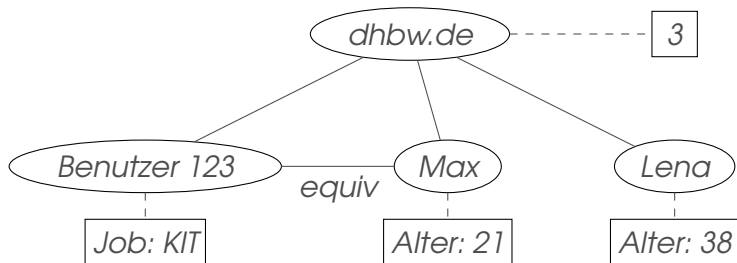
Fakten basiertes Datenmodell

⋮	⋮	⋮
Max	besucht dhbw.de	2010-12-11 09:34:15
Lena	besucht dhbw.de	2010-12-12 12:15:31
Benutzer 123	besucht dhbw.de	2010-12-13 21:43:06



Fakten basiertes Datenmodell

⋮	⋮	⋮
Max	besucht dhw.de	2010-12-11 09:34:15
Lena	besucht dhw.de	2010-12-12 12:15:31
Benutzer 123	besucht dhw.de	2010-12-13 21:43:06
Benutzer 123	Anmeldung (Max)	2010-12-13 21:58:06



Lambda Architektur

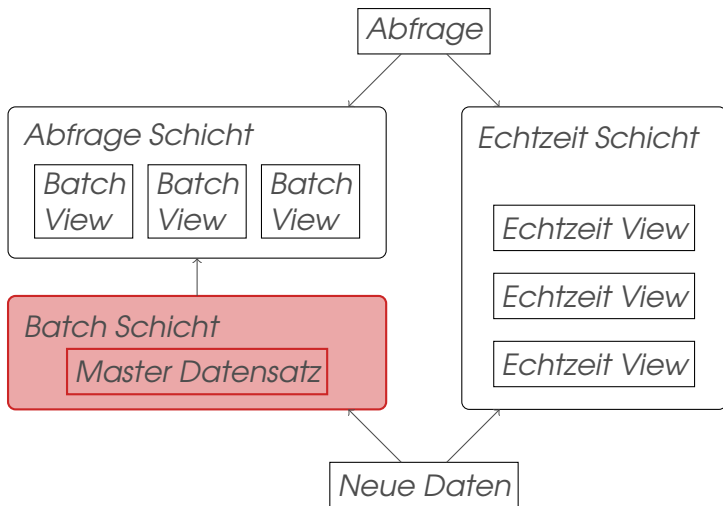
Datenmodell

Batch Schicht

Abfrage Schicht

Echtzeit Schicht

Lambda Architektur - Batch Schicht



Batch Schicht - Aufgabe

- ▶ Verwaltung Master Datensatz
- ▶ Berechnung über alle Daten
- ▶ Berechnung von Batch Views

Anforderungen an den Datenspeicher

- ▶ Anhängen neuer Daten
- ▶ Kein Verändern bestehender Daten
- ▶ Skalierbar
- ▶ Paralleles Lesen

Komponenten

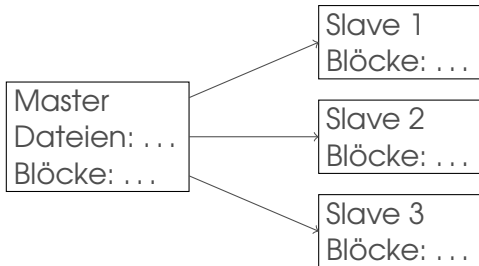
- Master Knoten zur Speicherung von Metadaten zu den Dateien (Namenode)
- Slave Knoten enthält die Blöcke der einzelnen Dateien (Datanode)

Verteilte Dateisysteme am Beispiel von HDFS

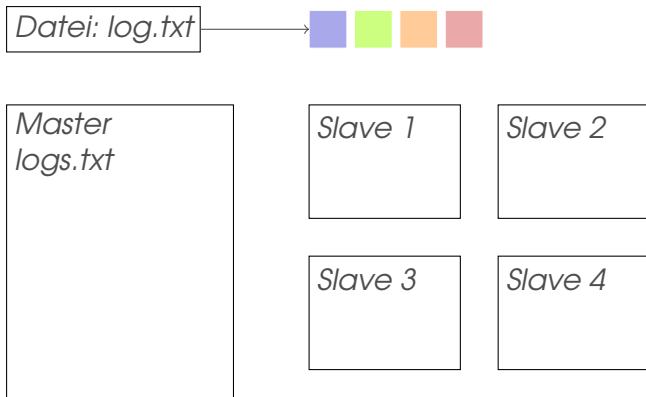
Komponenten

Master Knoten zur Speicherung von Metadaten zu den Dateien (Namenode)

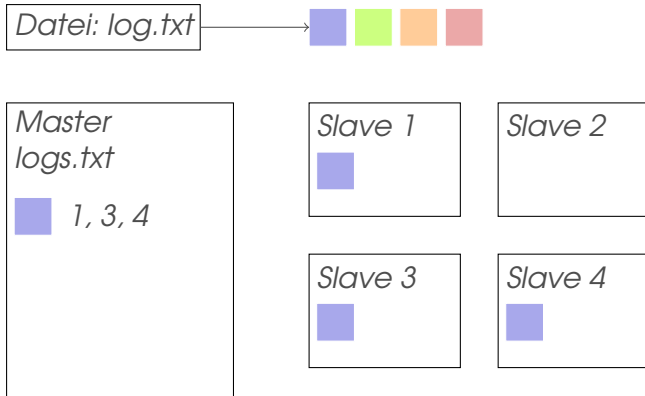
Slave Knoten enthält die Blöcke der einzelnen Dateien (Datanode)



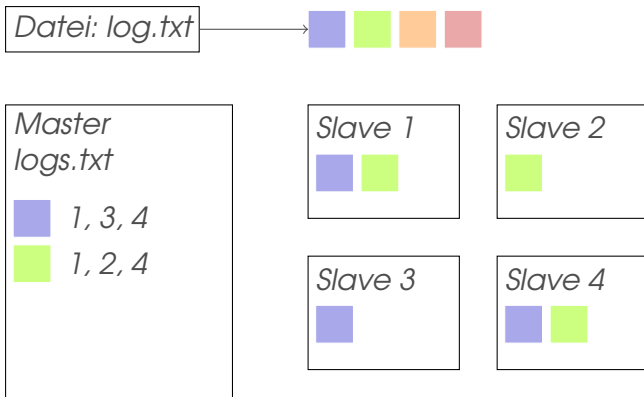
Speichern von Dateien



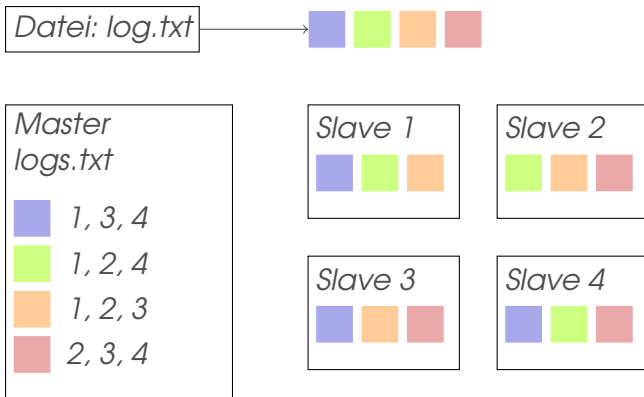
Speichern von Dateien



Speichern von Dateien



Speichern von Dateien




Zugriff auf Block:



Master
logs.txt

 1, 3, 4

 1, 2, 4

 1, 2, 3

 2, 3, 4

Slave 1



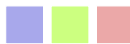
Slave 2



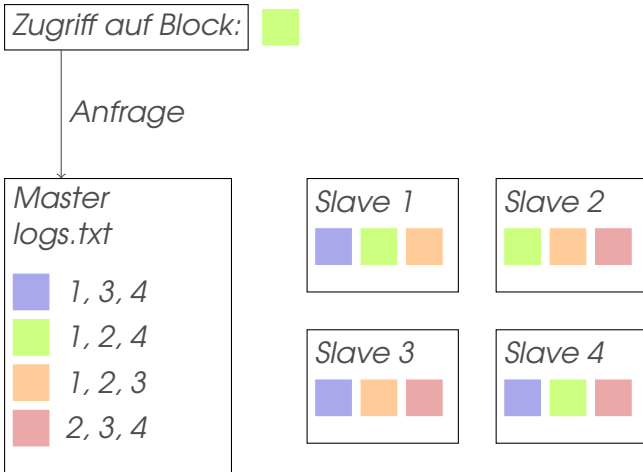
Slave 3



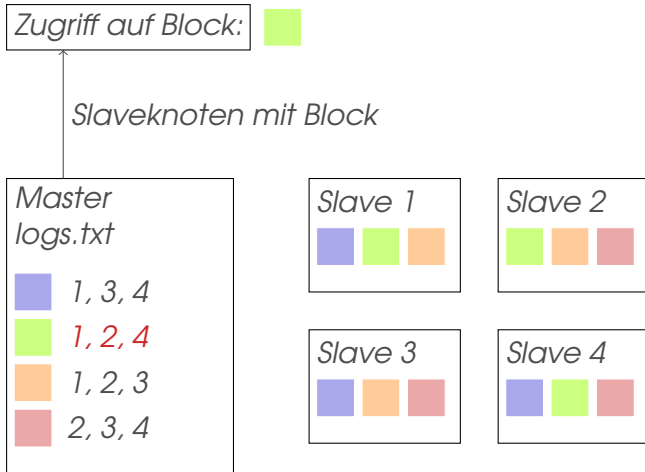
Slave 4



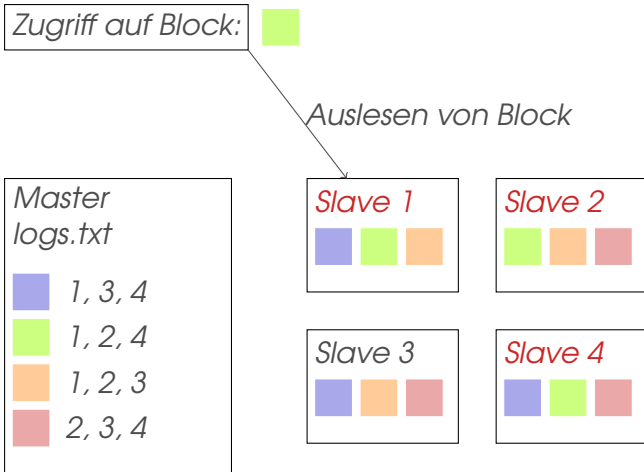
Lesen von Dateien



Lesen von Dateien



Lesen von Dateien



Verteilte Dateisysteme

- ▶ Dateien werden in Blöcke aufgeteilt
- ▶ Blöcke werden auf verschiedenen Knoten gespeichert
 - ▶ Typischerweise 3
- + Skalierbarkeit
- + Paralleles Verarbeiten
- + Fehlertoleranz
- Komplexer als lokales Dateisystem

Vertikale Teilung der Daten

- ▶ Berechnung in Batch Schicht aufwendig
- ▶ Nicht alle Berechnungen benötigen alle Daten
- ▶ Steigerung der Effizienz durch sinnvolle Filterung
- ▶ Baumstruktur optimiert Zugriffszeit

Vertikale Teilung der Daten

- ▶ Berechnung in Batch Schicht aufwendig
- ▶ Nicht alle Berechnungen benötigen alle Daten
- ▶ Steigerung der Effizienz durch sinnvolle Filterung
- ▶ Baumstruktur optimiert Zugriffszeit

Anmeldedaten einer Website

- ▶ Verzeichnis pro Tag / Stunde
- ▶ Je nach Größe mehrere Dateien

Verzeichnisstruktur

- ▶ Logins
 - ▶ 2015-02-09
 - ▶ logins-2015-02-09-part1.txt
 - ▶ logins-2015-02-09-part2.txt
 - ▶ logins-2015-02-09-part3.txt
 - ▶ 2015-02-10
 - ▶ logins-2015-02-10-part1.txt
 - ▶ logins-2015-02-10-part2.txt
 - ▶ logins-2015-02-10-part3.txt
 - ▶ logins-2015-02-10-part4.txt
 - ▶ logins-2015-02-10-part5.txt

Neuberechnung bei neuen Daten

Möglichkeiten zur Aktualisierung der Batch Views

- ▶ Komplette neu berechnen
- ▶ Inkrementell neu berechnen

Performance

- ▶ Komplette Berechnung benötigt mehr Zeit
- ▶ Inkrementelle Berechnung benötigt mehr Speicher

Neuberechnung bei neuen Daten

Toleranz gegenüber menschlichen Fehlern

- ▶ Bug beheben und komplette Berechnung starten
- ▶ Bug beheben und Fehler in den berechneten Daten suchen

Welche Variante?

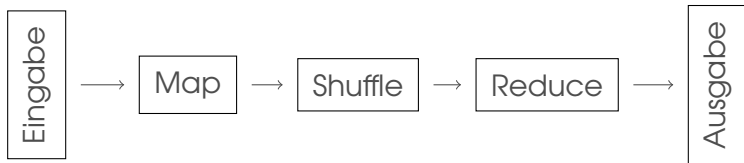
- ▶ Algorithmus zur kompletten Neuberechnung immer notwendig
- ▶ Zusätzlich evtl. noch inkrementelle Version

Pause

Pause

MapReduce Konzept

- ▶ Skalierbare Verarbeitung
- ▶ Aufteilung in Phasen
 - ▶ Parallele Verarbeitung innerhalb der Phasen



MapReduce Phasen

- Map Umwandlung der Eingabe in Zwischenergebnisse
- Shuffle Sortieren und Gruppieren der Daten nach dem Key
- Reduce Reduzieren der Zwischenergebnisse mit gleichem Key auf weniger Werte (typ. 1)

$$\begin{aligned}\text{map:} & \quad (k_1, v_1) \rightarrow \text{list}(k_2, v_2) \\ \text{shuffle:} & \quad \text{list}(k_2, v_2) \rightarrow (k_2, \text{list}(v_2)) \\ \text{reduce:} & \quad (k_2, \text{list}(v_2)) \rightarrow \text{list}(k_3, v_3)\end{aligned}$$

Beispiel: Zählen von Wörtern

Datei

Ein Mann läuft.

⋮

Eine Person kauft eine Lampe.

⋮

Ein Kind spielt.

Beispiel: Zählen von Wörtern

Datei

Ein Mann läuft.
:
Eine Person kauft eine Lampe.
:
Ein Kind spielt.

Map
Eingabe

0, *Ein Mann läuft.*
101, *Eine Person kauft eine Lampe.*
234, *Ein Kind spielt.*

Beispiel: Zählen von Wörtern

0,	Ein Mann läuft.
101,	Eine Person kauft eine Lampe.
234,	Ein Kind spielt.

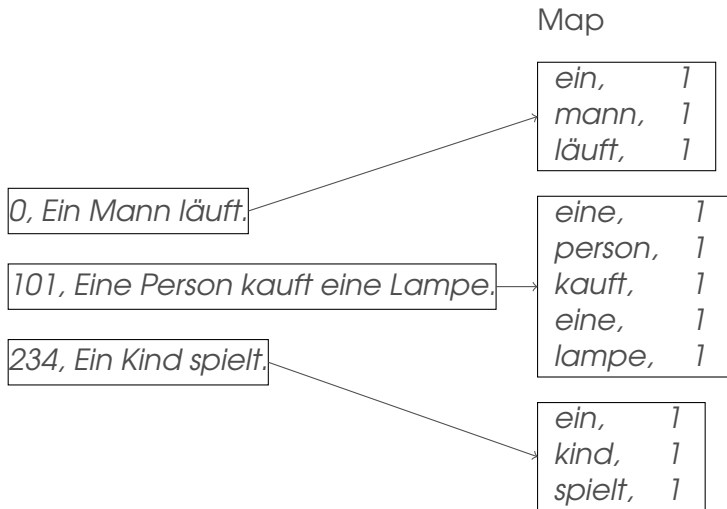
Beispiel: Zählen von Wörtern

0, Ein Mann läuft.

101, Eine Person kauft eine Lampe.

234, Ein Kind spielt.

Beispiel: Zählen von Wörtern



Beispiel: Zählen von Wörtern

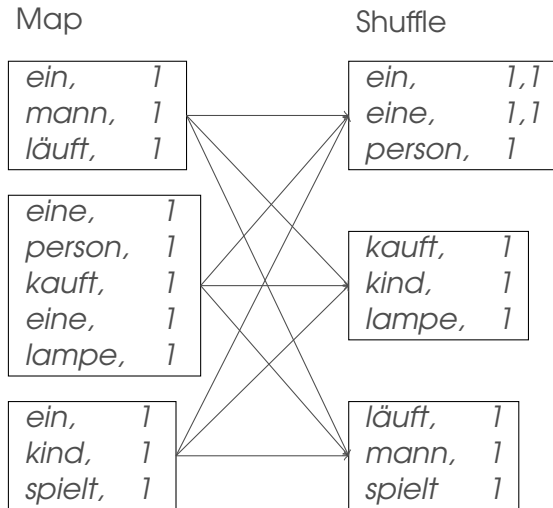
Map

<i>ein,</i>	<i>1</i>
<i>mann,</i>	<i>1</i>
<i>läuft,</i>	<i>1</i>

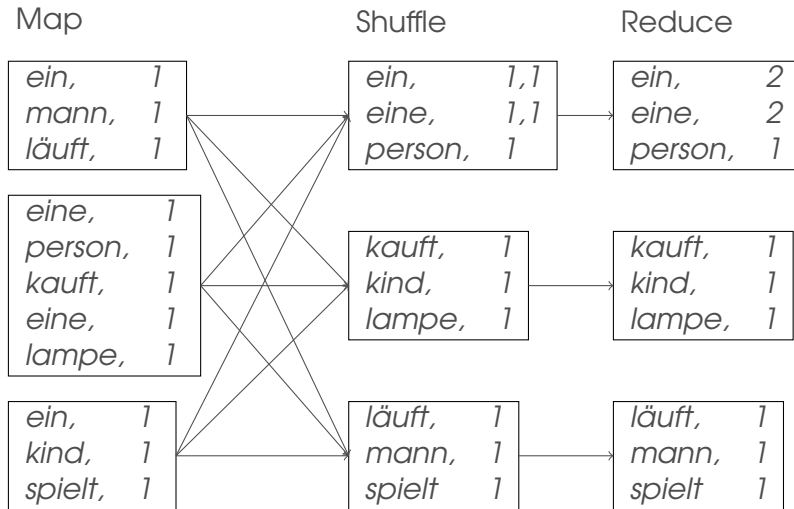
<i>eine,</i>	<i>1</i>
<i>person,</i>	<i>1</i>
<i>kauft,</i>	<i>1</i>
<i>eine,</i>	<i>1</i>
<i>lampe,</i>	<i>1</i>

<i>ein,</i>	<i>1</i>
<i>kind,</i>	<i>1</i>
<i>spielt,</i>	<i>1</i>

Beispiel: Zählen von Wörtern



Beispiel: Zählen von Wörtern



Beispiel: Zählen von Wörtern - Quellcode

Map Phase

```
map(String key, String value):  
    for each word in value.split(" "):  
        emit(word, "1");
```

Beispiel: Zählen von Wörtern - Quellcode

Map Phase

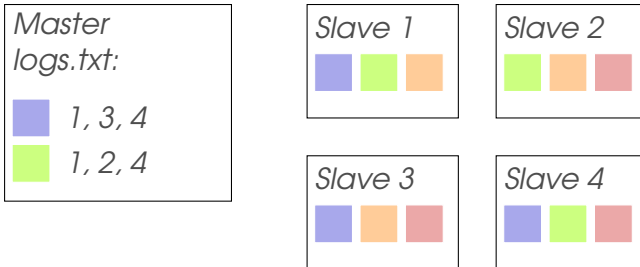
```
map(String key, String value):  
    for each word in value.split(" "):  
        emit(word, "1");
```

Reduce Phase

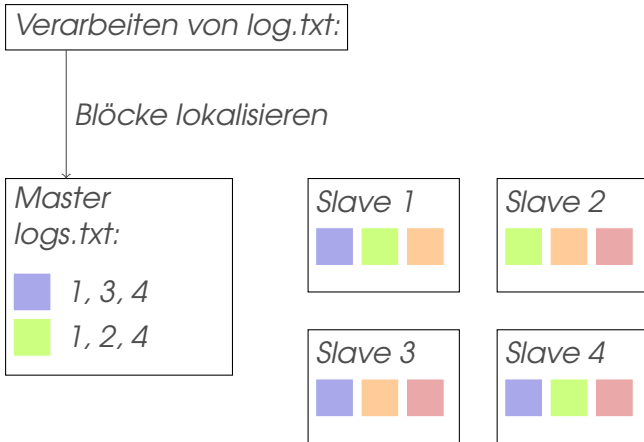
```
reduce(String key, Iterator values):  
    int result = 0;  
    for each value in values:  
        result += value;  
  
    emit(key, result);
```

Ablauf eines MapReduce Jobs

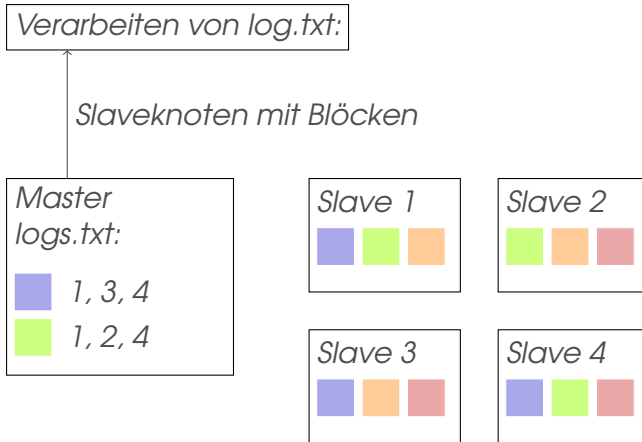
Verarbeiten von *log.txt*:



Ablauf eines MapReduce Jobs



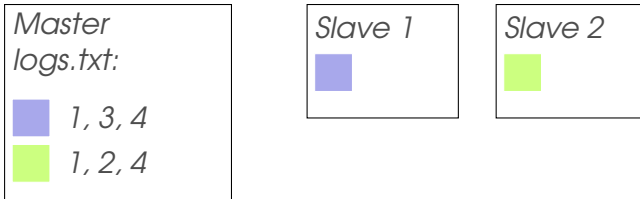
Ablauf eines MapReduce Jobs



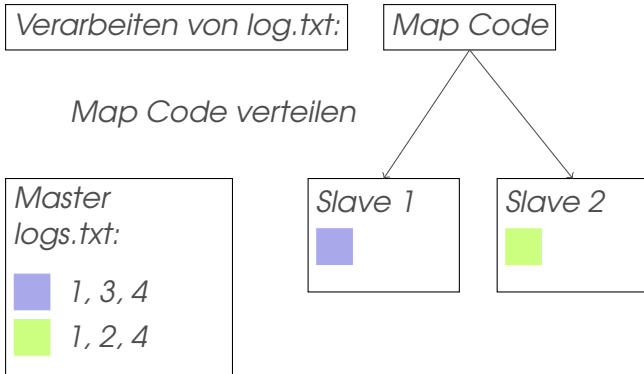
Ablauf eines MapReduce Jobs

Verarbeiten von *log.txt*:

Auswahl der Slaveknoten



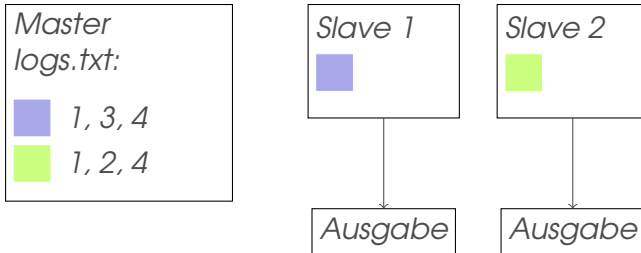
Ablauf eines MapReduce Jobs



Ablauf eines MapReduce Jobs

Verarbeiten von *log.txt*:

Map Code berechnen

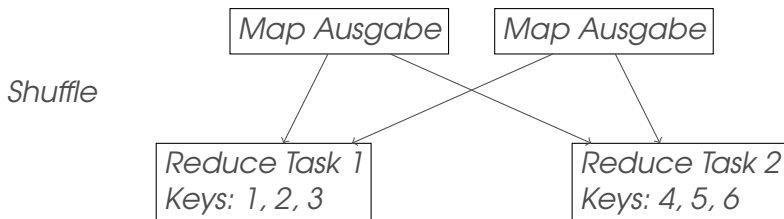


Ablauf eines MapReduce Jobs

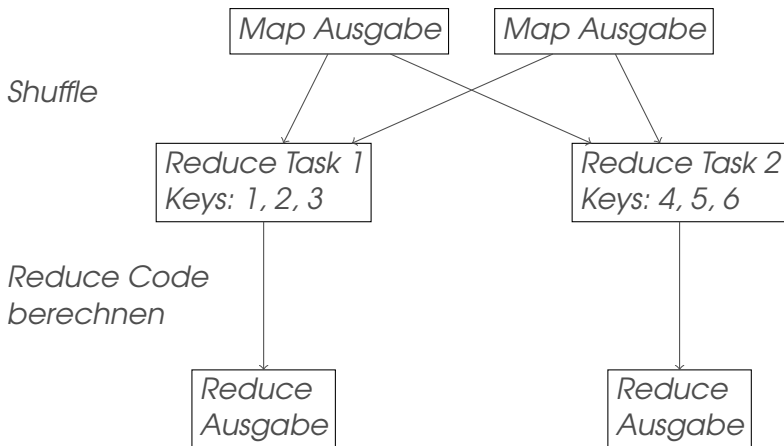
Map Ausgabe

Map Ausgabe

Ablauf eines MapReduce Jobs



Ablauf eines MapReduce Jobs



Ablauf eines MapReduce Jobs

1. Lokalisierung der Dateiblöcke
2. Verteilen des Map Codes auf die Knoten
3. Ausführen und Zwischenspeichern der Map Ausgabe
4. Weitergeben von Referenzen auf die Daten an die jeweiligen Reduce Tasks (Shuffle)
5. Sortieren / Gruppieren der Daten (Shuffle)
6. Ausführen des Reduce Codes und Speichern der Ausgabe

Aufgabe: Matrix Multiplikation mit MapReduce

Gegeben:

Matrix M mit I Spalten

Matrix N mit J Zeilen

- ▶ In wie viele Map und Reduce Tasks kann das Problem zerlegt werden?
- ▶ Was wird in den einzelnen Map und Reduce Tasks gemacht?

Job Größe und Skalierbarkeit

- ▶ Skalierbarkeit direkt eingebaut
- ▶ Task Größe abhängig von Teilung der Daten
- ▶ Kleinere Tasks besser als größere Tasks
- ▶ Typischerweise deutlich mehr Map/Reduce Tasks als Worker (Server)
- ▶ Beispiel von Google (2004)

Map Tasks	200.000
Reduce Tasks	5.000
Worker	2.000

Probleme

- ▶ Reduce Job muss auf langsamen Worker warten
- ▶ Map/Reduce Task stürzt ab
- ▶ Ausfall eines Rechners
 - ▶ Master
 - ▶ Slave

Probleme

- ▶ Reduce Job muss auf langsamen Worker warten
- ▶ Map/Reduce Task stürzt ab
- ▶ Ausfall eines Rechners
 - ▶ Master
 - ▶ Slave

Lösung

- ▶ Backup Tasks / Spekulative Ausführung
- ▶ Neu berechnen auf anderem Knoten

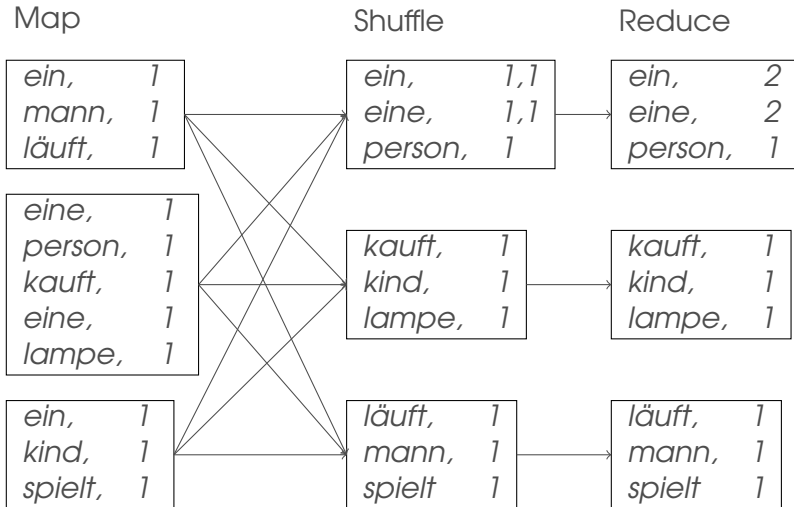
Combiner

- ▶ Map Task erzeugt evtl. viele Daten
 - ▶ Zählen von Wörtern: Mehrfach der gleiche Eintrag
- ▶ Daten werden für Reduce Task über das Netzwerk transportiert
- ▶ Erzeugt viel Last im Netzwerk und verlangsamt die Berechnung

⇒ Combiner reduziert Datenmenge auf der Map Seite

⇒ Combiner kann nicht immer verwendet werden

Combiner



Map

<i>ein,</i>	<i>1</i>
<i>mann,</i>	<i>1</i>
<i>läuft,</i>	<i>1</i>

<i>eine,</i>	<i>1</i>
<i>person,</i>	<i>1</i>
<i>kauft,</i>	<i>1</i>
<i>eine,</i>	<i>1</i>
<i>lampe,</i>	<i>1</i>

<i>ein,</i>	<i>1</i>
<i>kind,</i>	<i>1</i>
<i>spielt,</i>	<i>1</i>

Shuffle

Combiner

Map

<i>ein,</i>	<i>1</i>
<i>mann,</i>	<i>1</i>
<i>läuft,</i>	<i>1</i>

<i>eine,</i>	<i>1</i>
<i>person,</i>	<i>1</i>
<i>kauft,</i>	<i>1</i>
<i>eine,</i>	<i>1</i>
<i>lampe,</i>	<i>1</i>

<i>ein,</i>	<i>1</i>
<i>kind,</i>	<i>1</i>
<i>spielt,</i>	<i>1</i>

Combine

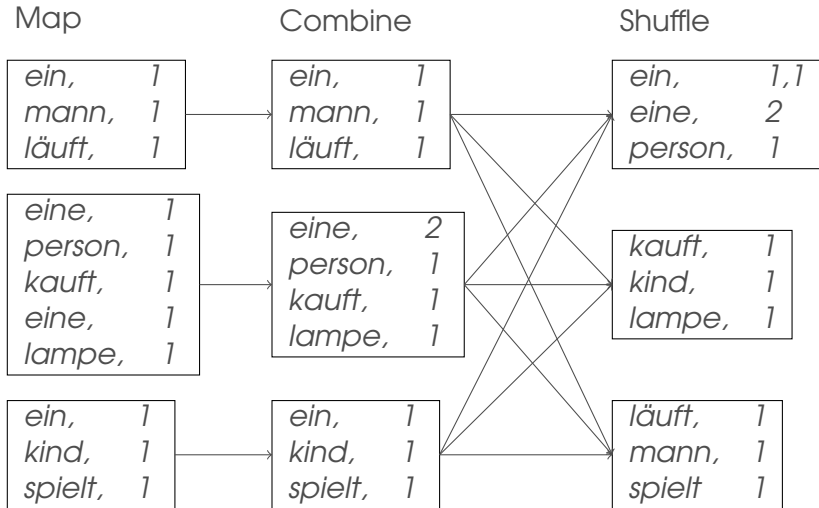
<i>ein,</i>	<i>1</i>
<i>mann,</i>	<i>1</i>
<i>läuft,</i>	<i>1</i>

<i>eine,</i>	<i>2</i>
<i>person,</i>	<i>1</i>
<i>kauft,</i>	<i>1</i>
<i>lampe,</i>	<i>1</i>

<i>ein,</i>	<i>1</i>
<i>kind,</i>	<i>1</i>
<i>spielt,</i>	<i>1</i>

Shuffle

Combiner



Probleme mit MapReduce

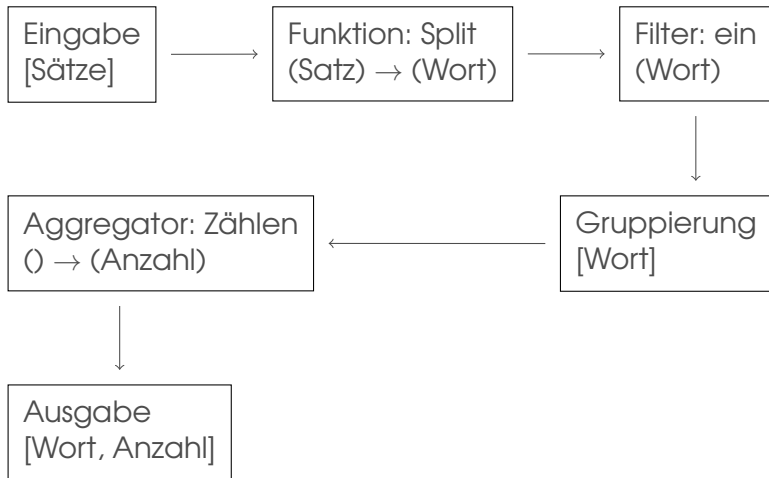
- ▶ Komplexität von MapReduce Jobs
- ▶ Zusammensetzen verschiedener MapReduce Jobs komplex

⇒ Pipe Diagramme vereinfachen die Darstellung

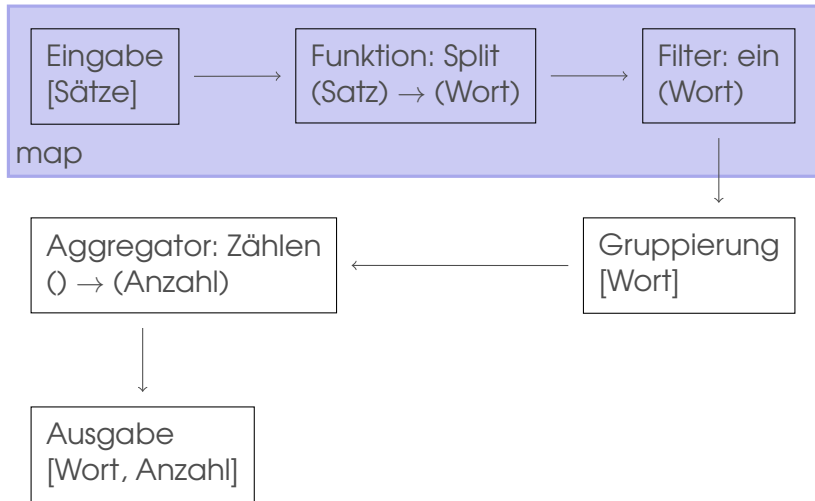
Pipe Diagramme - Bestandteile

Tupel	Key-Value Paar
Funktionen/Filter	Betrachten einzelne Tupel und wenden beliebige Transformation auf Tupel an
Aggregatoren	Betrachten eine Menge an Tupeln und wenden eine beliebige Funktion darauf an
Gruppierung	Values mit dem gleichen Key in einer Menge zusammenfassen
Joins	Kombination von Daten vgl. SQL
Merges	Code wird auf verschiedenen Datensätzen ausgeführt

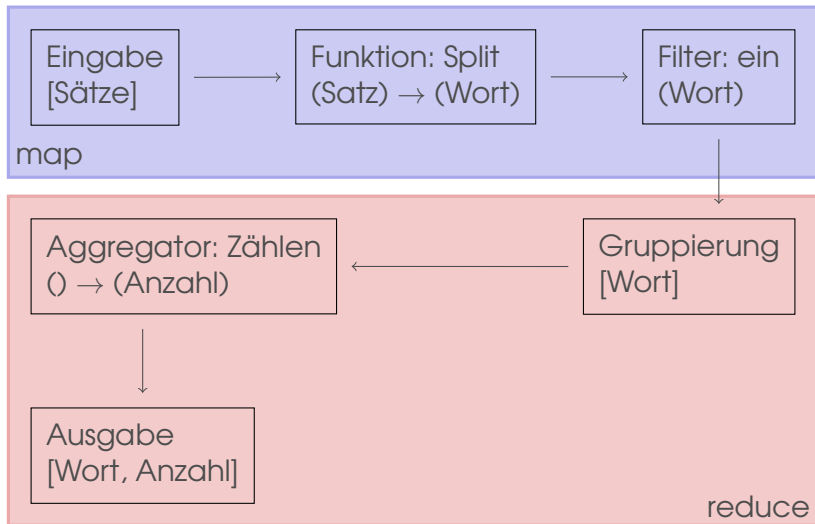
Pipe Diagramme - Beispiel Zählen von Wörtern



Pipe Diagramme - Beispiel Zählen von Wörtern



Pipe Diagramme - Beispiel Zählen von Wörtern



Live Demo

Lambda Architektur

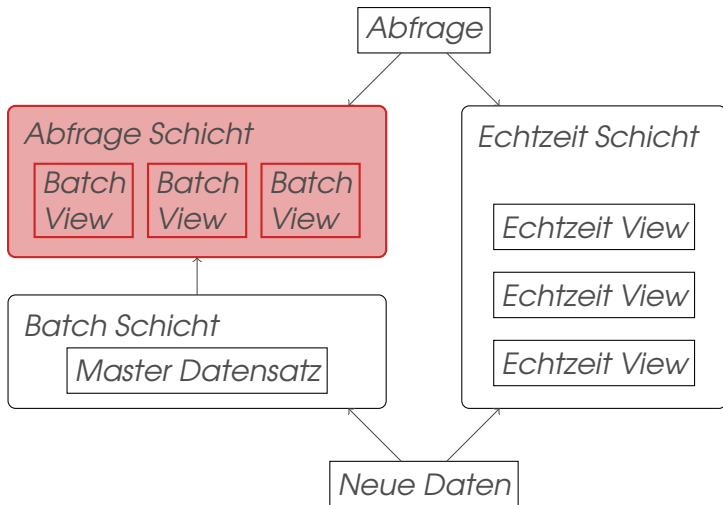
Datenmodell

Batch Schicht

Abfrage Schicht

Echtzeit Schicht

Lambda Architektur - Abfrage Schicht



Abfrage Schicht - Aufgabe

- ▶ Index Erstellung
- ▶ Schneller Zugriff auf Batch Views
- ▶ Beantwortung in Echtzeit

Abfrage Schicht - Performanz

- ▶ Verteilte Schicht
- ▶ Wartezeit
 - ▶ Server haben unterschiedliche Antwortzeiten
 - ▶ Wartezeit entspricht langsamstem Server
 - ▶ Viele Server = Hohe Wahrscheinlichkeit für langsamen Server
- ▶ Durchsatz
 - ▶ Abhängig von Zugriffszeit auf Festplatte
 - ▶ Suchen langsamer als Lesen

⇒ Daten nicht zufällig verteilen

⇒ Daten für gleiche Abfrage nebeneinander speichern

Normalisierung / Denormalisierung

- ▶ Normalisierung
 - ▶ Daten „sauber“ getrennt
 - ▶ Änderung einfach
 - ▶ Join in Abfragen zeitaufwendig
- ▶ Denormalisierung
 - ▶ Redundante Datenhaltung
 - ▶ Änderung aufwendig
 - ▶ Kein Join → Schnellere Abfragen

Problem

⇒ Bei relationalem System muss Entwickler vorher Verhältnis abwägen

Lösung durch Batch / Abfrage Schicht

- ▶ Master Datensatz normalisiert
- ▶ Batch View/Abfrage Schicht denormalisiert
- ▶ Redundante Daten in Abfrage Schicht ungefährlich
- ▶ Bei Fehler wird Abfrage Schicht neu berechnet

Abfrage Schicht - Anforderungen

- ▶ Batch beschreibbar
 - ▶ Auswechseln wenn neue Batch Views verfügbar sind
- ▶ Skalierbar
 - ▶ Batch Views können beliebig groß sein
- ▶ Zufälliges Lesen
 - ▶ Kurze Wartezeit bei Abfragen
- ▶ Fehlertolerant
- ▶ Kein Zufälliges Schreiben
 - ▶ Extreme Reduzierung der Komplexität

⇒ Beispiel Datenbanken: Voldemort oder ElephantDB

Lambda Architektur

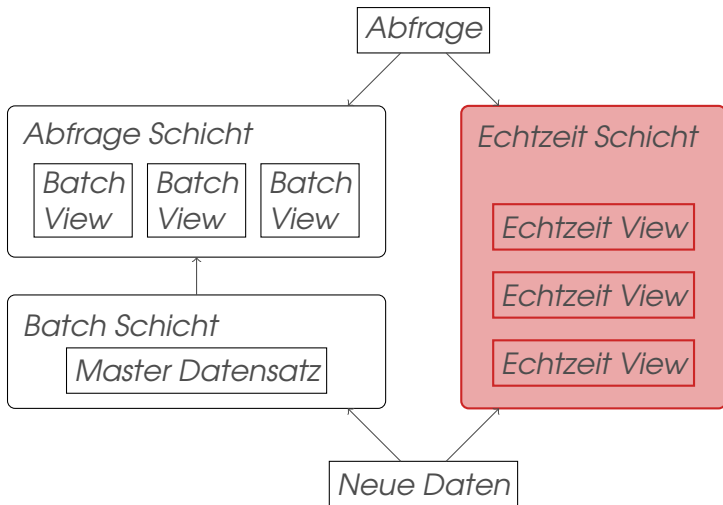
Datenmodell

Batch Schicht

Abfrage Schicht

Echtzeit Schicht

Lambda Architektur - Echtzeit Schicht



Fehlende Anforderung

- ▶ Batch Schicht langsame aber einfache Berechnung
- ▶ Abfrage Schicht schnelle Abfragen auf berechnete Batch Views
- ▶ Neue Daten seit letzter Batch Berechnung fehlend

Aufgaben

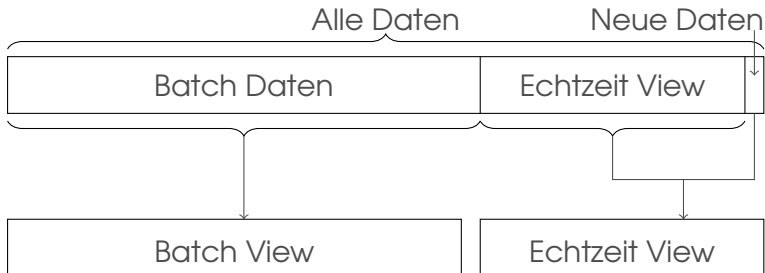
- ▶ Aktualisierung der Echtzeit Views mit neusten Daten
- ▶ Effiziente Speicherung der Echtzeit Views

1. Versuch

- ▶ Wiederverwenden der Batch Schicht für neuste Daten
- ▶ Neuberechnung bei jedem neuen Datum
- + Einfacher Ansatz ohne Redundanz im Quellcode
- + Ausreichend bei wenigen Minuten Reaktionszeit
- Reaktionszeit größer als 1 Sekunde
- Unnötige Berechnungen

2. Versuch

- ▶ Wiederverwendung bereits berechneter Echtzeit Views
- ▶ Inkrementelle Berechnung (vgl. RDBMS)



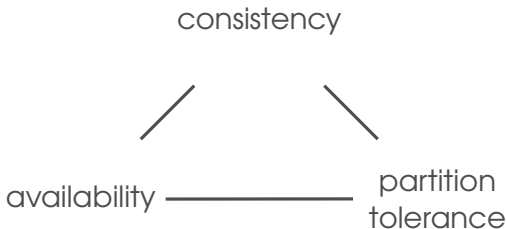
CAP Theorem

Consistency Konsistenz

Availability Verfügbarkeit

Partition-tolerance Toleranz gegenüber Verteilung

⇒ Nur 2 von 3 möglich



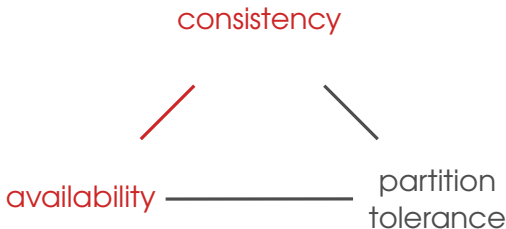
CAP Theorem

Consistency Konsistenz

Availability Verfügbarkeit

Partition-tolerance Toleranz gegenüber Verteilung

⇒ Nur 2 von 3 möglich



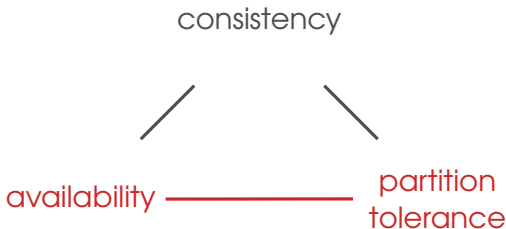
CAP Theorem

Consistency Konsistenz

Availability Verfügbarkeit

Partition-tolerance Toleranz gegenüber Verteilung

⇒ Nur 2 von 3 möglich



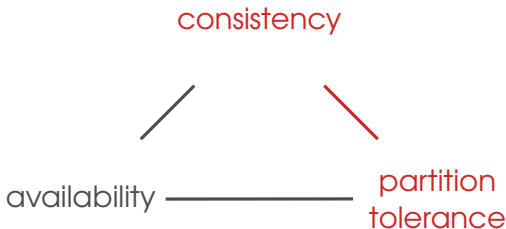
CAP Theorem

Consistency Konsistenz

Availability Verfügbarkeit

Partition-tolerance Toleranz gegenüber Verteilung

⇒ Nur 2 von 3 möglich



Anforderungen

- ▶ Zufälliges Lesen
- ▶ Skalierbarkeit
- ▶ Fehlertoleranz

Anforderungen

- ▶ Zufälliges Lesen
- ▶ Skalierbarkeit
- ▶ Fehlertoleranz
- ▶ Zufälliges Schreiben

Genauigkeit

- ▶ Gleiche Repräsentation der Daten wie in Batch Schicht
- ▶ Schätzungen der Werte
 - ▶ Reduziert Ressourcenverbrauch

⇒ Batch Schicht korrigiert Schätzfehler mit der Zeit

Speichermenge

- ▶ Kontinuierliches Speicheraufräumen (online compaction)
- ▶ Nebenläufigkeit

⇒ Gleiche Komplexität wie voll inkrementelles System

⇒ Komplexität nur in kleinem Teil der Daten

Welche Daten sind in einem Echtzeit View

- ▶ Noch nicht in Batch View integrierte Daten

Welche Daten sind in einem Echtzeit View

- ▶ Noch nicht in Batch View integrierte Daten

Abfrage
Echtzeit

Start der Anwendung, keine Daten vorhanden

Welche Daten sind in einem Echtzeit View

- ▶ Noch nicht in Batch View integrierte Daten

Abfrage

Echtzeit 

1. Batch Berechnung ohne Daten fertig.
Echtzeit Schicht enthält alle Daten.
Batch Berechnung mit ersten Daten starten.

Gültigkeit der Daten

Welche Daten sind in einem Echtzeit View

- ▶ Noch nicht in Batch View integrierte Daten

Abfrage

Echtzeit 

2. Batch Berechnung fast fertig.
Echtzeit Schicht enthält Daten von
1. und 2. Batch Berechnung.

Gültigkeit der Daten

Welche Daten sind in einem Echtzeit View

- ▶ Noch nicht in Batch View integrierte Daten



2. Batch Berechnung fertig.

Gültigkeit der Daten

Welche Daten sind in einem Echtzeit View

- ▶ Noch nicht in Batch View integrierte Daten

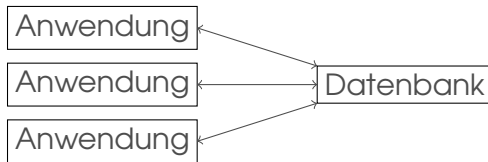


Daten aus Echtzeit Schicht gelöscht.

Verarbeitung der Aktualisierungen

Synchron

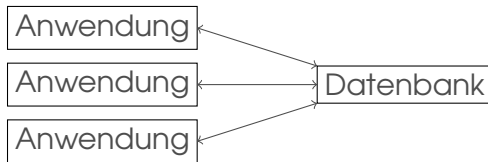
- ▶ Direkte Kommunikation zur Datenbank
- ▶ Blockierung bis Aktualisierung fertig



Verarbeitung der Aktualisierungen

Synchron

- ▶ Direkte Kommunikation zur Datenbank
- ▶ Blockierung bis Aktualisierung fertig



+ Schnell

- Kein Ausgleich von Lastspitzen

Verarbeitung der Aktualisierungen

Asynchron

- ▶ Aktualisierung wird an Warteschlange übergeben
- ▶ Tatsächliche Aktualisierung passiert später

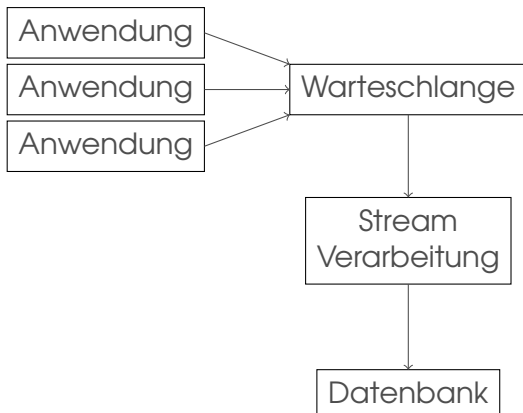
+ Höherer Durchsatz

+ Besseres Handling von Lastspitzen

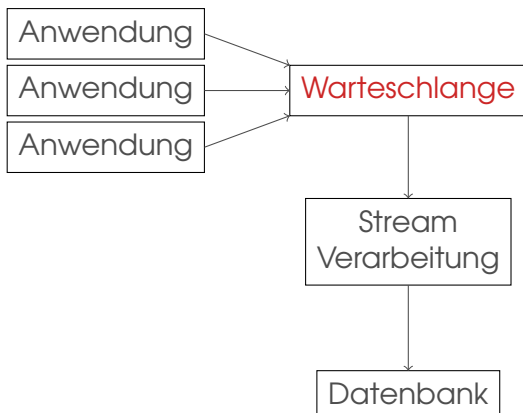
— Keine Kontrolle über Ausführungszeitpunkt

⇒ Vorteilhafter als synchrone Verarbeitung

Verarbeitung der Aktualisierungen



Verarbeitung der Aktualisierungen



Warteschlange

- ▶ Puffer zwischen Eingabe und Verarbeitung
- ▶ Verarbeitetes Element wird „konsumiert“

Warteschlange

- ▶ Puffer zwischen Eingabe und Verarbeitung
- ▶ Verarbeitetes Element wird „konsumiert“

```
interface Queue {  
    void add(Object item);  
    Object poll();  
    void acknowledge(Object item);  
}
```

- ▶ Puffer zwischen Eingabe und Verarbeitung
- ▶ Verarbeitetes Element wird „konsumiert“

```
interface Queue {  
    void add(Object item);  
    Object poll();  
    void acknowledge(Object item);  
}
```

add Hinzufügen neuer Elemente

poll Aktuelles Element abfragen und auf
 in Bearbeitung setzen

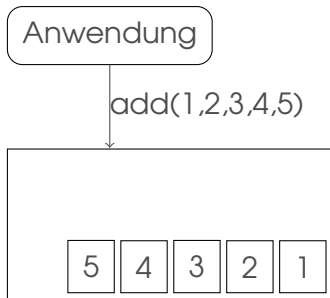
acknowledge Erfolgreiche Bearbeitung bestätigen

Anwendung



Verarbeitung

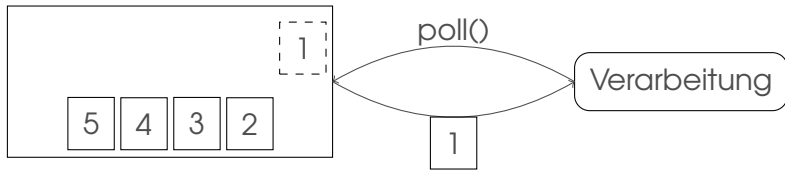
Warteschlange



Verarbeitung

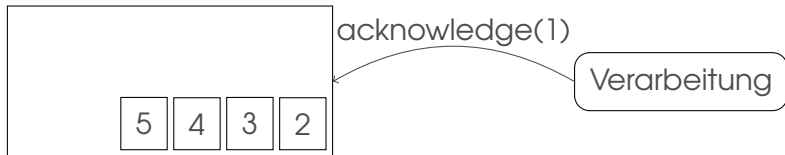
Warteschlange

Anwendung



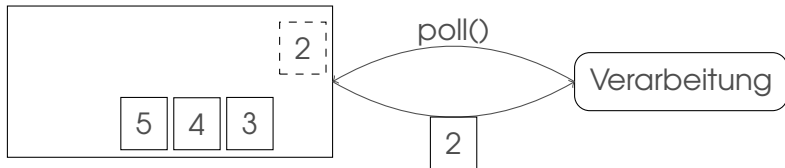
Warteschlange

Anwendung



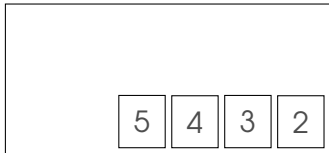
Warteschlange

Anwendung



Warteschlange

Anwendung

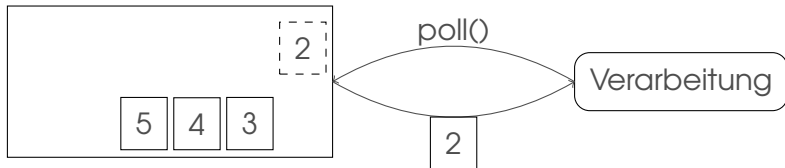


Timeout

~~Verarbeitung~~

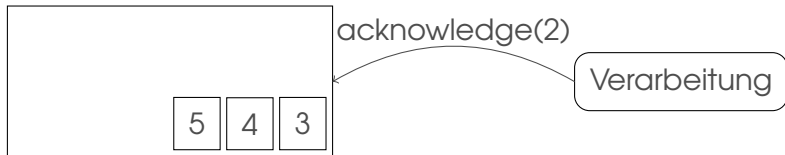
Warteschlange

Anwendung



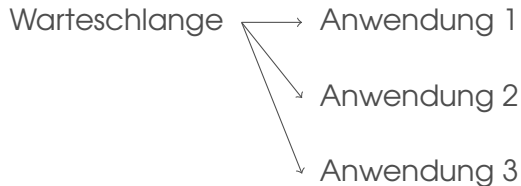
Warteschlange

Anwendung



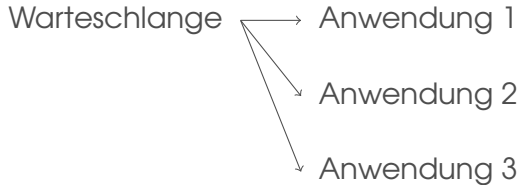
Warteschlange

Warteschlange mit mehreren Anwendungen?



Warteschlange

Warteschlange mit mehreren Anwendungen?



⇒ Sicherstellen, dass jedes Element von jeder Anwendung verarbeitet wird

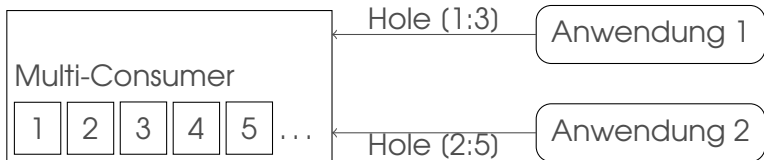
⇒ Zu viele Aufgaben bei Warteschlange

Besser

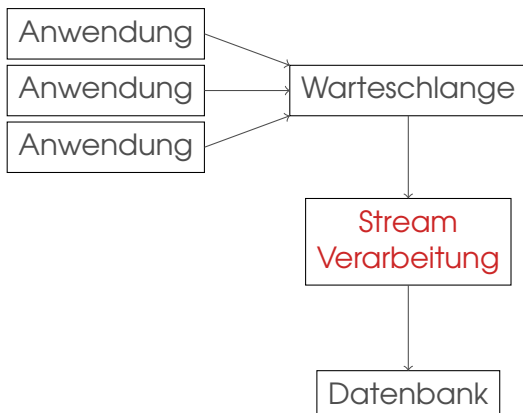
- ▶ Warteschlange enthält Elemente der letzten X Stunden
- ▶ Anwendung kann beliebige Elemente abfragen
- ▶ Anwendung verwaltet, welche Elemente erfolgreich abgearbeitet wurden

Besser

- ▶ Warteschlange enthält Elemente der letzten X Stunden
- ▶ Anwendung kann beliebige Elemente abfragen
- ▶ Anwendung verwaltet, welche Elemente erfolgreich abgearbeitet wurden



Stream basierte Verarbeitung



Stream basierte Verarbeitung

- ▶ Strom von Daten (Stream)
- ▶ Umwandlung und Ausgabe der Daten als Strom
- ▶ Beispiel
 - ▶ Java InputStream

Arten stream basierter Verarbeitung

Eins nach dem Anderen	Jedes Tupel getrennt verarbeiten
Micro Batch	Mehrere Tupel zusammen verarbeiten

Arten stream basierter Verarbeitung

Eins nach dem Anderen Jedes Tupel getrennt verarbeiten
Micro Batch Mehrere Tupel zusammen verarbeiten

	„Eins nach dem Anderen“	Micro Batch
Kurze Wartezeit	x	
Hoher Durchsatz		x
Mindestens einmal	x	x
Genau einmal	evtl.	x

Eins nach dem Anderen oder Storm Modell

- ▶ Benannt nach Apache Storm Projekt
- ▶ Verarbeitet unbegrenzten Strom von Tupeln

Bestandteile

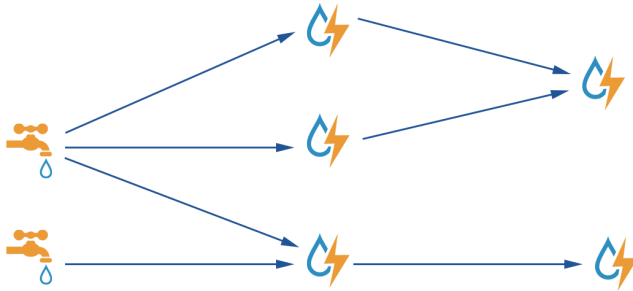
Spout Quelle von Tupeln

Bolt Verarbeitung von Streams und Ausgabe neuer Streams

Topology Netzwerk aus Spouts und Bolts

Task Instanz von Spout oder Bolt

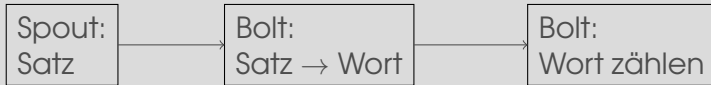
Eins nach dem Anderen oder Storm Modell



Verteilung der Tupel auf nachfolgende Bolts

- ▶ Analog wie bei MapReduce
 - ▶ „fields grouping“
- ▶ Zufällig
 - ▶ „shuffle grouping“

Zählen von Wörtern



Garantierte Berechnung jedes Tupels

Problem

- ▶ Ausfall von Worker
- ▶ Tupel wird nicht fertig berechnet
- ▶ Teil oder alles muss wiederholt werden
- ▶ Eventuell mehrfach in Echtzeit View

Garantierte Berechnung jedes Tupels

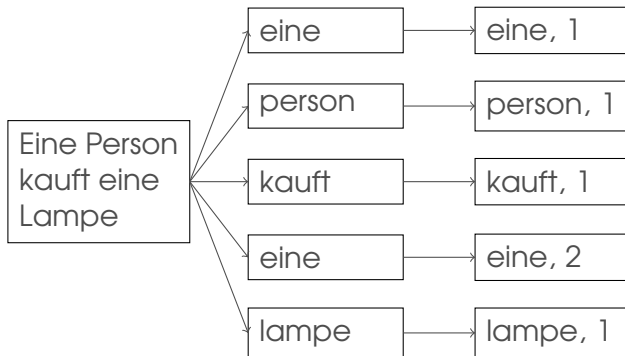
Lösung

- ▶ Tupel Verarbeitung als gerichteter azyklischer Graph (DAG)
- ▶ Überwachen der Tupel im DAG
- ▶ Bei Fehler komplettes Tupel neu bearbeiten

⇒ Garantiert Berechnung mindestens 1 Mal

Garantierte Berechnung jedes Tupels - Beispiel

Tupel DAG für „Zählen von Wörtern“



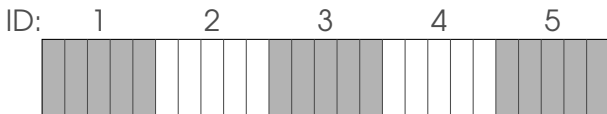
Live Demo

Micro Batch Stream Verarbeitung

- ▶ Menge von Tupeln zusammen verarbeiten
- + Höhere Genauigkeit
- + Garantiert Berechnung genau 1 Mal
- Größere Wartezeit
- Festgelegte Reihenfolge der Tupel

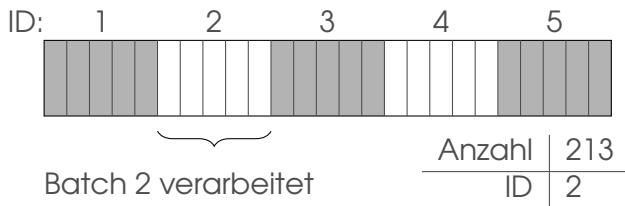
Micro Batch Stream Verarbeitung

- ▶ Tupel zu Menge zusammenfassen (Batch)
- ▶ Jeder Batch erhält eine ID
- ▶ Ergebnis der Berechnung mit Batch ID in Datenbank ablegen
- ▶ Atomare Transaktion



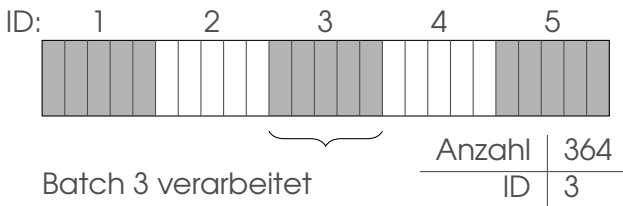
Micro Batch Stream Verarbeitung

- ▶ Tupel zu Menge zusammenfassen (Batch)
- ▶ Jeder Batch erhält eine ID
- ▶ Ergebnis der Berechnung mit Batch ID in Datenbank ablegen
- ▶ Atomare Transaktion

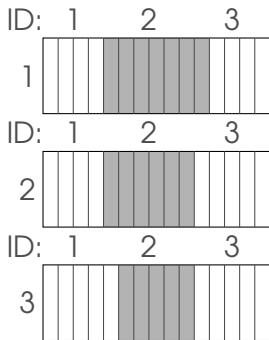


Micro Batch Stream Verarbeitung

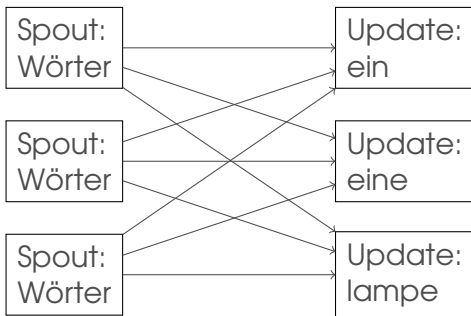
- ▶ Tupel zu Menge zusammenfassen (Batch)
- ▶ Jeder Batch erhält eine ID
- ▶ Ergebnis der Berechnung mit Batch ID in Datenbank ablegen
- ▶ Atomare Transaktion



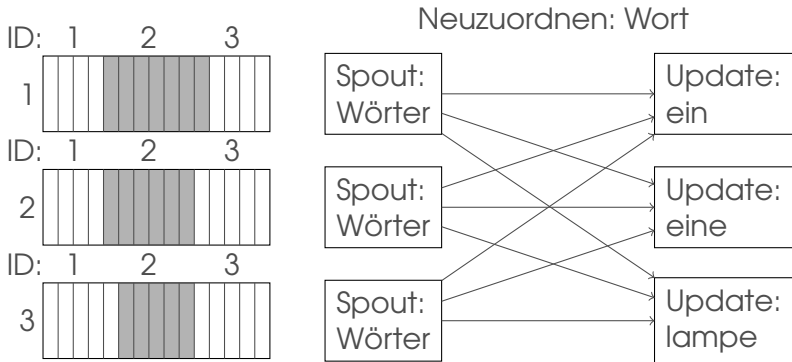
Micro Batch Stream Verarbeitung - Skalierbarkeit



Neuzuordnen: Wort

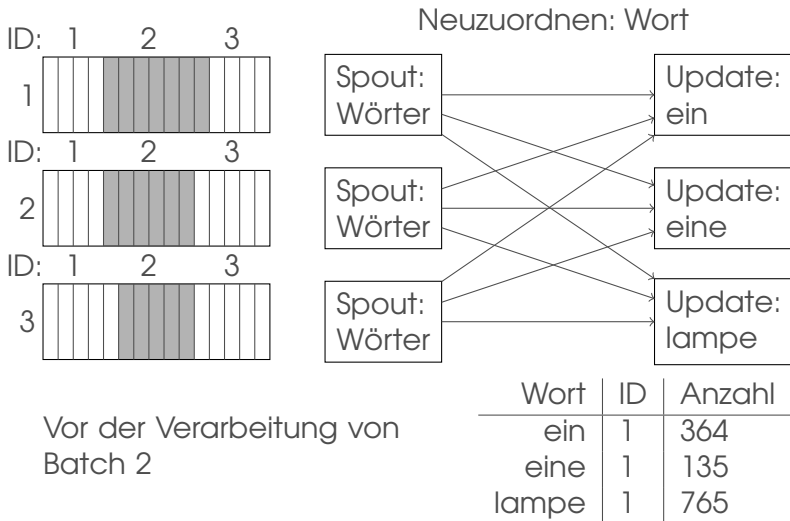


Micro Batch Stream Verarbeitung - Skalierbarkeit

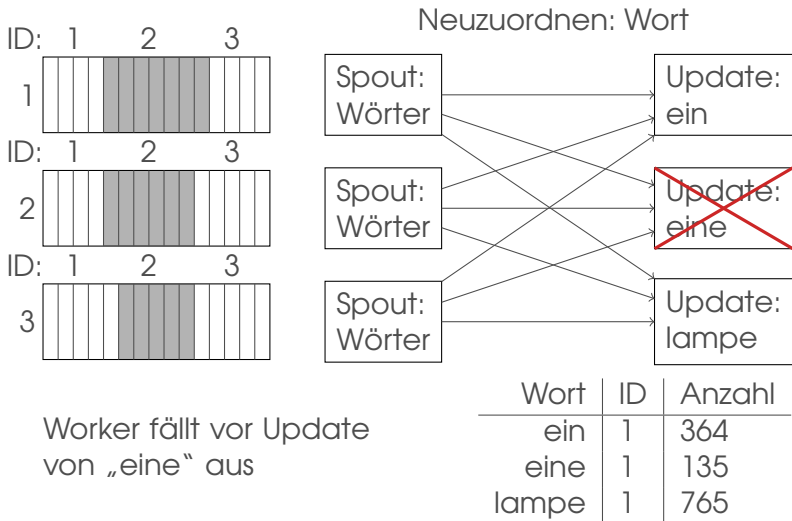


Beispiel: Ausfall von Update Worker

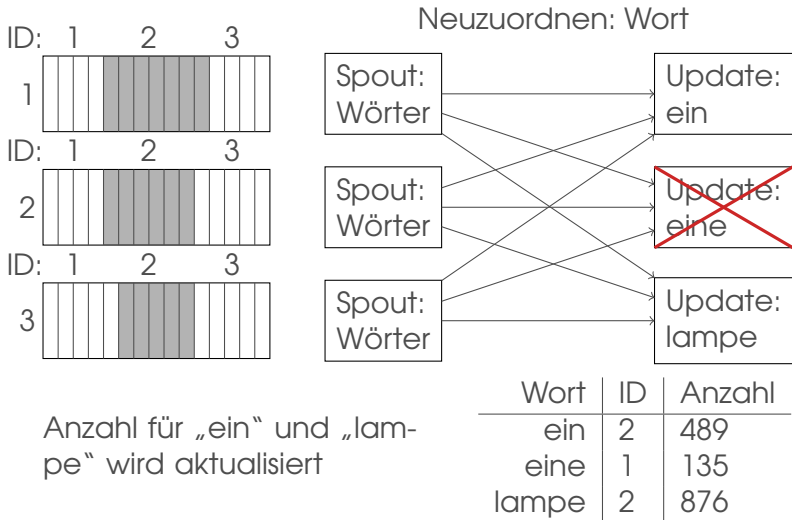
Micro Batch Stream Verarbeitung - Skalierbarkeit



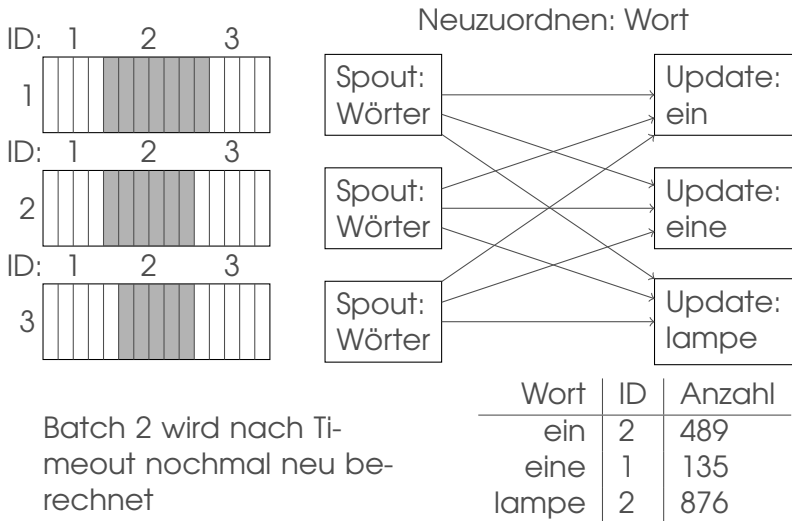
Micro Batch Stream Verarbeitung - Skalierbarkeit



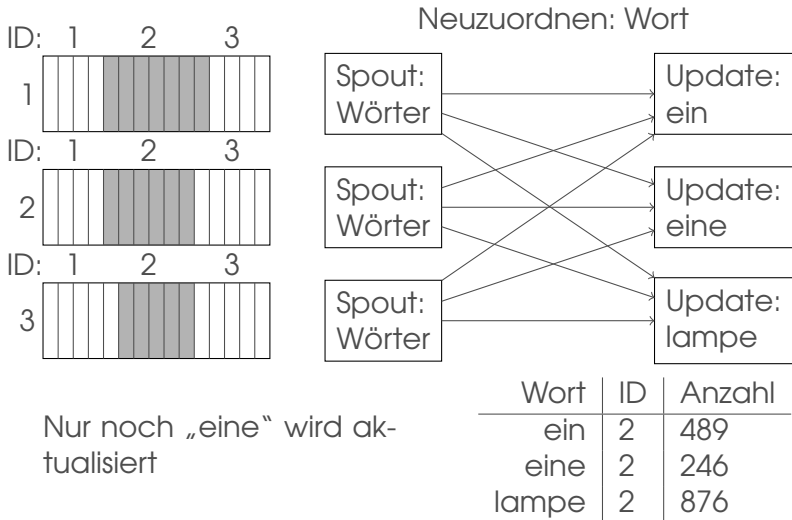
Micro Batch Stream Verarbeitung - Skalierbarkeit



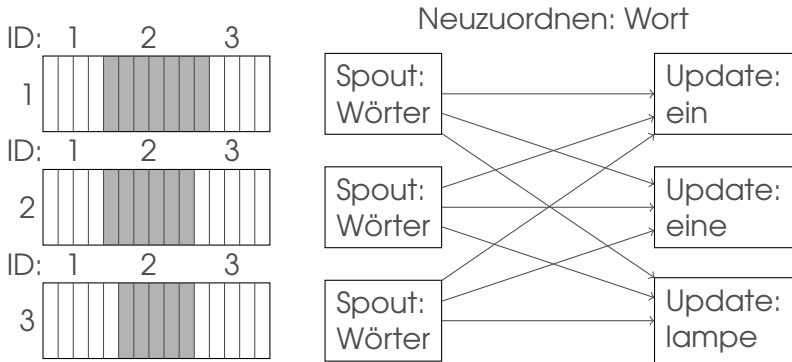
Micro Batch Stream Verarbeitung - Skalierbarkeit



Micro Batch Stream Verarbeitung - Skalierbarkeit

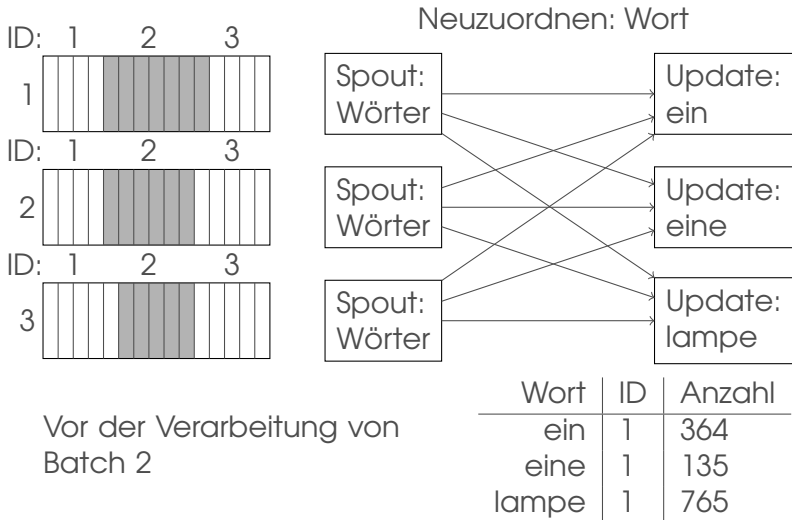


Micro Batch Stream Verarbeitung - Skalierbarkeit

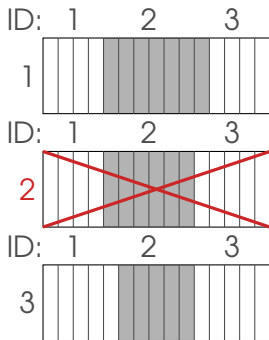


Beispiel: Ausfall von Partition / Spout

Micro Batch Stream Verarbeitung - Skalierbarkeit

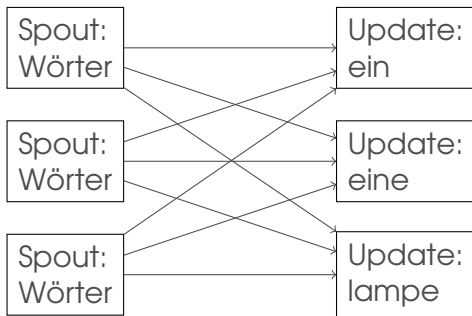


Micro Batch Stream Verarbeitung - Skalierbarkeit



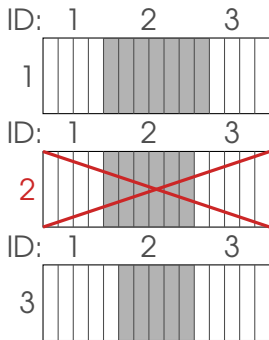
Partition fällt während Verarbeitung von Batch 2 aus

Neuzuordnen: Wort



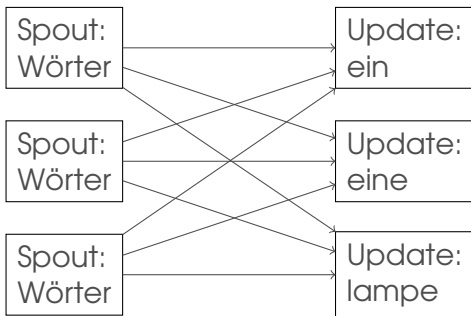
Wort	ID	Anzahl
ein	1	364
eine	1	135
lampe	1	765

Micro Batch Stream Verarbeitung - Skalierbarkeit



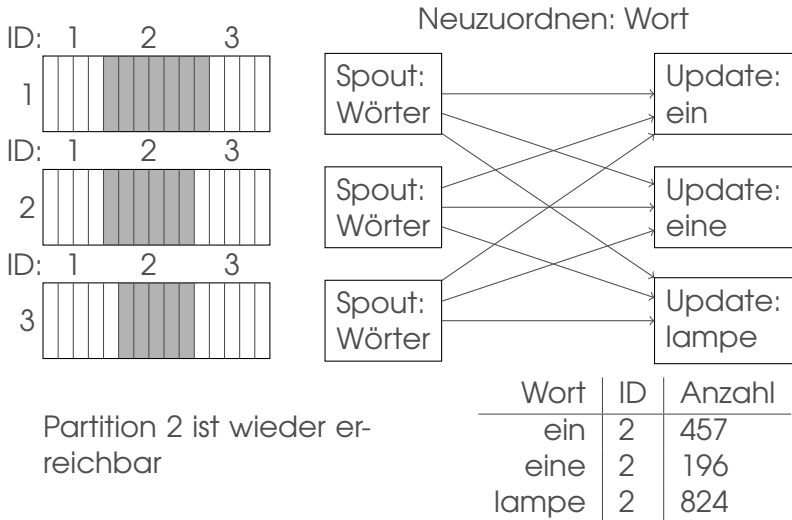
Batch 2 wird mit Partition
1 und 3 neu berechnet

Neuzuordnen: Wort

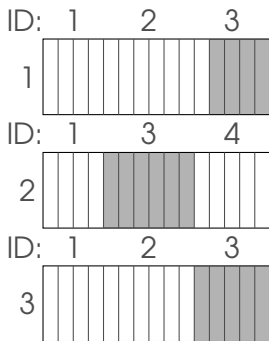


Wort	ID	Anzahl
ein	2	457
eine	2	196
lampe	2	824

Micro Batch Stream Verarbeitung - Skalierbarkeit

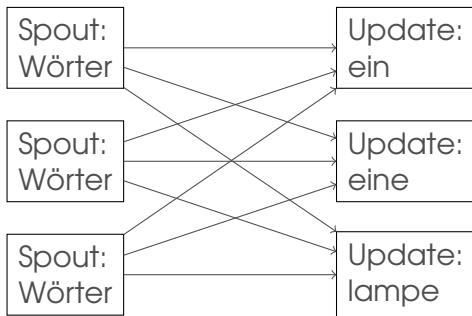


Micro Batch Stream Verarbeitung - Skalierbarkeit



Partition 2 wird in Batch 3
neu berechnet

Neuzuordnen: Wort



Wort	ID	Anzahl
ein	3	579
eine	3	348
lampe	3	943

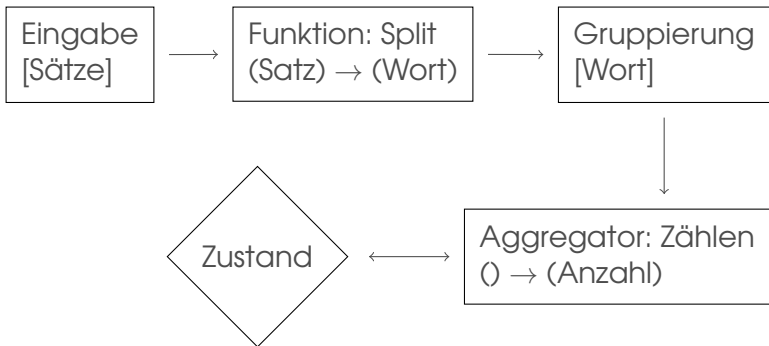
Verarbeitungsarten

- ▶ Innerhalb eines Batches (Batch-Local)
 - ▶ Benötigt nur Informationen des aktuellen Batches
 - ▶ Verarbeitung wie im Storm Modell
- ▶ Über mehrere Batches (Stateful)
 - ▶ Globaler Zustand wichtig
 - ▶ Speicherung mit Batch ID

Micro Batch Stream Verarbeitung

Erweiterung von Pipe Diagrammen

- Zustandsspeicher hinzufügen



Live Demo

Gliederung

Motivation

Lambda Architektur

Datenmodell

Batch Schicht

Abfrage Schicht

Echtzeit Schicht

Fazit

Literatur / Quellen

Lambda Architektur - Fazit

- + Erfüllt die Anforderungen an ein Big Data System
 - + Fehlertoleranz
 - + Skalierbarkeit
 - + ...
- + Neuberechnung der Daten explizit integriert
- + Reduzierung der Komplexität auf wenige Daten
- Getrennte Frameworks für Batch und Echtzeit Schicht
 - Programmierung für beide Frameworks
 - Wartung mehrerer Systeme
- Abstraktion aktuell noch nicht ausreichend

- ▶ Inkrementelle Berechnung von Batch Views
- ▶ Partielle Neuberechnung
 - ▶ Teil von Batch View Neuberechnen, der sich ändert
 - ▶ Rest von altem Batch View übernehmen
- ▶ Probabilistische Verarbeitung
 - ▶ z.B. Bloom Filter
- ▶ Mehrere Batch Schichten kombiniert
 - ▶ Komplette Daten (1 Monat)
 - ▶ Partielle Aktualisierung (wenige Stunden)

Gliederung

Motivation

Lambda Architektur

Datenmodell

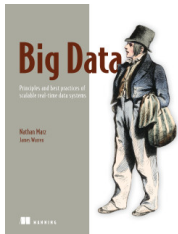
Batch Schicht

Abfrage Schicht

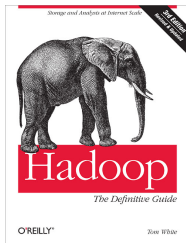
Echtzeit Schicht

Fazit

Literatur / Quellen



- ▶ Big Data
 - ▶ Nathan Marz, James Warren
 - ▶ Manning
 - ▶ ISBN: 978-1617290343



- ▶ Hadoop: The Definitive Guide
 - ▶ Tom White
 - ▶ O'Reilly
 - ▶ ISBN: 978-1449311520

► Internet

- MapReduce: simplified data processing on large clusters - Jeffrey Dean, Sanjay Ghemawat
- datasciencecentral.com
- <http://www.emc.com/leadership/digital-universe/2014iview/executive-summary.htm>
- manning.com
- oreilly.com
- <http://www.infoq.com/interviews/marz-lambda-architecture>
- storm.apache.org
- wikipedia.org